

FINAL REPORT

FAULT TOLERANT PROGRAMMABLE  
DIGITAL ATTITUDE CONTROL  
ELECTRONICS STUDY

(NASA-CR-142289) FAULT TOLERANT  
PROGRAMMABLE DIGITAL ATTITUDE CONTROL  
ELECTRONICS STUDY Final Report (TRW Systems  
Group) 479 p HC \$12.00

N75-17400

CSCL 22B

G3/15

Unclass  
17482

Prepared for  
CALIFORNIA INSTITUTE OF TECHNOLOGY  
JET PROPULSION LABORATORY

25 JULY 1974

Contract No. 953883

**TRW**  
SYSTEMS GROUP

ONE SPACE PARK • REDONDO BEACH, CALIFORNIA 90278

0011

# TABLE OF CONTENTS

		<u>Page</u>
1.0	<u>Introduction</u>	1
1.1	Programmable Digital Electronics	1
1.2	Fault Tolerance	4
1.3	Study Scope	5
1.4	Report Organization	7
1.5	Acknowledgements	9
2.0	<u>General Requirements and Constraints</u>	10
2.1	Functional and Operational Requirements	10
2.1.1	The Reference Mission & Spacecraft	11
2.1.2	Control System Requirements	15
2.2	Design Constraints	26
2.2.1	Life and Reliability	26
2.2.2	Availability & Recovery	28
2.2.3	System Safety	31
2.2.4	Growth Potential	33
2.2.5	System Autonomy	34
2.2.6	Physical	36
2.2.7	Technology	37
2.2.8	Other	38
3.0	<u>Design Considerations and Tradeoffs</u>	40
3.1	Redundancy Policy	41
3.1.1	Element Size Determination	46
3.1.2	Element Control Approach	69
3.1.3	Cross-Strapping Selection	79
3.1.4	Redundancy Management	95
3.2	Control System Design	102
3.2.1	Control System Implementation	102
3.2.2	Functional Redundancy	109
3.2.3	Interface Definition	112
3.2.4	Fail Safe Design	114
3.2.5	Power Interruption	125
3.3	Hardware/Software Requirements	129
3.3.1	Processor/Electronics Functional Requirements	129
3.3.2	Software Computational Requirements	131
3.3.3	Input/Output Requirements	140

## TABLE OF CONTENTS (CONT'D)

	<u>Page</u>
3.4 Processor Design	146
3.4.1 Arithmetic	146
3.4.2 Instructions	149
3.4.3 Interrupts	154
3.4.4 Timing	156
3.4.5 External Communication	158
3.4.6 Internal Communication	183
3.4.7 Memories	185
3.4.8 Technology	199
3.4.9 Architecture	204
3.4.10 Micro-Programming	208
3.5 Software Design	210
3.5.1 Software Requirements & Design Criteria	210
3.5.2 Program Organization & Modularity	211
3.5.3 Executive Tradeoffs	217
3.5.4 Reference Software Configuration	225
3.5.5 Diagnostics & Fault Detection	236
3.5.6 Software Sizing	240
3.6 Hardware Fault Detection	244
3.6.1 Error Detecting Coding	244
3.6.2 Self-test	254
3.6.3 Clock Faults	264
3.7 Fault Tolerance, Failure Detection & Recovery	269
3.7.1 Fault Tolerance Criteria	269
3.7.2 Fault Tolerance Requirements	273
3.7.3 Organizational Criteria for Fault Tolerance	277
3.7.4 Failure Detection Processes	281
3.7.5 Recovery Management Techniques	288
3.7.6 Recovery Implementation	296
3.7.7 Backup Mode Implementation Alternatives	301
3.7.8 Recommended Approach	305
3.8 LSI Applications	307
3.8.1 Commercial Parts	308
3.8.2 Custom Parts	309
3.8.3 Micro-processors	315
3.8.4 Micro-programming	384

## TABLE OF CONTENTS (CONT'D)

4.0	<u>System Description</u>	<u>Page</u> 320
4.1	Current Technology	320
	4.1.1 Performance Requirements	321
	4.1.2 Processor	325
	4.1.3 Peripherals	329
4.2	Advanced Technology	330
5.0	<u>Summary, Conclusions and Recommendations</u>	334
5.1	Summary and Conclusions	334
5.2	Recommendations for Further Studies	337
	Bibliography	340
	Appendix A - COPE (DPA) Specification	separately page numbered
	Appendix B - Glossary	starting after page - 340

## 1.0 INTRODUCTION

This report is presented to summarize work done by TRW Systems for Jet Propulsion Laboratory under contract 26084.00 for a "Fault Tolerant Programmable Digital Attitude Control Electronics".

To introduce the subject matter of this report, it is needed to consider the meaning of, need for, and implications of; the phrases "programmable digital electronics", and "fault tolerant".

### 1.1 Programmable Digital Electronics

Advanced capability spacecraft attitude control systems are now requiring

- Multiple operational modes with control laws having unique form and parameters for each mode.
- Greater accuracy requirements, resulting in increasing control law complexity.
- Seven to ten year (and longer) life spans with high reliability (0.9 to 0.95) requirements.
- Multimission capability with minimum redesign for the same hardware.
- Capability of contingency operation to achieve specified operation with shortened life or degraded operation without shortened life.
- Reduced system development span time.
- Reduced size, weight, power, and cost.

Traditionally, aircraft autopilots and spacecraft attitude control systems have utilized conventional analog circuitry. More recently, these special purpose control electronics have been designed using digital circuitry.

Advantages of digital circuitry are:

- Sophisticated system functions are often easier to implement.
- Precision can be achieved more readily (e.g., electronic drift and noise can be eliminated).
- System test and checkout can be accomplished more easily.
- The required redundancy is more readily implemented.
- The hardware analysis tasks are simplified.

Most existing recent spacecraft attitude control systems employ special purpose digital electronics. In these electronics the implementation is designed for the specific application, and each individual function is performed by unique circuitry, in parallel, by a distinct portion of the electronics. Such designs generally preclude the application of a developed design to more than one spacecraft. Circuits are usually extensively replicated (functionally), increasing power and weight.

The increasing demands and complexity have been approximately neutralized by the rapid advances in integrated circuit technology, and attendant increase in the equivalent functions obtainable in each part as the technology has progressed toward LSI (large-scale integration).

The extended lifetimes needed and increasing cost experienced as the level of integration increases, have shown that a new approach is needed.

Programmable digital electronics offers an attractive and potentially simpler approach. In such a design, the same circuits are used for all computations and less total electronics are required. The operation of these shared circuits is controlled by a stored program, which may be changed for each application or may be changed prior to or during a flight mission. Since the control laws are contained in the program (software), increased flexibility, reduced development span time, and reduced program development costs can be realized.

Such programmable digital electronics has many characteristics in common with a general-purpose digital computer; indeed, use of such a computer as part of a system is one approach. However, considering the spacecraft

requirements for attitude control, together with the reliability problem, the programmable digital electronics, designed specifically to perform attitude control functions, offers significant advantages.

At this point we shall define the subsystem as being composed of four types of equipment. These are:

- Sensors
- Actuators
- Peripheral Electronics
- Processor(s)

The sensors consist of all inertial and optical sensors that provide spacecraft rate, position, acceleration, etc. sensing for attitude control. Actuators include thrusters, reaction wheels, CMG's, vectorable engine controls, and experiment or payload pointing control means.

The processor is the "computer" portion of the control system electronics, containing the memories, the processing and some I/O (input/output) capability. The peripheral electronics is all other electronics; usually that electronics interfacing between the other three types of equipment.

The use of a processor allows maximum flexibility, in design, in design confirmation, in making post-design changes, and in modifying the control system data or program during flight. Several different sets of data and/or of programs can be stored and used as needed for different modes.

The same processor (with only software changes) can be used for many different program contracts. If the other equipment is modularly designed, then combinations of sensors, actuators, peripheral electronics and the one processor can meet any requirements. Program costs are thus reduced.

The development span time is reduced since hardware redesign and new development is not needed. The implementation of control laws into software is not restricted by hardware design time constraints.

Besides the savings in parts, the use of a processor also more readily accomodates the system redundancy necessary to meet the long life requirements at high reliability. With lower parts counts, the power, weight and volume of the system are reduced.

Reliability is a major reason why the processor needs to be developed specifically for spacecraft attitude control. General purpose computers, developed to perform functions far more general than those required for control, do not represent the best approach to achieving high reliability. In general, these computers are too capable (e.g., memory too large, too many instructions, too fast, etc.) to achieve optimum control system application. They also tend to be organized as "reliability monoliths", defeating efforts to divide them to achieve the redundancy necessary to meet reliability objectives.

With a proper examination of the requirements, it is possible to design a processor (and the associated peripheral electronics) that can meet generalized spacecraft long-life control system requirements. This was one part of the purpose of the study reported herein.

## 1.2 Fault Tolerance

Even if the spacecraft attitude control system is designed for long life; faults or failures can occur, which if not corrected, would cause a catastrophic ending to the mission (in so far as attitude control is concerned at least).

In many systems (almost all spacecraft systems, to date), the occurrence of a fault has been noted, either in real time or after-the-fact, upon the ground; and suitable commands are sent to the spacecraft to correct the fault source, usually by a redundancy reconfiguration. Some time necessarily elapses in this process.

In many, more ambitious, missions, this elapsed time can not be accepted due to economic, operational, stability or other constraints. The elimination of, or drastic reduction in, this elapsed time; and the incorporation of the means by which faults are masked, quickly corrected, etc. autonomously on-board the spacecraft; is "fault-tolerance".

Examples of missions requiring or desiring fault tolerance include:

- Communication satellites, where even momentary loss of communication capability (pointing) can be serious
  - Economically - for commercial ventures
  - Security - for military applications
- Military satellites requiring constant pointing and autonomous operation, even with loss of ground station.
- Spacecraft which are passively unstable and can not be operated for long without active attitude control and which may be difficult or impossible to recover from their stable attitude.
- Deep-space probes where the communication times are excessive for interactive ground control of redundancy, and which have:
  - Directive antennas that must be kept pointing at earth.
  - Critical operational events, including planetary encounters, trajectory corrections, etc. that include the primary mission purpose in a "must-do" span of time.

As will be seen in this report, there are many means that can be employed to achieve passive or active fault tolerance. However, to achieve fault tolerance, both the system and (particularly) the processor must be specifically designed for this feature.

### 1.3 Study Scope

The scope of this study and of this report can be seen from the contract statement of work:

"Perform an attitude control electronics mechanization study to develop a fault tolerant autonomous concept for a three axis control system with the versatility to fly on different missions. An existing programmable digital processor should be used to form part of the attitude control system. The processor

should be capable of interfacing with a variety of control components, e.g. celestial and inertial sensors, hot gas valves and thrusters, etc. The system should be for planetary space application with a 10 year life period and heavy emphasis on minimum weight and power. The system shall be capable of at least performing the minimum functions: cruise control, powered flight control, two axis articulating control of science platform, commanded turns, reprogrammable in flight, etc.

The attitude control system to be studied can be of a hybrid or a digital programmable control system. It shall consider the criteria for hardware optimization by the application of active, cooperative, block, functional redundancy, cross-strapping circuitry approaches, failure detection techniques, etc. The effects of an LSI microcomputer in the design shall be considered.

Provide a Final Report on the analysis, recommendations and conclusions from the study."

The existing digital processor design chosen to serve as a baseline for the design studies is the TRW-developed fault tolerant attitude control processor termed COPE (Control Processing Electronics) and sometimes referred to as "Digital Processor Assembly (DPA)". A combination of TRW and contract funds has developed two breadboard versions of COPE.

This design is optimum among existing processors in that it was specifically designed for the type of application and approximately the requirements needed for this study. COPE serves as a baseline, with all tradeoffs of design requirements, implementation, and parameters starting from the characteristics and features already known.

To aid those unfamiliar with COPE, a specification for it is given in Appendix A. It should be stressed that this describes the baseline processor design. The reasons for this design and possible (and desirable) variations from it, will be provided in subsequent sections of this report.

This report describes not only work done under this study contract, but includes work previously done at TRW on the same subject.

It should be noted that it was the stated desire of the customer, JPL, that this report not be influenced by previous work done at JPL, or their conclusions. The desire was for an independent study, which could point out new ideas (or confirm old ones), giving a thorough look at a dynamic subject. In so far as possible, this goal has been accomplished. Although TRW has been generally aware of the work which has been done at JPL (as we maintain awareness of all work in this field) we have not let this influence the study or its conclusions. The tradeoffs and recommendations reported on herein are solely those of TRW.

#### 1.4 Report Organization

The report is generally organized as follows:

- Section 2 defines the assumed mission, the functional and operational requirements and the design constraints.
- Section 3 provides the design considerations and tradeoffs, discussing all viable alternates for each feature or problem of design of the processor, peripherals, software, etc. This section is designed to have semi-universal application, well beyond the more specific mission defined in Section 2.
- Section 4 gives a recommendation for the system design hardware and its parameters, based both on current technology and more advanced developments.
- Section 5 provides a report summary, with conclusions and recommendations for future studies and hardware development.

Intrinsic in the report is the definition of terms or abbreviations or acronyms, nominally at the point of first use. A complete glossary is also included as Appendix B.

Although TRW has compiled an extensive library of reports and articles on the subject, an extensive bibliography and extensive references have been avoided in this report. A brief bibliography of the most useful and current papers has been included. Reference in the text (when used) is by author name.

The difficulty of the comprehensive treatment of such a complex subject, together with the logical explanation of the elements of the subject and their tradeoffs must be recognized. Almost every aspect of the requirements and the design options are intricately inter-related. The cross-combinations of all of these variables are astronomical. For these reasons, a study approach having the following features was adopted:

- Use of a baseline (COPE) processor approach
- Constraint of the mission requirements
- Treatment of each design decision variable as a deviation from the baseline, as independent as possible from the other design decisions.
- Parameterization of hardware tradeoffs in terms of part count (by general part type).
- Parameterization of system, program organization, and other software tradeoffs in terms of program and/or recovery speed.
- Extensive cross-reference between sections.
- Consistent definition of terms.

It is hoped that the reader will appreciate these features as he reads this report. It is our hope that this report will serve as a useful reference on the subject. The intent is to provide the information necessary so that intelligent system-level decisions can be made on the design and organization of fault-tolerant, programmable attitude control systems.

## 1.5 Acknowledgements

The principle authors of this report, and participants in the study were:

A. A. Sorensen, Study Manager & editor  
J. H. Decanni  
A. M. Frew  
J. F. Gregory

Other contributors included:

T. C. Alsbury  
T. C. Berg  
K. H. O'Keefe

Reviewers of the drafts were:

A. E. Sabroff  
D. J. Spencer  
W. L. Graves

Others, doing earlier work on the development of COPE, included:

J. D. Elbert  
L. R. Keranen  
P. F. Smitha  
J. M. Bordyn

A special thanks is due to M. J. Burton and G. Walton for their long hours of typing and proofreading.

## 2.0 GENERAL REQUIREMENTS AND CONSTRAINTS

Fault tolerance and autonomy are two important features required in long-range interplanetary spacecraft systems for the purposes of preventing the occurrence of irreparable failures and precluding reparable failures from causing defaults in critical system functions.

Requirements and constraints influencing the specification and design of fault tolerance and autonomy capabilities derive from

- Characteristics of the interface between real-time spacecraft subsystems and the ground control organization.
- The criticality of potential failures in relation to their consequences on spacecraft safety and operational success within the time interval from the occurrence of a fault to the successful completion of corrective action.

Therefore, the definition of representative mission requirements and spacecraft system configurations is an essential step to the determination of realistic and pertinent design criteria.

The attitude control system configuration selected as a reference for the study must include features that are common to or compatible with most potential inter-planetary missions for the future. Its hardware/software structure should be modular; this provides cost effective adaptability to the specific requirements of each particular mission.

### 2.1 Functional and Operational Requirements

Representative attitude control requirements, used in defining the reference control system configuration, have been derived on the assumption of a multiple outer planet fly-by mission including a Saturn swing-by. However, criteria for the definition of essential features and growth capabilities of the reference configuration include results of an assessment of the control requirements imposed by other long-life missions such as outer planet probe buses and orbiters.

Outbound missions have received preferential attention because of their more extended life and autonomy requirements, compared to inbound missions, which differ primarily in power sources, thermal environment, and communication requirements.

### 2.1.1 The Reference Mission and Spacecraft

A multiple outer planet fly-by, including a Jupiter or Saturn swing-by, is a representative example of a long-life mission for which mission planning and some flight data are available. The scientific objectives of the mission assumed for the study are the exploration of the atmospheres of planets such as Saturn and Uranus, the near space environment of these planets and the interplanetary space from Earth. The positions of Saturn, Uranus and Jupiter are favorable for launches in the 1979-1980 windows.

It is appropriate to note that, with both Saturn and Uranus as target planets, it is Uranus which imposes the more stringent requirements on the system design. The major areas so affected are mission lifetime (arrival at Uranus is about seven years after launch), communications capability (1024 bps telemetry rate with 64-meter antenna), propellant requirements, and onboard navigation requirements.

A unique characteristic of the assumed spacecraft is the use of RTG's as the primary source of electrical power. The communications round trip times at planet encounters are from 2 to 4 times longer than in the Pioneer 10 and 11 Jupiter fly-by. The delay is 2.6 hours at Saturn and 5.4 hours at Uranus. This imposes a significant constraint on command and verification sequences and requires provision of automatic backup modes aboard the spacecraft which are of interest to the study.

Unlike the Pioneer missions which can be controlled with sufficient accuracy by radio guidance from the Earth, a mission to Uranus via Saturn would be too inaccurate without the use of an additional onboard sensor. This is essentially a star sensor which uses stars and bright satellites of the target planets for optical navigation fixes.

The scientific payload includes equipment for particles-and-fields measurements and remote observation of physical properties of the atmospheres and other characteristics of Saturn and Uranus. There is a visual imaging system, mounted on a scan platform together with other instruments requiring gimbaling, which function to provide global views of the planets during the approach phase, obtaining partial views at high resolutions during encounter, and making observations of satellites during the fly-bys. Also, dual-frequency radio occultation experiments will be performed by means of

the dual X-S band communication system of the spacecraft.

An X-band communication system is required to meet telemetry performance requirements at the target planets. Telemetry rates of 2048 and 1024 bps are assumed at Saturn and Uranus encounters, respectively. An S-band, backup down-link capability is needed to support tracking and telemetry operations during initial mission phases, to provide communications over a wider range of attitude errors (or pointing angles) than with X-band, and to permit routine data acquisition from the DSN 26-meter network. The high-gain antenna consists of a 2.75-meter (9-foot)-dia. paraboloidal reflector and a dual S- and X-band feed. This antenna is fixed to the spacecraft structure and looks in the positive roll axis direction. Command access to the spacecraft, regardless of the vehicle attitude, is guaranteed over a limited range of distances by a medium-gain horn and a conical-log spiral antenna coupled to the S-band transmitter/receiver by means of a diplexer coupler.

Spacecraft telemetry functions are managed by the data handling system, which has real time and storage capabilities. Assumed control system sampling rates and data formats are described in Section 3.4.5

Commands are processed by the command system which demodulates the FSK subcarrier signal from the receiver and verifies the validity of each command received. The command interface with the control system is discussed in Section 3.4.5.

In addition to providing three-axis stabilization to the spacecraft, the control system has capabilities to establish any desired orientation (compatible with thermal and sensor field-of-view constraints) and control the attitude during propulsive maneuvers. In addition to attitude control functions, the control system controls the duration of propulsion firings and provides actuation signals to the scan platform drives and attitude data and gimbaling programs to science experiments. More detailed control system requirements and functions are discussed in the next section.

The following are brief descriptions of the assumed sequence of events and operational regimes:

- Launch - The first flight spacecraft is launched in November 1979. Arrival at Saturn is scheduled for April 1983 (3.4 years later) and

Uranus encounter occurs in October 1986. Other flight spacecraft are launched in the next Saturn opportunity (November-December 1980) to perform more advanced missions (probe deployments and orbiters). The launch vehicle assumed is a Titan 3E/Centaur/TE-364-4 and the launch site is at Cape Kennedy. The nominal payload capability of the booster is approximately 455 Kg (1000 pounds). Spin stabilization is used during TE-364-4 burn.

- Separation from Booster, Despin, and Attitude Acquisition - At TE-364-4 burnout, the spacecraft separates from the booster by means of a spring release mechanism. This event is initiated by the spacecraft sequencer. A separation switch activates a despin maneuver, which is the first in the sequence of operations for attitude acquisition. The spin rate is sensed by an Inertial Reference Unit (IRU) which, through the control electronics, operates the Reaction Control Subsystem (RCS) thrusters until the final rate is less than 1-5 degrees/second. Upon completion of the despin maneuver, RTG's and appendages are deployed and the control system executes the Sun and Canopus acquisition sequences.
- Sun-Pointing Cruise - During the first 50 days of the cruise phase, the spacecraft roll axis is pointed directly at the Sun to minimize thermal control requirements. Communications are established through the medium-gain and omni antennas. Three-axis stabilization is by means of a Reaction Wheel Assembly (RWA). The RCS thrusters are used for wheel momentum unloading when necessary.
- Trajectory Correction Maneuvers - Trajectory corrections are executed on days 5 and 20 after launch, 8 days before Saturn encounter, 40 days after Saturn swing-by, and 10 and 2 days before Uranus encounter. The spacecraft is reoriented in the desired thrust direction before execution. During propulsion, the attitude of the spacecraft is controlled by a jet-vane thrust-vector-control (TVC) system and the roll RCS thrusters. Attitude references are provided by the IRU, which also controls the duration of the firings. Upon completion of each  $\Delta V$  maneuver, the control electronics returns the spacecraft to the Sun or Earth pointing orientation by means of the RCS thrusters.

- Earth-Pointing Cruise - After 50 days from launch, the spacecraft assumes an Earth-pointing orientation to allow communications with the high-gain antenna. For routine telemetry operations, S-band provides 32 bps to a distance of 6 AU (on the 26-m system) and 256 bps at Saturn on the 64-m antenna. X-band is used for special events requiring higher data rates.
- Saturn Approach and Encounter - Accurate control of the fly-by distance is essential for reducing trajectory correction  $\Delta V$  requirements. An optical sensor is used to reduce trajectory determination errors to less than 100 km. Titan is a potential navigational reference, whose position can be measured (by a star sensor) relative to several stars. Navigational fixes are needed from 20 to 10 days before encounter and during the first 20 days after the swing-by. Scientific observations are made within  $\pm 10$  days from Saturn encounter. Crossing the ring plane at distances of 2.3 to 2.7  $R_S$  could present a definite hazard to spacecraft survival.

The sequence of operations is very rapid. Wide-angle high-resolution imaging starts a few hours before periapsis. Ring plane crossing occurs in about one hour after closest approach (ring plane survey and RF ring occultation begins). The eclipse and occultation period starts 2 hours after periapsis crossing and lasts 2.4 hours (during which RF occultation experiments and dark side observations are made). Planet environment observations and the ring plane survey continue for several hours after periapsis. Ring occultation lasts approximately 6 hours.

- Saturn to Uranus Cruise - Another sequence of navigation fixes will occur during the first 20 days after the swing-by. A departure trajectory correction will be executed about 40 days after Saturn encounter. The total Saturn-to-Uranus cruise time is 3.8 years and the operational modes are as in the Earth-to-Saturn cruise. S-band telemetry rates are 64 bps (to 14 AU) and 32 bps (to Uranus), using the 64-m DSN antennas. A 1024 bps rate is available up to Uranus on X-band.
- Uranus Approach and Encounter - Navigational observations are required for trajectory corrections to be executed 10 and 2 days before

encounter. By using Titania as a target, and assuming the most efficient estimation techniques available are applied, the residual navigation error is 1200km ( $1\sigma$ ) 20 days before encounter (if the sampling rate is 4 samples per day). At 24 samples per day the error reduces to 500 km. Careful planning of the arrival date at Uranus is required to avoid constraints due to inaccessibility of Titania for navigational fixes. Except for ring observations and potential ring plane penetration effects, encounter operations at Uranus are similar to Saturn's.

- Post Encounter Phase - Normal cruise operations are resumed to extend the mission as far as possible to the outer regions of the heliosphere. Three years after passing Uranus the spacecraft will be at a distance of 30 AU from the Sun and, theoretically, it can continue transmitting data at rates of at least 16 bps up to distances of the order of 80 AU.

#### 2.1.2 Control System Requirements

In the reference mission described in Section 2.1.1, the control system performs the following functions:

- Initial despin
- Sun acquisition
- Canopus search and acquisition
- Cruise attitude determination and control
- Sun/Canopus reacquisition
- Re-orientation maneuvers
- $\Delta V$  impulse and TVC attitude control
- Navigation data acquisition and conditioning
- Encounter attitude determination and control
- Scan platform control

Description of requirements associated with these functions are given in Section 2.1.2.2. where control modes are discussed.

##### 2.1.2.1 Sensors and Actuators

The control system configuration selected as a reference for the study is as shown in the simplified block diagram of Figure 2-1, where redundancies are not included for simplicity.

The wide angle Sun Sensor (WASS) consists of two subassemblies, mounted

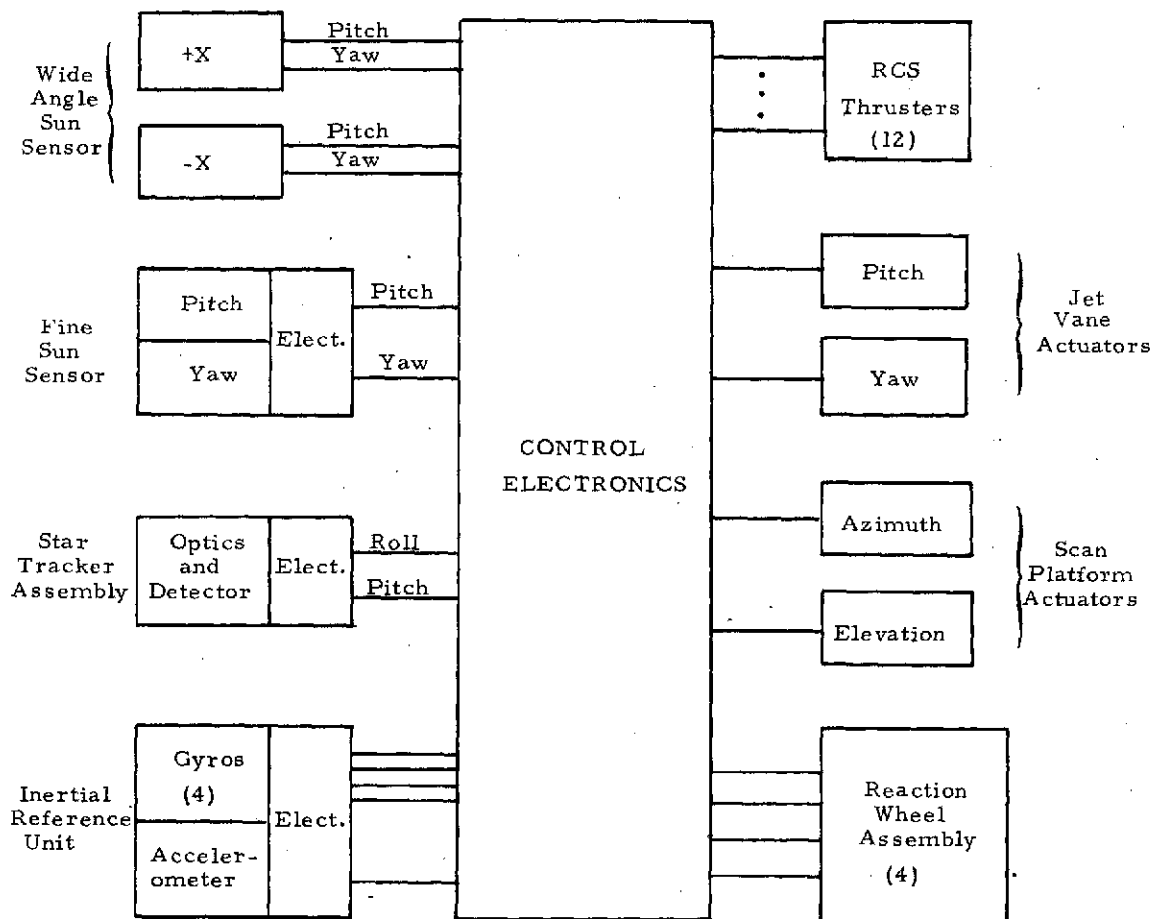


Figure 2-1 Reference Control System Configuration

on opposite sides of the spacecraft, providing a combined FOV of  $4\pi$  sterad. Each subassembly includes four sets of redundant detectors. Each detector includes two square solar cells and a shade configured to restrict the FOV to a region slightly greater than a spherical quadrant. The output signal from each of the four pairs of cells is a bipolar voltage approximately proportional to the cosine of the angle of incidence of the solar radiation. The WASS provides coarse pitch and yaw attitude data for sun acquisition and reacquisition and backup failure detection.

The fine sun sensor (FSS) consists of two single-axis digital sun angle detectors and the associated electronics. Two-axis attitude data are provided over a  $100 \times 100$ -degree FOV with an average resolution of  $1/28$  degree and quantum transition accuracies of  $\pm 0.05$  deg. The FOV assumed exceeds the required values to accommodate the range of sun cone angles typically found in Earth-Saturn trajectories (i.e.:  $\pm 30$  deg after 50 DAL). Measurement accuracies are sufficient to meet X-band antenna pointing requirements at Uranus. For resolutions finer than  $1/8$  deg, the minimum electronics required at the sensor consists of one amplifier, with variable threshold, for each coarse bit. The coarse angle data output is Gray coded. Four quadrature (sine-cosine) fine bit outputs are provided for external interpolation. Serial digital data output interfaces are required.

The star tracker assembly (STA) is a strapped down unit consisting of an optical system, an image dissector tube, and the associated electronics. The STA provides a roll reference, relative to Canopus, during cruise and supplies navigation data (for ground data processing) near the target planets. Requirements are determined by the guidance references, since preliminary estimates indicate a 4th magnitude detection sensitivity is needed and the accuracies for single-point source measurements should be: 2-3 arc sec (bias), 4-9 arc sec (noise).

The inertial reference unit (IRU) comprises a set of four non-orthogonal RIG's, an accelerometer and related electronics. A redundant, non-orthogonal gyro configuration has been selected because it provides an economical, in-line failure detection capability when all four gyros are operational. The accelerometer has its input axis parallel to the spacecraft roll axis to provide acceleration data during velocity correction maneuvers. These

are used to derive measures of velocity increments with which the durations of the firings are controlled. The accelerometer may also be redundant.

A monopropellant-hydrazine reaction control system (RCS) provides control torques for acquisition, reacquisition, momentum unloading, maneuvers, and overriding the reaction wheels in cases of emergency. The RCS comprises a set of 12 to 16 catalytic thrusters (with the corresponding valves), the propellant tanks, four solenoid-actuated latching valves, two filters, two  $N_2H_4$  fill valves, two  $N_2$  fill valves and two sets of propellant lines. The system is organized in two separate banks of thrusters which can be fed by either one of the propellant tanks. Propellant expulsion is forced by bladders pressurized by nitrogen gas. The thrusters provide thrust levels on the order of 0.5 N and, when the two banks are active, pure couples about each one of the control axes can be developed. Included in the RCS, but not shown in Figure 2-1, are the axial thrusters (provided with two-axis, jet-vane TVC equipment) and the associated hardware. The jet-vane actuators are limited-travel, spring-restrained DC torque motors provided with LDT shaft angle transducers. When the motors are de-energized, the jet vanes automatically return to their central positions. Limit switches provide redundant indication of hard-over driving conditions. Each axial thruster develops a nominal thrust level of about 45 N and is aligned so that the undeflected thrust vector passes through the mean CM location. Two actuators are required per thruster to provide pitch and yaw thrust vector deflections.

Azimuth and elevation scan platform axes are driven by geared stepper motors. Reductions include conventional gears and a harmonic drive to produce output shaft rotations in the range from 0.02 to 0.05 deg per step. Shaft position indication is provided by a multi-speed resolver. Limit stops are provided so that redundant position information can be derived by counting actuator pulses. Maximum slewing rates in the range from 1 to 2.5 deg/sec are obtainable.

The reaction wheel assembly (RWA) consists of a set of four non-orthogonal reaction wheels. Functional redundancy is provided in the sense that any combination of three wheels can be used for full three-axis control. All wheels are identical, with maximum capacities in the range from 1 to 6 Joule-sec. The wheels are driven by two-phase induction motors developing

peak torques on the order of 0.15 N-m over the operating speed range.

#### 2.1.2.2 Modes

In various operating regimes of the mission there are control system functions which have similar characteristics and, thus, can be implemented by means of a single mode. Each mode entails a particular sequence of operations using a specific, combination of sensors and actuators. The following modes will be implemented.

##### ○ Standby

The control system is in a quiescent state in which only command and telemetry functions are performed. This mode, initiated or terminated by ground command, is normally used during preflight operations and through launch phases prior to separation of the spacecraft from the upper booster stage.

##### ○ Despin

At TE-364-4 stage burnout the spacecraft spins at 60 rpm about the roll axis. Upon receipt of a signal from the separation switch, the control system initiates the despin maneuver. The roll thrusters are operated until the roll rate is reduced to a preselected value in the 1-to-5 degree/second range. During despin, rate information is provided by the IRU, which is activated by the spacecraft sequencer, prior to separation from the booster, to allow sufficient time for gyro spin-up. Breakdown of the despin maneuver into two or three partial despin and nutation damping operations may be required to limit transversal rate growth due to misalignments and/or mass property anomalies.

##### ○ Acquisition

After deployment of the RTG's, magnetometer boom, and other appendages, the spacecraft sequencer commands the control system to start acquisition maneuvers. Sun acquisition begins with activation of the roll attitude control loop for reducing the roll rate to a value comprised within a predetermined deadband. The IRU provides roll rate information and torques are produced by the RCS thrusters in response to signals developed by the control system electronics. After the roll rate is within the deadband over a specified time interval, the pitch

and yaw control loops are activated to proceed with the sun acquisition. Sun attitude information is supplied by the wide angle sun sensor and control torques are developed by the pitch and yaw thrusters. During the entire acquisition maneuver the roll channel maintains the roll rate within its specified deadband. As soon as the sun pointing error is reduced to an amplitude within the acquisition deadband, the pitch and yaw loops are switched automatically (by the control electronics) to operation with the two-axis Fine Sun Sensor (FSS) and the control deadband is reduced to a smaller preselected value.

The Canopus search and acquisition sequence is initiated automatically by the control electronics, following sun acquisition. If desired, the automatic sequence can be overridden by ground command, and acquisition can be executed under control by the ground software on the basis of telemetry data provided by the control electronics. In the automatic mode, the control electronics commands the RCS thrusters to establish a preselected roll rate (in the 1-5 degree/second range) while the pitch and yaw control loops maintain the Sun pointing orientation. Roll rate information is provided by the IRU and pitch and yaw pointing error signals are obtained from the fine sun sensor. The Star Tracker Assembly (STA) operates in the acquisition mode, providing star crossing and star magnitude data to the control electronics for either onboard processing or telemetry to the ground. When Canopus enters the FOV of the STA, the roll search is terminated and roll axis control is switched to the normal pointing mode, with error signals provided by the STA.

- Celestial Pointing

This mode can be entered either automatically, by a discrete event timer command from the control electronics, or by either sequencer or ground command. Pitch and yaw pointing error data are provided by the FSS and a roll reference is obtained from the STA, which is nominally locked on Canopus. Control torques about all three spacecraft axes are provided by the RWA. The RCS thrusters operate automatically for either momentum desaturation or overriding the RWA if pointing errors exceed pre-established deadbands. After 50 days from launch, the spacecraft is transferred from the Sun-pointing to the Earth-pointing mode for starting communications with the high-gain antenna. The yaw

and pitch control loops are biased to account for the Sun-Earth-Canopus-spacecraft geometry. These biases are functions of time generated by the control electronics on the basis of stored parameters, adjustable by ground command and sequencer timing inputs. Pointing accuracy requirements during cruise depend on communications ERP constraints, operating frequency, and antenna characteristics. Assuming a requirement to point at Earth within 10% of the half-power beamwidth, the corresponding attitude errors are 0.3 degree for S-band and 0.13 degree for X-band.

o Inertial Pointing

Re-orientation maneuvers are required during cruise for scientific observations and terminal guidance. Also, for performing trajectory corrections the spacecraft has to point with the roll axis in the desired direction of thrust. Re-orientations are performed on the basis of a program, stored in the control electronics, which is adjustable by ground command. A typical sequence includes a spacecraft rotation to the desired orientation, execution of programmed events, and return to the Earth-pointing attitude. A three-axis attitude reference is provided by the IRU and control torques are developed by the RWA and the RCS thrusters. For thermal reasons, maneuvers within the first 50 to 100 days of flight are constrained to keep the Sun in the + x hemisphere.

The control electronics also controls the durations of the firings by integrating an accelerometer signal provided by the IRU and comparing the resulting velocity change estimation against a stored limiting value input by ground command.

Guidance data are acquired by pointing the STA in specific directions so that the selected planet satellites and reference stars fall within the FOV of the instrument. These operations require a sequence of maneuvers (i.e., spacecraft rotations ranging from 70 to 110 degrees) programmed by the control electronics as for trajectory corrections. An essential requirement during data collection is that the spacecraft rates should not exceed a few degrees per second. During observation periods the control electronics samples the STA and IRU output signals,

performs formatting operations, and outputs the resulting data for storage in the Data Handling Subsystem's memory for telemetry to the Earth after returning to the Earth pointing orientation. High resolution imaging and other scientific experiments also require stabilization with low rates. In addition, eclipses and interference due to planet albedos may either prevent or significantly restrict the use of optical sensors during critical phases of the fly-bys.

During encounter operations, attitude references will be provided by the IRU. Drift corrections will be computed by the control electronics on the basis of star data obtained from the STA. Planet interference (and critical shading requirements) can be avoided by operating with stars more compatible with viewing constraints than Canopus (if necessary). Spacecraft maneuvers may be required to provide adequate look angles to both the experiments and the STA.

- Thrust Vector Control

Pitch and yaw attitude control during axial engine firings is performed by means of a jet-vane thrust vector control system. Roll torques are provided by the RCS thrusters to prevent the roll attitude errors from exceeding a pre-established deadband. The RCS deadband is wider than the ones used in the celestial and inertial pointing modes.

In addition to the functions included in the six primary operation modes described above, the control system is programmed to perform the following functions upon demand:

- Sun/Canopus Reacquisition - Sun reacquisition is initiated by confirmed loss of the fine Sun reference and is executed by means of the RCS thrusters. The same initiation and execution criteria apply to Canopus reacquisition, which differs from initial acquisition in the search strategy (which covers a smaller region for fast recovery).

- Attitude Determination During Cruise - FSS and STA data samples, at the rates required by scientific experiments, are processed and provided to telemetry in response to ground or sequencer commands.

- Attitude Determination During Encounter Operations

FSS, STA, and IRU gyro data samples are provided to telemetry. Sampling

rates are adjusted to meet the requirements imposed by scientific experiments.

- Scan Platform Control

The control electronics generates signals for driving the scan platform actuators to either fixed preselected positions or to follow desired scan profiles preprogrammed by ground command.

- Support to Science Payload and Other Subsystems

Discrete event timing (related to maneuvers or control sensor observations) and attitude and gimbal angle data are provided to the scientific experiments. Also, preprogrammed gimbal command profiles are supplied to experiments including individual gimbaling equipment.

Limited backup data processing can be provided in the event of data handling system failures.

- Test and Checkout

This function is used during integration and testing and preflight checkout. The control electronics interfaces directly with ground test equipment for performance verification and other system and subsystem tests.

- Housekeeping

This is an automatic monitoring function providing instrumentation and diagnostic data to telemetry, from which performance of the equipment can be analyzed and detailed diagnostics of failed units can be made by the ground software.

### 2.1.2.3 Ground Support Capability

The ground terminal for flight operations is provided by the NASA Deep Space Network (DSN), which utilizes stations covering several longitude bands. At the time the reference mission is flown, there will be complete networks of three or more stations (approximately 120-degree spacing) with both 26- and 64-meter antennas and the capabilities will be as follows:

Antenna Diameter	Station Locations	Uplink	Downlink
26 m	California Australia Spain South Africa	S-band, up to 10 kW. 1 to 10 sps with either FSK or PSK	S-band 2500 bps (uncoded) 2500 bps (block) 2048 bps (conv.)
64 m	California Australia Spain	S-band, up to 400 kW. Same rates as above	S- and X-bands 33 kbps (uncoded) 33 kbps (block) 2048 bps (conv.)

Ground station support can be available up to 24 hours a day if necessary. However, during the cruise phases, coverage may be reduced down to a few hours per week. The 26-meter stations have redundant facilities which can provide non-interrupted S-band support.

Spacecraft telemetry is managed by the data handling system in either real time or via mass memory storage. Downlink data rates available vary from 16 to 2048 bps.

The data handling system provides several format options for science and engineering, all selectable by ground command. The choice of science format depends on data rate required (e.g.: cruise, encounter, imaging) and data source (i.e.: real time, mass storage). In addition to the regular cruise and encounter engineering formats, there are formats available for accelerating portions of the engineering subcommutator channel to the main frame. As usual, the main frame includes control and synchronization data, fixed words, and the engineering and science subcommutated data.

The command system can provide serial, discrete pulse, and discrete level commands. The command format includes a preamble, a synch bit, address bits, the command message, and parity check bits. Command messages are typically from 8 to 16 bits long.

Assuming 8-bit command messages, a typical command word size may be 22 bits long. As a rate of 10 bps for the uplink, the maximum average data rate input to the control system will be about 3.6 bps.

In addition to real-time command processing, the command system includes a memory and logic for executing stored commands at predetermined intervals.

#### 2.1.2.4 Miscellaneous

Round-trip communication times are 8.306 minutes per AU. This makes interactive operations with the ground very difficult when the spacecraft is close to the target planets. For instance, command verification requires 1.44 hours from Jupiter, 2.64 hours from Saturn, and 5.31 hours from Uranus.

Radiation environments of the outer planets are potential sources of failures in the electronic equipment. Electron and proton fluxes are entirely due to the planetary radiation belts. Pioneers 10 and 11 were designed for fluences of  $5 \times 10^{10}$  electrons/cm<sup>2</sup> (5-100 MeV) and  $5 \times 10^{10}$  protons/cm<sup>2</sup> (0.1-4 MeV). For a Jupiter swing-by mission, maximum estimated fluences (based on SP-8091) are  $1.5 \times 10^{11}$  electrons/cm<sup>2</sup> (1-10 MeV) and  $3 \times 10^{10}$  protons/cm<sup>2</sup> (2-100 MeV). These fluences were determined by integrating fluxes along fly-by trajectories approaching planet centers within  $2.3 R_s$  and  $3 R_j$ .

The existence of radiation belts at Saturn and Uranus has been postulated based on the observation of UHF radio emissions and on the assumption that the mechanism producing radiation belts is similar to Jupiter's. However, as discussed in the Design Criteria Documents NASA SP-8091 and SP-8103, it is plausible that the densities, fluxes, and energies are much weaker than at Jupiter. With fly-by distances of  $2.3$  to  $2.75 R_s$  and  $4 R_u$  it is unlikely that radiation belt exposure should present a significant hazard to the spacecraft.

Micrometeoroid impacts during the crossing of the plane of Saturn's rings is a matter of concern regarding spacecraft survival. A preliminary estimate of an upper bound of the integrated flux of particles of various sizes was obtained based on the current model of particle flux density in Saturn's range (SP-8091). Results are given in the following table:

Min. Particle Radius (cm)	Min. Particle Mass [g]	No. of Impacts
0.1	0.0042	4.7
0.133	0.01	3.1
0.287	0.1	0.96
0.62	1.0	0.31

An exposed area of about  $1.9 \text{ m}^2$  was assumed, with the vehicle's relative velocity (11.4 km/sec) included relative to the ring particles velocity vector such that particles enter the vehicle from the rear, at an angle of 45 deg from the -X axis.

## 2.2 Design Constraints

### 2.2.1 Life and Reliability

The spacecraft systems must be designed with all expendables and known wearout phenomena sized for 10 years. Although there may be some unanswered questions which require additional study, these do not appear to be unmanageable problems at this time.

For instance, the star trackers may be subject to lens fogging (from long-term space radiation) and image dissector tube degradation. None of these are felt to be serious problems because integrated solar radiation and electron and proton fluences are either comparable to or lower than Pioneers 10 and 11 and 10-year lives are within the state of the art of dissector tubes.

There may be some concern about possible degradation of seals and valve seats in the thruster assemblies. However, hydrazine thrusters have demonstrated 500,000 pulse lives without leakage. Typical numbers of pulses for the reference mission will be in the 10,000 to 30,000 range. Proper thermal control will provide assurance of seal/seat integrity.

Catalyst bed degradation from cold starts can be a more serious problem since typical design lives are on the order of 150 cold starts. Extension of this limit to several thousand cold starts has been achieved by means of heaters. Either resistive or radioisotope heaters can be used.

Sun sensors include silicon solar cells or other photosensitive elements which can be degraded by exposure to electron and proton fluxes. Experience with Pioneer 10 shows this is no problem.

The primary source of concern from the reliability standpoint is the 10-year lifetime requirement. This is about four times the design life for Pioneers 10 and 11. In the reference mission, reasons for concern are mitigated by the following factors:

- Integrated exposure to solar originated environments is not increased due to inverse square-law effects.
- The worst expected radiation dosage is due to passage through the Jovian radiation belts.
- The extended part of the mission entails a relatively more benign environment in terms of temperature extremes and anticipated micro-meteoroid flux.
- Failures not related to depletion or wearout factors should have lower incidence rates during the extended part of the mission.

The reliability problem associated with 10-year mission lifetimes is shown by Table 2-1, which is based on Pioneer 10/11 reliability models, with extensions of the same failure rates to 10 years:

Table 2-1 Pioneer 10/11 Spacecraft Reliability (for 10 years)

Spacecraft System	Reliability
Structures and Ordnance	.9943
Thermal	.9971
Propulsion	.9112
Attitude Control	.8035
Data Handling	.6131
Antennas	.9673
Communications	.7676
Power	.9419
Command Distribution	.7835
<hr style="border-top: 1px dashed black;"/>	
TOTAL SPACECRAFT	.2439

The average system reliability is 0.8549 but the total spacecraft reliability is an unacceptable 0.2439. This extension is probably unduly pessimistic, because failure rates are based on data for shorter missions, including a number of earth-orbiting spacecraft programs. Thus, infant failures and failures in the temperature-cycling earth-orbit environment tend to produce failure rates which are too high for a long mission with an unchanging environment. The 0.2439 figure is neither a realistic nor an acceptable estimate of the probability of mission success. Nevertheless, the calculations point out which are the design areas where reliability augmentation would be comparatively most effective.

Reliability requirements for the control system depend on the reliability goal established for the entire spacecraft and the reliability allocations to other systems. Curve A in Figure 2-2 shows system reliability requirements and the assumption that all system reliability allocations are equal. Curve B corresponds to the case where the structure, thermal, antenna, and power systems have a combined reliability of 0.93 and the remaining systems have equal reliability allocations. Curve C was obtained assuming the control, propulsion, and data handling systems have equal reliabilities and the rest of the system has a reliability of 0.6524.

Typically, overall spacecraft reliability requirements must be in the 0.5 to 0.6 range (or better). As shown by the shaded area in Figure 2-2, control system reliabilities in the range from 0.88 to 0.94 will be required for meeting this system reliability objective.

### 2.2.2 Availability and Recovery

The first problem to be considered in the design of a fault tolerant control system is the determination of the maximum time that the system can be allowed to be down without compromising the safety of the spacecraft or causing operational mission failures. The spacecraft can be physically endangered when:

- The propulsion system is allowed to cause excessively high rotational rates.
- The thermal control system is subjected to damaging heat inputs as a result of prolonged exposure to the sun (during the early part of the interplanetary cruise).

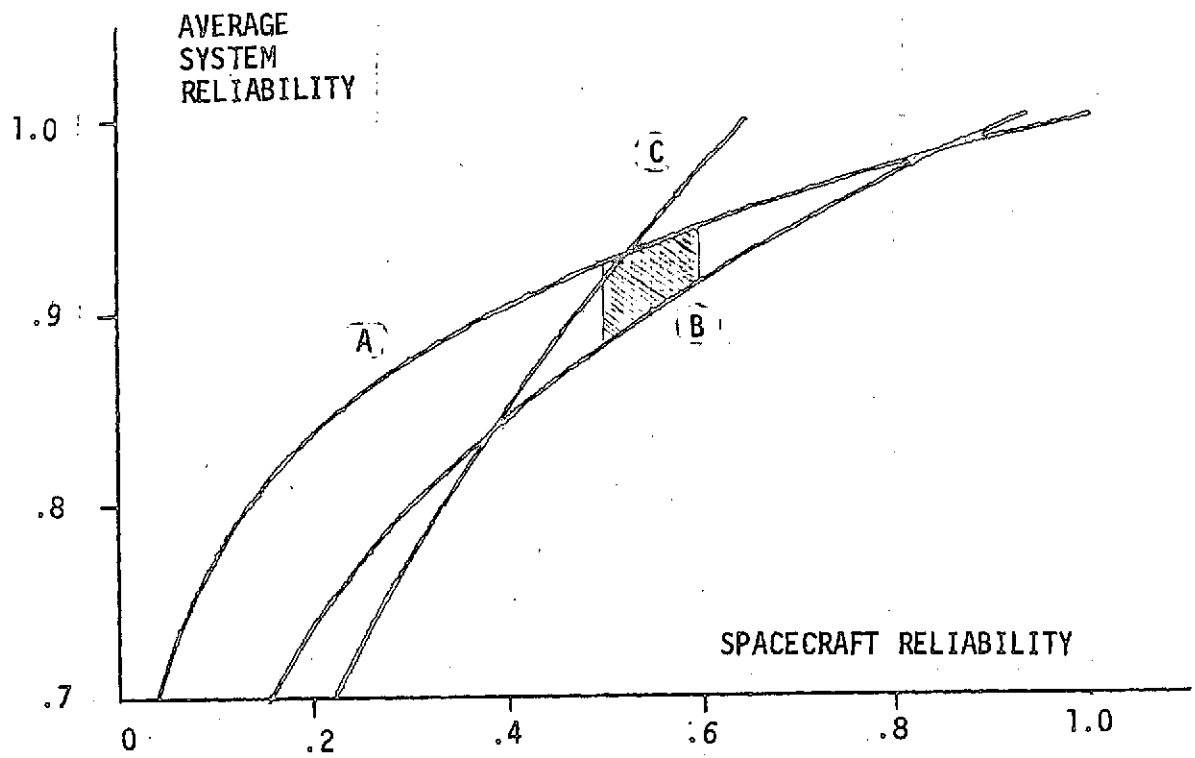


Figure 2-2. System Reliability Requirements

Operational mission failures can be caused by

- Loss of communications due to failure of the control system to return the spacecraft to a stable Earth (or Sun) pointing condition after off-pointing maneuvers. This can be caused by either excessive spacecraft rates preventing acquisition, or by failures causing stable pointing in unscheduled directions.
- Excessive propellant leakage caused by failures resulting in open valves.
- Excessive midcourse velocity correction magnitudes due to engine cut-off timing failures.
- Failures to point spacecraft/scan platform correctly (during critical encounter phases) producing losses of scientific data.

In normal mode (cruise) conditions, attitude control is accomplished by means of reaction wheel torquing. Assuming wheel motor torques on the order of 10 in-oz and minimum spacecraft moments of inertia of 350 slug-ft<sup>2</sup>, the acceleration caused by a hard-over wheel drive failure would be  $8.5 \times 10^{-3}$  deg/sec<sup>2</sup>. Assuming a wheel saturation momentum of 1 ft-lb-sec, the maximum spacecraft rate of 0.16 deg/sec can be attained in 192 sec, at the end of which the angular travel would be about 157 deg. If the fine sun sensor has a FOV of 40° x 40°, the time it takes to go from the optical axis to a FOV edge is 69 sec. Consequently, a recovery time on the order of 30 seconds is probably sufficient to handle wheel drive hard-over failures. The worst that can happen after recovery is a reacquisition by means of the RCS thrusters.

If a failure in the ACS electronics causes an RCS thruster or pair of thrusters to fire continuously, the resulting spacecraft rotational velocity is not bounded, as in the reaction wheel case, but increases linearly with firing time. Assuming a thrust level of 0.1 lbs and a moment area of 8 ft, the acceleration caused by a constant couple is 0.13 deg/sec<sup>2</sup>. A rate of 10 deg/sec can be attained in about 77 sec. The FOV of the fine Sun Sensor will be left in less than 18 sec. Recovery times no greater than 9 sec are desirable to permit reacquisition within the fine sun sensor FOV (initial sun pointing condition assumed with zero bias.)

During velocity correction maneuvers, shorter recovery times are required due to the higher bandwidths and control torques involved. With a spacecraft weight of 1000 lbs and a thrust level of 10 lbs, the firing time for a 164 ft/sec velocity maneuver is 8.5 min. If the moment arm is 4 ft, the torque produced by a 5-deg thrust vector deflection is about 3.5 ft-lbs, more than 4 times greater than the assumed attitude control thruster torques. The corresponding acceleration will be  $0.57 \text{ deg/sec}^2$ . Rotational rates of 10 deg/sec can be attained in 17.5 sec and attitude errors greater than 2 deg will be developed in less than 3 sec. In this case, recovery times on the order of 1 sec or shorter are desirable to prevent the development of dispersion errors requiring additional or larger correction maneuvers.

Scan platform actuators of the incremental type operate with step rates typically ranging from 0.02 to 0.05 deg. Assuming a maximum stepping rate of 50/sec, the corresponding slew rates will be from 1 to 2.5 deg/sec. Recovery time requirements imposed by scientific experiments are not believed to be critical unless there is equipment that can be damaged by accidental exposure to particles and fields without protection. If uncontrolled slewing is to be limited to  $\pm 30$  deg per axis for this reason, recovery times should not be greater than about 10 sec.

The preceding discussion shows that, in the assumed interplanetary mission, failure recovery is not required to take place faster than within about 1 sec.

The essential problem is to implement a failure detection and recovery concept that can function reliably without ground intervention. Autonomy is more important, in this class of applications, than speed of recovery.

### 2.2.3 System Safety

The safety of the spacecraft can be compromised by either hard-over failures resulting in prolonged actuator operation or by unscheduled interruptions of the attitude control functions causing either excessive heat inputs or damaging exposure of systems and components to environmental radiation over extended periods of time.

For safety, the following fail-safe design policies should be observed:

- The spacecraft systems should be designed with appropriate redundancy, workarounds and backup capabilities which will eliminate as many electronic, mechanical, and electromechanical failure modes as sources of spacecraft failure as practical. When redundancy or backups are employed, circuits, interfaces between units, and switching circuits should be designed with fault isolation so that a failure in one unit does not propagate into, or does not interfere with the operation of, the redundant units or backup modes.
- The spacecraft systems should be designed so that they can survive failures occurring when the spacecraft is not being monitored by the ground, since those that occur in deep space require long periods for telemetry detection, problem diagnosis and corrective command transmission.
- Electrical or electromechanical random single-point failures should be eliminated from equipment which must successfully operate at a high duty cycle throughout the mission, and from equipment which is especially critical to the success of the mission. For the purposes of this single-point failure criteria, failure or degradation from predictable wearout should not be regarded as random, and the design should be capable of surviving a single-random failure in addition to expected wearout failures.
- Failures in the control electronics should not cause hard-over conditions or prolonged operation of the actuators. Long duration thruster firings should require a sequence of turn-on commands instead of commands for turn-on and turn-off.
- In the absence of control signals, TVC actuators should return to centerline positions.
- Reaction wheel control should be overridden by RCS thrusters when control errors exceed predetermined tolerances.
- Failure recovery times of the control electronics should not exceed maximum durations allowing effective action against actuator failures producing sustained torques. (See Section 2.2.2).

- o All systems should have provisions for overriding automatic functions by ground command.

The above policies are intended to be general guidelines to system design, but are not intended to be inviolate or inflexible. In implementing fail-safe techniques in specific cases, competing factors -- such as cost, practicality, weight, redesign or repackaging of existing equipment, possible introduction of higher probability failures, increased risks of operator errors, etc. - should be taken into account and may possibly weigh heavier than the above listed criteria.

Functional redundancy and backup techniques should carry more influence in establishing the spacecraft design than numerical reliability calculations, because the assumptions underlying numerical calculations - exponential probability of failure - have questionable applicability to this class of missions. (Also see Section 3.7)

#### 2.2.4 Growth Potential

Growth potential is an essential feature in a multi-mission control electronics concept since a successful approach is not usually the one including every possible feature that may be needed in any of the future missions anticipated. Particularly when weight and power requirements are major areas of concern, it is desirable to have a modular configuration that provides a group of capabilities that is essential and common to most of the missions and equipment complements considered and also allows adaptation to the specific requirements of each application by addition of new modules and components.

Flexibility can be enhanced by making maximum use of software techniques for operational and housekeeping functions and by concentrating mission dependent interfaces in individual units of modular design that can be easily isolated from the rest of the system for development purposes.

One of the processor architectures that is most adequate for multi-purpose systems is the one where modularity is used to separate the application-dependent and technology-sensitive elements (e.g. memories, I/O units) from those which are needed in all cases (i.e: ACU) or whose design is not affected by mission or equipment changes (e.g: RCU, HCU).

A flexible memory organization is necessary because functional and fault tolerance requirements not only affect memory size requirements, but also may have a significant influence on operational characteristics (e.g: error correction, duplex operation).

Critical functions of growth missions should not be considered among the design criteria for a common baseline configuration but should be taken into account in the definition of requirements for special purpose interface units.

#### 2.2.5 System Autonomy

Autonomy requirements are determined primarily by the ground supervision constraints imposed by the long round-trip communication times encountered in long-range planetary missions.

The reliability of an autonomous system depends not only on the availability of equipment to perform the required functions through the entire mission life but also on the short-term operational stability and predictability of the system. The more autonomous a system is made, the more difficult it is to identify and evaluate its operation modes in cases of failures or when subject to disturbances. Due to technical and economical reasons, there are always limits to the amounts of testing and the effectiveness of tests which prevent a complete identification and evaluation of all the operation and failure modes that self-configuring system can have. Consequently, there will always be some degree of uncertainty regarding the ability of an autonomous system to perform its duties during critical phases of a mission without any ground supervision. Deciding what is the degree of independent decision making capability that a system should have is one of the key tradeoffs required in the design of a fault-tolerant, autonomous system. Intuitively, providing the minimum amount of intelligence that is necessary to guarantee a reliable recovery from most foreseeable failures and disturbances is a desirable approach, since this facilitates testing by reducing the number of operation modes and, most importantly, the system behavior would be more predictable.

Since not all missions, or mission phases, have the same autonomy and recovery requirements, design flexibility is desirable for adjusting or programming the system to provide the right amount of adaptivity for each specific case.

In a given system with specified autonomous functions, the following techniques can be effectively used to minimize operating mode uncertainties and improve testing capabilities:

- o Organize operations to be performed in a regular, predetermined sequence. This eliminates priority conflicts, facilitates scheduling of system resources (e.g., input/output devices), and minimizes fault propagation.
- o Minimize conditional interrupts. Whenever possible, critical functions requiring high-speed decisions should be handled by special purpose modes with fixed sequences of events. These modes should be accessible only when they are required.
- o Reduce number of resident modes. The number of operating sequences admissible should be reduced to a minimum. Reprogramming features should provide the capability to change the modes resident in the processor memory as required for each mission phase. The system may have a large number of operating modes throughout the mission but, at any given time, the number of modes available should be kept to a minimum.
- o Remove critical interfaces from the digital processor. Operations requiring wide bandwidth, fail-safe provisions, or unique capabilities (in terms of either operating risk or design requirements), should be handled through interface elements designed specifically for each application. These interface units not only provide compatibility between the operating hardware and the digital processor but also reduce the number of failure modes.
- o Do not attempt to start the execution of a job unless the processor is operating properly and the preceding task has been successfully completed. The number of failure modes that may occur, and their complexity, are direct functions of the amount of work that a machine is allowed to perform without supervision. Failure propagation is usually the cause of diagnostic/recovery confusions leading to permanent system down conditions.

- Recovery approach. For simplicity and expediency, recovery should be effected by switching redundant elements until a failed function is restored. No detailed diagnostics should be made in line during the recovery process since they are usually complex and time consuming. Whenever possible, peripherals should have provisions for allowing the software to determine their operational status.
- Concentrate functions of similar natures in single units. Failure modes and diagnosis and testing are simplified when only one unit is required in operation for performing related kinds of tasks. For instance, the ACU's should be self contained, not requiring the IOU's in order to be able to perform arithmetic functions. Similar considerations apply to recovery, input/output, and storage functions.
- Interleaved checkout capabilities. System and processor testing are more realistic when checkout routines can be executed without modification of the normal sequence of operations for each mode.
- Mode control. Mode control must be safe and reliable. Accidental mode transfers due to system or processor faults should be prevented. Transfers should not be made unless originating causes are confirmed.
- Backup reconfiguration mode. A redundant and independent backup reconfiguration mode, implemented by hardware without interfaces with the digital processor, is desirable to provide an alternate path for initiating reconfiguration in the event that the normal channels are disabled by multiple failures.

#### 2.2.6 Physical

The physical constraints on system design are well known by every design engineer. They are size and weight and power consumption (and dissipation). Of these, weight and power consumption are most important and are inter-related at the spacecraft level. That is, more weight can be used to create more power, and lower power consumption requires a smaller, lighter power system.

Not only are there maximum constraints (the spacecraft must fit in its shroud and be less than a certain weight to lift off), but there are tradeoffs within these constraints. As the supporting systems consume less size, weight or power, more is available for the payloads.

It is always desirable, therefore, to minimize the physical characteristics as much as possible, within the other constraints of technology availability, schedule, and (particularly) cost.

It is not just the particular system set of physical parameters that should be minimized, but those of the entire spacecraft. This demands a degree of non-parochialness beyond that usually encountered in human endeavors. For example, the weight of all the wiring connecting the control system within itself and to other systems must be considered, and circuitry may be required (or at least traded-off) to reduce this wiring. The interfaces between all "boxes" are important to this trade.

It is also important to minimize the number of different power supply voltages used and the degree of regulation necessary so as to reduce the requirements on the power system and make its design better.

The method of electronics packaging chosen has a large influence on size and weight (even with the same circuits). Obviously, this is greatly influenced by environmental, EMI, schedule and cost considerations also.

Some continuing means should be provided to trade-off (and monitor) the total weight and power to make sure that they are being minimized (and controlled) within the other constraints.

Care must also be taken that one does not go too far in this area, as that will increase the cost astronomically, may not help the reliability, and will reduce the possibility of applying the designs to other programs. It is good to design with a modularity to accept changes (program or technological).

#### 2.2.7 Technology

For the purposes of this report, the technology constraints are in two phases:

- o Parts, devices or circuits that are available now, that can be bought in quantity, made under high-quality programs, and that have been (or could be) qualified for spacecraft use. This includes existing LSI.

- Parts or devices that could be made within a few months, using existing technology and processes. This includes LSI in new topological configurations, but made with processes now in use.

The report will not concern itself with blue-sky or salesmen's-dream technology, as this cannot be confidently applied to near-term projects.

Note that control systems, programmable processors, fault tolerance, or other subjects of this study are not technology limited. No new devices or breakthroughs are needed. New devices can and will help, but what is available today is sufficient to provide autonomous fault-tolerant programmable digital control systems.

#### 2.2.8 Other

There are many other constraints besides those discussed in the preceding sections. Often they are more constraining than the operational, physical, or technological ones. They can include cost (more about this later) schedules, existing equipment availability, environmental considerations, and political (organization-imposed) constraints.

Environmental considerations are usually no different for this type of system than for any other in a spacecraft. One difference can be radiation and the effect it may have on these essential functions. Another could be arc discharges and the possibility of these creating transient faults.

Often, to exercise either real or false economy, existing equipment is utilized in new system designs. This is usually the sensors or actuators in control systems. It will be shown later that little or no modifications to this equipment is necessary for it to operate well in a fault tolerant control system using a processor.

Cost is almost always the overriding constraint. Everything else is equatable into cost. Usually the cost curves (for any feature or desire) exhibit a certain increase in desirability (of whatever) proportional to cost increase at a given slope, up to some point and then the slope changes with decreasing improvement for ever more cost. It is important to stay below the knee of this curve.

The economics of all design decisions made are very important. There is always a tendency in the design of a system for that system to grow (always in desirable features) up to (or beyond) the limits of cost, whatever those limits were. Each added feature or convenience is always desirable (to a degree).

In the design of a fault tolerant processor this law must also be observed. One can always continue to add features to speed up the operations, simplify the programming, shorten fault detection time, increase the degree of fault detection, reduce the power or weight, etc.; but each of these cost more money. Adding unnecessary features or characteristics must be strongly guarded against.

In this report the aim is to critically examine the need for each requirement. Also the penalty of each feature is given. The unique tradeoffs may then be made as a part of optimizing the system design.

Note that there is no single optimum. The optimization is always relative to the particular weighing of the occasion.

### 3.0 DESIGN CONSIDERATIONS AND TRADEOFFS

This section comprises the major portion of the report. It is concerned with the various tradeoffs necessary to create a design for a fault-tolerant, programmable digital control electronics system, which enables the subsequent optimization of that design.

As noted earlier, the COPE processor is chosen as the baseline design, with each tradeoff considered as a variation from COPE (where appropriate).

The subjects are closely interrelated and mutually dependent. For convenience in organization, the sections have been divided as follows:

- Redundancy Policy. How the system redundancy subdivision and control is obtained. Redundancy management and cross-strapping design are covered.
- System Design. The control system implementation, interfaces, and criteria for functional redundancy, fail-safe design, and power interruption, are developed.
- Hardware/Software Requirements. The requirements of Section 2 are developed into processor hardware and software needs.
- Processor Design. Design tradeoffs in processor hardware design are explored.
- Software Design. The software design organization tradeoffs are presented.
- Hardware Fault Detection. This covers processor and peripheral fault detection techniques.
- Reconfiguration. This section covers the complex subject of reconfiguration criteria and techniques.
- LSI Applications. The effect of further LSI usage is discussed.

### 3.1 Redundancy Policy

The purpose of redundancy is to achieve the specified reliability for the system. Reliability is a mathematical measure of proper system performance. Reliability is the probability of successful system operation for a given time period. The reliability of the system is made up of (the product of) the reliabilities of the individual portions of the system. In a non-redundant system, the reliability is a measure of the individual failure rates of those portions, down to the part level. Any part failure results in a system failure.

"Redundancy" is something added, which is not needed except to satisfy some contingency. If no faults or failures occur, the redundancy is/was parasitic and uneconomical. A classification of redundancy types is shown in Figure 3-1.

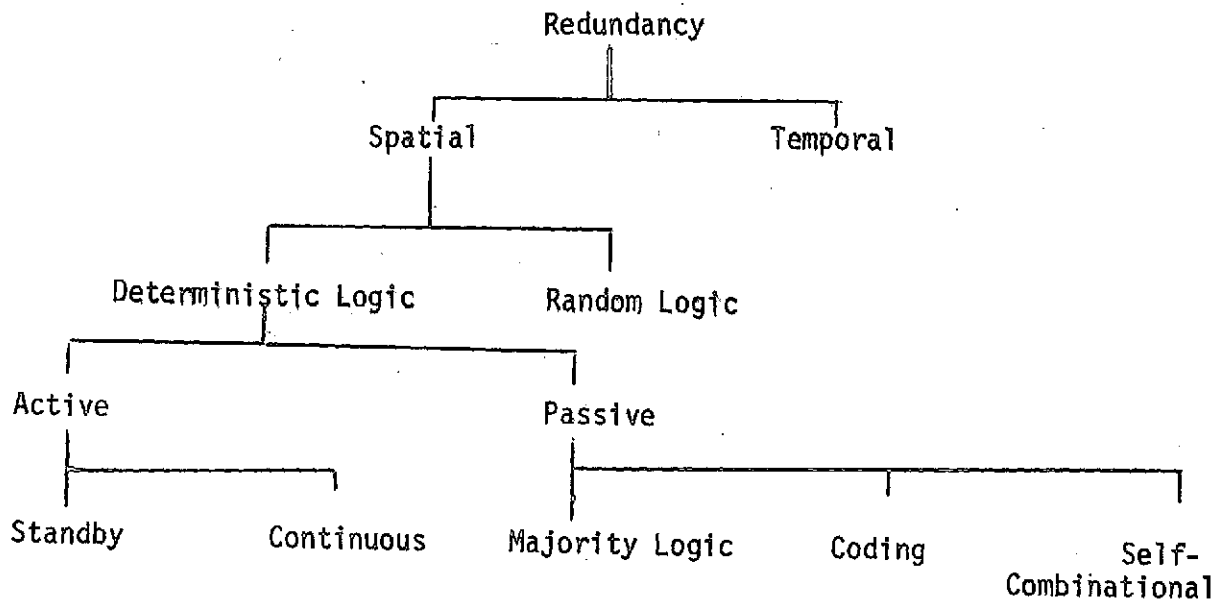


Figure 3-1  
Redundancy Type Classification

Some definitions for these redundancy types are:

- Spatial Redundancy - Use of physically distinct multiple channels for information transmission or processing. Works for transient or permanent faults.
- Temporal Redundancy - Repetition of the same information over the same channel at different times. Can only be used for transient faults.
- Deterministic Logic - Use of devices whose behavior is pre-determined and non time-variant.
- Random Logic - Use of devices whose behavior (or routing) is not deterministic.
- Active Redundancy - Use of some means of detection of faulty devices and subsequent rerouting of information. Also called dynamic or Block redundancy.
- Fault - A malfunction which in a non-redundant system would cause system failure.
- Failure - The unacceptable system performance caused by one or more uncorrected faults.
- Error - The difference between the actual and the correct output.
- Self-Combinational - An arrangement of devices which permits the logical output to be correct in spite of faults. (Series, shunt, quad, etc. arrangements)
- Majority Logic - Uses three or more channels with logic which delivers the majority output for a minority of channel faults. (Voting)
- Coding - The message includes a code which enables fault (error) detection and correction.

- o Standby - Unused elements are unpowered.
- o Continuous - Unused elements are powered.

Temporal redundancy implies the repetition of a part of the program to detect or correct or mask transient faults. Its primary use is communications systems where noise is a major problem. It will not help with permanent faults. Random logic is being developed, but is not yet applicable. It includes artificial neural networks and similar approaches.

In active redundancy, two or more functionally equivalent blocks are used, any of which may perform the intended function. These blocks may be switched into use by switching the signals in/out and/or power to the blocks. If a fault occurs (how this is detected is a separate subject), the faulty block is not used and a non-faulty block replaces it.

Passive approaches include use of majority logic, coding, or self-combinational logic. Majority logic uses an odd number of channels to process information. The outputs of all channels are compared and the resulting output is voted upon; that is, the output is determined by the majority result. (This is sometimes referred to as "modular" redundancy.)

Coding involves a message structure containing coded information allowing a correction of a small number of message faults. In one form, it becomes a type of temporal redundancy. In another, it can become a type of majority logic.

Self-combinational logic includes a passive arrangement of components which permit them, through series, parallel or quad configurations, to "cover" for faults in each other. The circuit or device is designed for a large tolerance of component characteristics, so large that component faults can be masked by the series or parallel component.

Note that passive redundancy approaches do not require a fault detection device (fault detector). Therein lies one of the principal advantages for this approach.

Redundancy always involves the addition of something, time or hardware. Only the redundancy necessary should be provided.

Standby redundancy is the most efficient redundancy in terms of power consumption since only the minimum equipment is on at any one time. It is also very efficient in terms of reliability obtained, particularly if the off failure rates are much less than the on failure rates.

An element is defined as a block on a reliability block diagram. There may be many different blocks, some identical and some different, in a system. Blocks that perform an identical function, in substitute for each other, in standby redundancy; are considered to be the same element.

Single standby redundancy involves one block in standby to another, which is in use. Dual standby redundancy involves two blocks in standby, etc. Sometimes these are referred to as 1 of 2 or 1/2, or 1 of 3 (1/3), etc. redundancy. We may also have cases of 2 of 6 (2 needed of 6 total), etc. In any case, the total array of blocks in standby to each other is one element.

A system may have different elements possessing different types of redundancy. The element need not bear any direct relationship to boards, assemblies or other physical packaging subdivisions. It usually bears some direct relationship to function, but a single function need not be a single element or vice versa.

The standby block of a redundant element is assumed to have a failure rate equal to or lower than the block in use. The ratio of these failure rates is given by an  $r$  factor ( $0 \leq r \leq 1$ ).

The blocks of the elements must be designed so that any failures within the block are isolated to that block and do not propagate (fault independence). Each block is interconnected to other blocks of the same element and other blocks of other elements. These inputs and outputs must be protected by cross-strapping or switching to prevent failure propagation. How this can be done is discussed in Section 3.1.3.

A control switch of some type must be used to turn the on-block off and the off-block on to control the element. This is usually done by controlling the power to each block. The switch is characterized by a failure rate,  $\lambda_s$ . The control switch designs are discussed in Section 3.1.2.

Methods must be included for fault detection. This detection may include some degree of fault diagnosis to isolate the fault to a block, circuit, or part. Fault detection is discussed in Sections 3.5.5 and 3.6.

If the fault is detected and diagnosed to a redundant element or block, then that block may be replaced by a standby block. This is termed reconfiguration. If such reconfiguration is controlled from overt action on the ground, it is manual. If it is controlled by the system itself, then it is called automatic or autonomous. The latter is a necessity for fault-tolerant systems. The details of reconfiguration are covered in Section 3.7.

Where fault-detection is difficult, or where the system "down time" associated with fault detection and switching is excessive (or required to be zero), or where there is no more ultimate decision making capability available, then majority passive redundancy may be needed.

Majority (voting) approaches have the inherent disadvantage of high power consumption since (as a minimum) three of each element (plus the voter circuit) must be on (compared to one element plus switch for the standby redundancy). Also no advantage can be gained by having a lower failure rate for the "off" elements.

For elements of the same size (and ignoring the switch or voting circuits), the standby redundancy will always give better reliability and lower power. Since the voter can be made very simple, however, the element size is made small and the switch and voter are included in the calculations, then the majority voter approach can give greater reliability.

The voting must occur between each set of identical elements. The outputs of all preceding elements must enter the voter for each succeeding element. These intermediate voters then become effectively a part of the elements and only the final voter has an appreciable effect on the reliability.

If this is not done, then all intermediate voters are effectively in series, reducing the reliability considerably. The final voter should be made redundant using some self-combinational approach.

It is apparent that a majority approach would provide problems of modularity also. It would be difficult to organize a processor for modularity in terms of reliability, memory size, etc. without an excessive wiring and connector bulk.

Majority approaches are not efficient of power, requiring three to four times the power of standby approaches. Their use should be minimized to the most "hard core" portion of the system. This hard core portion should also be minimized in extent as much as possible.

### 3.1.1 Element Size Determination

To divide up the system into redundancy elements to achieve a particular reliability requirement is the problem. To do this, some use of probability mathematics will be necessary. First, consider the case for one element made up of two identical blocks (single standby redundant). (See Figure 3-2).

Each block of the element has an on failure rate,  $\lambda$ . The  $r$  factor and  $\lambda_s$  have been previously defined. The probability of success,  $P$ , for the time,  $t$ , is:

$$P = e^{-\lambda t} + \frac{\lambda e^{-\lambda t}}{\lambda_s + r\lambda} \left( 1 - e^{-(\lambda_s + r\lambda)t} \right) \quad (1)$$

this equation is made up of terms reflecting the following:

- At any given time, both blocks may have had no failure (success), either block may have failed (success), or both blocks may have failed (failure).

- Failures may occur with equal probability at any time as determined by the failure rates  $\lambda$ ,  $r\lambda$  and  $\lambda_s$ .
- Ordinarily, block-1 is used until (and if) it fails. In the meantime block-2 is off at a failure rate  $r\lambda$ . If block-1 fails, there is some probability that block-2 will have already failed. The equation accounts for this joint failure probability.

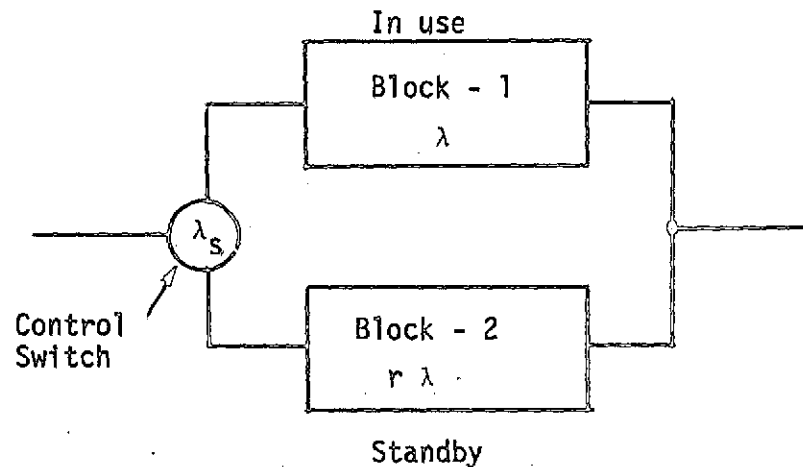


Figure 3-2  
Single Standby Redundant Element

- The switch may fail in either of two ways or it may not fail. The ways are:
  - The switch fails so that only block-1 can be used.
  - The switch fails so that only block-2 can be used.
- The switch failure possibilities are treated as equally likely. The equation accounts for the contribution of the switch failure rate to the failure rates of the blocks.
- The equation accounts for the integration of all possibilities up to the time,  $t$ .

Similar equations have been developed for other types of redundancy. Later, a general equation will be shown.

The cross-strapping circuitry can be shown to increase the value of the  $\lambda$  of the element (and not the switch) if the cross-strapping truly isolates failures and does not contribute single-point failures of its own.

We shall next examine the effect of the variables on the system reliability.

The reliability  $P$  of a system of  $N$  identical elements in single-standby redundancy, for the time  $t$ , is given by

$$P = \left[ e^{-\frac{\lambda_T t}{N}} + \frac{\lambda_T e^{-\frac{\lambda_T t}{N}}}{N\lambda_s + r\lambda_T} \left( 1 - e^{-\left(\lambda_s + \frac{r\lambda_T}{N}\right)t} \right) \right]^N \quad (2)$$

where  $\lambda_T$  = the total failure rate of the system  
 $\lambda_s$  = the "switch" failure rate (each switch)  
 $r$  = standby failure ratio

The units for  $\lambda_T$  and  $\lambda_s$  are failures /  $10^9$  hours. The units for  $t$  are hours.  $P$ ,  $N$  and  $r$  are dimensionless.

The failure rate for each element is  $\lambda = \lambda_T/N$ . A redundancy efficiency  $\eta$  can be defined as

$$\eta = \frac{\lambda}{\lambda + \lambda_s} \times 100\% \quad (3)$$

Equation (2) was programmed for Tymeshare and run for over 14,000 combinations of variables. These runs exist in the form of tables at TRW. The program (REX) is given in Basic form in Table 3-1.

The tables allow a quick determination of  $N$  for a given  $P$  and the other parameters. The program can also be used directly, modifying the appropriate statements to limit the values of  $T$ ,  $\lambda_s$  and  $r$  used. Since equation (2) is not easily solved for  $N$ , it is best to treat  $N$  as a variable, solving the equation for  $P$  for a variety of values of  $N$ .

Table 3-1

REX PROGRAM

```

10 ! RELIABILITY EXAMPLE (REX) PROGRAM
20 X = 9
30 Y = 5
40 Z = 3
50 DIM L(X), R(Y), T(Z), G(X), P(X)
100 FOR U = 1 TO 9
110 READ L(U)
120 DATA 2, 5, 7, 10, 20, 30, 40, 50, 70
130 NEXT U
200 FOR V = 1 TO 5
210 READ R(V)
220 DATA 0, 0.1, 0.2, 0.5, 1
230 NEXT V
250 FOR W = 1 TO 3
260 READ T(W)
270 DATA 5, 7, 10
280 NEXT W
290 PRINT
291 PRINT
292 PRINT "SUCCESS PROBABILITY FOR A SYSTEM WITH A TOTAL"
293 PRINT "FAILURE RATE LAMBDA, DIVIDED INTO N EQUAL ELEMENTS"
294 PRINT
300 FOR H = 1 TO 3 ! START H LOOP (T VARIATION)
310 FOR LS = 0.1 TO 1 BY 0.1 ! START LS LOOP (LAMBDA-S VARIATION)
320 FOR K = 1 TO 5 ! START K LOOP (R VARIATION)
360 Z$ = "%.% %-YEARS, LAMBDA-S = '%.%' KBIT, R = '%.%', LAMBDA (KBIT)
+ = '%/%'"
370 PRINT IN FORM Z$: T(H), LS, R(K)
380 PRINT IN FORM '3B""9(5B ZZ)%%': L(1), L(2), L(3), L(4), L(5), L(6),
+ L(7), L(8), L(9) ! COLUMN HEADINGS
400 FOR N = 1 TO 31 BY 2 ! START N LOOP (BLOCK)
410 FOR I = 1 TO 9 ! START I LOOP (CALCULATE ONE LINE)
420 TS = T(H)*8.76E-3
430 G(I) = LS + R(K)*L(I)/N
440 P(I) = EXP(-L(I)*TS)*(1 + L(I)*(1 - EXP(-G(I)*TS)))/(N*G(I))*N
450 NEXT I ! END I LOOP
460 FS = "2B ZZ 3B"
470 PRINT IN FORM FS: N
500 FOR J = 1 TO 9 ! START J LOOP (PRINT OUT ONE LINE)
510 F = "Z.ZZZ 2B"
520 PRINT IN FORM F: P(J)
530 NEXT J ! END J LOOP (LINE)
535 PRINT
540 NEXT N ! END N LOOP (BLOCK)
550 PRINT
560 PRINT
570 PRINT
580 NEXT K ! END K LOOP
590 NEXT LS ! END LS LOOP
595 NEXT H ! END H LOOP
600 PRINT
610 PRINT
620 PRINT "A. A. SØRENSEN      ": DATE:      TIME
630 PRINT
640 PRINT
900 END

```

In the real world, the  $\lambda$  of all elements in the system can not be the same. Because of that, the practical use of Equation (2) is limited. It is useful, however to generally determine the number of elements needed, their size ( $\lambda$ ), and the influence of changes in  $\lambda_s$  and  $r$ .

It is also possible to apply more sophisticated redundancy (such as dual-standby redundancy, 2 of 4 etc.) to achieve better reliability for smaller values of  $N$  (and possibly greater "efficiency"). This should probably not be done in general throughout the system for all elements, however. More will be said about this later.

Also to be noted at this time is the fact that the overall reliability is most efficiently achieved if all elements have approximately the same reliability. "Strong" links (higher than average element reliability) are of not nearly so much help as "weak" links (lower than average element reliability) are of harm. This is a sort of "chain is no stronger than ..." theory.

Figures 3-3 and 3-4 were plotted from data generated by equation (2). These figures show  $P$  vs.  $N$  for values of  $\lambda_T$  of 5, 10, 20 and 30 thousand bits (failures /  $10^9$  hours). Both figures are for 5 years and with  $r$  varied over its full range. Figure 3-3 is for  $\lambda_s = 100$  bits and Figure 3-4 for  $\lambda_s = 500$  bits.

Note that, as expected, increases in  $r$  or  $\lambda_s$  reduce the reliability and result in a larger value of  $N$  for a given  $P$  (or smaller value of  $P$  for a given  $N$ ).

Some examples, taken from these figures, are of interest to show the effects of  $r$  and  $\lambda_s$  on the redundancy efficiency,  $\eta$ .

Figure 3-3 P vs. N for  $\lambda_T$  and r

$T = 5$  YEARS  
 $\lambda_s = 100$  bit

ORIGINAL PAGE IS  
 OF POOR QUALITY

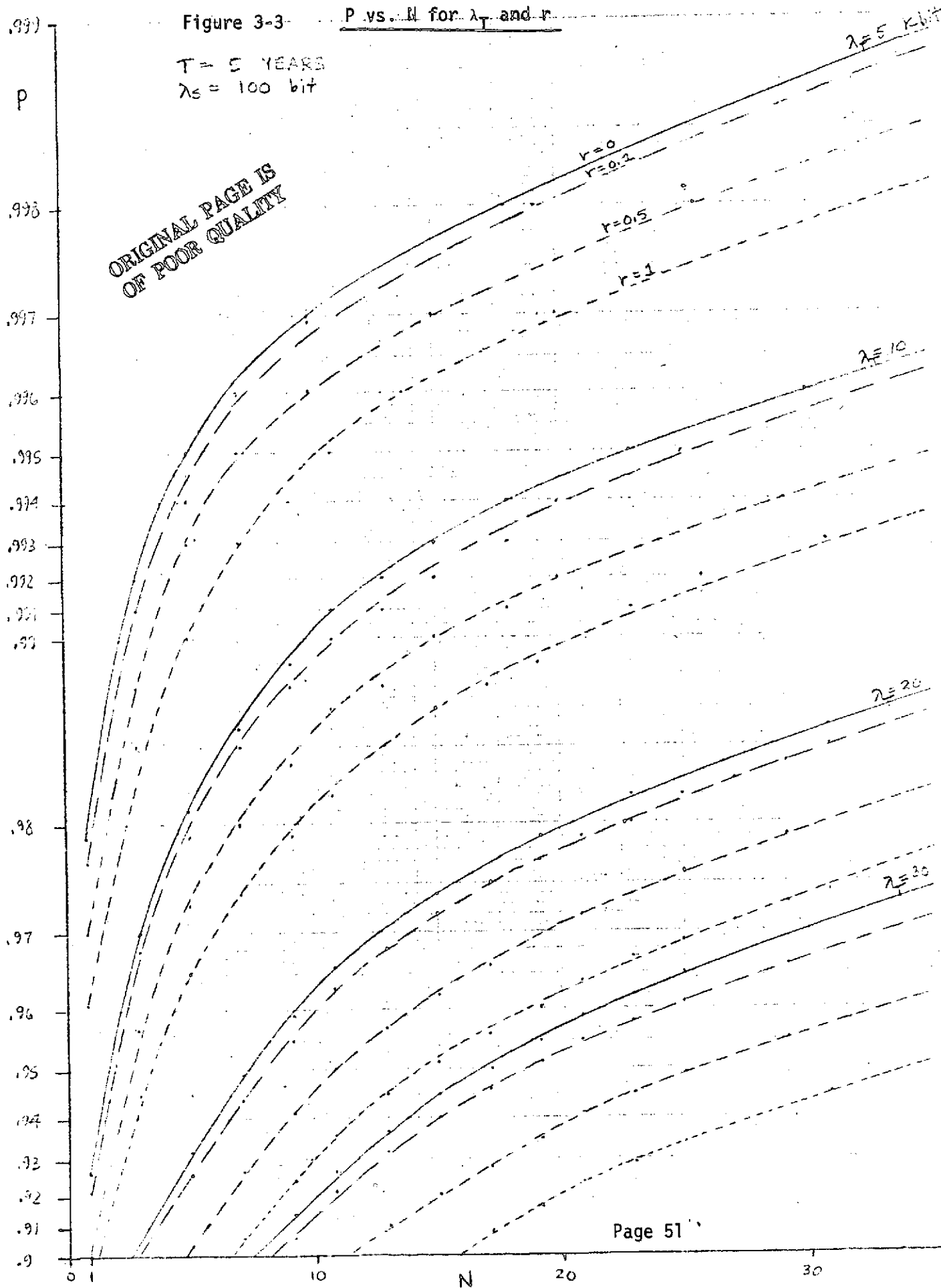
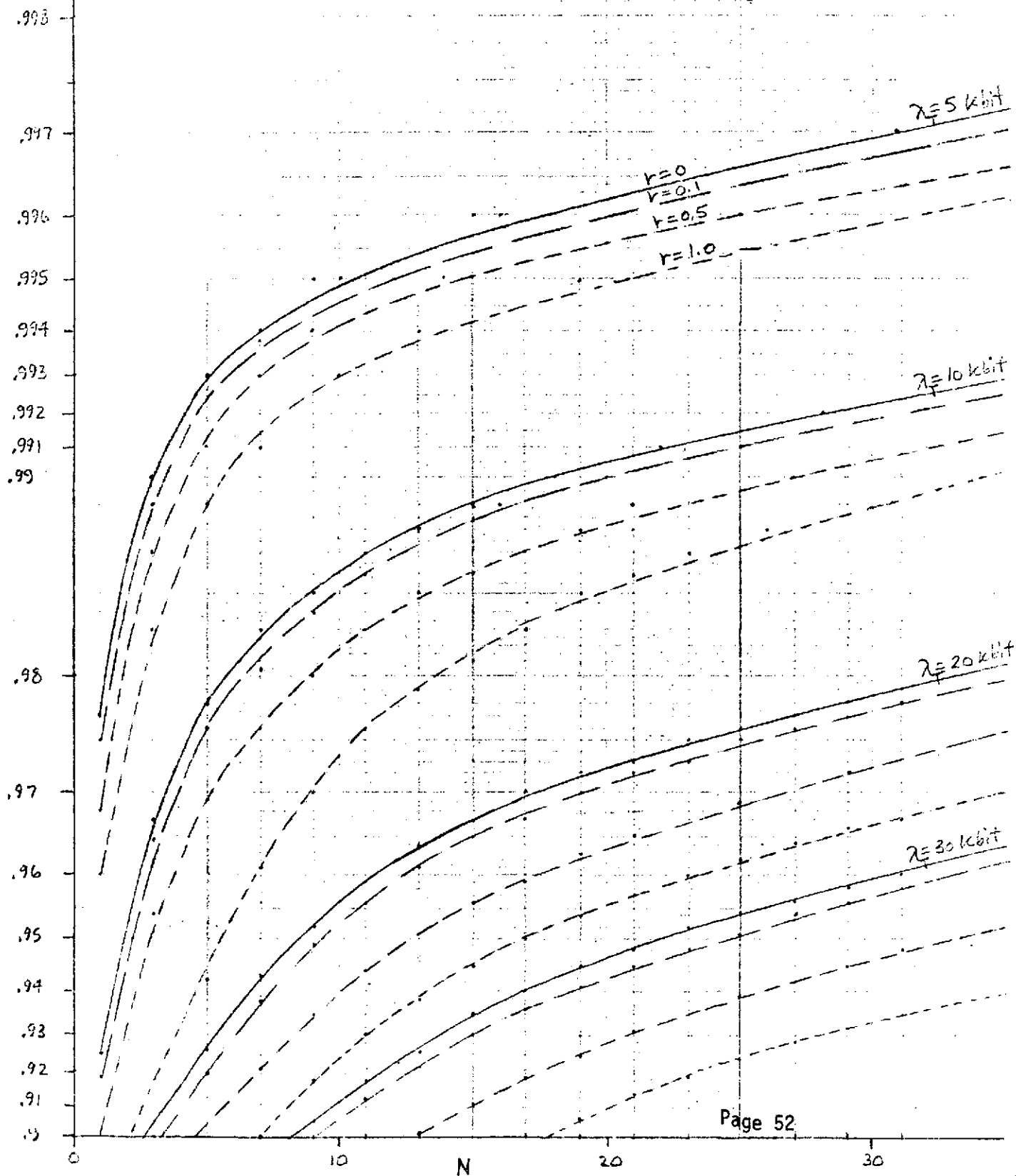


Figure 3-4  
 $T = 5$  YEARS  
 $\lambda_s = 500$  bit

P vs. N for  $\lambda_T$  and r

ORIGINAL PAGE IS  
 OF POOR QUALITY



Example -1:       $t = 5$  years  
 $\lambda_s = 100$  bits  
 $\lambda_T = 10$  K bits  
 $P \geq 0.994$

For  $r = 0.1$ ,       $N = 21$ ,     $\lambda = 476$  bits,  $\lambda_{st} = 2100$ ,  $\eta = 83\%$   
 $r = 0.5$ ,       $N = 29$ ,     $\lambda = 345$  bits,  $\lambda_{st} = 2900$ ,  $\eta = 78\%$

Example -2:       $t = 5$  years  
 $\lambda_s = 100$  bits  
 $\lambda_T = 30$  K bits  
 $P \geq 0.95$

For  $r = 0.1$ ,       $N = 17$ ,     $\lambda = 1765$  bits,  $\lambda_{st} = 1700$ ,  $\eta = 95\%$   
 $r = 0.5$ ,       $N = 26$ ,     $\lambda = 1154$  bits,  $\lambda_{st} = 2600$ ,  $\eta = 92\%$

Example - 3:       $t = 5$  years  
 $\lambda_s = 500$  bits  
 $\lambda_T = 10$  K bits  
 $P \geq 0.99$

For  $r = 0.1$ ,       $N = 20$ ,     $\lambda = 500$  bits,  $\lambda_{st} = 10,000$ ,  $\eta = 50\%$   
 $r = 0.5$ ,       $N = 27$ ,     $\lambda = 370$  bits,  $\lambda_{st} = 13,350$ ,  $\eta = 43\%$

Example -4:       $t = 5$  years  
 $\lambda_s = 500$  bits  
 $\lambda_T = 30$  K bits  
 $P \geq 0.95$

For  $r = 0.1$ ,       $N = 25$ ,     $\lambda = 1200$  bits,  $\lambda_{st} = 12500$ ,  $\eta = 71\%$   
 $r = 0.5$ ,       $N = 33$ ,     $\lambda = 909$  bits,  $\lambda_{st} = 16500$ ,  $\eta = 65\%$

See from these examples how the increase in  $r$  or  $\lambda_s$  drastically increases  $N$  (and reduces  $\lambda$  and  $\eta$  ).

The redundancy efficiency was defined as the ratio of the system  $\lambda$  divided by the system  $\lambda_s$  ( $\times 100$ ) in %.

This is the same as the element  $\lambda$  divided by the element  $\lambda_s$ .

$$\text{i.e. } \eta = \frac{\lambda_T}{\lambda_{sT}} \times 100 = \frac{\lambda}{\lambda_s} \times 100 \quad (4)$$

It is a measure of the penalty for making the system redundant, over and above the doubling of circuitry.  $\eta$  should be 80% or higher.

$\eta$  is a function of  $\lambda_s$  and  $r$ ; as well as  $\lambda_T$  and  $P$  &  $t$ . It is a very strong function of  $\lambda_s$  because as  $\lambda_s$  increases, not only does the denominator of the function increase, but the numerator decreases. [As  $\lambda_s$  increases the curves of  $P$  vs  $N$  for constant  $r$  and  $\lambda_T$  flatten, requiring a larger  $N$  for a given  $P$ . Since  $\lambda = \lambda_T/N$ ; this reduces  $\lambda$ .]

As  $r$  increases, the general shape of the curves stays the same, but  $P$  decreases (for all other variables constant). If  $P$  is to remain constant,  $N$  increases, reducing  $\lambda$  and  $\eta$ .

$\eta$  is a measure of the part count increase, weight increase, power increase, and cost increase in the system due to the redundancy cross-strapping and switching.

Another example can illustrate some limits on the value of  $\lambda_s$ .

#### Example 5

$$T = 5 \text{ years}$$

$$P \geq 0.95$$

$$\lambda_T = 30 \text{ K bit}$$

$$\lambda_s = 100 \text{ bit}, \quad r = 0.1, \quad N = 19, \quad \lambda = 1579, \quad \eta = 94.0\%$$

$$r = 0.5, \quad N = 26, \quad \lambda = 1154, \quad \eta = 92.0\%$$

$$r = 1.0, \quad N = 35, \quad \lambda = 857, \quad \eta = 89.6\%$$

$\lambda_S = 300 \text{ bit}$	$r = 0.1$	$N = 21$	$\lambda = 1429$	$\eta = 82.6\%$
	$r = 0.5$	$N = 29$	$\lambda = 1034$	$\eta = 77.5\%$
	$r = 1.0$	$N = 40$	$\lambda = 750$	$\eta = 71.4\%$

$\lambda_S = 500 \text{ bit}$	$r = 0.1$	$N = 25$	$\lambda = 1200$	$\eta = 70.6\%$
	$r = 0.5$	$N = 33$	$\lambda = 909$	$\eta = 64.5\%$
	$r = 1.0$	$N = 44$	$\lambda = 682$	$\eta = 57.7\%$

$\lambda_S = 700 \text{ bit}$	$r = 0.1$	$N = 29$	$\lambda = 1034$	$\eta = 59.6\%$
	$r = 0.5$	$N = 39$	$\lambda = 769$	$\eta = 52.3\%$
	$r = 1.0$	$N = 51$	$\lambda = 588$	$\eta = 45.7\%$

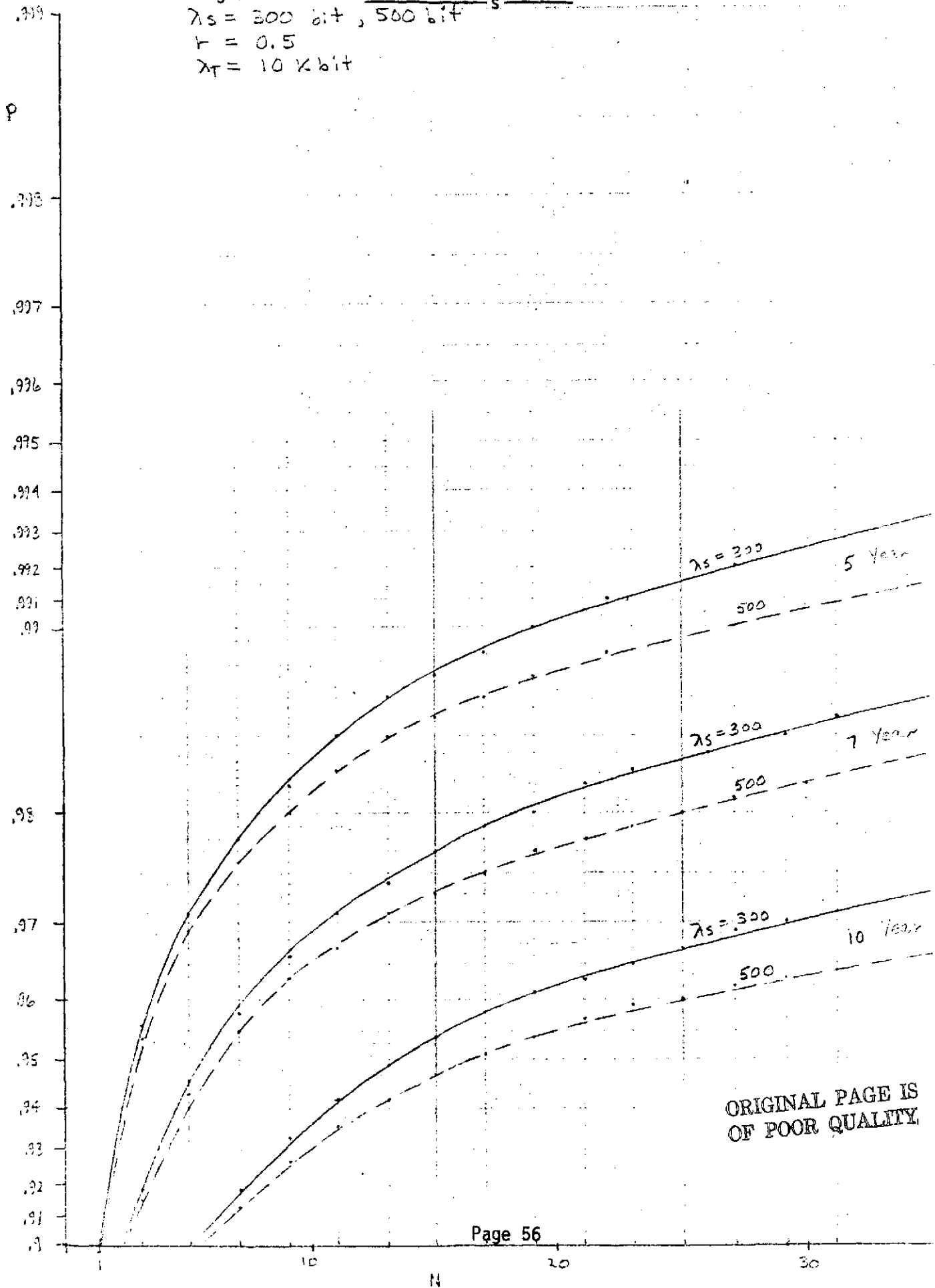
Here, the variation in  $r$  gives a 4.4 to 13.9% variation in  $\eta$ . The variation in  $\lambda_S$  (for  $r = 1.0$ ) gives a 43.9% variation in  $\eta$ . The systems with  $\lambda_S > 300$  bits are not sufficiently efficient. If the desired  $P$  were higher (or  $\lambda_T$  higher), then the equivalent point would occur for even lower  $\lambda_S$ .

All of this ignores the fact that  $\lambda_S$  may not be a constant, but is a function of the elements to be cross-strapped. Sometimes  $\lambda_S$  decreases somewhat with element size ( $\lambda$ ), but it has an irreducible minimum of the switch lambda. Sometimes  $\lambda_S$  can increase, as  $\lambda$  decreases because more interfaces must be cross-strapped as the subdivision of "blocks" between elements occur.

For a given  $T$ ,  $\lambda_T$ ,  $\lambda_S$ , and  $r$ ; a specified  $P$  may not be obtainable for a limiting minimum of  $n$ . In fact, it may not be obtainable for any  $n$ . (i.e.  $N \rightarrow \infty$ ).

The effect of time on  $P$  vs  $N$  is shown in Figure 3-5. Here, a  $\lambda_T$  of 10 K bit and  $r = 0.5$  was used. Curves are plotted for  $\lambda_S = 300$  and 500 bit and  $t = 5, 7$  and 10 years. An example is taken from this figure:

Figure 3-5 P vs N for  $\lambda_s$  and t.  
 $\lambda_s = 300 \text{ bit}, 500 \text{ bit}$   
 $t = 0.5$   
 $\lambda_T = 10 \text{ Kbit}$



Example 6

$$\lambda_T = 10 \text{ k bit}$$

$$r = 0.5$$

$$P \geq 0.97$$

As the time increases, note how N must increase: (For  $\lambda_S = 300$ )

$$T = 5 \text{ years} \quad N = 5$$

$$T = 7 \text{ years} \quad N = 11$$

$$T = 10 \text{ years} \quad N = 29$$

Again, note the influence of  $\lambda_S$ . For  $\lambda_S = 500$ :

$$T = 5 \text{ years} \quad N = 6$$

$$T = 7 \text{ years} \quad N = 13$$

$$T = 10 \text{ years} \quad N = 42$$

The influence of the change in  $\lambda_S$  is much greater for the longer time. The same principle can also be seen to apply on the effect of the  $r$  factor.

An important question is the proper value for  $r$ , the standby failure factor. Cases can be made for values of  $r$  over its full possible range (0 to 1). At the one extreme, it can be surmised that of course the failure rate is lower when the equipment is not in use. At the other extreme, it can be explained that the derating is so extreme (the components are operating at very low stress factors) that the on and off failure rates should be identical.

Experimental data can be found to support the full range of  $r$  factors. It is probable that, for mechanical devices or electronic components operating at 50% or greater stress that low values ( $r \cong 0.1$ ) are justified. For semi-conductor devices (non-power handling) such as digital integrated circuits, if they are properly designed and applied, then the standby failure rate should be essentially the same as the "on" failure rate ( $r \cong 1$ ).

Use of a constant  $r$  factor for all components in circuit, element or subsystem is incorrect, but must be done to simplify the reliability calculations. Because of this, an average  $r$  factor must be postulated for each element, taking into account the devices contained therein.

We have seen that the  $r$  factor has a fairly major effect on the system reliability or the system complexity to achieve the required reliability. Certainly more research and collection of data is necessary to establish the proper choice of  $r$  factors.

The use of constant failure rates in terms of numbers of failures per unit time is a convenient fiction of reliability work. This assumes a so-called "bathtub-shaped" reliability curve. That is, that the device will exhibit a high failure rate initially (infant mortality) and a high failure rate after a long time (wearout), with a lower failure rate in between over the majority of the life. This lower failure rate is assumed constant and is the familiar  $\lambda$  used in the reliability calculations.

The statistics are also a convenience to permit a mathematical treatment of reliability. Any one part does not have a constant failure rate - it can only fail once. Only when there is a large number of the same part, at the same stresses, applied in the same way, can one hypothesize a failure rate for that type of part. For any other type of part, or different stress or application, that failure rate is different.

It is further assumed that failures are random. That is, they can not be predicted in any way other than on the average. The constant failure rate implies that there will be an equal chance of failure in any equal increments in time, anywhere in the eon.

Now none of the preceding is probably true. Failures are probably all deterministic, at least in the solid-state electronics that we now have. There are no known deterioration or wear-out mechanisms. There is no aging and if the parts are properly designed and built, operated at low enough stresses and not abused, then they should last forever (i.e.,  $\lambda=0$ ).

That we do get failures is a result of the imperfection of the design, fabrication or application. The inability to predict exactly where these failures will occur results in the need for a statistical approach to failure prediction.

What has been done is to assemble large numbers of the same part, manufactured in the same way at about the same time. The same stresses are applied and the parts are put on life tests, with all failures and characteristic degradations recorded. After a long time (a year or more) some statistical determination of failure rate can be established, with the confidence factor increasing as the number of part-years is increased.

A similar process can be used to gather data from devices in actual use. Such data has been gathered from spacecraft flights. It is more questionable, however, since the conditions are less controlled and since it is often difficult (or impossible) to isolate failures to a specific part. Even with these limitations, orbital failure rates have been "measured" for many parts, generally supporting the rates established in life tests.

The failure rates used have been gradually getting smaller over the years. This is probably due to equal measures of more testing, better parts, improved processes, etc. This trend is expected to continue.

We must recognize what a failure rate of one failure per  $10^9$  hours implies (such as for a resistor). This says that if I have  $10^5$  such parts, only one (on the average) part can be expected to fail in 10,000 hours (over a year). This is truly remarkable.

For some complex devices, such as gyros, reaction wheels, etc. the establishment of a reliability figure is difficult. Due to economic limitations, not enough of the devices can be built and tested to establish a reasonable failure rate with any confidence level much above zero. Usually such devices are analyzed as being made up of their lower level parts (motors, bearings, etc.), where tests (or guesses) have indicated some appropriate failure rates. Here the statistics are even more suspect and wearout effects can probably not be ignored.

We saw earlier what a large effect the  $\lambda_s$  has on the reliability efficiency. It is very important to keep  $\lambda_s$  low. Assuming the use of standby redundancy,  $\lambda_s$  is made up of the failure rate of the element control switch and the cross-strapping circuitry (if active).

The control switch turns the power to the element on or off depending on the signal(s) on control line(s). The cross-strapping may be passive or active, depending on the type of signals or power to be cross-strapped. Since the failure rate of the cross-strapping is proportional to the number of signals cross-strapped, careful attention must be given to reducing the number of these signals when dividing the system into redundant

The design of the control switch and cross strap circuits is discussed in Sections 3.1.2 and 3.1.3. These designs have as their goal a minimization of failure rate, within the constraints of failure isolation, minimization of signal degradation, etc. If this is done, and if the elements are divided for minimum interface, then the  $\lambda_s$  will be minimized also.

While the  $r$  factor and  $\lambda_s$  are very important to the system reliability, the major influence is still the total failure rate of the non-redundant system,  $\lambda_T$ . This is made up of the addition of the failure rates of all of the system components.

We have seen that there are limitations to the system reliability that can be obtained with fixed values of  $\lambda_T$ ,  $t$ ,  $\lambda_s$  and  $r$ . However, as  $\lambda_T$  decreases, these limitations rapidly move out and the reliability efficiency ( $\eta$ ) quickly improves.

It seems almost too obvious to state that the most reliable system is the most simple one (all other things being constant), but it is true. Any function omitted can not fail. The system designers must be made aware of this so that the need for all the niceties and conveniences can be properly assessed. (Compare the reliability of a Pinto and a Continental, for example.)

Automatic features should be discouraged unless really necessary. Capabilities embodied into the ground station instead of into the spacecraft will be more reliable due to their ability to be repaired.

A second way to reduce the  $\lambda_T$  is to design the circuits required so that they require a minimum of parts, each of which is chosen for minimum  $\lambda$ . Such cleverness of design is herein assumed. Generally this implies a high degree of integration (more use of integrated circuits). It has been established that a small-scale integrated circuit (SSI) (such as flip-flop) has a lower total  $\lambda$  than the equivalent circuit designed with discrete components. Similarly a medium-scale integrated circuit (MSI) (such as a shift register) has a lower total than the SSI-circuit equivalent. The same is true of LSI (large-scale integrated circuits) relative to MSI.

It can be seen then that LSI should receive priority over MSI, over SSI, over discretes; etc., for purposes of reliability. That similar advantages accrue to volume and power (and often even cost) explains the increasing use of higher-scale integration.

The failure rate of an IC seems to be made up of a constant portion (for the package), a portion proportional to the number of leads (the lead attachment probably being the least reliable feature), and a small portion proportional to the complexity (number of equivalent gates) of the device.


Current lambdas for integrated circuits are in the range of:

SSI	10-20	bits (failures/ $10^9$ hours)
MSI	15-50	"
LSI	30-300	"

A general equation has been developed at TRW for the calculation of the reliability of standby redundancy arrangements of any complexity.

The equation is:

$$p = e^{-x\lambda t} \sum_{j=1}^{y-x+1} \left\{ \frac{(-1)^{j-1} e^{-R(j-1)\lambda t}}{(j-1)!} \sum_{q=0}^{y-x+1-j} \left[ \frac{1}{q!} \prod_{m=0}^{j+q-2} \left( m + \frac{x}{R} \right) \right] \right\} \quad (5)$$



if  $j + q = 1$ , this term = 1

where:  $p$  = the overall reliability  
 $x$  = the number of elements active and required  
 $y$  = the total number of elements (to start)  
 $\lambda$  = the "on" failure rate of an element (bits)  
 $t$  = time in hours  
 $j, q, m$  = operators

$$R = \frac{\lambda s}{\lambda} + r$$

$\lambda s$  = the failure rate of each switch used to switch between on and off elements (bits)  
 $r$  = the ratio of "off" failure rate to "on" failure rate ( $r \leq 1$ ).

This equation calculates the reliability of  $x$ (needed) of  $y$ (available) elements, with the standby elements ( $y + x$ ) having a failure rate lower than the active elements (as determined by  $r$ ).  $x$  and  $y$  can be any numbers, but  $x \leq y$ .

Equation (5) has been programmed in Basic language for Tymeshare. The program (URP2) given in Table 3-2 , provides for the simultaneous computation and printout for an entire system, having elements of different names, times,  $\lambda$ ,  $\lambda_s$  and  $r$ . A sample input of data and the resulting output is shown in Table 3-3 . Note that both the individual reliability and the cumulative reliability are printed.

This program has been the most useful tool for reliability calculations and system design used.

A systematic procedure for determining the number of elements needed, and their size, is given below:

- o Determine (or estimate) the total, non-redundant, system lambda ( $\lambda_T$ ).
- o Estimate the probable average value for  $\lambda_s$ . This is best done from previous experience, but the value should be from 100 to 300 (failures/ $10^9$  hours). (See Section 3.1.2 ).
- o Decide on the  $r$  factor.
- o Presumably, the value of  $t$  is given and a desired value of  $P$  is known.
- o Use equation (1) iteratively to find the value of  $N$  which gives a  $P$  just larger than required.
- o The average element lambda is  $\lambda = \lambda_T/N$
- o Determine the efficiency  $\eta = \frac{\lambda}{\lambda + \lambda_s} 100$
- o If  $N$  is too large,  $\lambda$  too small, or  $\eta$  too small; the requirements may be too severe for the contemplated system lambda. If possible, reexamine the requirements, the system design, and the estimates for  $\lambda_T$ ,  $\lambda_s$  and the  $r$  factor selected. If the numbers are reasonable, proceed.

Table 3-2

URP2 Universal Reliability Program -2

```

10 ! UNIVERSAL RELIABILITY PROGRAM - TWO (URP2)
20 READ M, L ! NO. OF ELEMENTS, INITIAL RELIABILITY
30 READ Q, V ! PRINT HEADING, NO. OF CODE SETS
40 ! IF PRINT HEADING = 0, THEN IT IS NOT PRINTED
45 ! V = NO. OF SETS OF T, S, R
50 PRINT
60 PRINT
70 IF Q = 0 THEN 180
80 PRINT TAB (26): "XYZ SPACECRAFT ACS" ! 18 SPACES
90 ! TAB NO. = 35 - (TITLE LENGTH)/2
100 PRINT TAB (23): "RELIABILITY CALCULATIONS" ! 24 SPACES
110 PRINT
120 Z$ = "7B %%.% % -YEARS, LAMBDA-S = 'Z,ZZZ' BIT, R = '%.%%', SET 'Z
+ /"
122 DIM T(V), S(V), SS(V), R(V)
125 FOR G = 1 TO V ! START G LOOP
127 READ T(G), S(G), R(G) ! YEARS, LAMBDA-S (KBIT), R FACTOR
130 SS(G) = S(G)*1000
140 PRINT IN FORM Z$: T(G), SS(G), R(G), G
145 NEXT G ! END G LOOP
150 PRINT
160 PRINT "ELEMENT-NAME":TAB(15):"LAMBDA (BITS)":TAB(30):"RED.TYPE":TAB(
+ 43):"REL.":TAB(51):"CUM.REL.":TAB(63):"SET NO."
170 PRINT
180 IF L = 1 THEN 220
190 PRINT "INITIAL RELIABILITY"
200 L$ = "40B %.5% 3B %.5% /"
210 PRINT IN FORM L$: L, L
220 DIM A(M), X(M), Y(M), P(M), B(M), AS(M), U(M)
240 STRING H(M)
250 FOR I = 1 TO M ! START I LOOP
260 READ H(I), A(I), X(I), Y(I), U(I)
270 ! ELEMENT NAME, LAMBDA (KBIT), ELEMENTS NEEDED, STARTING ELEMENTS, U
+ SE SET NO.
273 G = U(I)
276 T = T(G)*8760
280 RS = S(G)/A(I) + R(G)
290 A(I) = A(I)*1E-6
300 N = Y(I) - X(I) + 1
310 FS = 0
320 FOR J = 1 TO N ! START J LOOP
330 E = 0
340 FOR Q = 0 TO (N - J) ! START Q LOOP
350 D = 1
360 IF J + Q = 1 THEN 400
370 FOR K = 0 TO (J + Q - 2) ! START K LOOP
380 D = D*(X(I)/RS + K)
390 NEXT K ! END K LOOP
400 QS = 1

```

URP2 (continued)

```

410 IF Q = 0 THEN 460
420 W = Q
430 Q$ = Q$*W
440 W = W - 1
450 IF W > 1 THEN 430
460 E = E + D/Q$
470 NEXT Q ! END Q LOOP
480 G$ = 1
490 IF J = 1 THEN 540
500 W = J - 1
510 G$ = G$*W
520 W = W - 1
530 IF W > 1 THEN 510
540 F = (-1)^(J-1)*EXP(-R$*(J-1)*A(I)*T)*E/G$
550 F$ = F$ + F
560 NEXT J ! END J LOOP
570 P(I) = F$*EXP(-X(I)*A(I)*T)
580 IF I > 1 THEN 610
590 B(I) = L*P(I)
600 IF I = 1 THEN 630
610 I$ = I - 1
620 B(I) = B(I$)*P(I)
630 NEXT I ! END I LOOP
640 FOR C = 1 TO M ! START C LOOP (PRINT OUT)
650 PRINT H(C)
660 Z = "17B ZZ,ZZZ 7B %'/'%% 5B %.5% 3B %.5% 7B Z /"
670 A$(C) = A(C)*1E+9
680 PRINT IN FORM Z: A$(C), X(C), Y(C), P(C), B(C), U(C)
690 NEXT C ! END C LOOP
700 PRINT
710 PRINT
720 PRINT "A. A. SØRENSEN      ":DATE:      TIME
730 PRINT
740 PRINT
750 DATA 2, 1
751 ! NO. OF ELEMENTS, INITIAL RELIABILITY
760 DATA 1, 1
761 ! PRINT HEADING IF NOT 0, NO. OF CODE SETS
770 DATA 5, 0.5, 0.1
771 ! YEARS, LAMBDA-S (KBIT), R FACTOR (FOR SET 1)
772 ! SET 2, ETC.; FOLLOW IN 771 TO 779
780 DATA ELEMENT-A, 5, 1, 2, 1, ELEMENT-B, 10, 2, 4, 1
781 ! ELEMENT NAME, LAMBDA (KBIT), ELEMENTS NEEDED, STARTING ELEMENTS, S
+ ET NO.:      FOLLOW IN 780-899
785 DATA CHANNEL, 317, 1, 2
900 END

```

Table 3-3      URP2 Sample Run

```

750 DATA 4, 0.978
$ 760 DATA 1, 3
$ 770 DATA 0.01, 0.5, 0.1
$ 771 DATA 0.5, 1, 0.5
$ 772 DATA 5, 0.5, 0.2
$ 780 DATA ELEMENT-A, 0.2, 1, 1, 1
$ 781 DATA ELEMENT-B, 20, 1, 2, 2
$ 782 DATA ELEMENT-C, 5, 1, 2, 3
$ 783 DATA ELEMENT-D, 10, 2, 4, 3
$

```

RUN

XYZ SPACECRAFT ACS  
RELIABILITY CALCULATIONS

```

.01-YEARS, LAMBDA-S = 500 BIT, R = .10, SET 1
.50-YEARS, LAMBDA-S = 1.000 BIT, R = .50, SET 2
5.00-YEARS, LAMBDA-S = 500 BIT, R = .20, SET 3

```

ELEMENT-NAME	LAMBDA (BITS)	RED. TYPE	REL.	CUM. REL.	SET NO.
INITIAL RELIABILITY					
			.97800	.97800	
ELEMENT-A	200	1/ 1	.99999	.97792	1
ELEMENT-B	20.000	1/ 2	.99449	.97358	2
ELEMENT-C	5.000	1/ 2	.97359	.94690	3
ELEMENT-D	10.000	2/ 4	.92326	.87424	3

A. A. SØRENSEN      02/08/74. 10.32.17.

ORIGINAL PAGE IS  
OF POOR QUALITY

- Divide up the system into approximately  $N$  elements, attempting to keep them all near to the size  $\lambda$ . Work this out in detail, calculating the actual  $\lambda$  for each element. Make the divisions to try to fit the following criteria:
  - Minimize interface lines
  - If possible, make interfaces digital rather than analog
  - Divide along functional lines
  - Redesign circuits or blocks, if necessary
  - Reduce the number of different elements each element interfaces with
  - Use serial (instead of parallel) data interfaces
  - Maximum use of simplest cross-strapping
- Design the control switch and cross-strapping for each element. Preferably, these were previously designed and used, but some new designs might be required to suit circumstances.)
- Calculate the actual  $\lambda_s$  for each element. Pay particular care to associating the cross-strapping to the proper element. (This is not as easy as it would seem).
- Decide on the  $r$  factor for each type of element.
- Take into account the different duty factor of each element. Some elements may be in use for only a short time in the mission. The time,  $T$ , for each element can be adjusted to account for this. The environmental stress factor can also be accounted for here too.
- It may be that single-standby redundancy will not be adequate for some elements to get their reliability to the norm. That is, for best design or a variety of other reasons, the  $\lambda$  may be higher than the desired average determined earlier. If so, then dual-standby or other forms of redundancy may be applied.
- The reliability program (see Equation (5) ) should be used to calculate the individual reliabilities and the system reliability.

- The individual reliabilities should be examined for a general equality. If the numbers are too high, perhaps elements could be combined. More normally, there will be some weak links and the overall reliability won't quite be achieved.
- Redesign or apply higher levels of redundancy to the weak links, starting with the weakest links that can be strengthened with the least penalty.
- Continue to optimize, trying to achieve the reliability goal for the most efficient system. Use of the reliability program should be modified in the data statements as the design iterates, so that it always reflects the most real and latest configuration.

### 3.1.2 Element Control Approach

In standby redundancy the control of which blocks are in use and which are in standby is accomplished by turning the power on or off to the blocks. Usually the blocks are passively cross-strapped so that their selection is by such power control. Sometimes (for high power interfaces) the outputs/inputs must also be switched for the control. This might be due to difficulty of passive cross-strapping at such an interface.

In the previous section we saw that as the element size decreases the failure rate of the control switch becomes more important. Switch failure rate reduction can be achieved by:

- Reducing the number of voltages needed in the block.
- Simplification (or higher degree of integration) of the switch.
- Incorporation of a degree of passive redundancy in the switch.
- Proper interfacing of the switch with the controlling source.

Other important control switch design criteria are:

- Need for independent and mutually exclusive switching of power to each block.
- Low internal voltage drop and power loss. Low standby (non-switching) power consumption. (The switch power consumption can become a large part of the system power consumption.)
- No single point failures.

The filtering requirements of the power (for EMI) may also be involved in the control switch design. The distribution of the power to the blocks may also be influenced.

Classically, two somewhat different power control and distribution techniques have been employed:

- Centralized - a single unit or subassembly is configured to handle all power line filtering, power control command decoding, power switching and distribution to each block in the system.
- Decentralized - each block handles its own filtering and switching upon receipt of a logic command.

A key factor in the design is the number of blocks involved. For a complex spacecraft control system of long life and high reliability, it seems likely that the number of blocks required will range from 20 to 50, with 8 to 20 of these being in the processor and the rest associated with peripherals.

Any switching or processing circuitry functions that are decentralized to each individual block are obviously going to be multiplied by the number of blocks within the system. It is also obvious then that the most efficient system of power distribution and switching will be one that shares as much common circuitry as possible between the various blocks and minimizes the parts dedicated to each block.

For example, consider the power control switch developed for decentralized switching of dc secondary power in each block of the COPE processor. This circuit is shown in Figure 3-6. Assume that the system has 40 blocks, all using this circuit. Assuming half the blocks require + and - 15 volts as well as + 5 volts, the total parts required for the switching function (not including filtering) is:

Per circuit:

1.5 relays	} x 40 =	60 relays
2.0 coil drivers		80 coil drivers
1.0 flatpack logic		40 flatpacks logic
1.0 transistor		40 transistors
6.0 discrete parts		<u>240</u> discrete parts
		460 total parts

### Schematic Diagram of COPE Power Control Switch

Figure 3-6

Since each element contains all of its own switching circuitry, the required level of redundancy is automatically provided for. (This is not necessarily true for a centralized power switching system because portions of the circuitry are time shared between various elements.)

Figure 3-7 shows the mechanization of a centralized power switching for the same system employing a matrix array of coil drivers for forty elements requiring sixty relays. (This circuit was used on FLTSATCOM for control system power control.)

With this arrangement, the relay coils are driven at the junctions of an 8 x 10 array of vertical and horizontal drivers. The NH0008 drivers source current while the 5406 logic buffers sink current. The input control of the relay coils is a 7 bit binary word.

The parts count is:

60 relays
8 coil drivers
7 flatpacks logic
2 transistors
<u>106 discrete parts</u>
183 total parts

The problem with this approach is that it is only partially redundant and contains single point failures. A solution to this deficiency is to replicate the complete switching system with a standby redundant unit. Even though this doubles the parts count to 366 parts, it eliminates the single point failure modes and is still lower in parts count than that of decentralized control and provides a much greater degree of redundancy than that approach. A redundant centralized matrix switching system can have many failures that do not result in a loss of any circuitry block. Most failures only result in a partial loss of capability in a single switching unit which can always be provided by the redundant unit.

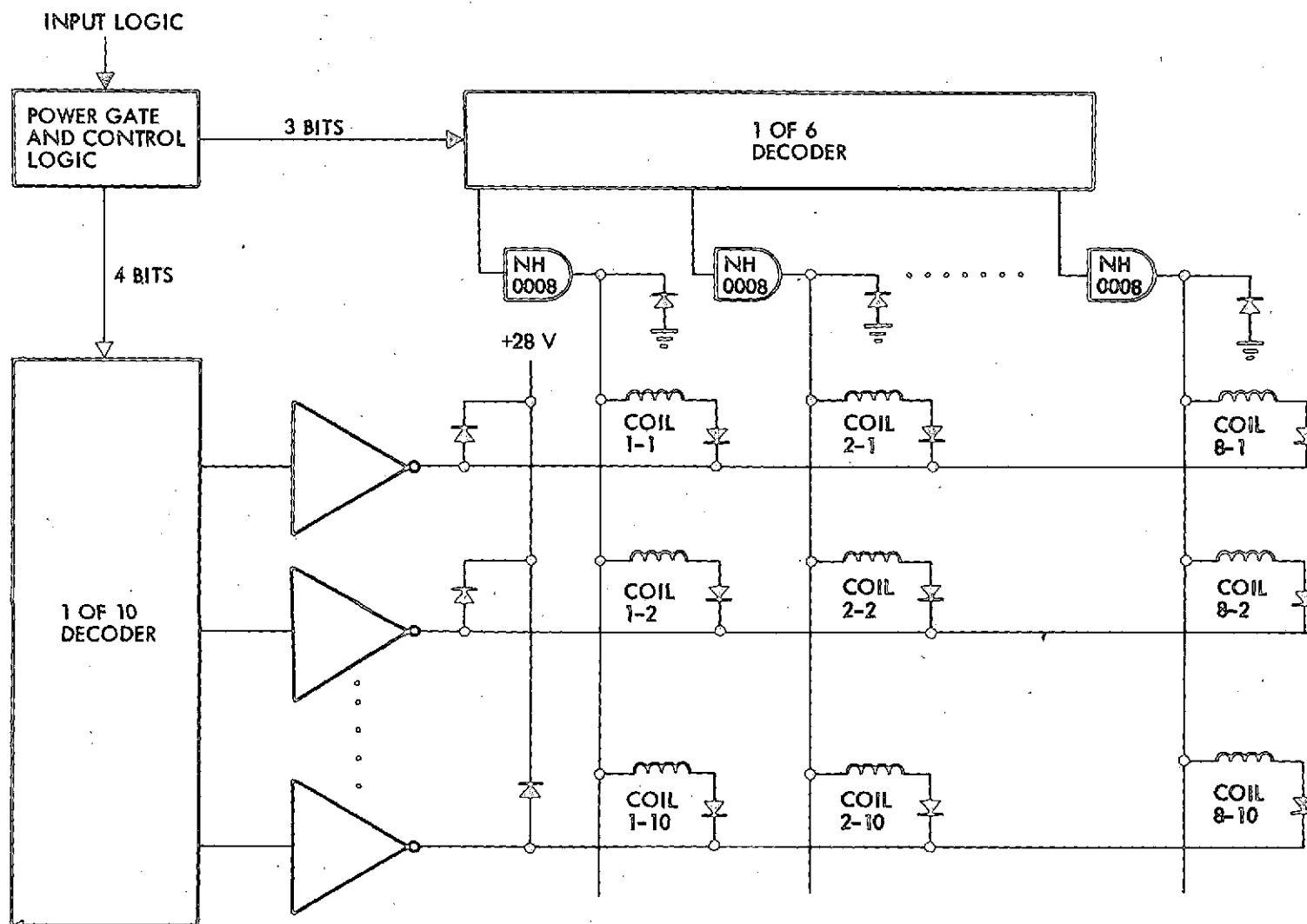


Figure 3-7

Centralized Power Switching Approach

Another approach is that shown in Figure 3-8. The control system elements are divided into two groups such that redundant elements are always in separate groups; i.e., single standby redundancy puts one in each group, double standby redundancy puts one in one group and two in the other, triple standby redundancy puts two in each group, etc. The switching matrix of Figure 3-7 is now cut in half from 80 coil drivers (40 elements) to 40 coil drivers (20 elements), arranged into a 4 x 10 matrix. This reduces the original parts count to:

Each switching assembly:

30 relays	} x 2 =	60 relays
4 coil drivers		8 coil drivers
7 flatpacks logic		14 flatpacks
2 transistors		4 transistors
62 discrete parts		<u>122 discrete parts</u>
		208 total parts

With this arrangement, it is necessary to utilize a particular power switching assembly to turn on or off a particular element; however, this should pose no significant problems to either ground command or processor controlled reconfiguration of the control system elements.

This approach meets the requirements of no single point failure and provides a minimum of parts count to accomplish the power switching function. A drawback is that there are a few failure modes that can wipe out up to half of the system redundancy. On a numerical reliability basis, this may be shown to be quite acceptable because of the small number of parts involved; however, from a practical basis, it may be worthwhile to provide the additional parts of the fully redundant, completely centralized power switching and distribution system.

The approach used may not be the same throughout the system. For example, it may be that the most important portions of the system (part or all of the processor) may require decentralized switching to preserve the processor modularity concept or because of the nature of the source of the control

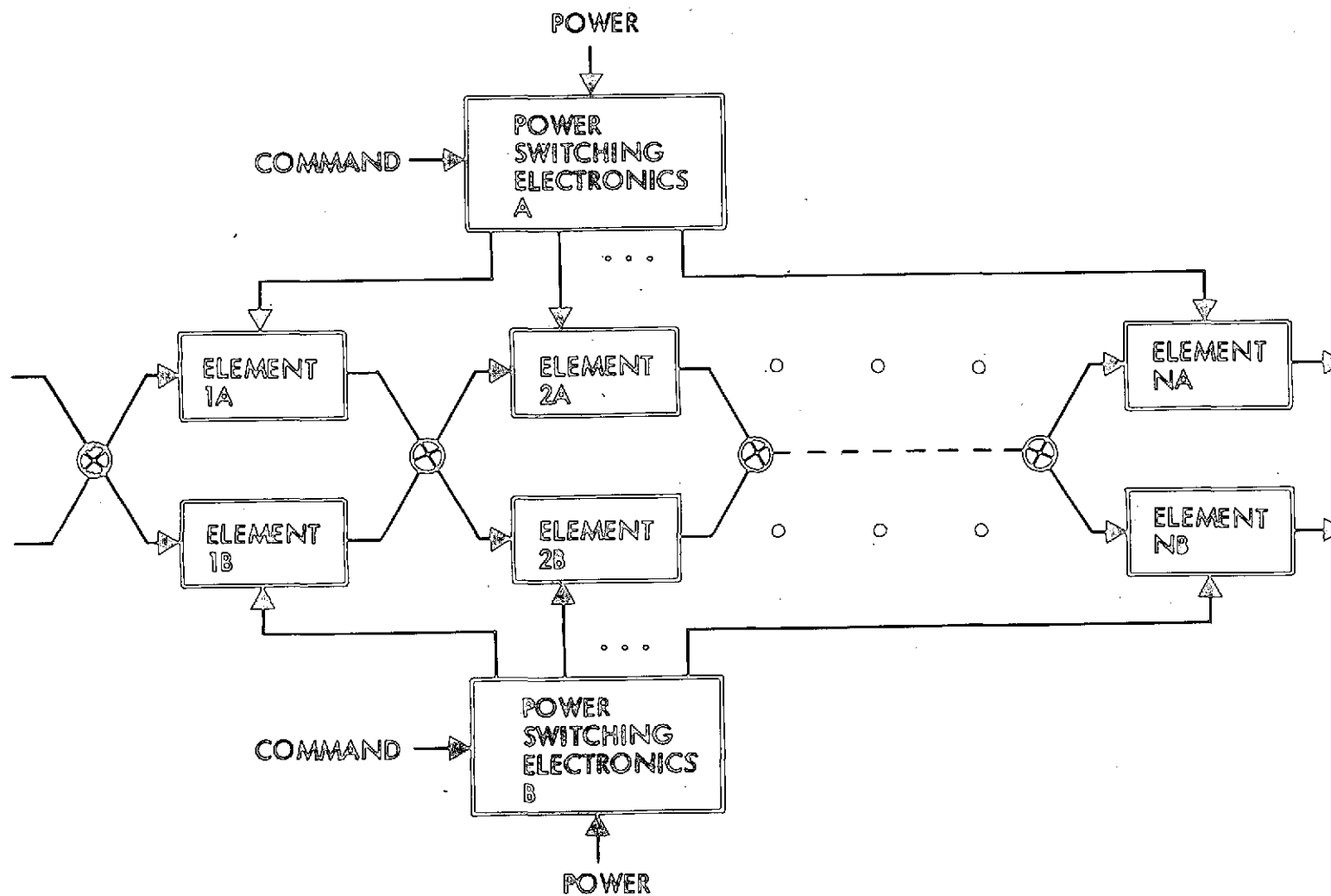


Figure 3-8

Group Switching Control

(reconfiguration) commands (the RCU, or equivalent). It may also be possible to centralize this power control into the RCU (but not necessarily with the same economies) while centralizing the peripheral power control in some peripheral element, under control from the IOU.

Note that for centralized control the control signal should be a serial word or words, decoded at the centralized location. For decentralized control the control signal is one bit. This may be a bilevel (as assumed in the circuit of Figure 3-6), discretes (which would simplify the control of latching relays), a switched clock (dc or ac coupled; which might reduce the number of lines to those peripherals already requiring clocks), etc.

The circuit shown in Figure 3-6 and as used on COPE uses latching relays to reduce standby power, combined with a novel circuit to allow the bilevel interface. Whenever the bilevel changes the relay switches. This is accomplished using relatively few parts.

The type of power switch utilized in the controlling and distribution of power to the various elements is of prime importance in terms of system power efficiency, failure modes and effects and reliability. Some of the candidate devices and circuits are:

- Relays
- Power semiconductors
- Power magnetics

Relays are the most commonly used devices for power switching and distribution systems. Historically, there have been a number of problems associated with the application of relays in spacecraft and other related aerospace hardware. Discussions with parts specialists and application personnel lead to very little hard evidence of a reliability or wear out problem associated with the use of relays if correct design and application criteria are followed. Most problems are a result of not carefully considering the environment the relay is being subjected to. During recent

years, where proper care has been exercised in the application of relays, there has been virtually no problems associated with their use.

In a power switching and distribution system utilizing magnetic latching-type relays, the duty cycle of relay operation is near zero. From a practical standpoint, the relays only get extensively exercised during ground testing. In flight they would be expected to be switched only once at power turn on (or maybe not at all if placed in the initial condition before launch and turn on of the secondary power converters). Even in the event of an automatic reconfiguration, any particular relay would not likely be operated more than a few times. A few of the blocks will be turned on and off during the course of the mission but the number of operations is always expected to be much less than what would be considered significant in terms of relay wear out phenomena.

The principal advantages of relays are:

- o Very high power handling capabilities and efficiencies.  
This is due to low on contact resistance
- o Zero standby power in either the on or off condition.  
Magnetic latching provides this feature
- o The magnetic latching feature also provides for a "hard" memory capability in a radiation environment or in cases of power loss or transient
- o Small size and weight. In general, a relay can handle higher voltages and currents than other candidate devices of equal size and weight.

The disadvantages of relays are:

- o Some people say they are unreliable ( probably because they are electromechanical devices rather than solid state)
- o Switching speeds in the 1 to 2 ms region (which is fast enough)

- Requires special attention in packaging design

Non-latching relays could also be used, but their constant power consumption weighs against them severely.

The use of power semiconductors falls into two categories:

- Straight on-off power switching
- Combined on-off power switching and regulation

On-off switching is straightforward and poses no special problems except for failure mode design. It is desirable for the power switch to fail in the off condition rather than on. This is particularly true if more than one block is available in standby redundancy since an "on" failure of the switch would be equivalent to failure of all blocks which are redundant with the block controlled by the failed switch.

The semiconductor switch also has the undesirable characteristic of requiring a voltage drop ranging from a few tenths of a volt to a volt in the "on" condition. The voltage drop and corresponding power loss is not insignificant when the total is considered. Any attempts to reduce the power loss of the semiconductor switching circuit results in an undesirable increase in circuit complexity and lower reliability.

The combination of power on-off switching and active regulation has been investigated as a means of letting the semiconductor switch perform multi-functions and, therefore, possibly eliminate other active regulators and filters in the system. As in the case for the on-off semiconductor switch, the circuit complexity gets out of hand when all of the desirable design criteria are met. If such a circuit could be made available as a one or two chip integrated circuit, it might prove to be a desirable approach.

The use of power magnetics for on-off control of individual blocks has been suggested in the past and certainly has merit from a conceptual standpoint. In this case, the power distribution is by an ac bus. The switching devices are reactors or transformers controlled by a dc current to cause saturation of the core material. The ac power is then rectified

and filtered in each block, as required. If three secondary voltages are required within the block, then six rectifiers and three filters would be required in addition to the switching reactors and/or transformer and current control circuits.

To our knowledge, this approach, while conceptually attractive, is not competitive on a power efficiency or parts count basis with that of a relay switching system. Like the switch-regulator, because the circuits must be replicated for each block in the system, the parts count is prohibitive if the switching circuit contains more than a few parts. For the same reason, even a small power loss in the circuits can add up to a very significant number if many circuit blocks are involved.

In summary, it may be stated that for control systems with a large number of circuit blocks:

- Centralized secondary power generation, conditioning and switching can be implemented with less total system parts and is more efficient in terms of reliability and cost
- DC secondary power distribution through magnetically latched relays is also more efficient in parts count and power loss than alternatives
- A matrix array for driving the power distribution switches greatly reduces parts counts and allows for multilevels of redundancy within the power switching circuitry.

### 3.1.3 Cross-Strapping Selection

In any system employing standby redundancy it is necessary for each block within the system to be provided with a means of communication with every other block that represents a needed input or output interface. This communication path is referred to as a cross-strap since it normally represents signal paths in a lateral direction to adjacent replicated circuits rather than end-to-end in the chain of signal processing. The generalized requirements on the cross-strap circuits are:

- Block Isolation. The cross-strap circuit should provide isolation of failure modes between the blocks. In addition, the cross-strap circuit itself should contain a minimum number of failure modes that affect more than one element. The ideal circuit will allow only one block failure per failure mode.
- Signal Path Switching. The control of the signal flow from one block to another should be as simple as possible. The ideal circuit allows communication only between blocks that are powered "on" and essentially ignores blocks that are powered "off".
- Signal to Noise Ratio. The cross-strap circuit should not degrade noise margins to less than that of a single noncross-strapped system. In logic interfaces, the waveform rise and fall time should not be appreciably degraded.
- Circuit Simplicity. The cost of cross-strapping many small blocks can become prohibitive if the cross-strap circuit represents an appreciable part of the individual block complexity. If a complex circuit is employed, the reliability gain of the standby redundancy approach is eroded and the system parts growth can have significant effects on size, weight and manufacturing cost. Circuit simplicity must be balanced with the other desired performance characteristics.

Generally speaking, a reliability diagram for a number of cross-strapped elements will be as shown in Figure 3-9. Each replicated block of an element has its own failure rate  $\lambda_{(x)}$  and to it must be added some portion of the failure rate of the cross-strap circuit. In some cases, there can be a portion of the circuit that represents an in-line failure rate and cannot be allocated to an element and is shown as  $\lambda_{cs}$ . A single point failure mode is obviously in this class; however, there are others as will be shown later in the detailed circuit descriptions. In the example, only one input and output to each block is shown (the minimum possible) for the reason of clarity. It should be realized that any number of inputs or outputs to a given block may be required. For

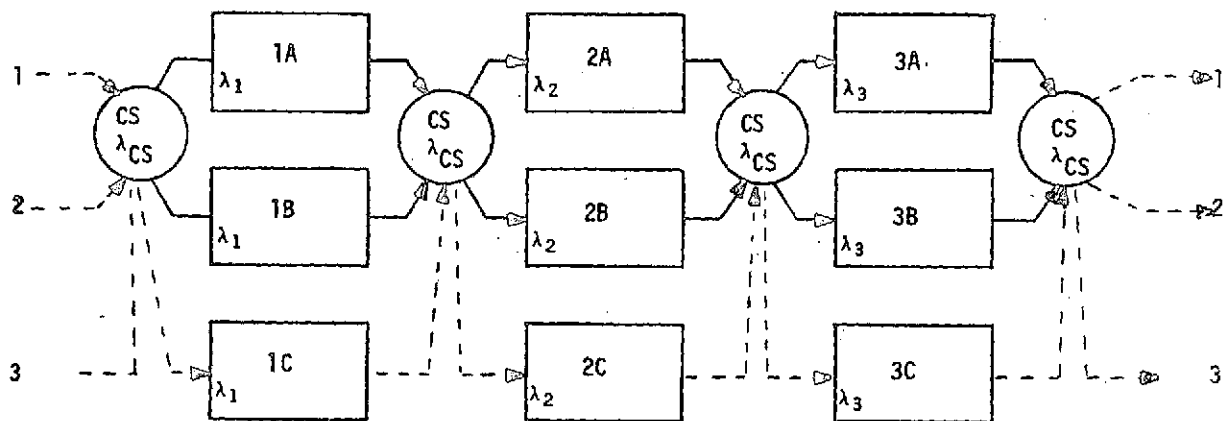


Figure 3-9 Typical Element Cross-Strapping  
(with single-point failures)

ORIGINAL PAGE IS  
OF POOR QUALITY

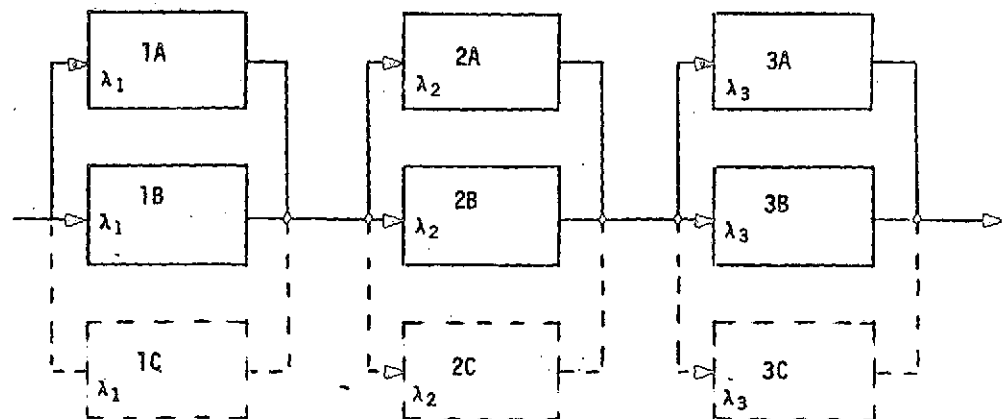


Figure 3-10 Typical Element Cross-Strapping  
(without single-point failures)

example, a block may require a single input and one hundred outputs, each to a different block or all to one or any other in-between combination. The obvious restriction is that if the number of inputs and outputs is too large, the requirements of simplicity cannot be met and the system must be repartitioned to reduce the number of element interfaces.

As shown, the replication of elements in the vertical direction (blocks per element) will generally be two or more while the number in the horizontal direction is determined by the total system requirement and choice of partitioning to control the size of individual element failure rates.

The cross-strap circuitry must always provide a signal path from any particular block output to the input of all replicated blocks it is feeding. For example, there may be only two number one blocks (1A, 1B) and five number two blocks (2A, 2B, 2C, 2D, 2E, 2F), etc. As can be seen, the ideal arrangement from a reliability standpoint would be an individual circuit path from output 1A to each of the number two block inputs and the same for output 1B. This would result in a large amount of wiring between elements, which is particularly undesirable if the elements are physically separated. A signal busing system with its own level of redundancy is the more optimum cross-strap for such cases.

If the cross-strap circuit meets the requirements of no single point failures and only one block being lost by any single failure, then the failure rates of the cross-strap components can be divided and summed with the failure rate of a particular block and shown as in Figure 3-10. It should be noted that quite often a particular part may be physically located in one block but its failure would actually result in a loss of a different block; therefore, its failure rate is assigned to the latter.

#### 3.1.3.1 Digital Logic Cross-Strap Circuits

The digital logic cross-strap is the most often encountered requirement in large systems with a high degree of redundancy. There have been a great many different circuits developed and used with varying degrees of success. In many cases, the designs were initially inadequately analyzed

and later were shown to have poor performance characteristics or subtle failure modes that compromised system reliability. In general, there are two ways to cross-strap logic:

- o Direct dedicated signal path between each block
- o Busing where the signal path is shared by more than one block.

The first is generally employed in systems involving single standby redundancy while busing tends to be more efficient in multiple standby redundant systems. The number of lines which must be cross-strapped are often also an influence.

The following circuits are representative of the different approaches and the trade-offs for digital (non-busing) signals.

Circuit-1, Figure 3-11. This approach to a cross-strap is a simple hard wire OR of the output and inputs of the four logic elements. It represents the minimum of circuit complexity, particularly when no special output or input buffering is employed. Either block 1A or 1B can communicate with block 2A or 2B through the single wire connecting blocks 1 to 2. There are two special requirements imposed on the use of this approach:

- o Either block 1A or 1B may be powered "on" but not both. The normal  $T^2L$  logic parts are not normally capable of a wire OR configuration.
- o The logic circuits employed must be capable of having their outputs pulled up to a logic "1" level when power ( $V_{cc}$ ) is not supplied to the element. (This requirement will be common to several other cross-strap circuits to be presented.) This requirement should not be confused with the so-called tri-state logic, whose outputs can be pulled to a 1 level with power applied to the logic element. The ability of a particular logic element to have its output pulled to a high state with no power applied is unique to the chip design of its output

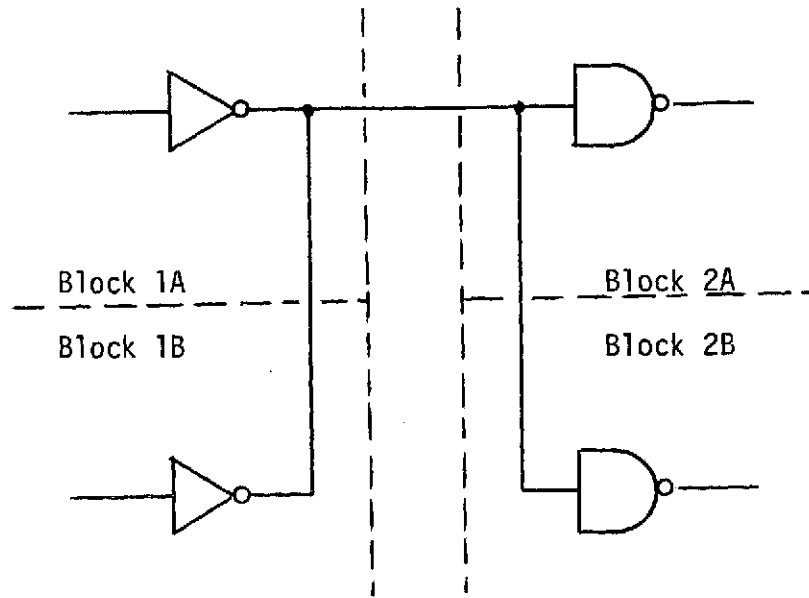


Figure 3-11  
Cross-Strap Circuit 1

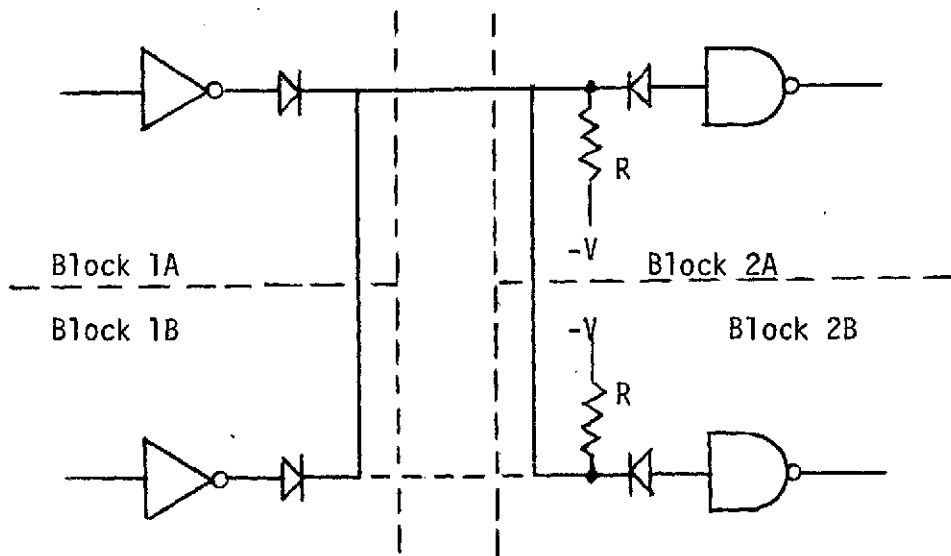


Figure 3-12  
Cross-Strap Circuit 2

stage and is different for different families of  $T^2L$  logic. For example, the TI series of low power  $T^2L$  has this capability while the Fairchild 93L series does not (it has an internal resistor connected between output and  $V_{cc}$ ). It should be noted that this characteristic is not a parameter covered by the normal device specifications.

- o In addition to providing the lowest complexity cross-strap, this approach provides the minimum wiring interface between the blocks. The obvious disadvantage of this approach is that it contains a number of single point system failure modes. A short to ground or  $V_{cc}$  of any one of the four blocks or a short or open in the wiring between the four blocks will cause loss of the complete system (assuming a mission critical signal path). This is usually an intolerable condition in large, long-life systems where individual block failure rates must be low (in the 2K to 4K bit range). A seemingly low failure rate in a nonredundant path will often turn out to be very significant in the overall total system reliability (see Section 3.1.1 ). Usually, top systems specifications rule out any designs that have single point failure from active components.

Circuit 2, Figure 3-12 . This approach has been used in several cases and eliminates the single point failure modes of the active components of circuit 1. It does not, however, eliminate the single point failures from opens or shorts in the wiring between units. Protection against opens, at the expense of increased probability of shorts, can be provided by an additional wire between units as shown in the dotted line. The diodes in series with the logic element inputs and outputs provide the isolation to protect against a short of the logic circuit to either ground or  $V_{cc}$ . A short of the diode will have virtually no effect on the block the diode is located in; however, it will fail the adjacent block if the logic blocks inputs or outputs cannot be pulled high with power removed.

The circuit does preserve good noise margins; however, it has the disadvantages of being somewhat complex and has poor fall time waveform response unless the pulldown resistor R is made small, which makes it inefficient in power.

Circuit 3, Figure 3-13. This cross-strap circuit is made up entirely of standard logic elements and meets the requirement of no single point failure modes as well as only a single block failure per cross-strap circuit failure. Also, there are no performance compromises to noise margin and waveforms. The major disadvantages are circuit complexity and the amount of interface wiring required. There are four wires in the horizontal signal path (which is generally more costly than adjacent signal paths) and one control wire to tell each receiver which sender is providing data. The control logic signal is obtained from the same signal source that enables power to the sending block.

Circuit 4, Figure 3-14. This logic cross-strap circuit is similar to the previous one except that no selection control logic is required. In addition, it is somewhat simpler in that the receiver requires only a single 2-input gate circuit with a pull-up resistor on each line. The pull-up resistors can be obtained thirteen to a flatpack integrated circuit so the cost in circuit board area and volume is minimal.

This circuit, like circuit 1, requires the capability for the logic circuit output to be pulled high with power "off". This function is provided by the resistor R which need only supply a few microamperes of leakage current at the block output. All single point failures are eliminated and any single failure only results in the loss of a single block. Four interface wires are required; however, this is necessary in all cross-strap circuits that meet the criteria of limiting all single failures to a single element. In addition to being relatively simple and completely automatic in signal selection, the circuit does not degrade noise margins or waveform characteristics. This circuit probably represents the best compromise between the various performance requirements and complexity for cross-strapping single standby redundant systems.

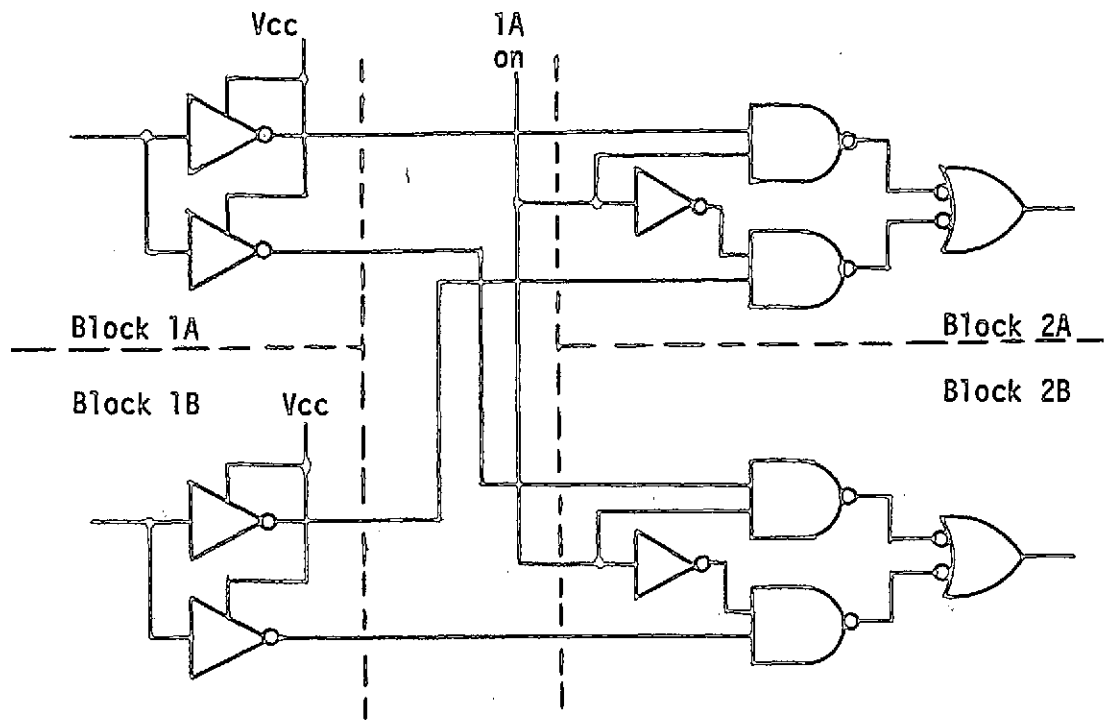


Figure 3-13  
Cross-Strap Circuit 3

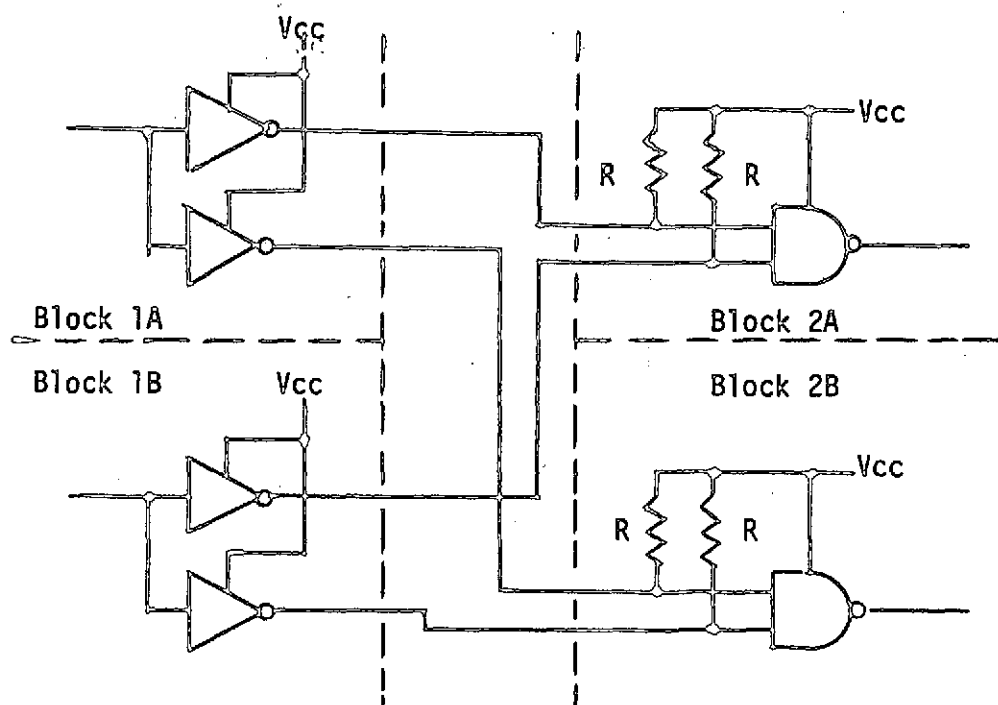


Figure 3-14  
Cross-Strap Circuit -4

### 3.1.3.2 Digital Busing Cross-Strap Circuits

Busing-type cross-strap circuits tend to be more efficient whenever there is multiple redundancy (dual standby or above) or where ever there are many elements interfacing with the same signal lines, such as in the processor, where most of the elements can interface with the same internal signal lines. For example, each address line goes to each memory block, as well as the ACU.

Circuit 5, Figure 3-15 . This circuit shows the arrangement of a redundant bus system employing standard  $T^2L$  logic gates. The selection of the bus in use is made by a bussed logic signal to each receiver. The circuit shown is for a single logic bit and two more bus lines are required for each additional logic bit at the interface. The number of blocks that can be replicated is limited only by the fan-in and fan-out loading of the logic employed. As in the single standby redundant cross-strap systems, the selection of the elements in use is done by the application of power to the block.

From a reliability standpoint, bus cross-strap systems must be treated differently than single standby cross-straps. Each bus and all the circuitry connected to it must be treated as an element itself and placed in series with the reliability calculations.

It is possible to also extend the redundancy of the busses by providing additional output drivers and receivers in each block along with additional bus selection lines and logic. Such a requirement is not likely for just cross-strapping replicated elements. It may, however, be a requirement for a data bus system, such as a processor input/output that is servicing a large number of different blocks.

Circuit 6, Figure 3-16 . Often the power consumption of the gates used in the cross-strap circuitry is of considerable importance. For the busing type cross-strap of Circuit 5, this power can be reduced to half by using power gating. The bus select signal allows the power to be turned on to half the senders and receivers at one time. In terms of input loading, an off receiver should be counted as one-half its normal load in the high state and as no load in the low state. The output of an off

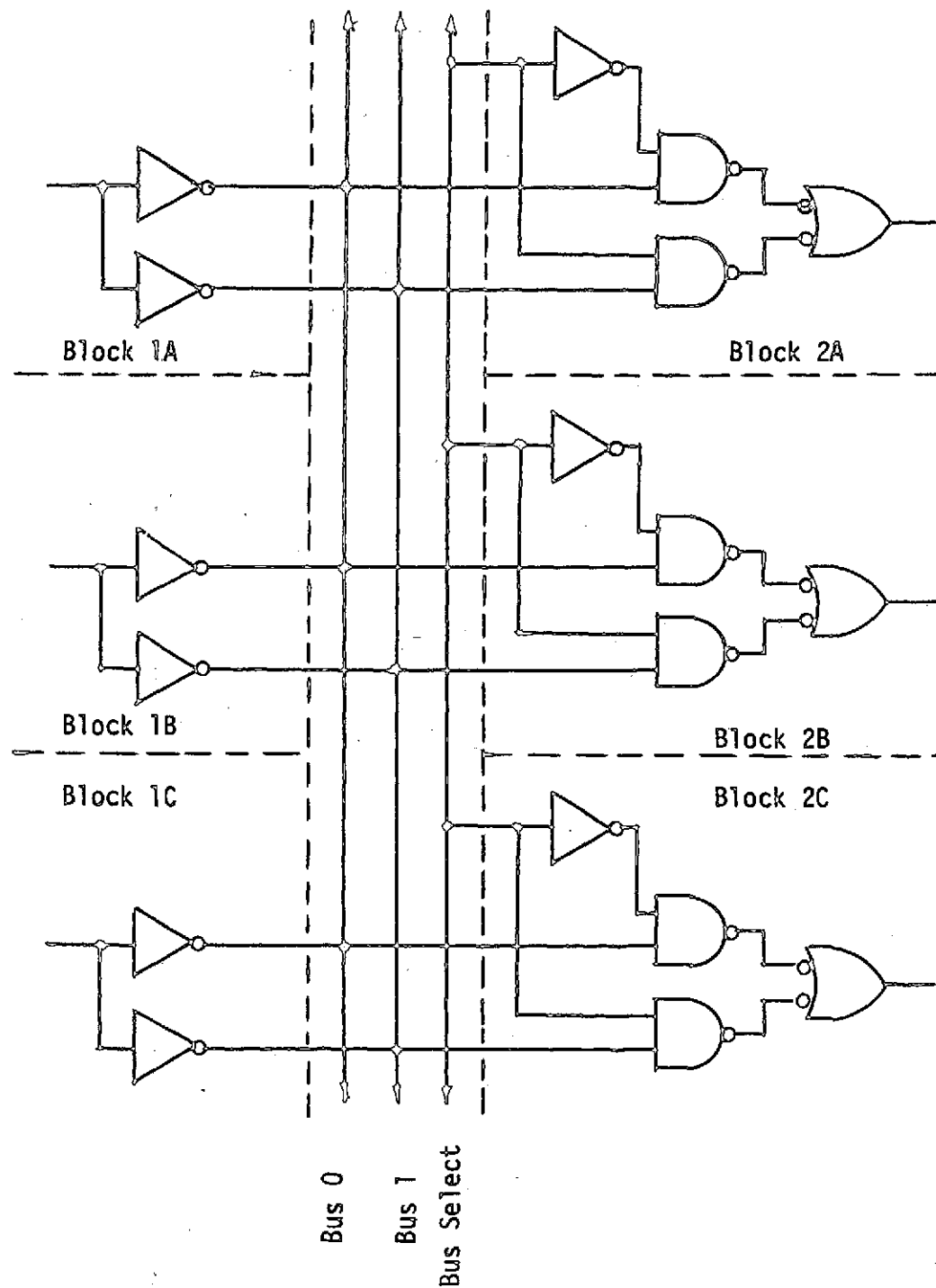


Figure 3-15  
Cross-Strap Circuit 5

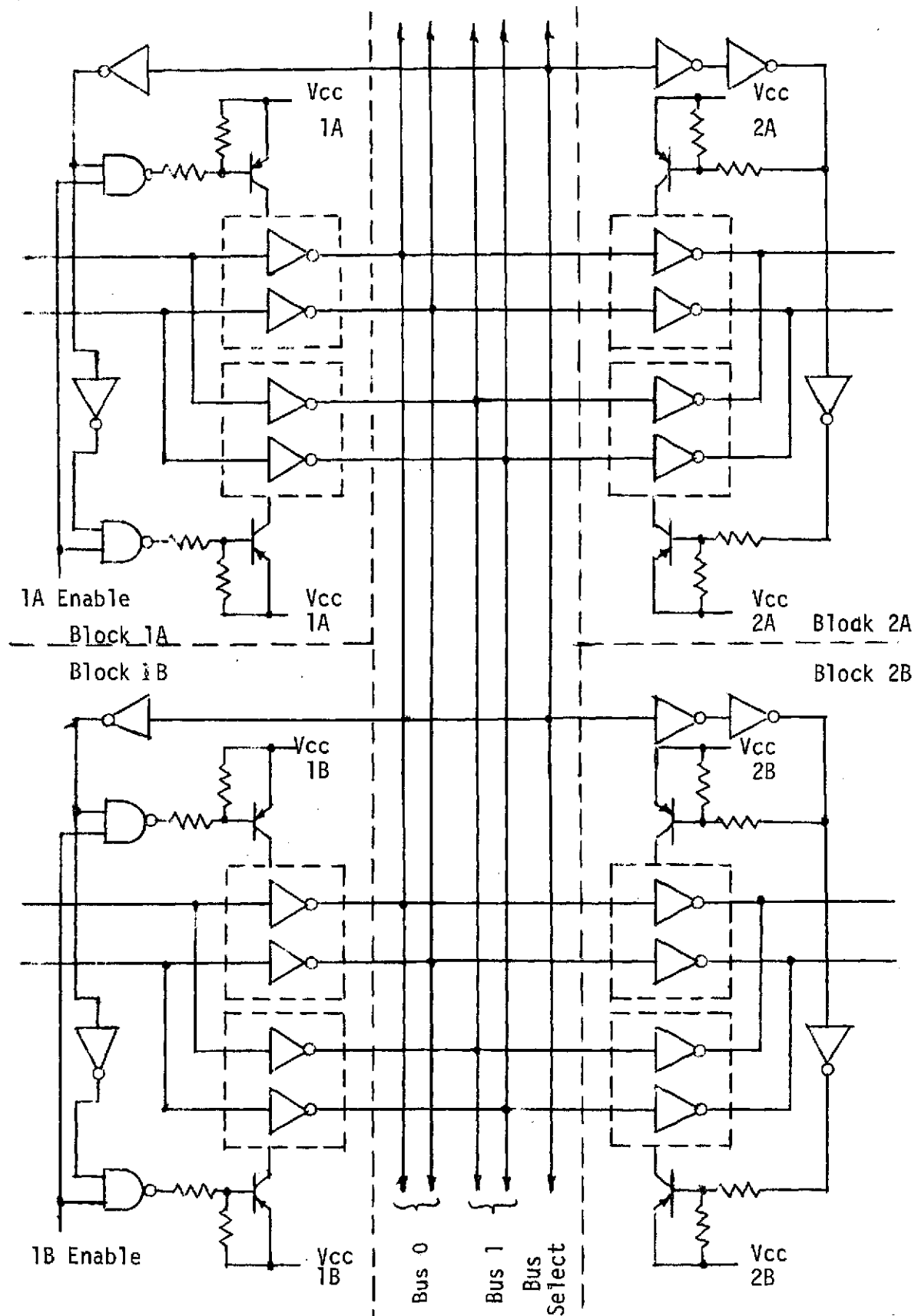


Figure 3-16  
Cross-Strap Circuit 6

receiver should be counted as one load in the high state and as no load in the low state.

Here the transistors gate the power into all the senders (or receivers) associated with the particular bus. Care must be used to segregate this division of gates within each block. Note that two lines are shown in this example. Only the two power gate circuits need be used for each block, no matter the number of senders and receivers (within transistor power limitations).

This is the circuit used for the internal bus of COPE. It is quite easy to see how bussing can reduce the number of parts (and wires) for the more complex systems.

Circuit 7, Figure 3-17. Another approach that can be used is the triple-redundant cross-strap with voting. All three buses are actively driven by the powered on block and majority voting is used by the powered on receiver(s) to determine the correct logic state.

Note that for any of the bus approaches, the bus may be either uni-directional relative to signal flow, or bi-directional. Bi-directional flow can only be accommodated if the information sources are controlled in a manner so that two sources will not be trying to put out information on the same line at the same time. The use of bi-directional buses can save lines.

It should also be noted that the buses discussed here are not the elaborate data bus systems such as proposed for space shuttle and other very large systems where the propagation delay from one end of the spacecraft to the other at the high bit rates is important or where wire weight of long runs is ruling. That type of system usually goes to single-wire bi-directional buses involving totally serial data interchange using self-clocking coding (Manchester or equivalent) and modem interfacing. Such systems are not needed for the control systems of moderate sized spacecraft.

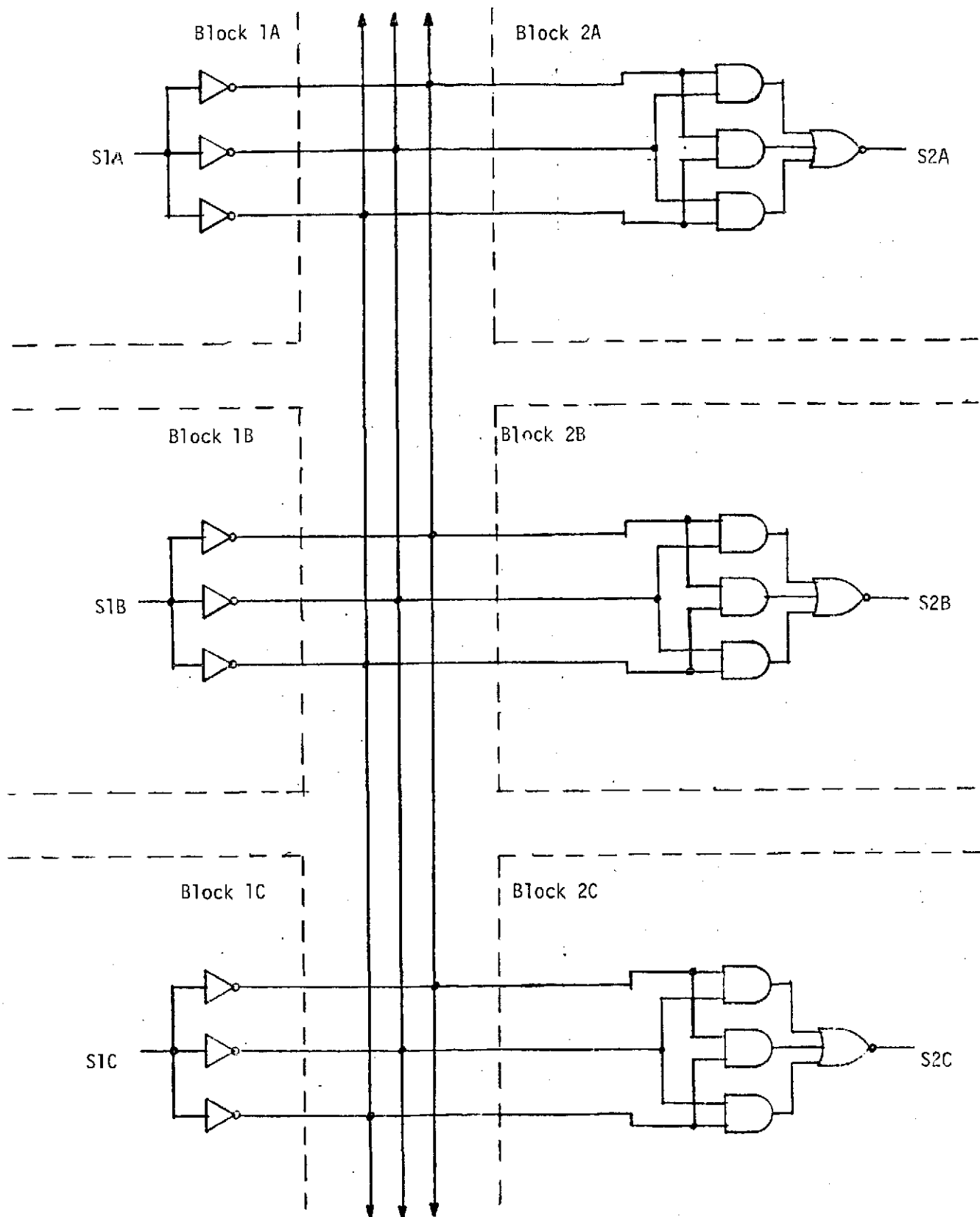


Figure 3-17  
Cross-Strap Circuit 7

ORIGINAL PAGE IS  
OF POOR QUALITY

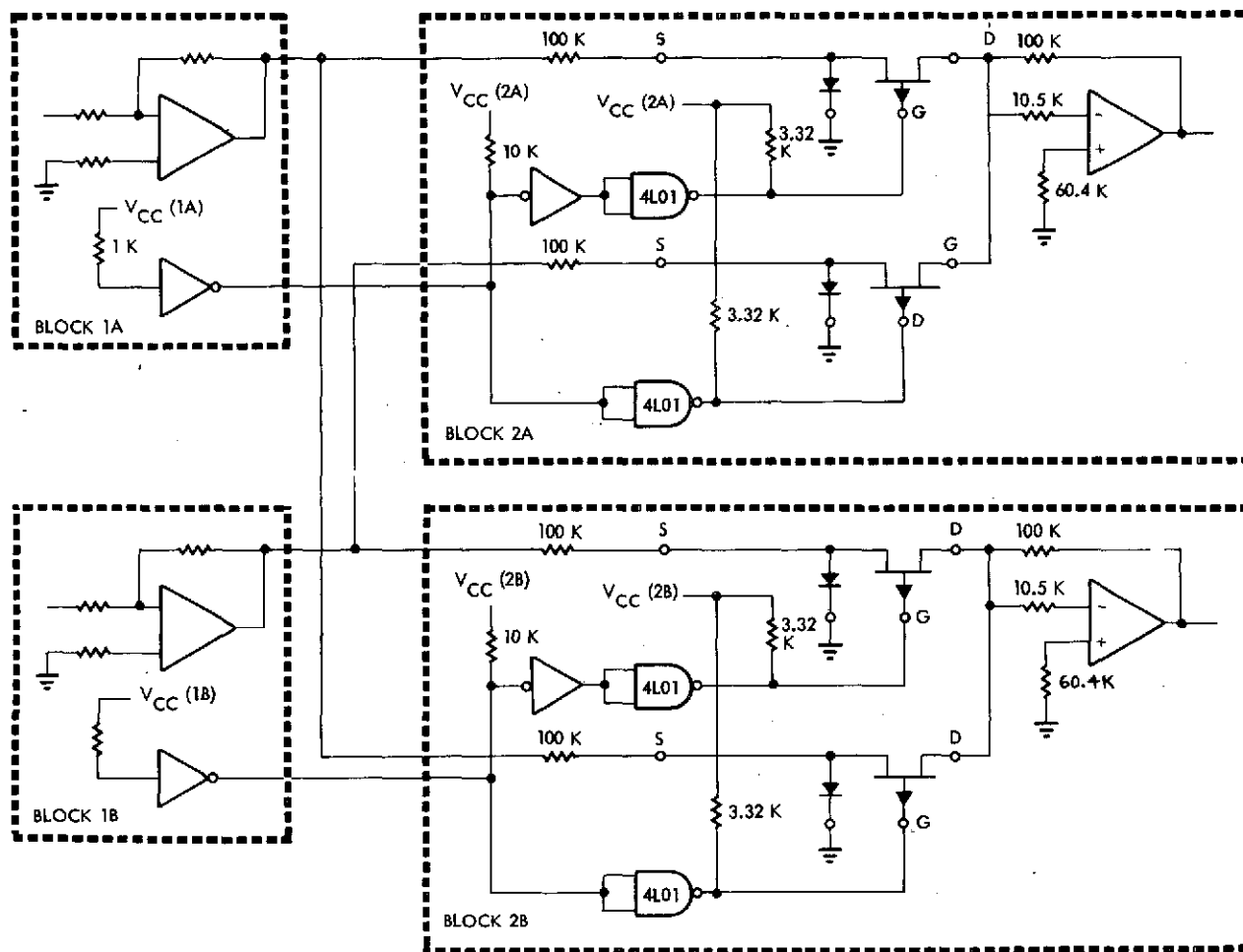
### 3.1.3.3 Analog Cross-Strap Circuits

Analog cross-strapping should be avoided if possible due to its complexity. If needed, there are proven methods that are available and have been used. Usually the best approach is to switch the signals at the receiving end, under the power control from the sending end, using FET switch analog modulators. An example of such a circuit is shown in Figure 3-18..

Two sending blocks and two receiving blocks are shown for one analog signal. The analog signal from block 1A is sent to both receiving blocks, as is the signal from block 1B. Since only 1A or 1B can be on, the digital control signals are also sent across to control the FET switches.

The combined load (of 50K) is well within the driving amplifier capability. This circuit has no single point failure. So-called analog switches, which have all of the drive circuitry within the DPDT package, can also be used.

Completely passive analog cross-strap circuits have also been invented, but they are usually inefficient and do not seem to possess any particular advantage over the circuit of Figure 3-18.



Analog Cross Strap Circuit

Figure 3-18

### 3.1.4 Redundancy Management

Without discussing for the moment the problems of fault detection and diagnosis (see Sections 3.5.5, 3.7.4 and 3.7.5) let us assume that faults can be reliably detected by combinations of hardware (BITE) and software techniques.

We shall assume that all of the peripherals and the majority of the processor is organized using standby redundancy. There will be some portion of the processor that will need a passive redundancy (the so-called "hard core"). All other parts of the system must be controlled or managed from some higher-level decision-making portion.

There are several ways in which the redundancy of the system can be managed. Some of these management systems which have been studied are:

- Centralized Hardcore System, Fixed
- Centralized Hardcore System, Bootstrapped
- Distributed Hardcore System, Bootstrapped
- Distributed Hardcore System, Parallel
- Distributed Computer Network

These systems will be briefly described in the following sections.

#### 3.1.4.1 Centralized Hardcore System, Fixed (Figure 3-19 )

Here the hardware is the system element that must function correctly for detecting failures in all system elements (possibly excepting peripherals) and performing reconfigurations. Reconfigurations are performed by following a hard-wired combinational approach which continues until the hardcore receives signals indicating that the system is in normal operating conditions. Performance monitoring is on the basis of software and hardware fault signals exclusively. The reconfiguration sequence can be structured in various ways, depending on speed of recovery and hardcore complexity considerations. The approach used in COPE starts by switching redundant units of different types, one at a time, until all types have been tried. If this process fails, a more comprehensive sequence is

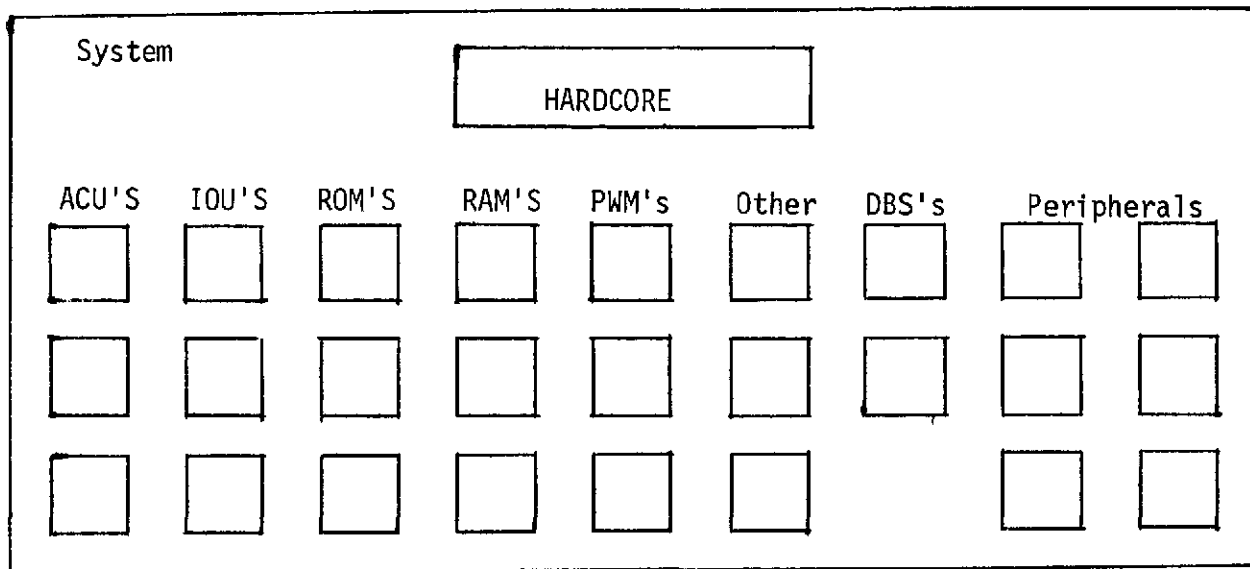


Figure 3-19 Centralized Hardcore System - Fixed

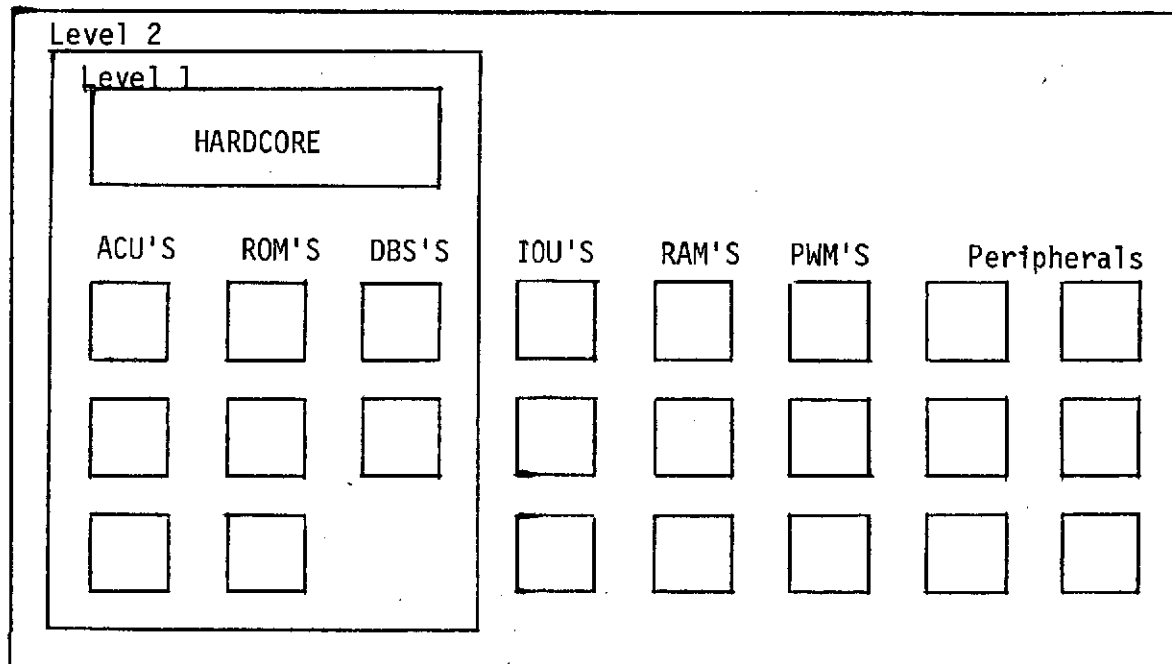


Figure 3-20 Centralized Hardcore System - Bootstrapped

followed, where all possible combinations of redundant elements of different types are tried.

#### 3.1.4.2 Centralized Hardcore System, Bootstrapped (Figure 3-20 )

Hardcore is defined here as that portion of the system that must function correctly to detect failures in Level 1 components and software and to perform reconfigurations. After the hardcore determines that the essential functions performed by Level 1 components are correct, both the hardware and Level 1 elements are used to test elements in Level 2. If reconfigurations are necessary, they are performed by the hardware upon request from the Level 1 system. After successful completion of tests, normal operation starts. The hardcore monitors performance either by means of fault signals or by periodically interrupting operations to repeat bootstrapping procedures, or by means of a combination of both methods. The Level 1 components are considered to be the "primary processor".

#### 3.1.4.3 Distributed Hardcore System, Bootstrapped (Figure 3-21)

In this approach, the hardcore functions are limited to monitoring the performance of any of the active Level-1 elements and switching redundant units in operation in case of failure. Each of the Reconfiguration Control Units in Level 1 performs diagnostics and reconfiguration of the elements in Level 2 which, when operating correctly, are used to test and reconfigure, if necessary, the equipment in Level 3. This concept has the advantage of minimizing the number and extent of the hardcore functions, which are normally made highly redundant to provide sustained operation under failure.

A slightly different version is shown in Figure 3-22 . This differs from Figure 3-21 primarily in that the IOU is shown as a separate level. The primary processor would utilize the IOU to control the configuration of all Level -4 components

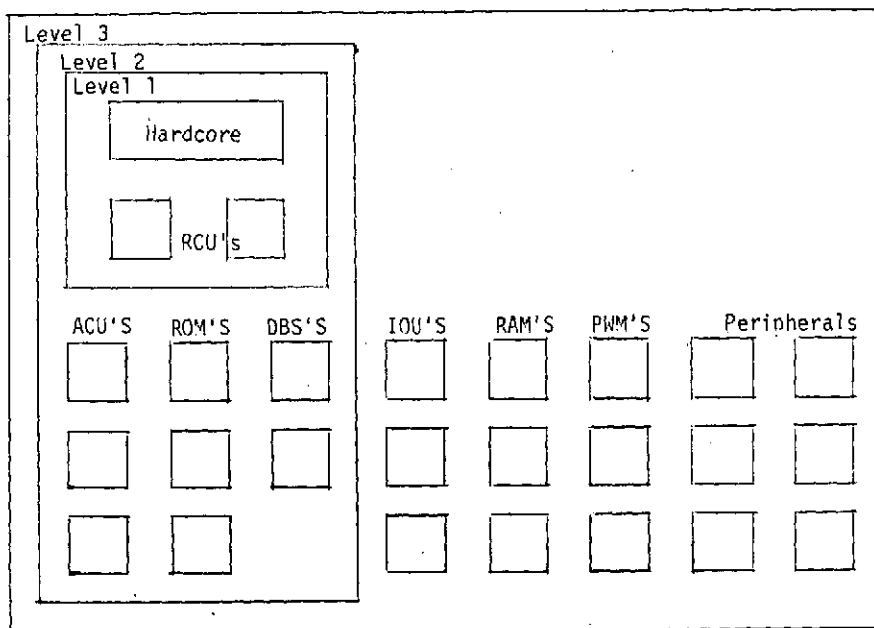


Figure 3-21 Distributed Hardcore System - Bootstrapped

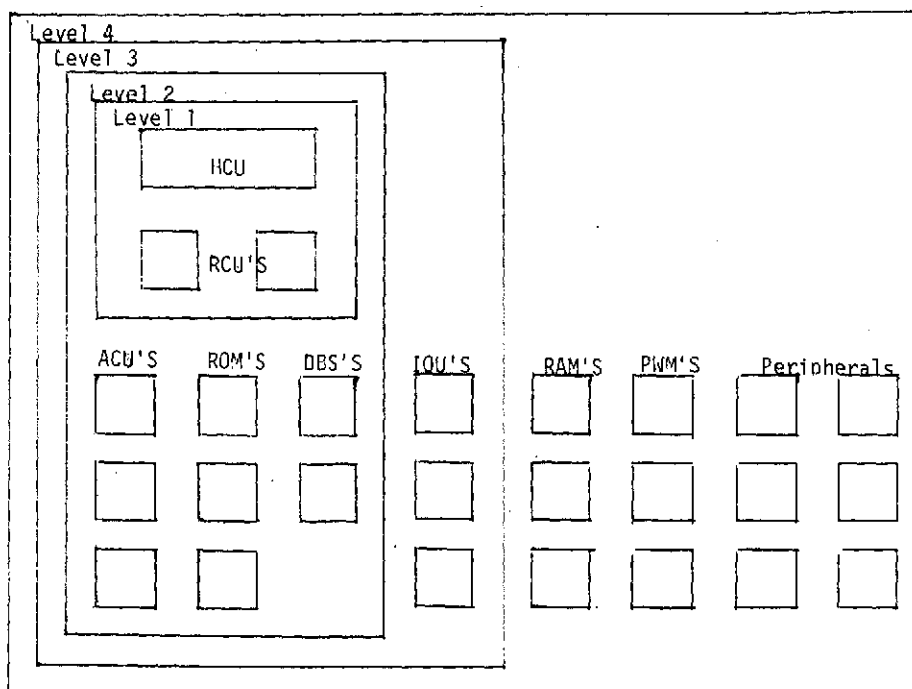


Figure 3-22 Distributed Hardcore System - Bootstrapped  
(with separate IOU level)

The hardcore is here called a Hard Core Unit or HCU. The RCU's are in standby redundancy to each other, with the RCU selection (management) being accomplished by the HCU.

#### 3.1.4.4 Distributed Hardcore System, Parallel (Figure 3-23)

This approach was originally proposed by Forbes et al, [9], and it was implemented in the IBM DX-1 computer. In the Level 1 components, the control and arithmetic elements are partitioned into two identical groups each working with half a word. During normal operation the two portions act in parallel performing operations on full-word operands. During diagnostics, these partitions operate independently to test each other. The hardcore includes the comparison and switching circuits and the common microprogram memory that controls the partitions. This hardcore is claimed to be about 10% of the DX-1 machine. Whether the system is truly distributed or not depends on the approach used for switching from one Level 1 machine to the other. This concept is probably more easily implementable in a machine designed on a bit-sliceable basis (byte organized).

#### 3.1.4.5 Distributed Computer Network (Figure 3-24)

This concept was proposed by Preparata, Metze, and Chien [10]. This figure shows diagnostic relationships between elements. E.g.,  $U_0$  diagnosis  $U_1$ ,  $U_1$  diagnoses  $U_2$ , etc. This implies that  $U_0$  accesses  $U_1$ , then transmits the patterns into  $U_2$  and obtains responses which are compared to prestored fault-free responses and the results are transmitted to other subsystems. This assumes that a majority of these subsystems are fault free. The scheme can be implemented by either software or microprogramming. Such a system is unnecessarily complex for a spacecraft control system.

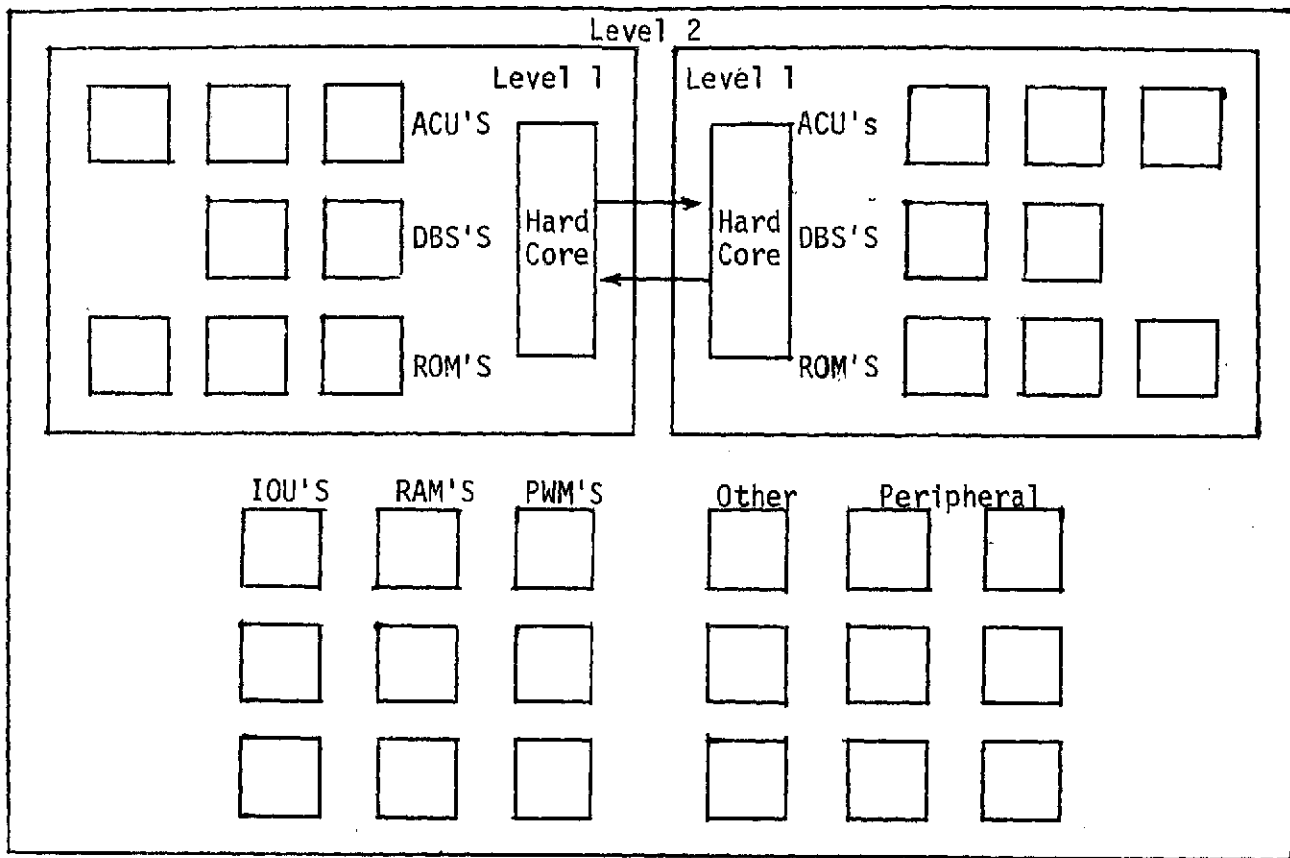


Figure 3-23 Distributed Hardcore System - Parallel

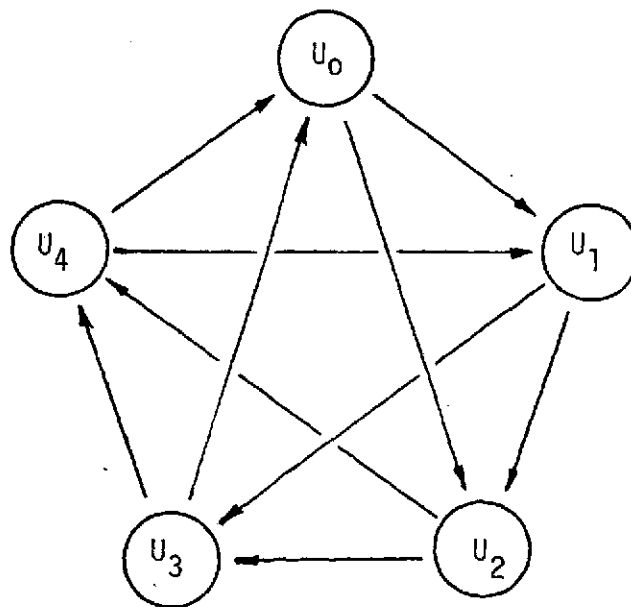


Figure 3-24 Distributed Computer Network

#### 3.1.4.6 Selection of Management Approach

Of the preceding approaches, the bootstrapped distributed hardcore system of Figure 3-22 seems to offer the most advantages for application to the spacecraft control systems. These advantages are:

- Minimization of hardcore (HCU) functions with resultant simplicity of HCU design and lowering of the TMR overhead to its absolute minimum.
- Reduction in tasks for the RCU's, by removing all but the primary processor (ACU, DBS & ROM) and the IOU control from the RCU.
- Control of the RAM, PWM & all peripherals through the IOU, under software management. These controls are through the normal outputs (bilevels or serial) of the IOU.
- The possibility of using centralized power control for most of the system is available (see Section 3.1.2).

The requirements for the RCU and HCU are discussed in Section 3.7.6. It should only be noted here, that the HCU need only detect faults in the RCU and then substitute the standby RCU for the faulty one. There are only three types of possible RCU faults. These faults lead to:

- A failure of the RCU to detect a primary processor fault (through its own faulting).
- A failure of the RCU to act to correct a detected fault (to reconfigure the primary processor).
- The RCU causes a reconfiguration to occur, even though no processor faults occurred.

The management of the available RCU's by the HCU is exactly analogous to the management of the processor by the RCU. Note that the HCU is TMR, with output (the power control signals to the ACU's) voting.

## 3.2 Control System Design

The control system design process is multi-faceted, encompassing functional characteristics, operational characteristics, and system interfaces. This section deals briefly with these elements of control system design specifically in relation to the implementation of fault-tolerant system design.

### 3.2.1 Control System Implementation

The control system is configured of sensors and actuators which are operated on by the fault-tolerant attitude control electronics to implement the mission requirements. The overall system design is developed to result in well defined performance characteristics. At the top level, this is accomplished by careful definition of the subsystem functional and operational characteristics, i.e., the control modes, the sensors and actuators used for implementation, the control processing required, and a review that all mission functions are achieved within this framework.

Control system modes are established to encompass a well defined system functional capability, and are therefore functionally unique and mutually exclusive. In other words, performing a certain control system function implies operation in a particular control system mode and vice-versa. This characterization results in the implementation of mode logic, i.e., the on-board process by which the control system makes decisions as to what to do in response to external stimuli such as commands, sensed performance criteria, etc.

It is key that there be no confusion as to the control system operational response to such stimuli. In implementing a fault-tolerant control system, such decision processes are especially critical and must be carefully developed.

This process, relative to the control system of interest as outlined in Section 2.1.1, is summarized in the chart of Table 3-4. In this case, a standby mode, five operating modes, and a backup mode have been defined. Within this structure there are also defined what may be called submodes, i.e., use of equipment or functions which are not of themselves mode/function unique, but are more related to the level of performance (e.g. accuracy) achieved. This encompasses: 1) selected use of wheels or RCS; or 2) selection of fine or coarse sun sensors.

The standby and operating modes were briefly summarized in Section 2.1.1. The backup mode is an "escape mode" in the event of critical failures portending power loss or gross errors in attitude control system performance. The objective is similar to the sun acquisition function of the Acquisition Mode, but the control is mechanized in special-purpose hardware external to the primary fault-tolerant electronics to relieve dependence on as much hardware as possible.

This backup mode is reached from any other (except despin) mode via malfunction indication logic. Malfunction indication is provided on-board with regard to critical performance measures (which may be mode related) but which are an integral part of the overall fault-tolerant design. This arises since the priority to reconfigure and recover via the fault-tolerant system design criteria is clearly higher than entry into a back-up mode in the event of control system failures. Backup malfunction indication logic must therefore be developed which is not at cross purposes with the overall design. Indication of continued low power, signals from the fault-tolerant electronics reconfiguration logic, or ground override are each valid elements of such logic.

This mode framework, although structured relatively rigidly, provides a great deal of operational flexibility. This is noted by considering the operating regimes defined in Table 3.4, the mission timeline outlined in Section 2.0, and the mode flow diagram of Figure 3-25. This figure develops the paths by which mode transition may take place. Certain characteristics can be readily noted (e.g. limited exit/entry to/from certain modes), but details of actual mode logic to implement mode transition keys upon a variety of factors, including:

- Assuming equipment to be used is in an operational "GO" status
- System within performance boundaries essential to performance in the selected mode.
- Command override: force-in/lock-out type operation
- Nature of autonomy and interaction with fault-tolerant system configuration control

Implementation of the control modes is performed within another functional framework characterized by division among meaningful processing functions. This is identified in the control system functional block diagram shown in

TABLE 3-4  
CONTROL SYSTEM  
FUNCTIONAL/OPERATIONAL CHARACTERISTICS

OPERATING REGIME	MODE	SENSORS	PROCESSING	ACTUATORS
<ul style="list-style-type: none"> <li>• Preflight</li> <li>• Launch</li> <li>• Other as Commanded</li> </ul>	Standby	---	Quiescent/Status	---
<ul style="list-style-type: none"> <li>• Despin</li> </ul>	Despin	Gyro(s)	Rate Control for despin axis	Despin Thrusters
<ul style="list-style-type: none"> <li>• Sun Acquisition</li> <li>• Canopus Search/Acq.</li> </ul>	Sun Acquisition	<ul style="list-style-type: none"> <li>• Sun Sensors</li> <li>• Roll Gyro</li> </ul>	<ul style="list-style-type: none"> <li>• Sun Acq. &amp; lock in 2-axes</li> <li>• Rate Control w/bias about sunline</li> </ul>	Thrusters
<ul style="list-style-type: none"> <li>• Cruise</li> </ul>	Celestial Point	<ul style="list-style-type: none"> <li>• Sun Sensors</li> </ul>	<ul style="list-style-type: none"> <li>• Sun Point w/bias in 2-axes</li> <li>• Star track constraint for orientation about sunline</li> </ul>	Thrusters or Reaction Wheels
<ul style="list-style-type: none"> <li>• Reorientation</li> <li>• Control during fly-by</li> <li>• Cruise (as desired)</li> </ul>	Inertial Point	<ul style="list-style-type: none"> <li>• IRU</li> </ul>	<ul style="list-style-type: none"> <li>• 3-axis stabilization based upon inertial derived error signal</li> </ul>	Thruster or Reaction Wheels
<ul style="list-style-type: none"> <li>• Velocity Change</li> </ul>	TVC	<ul style="list-style-type: none"> <li>• IRU</li> </ul>	<ul style="list-style-type: none"> <li>• Thrust vector control in 2-axes</li> <li>• Roll control using RCS</li> <li>• Control of thrust period based on accelerometer output</li> </ul>	Vane Actuators plus Thrusters
<ul style="list-style-type: none"> <li>• In event of catastrophic failures</li> </ul>	Backup	<ul style="list-style-type: none"> <li>• Wide-angle Sun Sensors</li> </ul>	<ul style="list-style-type: none"> <li>• Sun acquisition &amp; lock in 2-axes</li> <li>• Quiescent control about sunline</li> </ul>	Thrusters

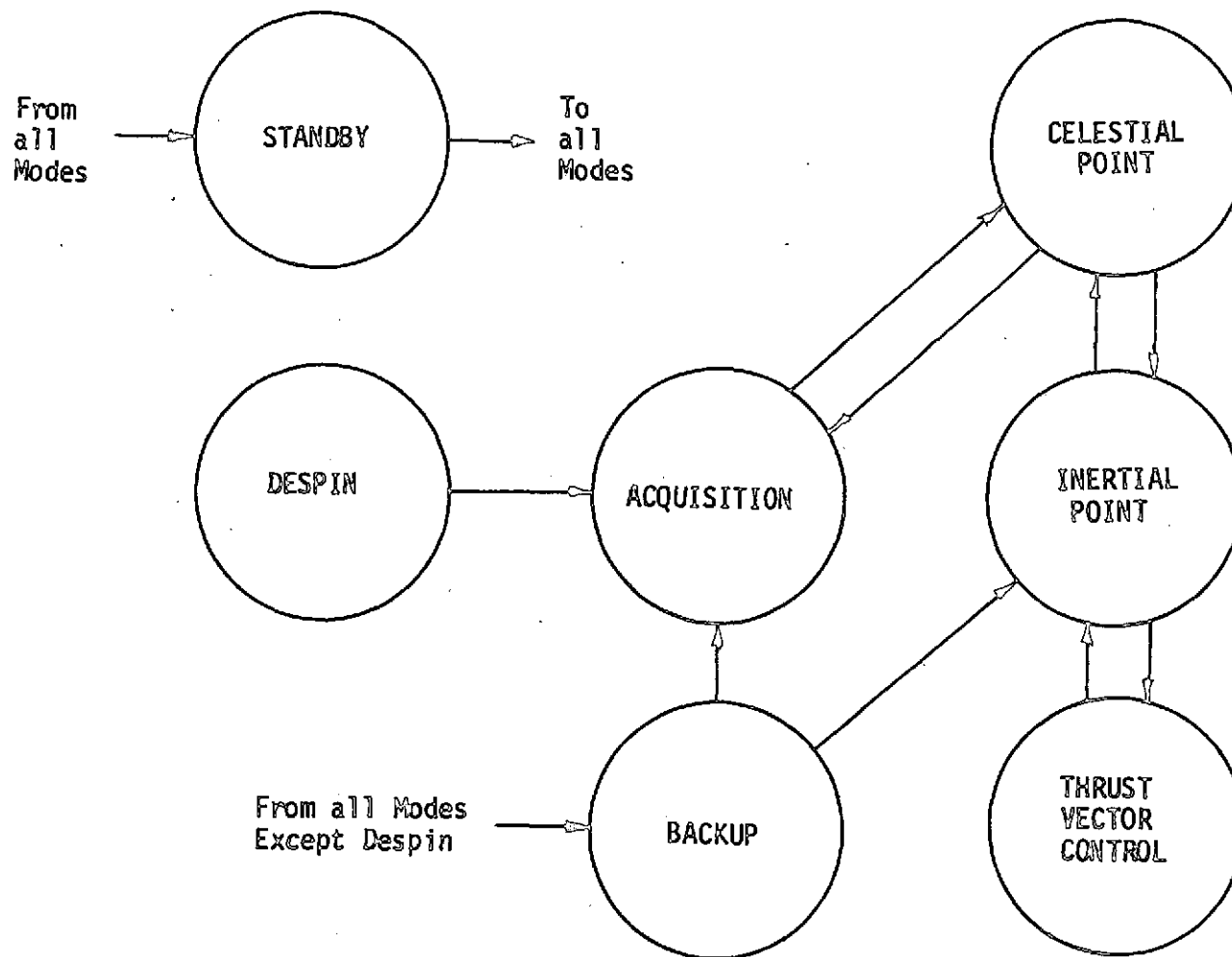


Figure 3-25 Control System Mode Switch Paths

Figure 3-26. Consider initially the sensor functions as observed at the processing interface:

- Sun sensors: two-axis measure of sun-line relative to the sensor defined coordinates.
- Star Sensor(s): two-axis measure of star line of sight relative to the sensor defined coordinates
- Inertial Reference Unit:
  - gyros provide inertial rates and/or (incremental) attitude as measured about the inbound input axis.
  - accelerometer provides acceleration (or incremental velocity change) along the sensitive instrument axis.

Likewise, the actuator functions as seen at the interface comprise:

- Reaction wheels:
  - reaction torque in response to drive signal
  - wheel speed measure, i.e., tachometer
- TVC vanes:
  - vane deflection in response to drive signals
  - measure of deflection angle
- RCS thrusters
  - thrust of fixed magnitude whose duration is in response to control signal
- Scan Platform Actuators:
  - actuator gimbal torqued in response to drive signal (two-axis)
  - platform gimbal angle measure in each of two axes

The means by which these interfacing signals and the sensor/actuator supporting functions are processed must be considered within the functional design of the control system. It appears to be particularly key to maintain an interface of low volatility (e.g. avoiding requirements of memory) and minimum complexity. In allocating the functions to be performed within the processor, it is desirable, in most cases, that not any one function became the demanding driving requirement on computer speed, e.g. "tail-wags'dog". Although this may happen in certain cases (e.g. a function

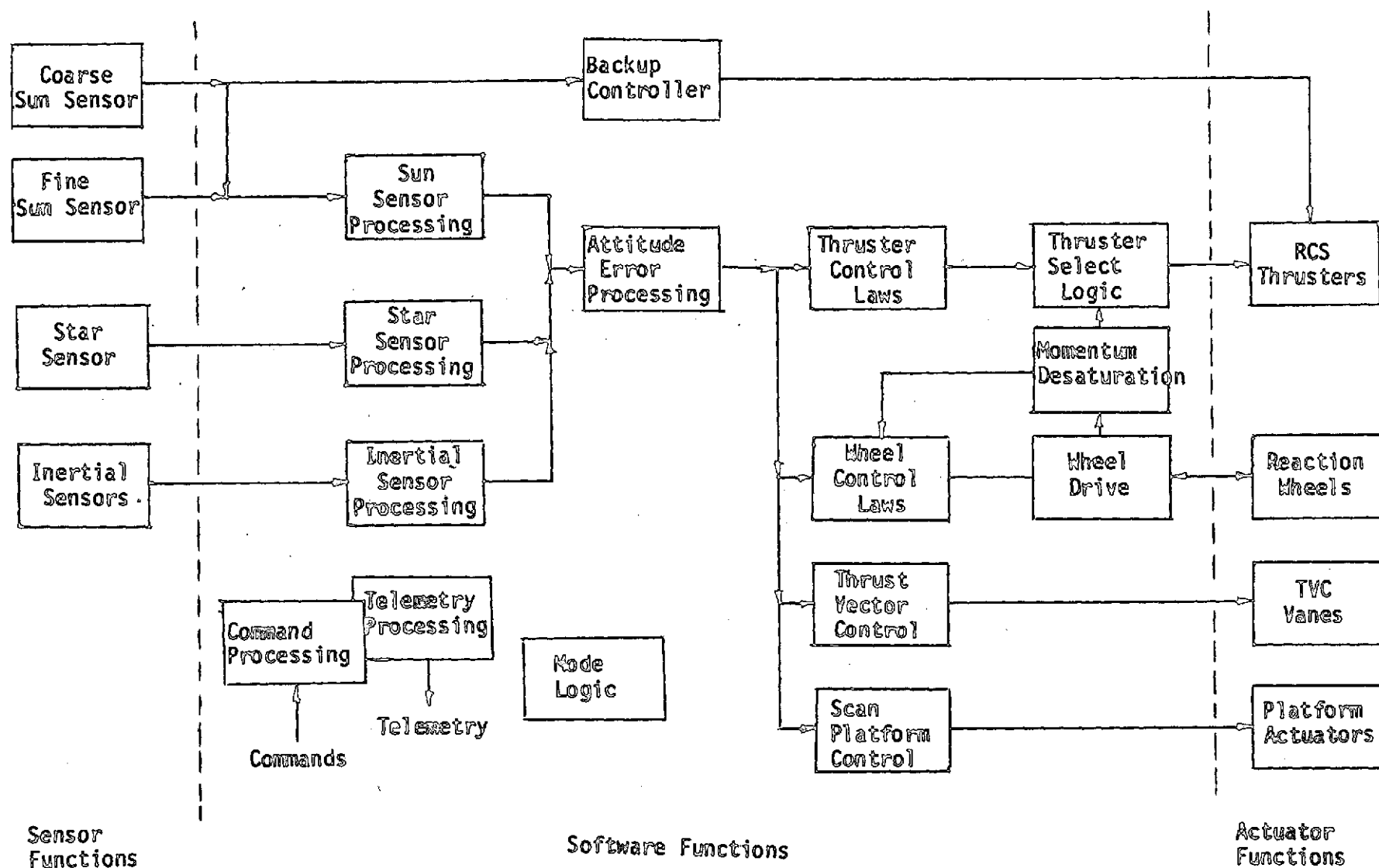


Figure 3-26

Control System Functional Block Diagram

must be performed within the processor), it is of particular concern if the function is one of a secondary nature or one which might be done as easily within special purpose electronics. Some of these type of functions have already been mentioned, e.g. sensor (high bandwidth) preprocessing, actuator servo logic, real-time clocking functions, etc. By relieving special burdens from the processor, the designer in general relieves potentially significant hardware constraints (e.g. speed) and software design complexity (e.g. asynchronous executive, very short minor cycle, etc.). Design guidelines frequently establish the processor minor cycle on the basis of the overall (e.g. primary) control loop bandwidth requirements. When sampling (bandwidth) of much greater frequency is required the related functions should be performed external to the processor unless significant reasons exist to do otherwise. Thus, the design presented here performs actuator servo loops (e.g. jet vanes, scan platform drive, wheel tach loop (if used)) and thruster pulse period clocking (e.g. thruster shut-off following turn-on with commanded ON-time) in associated special purpose (auxiliary) electronics. Sensor tracking loops (e.g. star sensor) or pulse count data collection (e.g. gyros, accelerometers) is also done within the specific peripheral device electronics. Analog IRU rate signal outputs have also been presumed.

The processor functions are briefly addressed, as these representative elements are used in the next section to characterize the computational requirements.

- Sun Sensor Processing - sun sensor outputs are input, scale factor and bias characteristics are incorporated and required (desired) prefiltering) of data is performed.
- Star Sensor Processing - logic is processed from which stars within the star sensor field-of-view are acquired/tracked by the sensor, star sensor outputs are input, scale factor and bias characteristics (and other compensation of systematic errors) is incorporated, and prefiltering of star sensor data is exercised.
- IRU Processing - gyro outputs are input and compensation made for input axis alignment, scale factor, and bias. The compensated gyro data is used to compute inertial attitude and rate. The IRU computation

of attitude (and other compensation parameters, if appropriate) is periodically updated from star sensor and sun sensor data. Accelerometer outputs are integrated as appropriate and compared to a commanded velocity change to provide signals for  $\Delta V$  thruster commands.

- Attitude Error Processing - the measured attitude relative to the attitude reference (e.g. sun, known star, inertial depending on mode) is combined with the commanded value (desired orientation) to derive attitude error signals (position and/or rate) for use by the control system. Spacecraft (large angle) maneuvers use stored commands via this error processing function to execute reorientation.
- Wheel Control Laws - use derived error signals within appropriate control algorithms to derive reaction wheel torque commands (in each control axis).
- Thrust Vector Control Laws - use derived error signals within control algorithms to derive TVC actuator vane deflection commands.
- Thruster Control Laws - derived error signals are utilized in control algorithms to command RCS thruster modulation "ON" time.
- Scan Platform Control - stored commands are processed to point the scan platform as a function of time.
- Wheel Distribution - for reaction wheel configurations in which the wheel axes are skewed with respect to the control axes, the geometric gain relation is computed and output/interface processing accomplished.
- Momentum Desaturation - The wheel speed is measured to determine stored momentum. The stored momentum is compared to threshold values above which wheel unloading using RCS thrusters is commanded. Thruster "on" time is determined.
- Thruster Select Logic - control axis and thruster "ON" commands are used within selection logic to enable/fire the appropriate thrusters. Output/interface processing with the RCS is accomplished.

### 3.2.2 Functional Redundancy

Control system functional redundancy, within the present fault-tolerant concept is defined to mean the accomplishment of the desired objective

(e.g. cruise control) via a means which is alternative to the primary approach. Functional redundancy can be achieved in one of several fashions, namely: alternate modes, different control laws, substitution of system components. Each will be addressed in turn. In general, it is desired to meet the performance requirements throughout the regime of functional redundancy. The implications of this latter point will be addressed subsequently.

As seen in Table 3-4, there exists flexibility within the control system design to utilize alternate modes during the various flight regimes. In fact, providing this flexibility of function was one of the driving factors in defining modes and associated mode logic. A key example is using either the Celestial Point Mode or Inertial Point Mode during cruise (and fly-by with appropriate consideration of the earth-spacecraft-target geometry). Thus, failures of complete redundant sensor systems (sun sensors, star sensors, or gyros) need not be catastrophic as there exists this functionally redundant capability. Failures of modes themselves can be provided with functional redundancy, albeit with some potential operational handicaps. For example, failure of the acquisition mode logic may be circumvented through use of the Inertial Point Mode. With orientation commanded for (roll axis) sun pointing, step rotation maneuvers about the sun line may be exercised until the star sensor locks on to the desired target star (at which time the Celestial Point Mode could be entered, if desired).

Another means of functional redundancy is obtained through use of alternate control laws. For example, if the reaction wheel control laws are flawed, use could be made of RCS or TVC control laws, as the form of the algorithms may be expected to be similar. Other means of functional redundancy include the flexibility inherent in choice of sensor and actuators for a given mode. Substitution of fine sun sensors for coarse sun sensors (or vice-versa) or of thrusters for reaction wheels (or vice-versa if momentum unloading capability remains) provides an enormous leverage capability. As a result of these approaches, it becomes clear that one need not want for elements of functional redundancy. However, the key question remains as to the effectiveness and/or cost implications associated with such techniques.

The first consideration is the desired autonomy to exercise functional redundancy. The complexity of software executive and mode logic functions to provide such capability weigh against other fault-tolerant design criteria for which simplicity and constrained behavior are virtues. Exercising functional redundancy also requires a system modularity introduced at a level which supports such wide operational flexibility. In a manner similar to the impact of autonomy, this level of modularity may be unrealistic when weighed against other dictates of hardware/software modularity (addressed in later sections).

Finally, there exists the real consideration of degraded performance and/or additional "cost" in the operating regime. This can take form with respect to a variety of interacting criteria including accuracy, rate of expendables, and power consumption. For example:

- o Substitution of RCS thrusters for reaction wheels will cost propellant (expendable) to maintain performance near the normal (reaction wheel) level of accuracy (or reduced accuracy to minimize loss of expendables).
- o Substitution of wheels for RCS or gyros for star sensor/sun sensor requires (potentially large) power increases - a key factor for nominally power constrained systems such as deep space missions using RTG's

In this same vein, the Backup Mode provides functional redundancy only in the sense of maintaining minimum orientation constraints. Such items include the coarse antenna pointing for communication, attitude for thermal constraints, etc. The key function, then, is preserving a process "external" to the fault-tolerant system structure to recover from catastrophic failures. The well-defined return path from Backup Mode provides a meaningful point from which to reset from a known quantity. It is noted, however, that an "external" backup mode is not essential to the fault-tolerant system design, but provides more in the way of a response to engineering judgement.

In conclusion, functional redundancy at a reasonable level (e.g. modes, some component substitution) which is relatively straight-forward as regards implementation appears well advised. Dependency on extensive functional redundancy as regards total system tradeoffs appears somewhat mission dependent and demands mission-to-mission reassessment. System-level

functional redundancy in lieu-of or in-combination-with other techniques (e.g. hardware component or software module replication) provides another factor/means for maximizing fault-tolerant performance potential while minimizing key criteria of size, weight, power, and cost.

### 3.2.3 Interface Definition

The primary control system interfaces include: the Reaction Control System (RCS) for operation of thrusters, isolation valves, heaters, and associated malfunction logic; the command and telemetry systems for command, telemetry, timing/sequencing functions, and mission-related data store; and the electrical power system (EPS) for essential and non-essential power allocation/distribution. Essential bus power is provided for failure mode operation. All other control system power is obtained from the non-essential bus. The implication on the electrical power system (e.g. providing essential and non-essential power buses) is not explicitly dealt with in the scope of the effort.

Control system elements for the RCS typically provide valve driver outputs, 28 volt coil enable switching, isolation valve power switching, and heater control (if required). Key other considerations are defining RCS related fail-safe and/or fault tolerant logic (see Section 3.2.4), and signal interface for valve selection/enable. The latter is important to the extent that it is desirable to put any "memory" (e.g. thruster ON-time) in a location external to the processor - I/O. For example, the processor - I/O interface would be a digital word in which the selected thrusters and ON-time are identified. RCS logic (within the control system electronics) would use the data to enable selected thruster firing and clock count-down of the commanded ON-time.

The control system command interface may actually result in a distributed interface structure. This arises since there is such a variety of functions which must potentially be performed. Processor memory data may utilize a DMA port or separate command channel, depending on speed and/or processor configuration. Control system commands may be expected to operate through a serial command bus with control system command decoding/processing under processor control. Pulse commands may operate through another I/O interface. Separate command structure also appears desirable for hard-core,

Reconfiguration Control Unit or other direct/override commands and/or power switching commands, and for operation in the backup mode function.

The telemetry interface is important to permit determination of status and performance on the ground. This is true even with the long data transmission time for deep planetary missions. Status data and engineering data which is not used for functional performance purposes or for fault detection (e.g. motor current, temperatures) is interfaced directly. Data available within the processing electronics may be manipulated into a variety of formats (e.g. depending on mode or certain hardware status) and output at the processor - I/O telemetry interface. The telemetry interface should be asynchronous with the processor - I/O, with self-developed sync pulse, read envelope, and data clock. The processor- I/O interface requires only serial inputs, while telemetry interface to other elements (e.g. sensors, actuators, RCU, etc.) will require analog, bi-level, and serial inputs.

Typical elements of control system command and telemetry lists are capsulated in Table 3-5.

TABLE 3-5. Typical Control System CMD/TLM List Elements

<u>Commands</u>	<u>Telemetry</u>
Mode Select/Override	Sensor Data
Configuration/Power Switch	Sun Presence
Processor Memory Load	Logic Status
Subcom Select	Enable Status
Heater On/Off	Configuration Status
Star Sensor Mode Control	Temperatures
Thruster Enable/Disable	Current (Voltage)
Isolation Valve Control	Fault Signals
	Wheel Speed
etc.	Gimbal Angles
	etc.

#### 3.2.4 Fail Safe Design

The most catastrophic effects to the spacecraft, as caused by failures in the control system, can occur only as a result of those failures that affect the system actuators. Such failures must be prevented from causing excessive spacecraft rates, or from depleting propellant supplies or excessively draining the power system.

Fail safe is defined to be the property that is designed into the system so that single failures, in any equipment of the system, will not cause loss of the spacecraft.

It is assumed that faults in the sensors will be sensed by the processor and the influence of these faults will not be allowed to improperly affect the actuators. Similarly, many faults in the processor or the actuators and their driving electronics may be similarly treated. However, some processor faults and many faults in the actuators and their electronics could cause the actuators to mis-function to the detriment of the spacecraft. The prevention of these occurrences is the subject of this section. The prevention of, and action during and following power faults will be covered in Section 3.2.5.

The two actuator types of principal concern are the thrusters and their control electronics and the reaction wheels and their control electronics. These are the only two means by which momentum can be transferred into the spacecraft, causing the excessive and potentially damaging rates. It was shown in Section 2.2.2 that hard-over failures of greater than 3 to 10 seconds should be avoided.

In the following sections, the fail safe design implications on these two critical actuator types will be considered.

##### 3.2.4.1 Thruster Control

The mechanization design of the control system thruster drivers and control logic is a special case because of the implications of failure modes that could place the spacecraft in undesirable conditions of attitude rates and the possible loss of thruster propellants.

The principal design criteria for the thruster control electronics are:

- No single point failure that can turn on a thruster continuously or repetitively in such a manner as to cause high attitude rates or significant loss of thruster propellant.
- Independent control of up to 16 thrusters.
- Provide both a pulse mode and continuous firing for each thruster, independently. Pulse mode of 10 milliseconds minimum to allow minimum limit cycle propellant consumption.
- Redundant I/O interface.
- Ability to utilize each thruster for any (possible) axis of control in any mode.

It is hypothesized that the thrusters will contain dual drive coils where either coil can operate the thruster. The degree of protection against failure modes obviously affects circuit complexity and therefore must be evaluated against the cost in terms of part count, size, weight and other design criteria. The following discusses several configurations with different levels of thruster failure mode protection and operational flexibility.

Figure 3-27 shows a basic Valve Drive Electronics mechanization that meets the specified design criteria.

Valve command data is inputted from the IOU in serial form via one of the redundant data bus channels. For the case shown, the data appears as a 16 bit word in a particular holding register. The word format is as follows:

Word A

Bit 0	- AI Pulse
Bit 1	- AI Continuous
Bit 2	+ AI Pulse
Bit 3	+ AI Continuous

Figure 3-27 Valve Drive Electronics

Word A (cont'd)

Bit 14 - + A4 Pulse  
Bit 15 - + A4 Continuous

Word B

Bit 0 - + A5 Pulse  
Bit 1 - - A5 Continuous

Bit 14 - + A8 Pulse  
Bit 15 - + A8 Continuous

The processor can then command either a pulse "on" or continuous "on" for each of the valves. To provide the fail safe feature of preventing continuous on failures of a valve, a power gate is provided in the 28 volt power bus line. This power gate is turned on by a retriggerable one shot that is in turn "fired" by the receipt of a valve command from the processor. The time period of the power gate one-shot is adjusted to be approximately 1.5 times the period of a complete major cycle of the computer. If the computer does not recommand the continuous valve "on" condition each cycle of the computation, the power gate one-shot will expire and the valve will shut off. This mechanization essentially protects against any single point failure in the valve drive electronics causing a continuous "on" condition of a thruster. There are undoubtedly processor failure modes that can command "on" a valve continuously, however these must be and can be protected against by certain processor software techniques as described in Section 3-5.

With the arrangement shown, logic is also provided to "off modulate" a thruster pair to maintain attitude control about any given axis in the event that both thrusters are turned on continuously, such as in the case of performing a  $\Delta V$  maneuver. For such a case, a pulse "on" command to any given thruster will be logically directed to turn "off"

the thruster of opposite sign, thus generating the desired direction of spacecraft attitude torque.

As pointed out previously, there are many levels of redundancy that can be applied to the thruster control electronics. One other possible arrangement is similar to Figure 3-27 except that each thruster pair would have its own power gate and retriggerable one shot that is controlled by a separate bilevel command input from the processor. This input can be generated by a different software routine within the processor and could be sent to the Valve Drive Electronics via a separate data bus. Such a scheme would allow for many individual failures within the Valve Drive Electronics without affecting the control and operation of the balance of the thrusters. As in the previous case, the retriggerable one-shot power gate prevents any single failure in the valve drivers, power gate or data buses from causing a continuous valve "on" failure.

Another possible arrangement for thruster control is shown in Figure 3-28. In this case, the drivers are of a type similar to the National Semiconductor NH0008 power switch operating from a logic input and sourcing load current from a 28 volt supply. The drivers are indicated as an emitter follower transistor, driven by a logic gate.

The output of two or more of the drivers can be placed in series to provide redundancy protection against shorting of the driver or failures in the input logic that results in a continuous "on" type command.

As shown in Figure 3-28, a series-parallel quad arrangement can be configured where the A and B inputs are driven from continuous logic inputs and the  $C_a$  and  $C_b$  inputs are driven by pulsed logic inputs. If the processor has independent control of each input, it can pulse fire a thruster by operation of B and  $C_a$  or A and  $C_b$ . Continuous operation is achieved by operation of A and B. As in the previous cases, to achieve a thruster turn on, at least two valve drivers must be turned on simultaneously. No single part failures or processor command can turn on a valve.

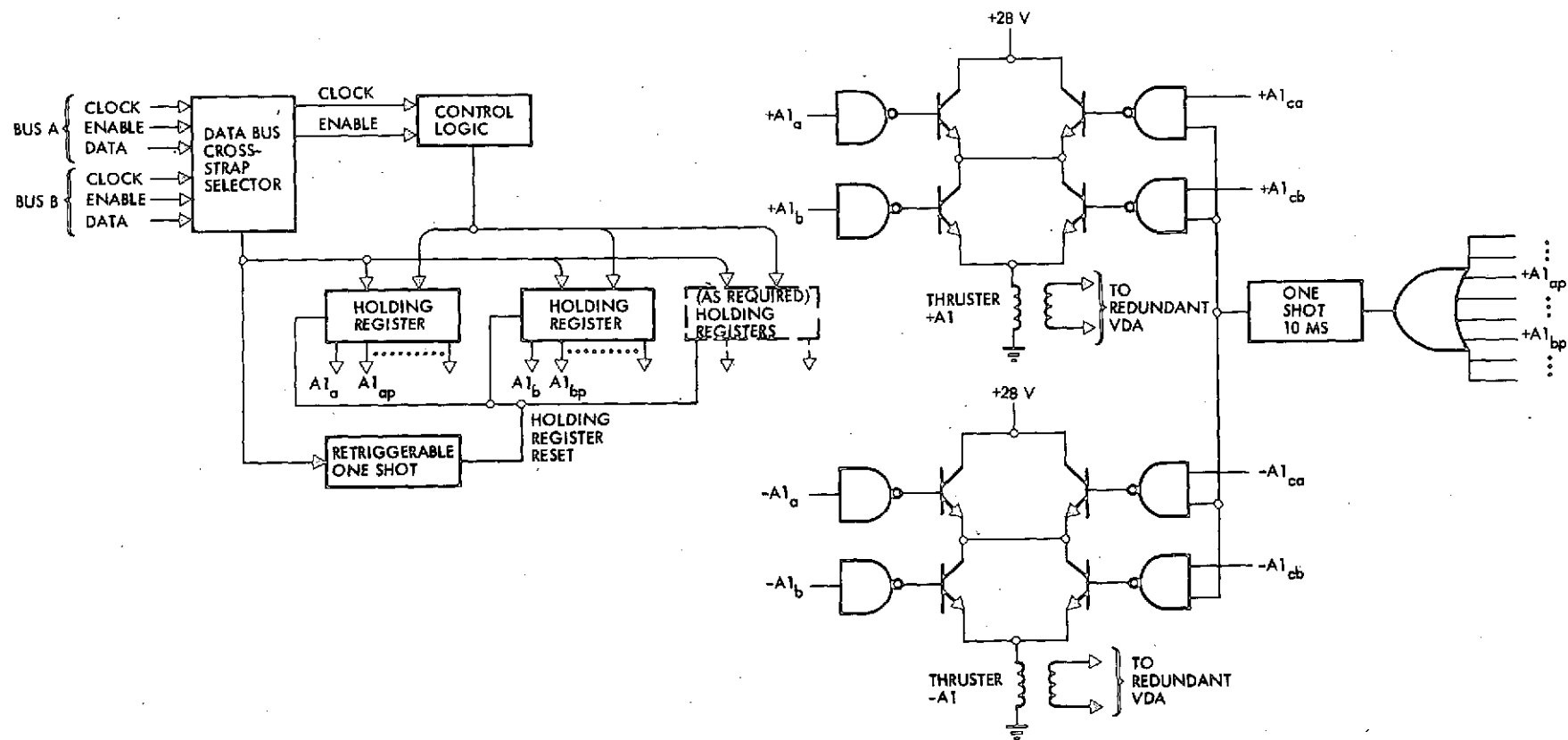


Figure 3-28 Valve Drive Electronics

Note that for this case there are two paths for the processor to achieve the pulse mode of control. If any of the pulse drivers fail in the open mode then the processor can control through the internally redundant path. Also note that each redundant path is testable by proper programming of the processor.

For the case of Figure 3-28, the "off modulation" control of the thrusters (not shown) can be done similarly to that of Figure 3-27. A retriggeable one-shot is also provided to reset and disable all command outputs of the holding registers if a new valve command is not received within 1-1/2 periods of a processor major cycle time. This will prevent a continuous thruster "on" command in the event of a processor failure or data bus failure after the valve has been commanded on in the continuous mode.

Note that it is important to place the A and B sections of the series redundant valve drivers on different command holding registers so that no single device failure can fail both sections on together.

As shown and discussed in the several examples, the key features of implementing a fail safe valve driver is to provide a series redundant path for driving the valve coil and then require that one path be continuously updated by command from the processor to remain in the "on" condition. Additional, similarly configured, parallel paths may then be added to provide more redundancy or alternate modes of operation (such as the pulse mode). At least one additional parallel path to turn on the thruster is required to provide the redundant control in the event of an open in the series path. This is generally best provided by a completely redundant standby Valve Drive Electronics operating into the redundant valve coils. The series redundant paths should be completely separated within the drive electronics from the drivers back to the retriggeable one-shot to prevent continuous "on" type failures. Command implementation to the Valve Drive electronics does not appear to be critical and can be handled in a number of ways since the failure mode protection is applied "downstream". The "off modulation" control of the valves is logically simple and does not significantly affect the design mechanization regarding protection against catastrophic failure modes.

#### 3.2.4.2 Reaction Wheel Control

The mechanization of drive electronics for a spacecraft reaction wheel has, like the thruster drive electronics, some special requirements over that of most of the control system peripheral electronics.

Because the reaction wheel can store considerable energy, it potentially can damage some other assembly or the spacecraft structure in such a way that would represent a single-point failure mode. Most efficiently designed reaction wheels are capable of overspeed type failures due to certain failure modes in the drive electronics.

Generally speaking, most reaction wheels contain either a brushless d.c. or 2 phase a.c. drive motor. The brushless d.c. motor is more efficient than the a. c. motor but it requires considerably more complex drive electronics which is particularly undesirable in the long-life missions. Unless very large reaction wheels are required, the 2 phase a.c. drive motor is the best choice.

Figure 3-29 shows a typical mechanization of a reaction wheel drive electronics using a 2 phase a. c. motor. In order to maintain the status of spacecraft attitude and rates if there is an outage of the processor, the reaction wheel control is mechanized as a speed control loop that is commanded by the processor. (This also reduces the bandwidth requirements on the processor.) If the processor fails to output new data, the reaction wheel speed will hold the last previous value, thus minimizing control torques applied to the vehicle.

The speed command data is received via one of the redundant data buses and stored in a holding register. The command is in the form of a binary word defining either the period between reaction wheel tachometer pulses or reaction wheel rate, depending on the mode of operation. If the wheel is operated with a speed bias to impart momentum to the spacecraft, then the more simple arrangement of counting clock pulses between tachometer pulses will suffice.

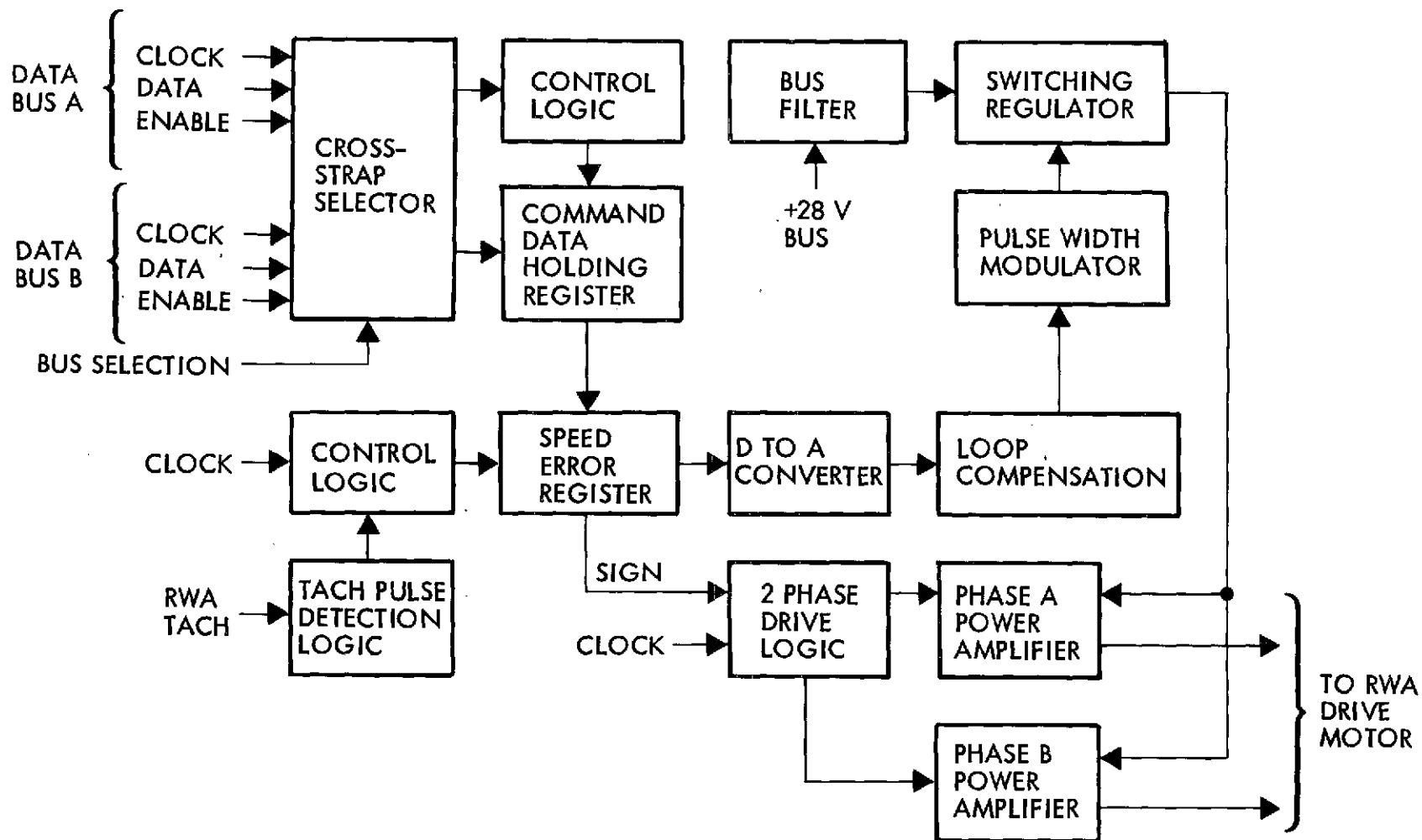


Figure 3- 29 Reaction Wheel Electronics

If the wheel is operated unbiased, where it can go to zero speed, then a rate command and sensing implementation must be employed. In either event, the speed error is detected as the residual contents in a digital register and converted to an analog voltage that can be appropriately frequency-compensated to control loop stability. The compensated error signal is then pulse-width modulated as the input to a conventional switching regulator. Depending on system considerations involving loop gain and bandwidth requirements, the compensation may not be required and the D to A converter and compensation can be bypassed by operating the digital speed error directly into a digitized pulse width modulator.

The output of the switching regulator is a variable amplitude d. c. voltage that is supplied to the 2 phase a. c. power amplifiers driving the reaction wheel motor windings. The power amplifiers are operated as square wave switches to conserve power. The direction of motor torque is controlled by reversing the phase of one of the motor drive outputs as determined by the sign logic derived from the error register.

With a single polarity power supply, as in the case of a normal 28 volt bus, the implementation of the drive electronics is greatly simplified if center tapped windings are employed in the reaction wheel drive motor. The power amplifiers are then transformerless, thus saving considerable weight in the electronics at the expense of a slight weight increase in the reaction wheel motor.

Another promising approach for single polarity supply and transformerless a.c. drive is shown in Figure 3-30.

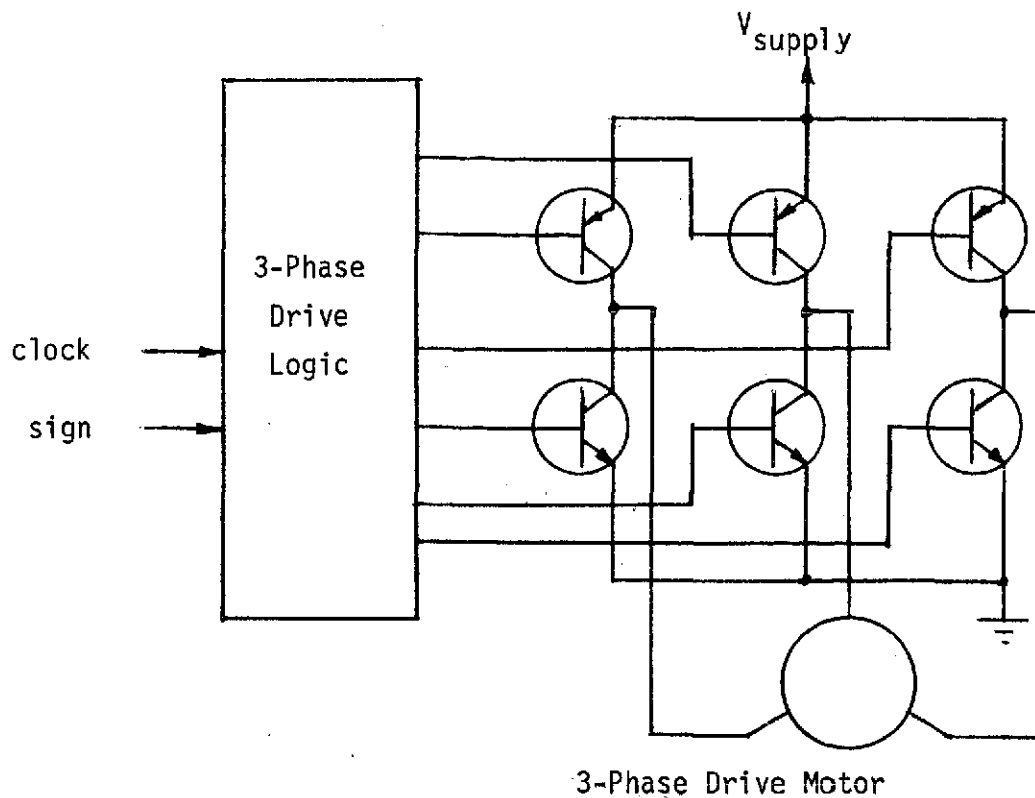


Figure 3-30  
Three Phase Motor Drive

In this case the design of the 3-phase motor is more efficient than the 2-phase center tapped motor and requirements for transformers are still eliminated. Two additional power transistors are required, but they can be somewhat smaller in rating for the 2-phase driver.

Wheel speed run away is controlled by picking the number of poles and operating frequency such that the synchronous speed of the motor is less than the design limit of the wheel. The electronics must then be designed to avoid any failure modes that could simultaneously increase the wheel drive frequency and apply full torque. This is not a severe design constraint as these circuits are normally quite separate and independent. Consideration must be given in the design of the power amplifiers to handle the large reactive components of motor current.

Another important consideration in the design of reaction wheel drive electronics is the design of the bus input filters and proper EMI protection. In cases where large motors are employed, the a.c. currents generated by the switching regulator and square wave drivers can potentially cause a significant EMI problem. Where fast reconfiguration of a fault-tolerant system is desired, the design of such filters must consider transient characteristics as well as frequency attenuation. Some form of active current limiting for start-up will undoubtedly be required.

### 3.2.5 Power Interruption

Power interruptions can occur in the system, as caused by the control system itself, by other systems, or by the power system. These interruptions can be brief (transient) glitches caused by EMI, switching transients or "glitches" of one sort or another; or they may last a relatively long time (several seconds, up to an indefinitely long time).

It has been assumed throughout this report that the power system supplies not only the primary power (normally a nominal 28 v d.c.), but the secondary voltages required by the control system. This allows the use of centralized power converters (and regulators) for the control, command, telemetry, and other spacecraft electronic systems. These converters must, of course, be redundant to provide the necessary reliability and lack of a single point of failure.

The converters should be designed so that their redundancy switchover is automatic and autonomous. This switchover normally occurs if any output displays a lack of voltage or a greatly excessive current consumption. The switchover should be fast enough that only a transient loss of voltages will be experienced by the control system.

The control system blocks must be designed so that their faults tend to open the power lines rather than short them (if there is any effect). This is particularly important on the primary power lines where fusing (or the equivalent) may be needed.

The primary power system of the spacecraft (particularly when using RTG's) is very reliable and usually redundant. The power delivered should be continuous. Note that this is not the case when using solar arrays, where eclipses may occur or the array may be mis-pointed. Even so, transients may still occur for a variety of reasons, including:

- Source switching
- Load switching
- Redundancy switching
- EMI
- Fault clearing (Fusing)

These primary transients, if of sufficient amplitude/duration may pass into the secondary power voltages. These voltages are also subject to the same transient influences.

It is easy to see, therefore, how some or all of the voltages supplied to the control system may exhibit transients. These transients should normally be of hundred's of milliseconds or less duration, but may be of plus or minus the voltages themselves in amplitude.

Since the circuits can not operate without voltage, the problems occurring when the voltage(s) disappear must be examined. Note that long time voltage outages are extremely unlikely in a power system using redundant RTG's and redundant autonomously-switched converters. None-the-less, the system should be designed for restart following such an occurrence. This, then, will also cover start-up upon initial power turn-on.

If the transient is short enough, then the capacitance of the power filters will maintain the voltages high enough to allow the circuits to continue working. Beyond this, where circuits may be upset, protection of one sort or another must be provided.

This protection must be for the actuator-type peripherals and for the processor. All other peripherals are assumed to be designed so that they are not harmed by the transients and, at worst, simply provide false outputs.

The actuators of concern are the thrusters and the reaction wheels. The thruster drive circuitry should be designed so that in the absence of any combination of voltages the thrusters do not fire (unless they should have been). The registers receiving the firing commands from the IOU should also be configured so as to greatly reduce the possibility of thruster firings (beyond a short pulse) when the voltages come on again and the latches, flip-flops, etc. assume presumably arbitrary states. The processor should soon correct any improper states, in any case.

The reaction wheel drive circuitry should be designed so that in the absence of any combination of voltages the wheel will not overspeed (or the drivers will not supply d.c. to an a.c. motor). The processor will quickly correct any bad speed commands resulting from power turn on.

In the processor, we must hypothesize a power glitch detector which responds to detect loss of voltages slightly faster than the decay on the load side of the filter capacitors. It should detect voltage resumption slightly slower than the rise on the load side of the capacitors. This detector can be used for a variety of functions in the processor. Examples are:

- Clock stoppage. When a glitch occurs, the clocks should be immediately stopped, which will prevent fault propagation. Following the glitch, a slight delay before clock restart should occur. The clock oscillator must be self-starting at the correct frequency and the phase dividers must start in the proper sequence.

- Prevention of erroneous write. When a glitch occurs, a memory write in process should be completed (easy if done in parallel) and further writes should be inhibited. This can be done if the memories have a slightly longer voltage retention (or are non-volatile) and the clock is stopped.
- BITE generation. This can provide a power interrupt signal (when the voltages resume), causing program roll-back, start-up, etc.

As noted elsewhere, non-volatile memories are very desirable, at least for short power outages. It is also advantageous if the redundancy status and mode status are non-volatile. Then, upon the restart signal, the processor can clear itself of the effects of the glitch and bootstrap itself back into operation without delays for reconfiguration or finding the right mode, etc. The transients will then have a minimal effect on the system operation.

Start-up upon power turn-on or resumption should cause the program to restart at the beginning of the executive. This software should be configured to not let the peripherals do "bad" things, while destroyed memory information is restored, proper configurations are established, and the normal operation is entered.

### 3.3 Hardware/Software Requirements

Characterization and allocation of requirements to control system hardware and software elements is key to the design process which translates the "system-design" into physical reality. Although the control system design has been developed only to a preliminary functional level (i.e. primarily quantitative as opposed to qualitative characteristics), important hardware/software requirements may be extracted and supported on the basis of prior design experience. This section treats the processor/electronics functional requirements, with emphasis on the processor - I/O as determined by a more explicit consideration of software and interfacing data flow.

#### 3.3.1 Processor/Electronics Functional Requirements

This section provides for characterization of functions required to implement system-level design as allocations to modular hardware elements. The hardware is divided into three groups, namely: processor - including ACU, memories, I/O interface, and special I/O electronics; auxiliary - controller-related special purpose electronics (e.g. backup electronics); and peripheral (e.g. sensors, actuators) (with associated electronics).

Processor - I/O functional requirements are divided into a number of areas, each of which is enumerated and treated separately.

Data Processing - the processor - I/O electronics is to perform all subsystem data processing. This data processing is characterized by the software functions which are to be performed. These are treated in some detail in the next section. However, it is reasonable to characterize these software functions within the following context:

- o Modular design with well defined inputs, processing, outputs
- o Straight-forward, generally simple algorithms/data handling
- o Some matrix operations, some logical operations
- o Minimal data manipulation
- o Minimum instruction set is adequate, i.e., no need for floating point, bit/byte manipulation, etc.

Speed - the processor speed should be such that the application software utilizes less than a 50% duty cycle. This leaves considerable time for program growth and executive and fault-tolerant related operations (e.g. retry of faulted tasks, etc.) A moderate processor speed (on the order

of 100-150 kops appears appropriate.

Accuracy - the processor word length (single or double-precision as appropriate) must be such that control system performance errors attributable to computation are negligible. Accuracy to fractions of a degree are reasonably achieved with 16 bits; accuracy to the  $\widehat{\text{sec}}$ -level requires the equivalent of 24 bits.

Memory - the processor memory size should be configured such that a 100% growth of the application programs and related data storage/scratchpad may be accommodated. In general, a relatively small processor memory will be required. Program memory of 4-8K and scratchpad of 1-2K appears generous for almost any application. Since memory size may directly affect reliability, it is important to have memory available in small increments (e.g. 1 or 2K for program, 256-512 word for scratchpad) in order to provide an optimum configuration. If Read-Only-Memory (ROM) is used for program storage, it is essential that the read-write (scratch) Random Access Memory (RAM) be capable of storing/executive program modifications (e.g. post-launch).

Input-Output - The processor - I/O must provide the interfacing of all data/signals to be acted upon by the software. This includes data interface from the auxiliary electronics, peripherals, telemetry and command, and other spacecraft systems. The data is in the form of serial digital, analog (input with A/D conversion to 12 bits), bi-levels, and command and telemetry data which must be asynchronously buffered.

Auxiliary Electronics functional requirements are most easily definable as related to specific controller-related functions. Modularity of these individual functions, within the redundancy approach for optimization, must be considered separately. The following is representative, based on the control-system design developed in the previous sections (see also Peripheral Electronics as relates to self-test electronics and interfaces).

Backup Electronics - Provide electronics for system malfunction indication and electronics to implement control laws using the sun sensor data to establish and maintain a coarse pointing of a defined spacecraft axis toward the sun.

Reaction Wheel Electronics - Provide electronics for reaction wheel drive and wheel speed (tachometer) readout processing. The electronics must accept serial digital data input from the processor - I/O. Digital output data is preferable.

Valve Drive Electronics - provide electronics for valve enable, valve drivers, isolation valve enable/switching, heaters (if required), and implementation of fail-safe operation. The electronics should be designed to accept serial digital data input from the processor - I/O. Digital output data is preferable (if required).

Jet Vane Electronics - provide electronics for jet vane servo in response to angle commands, vane actuator drive electronics, and angle readout processing. Digital input/output interface is as above.

Scan Platform Electronics - provide electronics for scan platform gimbal servo, actuator drive, and angle readout processing. (I/O guidelines same as previous.)

The distinction between peripherals and auxiliary is not a particularly strong one. However, in the present context the peripherals include primarily the sensors and actuators themselves. The key functional requirements on peripherals apply also to the auxiliary electronics, namely:

- Digital data interface is preferable
- Peripheral interfaces should be buffered, non-synchronous, and "ready"
- Peripheral performance monitoring requires only primary (applications) data. Some built-in-test (self-test) in the form of logic output may be useful but is not required.

### 3.3.2 Software Computational Requirements

The control system software is used to process sensor data, develop attitude error, compute control laws, and exercise control authority. The software is organized in terms of individual modules defined to handle specific functional requirements. The actual linking of these modules is under control of the executive software. The software functions were identified earlier in Figure 3-26. The software module design has been carried

to the level of preliminary equation definition and flow, and preliminary generic coding of the software was conducted to determine computational requirements.

An example software module design for the gyro processing module is included to summarize the requirements development process. The overall computational requirements for the control system software are summarized in Table 3-6 . A total program memory of 4K and scratchpad of 1K provides for almost 100% growth, and an average short instruction speed of 8  $\mu$ sec provides for less than 50% duty cycle for the identical applications programs. The sizing requirements of Table 3-6 are properly conservative and, in some cases, significantly constraining (e.g. control sample period of 100 ms, double precision processing for IRU) such that much computational margin would be anticipated.

TABLE 3-6  
COMPUTATIONAL REQUIREMENTS (TYPICAL - BOUNDED)

APPLICATION MODULE	PROGRAM	DATA/SCRATCH	INST/PASS	ITERATION/SEC (MAX)	KOPS (MAX)
Initialization/Mode Logic	250	50	120	10	1.2
Sun Sensor Processing	20	6	50	10	0.5
Star Sensor Processing	150	30	450	10	4.5
IRU Processing*	510	146	3200	10	32.0
IRU Update	200	50	1700	1/30	≤ 0.1
Attitude Error Processing	115	31	1200	10	12.0
Control Laws	200	40	500	10	5.0
Momentum Desaturation	80	26	380	1	0.4
Wheel Distribution	50	16	230	10	2.3
Thruster Select	20	8	70	10	0.7
Scan Platform Control	40	10	100	10	1.0
CMD Processing	20	--	30	10	0.3
TLM Processing	100	10	400	10	4.0
System Test	200	30	--	--	--
Utility	300	--	--	--	--
TOTALS	2255	452			64

\*Performed in double precision

## DESIGN EXAMPLE

### IRU Reference Module

#### Functional Requirements

This software module provides the functions associated with maintaining an inertial attitude reference in conjunction with a configuration of strap-down rate integrating gyros. The gyro outputs are processed to compensate for known misalignments, scale factor uncertainties and bias. Rate and attitude of a known reference frame, nominally fixed with respect to the gyro configuration, is derived with respect to a known inertial reference frame, e.g., Earth-Centered-Inertial (ECI). Updates to the attitude and gyro compensation, periodically available from an external source, are incorporated as available.

#### Computational Functions

The computational functions are summarized as follows:

- Process gyro output data and provide transformation to defined orthogonal spacecraft reference frame.
- Compensate for gyro error sources (e.g., drift bias, input axis alignment, and scale factor)
- Maintain precise angular rate reference
- Maintain precise short-term inertial attitude reference
- Compute elements of direction cosine matrix relating reference frame attitude to ECI.

#### Operating Modes

There are two basic operating modes for operation of the gyro reference software. They are:

- Free. The gyro outputs are periodically processed to maintain current attitude and rate reference.
- Align. This mode is used for initializing and/or updating the attitude and/or gyro compensation from an external source.

### Input/Output

- Input. The inputs which may be utilized by the gyro reference software are summarized.

Gyro Data: Each gyro output is a digital word whose scaled value represents the incremental angular rotation measured about the gyro input axis.

Mode Flag: A mode flag indicating when update information is available.

Gyro Parameters: Input axis alignment, scale factor, and bias.

Attitude Update: Attitude variables provided for update.

- Output. The outputs to be provided by the gyro reference software are summarized.

Kinematic Parameters: The kinematic parameters are available for use in other software functions

Rate: Angular rate components in spacecraft axes.

Direction Cosine Matrix: The direction cosine matrix relating spacecraft reference axes to ECI reference axes.

### Functional Description

The output of each of  $N_g$  gyros is scaled to yield the measured rate, and the resultant rate vector is transformed to a known fixed orthogonal reference frame. The "geometry matrix" used for the transformation incorporates terms which account for "known" individual gyro input axis alignment and scale factor. The gyro bias compensation is added to complete the rate estimate. The equation for estimated rate takes the form

$$\hat{\omega} = \hat{G} \tilde{\omega} - \hat{b}$$

where

$\hat{\omega}$  = estimate of rate in fixed coordinate frame,  $3 \times 1$

$\tilde{\omega}$  = measured rate along each of  $N_g$  gyro input axes,  $N_g \times 1$

$\hat{b}$  = gyro bias compensation,  $3 \times 1$

$\hat{G}$  = "geometry matrix",  $3 \times N_g$  (determined through preflight measurement and/or flight calibration).

Euler Symmetric Parameters are employed as the kinematic variables for computation of vehicle attitude. This choice, as opposed to selection of direction cosines, was based upon the fact that a four, rather than nine, parameter system of equations is involved; and also, the periodic re-normalization that must be performed to combat computer roundoff error is much simpler. As is well known,

$$\dot{\bar{\rho}} = \frac{1}{2} \Omega \bar{\rho}$$

where

$$\bar{\rho} = \begin{bmatrix} \xi \\ \eta \\ \zeta \\ X \end{bmatrix} \quad \Omega = \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & \omega_2 & -\omega_3 & 0 \end{bmatrix}$$

Selecting the integration period sufficiently small,  $\Omega$  can be assumed constant over the integration interval,  $\Delta T$ , and

$$\rho_{k+1} = \exp(\Omega_k \Delta T/2) \rho_k$$

where  $\Omega_k$  represents the constant matrix assumed on the interval  $[t_k, t_{k+1}]$ . It is possible to obtain a simple closed form expression for  $\exp(\Omega_k \Delta T/2)$  which tends to inhibit the truncation error that would normally exist in the power series representation of the exponential. This is given by

$$\exp(\Omega \Delta T/2) = (\cos b \Delta T) I + \frac{\sin b \Delta T}{2b} \Omega$$

where

$$b = \frac{1}{2} \left( \sum_{i=1}^3 \omega_i^2 \right)^{1/2}$$

The kinematic variables are used to derive the direction cosine matrix which relates the spacecraft axes to ECI. The matrix A has the form

$$A = \begin{bmatrix} p_1^2 - p_2^2 - p_3^2 + p_4^2 & 2(p_1 p_2 + p_3 p_4) & 2(p_1 p_3 - p_2 p_4) \\ 2(p_1 p_2 - p_3 p_4) & -p_1^2 + p_2^2 - p_3^2 + p_4^2 & 2(p_2 p_3 + p_1 p_4) \\ 2(p_1 p_3 + p_2 p_4) & 2(p_2 p_3 - p_1 p_4) & -p_1^2 - p_2^2 + p_3^2 + p_4^2 \end{bmatrix}$$

$$= [a_{ij}]; i, j = 1, 2, 3$$

### Design Requirements

#### Method and Flow

The algorithms and flow chart are shown in Figure 3-31.

#### Computation Requirements (Double Precision)

#### Storage Memory (Including Data Base)

	<u>Program</u>	<u>Scratch</u>
Gyro Processing	47	24
Attitude Algorithm	201	82
Direction Cosine Matrix	<u>262</u>	<u>40</u>
TOTAL	510	146

#### Execution (operations, including subroutines)

Gyro Processing	535A + 36 M + 3D
Attitude Algorithm	1334A + 117M + 11D
Direction Cosine Matrix	232A + 30 M

#### Subroutine Usage (10A/Subroutine call included above)

Gyro Processing	1 Matrix-vector mult. $[3 \times N_g] [N_g \times 1]$
Attitude Algorithm	1 sin/cos
	2 square root
	1 Matrix-vector mult. $[4 \times 4] [4 \times 1]$

Direction Cosine Matrix

Data Base                      Geometry Matrix  $[3 \times N_g]$

#### Usage

Figure 3-32 indicates the effects of roundoff error due to computer word length on computational accuracy. The resultant attitude error standard

Input and Scale  
Raw Gyro Data:

$$\tilde{\omega}_m^i = \frac{K^i N^i}{T}$$

Compensate for Alignment,  
Scale Factor, and Bias:

$$\hat{\omega} = G\tilde{\omega} - \hat{b}_g$$

$$\begin{aligned} \theta_0 &= T (\sum \hat{\omega}_i^2)^{1/2} & \theta_i &= \hat{\omega}_i T \\ S_0 &= \frac{\sin(\theta_0/2)}{\theta_0} & C_0 &= \cos(\theta_0/2) \end{aligned}$$

Form Transition Matrix:

$$R_K = \begin{bmatrix} C_0 & \theta_3 S_0 & -\theta_2 S_0 & \theta_1 S_0 \\ -\theta_3 S_0 & C_0 & \theta_1 S_0 & \theta_2 S_0 \\ \theta_2 S_0 & -\theta_1 S_0 & C_0 & \theta_3 S_0 \\ -\theta_1 S_0 & -\theta_2 S_0 & -\theta_3 S_0 & C_0 \end{bmatrix}$$

Propagate and Normalize  
Attitude Parameters:

$$\begin{aligned} \hat{p}_{k+1} &= R_K \hat{p}_k \\ \hat{p}_i &= \hat{p}_i / ||\hat{p}|| \end{aligned}$$

Compute Direction  
Cosine Matrix:

$$A = \begin{bmatrix} \rho_1^2 - \rho_2^2 - \rho_3^2 + \rho_4^2 & 2(\rho_1 \rho_2 + \rho_3 \rho_4) & 2(\rho_1 \rho_3 - \rho_2 \rho_4) \\ 2(\rho_1 \rho_2 - \rho_3 \rho_4) & -\rho_1^2 + \rho_2^2 - \rho_3^2 + \rho_4^2 & 2(\rho_2 \rho_3 + \rho_1 \rho_4) \\ 2(\rho_1 \rho_3 + \rho_2 \rho_4) & 2(\rho_2 \rho_3 - \rho_1 \rho_4) & -\rho_1^2 - \rho_2^2 + \rho_3^2 + \rho_4^2 \end{bmatrix}$$

Figure 3-31. IRU Reference Flow Diagram

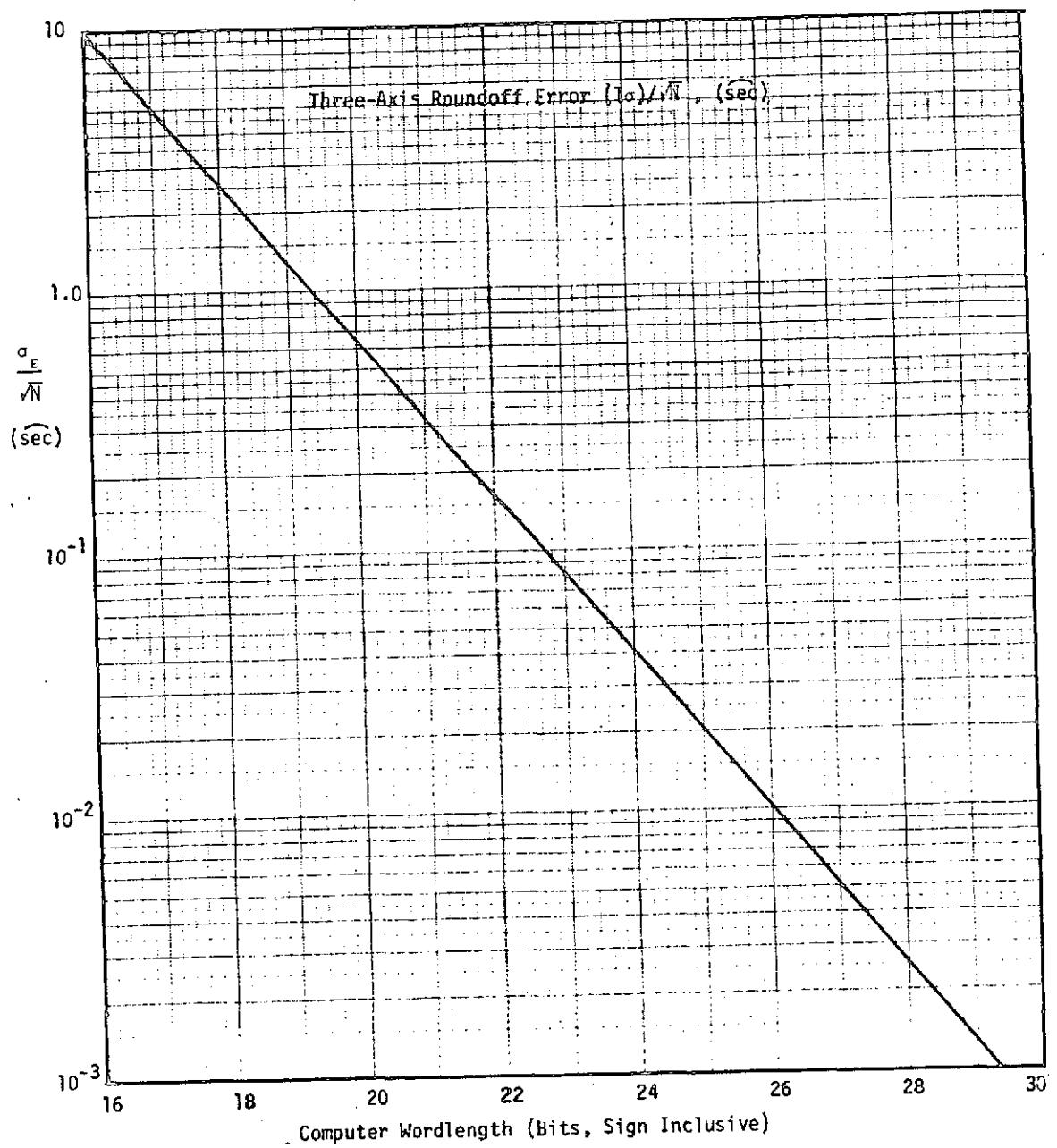


Figure 3-32 Roundoff Error Effects

deviation,  $\sigma_e$ , is a function of the number of iterations, N, over the time period for which the error must be bounded. Figure 3-33 indicates the computational error from drift effects due to commutation errors as a function of rates and accelerations. The results of this (or similar data for mission-related values of rate/acceleration) provide direction for selection of the iteration period.

### 3.3.3 Input/Output Requirements

The control system I/O requirements have been assessed through consideration of the use of multiple and redundant sensors and actuators. The requirements are listed in Table 3-7, where moderate analog, serial digital, and bi-level data interfaces are identified as well as telemetry and command.

This data is summarized for the I/O requirements (of an IOU) in Table 3-8. This tends to be the maximum requirement for an IOU for an interplanetary mission such as was postulated.

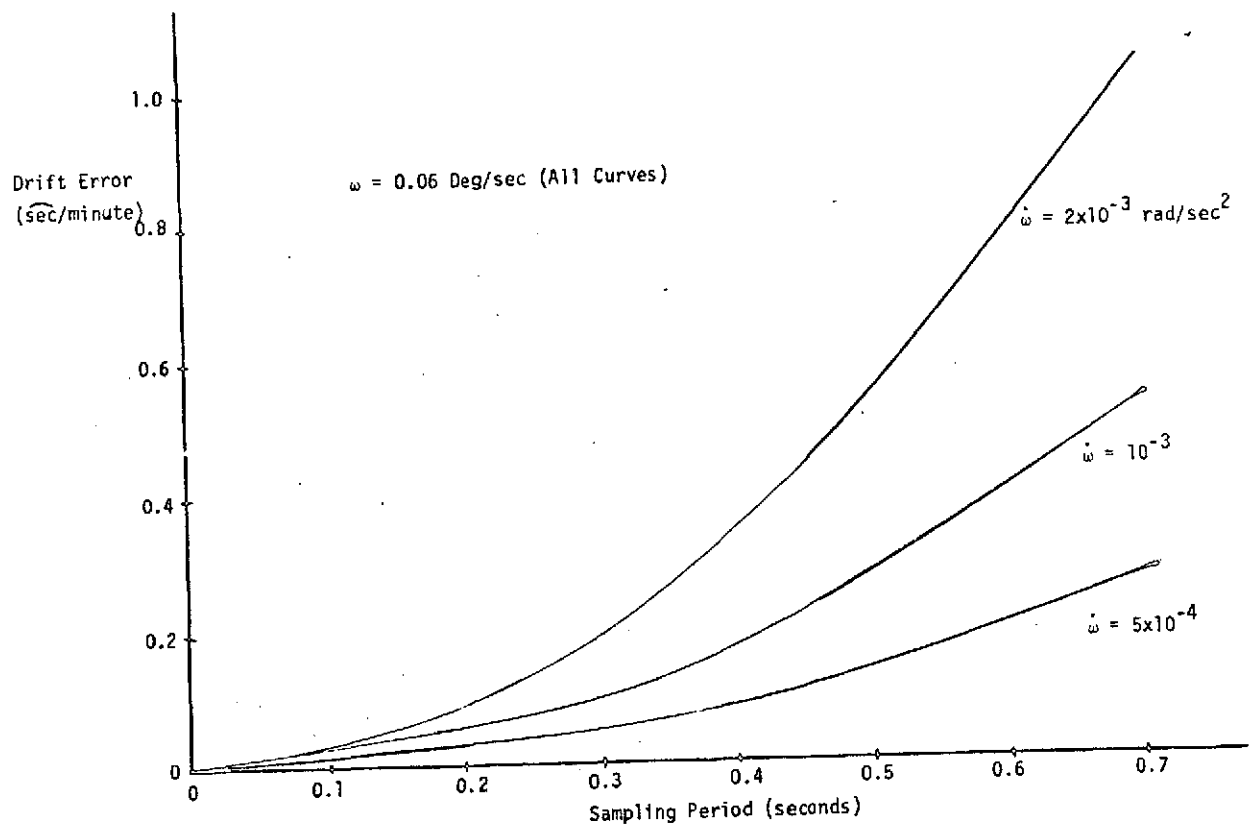


Figure 3-33 Computation Error Effects

Table 3-7

Input/Output Requirements From/To Peripherals  
(Relative to IOU)

Peripheral Name	Signal Name (at Peripheral)	No. of Inputs			No. of Outputs	
		Bilevel (bits)	Analog	Serial Words	Bilevel (bits)	Serial Words
Wide Angle Sun Sensor (4)	Pitch Output		4			
	Yaw Output		4			
	Temperature		2*			
Fine Sun Sensor (2)	Pitch Output			2		
	Yaw Output			2		
	Temperature		2*			
Star Sensor (2)	Yaw Output			2		
	Roll Output			2		
	Magnitude Output			2		
	Threshold Input					2
	Mode Input				4	
	Temperature		2*			
Inertial Reference Unit (1)	Gyro-A Output (R&P)		1	1		
	Gyro-B Output (R&P)		1	1		
	Gyro-C Output (R&P)		1	1		
	Gyro D Output (R&P)		1	1		

Table 3-7

Input/Output Requirements From/To Peripherals  
(Relative to IOU) (Continued)

Peripheral Name	Signal Name (at Peripheral)	No. of Inputs			No. of Outputs	
		Bilevel (bits)	Analog	Serial Words	Bilevel (bits)	Serial Words
Inertial Reference Unit (1) (Continued)	Gyro-E Output (R&P)		1	1		
	Gyro F Output (R&P)		1	1		
	Accel-A Output			1		
	Accel-B Output			1		
	Mode Input					1
	Torque Test Input					1
	BITE	6	2*			
Reaction Wheels&Elect (4)	Speed Command Input			4		
	Wheel Speed Output					4
	Mode Input				4	
	Temperature		6*			
	Wheel Voltage/Current		12*			
Thruster & Elect. (2)	Command Inputs			4		
	Mode Inputs				4	
Actuators & Elect. (8) (TVC & Scan Platform)	Position Inputs			8		
	Position Outputs					8
	Mode Inputs				8	
	Temperature		8*			

Table 3-7

Input/Output Requirements From/To Peripherals  
(Relative to IOU) (Continued)

<u>Peripheral Name</u>	<u>Signal Name (at Peripheral)</u>	<u>No. of Inputs</u>			<u>No. of Outputs</u>	
		<u>Bilevel (bits)</u>	<u>Analog</u>	<u>Serial Words</u>	<u>Bilevel (bits)</u>	<u>Serial Words</u>
Back-up Electronics (1)	Mode Status	2				
Power Switching Elect. (2)	Power Control					2
	TOTALS	8	14 + 34*	34	20	18

\* Slow speed analog data for telemetry (temperatures, etc.) should probably not go through the processor, but should interface directly with the telemetry subsystem.

Table 3-8  
Summary I/O Requirements

<u>Data Type</u>	<u>Input</u>	<u>Output</u>
Analog (12 bit A/D)	14	-
Serial Digital (Word Gate Bilevels)	34	18
Bilevels	8	20
Command	Single 32-bit Buffer	-
Telemetry	-	Dual 256-bit Buffer
NOTE: Analog slow speed data (marked * in Table 3-7 ) not included.		

### 3.4 Processor Design

The hardware design considerations of the processor are provided in this section. Given the requirements of Section 3.3, what are the tradeoffs of accuracy, speed, power, architecture, etc. that provides an optimum processor design? Note that these tradeoffs are strongly influenced by following sections (3.5, 3.6 and 3.7) with frequent anticipatory references to them.

The subjects of the following sections on processor design are:

- Arithmetic
- Instructions
- Interrupts
- Timing
- External Communications
- Internal Communications
- Memory
- Technology
- Architecture
- Micro-Programming

Since the COPE processor will be used as a baseline, considerable reference will be made to its design features (and to Appendix A).

The recommended design will be developed, subject-by-subject in this section, but will be completely summarized in Section 4.0. That design will be seen to have several differences from the existing COPE.

#### 3.4.1 Arithmetic

The arithmetic section of the processor is normally centralized into an arithmetic unit or an arithmetic and control unit (ACU) as in COPE. The organization of this arithmetic unit is determined by the computational requirements (see Section 3.3.2). The requirements for a spacecraft control processor were seen to be quite modest in terms of both accuracy and speed.

There are available MSI logic chips which contain 4-bit byte arithmetic and logic capability (Fairchild 93L40, for example) in a low-power design. Since these parts are more economical (in part count and power) than serial arithmetic, the use of serial arithmetic is not advisable, due to its lower speed as well.

The speed will be proportional to the number of bits processed at a time, and therefore, to the total word length. A fully parallel arithmetic organization will be the fastest, but as the degree of parallel operation increases, the part count and attendant cost, weight, and power penalties also increase.

The accuracy is directly proportional to the number of bits per word. As the word size increases, the accuracy increases, but the part count and allied penalties also increase. Double-precision arithmetic can provide an accuracy/speed trade-off also. Double-precision doubles the accuracy while halving the speed, with but a very slight increase in parts.

Since it is usually desirable to have the data word length the same as the instruction word length, and since there are usually minimum limits on practical instruction word length to allow coding of the operating instructions and addresses, 16 bit words are a practical lower limit for instructions without error detecting coding. As the instruction word length grows, it is much easier to incorporate the operation coding and greater direct addressing capability. This can serve to influence the data word length.

If error detecting coding is used, this also will lengthen the words. As will be seen in Section 3.6.1, this coding would most likely add one 4-bit byte per word.

For attitude control processors parallel arithmetic is probably not needed in any application, and is uneconomical. Serial arithmetic is no more economical than byte-serial arithmetic with a 4-bit byte. Byte serial arithmetic seems to be indicated, whatever the word length.

As new LSI parts become available (such as an 8-bit arithmetic chip) then perhaps a longer byte length would be better. Note, however, that the

pin numbers go up by 3 times the number of added bits, so that chips beyond 8-bits wide are unlikely (a more likely approach is the use of LSI microprocessor chips, discussed in Section 3.8).

The accuracy and speed requirements developed in Sections 3.3.2 and 3.3.4 indicate that a 16-bit word organized in 4-bit bytes is adequate if the operational speed is greater than 100 kops (thousands of operations per second) and if double precision capability is included for the inertial reference computations (which then take a large part of the time available (see Table 3-6 ). That is, for this application, the number (and speed of) double precision operations is the most severe restriction of the processor speed (time availability).

It should be noted that, for a given technology of circuits used, that the speed is inversely proportional to the word length. As the word length (and accuracy) increase, the speed decreases. However, for this application, at 24 bits word length, the need for double precision operations almost disappears, relieving the operating speed requirements, and making this alternative also attractive. The choice is a 16 bit word length, using considerable double precision operations, with a short instruction speed  $S$ , and an  $X\%$  cycle utilization; compared to a 24 bit word length, using little double precision operations, with a short instruction speed  $1.5S$ , and an  $0.8X\%$  cycle utilization.

In any case, the COPE processor used 16-bit word length and this still appears to be acceptable.

The penalties for word length increase are clearly not all in the arithmetic, but generally affect the entire processor. Based on a study made, using COPE as the baseline, the complete processor penalties as the word length increases, are shown in Table 3-9.

TABLE 3-9  
PROCESSOR PART COUNT PENALTIES FOR INCREASING WORD LENGTH

<u>Word Length (bits)</u>	<u>Part Count Penalty (normalized to 16-bit)</u>
16	1.00
20	1.05
24	1.11
28	1.15

As can be seen, the penalties of increased word length are actually quite low. Note that power, weight and cost are approximately proportional to part count. The conclusion is still the same, however, that the COPE word size and arithmetic organization are adequate for the job and since minimal adequacy is the measure of optimization used, this 16-bit word size and 4-bit byte arithmetic organization, is the choice recommended.

Another consideration of the arithmetic is the need for floating point vs. fixed point arithmetic. The only advantage of floating point is a slight advantage in programming ease. Its disadvantage is a hardware penalty. This hardware penalty is caused by the greater word length required.

It was decided to use fixed-point fractional two's complement arithmetic for COPE and this still seems appropriate as a recommendation. The words are organized with the most significant bit the sign bit and subsequent bits ranging from  $2^{-1}$  down to  $2^{-15}$  ( $2^{-31}$  for double precision).

### 3.4.2 Instructions

The subject of instructions encompasses not only the choice of the instruction set, but how they are encoded to limit the size of the instruction word.

The choosing of an instruction set is a series of tradeoffs (battles?) between the software and hardware designers. Any instruction added will

simplify the programming to some degree, but it will also add parts, cost, power, etc. to some degree. It is a virtual impossibility to arrive at a set that is truly optimum. Some relative optimum can be achieved, however, which is a compromise which apparently yields equal happiness for both disciplines.

Certainly, there is a minimum set of instructions necessary. One must be able to add, subtract, and input and output data between memories, ACU and IOU (& RCU). Control system requirements also indicate a need for logical operations, double precision add and subtract, and multiplication and division.

The instructions can be limited by the instruction word length, also. For 16-bit words or less, the word length is a restriction on numbers of instructions and on the direct addressing capability.

Typically, each instruction word is organized into two parts. One part is termed the operation, operation code, or op. code. The second part is the address and may refer to memory, IOU or other address.

The number of bits available for the address portion (field) determines the number of direct address word locations that can be provided. Often this address field is much less than the number of address word locations which are desired. For example, if the address field is 8 bits long, this provides direct address to only 256 words, which are much less than the total number of memory words available or desired.

To enable addressing of the total memory words desired (at least  $> 8\text{ K}$  words), some means of non-direct addressing is required. This can be done by including a data base register in the ACU that determines the sector of the total memory that the address field specifies. The data base register contents are appended (as the MSB's) to the address field, defining the total address, which can be as long as desired. The operand coding can determine whether this appending is done or not, as desired. An instruction is needed to load the data base register.

It is convenient to have indexing as an option determined by the op. code. When indexing is specified the contents of the index register are added to the LSB's of the address field. This new address is the one used (effective address). The index register should be the same length as the address field. Not all instructions necessarily need to be indexable.

The ACU must also contain a program counter, which is incremented by one, for each instruction, by most (but not all) instructions. The program counter is loaded from the effective address by an indirect jump instruction. A jump relative instruction is also convenient. This adds its "address field" portion to the program counter. Indirect addressing can also be included which treats the word addressed in the instruction as a new address (which has no op. code, and thus the entire word is available as an address) which is entered into the program counter. The program counter must be as long as the address field plus the data base register. The memory addressing takes place from the program counter.

If multiply, divide and/or double precision operations are needed, then the ACU must be designed with two operating registers as a minimum. These are usually referred to as the A-register (accumulator) and the Q-register (quotient). They are the work horse registers of the ACU, being used for temporary storage, inputting and outputting between the memories, IOU, arithmetic portion, etc. It must be possible to load into or empty out of (store) the A and Q registers from/to the memory-contained /containing operands.

The index register must also have load/store capability. There are therefore 6 instructions needed for register operations. They are:

- LDA - Load A-register
- STA - Store A-register
- LDQ - Load Q-register
- STQ - Store Q-register
- LDX - Load X (index) - register
- STX - Store X - register

A store operation does not disturb the register contents. The first four instructions should be indexable.

A certain number of arithmetic instructions are needed. All should be indexable. They are:

- ADD      -      Add to A-register
- SUB      -      Subtract from A-register
- ADQ      -      Add to Q-register
- SUQ      -      Subtract from Q-register

In each case the operand (or its two's complement for subtraction) is added to the contents of the specified register and the result is placed back in the same register.

For a multiply (MPY) the contents of the A register are multiplied. (using a suitable algorithm) by the operand and the results (which are double length) are placed in the A&Q registers.

For a divide (DVD) the double length contents of the A&Q registers are divided (again using a suitable algorithm) by the operand and the results are placed back in the A register. (With the Q register zeroed). Care must be taken to properly handle overflows of division.

Complement (CMP) instructions, performing a two's complement of the A-register contents, can also be easily provided (since it is already a part of the subtract operations) if desired.

It is often useful to also provide a limited number of logical instructions such as AND (logical product), OR (logical sum), and exclusive OR. These are not absolutely needed, however.

Because of the use of fixed point arithmetic, a number of shift instructions are needed. These can shift left or right and may operate only on the A-register or on the A and Q registers in series (for double-length words). The shifts may be designed so that over-or-under-flow bits are lost, or so that a cyclic shift occurs (where the LSB shifts to MSB,

or vice versa). The number of bits of the shift usually occupy part of the op. code or address field of this type of instruction.

Double precision operations can either be handled by the use of instructions unique to them or by particular juxtaposition of conventional instructions (such as ADQ and ADD, for example).

It may also be desirable to provide subroutine linkage instructions to permit direct loading or storing of the program counter. Instructions can also be provided for testing (usually for zero) the index register, program counter, or other register. These instructions are usually program conveniences and are not essential.

Another class of instructions is needed for input/output (I/O) operations. They are used to implement various operations in the IOU and to interface data between IOU and ACU. For I/O instructions, the address field refers to the I/O address. This may still be indexed analogously to the memory indexing.

These instructions usually include the capability to test (determine if it is a 1 or a 0) any IOU input or internal IOU indicator bilevel; to set or reset IOU bilevel outputs or internal IOU bilevels; and to shift data in either direction between the IOU register(s) and the ACU register(s). Usually these instructions can contain both a test and an action capability. If the test results in a 0, the next instruction is executed. If the test results in a 1, the setting, resetting, or shift is accomplished. The shifts can be of standard length or programmable. The shifts might be byte-serial or serial. One might also have more instructions to permit direct IOU/memory data interface.

Some other miscellaneous instructions may be needed because of the fault tolerance requirements. They can include delay or advance instructions and instructions involving data exchange between the ACU and RCU (or equivalent.)

In the COPE processor, a total of 38 basic instructions were used. See Appendix A for a complete listing of these instructions, their coding and use, and the specific multiply and divide algorithms used.

Note that with indexing and sectoring optional, the total instructions are much greater than the basic number. This total can be limited only by the ingenuity of the coding.

A basic instruction set of 35 to 40 instructions should be adequate for control system use. Note that this takes a minimum of 6 bits for the op. codes, without considering bits for indexing, sectoring, memory type specification, etc. Such additional features can quickly use up another 2-4 bits of op. code. We, therefore, see that a 16-bit instruction word minimum is indicated to provide the minimum 8-bit op. code and 8-bit address field.

A longer instruction word can permit simpler and better coding of the op. code; redundant or error-detecting coding of the op. code; a larger direct address field; error detecting coding of the address field; or combinations of these. Note the penalties of Table 3-9 in this, however.

Special, limited-use, instructions (and the resulting algorithms) for other operations can also be included, much as in a scientific calculator. Functions such as  $e^x$ ,  $\ln x$ ,  $x^y$ , trigonometric functions, etc. could be included. Generally speaking, such functions only simplify programming and always at the expense of hardware. Possibly in a micro-programmed LSI processor (see Section 3.8.4) the addition of such functions would be more practical. Again, hardware that is not included can not add weight, power, cost or failures.

### 3.4.3 Interrupts

Interrupts are typically used in asynchronous data processing systems for servicing priority demands. Interrupt generation and processing can be done by either hardware or software or both.

In an attitude control system including fault tolerance features, interrupt requirements are determined by operational and failure-handling requisites. Operational interrupts are necessary when rapid responses to priority demands are required without significantly increasing the processor burden. Otherwise, these demands can be serviced by periodically sampling the corresponding peripherals with the required frequency under program control. Since high bandwidth operations are performed by peripheral special-purpose electronics in order to make the system less sensitive to processor faults, sampling frequencies are relatively low (typically in the 0.5 to 5 Hz range). System reaction times of the order of 0.5 to 1 sec are adequate for most eventualities. Consequently, peripheral sampling can be used for all ACS functions without performance degradation or appreciable duty cycle penalty. Power interruptions can be handled without reinitializing interrupts if mode, status and rollback data are preserved through the transients. The processor should be designed to recover from failures caused by power interruptions without unnecessary reconfigurations.

Interrupts of various kinds may be used for recovery purposes. Unconditional interrupts are preferred to minimize the proliferation of failure modes. Two types of interrupts that have been found to perform very powerful recovery functions are the following:

- Cycle Timing Interrupt (CTI)

This occurs at regular intervals (e.g.: every minor or major cycle) and causes a reset of the ACU program counter (to start processing executive instructions). The main functions of this interrupt are 1) to get the processor out of any hangups or loops and 2) to initiate a bootstrapping recovery process controlled by the executive program and monitored by the RCU and HCU.

- Fault Interrupt (FI)

This is initiated by either software or hardware fault signals each time initiation of the bootstrapping recovery process is required. This interrupt can be used in addition to the CTI for speeding up the recovery process.

The recommended recovery approach (see Section 3.5.5) uses CTI's only because of the following reasons:

- Recovery time requirements can be met easily
- Additional complexity and operational risk of FI's is not warranted.
- Exclusive use of CTI's provides a more orderly and predictable program execution, thus enhancing operational reliability.

All of the preceding discussion assumes the use of a synchronous executive program, as discussed in Section 3.5.

Note that interrupts are not needed for peripheral, command or telemetry interfacing if data buffering is included in the sending peripherals, or in the IOU (for commands and telemetry).

The use of interrupts in control system processors seems to be traditional, and based on the "we've always done it that way" approach of general-purpose computers or aircraft computers (where it is needed and makes sense), rather than an examination of actual need. It is particularly undesirable as a feature of a fault-tolerant design.

#### 3.4.4 Timing

Timers or timing functions are required within the processor for the following functions:

- Master Clock

A multi-phase (usually four) master clock is required in the processor or somewhere in the system to provide the overall clocking of the processor data. The frequency is usually high (500 KHz to 1.5 MHz) and is derived from a crystal oscillator. Further discussion of this clock and its redundancy problems, is provided in Section 3.6.3

- Program Cycle Timing

Assuming the synchronous, fault-tolerant structure of the program, as developed in Section 3.5, then minor or major program cycle durations are determined by cycle timing synchronization signals derived from the master clock output by means of binary count down logic. Provisions should be made to allow hard-wired adjustments of this CTI frequency, prior to launch, within the range from 3 to about 50  $H_z$ .

- Status Check Window

Within a specified time interval after each CTI, the RCU should receive a correctly-coded status word from the ACU. If either no word or an incorrect word are received within the status check window, the RCU initiates reconfiguration of the primary processor units (ACU, ROM, DBS). (See Section 3.7). Status check window signals are generated by an adjustable one-shot multi-vibrator in the RCU's which is triggered by the CTI's.

- HCU Reconfiguration Monitoring

The HCU supervises RCU reconfiguration functions by means of an adjustable one-shot multi-vibrator which is triggered each time a correct status word is transferred from the ACU to the RCU. If a correct word is not received within the HCU timer period, an RCU reconfiguration is initiated. This period is equal to the longest possible reconfiguration time, plus the program cycle time.

- Discrete Event Timing

Timing functions exceeding the duration of the CTI period can be handled by the software with a quantization equal to the CTI period. If finer resolutions are required an optional real-time counter can be included in the processor (in the IOU). The timing in this case would be accurate only if no faults (requiring IOU

reconfiguration) occur. This timing could be obtained by dividing from the master clock, with or without a reset at each CTI period.

The master clock, program cycle timer and status check window generator should be located in the RCU. The redundancy management system of Figure 3-22 is used. None of these timers need to be protected by passive redundancy because the HCU will detect failures in the RCU indirectly when they cause distinguishable operational failures.

The RCU is a preferred location for the master clock and the program cycle timer because the operation of the RCU should not depend on functions provided by the units it is supposed to reconfigure, and the clock and timers are more reliable than the ACU's and, therefore, should not be replicated as much.

#### 3.4.5 External Communication

This subject generally relates to the signal interfaces between the processor and peripheral system electronics or the command, telemetry, or other systems of the spacecraft. These interfaces input and output from/to the processor are termed I/O functions and they are handled by the Input/Output Unit (IOU).

##### 3.4.5.1 Peripheral Interfaces

The peripherals refer to the sensors, actuators and associated electronics within the control system that must communicate with the processor. It is desirable that all peripheral interfacing (with the possible exception of the back-up electronics mode/use control and analog telemetry of low rate) be through the processor and not direct to other systems.

A given peripheral may have outputs or inputs (or both) which interface with the IOU, as shown in Figure 3-34.

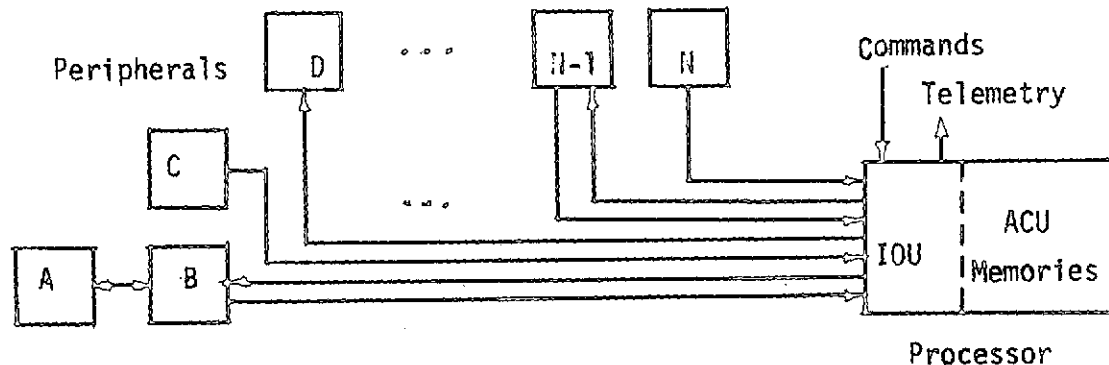


Figure 3-34  
General External Communications

If the peripheral is a sensor, it will primarily have outputs (signal, mode status, BITE, etc.) but it may also have inputs for power control, mode control, biases, thresholds, etc. If the peripheral is an actuator, its electronics will primarily receive inputs (biases, signals, power or mode control, gains, etc.), but it may also provide outputs (speed, position, mode status, BITE, etc.).

Because both sensors and actuators (including their electronics as part of the sensor or actuator), can function to either accept inputs (receive) or provide outputs (send) from/to the IOU, these peripherals will simply be characterized as being "senders" or "receivers".

The sender or receiver can produce/accept the following types of data:

- o Senders - Bilevels, Discretes, Analog, Digital Words  
(serial or parallel)
- o Receivers - Bilevels, Discretes, Analog, Digital Words  
(serial or parallel)

Here, a discrete is a bilevel of momentary character. Not all senders or receivers will (or should) have all of these types of data. In fact, certain types of data interfaces should not be used.

Generally speaking, the use of discretes should be minimized or eliminated. The IOU requires more circuitry to handle them as inputs than for bilevels because latches must be provided to store the discretes (which are reset upon readout). For bilevels, simpler digital gating may be used. Discrete generation by the processor is also somewhat more difficult and usually unnecessary as the receiving peripheral can be designed to recognize the change in state of bilevel signals instead.

Analog outputs from senders are appropriate if the source is intrinsically analog (but a digital signal should not be made into analog). The IOU will require an analog multiplexer and a single, shared A/D converter for these inputs. The A/D converter will require an accuracy and resolution commensurate with the most severe requirement on any input. A 12-bit A/D converter is realizable and adequate.

Analog inputs to receivers are undesirable. Any signal that is ultimately needed in analog form in a receiving peripheral originates in digital form in the processor. Of necessity it is time-quantized sampled data. The choices would be to provide a shared D/A converter in the IOU, together with analog output multiplexing; or to send the data to the peripherals in digital form, with a D/A converter in each receiving peripheral. In either case, because the data origination is non-continuous, some "holding" of data must be done between updates. In the first case this would have to be done in analog sample-hold circuits in each receiving peripheral. In the second case, it can be done in holding registers in each peripheral receiver.

The two choices are shown in Figures 3-35 and 3-36. Analog sample-hold circuits are somewhat difficult to design and also tend to "forget" as they discharge. They also take more parts to create them than digital holding registers. The D/A converters in Figure 3-36 can each be designed for just the number of bits of resolution/range needed on that output. It can be shown that the circuitry of Figure 3-36 is less

complex and requires fewer parts. For this reason, as well as for its "perfect" memory, it is the best choice. Analog outputs from the IOU are therefore not necessary.

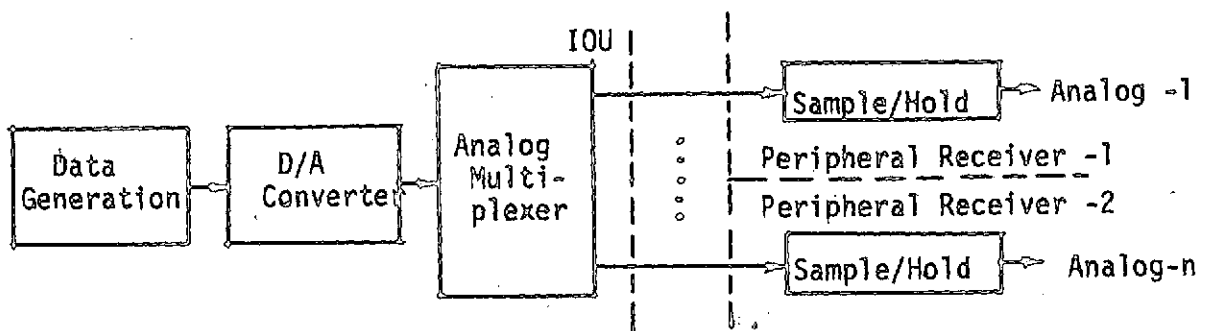


Figure 3-35  
A Method For Analog Handling of Processor "Analog" Outputs

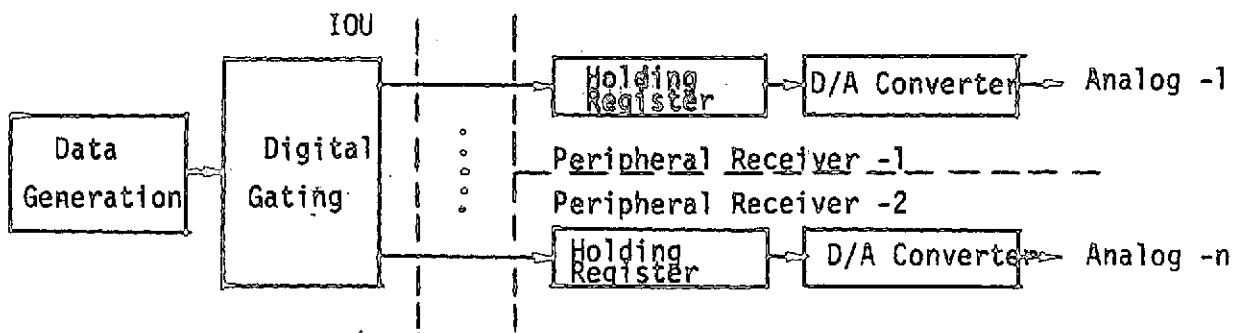


Figure 3-36  
A Method For Digital Handling of Processor "Analog" Outputs

As far as the choice between serial and parallel digital data interfaces are concerned, the serial is to be much more greatly preferred. Parallel data interfaces are simply a quantity of bilevels and require much more IOU circuitry, no real savings of circuitry in the receivers and provide no real speed advantage (which would not be needed in any case). All digital data word interfaces will therefore be assumed to be serial (for both senders and receivers).

The senders or receivers therefore should produce/accept only the following types of data:

- Senders - Bilevels, Analog, Serial Digital Words
- Receivers - Bilevels, Serial Digital Words.

The bilevel inputs or outputs should be dedicated to each sender/receiver function. Therefore, there will be as many lines as there are signals and sources or destinations. Cross-strapping can be normal (See Section 3.1.3) or it can be by duplication at the IOU. The latter should be explained.

See Figure 3-37 . Here it is assumed that each element is composed of two blocks. One sender bilevel and one receiver bilevel are shown.

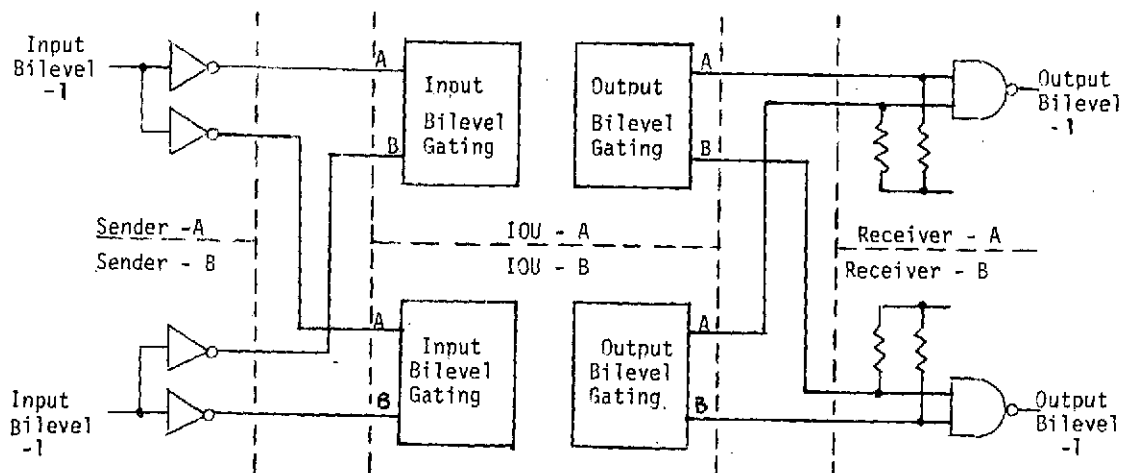


Figure 3-37

Duplication Cross-Strapping (for Bi-levels)

Each of the peripherals has the normal cross-strapping, but the IOU duplicates the inputs (and the outputs) to provide the equivalent of cross-strapping. The processor keeps track of which block of each element is on (in use) and addresses the proper input and output (sets). This saves the cross-strap circuits in the IOU, but at the expense of double (assuming single-standby redundancy in the peripherals) IOU bilevel inputs and outputs, and some slight software overhead. The alternative is the conventional cross-strap circuitry of Figure 3-14.

The analog inputs redundancy is best handled by duplication (wherein the processor keeps track of the functioning block of each element) also. In this case, the added analog multiplexer inputs are much more economical than analog cross-strapping.

The remaining interfaces are serial digital words. There are some tradeoffs in how these interfaces are best accomplished, that are discussed in the next section.

#### 3.4.5.2 Serial Data Word Interfaces

Consider first the situation where the data originates in a sending peripheral. This data generally is updated into a counter or register in the sending peripheral asynchronously with the processor clock or cycles. The updating may be more or less frequent than the readout.

- o If the readouts are less frequent than the updates, then the readouts between updates should be repeats. (Reading the same thing on subsequent readouts is less confusing to the processor and then the new update need not be signalled across.
- o If the updates are more frequent than the readouts, then the readout can be destructive and the updates must each replace the older data.

The data should be clocked into the IOU using a processor clock. This is usually done using a two-phase clock arrangement, with one clock clocking the data in and the other (IOU internal one) reading the data.

The best interface approach appears to be to use a shared input data line from all sending peripherals into the IOU, together with a peripheral clock which goes from the IOU to all sending peripherals.

Each sending peripheral must be told when it is to shift the data from its register into the (shared) serial input register of the IOU. This can be done in several ways:

- Separate bilevel word gates to each sending peripheral.
- Separate, selected clock signals to each sending peripheral.
- A parallel coded address bus going to all sending peripherals, with each detecting its own address.
- A coded address sent in serial to all sending peripherals, with each detecting its own address.

A tradeoff has been made between these approaches. This tradeoff starts with the COPE approach (separate bilevel word gates) and the signal terminology used in COPE.

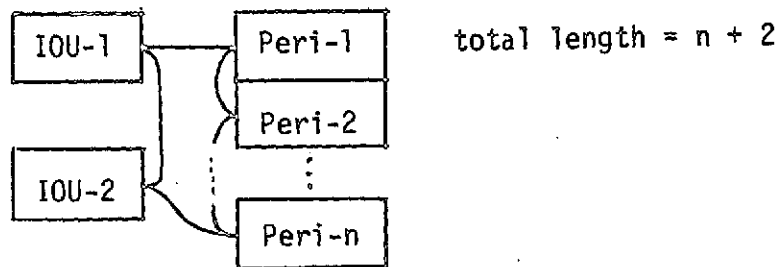
The approaches will be described in terms of number of inputs and outputs for IOU and peripherals. Then connector pins, line lengths and circuit differences will be calculated. To perform the tradeoffs, a system of average complexity is assumed:

- Two IOU's (single-standby redundancy)
- All peripherals use single-standby redundancy (two blocks per element)
- Two elements which only send data
- Two elements which only receive data
- Twelve elements that both send and receive data.

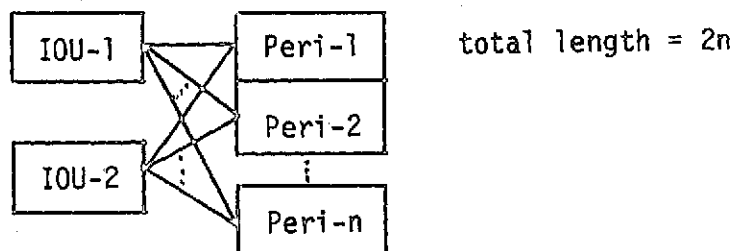
Both sending and receiving peripherals are covered in this same tradeoff.

Also assume that connector weight is proportional to the number of pins (not quite true) and that line wire weight can be calculated as follows:

- Assume all "units" (or blocks) are equal distance from each other (unit distance)
- For each signal on a "bus"-type line, we use daisy-chain wiring, resulting in: ( $n$  = total number of peripherals)



- For each signal on a non-bus-type line, point-to-point wiring must be used, resulting in:



The four approaches are:

- Separate Bilevel Word Gates

Each IOU has the following outputs:

DATAI - A & -B

CLKP - A & -B

BUSBL

up to 16 - SBL0XX

up to 32 - SBL1XX

} go only to the peripherals, as needed.

And the inputs:

DATA0-A & -B

Each peripheral has the following inputs:

DATAI -A & -B (for senders)

CLKP -A & -B

BUSBL

SBLIXX (for senders)

SBL0XX (for receivers)

And the outputs:

DATA0-A & -B (for receivers)

All inputs/outputs from/to each peripheral need bus senders/receivers except BUSBL). The same is true of all inputs/outputs from/to the IOU (except SBL0 & SBLI, which are assumed cross-strapped by duplication.)

The IOU contains SBL0 and SBLI decoding (6MSI & 10 SSI) circuitry. The peripherals contain only the data gating into shift registers (which all of the approaches have in common).

- Separate Selected Clock Signals

Each IOU has the following outputs:

DATAI -A & -B

BUSBL

up to 16 - CLP0XX } go only to the peripherals, as needed  
up to 32 - CLPIXX }

And the inputs:

DATA0 -A & -B

Each peripheral has the following inputs:

DATAI -A & -B (for senders)

BUSBL

CLP0XX (for senders)

CLPIXX (for receivers)

And the outputs:

DATA0 -A & -B

The only circuitry differences from the separate bilevel word gate approach is the need for "and" gating of the CLKP internal to the IOU (about 12 SSI added). There are no savings in the peripherals.

o Coded Bilevel Address Approach

The IOU has the following outputs:

DATAI - A & -B

CLKP -A & -B

BUSBL

6 wire Address Lines (-A & -B) Allows up to 64 addresses.

And the inputs:

DATA0 -A & -B

Each peripheral has the following inputs:

DATAI -A & -B (for senders)

CLKP -A & -B

BUSBL

6 wire Address Lines (-A & -B)

And the outputs:

DATA0 -A & -B (for receivers)

All inputs/outputs from/to each peripheral need bus senders/receivers (except BUSBL). The same is true of all inputs/outputs from/to the IOU.

Each peripheral sender or receiver needs a decoder (1 SSI) to decode the six lines and generate an internal word gate.

The IOU does not need any decoders, but each of the peripherals do. This adds approximately 1 MSI and 1 SSI to each peripheral and saves 6 MSI & 10 SSI in the IOU.

Also added are 6 x 2 senders (2SSI) to each IOU and the same for receivers (2SSI) to each peripheral.

Note that the peripheral cross strapping is here more conventional and the two (or more) redundant blocks of the same element will have the same address and the processor does not need to keep track of which is on, since only the on block will respond.

- Serial Coded Address Approach

The IOU has the following outputs:

DATAI -A & -B

CLKP -A & -B

BUSBL

And the inputs:

DATAØ -A & -B

Each peripheral has the following inputs:

DATAI -A & -B (for senders)

CLKP -A & -B

BUSBL

And the outputs:

DATAØ -A & -B

All inputs/outputs (except BUSBL) need bus senders/receivers. The assumption is made here that the coded address is a part of the DATAØ (serial output data line). Six bits are added in the MSB position.

All peripherals would shift in all bits of all words on the word line in synch with the gated clock. Only the one peripheral whose address is proper (as determined by the MSB's (address) as decoded in the peripheral) would permit storage and "execution" of the data portion of the word.

Inputs to the IOU would require handling in a different manner. First the peripheral would need to be addressed (in the same manner as for an IOU output) with a code indicating a serial IOU input (peripheral output)

will follow. Then the next "word length quantity" of clock pulses will cause the word data to shift into the IOU on the serial input word line.

The bilevel decoders in the IOU (6MSI & 10 SSI) could be deleted. Address decoding would be needed in each peripheral.

Note that a much longer time is required to input data to the IOU (the address time, plus two serial words of data).

The important characteristics are summarized in Table 3-10. This data is used to provide the further summary of Table 3-11, where comparisons are made to the COPE approach.

It is seen that neither of the coded address approaches are favorable. The separate selected clock signal approach has a slight weight advantage and a slight part count disadvantage, relative to the separate bilevel word gate approach, and either of these might be selected.

TABLE 3-10 SUMMARY OF PERIPHERAL DATA GATING COMPARISONS RELATIVE TO SEPARATE BILEVEL WORD GATE APPROACH

<u>Characteristic</u>	<u>Separate Selected Clock Signals</u>	<u>Coded Bilevel Address</u>	<u>Serial Coded Address</u>
Circuits	Add 7SSI	Add 20 MSI, 80SSI	Add 74MSI, 172SSI
Circuit Weight*(lb.)	+0.2	+2.0	+5.0
Line Lengths	-34	+84	-118
Line Weight #(lb.)	-0.3	+0.7	-0.9
Connector Pins	-68	+200	-176
Connector Weight \$(lb)	-0.3	+1.0	-0.9
Net Weight (lb.)	<u>-0.4</u>	<u>+3.7</u>	<u>+3.2</u>
Complexity Increase	<u>slight</u>	<u>large</u>	<u>quite large</u>

\* - Assumed 50 IC per pound

# - Assumed 2 feet average per length at 4 lbs/1000 ft.

\$ - Assumed at 0.005 lb/connector pin.

<u>CHARACTERISTIC</u>	<u>SEPARATE BILEVEL WORD GATE (COPE)</u>	<u>SEPARATE SELECTED CLOCK SIGNALS</u>	<u>CODED BILEVEL ADDRESS</u>	<u>SERIAL CODED ADDRESS</u>
IOU Bus Senders	8	6	32 (add 4SSI)	8
IOU Bus Receivers	4	4	4	4
Peri Bus Senders	56	56	64	60
Peri Bus Receivers	120	88(delete 5SSI)	512 (add 64 SSI)	128
Decoders, IOU	2	2	0 (delete 12 MSI, 20SSI)	0
Decoders, Peri	0	0	32 (add 32 MSI, 32 SSI)	32 (add 64 MSI, 160 SSI)
Encoders, IOU	0	0#(add 12SSI)	0	2 (add 10 MSI, 12 SSI)
Line Lengths	252	218	336	134
Connector Pins, Per IOU	67	65	19	7
Connector Pins, Per Peri*	9	7	19	7
Connector Pins, Total	414	346	614	238

TABLE 3-11

## COMPARISON OF PERIPHERAL DATA GATING APPROACHES

\* For those both sending and receiving

# For the clock gating circuits

The circuitry used in the peripherals for sending and receiving should next be discussed. Note that a single peripheral block may be a sender only, a receiver only, or both a sender and a receiver. It may also send and/or receive more than one word. One word gate enable will be received per data word sent/received per peripheral block.

For a receiving peripheral, the circuit block diagram would look like Figure 3-38. Here, it is assumed that three words might be received by this peripheral. The shift register and control logic are shared by all words.

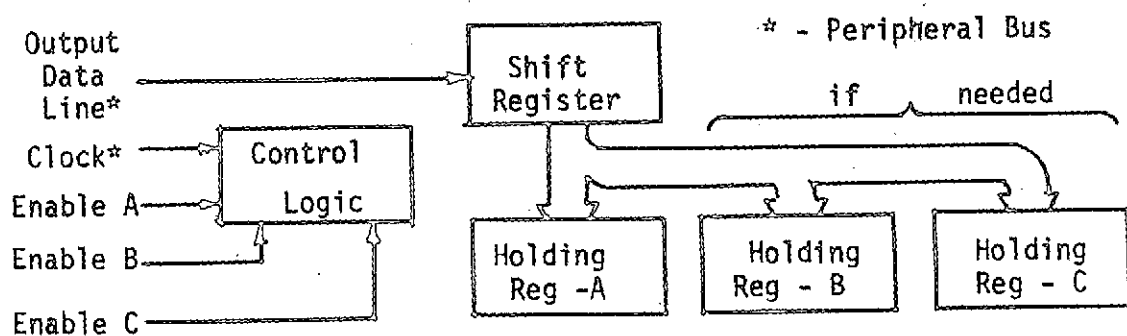


Figure 3-38  
Receiving Peripheral

When the enable goes high (any enable appropriate to this peripheral) the clock shifts the data into the shift register from the data line. When the enable goes low, this data is parallel transferred to the proper holding register, updating (replacing) the data there, which is then remembered until the next update.

The outputs of the holding registers may be used in the peripherals as needed. Some uses are:

- inputs to D/A converter for analog outputs.
- inputs to counters, etc.
- bilevel functions (such as valve on/off, etc.)
- other decoded functions

The circuits for the shift register (assumed 16 bits), control logic and one holding register require 8 MSI & 2 SSI. Each additional holding register requires 4 MSI & 1/2 SSI. This does include peripheral data bus receivers (not shown above).

If fewer bits are needed, the registers may be shortened (at the rate of 1 MSI/4 bits), but the data must be placed in the MSB's of the word.

For a sending peripheral, the circuit block diagram would look like Figure 3-39. Here, it is assumed that two words might be sent by this peripheral. Again, the shift register and control logic are shared by both words.

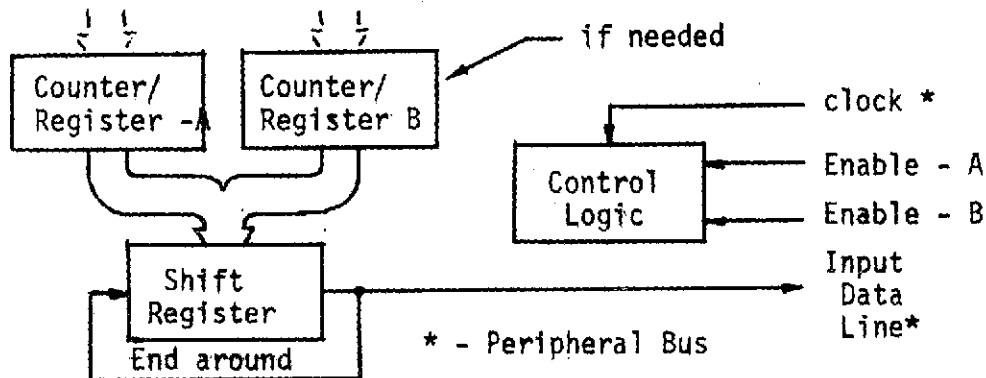


Figure 3-39  
Sending Peripheral

The case shown is where the peripheral accumulates data on a cyclic basis in a counter or equivalent. As each cycle is completed, the "counter" parallel transfers the data into the shift register (unless the enable is high).

When the enable goes high, the clock shifts the data out on the data line to the IOU and also recirculates the data back into the shift register input (to allow subsequent reads prior to cycle end.) The logic prevents updates while a readout is occurring.

Another case may occur where the data is always available (in parallel) and no cycles are involved. For this case, when the enable goes high, the parallel entry of data is disabled, and the data in the register is shifted out. No recirculation is necessary.

The circuits require (not including the counter or equivalent) approximately 4 MSI & 2 SSI for 16 bits, including peripheral data bus interfacing. Again, fewer bits (and parts) may be used, but here the data will be placed on the LSB's of the word.

For some applications, it may be feasible to combine the circuitry for sending and receiving data, as shown in Figure 3-40. Here, only one input and one output word are shown.

Here, the shift register is used for both inputting and outputting data, with an attendant savings (4 MSI) in circuitry. The logic controls the transfers and shifts appropriately.

Note that this circuit may have other utility, since it can be used to transmit a received (inputted) word back to the IOU, confirming proper receipt. This word could be compared with the sent word in the processor as a check of the peripheral data loop.

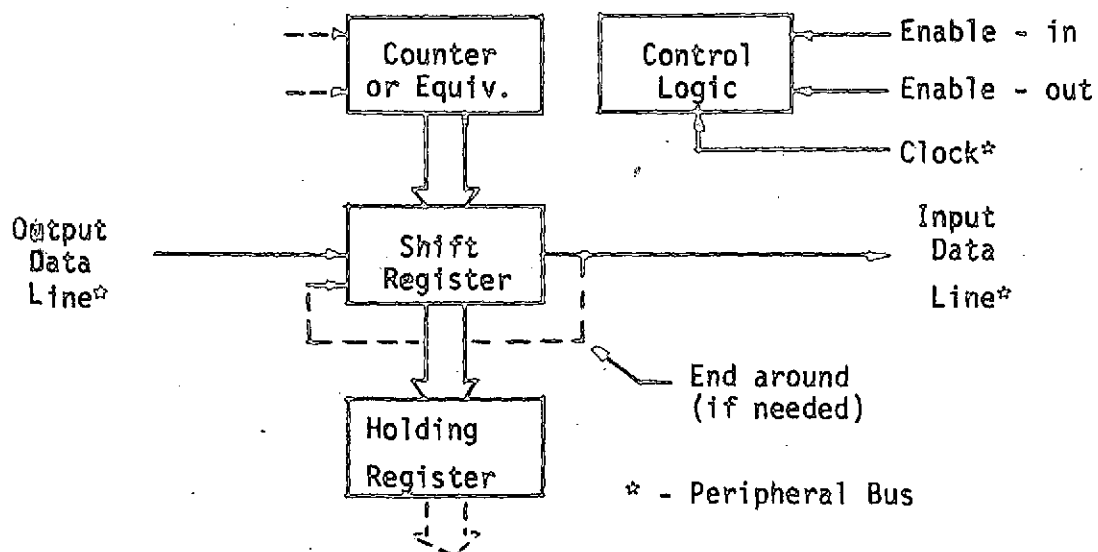


Figure 3-40

Figure 3-40 Sending and Receiving Peripheral

### 3.4.5.3 Command and Telemetry Interfaces

Requirements for the command interface are:

- Full IOU buffering to permit asynchronous command entry.
- Assume that partial command decoding will occur outside the control system (in a "command" subsystem). The IOU would then receive the following signals:
  - Command data line
  - Command clock
  - Command enable (word gate)
  - Command execute, verify, parity, or whatever (optional)

The command system will strip off all other data, decoding the "system" address and enabling the IOU (through the word gate enable).

- The "command system" will be redundant and all command lines will be cross-strapped into the IOU(s).
- Command clock rates will most probably be in the 100 - 300 Hz range, but rates down to essentially zero may need to be accommodated.
- The command system of different spacecraft may have differing "word" lengths ranging from 8 bits upward to perhaps 16 bits. The IOU design must be able to accommodate these variations.
- The required word length for a command is dependent on the following factors:
  - the data word length of the processor
  - the address capability of the processor
  - the presence of check bytes/bits in the word

We will consider here ranges of:

- 16 to 24 bits for data words
- 8 to 16 bits for addresses
- 0 to 4 bits for check bytes/bits

o The word should be arranged so that the address occupies the LSB position and the data the MSB position. The check byte/bit may be between address and data portions, following data or both places.

o To obtain the needed command word length for the processor may take the sequential receipt of several "command system" words. Examples are shown below:

NUMBER OF COMMAND SYSTEM WORDS NEEDED

Command System Word Length	Needed (Processor) Command Word Length					
	24	28	32	36	40	44
8	3	4	4	5	5	6
10	3	3	4	4	4	5
12	2	3	3	3	4	4
14	2	2	3	3	3	4
16	2	2	2	3	3	3

- o If the processor uses check bytes for error detection, the check byte should be included in the command word. This ensures accuracy (or non-execution) in the received command.
- o The IOU must contain a command buffer register which can contain the entire needed command word length. (Alternately, the command can be received in two parts, address first and data last, but with more complexities in the software.)

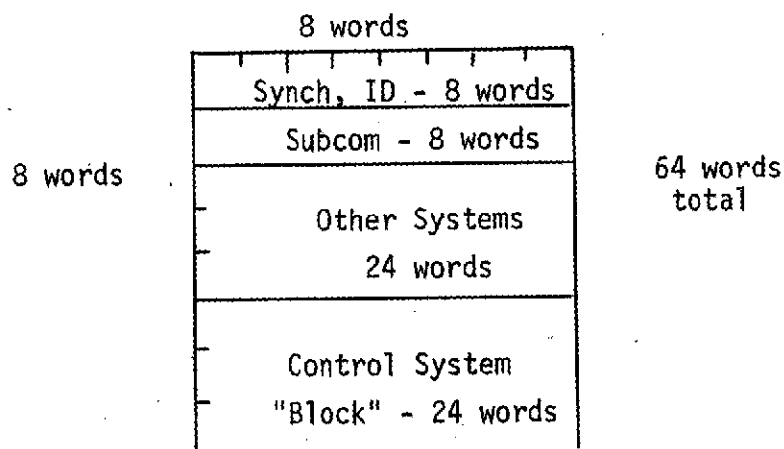
- The processor must "keep track" of the proper number of received "command system" words (by counting the falls of the command enable signal.) When this count indicates that a whole message has been received, the buffer register is emptied and readied for the next command. The counting should be done in the IOU.
- The routine is as follows:
  - Periodically, the program examines the IOU command counter to see if a command is ready for processing. This is done with a SBL instruction (set bilevel). A "no response" indicates a return to the program. A response indicates the command is ready and sets a bilevel in the IOU.
  - The INS (input serial) instruction is then executed as many times as necessary to read the command address and data into the accumulator.
  - The RBL (reset bilevel) instruction is then executed to free the command buffer for the receipt of new command messages.
  - Other instructions are also needed for routing of the address and data portions of the message properly.
- Protection must be provided to prevent writing of a command into "program" memory unless precautions are taken to prevent errors. This can be done by "lock & key" techniques or equivalent. (See Section 3.4.7.)
- As stated elsewhere, the processor should handle all control system commands, except for:
  - Those needed for "lock & key" input enable/disable
  - RCU override control of reconfiguration or on/off status.

- The IOU command circuitry on COPE (32 bit buffer) required 4 MSI & 4 SSI.

Requirements for the telemetry interface are:

- Full IOU buffering to permit asynchronous interfacing with the telemetry system.
- Versatility in IOU design is required to permit interfacing with a variety of different configurations of spacecraft telemetry systems. Those variations might include:
  - Word size (6 to 24 bits)
  - Frame size (almost anything in multiples of  $2^n$ )
  - Clock Rate ( $4H_z$  to  $4KH_z$ )
  - Use of sub-commutation
  - Word Rate (wide range)
  - Frame Rate (wide range).
- The degree of control exerted by the telemetry system on the control system data entry should be minimized. In explanation:
  - Requiring particular data formatting, such as sun sensor yaw error in a particular main frame/subcom word slot would require an unfortunate degree of control of the processor by the telemetry system.
  - Such formatting would require many inputs to the IOU to provide the control, would slow-down the processor (effectively "slaving" it to telemetry), and would provide no real advantages.
- A better approach would be to assign "blocks" of words to the control system within the telemetry format. These words are best contiguous. On the ground these words are only known to be "control system telemetry". The words are individually identified by use of an identification word within the control system words. To make this more clear, consider:

- The telemetry main frame normally consists of a square matrix of words. (Assume, for example,  $8 \times 8 = 64$  words.)
- The first few words are dedicated to synchronization, ID, etc. (Assume 8 words).
- Some main frame words are subcommutated for slower data rate uses (assume 8 words).
- The remaining main frame words are dedicated to the various spacecraft system telemetry needs. (Assume the control system gets the last 24 words.)
- See the figure below:



- This frame repeats over and over again. The "normal" decoding on the ground counts words from the beginning of the frame to locate (identify) any particular data word. This is a decommutation process.
- The control system block is decommutated only to identify the block (not individual words) and the entire content of that block is recorded, with sequential blocks following each other.

- The processor generates an identification word which may reside in any word slot of any of the control system blocks. These identification words are separated by all of the unique control system data words of that particular control system telemetry format (sequence). This sequence length may be more or less long than the block length and is not tied in to the start of the block.
- The decommutation of the control system telemetry data on the ground is done by the word position relative to the identification word.
- The telemetry system word length and the control system word length need not be the same.

o The approach as outlined above has several advantages:

- It permits completely asynchronous operation of the systems.
- Programmable (commandable) telemetry formatting of control system data is enabled. Any program for the formatting may be commanded into the processor, as long as its inverse is used on the ground for the decommutating. Thus the format may be changed at will, providing an extremely powerful tool for:
  - System monitoring
  - Verification of memory
  - Verification of command data
  - Error Diagnosis
- The requirements on the spacecraft telemetry system are eased.

o Note that another, somewhat less flexible approach can be used that has the advantage of more conventional ground data decommutation. The change is only to force the control system

data to repeat starting with the start of the block. This implies:

- No need for an identification word
- A need to constrain the sequence length to the block length (or integral multiples of the block length)
- A variable format within these constraints can still be enjoyed.
- The processor must keep track of the start of the blocks.

- The telemetry system is assumed to be redundant and all lines from/to the IOU are cross-strapped.
- Since the clocking of telemetry is assumed to be continuous (no gaps, except between "blocks"), and since the total block length is expected to be long (at least 128 bits), and since the higher telemetry clock rates are quite high; the IOU could not be expected to load a buffer register (from the ACU) between telemetry clock pulses. Therefore, dual buffer registers will be needed, with one being loaded while the other is emptied and vice versa. The size of the buffers should preferably be a common multiple of the processor word length and the telemetry system word length. A size equal to the block length is ideal.
- The inclusion of the check byte/bit on the telemetry words is optional (if they are used in the processor), depending on the need for confirmed accuracy on the ground of the received data and the need for a suitable "checker" on the ground. Generally the check byte/bit would not be used.
- The buffer length minimum (and block length minimum) can be determined from the table below:

MINIMUM BLOCK & BUFFER LENGTH (bits)

Telemetry Word Length (bits)	Processor Data Word Length (bits)			
	<u>16</u>	<u>20</u>	<u>24</u>	<u>28</u>
6	48	60	24	84
8	16	40	24	56
10	128	20	120	140
12	48	60	24	84
16	16	80	48	112
20	80	20	120	140
24	48	120	24	168

For COPE it was decided to make the buffer length 256 bits, since this was bigger than any of the minimums above and was well suited to the use of a 256 x 1 bit RAM chip. (1 LSI).

- o The interfaces with the telemetry system are as follows:
  - Telemetry data line
  - Telemetry clock line
  - Telemetry word gate (for telemetry system word lengths)
  - Telemetry synch signal. (This is preferably a signal that indicates the control system block enable.)
- o The IOU must contain logic to alternate the buffers and note when they are empty (unloaded). The buffer entry status may be determined by the program.
- o The routine is as follows:
  - An SKE (skip on external) instruction tests whether a buffer is full or empty. If full, the program proceeds and retests later. Note that this test period must be fast enough to ensure that the data is always loaded and ready when needed by telemetry. This test period must be faster than the buffer unload rate (buffer length x telemetry clock rate) including the service time for the buffer (testing & completion of load).

- If the buffer is empty, a SBL (set bilevel) is executed to enable buffer loading.
  - An ARC (serial output) instruction is used to load the buffer. The software keeps track of the number of words necessary to fill the buffer.
  - A RBL (reset bilevel) completes the load.
  - The main program is re-entered and a test is made again within the test period.
- The telemetry synch signal enters the IOU as a bilevel input and can be tested to indicate the start of the control system block for format start or to issue a sequence start identification word, etc.
  - Note that neither the processor or the telemetry system needs to know which of the two buffer registers is in use at any time.
  - Note the considerable power that command-variable telemetry formats can provide. It may also be possible to vary the format in the telemetry system, expanding or contracting the control system block size and/or the telemetry rate, adjusting the quantity and frequency of data upon need. This is a significant advantage of control systems employing processors since the advantage can be obtained at practically no cost.
  - As stated elsewhere, the processor should handle all control system telemetry, except for:
    - That needed for RCU monitoring
    - (Possibly) some low-bandwidth housekeeping functions, such as (analog) temperature measurements.
  - The IOU telemetry circuitry on COPE (dual 256 bit buffers) required 2 LSI, 4 MSI & 12 SSI.

### 3.4.6 Internal Communication

Internal Communication relates to the internal processor interfaces between the ACU and memories and IOU. (And also the RCU, or equivalent).

These interfaces will be determined by the character of the data handled. Some idea of this can be obtained by considering the inputs and outputs necessary for each processor block: (Assuming a 16-bit word length).

- ROM
  - ROM Enable (bilevel)
  - Memory Address (parallel - 15 bits)
  - Memory Read Data (parallel - 16 bits)
  
- RAM
  - RAM Enable (bilevel)
  - Memory Address (parallel - 15 bits)
  - Memory Read Data (parallel - 16 bits/byte-serial - 4 bits)
  - Memory Write Data (parallel - 16 bits/byte serial - 4 bits)
  - Read/Write Mode (bilevel)
  - Clocks (1 or 2 phase)
  - Micro-timing (2 phase)(Used only for byte-serial read/write)
  
- PWM
  - Same as RAM, plus ...
  - First/Second word (bilevel)
  
- IOU
  - I/O Function Code(Address) (Parallel - 3 - 4 bits)
  - Address Gate (bilevel)
  - Output Data (byte serial - 4 bits)
  - Input Data (byte serial - 4 bits)
  - Data Gate (bilevel)
  - Response Line (bilevel)
  - Clocks (2 to 4 phases)

• ACU

- All of the above, plus ...
- Lines associated with RCU data exchange
- Master Reset (from RCU)
- Program Synch Signal (from RCU)

Not included in the above list are BITE signals, or power control signals (from RCU or IOU).

Some of these signals are unique and no consideration need be given to bussing (mostly those associated with the RCU). Most of the signals, however can be considered as candidates for an internal data bus (See Section 3.1.3).

The memory address for all memories is common. It is probably desirable to separate the IOU Function Code Address from this. The memory read data for all memories is common. The memory write data for all memories is common. The IOU output data might be combined with the memory write data at some small savings or might be separate. Similarly, the memory read data and the IOU input data may be separate or combined.

The clocks should be bussed to all users from their origin (the RCU in COPE). All of the control bilevels (mostly from the ACU) can be bussed to the blocks using them.

It was noted in Section 3.1.3 that bussing is advantageous when there are more than two senders and/or two receivers associated with a signal. This is generally the case in the processor. Also, the probable need to accomodate different numbers of memory modules (to handle different mission requirements) makes a bus structure advantageous.

Consideration might also be given to bi-directional bussing (rather than the uni-directional approach discussed above). This might provide a slight savings in circuitry, but care must be used that the data is separated in time.

### 3.4.7 Memories

The function of the memory in a processor is to store data and instructions for the control of and use in the processor operations. Although some such information is also transiently stored in the registers of the arithmetic and control unit, the memories are herein defined to mean the centralized memories serving as repositories of data and instructions which may be addressed from the ACU to read words out of (for all types of memories) or to write words into (for some types), the ACU.

These centralized memories must all, therefore, be addressable and readable; and some must be writeable. The memories are organized as a matrix of bits into so many words of so many bits each. Usually the data and instruction word sizes are made the same (see Section 3.4.2) and all memories of a given processor have the same word length. The number of words is chosen to suit the needs of each given memory type (within hardware quantization constraints).

Memories may be volatile, non-volatile, or semi-volatile. Volatility is the property of maintaining (not forgetting) all words stored, even with the power off. A volatile memory must have all words re-written following the resumption of power. A semi-volatile memory can remember for a short time (usually < 1 second) and then forgets. Such a memory is used to protect against alterations due to short power transients.

The information that must be stored in memory is of essentially three types, with differing requirements:

- Fixed data or instructions: This includes the program executive, non-variable routines and diagnostics, constants and other data that need never be changed once the system is built. This information must be non-volatile and may not be written into from the processor (some types of memories enable this information to be altered by reprogramming the memory, prior to flight, using special test equipment.)

- Variable data or instructions: This includes variable routines and diagnostics, mode information, "constants" that may need changing and general program variables. This information may be changed through the action of the processor, either as a result of the processing or through ground-derived commands to the spacecraft. Some of this information may be volatile, some should be semi-volatile, and it is even desirable that some (such as mode control) be non-volatile. Some of this information should also be "protected". That is, protected so that processor faults have a much reduced probability of erroneously writing into these portions.
- Read/Write information: This portion of the memory serves as a "scratch-pad" for relatively short term memory of the intermediate results of computations, of data inputs and outputs, etc. Such information may be recreated if it is lost and so it may be volatile. This portion of memory may be read into from ground-derived commands, but needs no protection.

The first type of information storage leads to a type of memory known as a Read-Only Memory, or ROM. This memory can not be written into by the processor. The information is entered (programmed) at manufacture or by a special process, if the ROM is of a reprogrammable type.

The third type of information storage leads to a type of memory termed a Random Access Memory, or RAM. These are also sometimes called read/write memories. This memory can be written into by the processor, or through the processor, while on the ground or in flight. This type of memory may be and is volatile and no write protection is needed (although coding can be used (see Section 3.6.1)).

The second type of information storage produces several alternative memory types. All of these memories must be writeable through or by the processor, but for some information write protection is needed and for some it is not. Also, for some of the information, non-volatility is desired or required; for some, semi-volatility seems essential; and some may be volatile. The combinations lead to the following choices:

- Volatile and unprotected - the normal RAM is suitable.
- Semi-volatile and unprotected - A RAM may be modified to provide semi-volatility by slowing the decay of power with line capacitors. This is particularly applicable where the RAM power is low, such as when MOS technology is used.
- Non-volatile and unprotected - there probably is no requirement for such a combination.
- Volatile and protected - there probably is no requirement for such a combination.
- Semi-volatile and protected - A memory type (herein defined as) a Protected Write Memory (PWM) is needed for this requirement. Its needed characteristics and design will be discussed later in this section.
- Non-volatile and protected - This memory information is the most "hardened" of the information that is writeable and should be restricted to a few words of mode information, etc. This memory type will be defined as Hard Core Memory (HCM) and its characteristics and design will be discussed later in this section also. This HCM may not be a part of the generalized processor memories, but because of its small size in number of words, and special character, may be located in other parts of the processor or system.

Another type of memory is very useful and is not covered by the previous discussions. This is the Alterable Program Memory, or APM. The APM may be used interchangeably with the ROM for ground testing and software development and test. It simulates the ROM relative to the processor, but its information contents may be changed in whole or in part from the test set by manual or tape inputting of new information. It may or may not be volatile, depending on design.

In all of the preceding discussions on memories it should be noted that volatility is not a desirable feature as such. Its presence in some types of memories can be tolerated (as noted), but the less volatile a memory is, the better it is (all other features being equal) for a fault-tolerant processor.

Other memory features of importance include the normal parameters of power, weight, volume, reliability and cost; as well as speed. Any memory should be compatible with the speed of the remainder of the processor in both reading and writing. Power is extremely important since the memories may use more power than all of the rest of the processor. Where feasible, power gating should be used to minimize average memory power consumption. This is only possible in non-volatile technologies, although some approaches permit "idleing" at reduced voltage without loss of memory between read or write accesses.

From its earliest history, the computer art has devoted a large part of the work to memory development. From the earliest drums, storage tubes, etc. up to the latest devices of today, hundreds of different memory types have been invented and used.

At the present time, however, three basically different technologies are in use, and applicable to, spacecraft systems. They are magnetic cores, plated wire, and solid state memory systems. All are relatively competitive in speed. In other respects, there are marked differences.

- Magnetic Cores - This is probably the oldest technology, but seems somewhat arrested in terms of maturity. It is intrinsically non-volatile, can be made writeable, selectively writeable, or non-writeable. It can be designed with protection or selective protection. It can thus be used as a ROM (with no APM needed), a RAM, a PWM, or a HCM (with a few tricks). Unfortunately, it requires a great deal of read and write circuitry and relatively high power while being accessed. It does not suit itself well to division into the small sizes necessary to meet long-life spacecraft reliability. To do this also imposes additional power penalties. It is relatively

expensive (mostly in the read/write circuits) and gets even more expensive as the size goes down.

- o Plated Wire - Most of the comments on magnetic cores apply also to plated wire. This is a newer, and yet more mature, technology. It is somewhat less expensive and up until recently has been the primary memory choice of spacecraft systems.
- o Solid State - Solid state memories are the newest, but the fastest evolving of the technologies. There are many different semi-conductor approaches used, each with their own relative advantages and disadvantages (to be discussed later). All of them have several common features. All viable candidates use large scale integration techniques to pack a maximum of of bits in each chip. Currently, from 256 to 8,000 bits can be obtained in each integrated circuit package. The power levels are dropping and the power per bit now has a wide range, both higher and lower than the cores and plated wire; but advantageously lower in the small memory sizes. The cost per bit has also lowered markedly, making solid-state the lowest cost, particularly in small memory sizes. The primary advantage of solid-state memories is the ease with which they can be made in small sizes, without power, reliability or cost penalties. This is particularly true of the reliability, which is very high since the redundancy subdivisions can be made almost as small as desired. An undesirable feature of solid-state memories is that the RAM's, ROM's, PWM's & HCM's are all different from each other and non-volatility in writeable versions is obtained only with some difficulty.

It appears that, for spacecraft processor applications, the solid-state memories offer significant advantages in cost, power, weight and reliability; and these advantages are rapidly becoming even greater as this technology advances. For this reason, all further discussions (with the possible exception of HCM) on memories will be restricted to solid-state approaches.

#### 3.4.7.1 Solid-State Memories

The solid-state memories are distinguished both by technology and by organization. There are a great many technologies (as for all semiconductor devices) that can be and have been used. (Also see Section 3.4.8). The two most common basic technologies are:

- Bi-polar - This is the common integrated circuit technology made up of essentially transistors and diodes of the NPN/PNP type. This family is fast and radiation resistant, but consumes relatively high power. Many memory devices are available. The TTL and DTL families are of this type.
- MOS - These metal-oxide-silicon devices are also very common. A distinction is made between NMOS (rare), PMOS (the N and P refer to the semiconductor type of the channel), CMOS (complimentary), etc. Generally speaking all are slower than bipolar, but consume less power. CMOS is somewhat intermediate between bipolar and PMOS in speed, power and radiation resistance.

Either a ROM or a RAM part can be made with any of these technologies. Either is made as an array of bit positions. Either has a number of coded address lines as inputs. Internal decoding is used to address the respective coordinates of the array. In the ROM, the array intersection consists of a diode/or equivalent gate) which, when addressed produces an output of a zero or a one.

In the RAM, the addressed intersection consists of a flip-flop (latch, or equivalent), that may be either read from or written into (set to a 1 or 0). Each type of part may be organized so that all bits are read out on one output or so the bits are in sets, with each set reading out on different outputs.

The quantity of bits of memory in a part (chip) may be different and the organization of the chips may be different, depending on the application, technology, etc. For example, ROM's normally have a density four or more times greater than RAM's in the same family. The parts may vary in organization for the same number of bits. For example, a 256 bit RAM might be organized as 256x1, 128x2, 64x4, 32x8, etc. The last number is the quantity of outputs (bit sets). The number of addresses is equal to the binary representation of the total bits. (For example, a 256 bit chip has 8 address lines). Each chip will also have a chip enable or select line for control of which chip is addressed.

The parts are available in binary progressions of bits (total) starting at 64 for RAM's and ranging (currently) up to about 1,024; and at about 256 for ROM's and ranging currently up to about 8,096.

Currently available RAM parts range from 20 to 500 usec in access time, with the faster parts consuming the most power. For bipolar RAM's the access time and cycle times are about the same. Cycle time is the time it takes to complete a read or write operation. Dynamic memories have long cycle times because of the need to refresh the data. (About every 2 milliseconds).

Static memories are to be preferred over dynamic memories for fault-tolerant processors as they are less likely to fault, do not need clocks and require fewer power supplies. They are also faster, but do consume more power and cost more.

Examples of currently available RAM's by technology type, are given in Table 3-12.

TABLE 3-12

## CHARACTERISTICS OF CURRENTLY AVAILABLE RAM'S

Technology Type	Bits per Chip	Speed (nsec)		Power(mw)per chip	
		Access	Cycle	Active*	Standby
Bipolar	64	20-50	20-50	250	250
	256	30-60	30-60	350	350
	1024	60-90	60-90	500	500
Dynamic n-channel, MOS	1024	60	180	450	60
Static P-channel, MOS	1024	300	600	450	60
Static n-channel, MOS	1024	500	500	450	60
MOS	4096	200-350	400-700	350	30
CMOS Static	256	350	350	20	0.2( $\mu$ w)
	1024	600	600	30	0.3( $\mu$ w)

\* - Read or write

Any of these speeds are fast enough for a COPE-type processor, which requires a cycle time  $< 1 \mu$ sec. (For parallel read/write).

The ROM parts may be either factory or field programmed. In factory programming, a mask is used in the chip fabrication, which contains the interconnects defining the 1 and 0 bit pattern desired. Once built, the part can not be changed.

Field programmable ROM's, or PROM's as they are termed, are built with an array of diodes in such a way that the chip may be placed in a machine and the individual diodes are addressed and a high current is used to "burnout" those diodes not desired, thus programming the chip. This process can be highly automated. Such programmed chips are not generally re-programmable, although individual bits can be changed (in one direction only). Erasable and re-programmable ROM's are also now being developed.

Currently available PROM's are listed in Table 3-13

TABLE 3-13

CHARACTERISTICS OF CURRENTLY AVAILABLE PROM'S  
(ALL BIPOLAR)

Total Bits	Organization	Access Time ( $\tau$ sec)	Power/Chip (mW)
256	32 x 8	50	500
1024	256 x 4	60	650
2048	512 x 4	70	650

Devices have been announced using MOS technology, ranging up to 16,384 (2048 x 8) bits, but are not programmable. Other MOS devices are available with fewer bits that are programmable. Some of these can even be reprogramed using ultra-violet light for erasing.

A new type of memory device using amorphous semiconductor technology has been developed. These devices exhibit a non-volatile memory and are effectively re-programable ROM's. The re-program or write process is quite slow (10-15 m sec) and involves relatively high currents into low impedances. The reprogramming can occur indefinitely ( $>10^7$  times) and the non-volatility is dependable.

The device is currently available only in a 16 x 16 bit chip. Read times of approximately 50 nsec can be obtained. The outputs are not directly compatible with TTL circuitry

It would appear that this device could find application as either a replacement for PROM's or as a protected write memory (PWM). In the first case, the advantage would be the ability to reprogram the memory on the ground. In the second case, the amorphous memory could provide a desired non-volatility for those portions of memory which must be changeable and yet not easily loseable.

Another approach to non-volatile memory devices is the metal-nitride-oxide-silicon (MNOS) technology. For low values of applied voltage the device operates like a conventional P-channel MOS transistor. For high gate voltages an alteration of the internal electrical charge occurs, altering the turn-on voltage. This charge can be retained for several years, providing an effective non-volatility

They are relatively fast, providing write times of 1 - 100  $\mu$ sec. Read times are similar. The write operation involves an erase of all bits, followed by a selective write of 1's.

A currently available device is 8 x 8 bits in a 24 lead package. Writing requires bipolar voltages  $\sim$ 35 volts. Some difficulties have occurred with compatibility between these devices and either MOS or TTL circuits.

This MNOS device might also be used either as a reprogrammable ROM or as a PWM. At this time it is not felt that either MNOS or amorphous approaches are developed to the point that their use could be recommended.

The technology that is chosen for each of the memory types must be compatible with the requirements for:

- Speed
- Power consumption
- Radiation resistance
- Volatility considerations
- Chip density
- Cost
- Package style

#### 3.4.7.2 Memory Organization

Each of the memory blocks, of whatever type, must contain many words of information. This will generally require several memory chips of either the RAM or ROM type. (We will see how a PWM can be made up of RAM chips). Also required will be chip address decoding circuits, any parity or other error detection forming/checking circuits needed, and

any redundancy cross-strapping/bus interfacing/ power control circuits needed. As the memory size increases, the memory chip count increases proportionately, but the remaining circuits do not, tending to make the larger memories more "efficient" in terms of overhead.

On the other hand, the memories should not be made larger than needed and reliability restrictions on element size (see Section 3.1.1 ) may dictate smaller sizes. Also note that if (for example) the chip organization is  $256 \times 1$ , no smaller than a 256 word block can be built (etc.).

Note that power gating may be used with any non-volatile memory. An enable signal is used by the processor (ACU) to bring the power up to the memory block (or only a group of chips) just prior to a read (or write) operation. This can save a great deal of power as the duty cycle of access is often quite low.

Another aspect of the memory organization is the manner in which the memory address, inputs and outputs interface with the remainder of the processor. These interfaces may, in general, be serial, byte-serial, or parallel. How these interfaces are organized will have major effects on processor speed. Generally speaking; serial interfaces will be seen to be too slow for most applications.

Although the COPE processor was designed originally for byte-serial interfaces for data entry and exit from RAM, this is now seen to be an unfortunate limitation, and new COPE RAM blocks have been designed for fully parallel addressing, data entry and exit, with much faster operation resulting. The COPE ROM's were always designed for parallel addressing and readout. Note that even if (as in COPE) the processor is designed for byte-serial arithmetic, the memory read/write functions may be (and probably should be) in parallel. The use of parallel RAM entry and exit in COPE also permits the use of CMOS technology, drastically reducing the RAM (and COPE) power requirements.

The lower power requirements of a CMOS RAM (which are extremely low  $\sim 1$  mw for 8000 bits) when the RAM is not being accessed, permits semi-volatility to be economically added, simply by increasing the size of the filter capacitors. Capacitors of 600 uf total will give one second of memory retention, while still maintaining the voltage above 90% of nominal. These capacitors must be installed "inside of the bus or cross-strap interface circuits so their charge does not leak back into the rest of the system.

It was seen earlier that a protected write memory (PWM) capability was needed. This protection should reduce the probability of writing in a word:

- In the wrong address
- Incorrectly in whole or part
- Incompletely.

Only critical program, mode, etc. words should be put into PWM to restrict the size. The PWM should also be semi-volatile. The PWM should use parallel write so that the words either get in or do not, whether the clocks stop or whatever. (In byte-serial or serial entry, words might be written incompletely.)

The PWM may be achieved by modifying the standard RAM organization, as follows:

- Address error can be protected by using a dual address entry technique. An enabling address is first sent in, which is stored in a register in the PWM. This is followed by the "regular" address. The regular address is bit-by-bit compared (using exclusive or gates) with the stored address, and if they are the same, is passed on, enabling the write. A BITE signal may be issued if they are not the same. Note that the enabling address could be shorter than the regular one, comprising a sort-of "code word for enabling.

- o Write word error can be protected using a similar process. The word is sent in twice, with the first entry being stored in a register in the PWM. The words are bit-by-bit compared, and only if they are identical is the word written. Again a BITE signal could be issued if they are different (indicating to the processor the need to try writing again).
- o The PWM thus needs two extra storage registers of the required word length, plus the comparison logic. An additional input would be required so that the enabling address and word would be distinguished from the "regular" ones. (PWM enable.)
- o Although the PWM could contain both conventional RAM (un-protected write) and protected write functions in the same unit, the protected write portion should be segregated so that conventional write can not occur into it. The read function can be the same for both portions.
- o The PWM should have only about a 15% part penalty over a conventional RAM (in the 512 word x 16 bit size).

Note that the key is to prevent words from being erroneously written. Writing destroys old information, which will be more correct than erroneous information. The idea is that it is better to retain the old, than to destroy it completely, if the new can not be trusted.

Reading out of the information may also be done redundantly, under software control. This is done by sequential read out instructions. The read-out information should then be compared (by software control in the ACU) for identicalness prior to use as an instruction or data word.

It may also be desirable, for the most critical information, to store it at two or more redundant addresses. This information should also be compared prior to use.

If a protected memory is achieved only by writing in redundant addresses, this is not as good a protection since one word can be written (replacing the old word), and then the second word could be in error (replacing that old word). Now we have two words that are different and no good way of knowing which is correct. If three words are written, the protection is much better, etc.

To achieve Hard Core Memory (HCM), which is writeable and non-volatile and protected, is somewhat more difficult. This could be done by using all of the techniques described for the PWM, but using a different type of circuit for the RAM's chips. This could be technologies such as amorphous, MNOS, or it could be magnetic in nature (cores, wire latching, relays, etc.). For the HCM, bits instead of words might be stored and the HCM might well not be organized like the other, conventional, memories. The only information needed in HCM should be the processor operational mode that is current. Note that the system configuration status is stored in the power control latching relays, if they are used.

Also note that the PWM protection is not absolute and erroneous words may still be written, although the probability of this is considerably reduced.

From a redundancy standpoint, note that the use of ROM's, PWM's and RAM's, provides some characteristics to be aware of.

- If more than one ROM element type is used, they are different and non-interchangeable.
- If several PWM's or RAM's are needed to be in use at one time to contain the information required, then these are interchangeable (within the PWM or RAM category). That is, any RAM available can fulfill the function of any RAM required. (Example: Suppose two RAM blocks are needed to provide sufficient memory, out of 4 available. Any 2 can do the job.)

- o Because of this last characteristic, the RAM's (or PWM's) must be provided with information from the part of the system controlling reconfiguration as to what address they each are. This is called the page number.

Sometimes processors are organized so that the IOU and/or peripherals have access directly into the memory (not through the ACU, as on COPE). This is done to reduce access time for telemetry/commands or peripheral data. It is felt that this is not necessary for spacecraft control system application. It does impose hardware penalties if it is used, as well as software penalties (to provide non-interference between ACU and other access). It is also not compatible with a synchronous program structure.

In summary, the true processor memories that are needed are:

- o ROM - for program storage of executive, and unchanging routines and constants.
- o RAM - for read-write and other volatile, short term storage (although it might as well be made semi-volatile).
- o PWM - for variable program and data storage, which should have protected write and semi-volatile status. It was seen that a PWM is a RAM with a few additional circuits.

#### 3.4.8 Technology

Technology, as used here, refers to the devices (parts, components) used in the processor design. These devices have two technological dimensions:

- o Physical Technology
- o Topological Technology

The physical technology refers to the organization from a semiconductor electronics standpoint of the devices in a family. Most devices are

either bipolar or channel semiconductor devices. Bipolar devices may be TTL, DTL, etc., depending on the organization of the n and p layers into diodes and transistors and thence into gates, etc.

The channel semiconductor devices are mostly MOS (metal-oxide-silicon) and may be n-channel, p-channel, complimentary, etc., resulting in NMOS, PMOS, CMOS, etc. The MOS devices are usually slower and less radiation resistant than bipolar devices, but they consume much less power. There is no real difference in reliability.

The choice of parts from a physical technology standpoint must be made with regard to the speed/power/radiation susceptibility standpoint. There is little cost difference. For most control system processor applications, MOS parts are adequate for use, although bipolar is better in every respect except power.

Both basic families of physical technology are about equally applicable to higher scale integration (more gates or equivalent per package), although the power dissipation of bipolar can serve as an eventual limit.

The technologies can also be mixed in application if care is used in the interfacing. Generally, the bipolar parts operate from +5 VDC, while the MOS parts may use +5V, as well as a higher positive voltage and/or a negative voltage. Circuits have been developed (as both discretes, IC's and parts of other IC's) to provide interface compatibility, while retaining adequate noise margin.

The topological technology refers to the organization from a chip architecture standpoint. That is, from the "block diagram" standpoint of what functions the part contains. Generally, this is independent of the physical technology used.

Parts have been characterized relative to their degree of integration. Currently, small scale integration (SSI) refers to parts with 1 to 6 gates or 1 to 2 flip-flops, or equivalent, usually in a 14-lead package. Examples might include:

- o Hex inverter
- o Quad 2-input NAND Gates
- o Quad 2 input NOR Gates
- o Quad Exclusive OR Gates
- o Triple 3-input NAND Gates
- o Dual 4-input NAND Gates
- o 8-input NAND Gate
- o Dual and/or/invert gates (and others)
- o Dual JK flip-flop (also type D and RS)

Similarly, medium scale integration (MSI) refers to parts, still usually in 14 to 16-lead packages, having equivalents of 10-50 gates and/or 4-16 flip flops. Some current examples include

- o 4-bit shift register (also 8-bit)
- o 4-bit counter (also 8-bit)
- o Decoders (1 of 10 and 1 of 16)
- o Dual 4-bit latch
- o Dual 4 input multiplexer (also quad 2 input)
- o 8 input multiplexer
- o Comparators (4 and 5 bit)
- o 4-bit arithmetic element
- o Parity generator/checker
- o Monostable multivibrator

Large scale integration (LSI) refers to parts containing still more circuits, almost without limit. These parts are usually in 24-, 40- or 64-lead packages; one of the principle restrictions being the number of leads available.

The LSI parts that are non-custom are mostly for memories (see Section 3.4.7). The use of custom LSI is discussed in Section 3.8.

In the bipolar technologies (and to some extent for MOS), there is a speed/power trade also available. For example, in the common bipolar TTL 54/74 series, there are available (but not for all topologies) a range of five speed/power possibilities. The are:

•	54/74	H	High Speed	6 nsec	22 mw/gate
•	54/74	S	Schottky	3 nsec	19 mw/gate
•	54/74	-	Standard	10 nsec	10 mw/gate
•	54/74	LS	Low Power Schottky	10 nsec	2 mw/gate
•	54/74	L	Low Power	33 nsec	1 mw/gate

The "54" refers to the broader temperature range (-55°C to +125°C) and the "74" to the narrower (0° to +70°C) range. This particular family is available from several suppliers and the range of parts are also compatible with the Fairchild 93L series, which contains some particularly useful MSI parts.

The interfacing of parts from different families (of technology or speed/power) must be done with proper awareness of the loading rules, as they are often defined differently and must be properly equated. Sometimes higher power (and faster) parts are used for their drive (fan-out) capability of driving large numbers of lower power circuits.

A processor could be designed using nothing but 2-input NAND gates. The use of more part types makes the design job easier and reduces the part count and the power. On the other hand, the use of too many different part types is extravagant, since (generally) each part type must be separately qualified and quantitative discounts might not be realized. Usually, a part type should not be used unless a sufficient number will be used (or there is some other, over-riding reason). The number of part types used is one more tradeoff in the design.

In summary, the choice of the parts used in the design should be based on a tradeoff between:

- o Speed requirements
- o Power consumption
- o Fan-out capability
- o Logic types available
- o Package style
- o Availability (number of vendors)
- o Cost

IC parts are usually available in DIP's (Dual-in-line packages) or in flat-packs. The former are usually better for breadboarding and the latter for flight-use. Flat-packs occupy less volume and are more suited to multi-layer board use.

While the preceding discussions have been concerned with integrated circuits, it should be recognized that some number of discretes will also be necessary. This use should be minimized. Note that resistors are available in IC-like packages. The choices of other discrete parts should be guided by the normal spacecraft high reliability part requirements.

The COPE processor design chose to use a mixture of SSI and MSI parts from the 54/74 families, and the 93L and 9L families. Three Schottky device types were used, 8 of the standard line, and 28 of the low-power line. In a redesign, it is anticipated that many of the standard lines would be replaced by the low-power Schottky devices (that were not available at the time of the initial COPE design) at a power savings. This total of 39 IC-part types (not including memory chips, A/D converter IC's and discretes) is larger than desirable. It should be possible to design a processor using current SSI and MSI parts with less than 30 different digital IC part types.

#### 3.4.9 Architecture

This section will concern itself with the topological organization (architecture) of the arithmetic and control portion of the processor. It will be assumed that the memories and the input/output portions are separate from the arithmetic and control portion, but connected to it by means of an internal data bus. It is further assumed that the IOU contains buffering registers for all input and output data.

The arithmetic and control unit (ACU) provides the central arithmetic processing, logic processing, and control for the entire processor. It provides memory and I/O addressing: cycle, program and micro-timing; instruction decoding; and the necessary registers, counters and logic.

To perform these functions, the ACU must contain one or more adders or computational blocks, together with a number of registers (or counters) to temporarily contain data or instructions during the operations. A minimum set of registers for a processor using indexing and address modification (see Section 3.4.2 ) is:

- Arithmetic (or "A") register
- Quotient (or "Q" register
- Address register
- Instruction register
- Memory register
- Data base register
- Index register

Counters that are needed are:

- Cycle counter
- Program counter
- Micro-timing counter

Logic and gating are necessary for selecting and routing data between the inputs, outputs, registers, counters and adders and for decoding instructions.

A block diagram of the COPE ACU is shown in Figure 3-41 to enable an understanding of a typical (non-micro-programmed) ACU. Note that COPE is organized for byte-serial arithmetic and the instructions discussed in Section 3.4.2.

The adder can provide addition, subtraction, complementing and logical operations. It is used for these operations not only on data, but it is also used to modify addresses (indexing), and for all byte-organized data transfers within the processor. Its two inputs come from the two data-selectors. Its outputs can be directed to the A-, Q-, memory-, or index registers and the program counter. The adder also contains carry and overflow flip-flops.

The data selectors are collections of logic which route data into the adder. Data selector-A accepts data from the A-, Q-, or index-registers and the program counter. Data Selector-B accepts data from the A- or Memory registers and from the data bus. These selectors are one byte wide.

The A-register is the primary arithmetic register and is 16 bits wide. It can shift right or left by one bit or one byte. Serial data can enter from the Q- register or from the data bus. Byte data can enter from the adder. Outputs are to the Q-register (serial) or to either of the data selectors (byte).

The Q-register is the secondary arithmetic register. It is used for double precision, multiplication, division and some other operations. It has the same shift capabilities as the A-register. It can receive inputs from the A-register (serial) and the adder (byte). It has outputs to the A-register (serial) and data selector-A (byte).

The index register is used to modify memory and I/O addresses. It can operate as a down counter or a shift register, being loaded from the adder.

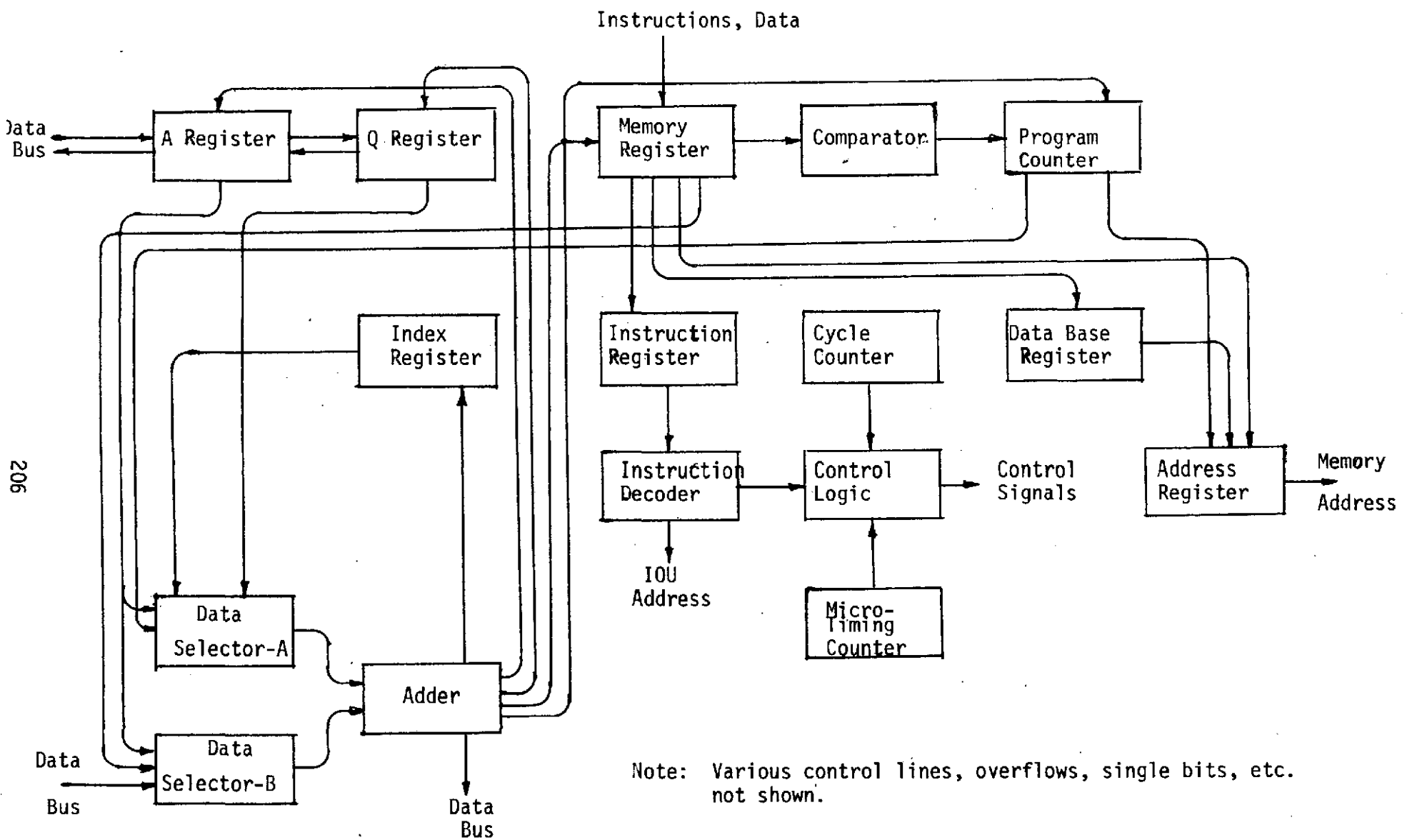


Figure 3-41  
Typical ACU Block Diagram

The memory register receives the instructions or data from the memories. It can also be loaded from the adder. It has outputs to the instruction register, data base register, and address register, as well as data selector -B. The instruction register serves as a temporary storage for instructions from the memory register. The instruction decoder provides the decoding of these instructions into the control logic.

The program counter is an up counter which is incremented upon each instruction. Its output, through the address register, is used for addressing. It can be loaded from the adder or the memory register (through a comparator).

The data base register provides the operand address to the address register. It can be loaded from the memory register. The address register supplies the memory addressing.

The cycle counter is used to count operations performed in multiply and divide and shifts applied in shift or serial instructions. The micro-timing counter generates a sequence of "T" times that control the events occurring in the execution of instructions. The number of states of the counter is dependent on word length and arithmetic organization. The counter starts at its initial state on each instruction and goes through its states in progression, routing data, etc. under the control of the control logic.

Other, somewhat different, organizations may be used for the architecture of the ACU, of course. The problem remains one of organizing data and instruction flow and processing between the adder (or adders), registers and counters. Tradeoffs exist between the number and versatility of the registers (more can be used than the minimum discussed above). These tradeoffs become between the circuitry involved (and its attendant penalties) and the possibly increased speed and/or programming versatility possible.

The next section will discuss another type of architecture (or improvement in the conventional approach), called microprogramming. In Section 3.8, the use of LSI (as applied to the ACU functions) is discussed, including the application of so-called microprocessors.

#### 3.4.10 Microprogramming

It was shown in the last section how the architecture of the ACU portion of a processor is conventionally organized. This was seen to be a collection of relatively special purpose registers and counters, connected to each other, the adder and inputs and outputs by hard-wired predetermined logic. The logic is all unique and consumes a major part (60-70%) of the total part count (and power). The logic is completely determined by the instruction set.

Just as a programmable set of general purpose electronics (a processor) has advantages of versatility, cost and power over the special-purpose electronics it replaces; so can a programmable set of general purpose electronics (microprogramed) within the processor have advantages over the special-purpose equivalent.

Microprogramming relates to several concepts. It relates to the use of the ACU registers and counters as general-purpose (or more general-purpose) blocks. It includes the concept of having the interconnecting logic be interchangeable (usually consisting of LSI chips) to provide completely different instruction sets by changing this logic. It includes the increased possibilities of speed and/or programming versatility brought about by the use of general-purpose registers.

A so-called microprocessor need not necessarily be micro-programmed. A micro-programmed processor need not necessarily be built using LSI (although it usually is, at least for the "logic" portion). The logic could be built of conventional SSI or MSI, but organized in a micro-programmed way. It could be built on plug-in circuit boards, which could be interchanged to provide different program sets.

For spacecraft attitude control applications, the ability to more easily change programs is not particularly important since the instruction sets are usually the same for all programs. More important is the capability to have increased versatility in the registers. This can improve speed since many intermediate results may not have to be stored in RAM. In

fact, it is somewhat like putting a part of RAM into the ACU. It can also (to a slight degree) simplify the software programming.

The most important likely benefit can probably be gained by converting the logic to LSI, using ROM chips, programable logic arrays (PLA's), or configurable gate arrays (CGA's) or equivalents, as discussed in Section 3.8. This can save parts, power, and cost and still provide the other benefits of microprogramming discussed earlier.

For COPE, approximately 110 IC's in the ACU could be replaced with 10 to 20 PLA's or CGA's. This would save approximately one watt of power.

### 3.5 Software Design

This section treats the software design for a fault tolerant programmable digital processor for attitude control system applications. The previously-developed COPE (Control Processing Electronics) processor, along with the software system discussed here, are considered for application to the three-axis attitude control of spacecraft in deep space flight for missions with lifetimes of up to 10 years.

An important consideration in the overall design of the fault tolerant control system is software design because of significant implications on system reliability, operational flexibility, cost, complexity and hardware requirements. Due to long round-trip communication times to the middle and outer planets of the Solar System, the control electronics must operate with a high degree of autonomy. This imposes stringent requirements and constraints upon both hardware and software fault detection, isolation and reconfiguration procedures, which must control recovery so as to insure control system reliabilities on the order of 0.9 over 10 year lifetimes.

The discussion that follows gives an overview of key software design considerations and tradeoffs and describes a recommended executive program concept for the control electronics processor based on the failure detection and reconfiguration approach recommended in Section 3.7.

#### 3.5.1 Software Requirements and Design Criteria

The selection of a software organization for multiple applications, providing fault tolerance and autonomy, is governed by the following general requirements:

- Flexibility is required in that the design must be adaptable to a variety of mission requirements.
- Reliable failure recovery is necessary, yet in-line diagnostic overhead should be as small as possible.
- The set of diagnostics chosen should be powerful enough to ease hardware diagnostic requirements.
- The program execution structure chosen should have a well-defined behavior, so as to increase operational reliability and facilitate programming, checkout and verification. A predictable sequence is

essential for real-time operations in a fault-tolerant environment. Thus, conditional interrupts causing execution sequence changes should be minimized.

- The number of operation modes possible in each mission regime should be minimized in order to reduce execution verification complexity, minimize the probability of accidental mode transfers, and increase recovery speed.
- Critical interfaces should be removed from the digital processor (e.g., dedicated control electronics should be provided for any high bandwidth loops as this will significantly ease the rest of the burden on the digital processor)
- The execution of a second job should not be permitted until a first job that failed has been successfully retried. This implies that machine capabilities must be restored before successful re-try is possible.
- Reconfiguration during recovery must be under the ultimate control of some hardware "Hardcore", which should be one of the most reliable elements of the system.
- Only the minimum autonomy required should be provided for each mission phase (in order to increase simplicity and predictability)
- A means of providing fail-safe mode storage and transfers is necessary.
- Program execution should be divided into small segments in order to minimize the propagation of faults. Frequent processor status and program execution checks, along with careful preservation of status and rollback data, are also important in that regard.

### 3.5.2 Program Organization and Modularity

A reduction in the cost of onboard software, together with an increase in reliability, is realizable through the utilization of modular software structures. Cost savings result from a reduction in the verification and modification portions of software development. Greater reliability is achieved through the well-defined properties of the modular structure. Included in these properties is the explicit definition of software module interfaces. The following summarizes the prime reasons for modularization:

- Systematic program construction is easy to understand, debug and validate
- Internal interfaces are well-defined
- Executive functions can be isolated from functional coding
- Eases the program fault detection and code replacement problems in a fault-tolerant environment.

Desirable features that each application module (AM) should have are:

- Meaningful functional responsibility
- Execution time consistent with executive constraints
- Minimal argument transfer requirements

Figures 3-42 and 3-43 illustrate two methods for linking programs which have been modularized. In the structure of Figure 3-42, the burden of providing self linking falls upon the applications programmer (including providing paths to redundant AM's not shown). In Figure 3-43, an executive (or scheduler) program provides the module linking. Note that a larger number of redundant copies may be kept of certain AM's than others when they are more critical. The structures of both Figure 3-42 and Figure 3-43 will now be discussed in more detail.

A straightforward approach to the sequential program organization having a fixed structure has been illustrated in Figure 3-42 where the sequence of operations is controlled by pre-programmed mode switches and logical tests. The flow is continuous in the sense that the execution of a given path proceeds automatically from one stage to the next one without interruptions or external actions. Due to the sequential nature of the program flow, multiple exit and re-entry points are required to provide capabilities for program or parameter changes. Also, the interconnected structure of this type of program may complicate the switching of redundant units (during reconfigurations) if paged memory units are required to store the entire program. Since there is a high risk of failure of the entire program cycle due to a single fault, a relatively large number of hardware diagnostics and checks will be required.

In the typical modular program structure with an executive (Figure 3-43) however, each program module is isolated from the rest and can be entered only through the executive. After a module is completed, program execution

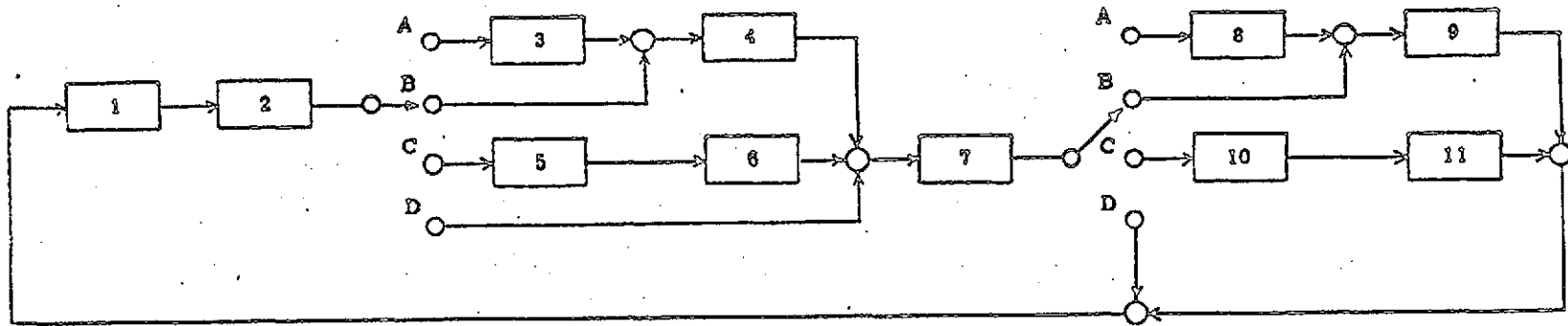


Figure 3-42 Self-Linked Modular Program Organization

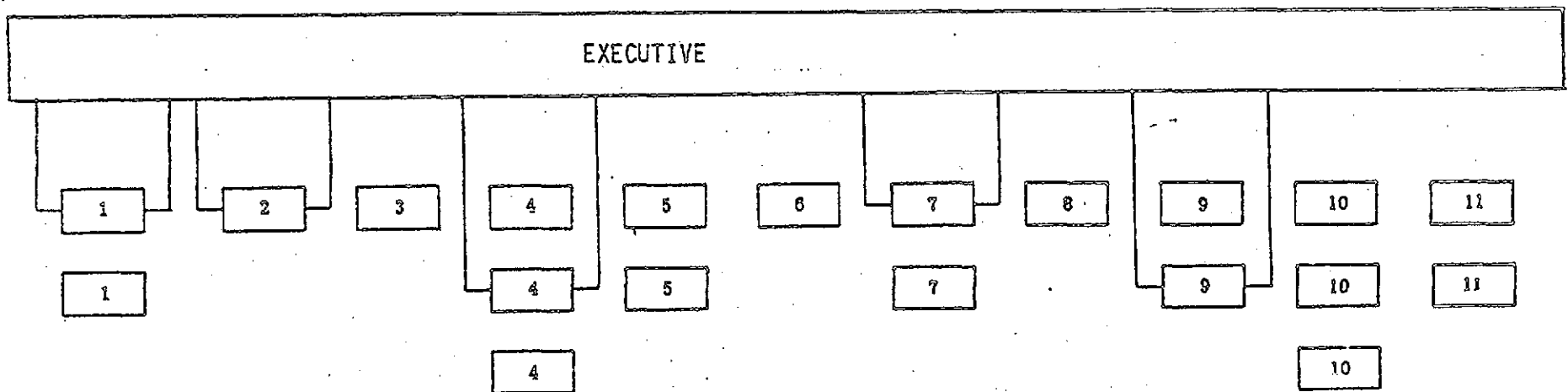


Figure 3-43 Modular Program Organization with Executive

is transferred back to the executive by either a software instruction or the hardware. The executive includes diagnostics and tests for verification that the system is operating properly before program execution is attempted. Also, it contains the scheduling software to direct program execution to the modules required by each mode with the appropriate frequency and sequencing. Scheduling can be accomplished simply and reliably by means of task tables, in which case a mode is defined by its task table (or job control module).

Each task table contains the addresses of the program modules to be executed and data determining the respective frequencies of execution and sequencing. Program modifications can be made very easily by changing parameters in the task tables. Failure of a program module will not necessarily cause failure of the entire program. Another important feature of this approach is the simplification attainable in the process of recovery from system failures due to the fact that only the executive is required to determine whether the machine is operational. A modular program structure with an executive and hardware reset is preferred because of the following reasons:

- Reliability
  - Memory reliability is improved, since partial failures can be bypassed by changes in the task tables. Replacement modules can be stored in other memory locations or other memory units.
  - Program module redundancy can be selective, i.e., there may be modules that need not be redundant, while others may need to be replicated more than by the number of available memory units.
- Program Development
  - Entire programs are laid out in a systematic manner and internal interfaces are well-defined.
  - Checkout and program validation is facilitated because basic functions can be "debugged" without requiring use of the entire program.
  - Work can be conveniently divided among several programmers, not all of whom are required to understand the entire program.

- Software for hardware interfaces may be developed by hardware specialists.
- Executive functions can be isolated from functional coding.
- Changes are readily facilitated since often it is only necessary to modify small portions of the program that do not themselves interface with the remainder of the program.
- Internal interfaces can easily be maintained.
- A high degree of management visibility is provided since documentation is relatively simple, straight-forward and readily understood. Program capabilities are readily determined.

o Program Simplicity

- Simplicity is greater because mode switching is reduced to a minimum, exit and re-entry points for program changes are eliminated, and tests required to monitor the program execution sequence are simplified and are more systematic.
- The number of modes (task tables) to be permanently stored in memory is small since these must be only the ones required during periods of critical or prolonged autonomous operation (e.g., cruise, maneuver, encounter, backup, and initialization). However, the system can be made to operate in any number of modes without increase in program complexity by simply changing task tables by command.

o Operational Safety

- The probability that a single failure may cause a system down condition can be minimized by forcing an unconditional return to the executive after completion of each program module. This allows the executive to perform tests to verify whether the machine is in operational condition before proceeding with the execution of the next program module. If a failure is detected, immediate action can be taken to bring the machine back up. If a program module fails, execution will be repeated and, if required, additional re-configurations can be made until this module (or a redundant one) is successfully run. This minimizes the probability of recurrent faults.

- The risk of accidental mode changes due to faults is reduced by operating with a minimum number of modes (task tables) in memory.
- Sensitivity to Faults
  - Effects of a fault can be limited to total failure of a single program module run. This would no longer imply a complete program execution failure.
  - Execution of a given module can be repeated easily until a successful run is obtained.
  - Diagnostic tests performed by the executive on a regular basis minimize the sensitivity to faults since corrective action can be taken before entry to the scheduling routine is made.
- In-Flight Reprogramming and Parameter Changes
  - Reprogramming can be done very simply by ground command, since it only requires inputs of additional modules and/or new task tables.
- Diagnostic Overhead
  - Overhead can be minimized because a hierarchy of diagnostics can be easily established.
  - The executive includes only the diagnostics required to verify the correct operation of essential system units.
  - Special purpose tests required during program execution (e.g., peripheral device tests and verification of certain I/O functions) can be stored either as separate program modules or as parts of the program modules where they are needed, depending on the duty cycle.
  - Housekeeping and preventive maintenance tests can be scheduled by the executive on a priority basis.
  - More detailed diagnostics and off-line tests of failed units can be performed under ground control by means of special operation modes (required task tables input by ground command).
- Recovery
  - The speed of recovery from failures can be improved by the diagnostic capabilities of the executive, if reconfigurations can be performed in response to software instructions.

- Hardware-controlled return of program execution to the executive provides an automatic roll-back capability and simplifies and speeds the recovery process.
- Executive diagnostics minimize reconfiguration of equipment not essential for restoring software control of the system (e.g., I/O equipment, memory units not containing the executive and peripherals, can be switched by executive request.)

### 3.5.3 Executive Tradeoffs

In a fault-tolerant processor, the executive is the software responsible for top-level diagnostic and recovery functions and for program module sequencing and control. In addition to providing capabilities to change modes of operation, the executive handles interfaces between program modules, updates system state and equipment status information, schedules high priority functions, and controls in-flight program changes.

Real time digital processor operation with a modular program organization will consist of a sequence of minor and major cycles (see Figure 3-44). Each minor cycle is a machine run beginning with a series of executive diagnostics, proceeding with the execution of the corresponding program module(s), and concluding with the performance of software tests (for determining whether the run was successful or not) and a hardware return to the executive. A major cycle is the sequence of minor cycles to be periodically executed in each mode.

A breakdown of the operation into minor and major cycles has the following advantages:

- Failure of a minor cycle will not cause failure of the entire major cycle.
- Rollback to the program module that failed can be implemented easily.
- A hierarchy of diagnostics can be established with minimum overhead. Essential system tests are conducted by the executive at the beginning of each minor cycle. Performance tests are made at the end of each program module. Other diagnostics and tests not required on a periodic basis can be included in a specific program module which is executed as part of a major cycle only when it is required.

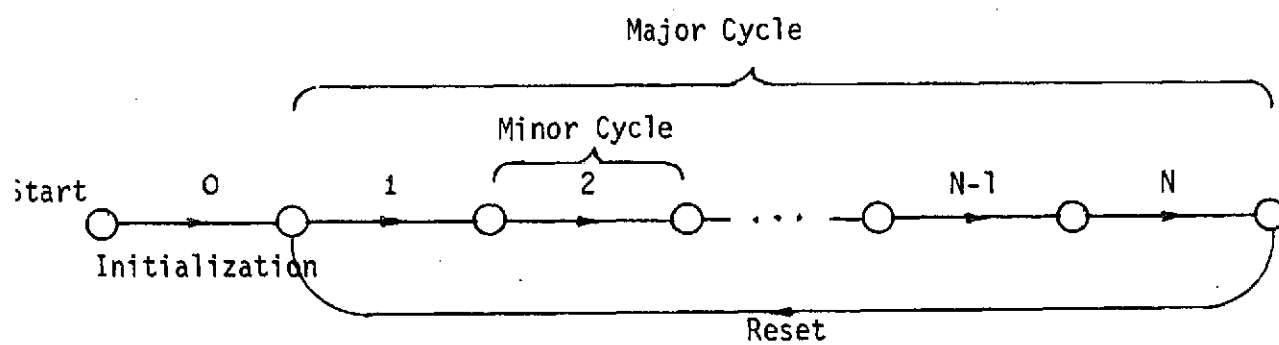


Figure 3-44 Flow Graph Showing Real-Time  
Digital Processor Operations Breakdown  
with Minor and Major Cycles

The execution of a program (organized in major and minor cycles) can be made in many alternative ways, depending on whether its cycles are of equal duration or not, and on whether major cycles consist of variable or fixed sequences of minor cycles. In this Section, the following approaches to program execution are considered and compared.

- o Asynchronous
- o Synchronous
- o Hybrid
- o Synchronous with asynchronous overlay

A modified version of the synchronous executive with asynchronous overlay is preferred because it has most of the advantages provided by the synchronous type of execution while retaining the flexibility and fault-tolerant features of asynchronous operation.

#### 3.5.3.1 Asynchronous Executive

Figure 3-45 shows an example of asynchronous program execution where the sequence of tasks (minor cycles) can change on a priority basis. Minor cycle durations may also be variable. The asynchronous executive provides a very high degree of flexibility since diagnostics and reconfigurations can be performed without time constraints whenever they are needed. Jobs that fail may be repeated until they are successfully completed. Asynchronous execution leads in general to highly efficient utilization of a processor, but suffers from the following disadvantages:

- o Complexity of the executive due to priority structure and timing constraints.
- o Complexity of program debug and checkout procedures due to large number of system states
- o Unpredictability of future system state

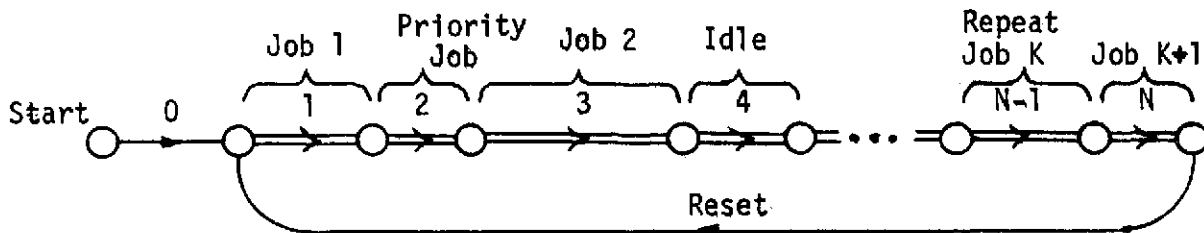


Figure 3-45. Asynchronous Program Execution - Minor Cycle Durations and Number of Minor Cycles Per Major Cycle May be Variable.

### 3.5.3.2 Synchronous Executive

In a synchronous executive, each minor cycle has a specific job assignment and the same duration. Major cycles consist of the same number of minor cycles and no sequence changes are allowed. As shown in Figure 3-46, optional jobs can be performed only during those minor cycles specifically reserved for external assignment (ground command or on-board priority scheduler).

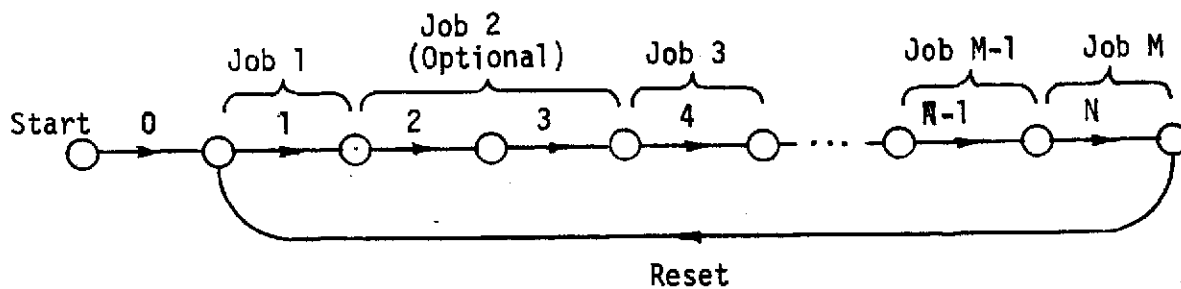


Figure 3-46. Synchronous Program Execution - Number of Minor Cycles and Job Assignments Within Major Cycle are Always the Same - No Sequence Changes Allowed

Synchronous execution has the following advantages:

- o Program simplicity and reliability. There are neither priority problems nor timing constraints to worry about during program execution.
- o Debugging and checkout are simple because the sequence of operation is almost always the same and events occur in an orderly manner. Tests can be performed without affecting the normal sequence of operations.
- o Real time operations are facilitated because future system states are predictable.

One of the main disadvantages of a synchronous executive is its relative inflexibility. The efficiency is lower because a number of minor cycles must be reserved for diagnostics and external demand jobs. A strictly synchronous program execution is not compatible with a machine where minor cycle failures are allowed, for repetition of failed cycles is not allowed.

#### 3.5.3.3 Hybrid Executive

There are many ways in which a hybrid executive can be implemented. Figure 3-47 is an example where every other minor cycle is reserved for synchronous operations. Synchronous tasks are assigned and scheduled as in a synchronous executive. Asynchronous functions are interleaved with the synchronous ones and are scheduled on a priority basis. The hybrid executive includes an asynchronous scheduling routine which is simpler than the one required in a purely asynchronous executive because only a fraction of the system functions are managed on a demand or priority basis. This approach provides greater flexibility (at the expense of a greater program complexity), but still the problem of compatibility with a fallible machine is not completely solved. Minor cycles may fail more than once and, consequently, there may be instances where the buffering provided by asynchronous cycles is not sufficient. Hybrid execution has the advantage of providing improved debugging and checkout capabilities (with respect to the pure synchronous case) because auxiliary routines for debugging and checkout may be run without affecting the sequence of synchronous functions. Naturally, the utilization efficiency cannot be as high as with a purely asynchronous executive unless asynchronous cycles can be used for detailed diagnostics and preventative maintenance when there is no demand for higher priority functions.

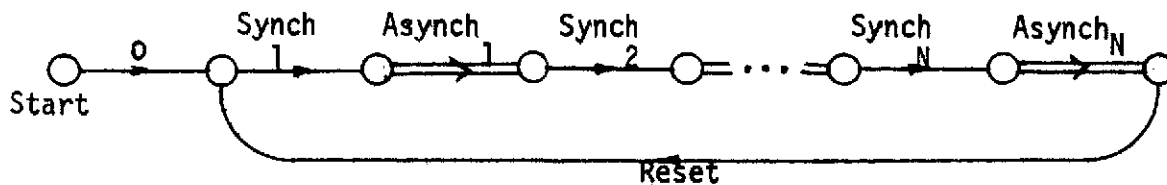


Figure 3-47 Hybrid Program Execution - Constant Number of Minor Cycles per Major Cycle - Every Other Minor Cycle Reserved for Synchronous Functions - Asynchronous Functions are Performed on a Priority Basis During Asynchronous Cycles.

#### 3.5.3.4 Synchronous Executive with Asynchronous Overlay

As shown in Figure 3-48, each minor cycle is assigned specific synchronous functions and, also, asynchronous operations. Asynchronous assignments are made by a scheduler routine on the basis of priority and availability of time within each cycle. In some cases, this executive can be more flexible and efficient than a hybrid, but, its compatibility with a fallible machine is not so good since failure of a minor cycle cannot be corrected without affecting the sequence of operations. However, as will be discussed later, the loss of one minor cycle time during recovery from a fault can be tolerated (counters can be reset or masked). Except for these potential differences in efficiency and compatibility, the main advantages and disadvantages of the synchronous executive with asynchronous overlay are similar to those of the hybrid executive.

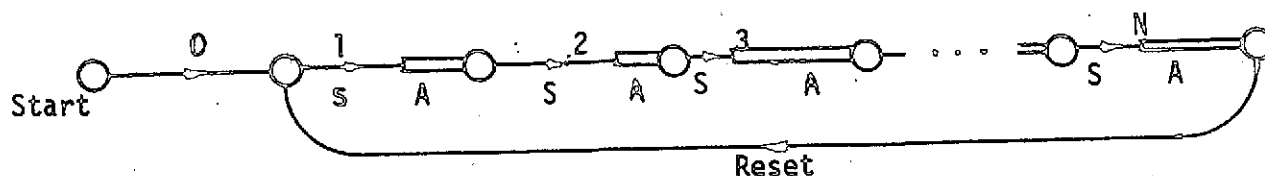


Figure 3-48 Synch Execution With Asynch Overlay - The Organization is Synchronous with Fixed Number of Minor Cycles - Asynch Functions are Performed within Each Minor Cycle After Synch Tasks are Completed

#### 3.5.3.5 Relaxation of Synchronism Requirements

Synchronous execution is a desirable feature for real time operation because of the high predictability attainable of the times of occurrence of processor-controlled events. However, from the fault tolerance and recovery viewpoints, synchronism is not necessary unless the real time system is based on the simultaneous operation of two or more processors.

In a single processor environment, preservation of the order in which operations take place is probably more important than providing synchronism. If the timings of discrete events are delayed by a few tenths or hundredths of a second because a minor cycle failed and the machine was reconfigured, the consequences are insignificant, but the operational reliability provided by a sequence of operations where jobs are not aborted but retried until successfully completed and priority conflicts in the demand of resources are eliminated, is a highly important feature.

Relaxation of synchronism requirements implies the repetition of failed cycles is allowed in any one of the preceding executive alternatives.

The synchronous executive would provide an invariable sequence but the numbers of minor cycles per major cycle might not be always the same. Hybrid executives would provide a higher degree of real-time predictability because the buffering provided by interleaved asynchronous cycles would be sufficient for recovery from most synchronous cycle failures. The synchronous executive with asynchronous overlay would provide a uniform sequence with the degree of single failure buffering dependent on the time allocations for asynchronous jobs (relative to synchronous ones) and the effects of failures.

To differentiate between strictly synchronous executives and those in which synchronism constraints are relaxed for providing tolerance against minor cycle failures, the following designations will be used

Synchronous executive	→	Isosequential executive
Synchronous executive with asynchronous overlay	→	Isosequential executive with asynchronous overlay

#### 3.5.3.6 Selection of Approach

The isosequential executive with asynchronous overlay is preferred to the other approaches considered because of the following reasons:

- Except when certain faults produce complete failure of a minor cycle, real time operation will be synchronous, with all the consequent advantages.
- Asynchronous overlays within the minor cycles allow accommodating synchronous jobs of different lengths with low overhead penalties.
- Programmers are relieved from the burden of fitting programs into given fixed-time intervals.
- Most failures in synchronous jobs can be handled during the asynchronous parts of the corresponding cycles without loss of synchronism.
- Loss of synchronism, in the event of repeated reconfigurations and retries, may cause small delays in the timing of discrete events but will not compromise or degrade the fault tolerance and recovery capabilities of the system.

#### 3.5.4 Reference Software Configuration

The executive software plays key roles during normal operations as well as during failure recovery. A representative configuration has been designed to show functions, important features, and design tradeoffs peculiar to an isosequential executive with asynchronous overlay. This reference software configuration consists of the following programs:

- Executive (EXEC)

The executive performs top level failure detection functions, controls the primary phases of the bootstrapped recovery sequence (see Section 3.7), manages mode switching, updates status and roll back data, processes reprogramming functions, controls the timing of discrete events and schedules applications and asynchronous program modules for execution.

- Applications Modules (AM)

The applications modules are the basic functional software elements required to perform all the required subsystem functions, which include control laws, maneuver programming, sensor data processing and conditioning, actuator control, command processing, and telemetry data acquisition and processing. Also, AM's may be included to perform diagnostic, failure detection, or housekeeping functions of a more specialized nature than those of the executive, but which require specific sequencing in order to prevent interference with normal operations (e.g. diagnostic tests of failed units).

- Asynchronous Modules (ASM)

Asynchronous modules are the software elements that process off-line and major cycle jobs not involving either time or sequencing constraints. Functions performed by the ASM's include program verification and debugging (pre-launch operations), background data processing, routine system performance, monitoring, diagnostic tests, detailed failure detection routines, diagnostic command and telemetry interfaces and failed-job retry routines.

EXEC is entered periodically every minor cycle, by an unconditional interrupt (program synchronization signal) controlled by the RCU. In addition,

entry to EXEC can be through software jump instructions when either scheduling or other EXEC functions are required within a minor cycle (e.g. minor cycles including several AM's or during ASM execution). The basic elements included in EXEC are:

- Executive Bootstrap Routine (EBR)

EBR is responsible for the primary failure detection and recovery functions within the processor. Its first action consists in exercising key functions of the ACU and transmitting results of the tests to the RCU through a coded word. If the RCU receives this word within a predetermined interval (from the occurrence of the interrupt synchronization signal) and the coding represents an admissible state, operations are allowed to proceed. Otherwise, the RCU reconfigures ACU's, ROM's and DBS's (the primary processor) until a correct word is received on time. RCU performance is monitored by the HCU.

Other functions performed by EBR include checkup and (if necessary) reconfiguration of other processor units, verification of EXEC performance, and fault signal acknowledgement.

- Control Executive Routine (CER)

The CER checks operating memory units, verifies AM scheduler performance and checks ASM scheduling functions and status. In addition, CER updates rollback data, manages mode changes and reprogramming, and controls discrete event timing.

- Application Module Scheduler (AMS)

AMS determines the sequence of AM's to be executed in each mode. When executive transfer is initiated by fault signals, AMS selects the rollback point as the next job pointer. This ensures retry of a job that failed.

- Asynchronous Task Scheduler (ATS)

The ATS is entered at the end of the isosequential operations of each minor cycle for scheduling asynchronous jobs on a priority basis. Either ASM's or AM's may be scheduled by ATS for execution. The latter can be for either retry or diagnostic purposes.

There are many ways in which the component routines of EXEC can be implemented. Figure.3-49 shows the organization of an example configuration comprising the following subroutines:

- Primary Bootstrap Subroutine (PBS)

PBS is entered automatically after each program synchronization interrupt. If the system is designed to operate with either hardware or software fault interrupts, then the first step necessary is to momentarily disable further interrupts, since modification of tables by the executive must be completed once it has begun. In all cases, any signals used to enable writing on PWM's must be unconditionally reset to prevent accidental destruction of PWM contents (e.g. in the event that the subroutine for writing on PWM either failed to disable the permissive logic or was interrupted by a fault signal). Next, the process of quorum of primary processor elements (i.e. ACU, ROM, DBS) begins. Under supervision by the HCU and RCU, a sample problem is executed by PBS for exercising essential arithmetic, logic and control functions by the primary elements. The outcomes of the sample problem are the assembly and transfer to the RCU of a coded word or password within a given window. If a correct word is received by the RCU before the window falls, the test is successful and operations are allowed to proceed. Otherwise, reconfiguration takes place by switching units, one at a time, under RCU control. Using a quorum of primary elements, instead of the complete system, as the basis for reconfiguration allows substantial simplifications in the RCU and HCU, enhances operational reliability, and accelerates system recovery.

- Secondary Bootstrap Subroutine (SBS)

Given that the primary processor elements operate properly, a more complete bootstrap test and recovery process can now be performed with a minimum of trial-and-error substitution. The SBS checks the functional status and integrity of the rest of the system, including PWM, RAM, IOU, PDB, and necessary peripherals (i.e. sensors and actuators). The first element tested is the IOU because it is needed for reconfiguring the other units. If the IOU is found defective, SBS requests reconfiguration to the RCU through a TRC instruction.

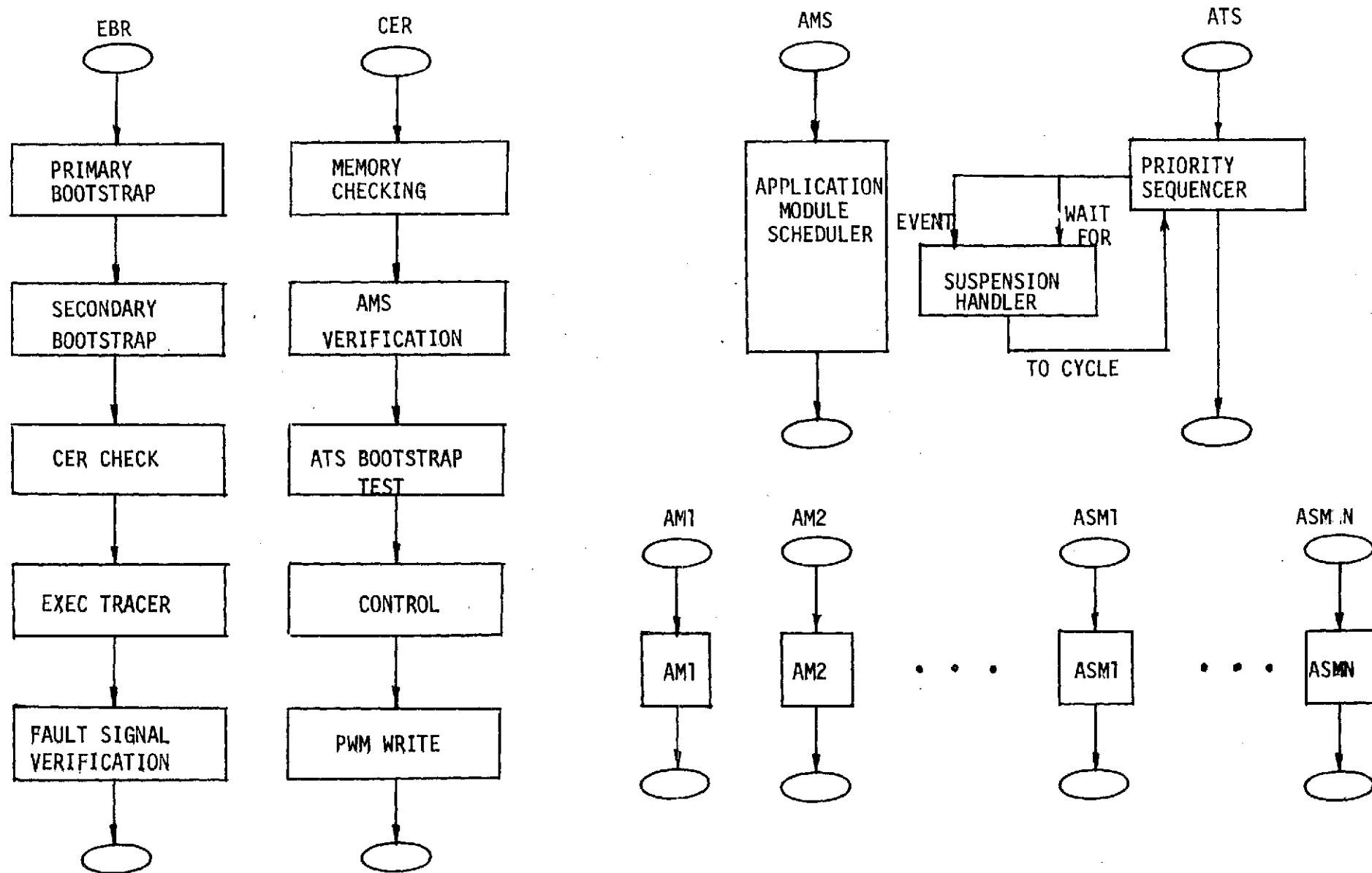


Figure 3-49 Reference Software Organization

- o CER Check Subroutine (CCS)

CCS performs instruction checks which may vary from a simple sum-check to a detailed audit (instruction-by-instruction comparison of redundant CER's). Functional testing of CER is difficult to perform without risk to data integrity or compromising system safety and recovery reliability. For this reason, CCS is used as a substitute, in conjunction with ETS (described below) and other indirect means of verification (e.g. results of AMS and ATS tests).

- o EXEC Tracer Subroutine (ETS)

This is a subroutine which verifies that the EXEC did not loop on itself during execution of the last minor cycle. ETS uses two tables, the EXEC Sequence Table and the EXEC Tracer Table, for diagnostic purposes. In the event of detection of EXEC looping, a fault signal (with appropriate flags) is issued to initiate reconfiguration.

- o Fault Signal Verification Subroutine (FVS)

Determination of the authenticity of fault signals is an important function in fault-tolerant systems. Fault signals that give erroneous information may cause serious problems during recovery, and in fact may cause an otherwise functioning system to lose its ability to perform its required operational tasks. (Hence, it is in the interests of reliability to reduce the number of fault signals to a necessary minimum).

Disabling the fault signal and acknowledgement that it is functioning correctly should be accomplished under program control.

Prior to transferring to the EXEC, determination of the interrupting source is made, and if the source is a fault signal, masking (disabling) of the next program synchronization interrupt is performed (so that recovery processing by EXEC is permitted to be completed before being interrupted).

- o Memory Checking Subroutine (MCS)

The bootstrap procedure consists of a sequence of memory tests. EBR checks the memories required for correct operation of CER and MCS does a similar thing with the memories used for AMS and ATS functions.

These checks may be in the form of of sum checks, parity checks, audits, or any combination of techniques.

- AMS Verification Subroutine (AVS)

AVS performs a functional verification of AMS performance to determine whether scheduling operations take place as specified by the corresponding task table. A log of completed jobs is kept in the Job Executed Table (JET). The contents of JET are compared to the sequence defined by the main Job Control Table (JCT) for agreement. If disagreement occurs, a software fault is indicated. The AVS can be scheduled optionally every minor cycle, at completion of each major cycle, or as often as desired.

- ATS Bootstrapping Test Subroutine (ABS)

ATS can be tested by either functionally exercising it and checking the response obtained, or by monitoring the past history of ATS operation against some predetermined performance criteria based on an entry/exit flagging scheme for tracing scheduling sequences. The first-mentioned approach is too time consuming if applied for direct in-line checking, and the second one has the disadvantage of requiring performance criteria that are very difficult to define. A compromise providing an efficient and reliable way to test ATS consists in exercising priority scheduling functions by means of a bootstrapping subroutine and a set of dummy ASM's. ABS generates job requests (with various priorities) to be serviced by the dummy modules and monitors results obtained. Dummy ASM's require very little execution time and may be used to perform routine functions if desired.

- Control Subroutine (CS)

CS performs top-level management functions, including mode changes and reprogramming, updates roll-back data, and controls discrete event timing. Most of the CS functions are performed in conjunction with the PWM write subroutine described below.

Mode changes are executed at the completion of a major cycle by changing the JCT in AMS. The new JCT is retrieved from ROM or, if input by command, from PWM. A copy is placed in a specified location in

RAM and is subject to parity and sum check verifications prior to transfer to PWM.

Every minor cycle, CS begins the process of assembly of the roll-back data necessary for software recovery. These data are transferred to a specified block in RAM and parity and sum check verifications are made prior to transfer to PWM. Before RAM-to-RAM data transfers are made, checks are performed to verify that the initial and data block address codes correspond. Reprogramming is handled by similar techniques since it involves writing on specified blocks of PWM.

Discrete event timing is made directly by the CS through the IOU. The resolution is determined by the minor cycle time interval. Specified bi-levels are turned on when the corresponding timing counts elapse. Flags are set for bi-level turn-off in the following cycles. This function is assigned to the CER because it requires updating counts in PWM every minor cycle.

#### o PWM Write Subroutine (PWS)

This subroutine is intended to protect the system from indiscriminate overwriting of data in PWM that may be necessary for mission success. Several sequential checks must be passed before permission to write into PWM is granted, and if any fail, a fault interrupt is issued, forcing the system into EBR for fault isolation and reconfiguration processing, if necessary.

First, a check is made to see if there was indeed an AM request to write in PWM in the last minor cycle. If so, verification that the PWM write enable (WE ) is "off" is made. If WE is "off" as it should be, a request is made to the IOU to issue a WE bi-level. Next, a check is made to see that the WE signal is issued by the IOU, a step necessary to ensure that critical data are not lost accidentally. Next, subtractions of addresses (first and last addresses) are performed to determine the sizes of the data blocks. A check of each block size, against a previously stored number, reveals if this size is correct (to prevent an incorrect address permitting the overwriting of part or all of the protected data).

As a backup check, the data are coded so that only data in certain locations is permitted to write under PWS control as derived from the CS. Only if the original code corresponds to the code of the data assembled in RAM is writing permitted. If all these conditions are met, data transfer between RAM and PWM takes place.

After writing, a check is made to see if the PWM and RAM data are identical (verifying that the write operation was successful). If so, the WE must be disabled in order to protect PWM data. Furthermore, a check of the disabling operation is required to verify that this has been accomplished (the IOU may have been requested to disable the WE and failed to respond). If all these tests are successful, exit from the PWS subroutine occurs.

- Application Module Scheduler (AMS)

The main function of the scheduler, in a given system mode, is to select a sequence of AM's from information contained in the JCT for that mode. For typical minor cycles where executive transfer is initiated by a cycle-timer interrupt, the selection is based upon the past minor cycle sequence and major cycle index. Information useful for this scheduling is found in the JET.

When executive transfer is initiated by fault signals, however, the scheduler selects the rollback point as the next past job pointer. This ensures retry of a job that faulted (or execution of the AM succeeding the last one completed when the fault is not due to an AM execution failure). Transfer from AMS is made directly to the selected AM.

- Priority Sequencer (PS)

Transfer is made to ATS at the end of synchronous task calculation for scheduling of asynchronous jobs on a priority basis. During this asynchronous time both ASM's and AM's may be scheduled by ATS, the latter for functional test purposes. PS processing begins by checking mailbox locations for I/O event reports and command messages. If an event report is present, transfer is made to the suspension handler (described below), while if no event is present, transfer is made to

CYCLE (a PS entry point). A command message is handled by putting a high (est) entry on the priority table and transferring to CYCLE.

CYCLE is the scheduling part of PS, where the highest priority job in the ATS job table is selected for execution, table priority entries are modified, the state vector for that job is loaded, and transfer from ATS is made to the corresponding AM or ASM. Return to ATS is made via a WAITFOR request or an END message.

o Suspension Handler (SH)

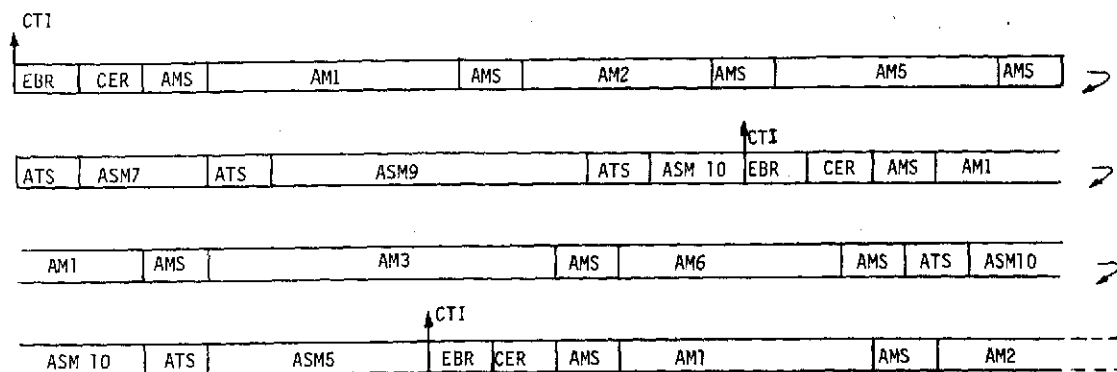
This subroutine has two entry points: EVENT and WAITFOR.

When an I/O event occurs, the subroutine executes EVENT, by which the appropriate entry in the wait-for-pending table is removed (if it is there, otherwise there is either an error or a command-type message has arrived and the job is placed with high(est) priority in the job table. Transfer is made to CYCLE in PS.

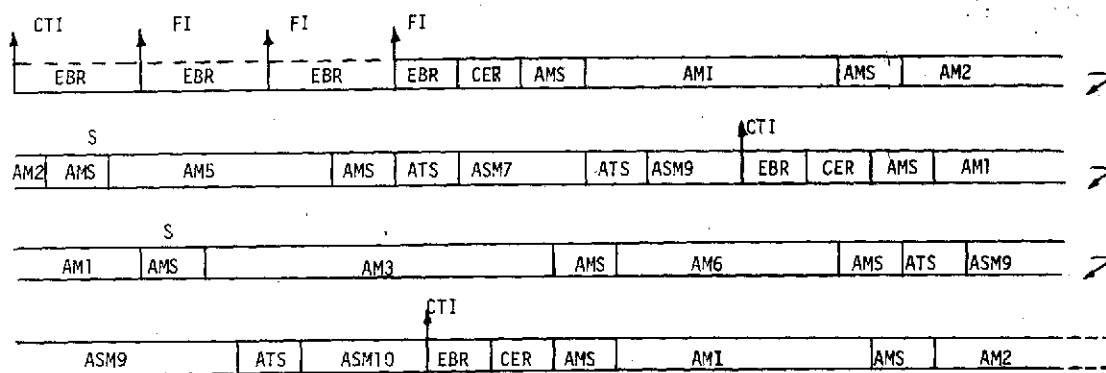
If a job is suspended, a WAITFOR message is issued, the I/O event in question is marked in the wait-for-pending table and the job is put into a suspended status in the job table. Transfer is made to CYCLE in PS.

Typical processing sequences with the multi-programmed, real-time reference software system described above are shown in Figure 3-50 where minor cycles are the time periods between cycle timer interrupts (CTI). Processing during each minor cycle interval is divided between the executive, the regular AM sequence, and the priority-scheduled ASM portions as shown in Figure 3-50 a). The executive is entered after every CTI for reasons of fault tolerance, since reliable failure detection and recovery can be achieved when periodic checks are caused by a dependable interrupting source, i.e. the RCU with HCU supervision.

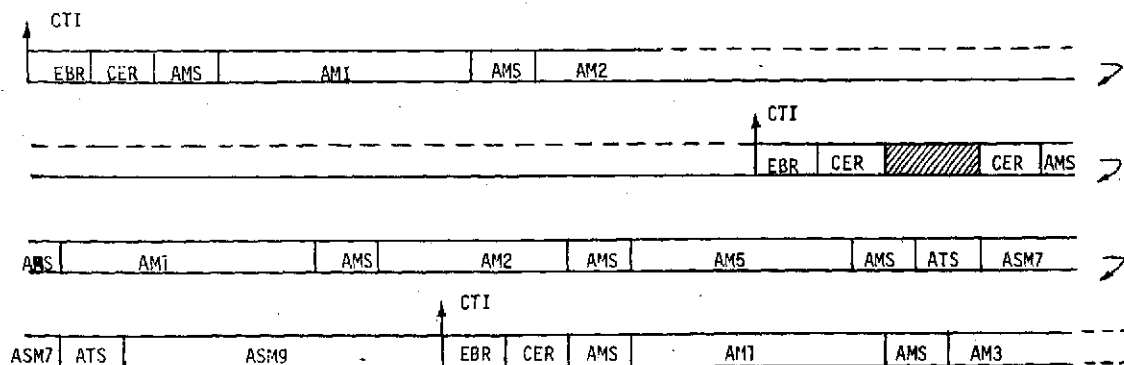
Isosequential processing of the AM's enhances fault tolerance and recovery reliability because the execution of on-line functions is predictable and well defined and performance verification is greatly facilitated. Asynchronous processing of ASM's facilitates programming by easing time constraints and contributes to obtaining a higher operational efficiency.



a) Without faults



b) EBR failures detected by RCU - 3 reconfigurations



c) Failure detected by ECR

Figure 3-50 Typical Processing Sequences of the Reference Software Configuration

ORIGINAL PAGE IS  
OF POOR QUALITY

A key feature of the reference software concept is the high degree of flexibility attained, as demonstrated by the following properties:

- o The relative period of AM and ASM time (in each minor cycle) may be determined by the programmer according to application and mode.
- o The executive processing interval is optional and can be chosen to be
  - short enough to keep overhead within desired bounds
  - long enough to ensure satisfaction of fault tolerance requirements (reducing fault propagation, reducing recovery time, etc.)
- o The minor cycle interval can be chosen to be
  - consistent with fault tolerance requirements (short enough to interrupt a looping job in the time required)
  - compatible with AM processing (long enough to minimize segmentation or "chopping-up" of AM's).
- o If desired, minor cycles can be scheduled to include AM's only or, alternatively, AM's and ASM's can be alternately scheduled after synchronization interrupts. The designer has freedom to choose the best approach for each mission regime.

Figure 3-50 b) shows an example case where a fault occurring before CTI causes a failure of EBR to transfer a status word to the RCU within the window period. When the RCU window falls the reconfiguration logic is activated, one of the primary processor units is reconfigured and, after reconfiguration, a reset interrupt (RI) is issued by the RCU to start execution with EBR. In the example, three consecutive failures are assumed. After the third reconfiguration, processor operation is resumed. The AM's scheduled for this minor cycle are executed with a small delay but the proper sequence is observed. The ASM execution sequence changes because ASM9 cannot be completed as in a) and ASM10 is obviously deferred to the next minor cycle.

The example shown in Figure 3-50 c) illustrates the case where a RAM memory failure causes an endless loop during execution of AM2. The loop continues until CTI starts an executive run. ESR tests are passed and CER discovers a RAM failure. The IOU is commanded to reconfigure RAM's and

execution begins when ACU operation is halted by a DLY instruction. The IOU reactivates the ACU by means of an ADV signal when reconfiguration is completed, and a software jump to the CER entry point causes a new series of tests. After passing the CER tests, AMS execution starts. Since there are no indications that the AM sequence was completed, AM's 1, 2, and 5 are scheduled for execution. These jobs are done with one minor cycle period delay but the sequence is not changed. If roll-back data placed by AM1 in RAM had been preserved, execution after recovery would have started with AM2. Repetition of the failed minor cycle sequence is made possible by the JCT and JET stored in PWM.

Validation of an executive program as discussed here requires further definition of routine and subroutine functions and a detailed failure mode, effects, and criticality analysis (FMECA). The multistage bootstrapping technique on which this program is based appears to offer a promising and foolproof means of ensuring that recovery is possible regardless of program failures (especially the difficult ones to diagnose, e.g., those which do not generate fault signals but may cause looping in one program, incorrect jumping to another program and/or incorrect data generation). Increasing the number of diagnostics and fault signals with the intent of aiding diagnosis by reducing the possibility of unsignaled faults may only complicate the problem; extra options may cause ambiguities which are difficult to resolve during recovery and the fault signals themselves may fail, greatly increasing the burden on system recovery procedures.

#### 3.5.5 Diagnostics and Fault Detection

Diagnostic and fault detection alternatives and tradeoffs are covered in general in Section 3.7 and hardware diagnostic techniques are discussed in detail in Section 3.6.2. The object of this Section is to categorize and discuss briefly the various software diagnostic techniques applicable at the system and lower levels.

As is the case with the operational software, software diagnostics can fail and, consequently, bootstrapping techniques and hardware monitoring are necessary for adequate validation.

Software diagnostics are particularly attractive for multipurpose, fault-tolerant systems because of their flexibility and compatibility with

function and equipment options. The hardware monitors are the only elements that should remain invariant in a truly multi-mission system.

The organization of software diagnostics is normally influenced by the recovery management structure adopted for the system, since one of the key objectives of failure detection is to localize failures at the replacement level.

System failure detection can be performed either in line or off line. The organization can be either centralized or distributed (e.g., bootstrapped). Failure concealment can be prevented by means of redundancy, reliance on propagation, or failure detector monitoring. Most frequently used (or proposed) techniques for system function validation include:

- Performance Monitoring

This is an off-line function that can range from simple checks of the outputs of identical or functionally redundant sensors to highly sophisticated estimation techniques. An approach that is very effective and simple to implement consists in monitoring the total spacecraft momentum. Momentum inputs caused by thruster firings and disturbance torques are estimated and spacecraft rates derived from sensor readings are compared to values predicted from momentum wheel speed measurements.

- Reasonableness Checks

These are in-line checks of system operation against pre-stored limits or tolerances or calculated regions of normal or expected behavior. Typical variables monitored are frequencies of pulses, durations of pulses, rates of change and limits of sensor outputs, etc.

- Parallel Operations

Sensors and/or control laws are operated in parallel to check for agreement. An alternate approach consists in switching to redundant elements and repeating the operations performed by the original configuration. If the two results agree the operation is assumed correct.

- Responses to Stimuli

This is a test where the response produced by a known input is compared to a known result. This approach is very effective for revealing

whether an unflexed response is due to absence of commands (or inputs) or to system failures. Diagnostics are typical elements with unflexed outputs requiring external stimuli for performance verification (i.e., simulated fault conditions or signals).

Detection of failures in memories is difficult because there are mechanisms likely to produce multiple failures. In addition, LSI components are more prone to exhibit burst-type failures than discrete elements. The following are some of the most commonly used approaches for memory failure detection:

- Coding/Parity Checks

Coding or parity checking are effective when the fault patterns of the system to be validated are predictable. This allows designing the tests for detecting the majority of possible errors.

Coding or parity check techniques are very effective when combined with other methods of failure detection (e.g., sum checks) or failure prevention (e.g., hardware techniques to prevent failure patterns not detectable by the code).

- Check Sums

The sum of a group of instructions or data words in a memory block plus a code word stored somewhere else should be zero for the check sum test to be successful. This is a very simple and effective error detection scheme but it fails to reveal multiple errors of the  $0 \bmod 2^n$  variety (e.g., two errors in the 14th-bit position).

- Sample Problems

Using a combination of system components, problems with given inputs can be run and the answers compared against known results. It should be realized that checking a program in this manner will not necessarily exercise each program branch since branching is generally data dependent. Thus, errors that occur in certain branches will not be detected by a test problem that does not exercise the defective branch. The entire system or only a part of the system (e.g., ACU adders and registers) can be tested in this way.

- Duplex Operation

This approach consists in straightforward, in-line, parallel memory operation with coincidence checking. When high speed computations are required, coincidence verification is done by hardware. Software implementation of duplex operation can be used for data handling at slower rates than in a duplex mode.

- Auditing

Auditing is an off-line word-by-word verification of the contents of a working memory by comparison to identical information stored in either a redundant ROM or a bulk memory unit.

Arithmetic control units may be checked by either running sample problems, as described for memory units, or performing a series of multiply, jump, and indexing instructions with which virtually all portions of the ACU can be tested.

Input/Output units can be checked by means of one or more of the following techniques:

- Bootstrapping Checks

Facilities are provided in the IOU's or peripheral units for storing test data. Bootstrapping tests typically consist in the transmittal of numbers stored in ROM to a remote storage register and the retrieval of these data for coincidence checking.

- Echo Checks

These tests are also bootstrapping techniques except that different channels or data links are used for in-bound and out-bound data transmission. A typical example of echo check is the method used in the COPE processor for testing bi-levels. The bi-level logic can be checked by turning a dummy bi-level on by means of an SBL instruction and then checking the status of this bi-level by means of an SKE command. If the bi-level is high, an echo signal is returned by the IOU on a separate response line and the ACU executes the next instruction. Otherwise, this instruction is skipped.

- Responses to Stimuli

These tests are as described for the system, except that input signals should be chosen to enhance the sensitivity of the responses to input/output errors.

- Test Signals from Peripherals

Inbound communications are tested by means of known signals generated by peripherals.

Failures in peripheral devices can be detected by performing parallel operations (e.g., checking the outputs of redundant sensors), applying known stimuli to verify the resulting responses, or by means of test modes implemented within each peripheral unit.

Data buses are checked together with memory units by reading data from ROM and storing and reading data from RAM's.

### 3.5.6 Software Sizing

Table 3-14 contains preliminary estimates of the storage and processing requirements of the reference executive program described in the preceding section. The procedure used to derive these data consisted of the following steps:

- 1) Detailed definition of the functions to be performed by each subroutine.
- 2) Implementation of the subroutines by means of a set of generalized statements.
- 3) Implementation of the generalized statements by machine language instructions.

The following is a summary of the assumptions made in the preparation of Table 3-14:

- PBS

In addition to instructions for exercising primary machine functions, 8 tests of internal indicators, 10 tests of external indicators and a sum check of SBS are included.

TABLE 3-14 - ESTIMATED STORAGE AND PROCESSING REQUIREMENTS FOR THE REFERENCE EXECUTIVE

Routine	SubRoutine	ROM		RAM	PWM	INSTRUCTIONS/RUN	
		Prog	Data			Faults	No Faults
EBR	PBS	60	10	-	-	359	253
	SBS	80	18	36	15	1768	1048
	CCS	16	4	4	2	420	410
	ETS	8	2	2	2	84	80
	FVS	45	10	10	10	160/240	120
CER	MCS	30	4	2	2	517	505
	AVS	10	2	4	20	10	8
	ABS	30	10	5	2	74	61
	CS	115	25	40	40	150/1110	150/1110
	PWS	20	4	10	2	370	365
AMS	-	30	4	5	5	30	30
ATS	-	50	6	15	15	50	50
TOTALS		494	99	133	115	3992/5032	3080/4040

- SBS  
This includes RAM and PWM checks, verification of status and tests for up to 36 system elements, reconfiguration instructions, and sum checks of CCS, ETS, and FVS.
- CCS  
A sum check of CER and flag testing instructions are included.
- ETS  
This includes instructions for checking tracer tables and setting fault flags.
- FVS  
Processing of 10 fault signals is assumed. Possible actions are:  
1) problem solved, lift flag, 2) transfer to diagnostic routine in AM, 3) place request for off-line device test, and 4) reconfigure device.
- MCS  
This includes testing of 5 fault flags and sum checks of AMS and ATS.
- AVS  
Processing of 10 AM's per major cycle is assumed.
- ABS  
Verification of ATS is assumed on the basis of 5 dummy ASM's. Instructions include by-passing normal sequence to enter ATS and return to ABS for checking test results.
- CS  
Lower bounds correspond to normal operating conditions without mode changes or reprogramming inputs. A rollback data block of 40 words is assumed. Program changes are handled in blocks of up to 10 instructions.
- PWS  
This includes instructions for testing PWM write enable/disable logic and transferring data to PWM in blocks of 10 words each.

- o AMS  
Job control and tracing tables are included in AVS.
- o ATS  
Sizing is based on a maximum of 10 ASM's in the priority list.

Totals for RAM storage requirements and instructions/run are not representative of actual conditions since memory space can be shared to a great extent and not all subroutines must be executed every time.

### 3.6 Hardware Fault Detection

It has been seen that fault detection may be by software or hardware techniques or a combination of both. Software fault detection and diagnostics were discussed in Section 3.5.3. System performance monitoring (which is a type of fault detection) was discussed in Section 3.5.4. The subject of failure recovery was introduced in Section 3.5.5 and will be expanded on under reconfiguration in Section 3.7. This current section restricts itself to the methods of hardware fault detection in the processor and peripherals, to fault correction or masking (if used), and to the problems attendant in providing the system clock.

#### 3.6.1 Error Detecting Coding

One powerful method for hardware fault detection is the use of error detecting codes. Error detecting codes may also be expanded into error correcting codes, but only with a penalty of at least doubling the word length. Error correcting codes are one method of fault masking, a technique that is not deemed to be necessary in a spacecraft control system processor, except possibly in the "hard core" portion. No further consideration will be given here to error correcting codes. Their use is more applicable to transient communications errors (which are noise induced) than to the more-likely permanent faults of a processor. If interested, error correcting codes are well covered in the literature.

Error detection is a relative operation. The overhead increases as the degree of detection increases. Detection of single errors requires only an extra bit (parity coding). Detection of all errors will require that the words be completely redundant. The immediacy of error detection is also a variable. How often in the operations the words are checked can be suited to the speed of detection requirement for errors.

Immediate detection of all processor errors is very expensive in terms of hardware (and software) overhead and processor speed. Fortunately, there is no need to detect all possible errors and immediate error detection is not needed.

Error detecting codes are codes that modify a word with extra bits in a systematic way so that errors in some or all of the bits may be detected by recognizing that the code is improper.

Error detecting codes can be applied to data words and/or instructions. They make it possible to detect the presence of errors at each point of detection. The generation of a code requires a hardware circuit termed a "former" and the detection of the code requires a hardware circuit termed a "checker". Formers and checkers add to the part count, consume power, etc. and so should be minimized in use to reduce such hardware overhead.

Error detecting codes also add to the word length, increasing the size of registers, memory, data buses, etc; also adding to the hardware overhead. The codes also require more time in serial or byte-serial organized processors, providing a penalty here too.

Errors can be detected with coding to any desired degree of completeness or immediacy, but only with increasing penalties. Note that, as one extreme, no error detection coding is a viable alternative (and, in fact, the most common one).

Codes with a single parity bit are often used for error detection in memory. This was the technique used for error detection in RAM in COPE. The penalties for such a code are very small. Parity codes can only detect single errors or odd numbers of errors. Parity codes are not preserved through arithmetic operations. Parity codes are also not useful for checking serial or byte-serial paths, where the total word or the corresponding bit of each byte may be in error.

A more desirable error detecting code would be one that could be applied to data words (for now) and which would be preserved through arithmetic.

Prof. Avizienis and his students have extensively studied such codes with the conclusion that product codes of the form  $2^4-1$  are particularly useful for checking byte-serial organized processors whose byte length is 4 bits.

Such a "Modulo 15 residue coded check byte" offers distinct hardware advantages in terms of circuit minimization. This code will detect all errors except those that are multiples of 15.

A check byte is added to each data word in the most-significant byte position. This byte is formed so that the total word is a multiple of 15. This is done by a process analogous to casting out 9's in decimal arithmetic. To form the check byte, add all of the other bytes together (observing carries), and complement the result. (bit for bit)

An interesting feature of this code is that it is preserved through simple arithmetic operations, if the carries are properly observed. This is a property fairly unique among arithmetic codes. It is not preserved through logical operations. Arithmetic operations which preserve the code are:

- Addition and subtraction
- Multiplication and Division (with precautions)
- Left and right shifts (if over-or under-flows do not occur or are accounted for)
- Double precision addition and subtraction (with proper precautions).

To check if any errors have occurred, add all bytes of the word (including the check byte) together, observing carries. The result should equal 15 = 1111. If not, one or more errors have occurred.

To illustrate, consider the following examples (using 8 bit words):

- Word-A =            1111 0100    to form the check byte:

$$\begin{array}{r} 1111 \\ + 0100 \\ \hline 1\ 0011 \\ \hline \end{array}$$

0100 = check byte.            The total word is:

$$A' = \quad 1011 \quad 1111\ 0100$$

(Note this adds up to 1111)

- Word-B =            0000 1010    Forming check byte:

$$\begin{array}{r} 0000 \\ + 1010 \\ \hline \end{array}$$

1010 = check byte.            The total word is:

$$B' = \quad 0101\ 0000\ 1010$$

- Adding words  $A' + B' = C'$

$$\begin{array}{r}
 1011 \ 1111 \ 0100 \\
 +0101 \ 0000 \ 1010 \\
 \hline
 1 \ 0000 \ 1111 \ 1110 \\
 \begin{array}{l} \text{L} \\ \rightarrow \end{array} \underline{1} \\
 \hline
 C' = \quad 0001 \ 1111 \ 1110
 \end{array}$$

Checking this:

$$\begin{array}{r}
 0001 \\
 1111 \\
 + 1110 \\
 \hline
 1 \ 1110 \\
 \begin{array}{l} \text{L} \\ \rightarrow \end{array} \underline{1} \\
 \hline
 1111
 \end{array}$$

which is proper.

- If the addition resulted in an error in one bit:

$$C' = \quad 0001 \ 1101 \ 1110 \quad \begin{array}{c} \text{Error} \\ \downarrow \end{array} \quad , \text{ the check would give:}$$

$$\begin{array}{r}
 0001 \\
 1101 \\
 + 1110 \\
 \hline
 1 \ 1101 \\
 \begin{array}{l} \text{L} \\ \rightarrow \end{array} \underline{1} \\
 \hline
 1101
 \end{array}$$

1101 which is  $\neq$  1111, so the error is detected.

- If the addition resulted in an error in the same bit position of each byte: (stuck on zero)

$$C' = \quad \begin{array}{ccc} \downarrow & \downarrow & \downarrow \\ 0001 & 1101 & 1100 \end{array} \quad \begin{array}{c} \text{Error} \\ \downarrow \end{array} \quad , \text{ the check would give:}$$

$$\begin{array}{r}
 0001 \\
 1101 \\
 + 1100 \\
 \hline
 1 \ 1010 \\
 \begin{array}{l} \text{L} \\ \rightarrow \end{array} \underline{1} \\
 \hline
 1011
 \end{array}$$

1011 which is  $\neq$  1111, so the errors are detected.

Note that the overflow carry is added as a least-significant-bit (end-around carry). Rules have also been formulated to handle the carries resulting from multiplications, divisions and shifts.

We see therefore, that the check byte former can be a circuit that adds (in byte form) the bytes of the data word, observing carries, and complements the sum, tacking on this check byte at the end of the word.

The check byte checker is a circuit which adds (in byte form) all of the bytes of the total word (including check byte), observing carries, and compares the total (residue) to 1111, issuing an error signal if the comparison is improper. All of the word can shift on through, the checking occurring simultaneously with the word shift.

Since the functions of forming and checking are so similar, they may be combined in a single circuit (see Figure 3-51 ). This circuit can be controlled to either form or check the check byte of 4-bit byte serial organized words (of any length) or to neither form nor check the check byte. It issues an error signal if the checking is improper.

This circuit requires 1 MSI (the adder) and 7 SSI parts, would consume about 150 mw and have 225 failure bits. It could be used wherever needed in the processor. If check bytes are used, the timing must be modified to provide "room" for the check byte, and all registers must be expanded by one byte in size.

If the check byte is used for error detection, where in the processor should it be used? Its use should be minimized because of the hardware penalties and yet it should be used at those points in the data flow most likely to promptly show up errors.

The former should be used:

- Wherever data words initially enter the system
- Wherever the check byte needs to be reapplied to words (such as where words have been stripped of the check byte because of logical operations).

These two requirements translate into using the former:

- Out of the IOU
- Out of the adder of the ACU

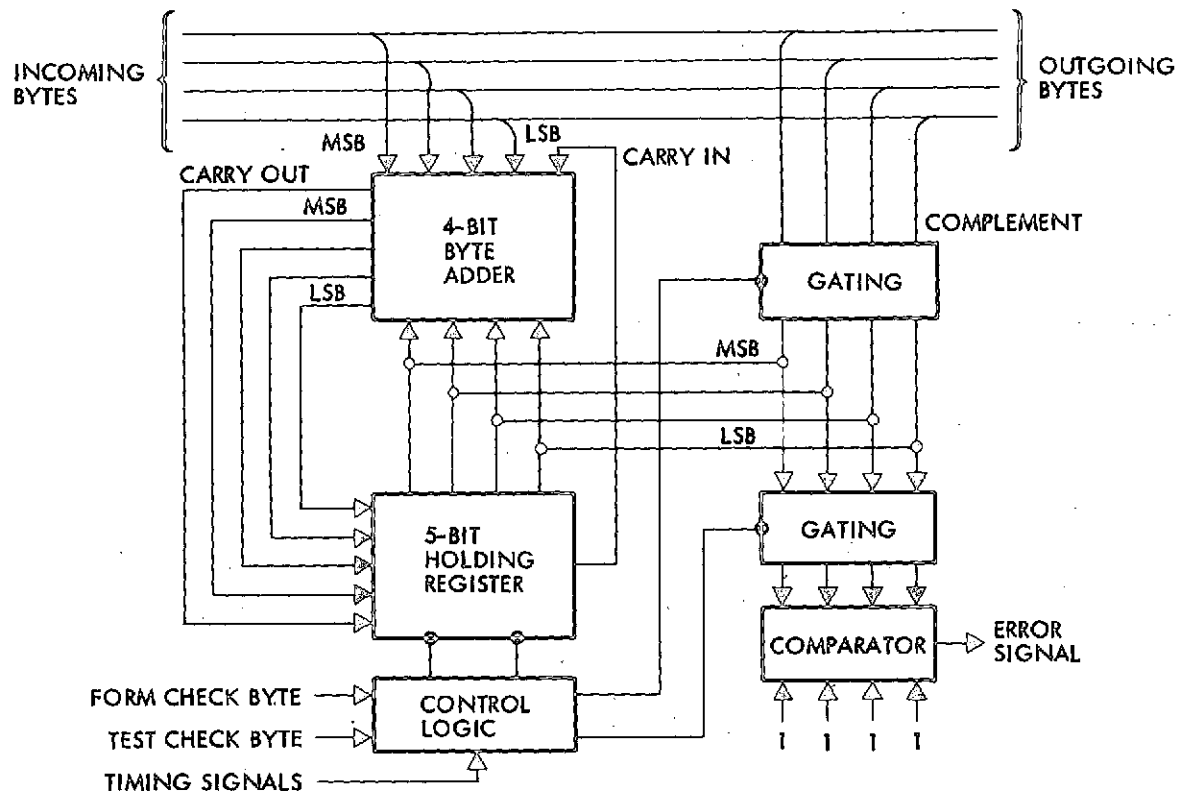


Figure 3-51  
Check Byte Former/Tester

The checker should be used:

- At the end of data transmission paths
- Following arithmetic operations
- Prior to writing into memory
- Prior to use in I/O.

These requirements indicate use of the checker:

- Into the ACU (A-register in COPE)
- Out of the adder of the ACU (Same as the former, above)
- Into the RAM or PWM
- Into the IOU.

This means that the IOU and ACU would each have two former/checkers, the RAM and PWM would each have one, and the ROM wouldn't need any. If this were done for a 16-bit processor (COPE), it would result in a 1.25 x time penalty and about a 1.25 x hardware penalty. (So far, only for data words).

Next, the use of error detection coding on the instruction words should be considered. The instruction words are made up of two portions, the operations code (op. code) and the address. The operations code is (relatively) immediately decoded into the specific instructions. The address field may be operated on, shifted between registers and counters, incremented, added to other addresses, etc. in the ACU.

It is clear that these two portions of the instruction:

- must be handled separately insofar as error detecting coding is concerned
- may not require the same methods of error detecting coding.

Whatever is done must also:

- Contain the same information in the instruction word
  - No reduction in direct address field
  - Retain the same number of decoded instructions
- Maintain the instruction word length equal to the data word length.

Since the address field portion of the word is arithmetically operated on, it makes sense to add a check byte to detect errors in incoming or outgoing (from the ACU) address field portions of instruction words. This adds

4 bits to the instruction word length. If a check byte was also added to the data word, and if the data and instruction words were originally the same length, then this uses up all available bits without providing any error detection for the op. code., unless the data word is extended (more accuracy?).

Op. Code error detection is probably more important than either data word or address error detection, since an error in even one bit completely changes the instruction executed and the subsequent processor operation, most likely propagating the fault quickly. On the other hand, the op. code is fetched from ROM (or PWM), where faulty words are less likely, and is decoded relatively immediately without arithmetic operations, so that errors are less likely.

Assuming 8 bits for the Op. Code (as in COPE), there are probably little spare (unused) decoded states. COPE used 5 bits for 30 instructions (2 spare) and the remaining 3 bits for tags of indexing, etc.

One technique for op. code error detection coding would be the use of the popular parity bit. This would be useful, as no arithmetic operations occur and the bit can be easily formed and checked. Its use adds one bit, which may not fit in well with the word length requirements. If used, the parity bit should be checked just prior to (or as a part of) instruction decoding. It would need to be formed at PWM write.

Another technique that is attractive would be to code the op. code separately with a multiple error detecting code. The check byte approach discussed previously might be used, but is not particularly attractive since the op. codes are transferred in parallel (most likely) and the checking should be through the decoding.

A method that has been used, is the use of a 2 out of 4 (or equivalent) code. A 2 of 4 code gives 6 permissible states (out of the 16 possible). Only combinations having exactly two 1's are recognized and decoded. All other combinations are faulty. Note that the decoding (and error recognition) are very simple.

The number of acceptable codes (for total bits used) are:

- 4 total bits = 6 codes
- 8 total bits =  $6^2 = 36$  codes
- 12 total bits =  $6^3 = 216$  codes
- 16 total bits =  $6^4 = 1296$  codes.

For most processors, 216 codes should be sufficient and 12 bits could be used for 2 of 4 op. code coding. Note that an uncoded 8 bits gives 256 possibilities, which is only slightly more. The use of 12 bits would add 4 bits to the instruction word length for op. codes.

If error detecting coding is used on the instruction word, the "best" approach would result in: (as applied to COPE)

- 12 bit op. code, coded as 2 of 4.
- 4 bit check byte for address field.
- 8 bit address field.

This totals 24 bits. The COPE data word length is 16 bits (20 bits for use of check byte). To make the words the same length, the data word length could be extended to 20 bits, providing better (but unneeded) accuracy. This would result in a 1.5 x penalty in speed and about a 1.4 x penalty in hardware.

As discussed in Section 3.4.1, if the processor is a 24 bit accuracy machine, the use of double precision instructions would be reduced to the point that the true speed might be enhanced (since double precision takes twice as long), even though the actual speed was reduced by the longer word length. Considering the error detection, this could result in a total word length of 28 bits, giving for data words:

- 4 bit check byte
- 24 bit data field

For the instruction words, this could give:

- 12 bit op. code, coded as 2 of 4
- 4 bit check byte for address field
- 12 bit address field (enabling direct addressing of 4,096 words).

Such a processor would have a 1.75 x penalty in speed and about 1.5 x penalty in hardware. (Although the "real speed might be about the same.)

What do we get for these penalties? We get somewhat better and somewhat quicker error (fault) detection than can be achieved by other means.

Whether this is necessary is a question for other sections. The conclusion seems to be that error detecting coding is not needed (with the possible exception of the op. codes).

A remaining subject of the consideration of error detecting coding is the need for use of such coding on data transmitted between the IOU and peripherals. Note that (on COPE, at least) this data transmission is in serial form.

If such error detection is used, then its primary purpose must be to detect errors caused in the communication process, across the interface although it can also detect errors in the ACU/IOU interface and in the IOU logic.

If the processor uses error detecting coding, then the same technique can be carried across to the peripherals. If a parity bit is used, this works quite well across the serial interface, as parity bit forming and checking on serial data is very simple.

If a check byte is used (in the processor) then this can be shifted out in serial as the 4 MSB's of the serial word. The peripheral checker must include a 4-bit buffer register, besides the circuitry of Figure 3-51, and the other receiving circuitry. This adds quite a bit of circuitry if the checking is done at each peripheral.

For data going the other way (to the IOU), the same problem occurs. If check bytes are used, they should be formed in the peripheral. No (additional) check byte former/checkers are needed in the IOU.

If the parity bit or check byte received by the peripheral is incorrect, then the choices are:

- Do not act upon the data
- Do not act upon the data and issue a BITE fault indication
- Act on the data and issue a BITE fault indication.

Similar choices exist for the IOU. The best policy would appear to be to issue a BITE fault indication and do not act on the data.

It appears that if any peripheral interface error detecting coding is used (and none may be necessary) that the parity bit is the best choice. This adds a time penalty of  $17/16 \times$  (for COPE) and about 1 SSI for each former or checker.

### 3.6.2 Self-Test

The self-testing is regarded here as the hardware fault detection techniques other than error detection coding. This self-testing hardware is often referred to as BITE, or built-in test equipment, although a somewhat more descriptive term might be devised, since fault detection is the function being performed.

The self-testing should be divided into that for and by the processor and that done in the peripherals.

#### 3.6.2.1 Processor Self-Test

The hardware self-test provisions designed into COPE are representative of processor self-test techniques. These provisions are:

- Detection of illegal instructions (in the ACU instruction decoder) and illegal device addresses (in the IOU). These detections are of limited use since they detect only the few (out of many) illegal possibilities. Note that this is illegal, not improper or incorrect detection. In a new COPE design this BITE would probably not be included.
- Detection of micro-timing counter overflow (in the ACU). This detection of the normally unused states also has limited usefulness and would probably not be done again.
- Detection of RAM parity error. A parity bit is formed and checked on each RAM read or write operation. This would probably be done again for RAM (and PWM) if other error detecting coding were not used.
- Detection of actual block power condition disagreement with power control signal. This check, done in the RCU, provided the RCU with an indication that the block was or was not following power control instructions. It is debatable if this is necessary, since if the block power status is improper then the processor cannot operate properly and this will be detected by software soon enough.

- Detection of improper program timing or synchronization. This was done in COPE by timers in the IOU and RCU. These timers required the execution of an ADVance instruction within a predetermined window or a fault is registered. As noted elsewhere, it is now felt that the ACU should be synchronized to synch signals from the RCU and the RCU should receive data from the ACU a prescribed time later or a system fault is registered. This is the principal fault detection technique in the processor.
- Power fault detection. This was used in COPE, and should be retained, to indicate power interrupts. Similar circuits have also been devised to detect nuclear events or large E or H fields.

No path, bus or ROM error detection was used on COPE. If this is wanted, then some form of error detection coding can be used.

Note that the hardware BITE that should be used in the processor consists mainly of the power fault detection and timing fault detection.

#### 3.6.2.2 Peripheral Self-Test

The peripheral BITE used should be only that which enhances the ability of the processor to more quickly or more definitely detect faults in the peripherals. If those faults are either of no importance or they can be detected from the normal signals from the peripherals (by the processor software), then no peripheral BITE is needed.

All peripheral BITE fault signals are assumed to be bi-levels interfacing with the IOU.

The different peripherals are:

- Wide angle sun sensor (probably analog) and electronics
- Fine sun sensor (digital) and electronics
- Star sensor and electronics
- Inertial reference unit and electronics
  - gyros
  - accelerometers
- Reaction wheels and drive electronics
- Thrusters and drive electronics
- Jet-Vane actuators and drive electronics

- Scan platform actuators and drive electronics
- Back-up electronics.

Each peripheral is taken in turn and discussed relative to possible BITE approaches.

#### Wide Angle Sun Sensor

- This sensor is used for acquisition (using the processor) and for direct control through the back-up electronics.
- The function is to sense the sun in two axes and provide analog error signal outputs. These outputs limit for large angles and then drop to zero with an unstable, broad null at  $-180^\circ$ .
- Possible BITE approaches:
  - Use of a separate detector to indicate sun presence.
  - Summing of all cell outputs to indicate sun presence.
- Discussion of BITE approaches:
  - Probably neither approach is needed, although neither is expensive. Both indicate sun presence. Absence of the sun and null output would indicate a fault. Also absence of the sun and any combination of two axis outputs short of saturation would indicate a fault (assuming proper relative field-of-view.)
  - The proper functioning of the sun sensor can be determined by the use of the other sun sensors and star sensors and by system level tests.

#### Fine Sun Sensor

- This sensor is used for fine pointing and to update the inertial reference input.
- The function is to sense the sun in two axes and provide digital (probably) error signal outputs. The sensor would probably be a mask-coded digital sun sensor. Beyond the field-of-view, the outputs have no significance.
- The electronics needed would range from simple amplifiers and level detectors for parallel outputs (and moderate accuracy) to this plus

other logic (and peripheral sending circuitry, see Section 3.4.5) for serial interface and more stringent accuracy requirements.

- Possible BITE approaches:
  - Internal LED's to stimulate the cells, used by command.
  - Commutated test of individual amplifier inputs, used by command.
  - Checks of the reasonableness of the Gray-coded cell outputs (only one bit-change per mask transition).
- Discussion of BITE approaches:
  - Commanded stimulus approaches do not seem to be needed for this sensor. Neither do reasonableness checks.
  - Again functional redundancy can be used, particularly with the wide angle sun sensor. Other system-level reasonableness checks can also be used (comparison with gyro data, etc.)

#### Star Sensor and Electronics

- It is presumed that the star sensor is either a single axis or a two-axis sensor with a digital serial output of the error signal(s). The sensor might also have a star magnitude output (digital).
- This sensor is used for fine pointing and to update the inertial reference input.
- The electronics used would be relatively complex and primarily of digital implementation (whether or not an image dissector tube or a solid-state detector were used.) A high-voltage power supply would be needed in the former case.
- Possible BITE approaches:
  - Use a detection of voltage on internal power supplies
  - Use internal detection of illegal counter states, etc.
  - Use a commanded LED stimulus internal to the sensor
  - Detect sensor mode (track vs. acquisition scan, etc.) as being different than proper.
  - Command deflect scan and note proper reacquisition
  - Detect sun entry into field-of-view.

- Discussion of BITE approaches:
  - Functional redundancy can be used, relative to the sun sensors and the internal reference and should be very powerful for fault detection.
  - Any of the listed BITE techniques can be used, with the preference given to those not requiring commanding.

#### Inertial Reference Unit

- This sensor is used for detecting vehicle angular position, angular rates and linear accelerations. It contains gyros and accelerometers. The accelerometers are used during mid-course corrections for on-board control of velocity corrections.
- It is assumed that the gyros are used for measuring rate, and by internal IRU integration, determining position. The IRU may contain one gyro per axis and two IRU's are used for redundancy; or four or more gyros may be used oriented at angles to each other so that any 3 gyros may be used for the 3 axes. In the latter case, more than the minimum 3 gyros may be operated to provide a functional redundancy (which is a form of BITE).
- The loop closure is assumed to be in the IRU for all loops. The interface with the IOU would consist of one serial word per axis of output with the interface asynchronous. Where multiple modes are present (rate/position), the modes would be commandable by bi-levels from the IOU with the same output interface used. Scale factors might also be commandable.
- The IRU would probably contain the AC power supplies needed for the gyros and accelerometers.
- Possible BITE approaches include:
  - Spin motor monitors on all gyros indicate synchronized operation of the gyros.
  - The proper voltage, frequency, phase relationship of the AC supplies can be detected and made into a go/no-go signal.

- A commandable torquer stimulus can be incorporated so that the processor may command torque each gyro (at a pre-wired rate) to provide IRU and system stimuli.
- Limit sensors can be built in to measure saturation of amplifiers, non-presence of torquer pulses, etc.
- o Discussion of BITE approaches:
  - Spin motor monitors should be included as a minimum for each gyro. This covers the most likely failures.
  - The commandable torquer stimulus is a nice feature for use in system fault detection.
  - The other BITE approaches have limited usefulness in comparison with the hardware added, and are probably not necessary.

#### Reaction Wheels and Drive Electronics

- o For reasons explained in Section 3.1.3 , the reaction wheels and their drive electronics should be permanently coupled (paired), per axis from a redundancy and power control sense.
- o The reaction wheel(s) is/are used to torque the spacecraft. This is done by making speed changes from a biased value, with the value either zero or some nominal speed providing net momentum.
- o The reaction wheel contains a tachometer (usually pulse type) to measure the speed.
- o The electronics drives the wheel and closes the speed control loop, using the wheel speed as feedback. The signal to the wheel is normally 2 phase, high power modulated ac. The speed data must be delivered to the processor also.
- o The interfaces with the processor are:
  - wheel speed command from the processor, consisting of a data word proportional to speed (and direction) commanded.
  - wheel speed to the processor, consisting of a data word proportional to actual speed (and direction).

- Other bi-level commands indicating start-up mode or gain changes, etc.
- Possible BITE approaches are:
  - Detectors for wheel voltage, current and/or temperature can be provided to create bi-levels. These parameters may also be delivered to the IOU in analog form.
  - Modes can be indicated (confirmed) by bi-levels.
  - The modulator duty cycle can be provided as a linear signal or detected bi-level.
  - When the difference between commanded and actual speeds is above a certain amount, this can be detected and delivered as a bi-level.
- Discussion of BITE approaches:
  - Data already available in the processor (such as speed difference) should not be separately provided. This includes voltage, current, temperature, etc; if already provided for telemetry. Otherwise, they should be combined for BITE.
  - Mode confirmation should be used.
  - It appears that no other BITE is indicated.

#### Thrusters and Drive Electronics

- The thrusters are used for 3-axis attitude control of the spacecraft and for low-valued velocity corrections.
- The coupling of the thrusters and their drive electronics for redundancy and single-point failure prevention is discussed in Section 3.2.4.
- The resulting interface with the IOU will consist of several words of redundant thruster valve command data. Mode bi-levels may also be included, as required.
- The BITE problem is to detect improper functioning as far "downstream" as possible, preferably to the thrust output of the thrusters. Possible BITE approaches include:

- Detection of actual thruster firing. This can be done by the following techniques:
    - Use of 3-axis accelerometers to determine accelerations
    - Use of gyros to detect rates
    - Use of pressure transducers to detect thruster chamber pressures (or pressure "switches")
    - Use of thruster chamber temperature transducers.
  - Detection of the command signals to the thruster valves.
- All of these BITE approaches must be able to recognize both short (few millisecond) and long (greater than several seconds) firings; both firings that should occur and those that should not. Normally, when pressure switch or valve driver command data is used (the most common techniques), some data storage in the valve drive electronics is needed to accommodate "memory" between processor input cycles. This storage is probably best done by the use of a set of latches (one per thruster) which store whether or not each thruster has fired since the data was last read into the processor. The readout then clears (resets) the latches for new data. This technique allows the processor to determine the valve firings (but not their length, unless it is quite long).
  - The use of other spacecraft sensors (that are needed for normal position sensing) to detect thruster firing, is advisable. On the other hand, use of separate rate or acceleration sensors is not warranted.
  - The addition of pressure switches to each thruster has hardware difficulties and a moderate weight impact. There have also been problems with the reliability of such sensors. Their use is probably not advisable.
  - Monitoring the valve command signals from the valve driver can be (and has been) done, but its worth is marginal.
  - Probably no (separate) BITE from the valves or drivers is necessary.

### Actuators and Drive Electronics

- Discussed here are both jet-vane and scan platform (or other experiment) actuators, which have similar problems.
- These actuators are used to control the pointing of a liquid rocket engine (used for large value midcourse corrections) or experiment platforms. The actuators are linear or rotary and involve stepper or servo motors and gear trains to provide torque.
- The servo loops involved should be closed in the drive electronics, unless (as might occur in experiment drives) the bandwidths involved are sufficiently low that they impose no penalties on the processor cycle.
- For each servo, a position transducer of some type provides position feedback. Rate (velocity) feedback may be derived or may be developed from separate transducers. The position information may also be needed in the processor.
- Normally each drive actuator and its electronics are permanently coupled together and switched as a block for redundancy (see Section 3.1.3).
- The signal from the IOU to the drive electronics normally consists of one data word per actuator containing a position command. As noted, the position feedback may also be delivered back to the processor. Mode, gain or other bi-level signals may also be used.
- A malfunction in the functioning of the actuator and its electronics is probably most easily detected by the effect on the spacecraft or experiment pointing/rates, etc. This can be detected by the use of the normal sensors/experiment sensors, etc. Other possible BITE approaches include:
  - Use of voltage, current, temperature monitors for the actuators
  - Detection of servo loop malfunction (wrong mode, excessive error, etc.)
  - Use of redundant position transducer indications.

Generally speaking, these latter approaches are unnecessary.

### Back-up Electronics

- The back-up electronics consists of a simple analog control system involving the wide-angle sun sensor and thruster modulation control to permit coarse sun-pointing for catastrophic malfunction of the remainder of the control system. It is not used when the processor is functional.
- The back-up electronics is commanded into and out of use either from the ground (directly, not through the processor) or from the processor software or "hard-core" portion if a non-reconfigurable failure condition is sensed.
- It appears that the only "BITE" necessary is an on/off indication direct to the telemetry subsystem and also to the processor. No other fault/failure detection seems indicated.

The BITE techniques that could be used in the peripherals fall into three classes:

- Detecting that something did happen that should have (confirmation)
- Detecting that something did not happen when it should have (absence of confirmation)
- Detection that something did happen that should not have (fault detection) (the true BITE situation.)

A further distinction can be made between:

- Malfunctions that can be detected by other portions of the system (sensors), which accomplishes the detection by a form of functional redundancy. These are system-level fault detections.
- Malfunctions that can only be detected by BITE.
- Malfunctions that can also be detected by BITE and provide an "earlier" or more convenient detection.

It is a contention here, (but unable to be proved), that there are not any malfunctions that can only be detected by BITE. Certainly one wants to maximally utilize the system level fault detections, being restricted only by program size and time available. All peripheral BITE, therefore becomes optional, since none is required. There are some techniques whose

worth is considerable in comparison to the penalties of their inclusion. Those true BITE techniques recommended are: (not included are confirmation techniques, normally considered).

- No sun or star sensor BITE.
- Use a BITE bit for each gyro for spin motor monitoring (this is the highest failure probability item, and a slow degradation of spin speed is hard to detect otherwise, since its first effect will be an increase in drift, which is difficult to detect)
- No reaction wheel or actuator BITE.
- No back-up electronics BITE.

The recommended result is that no BITE (other than the gyro spin motor monitoring already incorporated into IRU's) is needed. This means that existing peripherals need no modifications for this feature.

#### 3.6.2.3 Clock Faults

The processor clock affords a separate situation for fault detection. It is assumed that all of the peripherals use the same clock (since they are controlled by the processor) so this clock becomes the one for the entire control system.

This clock is very vital to the system operation. If the clock fails or stops, then the system fails. If the clock is too far off in frequency, then the system also may fail or be degraded in some way.

In some cases the clock may be supplied from outside the system and be derived from or shared by other systems in the spacecraft. If so, then that clock source must be "perfectly reliable" or must possess the same kinds of fault detection/redundancies discussed here.

For purposes of this discussion, the clock will be assumed to be somewhere in the system and is probably a separate reliability element. The clock source (oscillator) will probably have to be crystal controlled to obtain the timing accuracy and frequency stability desired (if any critical timing functions are needed in the system).

Most processors will require multi-phase clock signals (2 or 4 phase) and the phase relationship must remain proper. The phase relationship of each phase to the others must be maintained. The frequencies (of the phases)

are usually in the range from 500 kHz to 1.5 MHz, indicating an oscillator frequency of from 2 to 6 MHz (for a 4-phase clock).

If any phase fails, that is regarded as a clock failure. Failures can result in (for any/all phases):

- o No frequency - stuck on zero
- o No frequency - stuck on one
- o One clock phase becomes the same as another
- o Frequency too low
- o Frequency too high

The phases (four phases assumed) are normally at a 3:1 duty cycle and are obtained from a divide by four counter (with gating) or from a 4-stage shift register. The resulting phases are shown relative to the oscillator frequency in Figure 3-52.

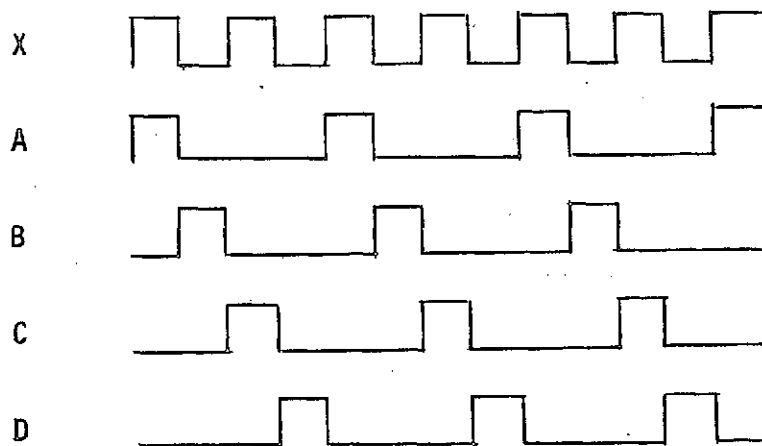


Figure 3-52 Clock Phases

Direct detection of clock phase faults can be done using these five signals. Frequency errors can best be detected on X. This can be done by using one-shot multivibrators and gating. The one-shot is started by the rise in X. A window is enabled that the next rise of X must fall within or else a fault is indicated. On COPE the one-shot was replaced by a delay line.

Another approach is to use a delay line or one shot to delay the X pulses exactly one period and then run the delayed and undelayed pulses into an

exclusive or gate. Its output can be sensed by a level-detector to determine a thresholded degree of frequency difference.

The phase signals can all be and'ed to indicate if they overlap each other (which they should not). This also handles the failures of stuck on one. If they are or'ed, then any gaps (where a phase is missing) will show up as a gap in the or'ed signal. This detects stuck on zero for any or all phases.

The or'ing of these three detectings (frequency not proper, overlap (or stuck on one), and a gap (or stuck on zero) can provide a clock fault signal which can be used to switch the redundancy of the clocks.

The cross-strapping of the clock phases must be carefully handled to ensure no single-point failures. Since the clocks are used by many elements of the system, a clock bus approach is probably better.

Such an approach is shown in Figure 3-53. Here only one (of the five) lines is shown. Both oscillators are always running (but unsynchronized). The flip-flop arbitrarily selects one oscillator/bus to provide all users and the fault detector with that clock. This is done using power-gated bus line receivers for each user and the fault detection circuit. If no fault occurs in that selected oscillator/bus, then it continues to be used. If a fault occurs, then the fault detector causes the selection flip-flop to toggle, selecting the other oscillator/bus. Note that the power to the oscillators could also be switched on and off by the flip-flop, if the start-up delay/transients are acceptable. This saves more power.

Note that if a switchover occurs, due to the non-synchronized phase relationship of the two oscillators, clock glitches can occur because phase C may follow phase A, etc. This could cause other (unnecessary) processor re-configurations. In COPE this was handled by the approach shown in Figure 3-54.

Here, the two oscillators were always running. The fault detectors would select one of the two oscillator outputs to run into triplicated logic and dividers. The four-phase outputs of the counters were voted upon in two sets of voting circuits (one per phase), with one set being for each system bus. This approach is satisfactory, but rather extravagant of power.

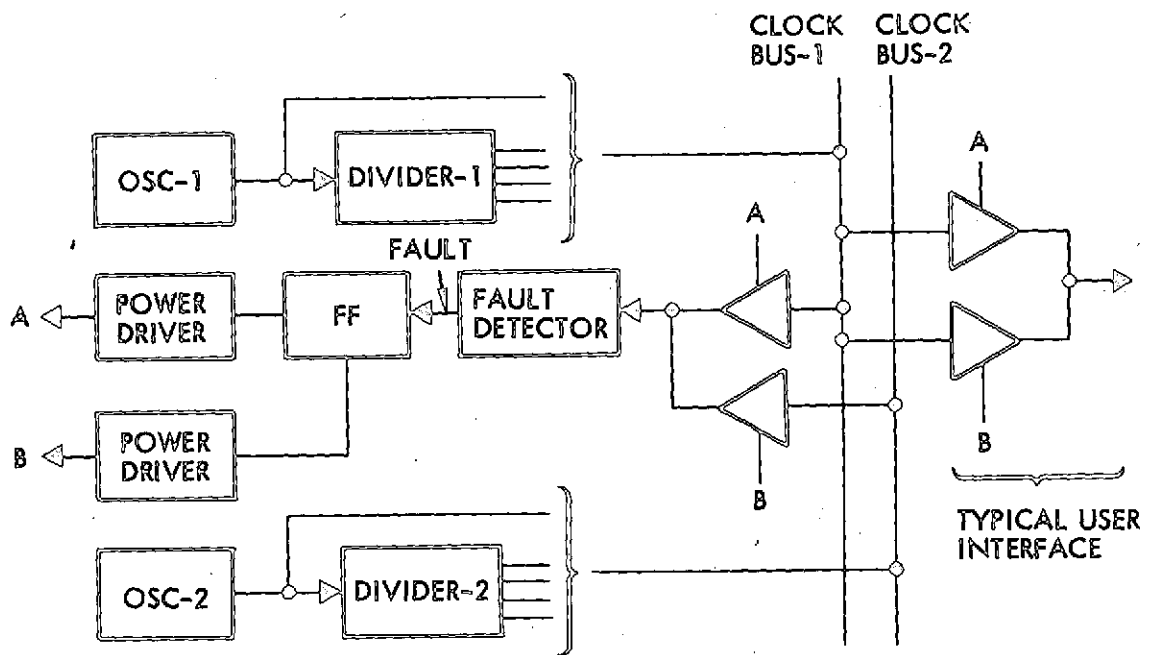


Figure 3-53 Block Diagram of Clock Redundancy - Method 1  
(one line (of five) shown)

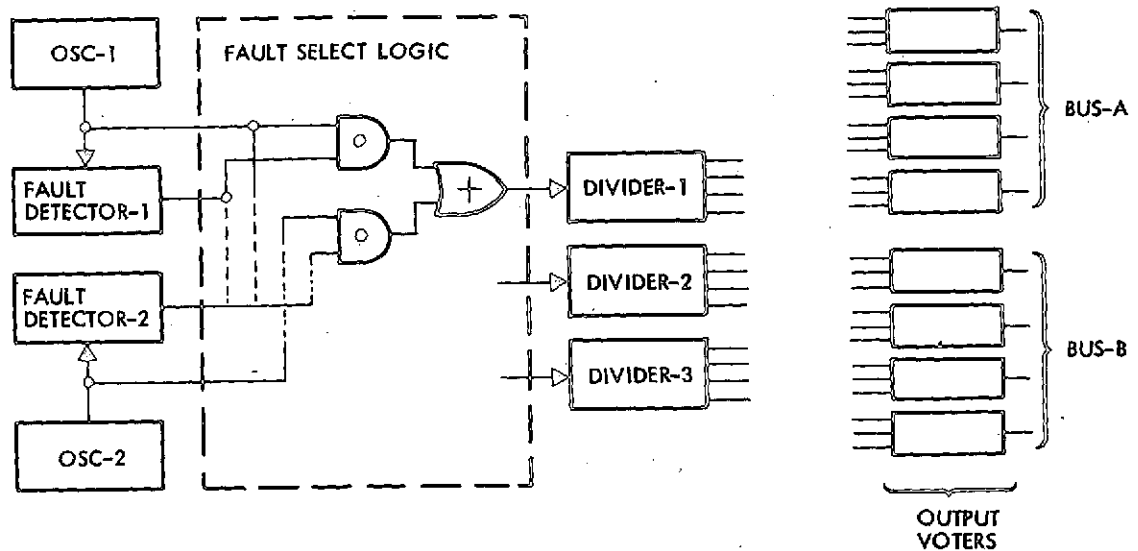


Figure 3-54 Block Diagram of Clock Redundancy - Method 2

Another approach possible when the RCU's are standby redundant and are monitored by a HCU, is to have a clock oscillator and the 4-phase divider in each RCU, with the outputs cross-strapped or bussed together. (See Section 3.1.3).

If a clock stops, changes frequency, etc, this will be detected by a change in the timing or coding of the status word periodically sent from the ACU to the RCU (just as any other fault would be indicated). The RCU will attempt to correct this fault by reconfiguring the primary processor. This will be unsuccessful since the fault is within the RCU. The HCU will detect the lack of reconfiguration and replace the RCU. Restart will then occur and operation proceed satisfactorily.

Other approaches can also be devised for clock maintenance. The important criterion is the absence of single-point failures. After that, power minimization and other features should be optimized.

### 3.7 Fault Tolerance, Failure Detection and Recovery

#### 3.7.1 Fault Tolerance Criteria

Fault tolerance appears as an elusive property because current definitions either lack generality or are rather vague. In fact, most of the definitions found in the literature reflect requirements and/or objectives associated with specific applications. For instance, Hopkins et al [1] define fault tolerance as the ability of a space vehicle to achieve its mission despite component failures or induced errors. In another context, Avizienis [2] defines fault tolerant computing as the ability to execute specified algorithms regardless of hardware failures. A more general view is expressed by Newmann, Goldberg, Levitt and Wesley [3], who consider fault tolerance as the ability of a system to withstand various kinds of hardware malfunctions and mishaps, but with potential flexibility implying the acceptability of incorrect or degraded performance in certain portions or functions of the system.

Man-made systems are programmed or structured to perform specific functions. If the components used to implement a system in this class are ideal (i.e. they cannot degrade or fail), the dynamic behavior is deterministic in the sense that input/output relationships will be strictly as determined by the program or functional structure. When failures in the components may occur, the behavior patterns of the system will be more numerous, since each type of failure produces a new set of behavior patterns (derived from the ideal set). Several of these behavior patterns will be acceptable and the rest will not. Since failure mechanisms are stochastic, a probability of occurrence will be associated with each system behavior pattern. The acceptability of the behavior patterns of a system subject to failures, and their relative probabilities of occurrence, provide general criteria for defining its fault tolerant qualities as follows:

- Definition 1: A system is completely fault tolerant when all its behavior patterns are acceptable.
- Definition 2: A system is partially fault tolerant (fault tolerant) when the probabilities of occurrence of unacceptable behavior patterns are  $p(B_i) \leq f_i$  where the  $\{f_i\}$  constitute the fault tolerance criteria.

A simple example illustrating the concept of fault tolerance embodied in Definition 2 is the power steering mechanism of an automobile. Failures in the hydraulic components produce acceptable behavior patterns because, although more effort is required to operate the steering wheel, control of the vehicle is not completely lost. Fracture of the steering column may cause a series of unacceptable behavior patterns (e.g., collision, abort of a trip, accident, etc.) but since this is a very unlikely event, the steering system of the automobile is still considered fault-tolerant.

One important conclusion proceeding from the definitions is that, in order to evaluate or characterize the fault tolerance of a system on their terms, all behavior patterns must be identified and their probabilities of occurrence must be determined. In general, this is either an impossible or an impractical task for all but the simplest of systems. To reconcile the definitions with the possibility of being unable to identify all the  $B_i$  and compute the  $p(B_i)$ , confidence criteria can be assigned (e.g., in terms of confidence intervals and confidence levels).

The question of how confident the design should be on the fault tolerance of a system is not so critical for a ground-based installation as it is for a space-borne application when the round-trip communication time can be in the four to six-hour range. The autonomy requirements of the application or mission have a significant influence on the definition of which behavior patterns are unacceptable and how much confidence is needed.

A positive approach to improving fault tolerance confidence consists in structuring the system so that the number of behavior patterns is reduced to a number which makes it possible for the designer to identify them all. This structured design concept has been proposed by Dijkstra [4] as a methodology providing a short cut to the well-known, but yet unsolved, software verification problem.

There is a great deal of overlapping between the domains of reliability and fault tolerance, but the main distinction is in that reliability; in the classical sense, is concerned with the ability of a system to perform its functions over a specified lifetime; while fault tolerance is a quality of the system's behavior in the presence of faults.

As is the case for reliability, fault tolerance can be achieved by individual or combined use of design, operating, and maintenance techniques. Design techniques include the provision of redundancy in the hardware and software, recovery management facilities, and self-contained capabilities for in-line functional performance verification. Operating techniques imply the existence and implementation of policies for attaining acceptable system behavior in the presence of faults. Maintenance techniques include software and hardware for checkout and status determination, off-line diagnostics of field units, test equipment, and repair facilities. An important conclusion that can be derived from this list of applicable techniques is that the elements essential for fault tolerance are:

- o Unambiguous policies for establishing acceptable patterns of behavior in the presence of faults.
- o Facilities for implementing these policies (e.g., failure detection mechanisms, redundant resources, controllers).

All systems are, to some extent, tolerant of faults, if they are sufficiently minor. All systems which employ redundancy and are designed for survival of all single point failures are fault tolerant (but may not be tolerant with sufficient rapidity to accomplish system goals). Note that fault tolerance (like all tolerance) has its limits. Eventually faults may occur that are so massive, or frequent, that the system fails, either catastrophically, or into a degraded operation. As with any desirable feature, fault tolerance has its cost. This is in terms of system cost, size, weight, power, speed, etc. The degree of fault tolerance that is included should not be greater than that required. The use of massive redundancy (TMR, duplex, etc) approaches to achieving fault tolerance is uneconomical. These techniques should only be used where there is no alternative, i.e., the need to achieve total availability, with zero loss of time for recovery from faults. This is not the case with any spacecraft control system designed to date.

Fault tolerance can be implemented at various levels, i.e., it may be either centralized or distributed. In the distributed case, the organization may be either parallel or hierarchal. The choice of approach depends on the particular requirements of each application. For instance, in a

simple system, a centralized approach where all fault tolerant facilities are concentrated in a single unit (e.g., redundancy switching network) may be sufficient. In a complex system including a real-time digital processor, however, a multilevel hierarchal organization where each level includes a series of fault-tolerant functions or subsystems may be necessary in order to have a manageable number of behavior patterns. It can be shown that, provided certain requirements on the selection of the  $B_i$  are met, a sequence of two fault-tolerant processes is also fault tolerant.

There are two basic approaches to designing fault tolerant systems: in one, which might be designated as analytical, the system is given, its failure modes are determined and evaluated, and policies to cope with these patterns of behavior are designed and implemented; in the other, the synthetic approach, fault tolerance policies and facilities are designed in parallel with the system which, as a result, can be configured to produce a set of compatible failure modes. The advantages of the second approach are numerous, the most significant being

- Small number of unacceptable behavior patterns ( $B_i$ )
- Low probabilities of occurrence (of these  $B_i$ )
- High confidence levels
- Simple, reliable policies and implementations
- High over-all system reliability
- High degree of autonomy attainable.

Autonomy is one of the essential constituents of a fault tolerant system since, as is the case with all man-made systems, they are intended to be controlled by a human operator. The degree of supervision required from the human operator is what defines the autonomy of a system. Autonomy requirements influence the design of a fault-tolerant system through the definition of which patterns of behavior are not acceptable from the standpoint of compatibility with the human operator interface.

An important observation regarding the human operator is that, in general, he does not supervise system functions directly but, instead, he monitors secondary processes which are indicative of the system status or provide him with measures of performance. The concept of indirect observation of system performance can be applied to the design of efficient and economical failure detection schemes. By analogy to the human operator, who uses

indicators to determine whether a system is functioning correctly or not, a fault tolerant system can use secondary variables or adjoined processes for detecting failures. This approach is very old and has been used successfully in a variety of applications. Its advantages are greater and more obvious when it is applied to a system including a digital processor operating in real time. The cost of failure detection (e.g., redundancy, overhead, failure detector complexity) can be significantly reduced if a simpler process, executed in parallel with the primary one, is monitored.

A digital processor is considered operational when programs, written according to the specifications, run, and the desired results are obtained [5]. This idea can be extended to the multiprogrammed environment where a digital processor and a subset of programs are assumed operational if another subset of programs, specifically designed for checkout purposes, run, and the anticipated results are obtained.

More details on the various points discussed above are given in the following sections. Before concluding this introductory section it is enlightening to mention some well-known features of the human body, which is probably the best example of fault-tolerant design. It includes balanced apportionments of redundancy, failure detection and correction, adaptivity, and degradation capabilities. Not everything is redundant and very few faults (in proportion to the number of elements) can cause complete system failure. There are provisions for functional redundancy and, in case of sickness, non-essential functions are disabled to promote recovery. Built-in recovery and repair facilities are provided. Degradation is gradual and uniform; and wearout is predictable.

### 3.7.2 Fault Tolerance Requirements

The intent of this section is to make precise the meaning of unacceptable behavior patterns in the case of a multipurpose control electronic system, including a programmable digital processor, required to manage attitude control functions aboard spacecraft for interplanetary missions.

The main objectives of fault tolerance in long-range interplanetary spacecraft systems are to prevent

- The occurrence of irreparable failures
- Repairable failures from causing failures of critical system functions.

Fault tolerance criteria are determined by

- Characteristics of the interface between real-time spacecraft systems and the ground control organization
- The criticality of failure modes in relation to their consequences on spacecraft safety and operational success within the time interval from the occurrence of a fault to the successful completion of corrective actions.

Section 2.0 described in detail the mission, and the attitude control configuration assumed for reference purposes, and identifies the following requirements as most significant from the fault tolerance standpoint.

- Availability and Recovery

There are no functions in the attitude control system requiring 100% availability of the control electronics. The system can be allowed to be down for short periods of time without significantly affecting spacecraft safety and mission operations. Maximum allowable down periods are determined by actuator failure modes (e.g., thruster valve stuck open, TVC jet-vane actuator with hardover signals) and typically range from 1 to 10 sec. Recovery reliability is more important than recovery time in the assumed mission and spacecraft.

- Autonomy

The control electronics must be able to withstand, and recover from, upsets caused by external disturbances or internal equipment failures without action from the ground. The ground facilities can provide intermittent supervision during cruise and continuous, round-the-clock monitoring through encounter operations, but the long round-trip communication times (5.3 hours from Uranus) preclude interactive real-time operations.

Based on the preceding general requirements and the system configuration assumed as reference, the following events can be identified as unacceptable patterns of behavior:

- Failure to recover from down conditions in less than 0.5-to-1 seconds.
- Periodic or intermittent down conditions not originated by a sequence of new failures.

- Inability to perform the required functions within specifications, given that fault-free resources are available.
- Failure to report to the ground that a failure has occurred or a re-configuration has been made.
- Inability to detect and correct failures, masked by redundant mechanisms, which may cause other system failures or depletion of resources (e.g., thruster valve leakage causing excessive operation of the opposing thrusters).
- Failures in the control electronics causing prolonged thruster operation and/or hardover actuator driving signals.
- Inability to process ground commands correctly.
- Failure to disable and override automatic functions by ground command.

Failures producing one or more of these unacceptable behavior patterns may be caused by sources either internal or external to the system. Internal causes may be either hardware or software faults. External sources can be either environmental or intrinsic to the controlled processes. Environmental factors define the nature and characteristics of the disturbances or influences the system must handle while performing its normal functions. The controlled processes react upon the system through demand (e.g., power or throughput requirements).

The methodology for providing fault tolerance in a multi-purpose attitude control system will be developed on the basis of the simplified configuration of Figure 3-55, which represents a typical function both structurally and operationally.

The control process can be the orientation of a spacecraft axis in a specified direction (i.e., relative to two celestial bodies) or the pointing of a device (e.g., experiment scan platform) at a moving target. Depending on the types of controlled process and operational regime, the actuators can be reaction wheels, TVC jet-vane drivers, RCS thrusters, or scan platform servo motor drivers. Correspondingly, the sensors can be for either attitude determination (e.g., coarse and fine sun sensors, star tracker) or position indication (e.g., scan platform resolvers, jet-vane LDT's).

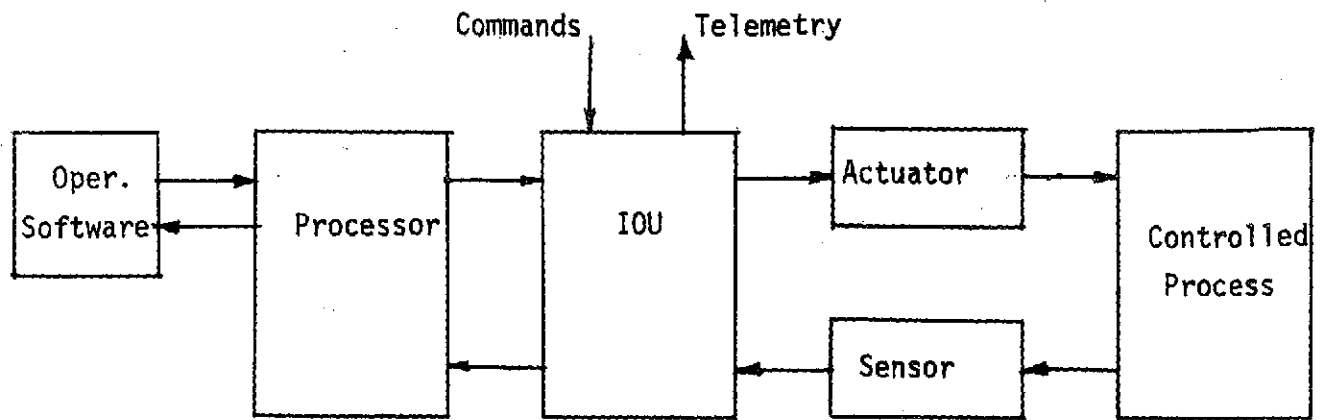


Figure 3-55 Simplified Attitude Control Configuration

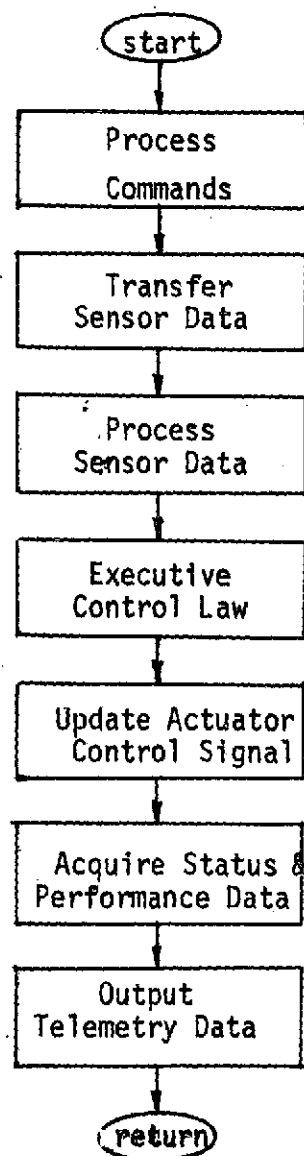


Figure 3-56 Typical Sequence of Operations

Communications between the processor and the ground are through the input/output unit and the command and telemetry systems. The IOU also provides data interfaces between the processor and sensors and actuators. The processor includes arithmetic control elements, memories, and data buses or lines. All functions, as exemplified in Figure 3-56, are performed under control provided by the operational software, based on data and parameters input by command.

The elementary system of Figure 3-55 and the operational sequence assumed in Figure 3-56 include neither redundancy nor any provisions for fault tolerance. They are presented to provide a common basis for the discussions that will follow.

### 3.7.3 Organizational Criteria for Fault Tolerance

The fundamental principle of fault tolerant design is that no system, subsystem, or element can be considered infallible and, thus, control policies must be implemented to either eliminate unacceptable failure modes or reduce their probability of occurrence to the desired levels.

In order to be able to implement a fault tolerance policy, the following elements are required in general:

- Failure Detector

This can be a direct or indirect function performed by either hardware or software elements, or both.

- Failure Corrector

Failure correction implies some form of redundancy which, in broad categories, can be either active or passive. Active redundancy is the approach based on the switching of standby units or elements. Passive redundancy involves the simultaneous performance of a given function by a group of redundant units; if one of them fails, the others resume the function without interruption. Thus, the constituents of a failure corrector are redundant elements and a redundancy management device.

- Buffer

If a failure occurs, the system may be subject to unacceptable disturbances. The buffer is the element providing failure isolation in these cases.

Fault tolerant design requires knowledge of the patterns of behavior of the system in the presence of faults. The problem of failure mode determination is very difficult to solve when the system is provided with decision making capabilities. From the standpoint of complexity this is a situation analogous to the software testing problem. Software testing complexity is a function of the numbers of decision elements and branches in these decision elements. For instance, in a program with 25 decision elements, if each of these elements has 3 outgoing branches the number of test cases required for verification is  $8.473 \times 10^6$ . If each test case takes 1 msec, the complete verification process would require 26 years and 10 months. Clearly, exercising every possible path is out of the question in a case like this, and a more efficient and practical approach is needed. If the system is structured so that operational paths between decision elements are independent and their input-output characteristics are unique and well defined, decision elements can be tested one at a time and their verification would require 75 test cases only. A system of this type can be represented by a directed graph with 25 nodes. Since the number of arcs cannot be greater than

$$N_a = 2 \times \binom{25}{2} = 600$$

the total number of test cases will be less than 675. Assuming 1 minute per test case, the complete operation would take 11 hours and 15 minutes.

One very important observation inspired by the preceding example is that a considerable simplification of the testing has been achieved simply because the test procedure could be organized according to the structure provided in the system. This principle can be extended also to fault tolerant functions such as failure detection and recovery, in which considerable simplifications can be attained by conveniently structuring the system.

The basic ideas of structured design have been formulated by Dijkstra [4] for application to the software verification problem. Structured programming is defined as a methodology for constructing programs with a structure such that, at every stage of the testing procedure, the number of relevant test cases will be small enough to allow trying them all. One of the key

ideas supporting structured design principles is that proving whether a system will be able to perform correctly or not will be very difficult (or impossible) unless the feasibility of validation is considered as one of the essential requirements throughout the entire design process. According to Dijkstra, exhaustive validation of a computer implies feeding it with all possible programs.

Structured design is a procedure based on the establishment of partitions and hierarchal levels and the definition of the interfaces so that specific characteristics of the system elements can be made either visible or invisible at these interfaces.

The basic element in a structured system is a sequential process where the only meaningful characteristic from the logical standpoint is the sequence of states or events and not the speed with which the process evolves. Thus, a structured system is an aggregate of sequential processes. An individual process may invoke or create other processes but in itself there are no simultaneous events. Each basic process is independent of the others and is designed so that all its operations are sequential. Processes may use either real or virtual processors. This is an organization based on subdivision according to a principle of non-interference requiring that

- The performance characteristics of the entire system be determined by the external specifications of the parts and not by the particulars of their internal construction or organization.
- The individual parts be mutually exclusive in the sense that each can be defined and implemented independently from the others.

In a structured organization, each hierarchal level is characterized by a particular view of the actual system. The same system is seen as a different virtual system from each level. For instance, in the "THE" multiprogramming system [4], level 0 allocates processors to processes and handles the real-time interrupt. Above level 0, the number of processors shared is no longer relevant. Level 1 includes the so-called "segment controller", which is a software device that makes actual machine addresses invisible or irrelevant to upper levels. Level 2 is responsible for the allocation of console keyboards via which communications between the

operator and higher-level processes are made. Above level 2, it is as if each process had its own console. The process of defining abstractions continues in a similar fashion until at level 4 one finds the user programs and at level 5 the operator.

Based on the objective of being able to prove correctness of a program, Dijkstra formulated the following rules for structured programming:

- Make complete specifications of the individual parts
- Satisfy yourself that the total problem is solved provided that the program parts meet their respective specifications.
- Construct the individual parts so that, besides satisfying the specifications, they are independent of one another and, also, they are independent of the context in which they will be used.

The principles advocated for structured programming can be applied also to the fault tolerant system design problem, where a unified and integrated approach to the problems of providing autonomy, failure detection, recovery, and graceful degradation capabilities is needed. In a system including a digital processor, software and hardware design activities cannot be independent of each other since there is a significant amount of interaction at all structural and operational levels.

Hierarchical or virtual levels within a fault tolerant system should be determined primarily to meet structural and operational requirements of failure detection and recovery processes. These processes are usually provided with either common or similar structures, since, in an autonomous and remote system, failure detection beyond the lowest replacement level has no justification.

Built-in performance verification techniques or devices not only provide more reliable means for proving corrections but also contribute to reducing the duplication of effort involved in software testing. According to Paige [6], "much redundant examination of a program is inherent in the program proving process since this effort must essentially rediscover the operation of the program".

#### 3.7.4 Failure Detection Processes

A fault is an internal malfunction within a part. Faults may or may not result in observable errors or system failure. Faults may be transient or permanent. A transient fault may be caused by interference, "loose" wires, etc. Faults are associated with hardware; but do not include misuse or mistakes in design. A logic fault is a fault that forces a logic variable into a temporarily or permanently incorrect state.

Some faults at the system level are considered as catastrophic (even though transient). Examples include loss of power, severe environmental effects, etc.

An error is a difference between an actual output or result and the correct output or result. Errors imply incorrectness. Errors may be single or multiple and may be detected or undetected.

Transient faults are probably more rare in spacecraft than commonly supposed. Causes are usually electrical or electro-magnetic in nature, being caused by transient switching of power loads, arcing, radio-frequency interference, the effects of faults in other systems, etc. In a well-designed and tested system, these transients will be minimized. An assumption of the frequency of transient faults and the protection to be afforded (relative to permanent faults) is necessary to optimize the fault-tolerant system design.

The degree of independence of faults is also important. It is important for redundancy reasons that faults be independent from one element to another, as far as possible. (Otherwise all the reliability calculations, which assume independence, are invalid.) Generally, faults within an integrated circuit chip cannot be assumed to be independent and chips should not be shared between circuits which should be independent in faulting. This fault independence is of importance when considering how many bits of a word may be faulty, etc.

A fault may not be detected. It may then either propagate or not. If it propagates it may lead to more faults and/or errors. These errors may vary in criticality and may range from immediate system "crashes", down through the varying degrees of correctness, until the error has no importance and is undetectable. A failure is defined as a critical, uncorrected fault.

Faults, therefore, lead to loss of correctness (by degree) and/or loss of availability (by degree). Availability is the measure of the resource presence relative to the total time.

Most of the work on fault detection and diagnosis has used the assumption that failures will result in a "stuck-at" condition. That is, that a failure in a logical circuit will result in one or more lines out of the circuit taking on a constant logical value, either one or zero. These faults are termed stuck-at-one (s-a-1) and stuck-at-zero (s-a-0), respectively. Apparently most (if not all) failures are of this type and such a model is appropriate.

The fault model may be assumed to consist of single independent faults. This model may be largely fictitious for MSI and LSI circuits, where faults may cause several outputs to fail simultaneously.

In general then, a fault is a necessary cause for a failure but it is not sufficient. There are many situations where a fault either produces no visible effects or masks another failure which, if having occurred alone would have been detectable. In many cases, it is difficult to define a clear boundary between failure and no-failure conditions. Failures, considered here as manifestations or consequences of faults, may be acceptable in some cases. For instance, a stuck last significant bit may have insignificant repercussion in an addition operation, but in a logical operation the consequences may be catastrophic.

Fault detection schemes are normally based on the identification of failures. Since faults may produce no visible effects, failure detection can fail even when the detection device has no faults. This brings up an important point regarding fault tolerant processes: the fact that a system is internally fault tolerant does not imply that the process performed by the system is also fault tolerant. Fault tolerance in a process must include acceptability criteria imposed on the system inputs.

Before discussing ways of providing fault tolerance in failure detection schemes, it is important to examine their general structure. The block diagram of Figure 3-57 shows the component elements and organization of a generalized failure detection process. The primary process performs the required functions, in response to a set of input functions, by means of a

series of subprocesses (only two are shown in the figure for simplicity) which are assumed to be mutually independent. A secondary process, specially designed and constructed for failure detection purposes, operates upon system inputs and primary process intermediate variables and outputs. The failure detector determines whether the primary process behaves acceptably or not by examining the secondary process outputs. The generalized model not only provides a high degree of flexibility for representing most widely used failure detection approaches but also indicates a *modus procedendi* for the synthesis of efficient and economical (from the redundancy standpoint) techniques.

Figure 3-58 is an example showing the classical method for failure detection based on the comparison of the outputs of two identical systems. In this case, the secondary process consists of a replica of the primary system and a comparator. At the expense of 100% redundancy, this approach offers the advantage of not requiring an exhaustive evaluation of the failure modes of the primary system. Comparison schemes minimize the occurrence of failure modes not recognizable by the failure detector, but they can be defeated in cases where system inputs produce unflexed outputs. Depending on the system configuration and speed (or bandwidth) requirements, the operations of the primary and redundant systems can be either simultaneous or sequential. One of the main disadvantages of the comparative approach is that the occurrence of a disagreement does not imply that the primary system is faulty.

The configuration shown in Figure 3-59 eliminates ambiguities regarding the location of a detected failure by using a higher level of redundancy. The secondary process includes two replicas of the primary system and a voting comparator. This method assumes fault-free conditions in two-out-of-three systems when their outputs agree. Normally, only one redundant system is used for comparison tests. The second one is invoked only when a disagreement occurs. One of the main advantages of triple-modular redundant configurations is that in-line failure detection and correction be implemented where high speed of response is required.

Another example of failure detection approach representable by the model of Figure 3-57 is shown schematically in Figure 3-60. The secondary process

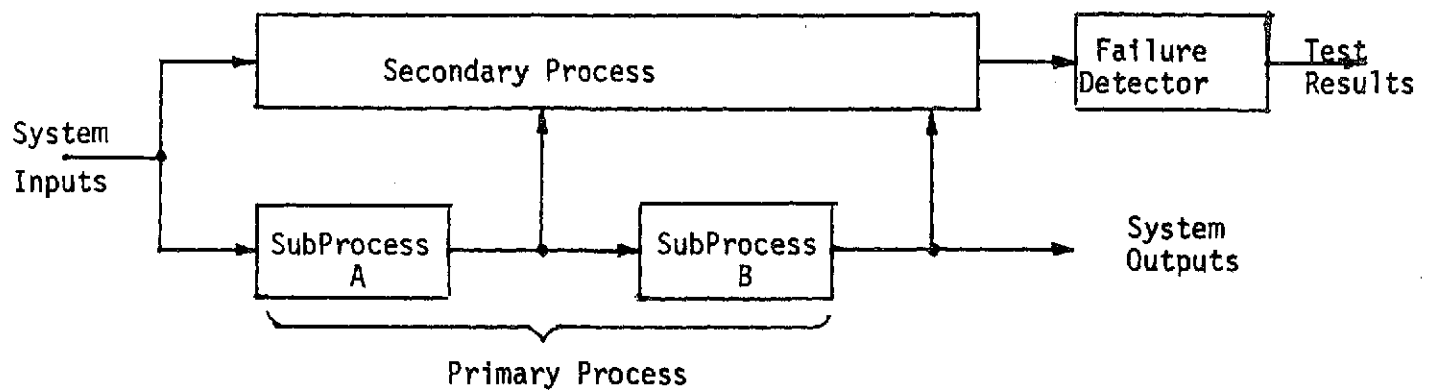


Figure 3-57 Generalized Failure Detection Process

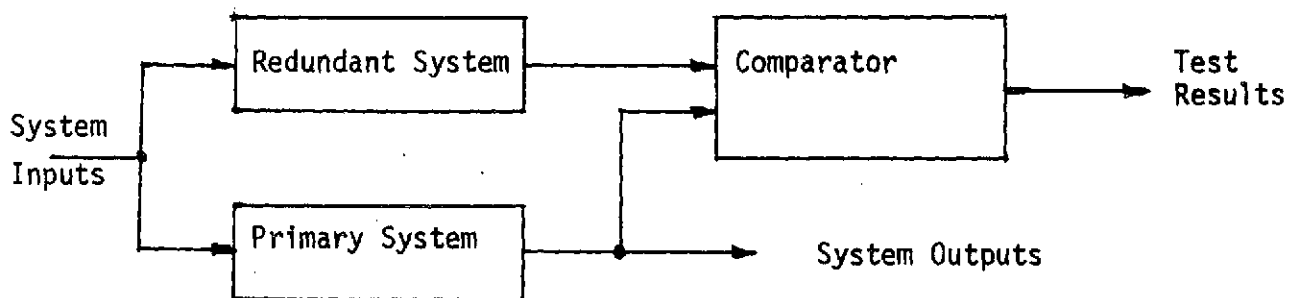


Figure 3-58 Operating Failure Detection

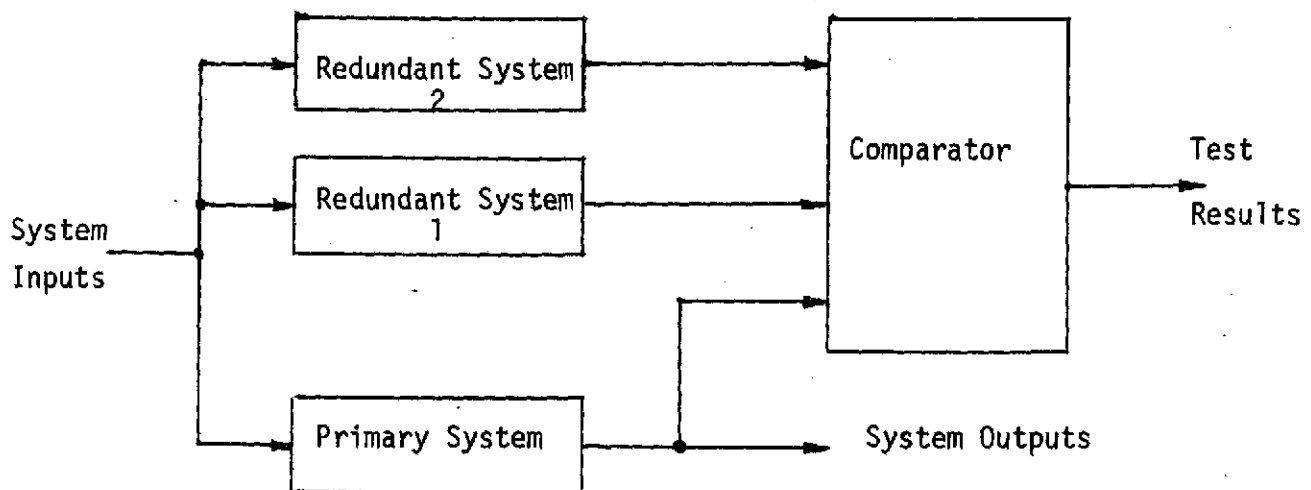


Figure 3-59 Voting Failure Detection and Location

is normalized so that its output meets a criterion of the type

$$F * \bar{F} = Q$$

where the symbol (\*) represents an operational relationship (e.g., sum or product of instantaneous samples, sum of squares, convolution, etc.) and  $Q$  is a normalizing constant or function. Typical examples of normalized operations with  $Q = 1$  are the computation of sine and cosecant functions and the generation of complementary logic functions. Normalization can be done either in parallel or sequentially and its main advantage is that, in some cases, conjugate processes may require less than 100% redundancy. Also, this approach provides greater sensitivity to system faults producing no detectable failures where identical operations are compared. Failure detection by normalization is particularly attractive for software implementation in simplex systems since it provides a more comprehensive verification of machine resources than repetition of computation sequences.

The scheme shown in Figure 3-61 is also based on the use of a secondary process, completely different from the primary process. The figure represents an example configuration where the primary system is checked out by application of a set of predetermined input sequences and comparison of the corresponding responses to stored results. The options exist to perform the tests concurrently with each primary system computation, or periodically, or only when necessary. The choice depends on the performance and diagnostic requirements of each application. Since the primary system can be tested dynamically, high sensitivity to faults likely to produce no detectable failures is obtainable. Failure diagnosis by programmed tests is particularly attractive for multiprogrammed systems, where the primary process is undefined, and applications where the primary systems may be subject to unflexed input signals. The degree of success attainable with this approach depends on the choice of test cases. Every relevant program path or machine resource must be exercised in order to provide adequate validation of the system status. Structured design techniques and path sensitizing methods play important roles in the process of selection of input sequences and states.

Figure 3-62 shows a very simple and effective approach for failure detection in real time systems comprising a digital processor. The idea of using a secondary process for failure detection is exploited to minimize the amount of in-line redundancy required and for greatly simplifying the mechanization of the failure detector. The primary process is subdivided into short execution cycles of equal duration by means of a sequence of interrupts developed by the cycle timer. The primary process is structured so that timing signals are generated within prescribed intervals (relative to the interrupts or reset signals) if, and only if, no failures occur during the synchronous computation cycles. The timing checker develops an output pulse each time a timing signal of the correct type is received within the corresponding acceptance window. If the pulse train generated by the timing checker is discontinued, this means that a failure has occurred in either the primary process, or the cycle timer or the timing checker itself.

There are many types of secondary or adjoined processes that may be used for failure detection. The key problem is to prove that a mapping exists between the failure modes of the primary and secondary processes. Structured design techniques can be used for this purpose as effectively as in the program verification problem. Failure detection processes can be organized in tree-like hierarchal structures where each level operates with its own individual adjoined process. This allows subdividing the failure mode dictionary of the primary system and provides capabilities for checking the checkers. The top level decision making element is the so called hardcore device, which can be monitored only by the human operator.

Failure detection approaches can be implemented by either hardware or software techniques or a combination of both. Software-controlled implementations are very attractive because they facilitate the application of simple and powerful bootstrapping techniques. Hardware monitors are usually required to verify that software failure detection methods are operational. Maximum protection against failure concealment is needed in hardcore devices, which normally are provided with highly redundant failure detection mechanisms.

In a well designed recovery management system, failure detection is usually distributed. Simplicity, safety and efficiency are promoted by decentralization of both failure detection and reconfiguration capabilities. This

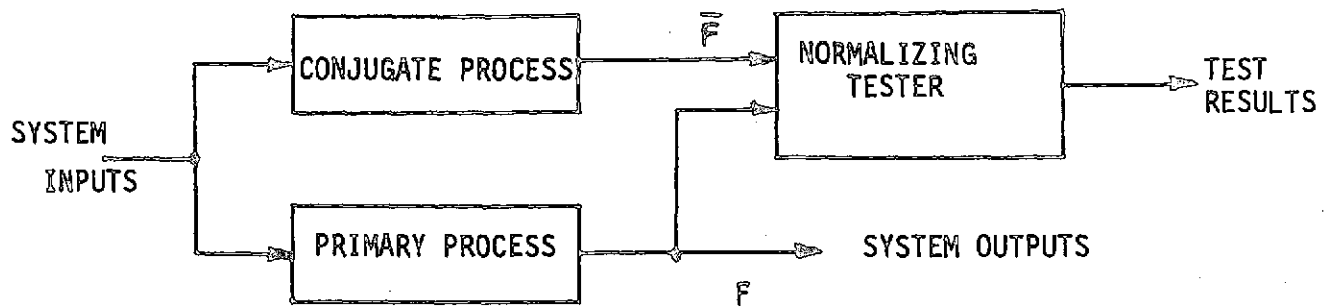


Figure 3-60 Failure Detection by Normalization

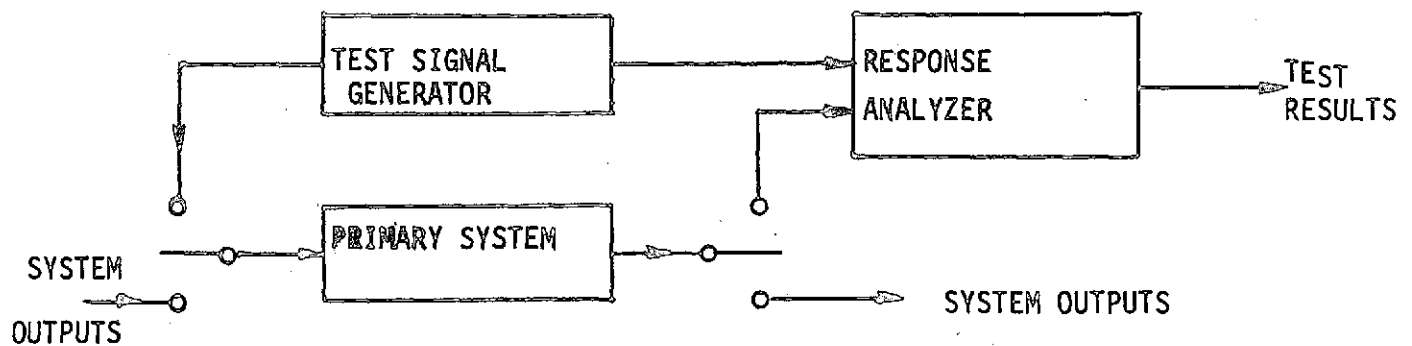


Figure 3-61 Failure Detection by Programmed Tests

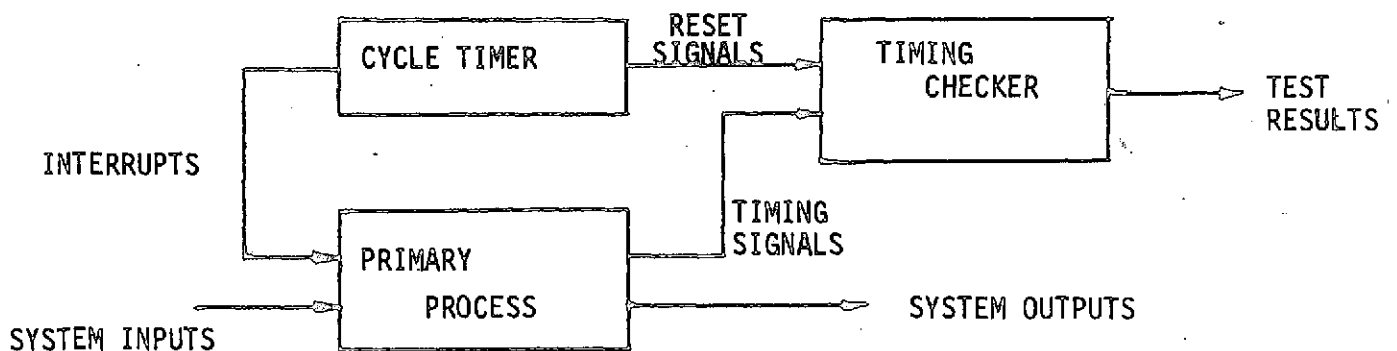


Figure 3-62 Failure Detection by Means of Adjoint Process

approach significantly reduces the complexity of the hardware element which, otherwise, would have to service all system/equipment malfunctions.

Software diagnostics have the advantages of flexibility and compatibility with function and equipment options. Their main disadvantages are the associated computational overhead and requirements for reliable input/output devices and monitoring equipment. The problem with software failure detection is that it does not exist unless all the processing hardware required is fully operational.

Hardware diagnostics have the advantage of remaining operational when the processor is down. In general, hardware failure detection equipment fails selectively instead of massively.

The selection of implementation approaches to failure detection is influenced by a number of factors, the most important of which are the type of recovery management concept adopted for the system and the criticalities of failures (to be handled by each level) from the standpoints of safety and recovery reliability.

#### 3.7.5 Recovery Management Techniques

Recovery management is a function involving the operational control of system facilities to provide a safe and speedy recovery from a failure or unscheduled system interruption or upset resulting from either equipment malfunctions or external perturbations. Its primary concern is to restore system operation (total or as much as feasible) with minimum impact upon the availability of system resources. High degrees of system reliability and availability are obtainable when recovery is organized in a multi-level hierarchical structure. In general, no recovery facility can be designed to handle all types of system (or machine) failures.

In a multipurpose system the recovery organization must be flexible and compatible with function and equipment options. Some recovery facilities must be optional and, as such, must be specified by the user at the time of system definition for a specific mission.

Recovery can be accomplished by either passive or active redundancy techniques. In the passive case, parallel operations are performed in line by a set of identical elements so that a failure in one of these elements will not produce interruptions of the system functions. Typical examples

of passive implementations are TMR and quad logic configurations. In these, internal failures are detected and localized automatically and replacement of failed units can be made at leisure for restoring error correction capabilities.

Active redundancy techniques are used when system interruptions are acceptable. When a failure is detected, standby redundant units are switched into operation for replacing the failed units. The fundamental difference between this approach and the preceding one is that, in the active redundancy case, failure detection and reconfiguration functions are more critical and complex.

For interplanetary spacecraft applications, the use of passive redundancy techniques at the system or subsystem levels is prohibitive in terms of weight and power, and it is not needed because most functions can be interrupted for short periods of time without causing detrimental effects on system performance. Active redundancy methods are preferred because significant economies of equipment and power can be achieved. The hardcore elements, however, must be provided with failure masking protection in order to achieve the required degree of autonomy.

An important problem in the design of an active recovery system is the definition of an efficient rollback strategy. After a failure is corrected, the question is how to restart the system operation. In a real time system controlled by a digital processor it may not be feasible to simply start running the entire set of programs from the beginning, either because of time limitations or as a consequence of inadequacies in the current input and state data.

Rohr [7] defines rollback as a hardware-initiated transfer of control to a software-specified restart address for program resumption after fault detection. This restart address is usually called the rollback point. Program and processor status information must be saved to allow proper initialization after rollback and retry the portion of the program that failed. As pointed out by Rohr, action following rollback depends on the type of activities that were being performed when the failure occurred. In the case of normal computations, it is sufficient to retrieve the latest copy of the rollback state vector and retry the failed program module. If a repeatable input-output function was being performed,

additional operations may be required to re-establish initial conditions. In the case of non-repeatable events, recovery operations may be more complex since there may be circumstances where retry has to take place without repeating some of the events that have been executed already.

There are various strategies that can be used to set up rollback points. As in the SABRE 7090 and IBM 9020 systems, rollback points may be inserted at periodic intervals, irrespective of the particular programs being processed. Another approach, which is preferred for the reference configuration, consists in subdividing the program into self-contained modules, which are executed as a sequence of cycles of equal duration, and choosing the beginning of each module as the corresponding rollback point. This technique leads to a very simple and straightforward rollback organization and is preferred because it has many other attractive features from the fault tolerance standpoint. A third approach, providing flexibility for optimizing rollback strategies for each particular program, has been extensively analyzed by Chaudy and Ramamoorthy [8] and consists in defining the rollback points during the programming effort to meet a specified set of performance criteria. In all cases, non-volatile, secondary storage capabilities are required to preserve state data through recovery operations, and saving these data securely implies some non-negligible penalty in terms of computational overhead. Section 3.1.4 (Redundancy Management) discussed options for the management of system redundancy as involved in recovery. Some of this material is repeated and expanded upon here.

An example showing the basic elements and organization required for implementing an active recovery approach in a real-time control system using a digital processor is shown in Figure 3-63. The hardcore is an autonomous element, provided with a high level of redundancy, which is responsible for monitoring the system performance and performing reconfigurations of the processor elements in cases of failures. Hardcore status and functions can be monitored only by the human operator (i.e., ground station) who, if so desired, can intervene by disabling automatic operations and overriding hardcore decisions by command. Sensors, actuators, and other peripheral devices do not need to be under direct supervision by the hardcore since they can be monitored and reconfigured by the processor through the input-output units (IOU).

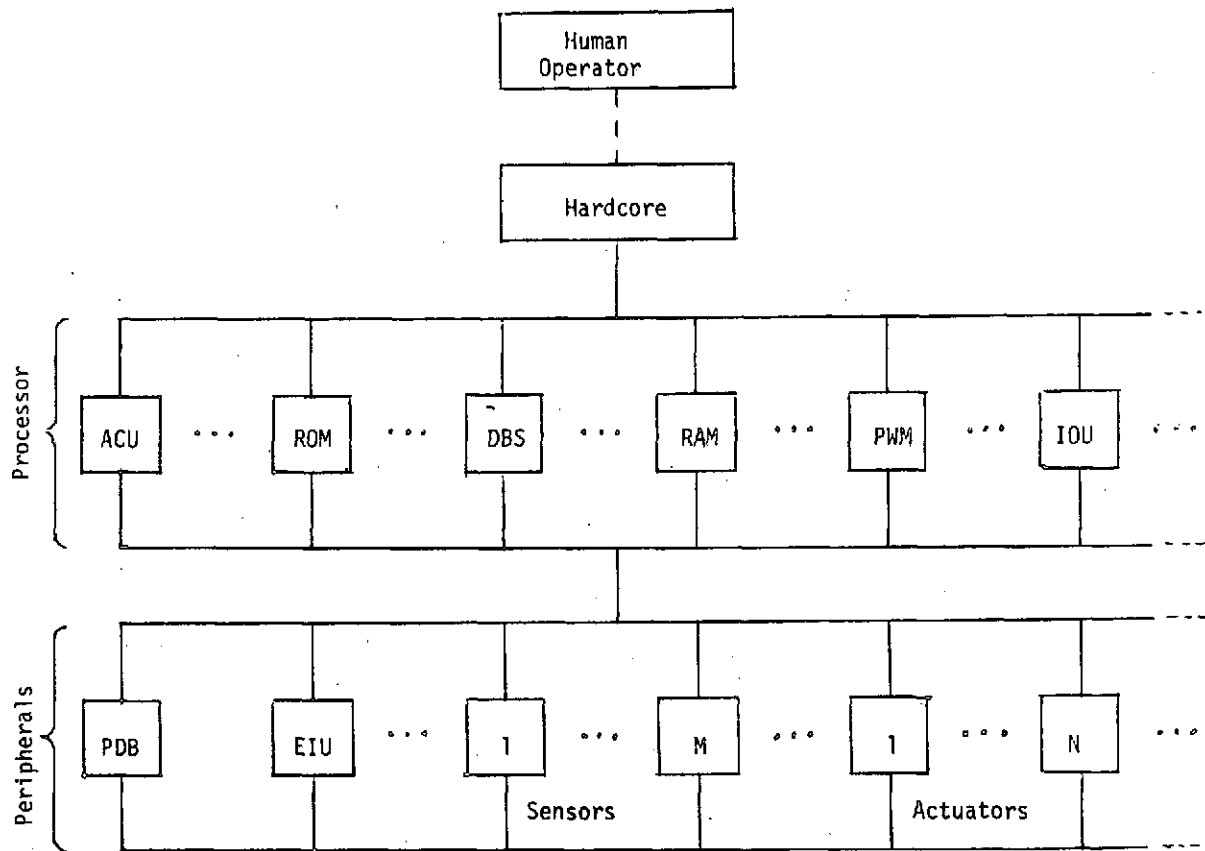


Figure 3-63 Organization For Active Recovery Management

Recovery of the processor elements can be effected by means of either a centralized or a bootstrapped approach. Figure 3-64 shows the centralized concept implemented in the COPE system. The Reconfiguration Control Unit monitors processor performance through a series of hardware- and software-generated fault signals. Redundant processor units are reconfigured directly by the RCU, which follows a hard-wired combinational sequence until signals indicating that normal operation has been restored are received. The reconfiguration sequence starts by switching redundant units of different types one at a time until all the basic units have been tried. If this process fails, a more comprehensive sequence is followed until all possible combinations of redundant elements of different types have been tried. Except for the main data buses (DBS) and the peripheral data buses (PDB), the numbers of redundant units shown in the figure are arbitrary. The DBS are shown separate only to differentiate between parallel and byte-organized data channels. The peripheral data bus (PDB) includes many other signal lines in addition to the serial data channel represented in the figure. The switches acting on the RAM's are intended to show the capability to select two out of four elements.

The bootstrapped approach shown schematically in Figure 3-65 simplifies the hardware by restricting its sphere of influence to only the primary components of the processor. After the hardware determines that the essential functions performed by these components are correct, both the hardware and the elements in level 1 proceed to test and, if necessary, reconfigure the elements in level 2 through the input-output unit (IOU). After successfully completing the bootstrapping recovery procedure, normal operations are resumed and the hardware monitors performance either by means of hardware and software fault signals or by periodically interrupting applications program execution to repeat the bootstrapping sequence, or by means of a combination of both methods.

Further simplification of the hardware can be accomplished by going to a distributed system as shown in Figure 3-66. System monitoring and reconfiguration functions are performed by the reconfiguration control units (RCU) which are standby redundant. Hardware functions are reduced to monitoring and reconfiguring the RCU's by use of a Hardware Unit (HCU), which is passively redundant. Recovery of the rest of the system is by means of a bootstrapping procedure as in the case of Figure 3-65. An important advantage

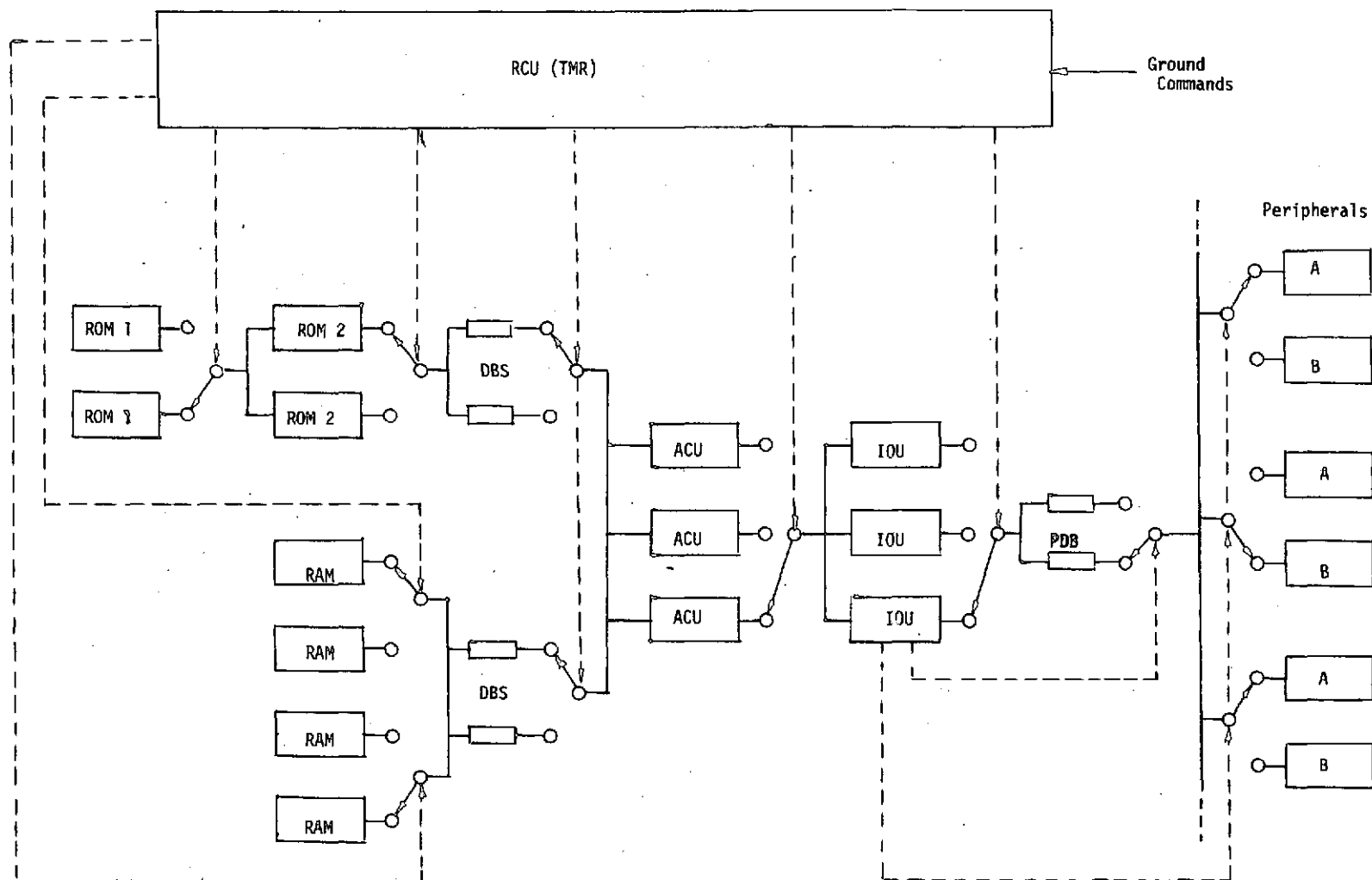


Figure 3-64 Centralized Recovery Approach Used in COPE System

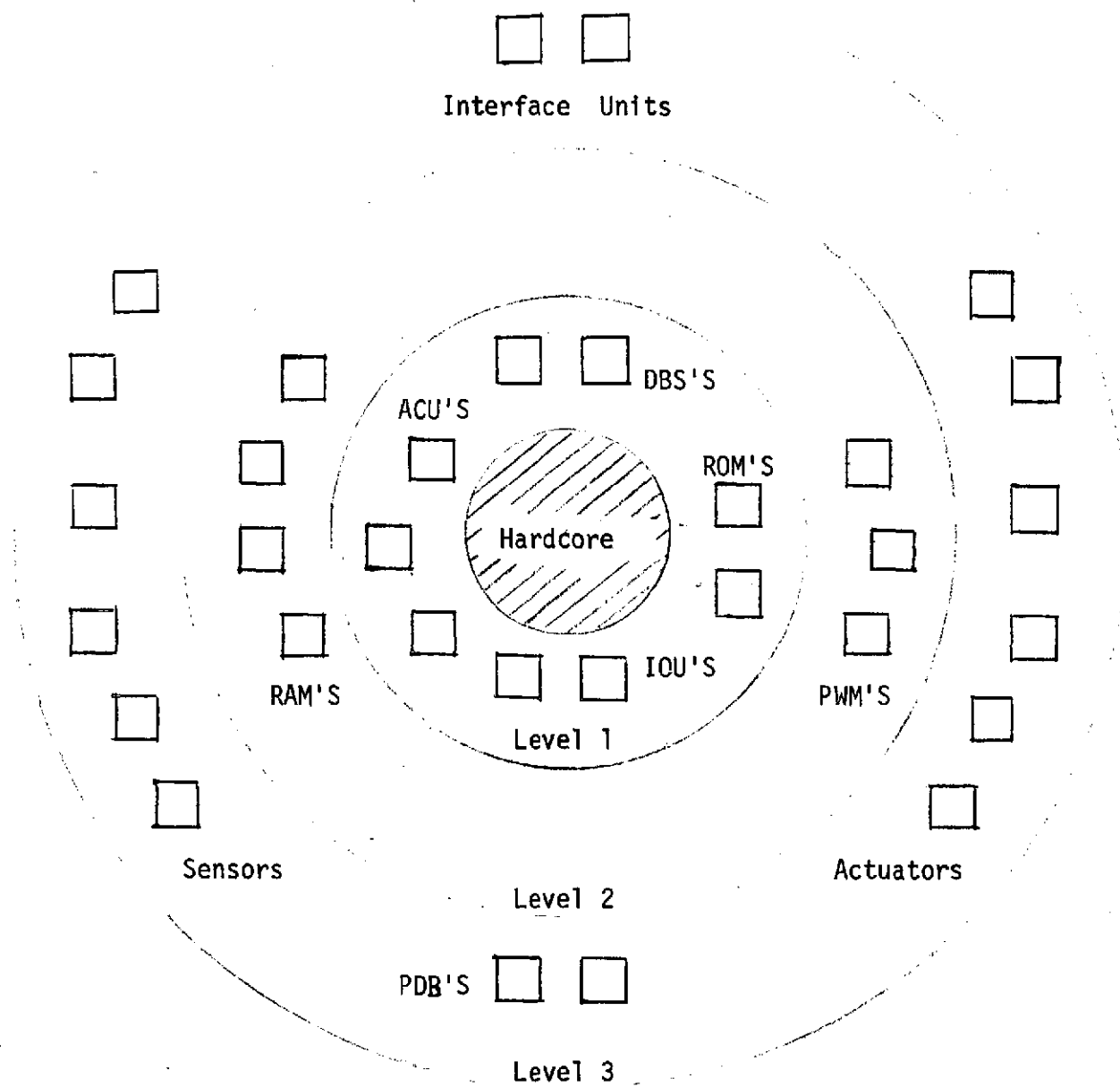


Figure 3-65 Bootstrapped Recovery Approach

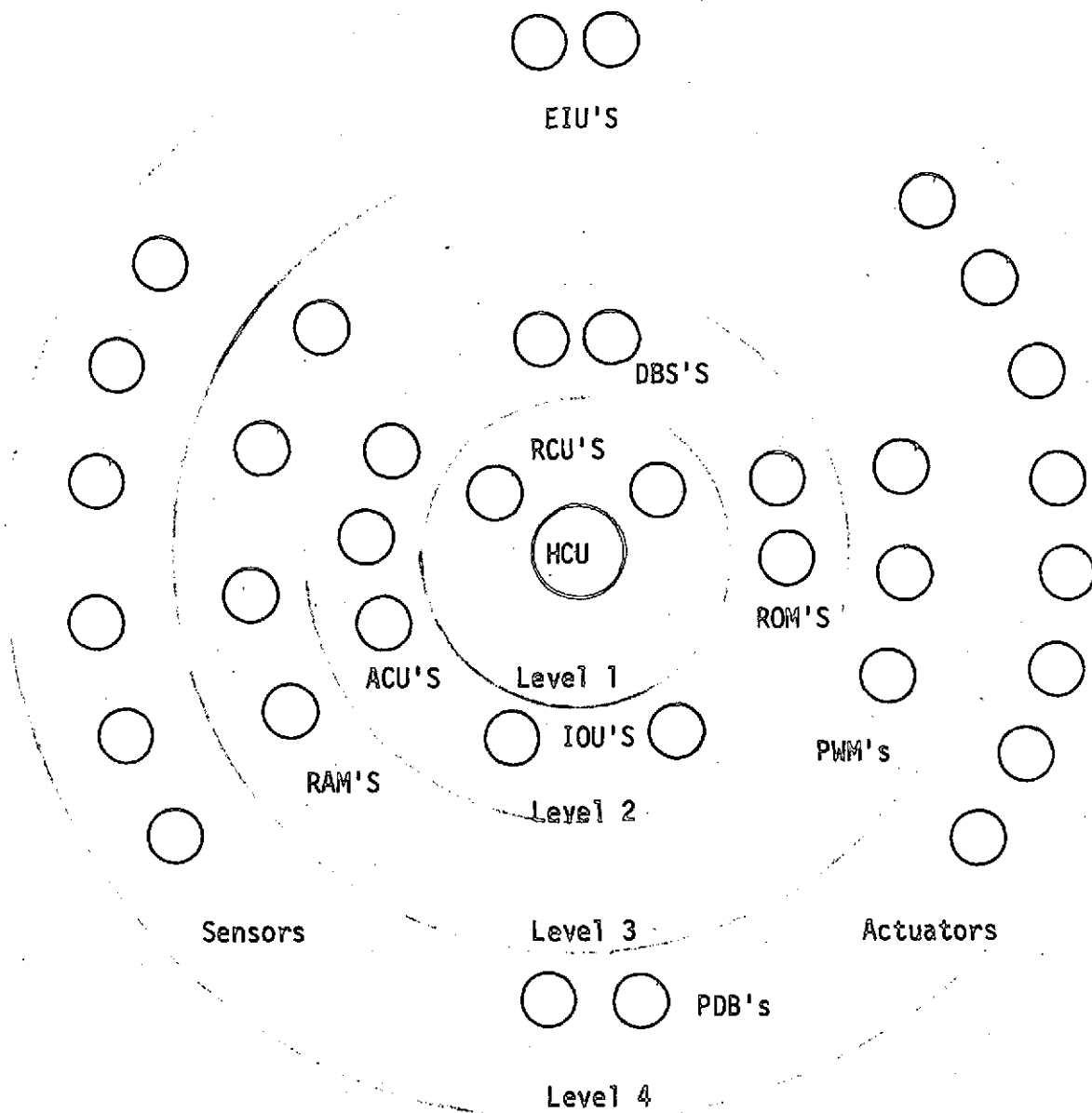


Figure 3-66

Distributed Recovery System

of this approach is the power economy provided by simpler operation of the RCU's.

Other approaches to hardcore system implementation proposed in the literature are discussed in Section 3.1.4 but are not considered as potential candidates for the reference configuration at the present time because considerable architectural changes are required for their implementation.

In the configurations of Figures 3-65 and Figure 3-66 recovery of the primary processor elements can be made by either a simple combinational approach, as in the COPE system, or a strategy based on the simultaneous switching to a redundant set of units preselected by off-line tests. High speed of recovery can be attained by initially switching to a stand-by processor configuration which is known to be fault free. If this action fails to restore normal operating conditions, then a combainational recovery sequence can be initiated to try other combinations of redundant units.

### 3.7.6 Recovery Implementation

In the previous section, the requirement for an RCU and an HCU were defined.

The functional requirements of the RCU are:

- Control the redundancy of the primary processor, including:
  - 2 - ROM
  - 2 - DBS
  - 2 - AU } or 2-3 ACU
  - 2 - CU }
- Issue cycle initiation (program synch) signals to ACU periodically.
- Exchange data with ACU A-register: (16 bits is enough)
  - Output the primary processor redundancy selection status (for telemetry).
  - Accept the ACU coded status word (indicating proper operation).
- Data exchange occurs periodically upon execution of TRC instruction.
- Accepts direct BITE bilevels from primary processor hardware fault detection (if any).
- Checks coded status word for correctness of:

- Proper coding
- Arrival within a specified time after the program synch. signal.
- o Provides for a direct command override of power control
- o Provides a reconfiguration bilevel to the HCU
- o Accepts power control commands from the HCU
- o Provides short-term memory for power outages
- o Reconfigures primary processor if coded status word is incorrect
- o Contains system 4-phase clock and controls clock bus (see Section 3.6.3)
- o Does not need to confirm faults prior to reconfiguring.
- o Does not need any power glitch detector. (Reset is now done by the program synch signal - where needed only.)
- o Uses wire - or'ed power control outputs.

The reconfiguration process should proceed in the following steps:

- o Switch all primary processor units simultaneously to their standby blocks. If no prior failures have occurred, this instantly yields a "new" processor, eliminating the faulty unit, whichever one it was.
- o If this is unsuccessful, then iteratively go through all the unit combinations (16 for the 4 unit types shown). The least reliable (highest  $\lambda$ ) elements should be switched first.

The RCU will require a few direct commands (that do not go through the IOU) from the command system. These may all be bilevels and are:

- o Auto/non-auto reconfiguration
- o Step AU
- o Step CU
- o Step DBS
- o Step ROM

The RCU could provide direct telemetry (not through the IOU), but this is probably not needed. If needed, the outputs can be taken as power bi-level status direct from each primary processor block (plus RCU blocks) into the TLM system. One might also want auto/non-auto status and "reconfiguration

occurring" bi-level telemetry.

There are many functions in the COPE RCU that are not needed with an HCU/standby-redundant-RCU approach. They are:

- Any voting/differencing circuits
- RAM circuits
  - RAM Control Register
  - RAM Sequence Length Counter
  - RAM Initialization Counter
  - RAM Initialization Code Generation Logic
  - RAM Page Assignment Logic
  - RAM Fault (BITE) or'ing
- Module Power Generation Logic
- RCU Telemetry Circuits
- RCU Command Decoder Circuits
- Power Glitch Detector
- System Reset Timing Circuits
- Transient Timeout Circuits (Fault Confirmation Logic)
- Program Cycle Timeout Circuits
- IOU and ROM-2 Control Registers
- Redundant Clock/ Detectors/Dividers/etc.
- All 32 Bits of Communication Register to/from ACU.

The resulting (new) RCU (of which there would be two used per system) is shown in block diagram form in Figure 3-67.

This new RCU would have approximately 10 discretes, 1 analog IC, 8 MSI and 27 SSI for 46 total parts. The old (COPE) RCU had 3 discretes, 5 MSI and 46 SSI in each TMR section; plus 40 SSI in the voting/differencing section; plus 24 discretes, 4 analog IC, 2 MSI and 30 SSI in the clock section; for a total part count of 450.

The functional requirements of the HCU are:

- Controls the redundancy of the RCU's
- Accepts the ACU coded status word, and checks for
  - Proper Coding
  - Arrival at a rate higher than (the longest reconfiguration time plus the normal synch. cycle time).

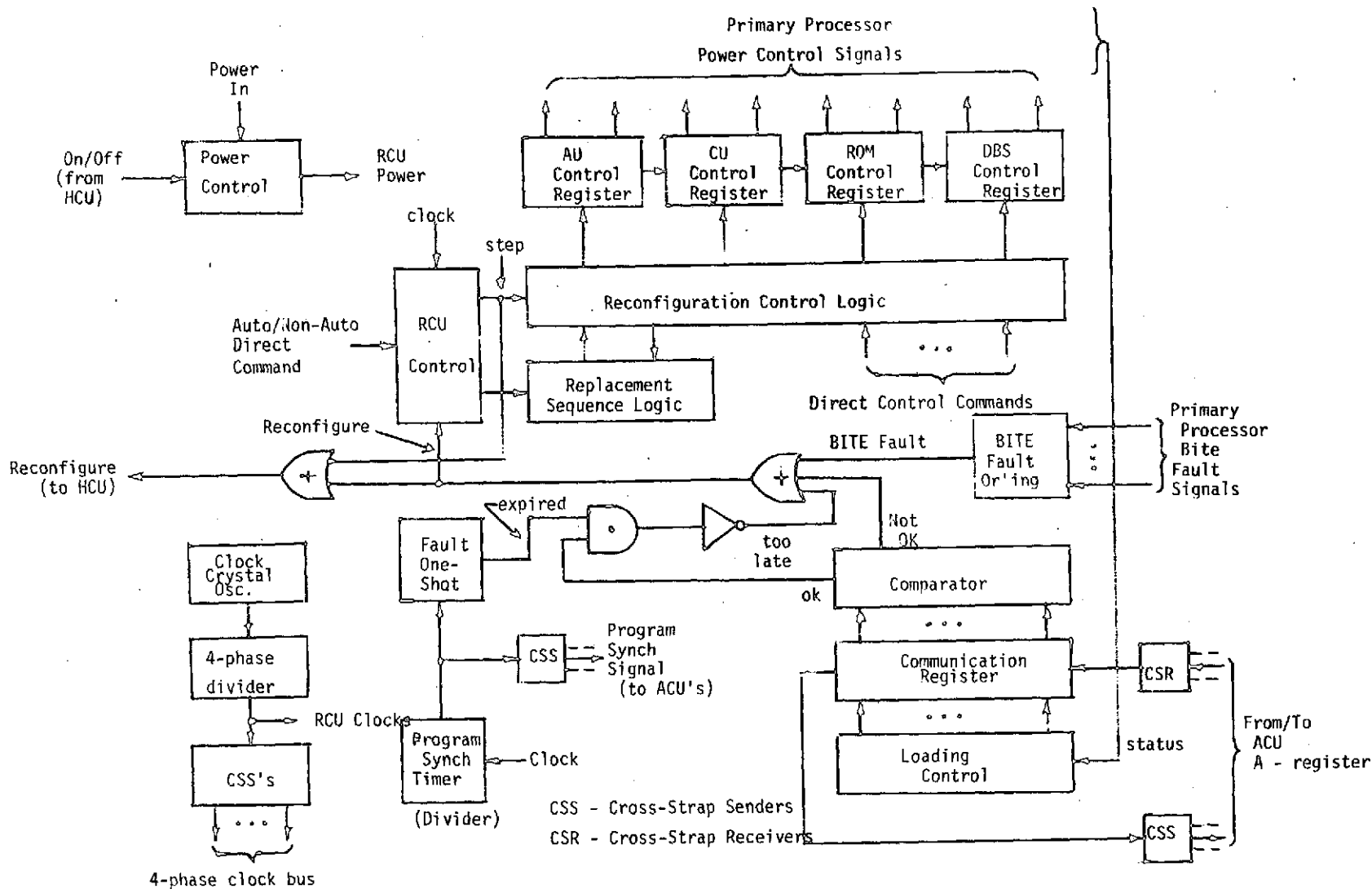


Figure 3-67

RCU Block Diagram

- Changes the RCU configuration if:
  - The coded status word is properly coded and properly timed, but reconfigurations occur: or
  - The coded status word is occurring at too slow a rate or remains improperly coded for too long.
- Accepts the reconfiguration occurring bi-levels from the RCU's
- Operates with its own independent timing/clock.
- Is passively redundant, using TMR with voting
- Provides for a direct command override of RCU control.

An implementation of these requirements is shown in the block diagram of Figure 3-68.

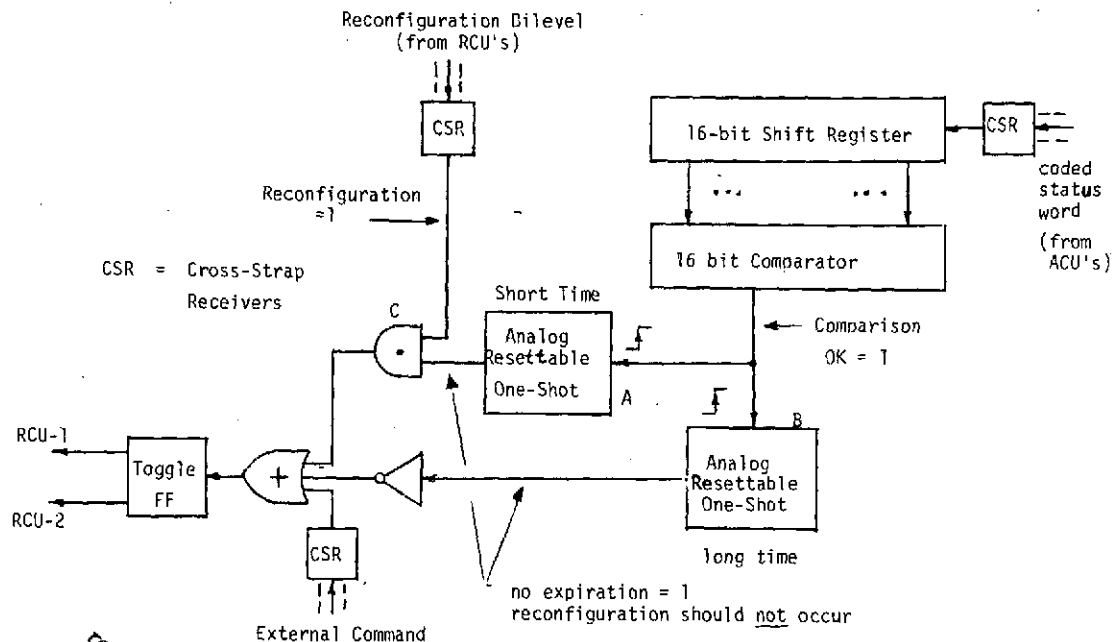


Figure 3-68 HCU Functional Block Diagram

If one-shot-A expires then a reconfiguration should occur. If not, then it shouldn't. Gate-C notes when a reconfiguration should not occur and yet does.

One-shot B detects too long a time to have permitted a reconfiguration. (Either caused by RCU failure to recognize a fault or to do anything about it.)

All of the circuit needs to be triplicated and the two outputs (RCU-1 and RCU-2) need to be voted.

An estimate of HCU part count is 2 MSI and 9 SSI per TMR section; plus 2 SSI for voting for a total of 35 parts.

We saw that the old (COPE) RCU had 450 parts, all of which were always on. The new HCU and two RCU's have a total of 127 parts, only 2/3 of which are on at a time. This is 5/18 the parts and  $< 1/5$  the power. This does not even include the effect of the deletion of the timers from the IOU's. These savings show a considerable advantage for this approach.

### 3.7.7 Backup Mode Implementation Alternatives

Providing backup modes for protecting critical functions is a matter of concern particularly in the case of an autonomous attitude control system for deep space applications. There are many ways in which backup capabilities can be implemented to bypass the primary system in the event that the normal recovery procedures fail. Figure 3-69 shows a configuration where a separate control loop is included to implement a backup mode independent from the processor. This approach has been proposed in several control electronics designs to perform an automatic sun acquisition in the event that attitude errors exceed a predetermined deadband. The backup deadzone is usually broader than the deadband used for normal mode operations. Typically, this backup mode operates with an exclusive analog control channel driving the same RCS thrusters used for normal mode control or acquisition. In addition to implementing control laws for the backup mode, the analog controller provides fault signals to initiate processor reconfigurations. The RCU attempts primary system recovery until the backup mode is disabled. Figure 3-70 shows the failure detection structure of the primary system. Recovery reliability is greatly enhanced by the direct redundant path provided by Sensor 2 and the controller. This configuration

NOTE: Processor And Sensor Redundancies Not Shown For Simplicity

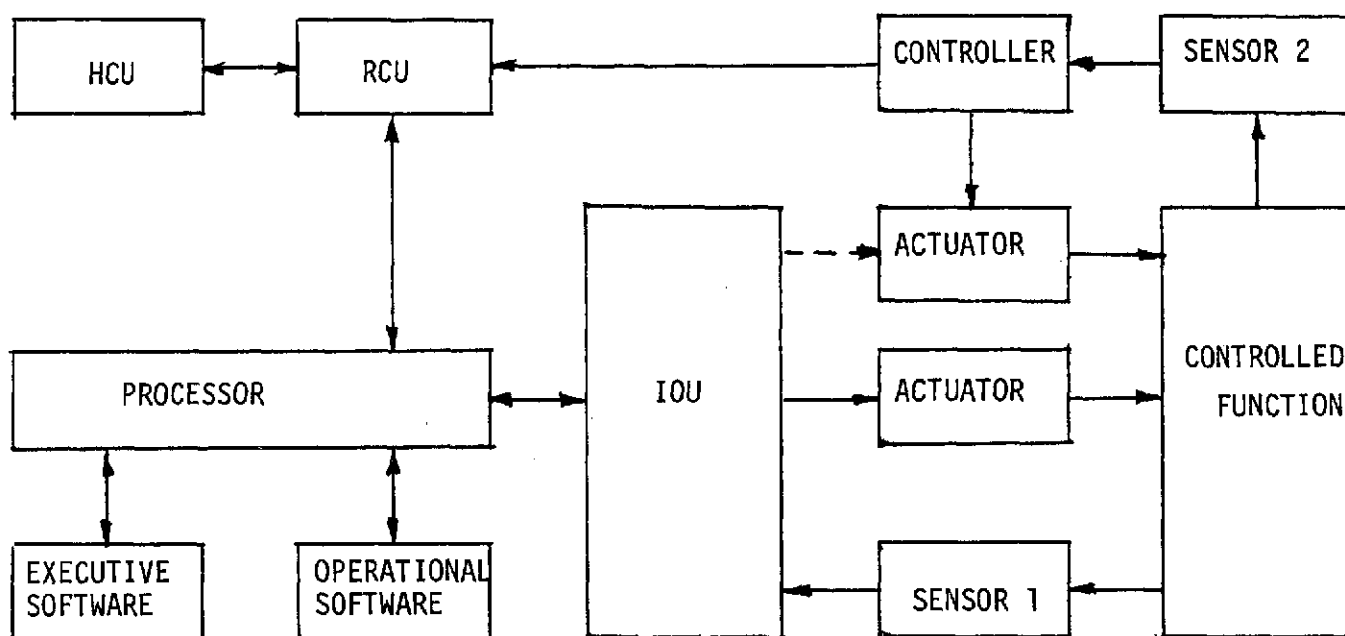


Figure 3-69 Backup Configuration Including Independent Control Channel

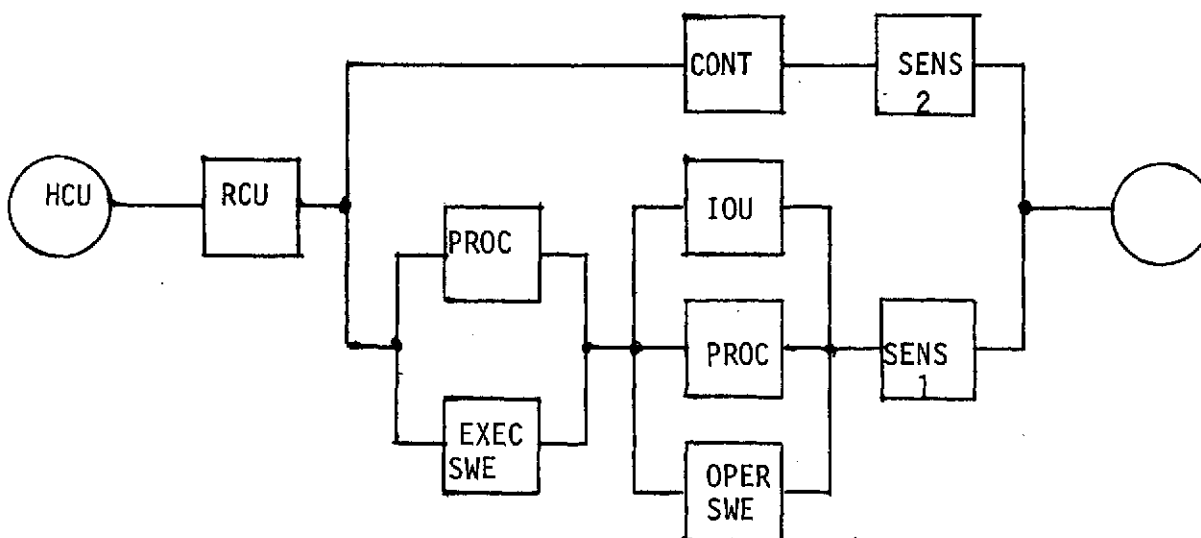


Figure 3-70 Failure Detection Structure For Primary System Recovery In The Configuration Of Figure 3-69.

is very reliable when separate actuators (i.e., thrusters) are used for the backup mode. However, when actuators must be shared with other operating modes implemented by the primary system, many of the advantages of having a separate controller are offset by the additional complexity of the cross-strapping elements required.

The configuration shown in Figure 3-71 features an independent failure detection channel providing a backup to the recovery activation facilities included in the primary system. Controlled function failures are normally detected and corrected by means of a bootstrapped recovery sequence as described in the preceding section. Sensor 1 is tested by the software by means of either activation of a standby redundant unit or application of known stimuli to the system. In case of failure of the processor recovery function, Sensor 2 will eventually detect excessive deviations in the controlled function and will develop fault signals to force reconfigurations until processor operation is restored by means of independent initialization routines. Actuator operation can be reliably monitored by the software. Effects of catastrophic actuator failures can be mitigated by providing a fast recovery capability under software control. Figure 3-72 shows the redundant failure detection structure assumed in the configuration of Figure 3-71. This approach is preferred to the preceding one for the following reasons:

- Recovery times on the order of 1 sec are easily obtainable. Simultaneous actuator and processor failure can be handled without risk to the spacecraft by even a simple combinational reconfiguration approach.
- Reliable recovery capabilities are provided by using a bootstrapping technique (under hardware supervision) and a direct failure detection channel independent of the processor status.
- Power requirements are minimized by restricting the use of passive redundancy techniques. The only element using TMR is the HCU, which monitors RCU functions only.
- A separate backup mode is difficult to implement reliably. Simple and effective redundant control modes can be implemented by means of the processor using fail-safe actuator control techniques.

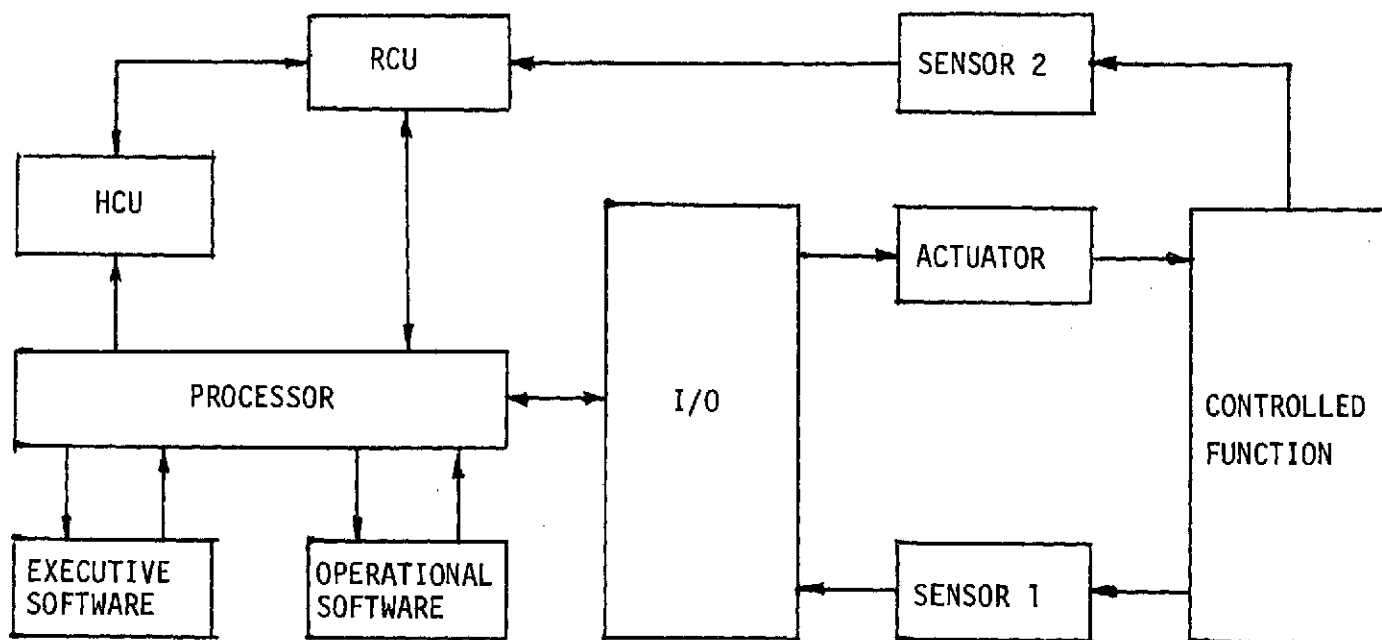


Figure 3-71 Backup Configuration With Independent Failure Detection Channel

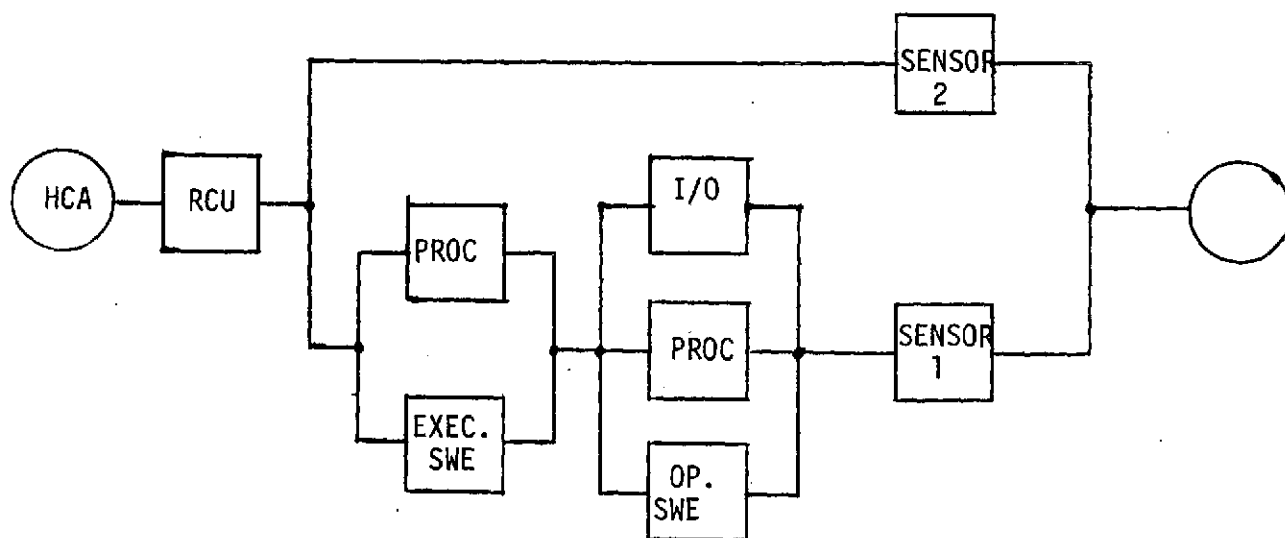


Figure 3-72 Failure Detection Structure For Primary System Recovery In The Configuration of Figure 3-71.

### 3.7.8 Recommended Approach

The recommended approach to providing fault tolerance capabilities in the reference configuration is shown schematically in Figure 3-73. The hardcore unit (HCU) is a triple modular redundant element whose functions are to monitor overall processor performance and to reconfigure the RCU's in case of failure of their recovery functions.

Processor operation is organized in minor cycles of equal duration. At the beginning of each minor cycle, the executive software assembles a status word which must be transferred to the RCU within a specified time-interval from the cycle timing interrupt. If the RCU does not receive a correct word within the acceptance window it initiates reconfiguration of the primary processor elements. Redundant elements are switched one at a time until processor operation is restored.

The HCU contains a one-shot multivibrator which is triggered each time a correct status word is transferred by the processor. The duration of the one-shot on-time is adjusted to be greater than the maximum expected reconfiguration time. If the one-shot signal falls, reconfiguration of the RCU's is initiated.

The RCU's generate clock signals and cycle timing interrupts, and manage the reconfiguration of the processor elements essential to start a bootstrapped recovery process. The first level of recovery is controlled by the executive program which, in the figure, is assumed to be stored in the ROM's.

Once the primary processor elements enable the fault-free operation of the executive software, the remaining elements are tested and, if necessary, reconfigured through IOU facilities.

Rollback points are automatically set by the main program execution scheduling routine which is entered at synchronous intervals established by the cycle timing interrupts.

More details on the hardware implementation are given in Section 3.7.6 and the software organization is described in Section 3.5

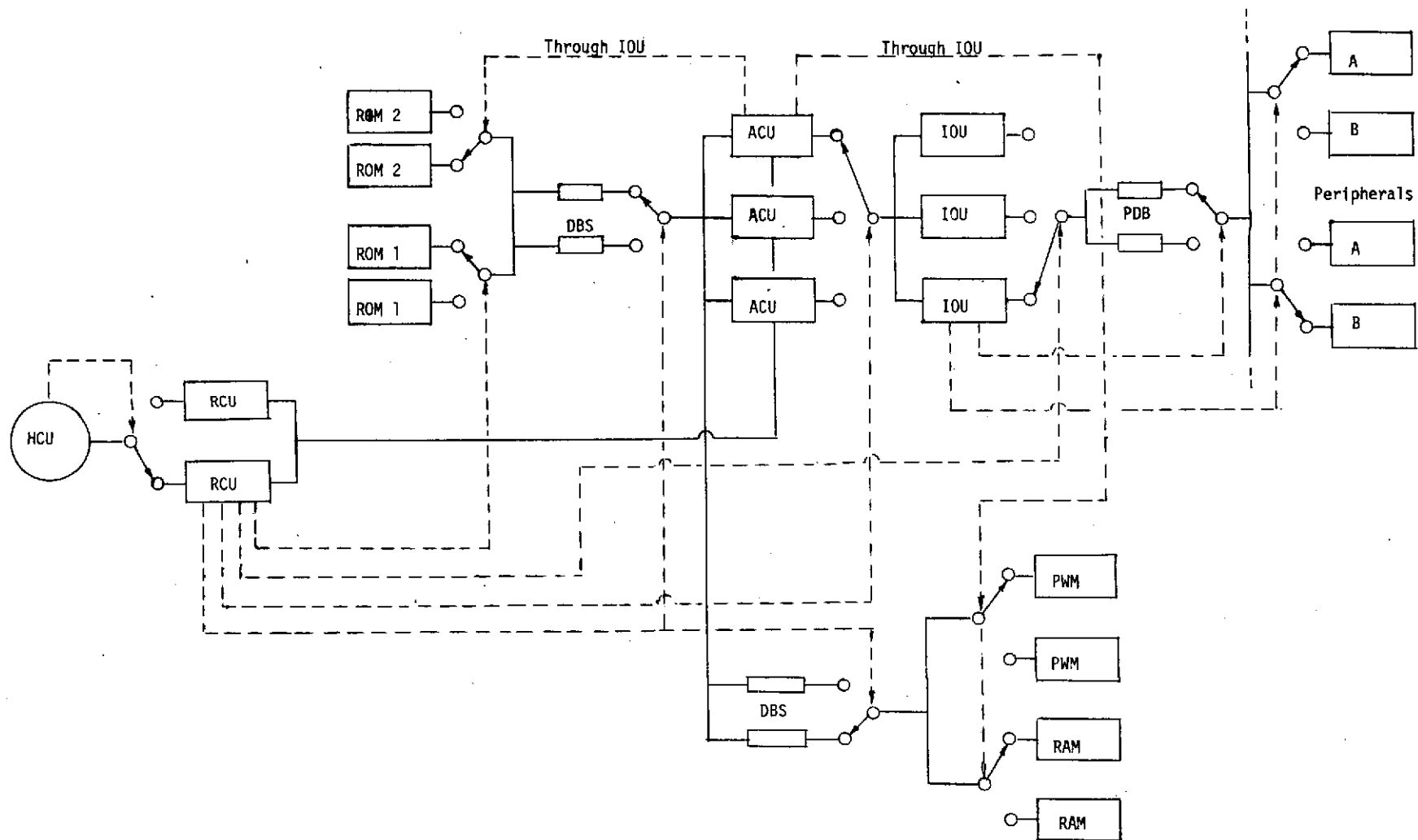


Figure 3-73 Recommended Fault Tolerant Organization

### 3.8 LSI Applications

Large Scale Integration (LSI) was defined earlier as the use of integrated circuits of "large" capabilities, containing the equivalence of hundreds of gates. LSI is a technology that may be applied to bipolar or MOS circuitry. The limitations to the application of LSI are:

- Number of leads. The packages available for integrated circuits have 14, 16, 24, 40, 64, etc. leads available, with the package physical size growing with lead count. Often, a primary limitation of the circuits implemented with LSI is the number of leads necessary for inputs, outputs, power, etc.
- Power dissipation. The amount of circuits put on to one LSI chip can be limited by the circuit power dissipation. This is particularly possible with LSI circuitry. In any case, heavy current driving requirements should not be implemented with LSI.
- Yield. As the amount of circuitry on a chip increases, the yield of good chips goes down (increasing the cost). Below 5% or so the yield is not economical. This is often the most important limitation on LSI "size". Fortunately, this is the area where improvement is being made most rapidly.
- Cost. This is the overriding limitation of LSI. Each new LSI circuit configuration costs a considerable amount of money, just to make the unique masks, and to qualify and prove the design, no matter what the quantity of the order. Unless the new part has a very general or large quantity use, this translates into a very large cost per part; in most cases larger than the cost of the equivalent SSI and MSI parts (including their packaging) that are replaced.

The advantages to the use of LSI are many. Some primary ones are:

- Reduction in power. Although putting the exact equivalent circuits into LSI does not reduce power in itself, usually a slight reduction can be effected since less (equivalent) gates

can be used. Also, often the impedances can be better controlled (optimized) relative to speed and driving capability, also reducing the power. Sometimes the MOS and bipolar parts can be combined in the same chip, using MOS for the logic or memory functions and bipolar for interfaces.

- Reduction in size and weight. One LSI chip takes much less board space than its equivalent in MSI/SSI. This reduces the volume and weight of the total assembly.
- Improvement in reliability. To a large extent, the reliability of an integrated circuit is proportional to the number of leads, since lead bonding seems to be the weakest link. Because of this an LSI part is (or should be) intrinsically more reliable (i.e. lower bit failure rate) than the MSI/SSI it replaces by a factor of 5 to 10 or more times. This is probably the most important advantage of LSI.
- Cost savings. If the quantity usage of an LSI part type is high enough to amortize its development cost, then LSI usage can offer definite cost savings. Beyond the development cost, the per unit cost should be proportional to lead count and yield achieved, with LSI parts costing only a few times the cost of MSI/SSI parts (and much less than the equivalent MSI/SSI ). Added to this are the cost savings of less circuit-level design time, less product engineering (packaging), fewer manufacturing operations, etc.

Wherever the advantages outweigh the disadvantages, LSI should be used. The disadvantage is primarily cost (although schedule, difficulties of qualification, lack of experience in use, etc. could also be important ).

### 3.8.1 Commercial Parts

The LSI parts fall into two categories: those parts that have already been developed and that can be ordered "commercially" and those that use existing technology, methods and processes, but must be designed and masks must be made, etc. Parts using new processes, not yet proven, are beyond the scope of this report.

The main limitations on the use of existing LSI parts are:

- Although there are many LSI parts in existence, most are not applicable to high-quality spacecraft programs. Most have been developed for low-cost commercial use, including calculators, cameras, automobiles, TV, etc. and are simply not usable.
- Existing "computer-type" LSI parts (other than those for pocket calculators) are usually designed for ground-based usage and will not take the environmental extremes.
- The high-volume commercial vendors are not interested in the low-quantity, high-quality market because there isn't enough money there to make it worth all the trouble.

Fortunately, there are some LSI parts available for use, and they are the most important ones, the memories. As discussed in Section 3.4.7, many LSI memories are now available and rapid progress is being made in bringing out new types. Some special-purpose LSI chips have also been developed, but they are usually too special purpose for anyone but the original developer to use and are not generally available. An exception may be some micro-processor chips (see Section 3.8.3 ), being developed under government-supported research programs.

### 3.8.2 Custom Parts

Within the current technological limitations, which really are non-constraining, an enormous variety (infinite.) of LSI part types could be built for any situation where the advantages out-weigh the cost of development. It clearly does not pay to invest any money in memory LSI chips, where other influences are already doing very well in developing new, more dense, lower power parts.

Custom LSI parts are justified for spacecraft use where:

- The parts have a sufficiently general usage that the quantity of use is high (thus lowering the cost), and the advantages of use are multiplied by those quantity numbers. Possible examples are:
  - Cross-strap circuits, bus or conventional
  - Power (redundancy) control circuits
  - Voting circuits
  - Serial data sending/receiving circuits
  - Check byte former/checker circuits

These circuit examples might be used not only throughout the control system, but in other spacecraft systems as well.

- The advantages of use are overwhelming or of so much importance relative to cost. Most likely, such usage will be in the control processor, in the ACU or IOU. Examples include microprocessors (see Section 3.8.3) and microprogramming (see Section 3.8.4). In most of these usages, the quantity of use is low.
- The parts have a high probability of use on other (future) programs. Such predictions are, at best, a risky business.

Some recent approaches have been devised to reduce the cost of custom LSI for the replacement of MSI/SSI logic circuitry. Three approaches have surfaced of interest. Each seeks to solve the problem of offering a low-cost LSI replacement for complex random logic. The approaches are:

- Use of ROM chips
- Use of Programmable Logic Arrays
- Use of Configurable Gate Arrays

These approaches are somewhat interchangeable, but each with its own advantages.

A conventional ROM is organized as having  $2^n$  words of  $M$  bits each. It receives  $n$  address lines into an address decoder. Each address combination selects one word, which appears on the  $M$  output lines. Each of the  $2^n$  decoder states addresses one word line of a matrix of  $2^n \times M$  intersections. Each intersection may be programmed (via mask or "PROM" approach) with a 0 or a 1.

Random logic can be simulated by the ROM by programming the output lines to have the proper correspondence to the input lines. One output word (set of lines) can be addressed at a time.

Although use of ROM's for this purpose is somewhat limited, often considerable logic savings can be realized since ROM's have become relatively low-cost. Also, if PROM's are used the advantages of "field" programming allow considerable versatility and very quick response time.

The ROM offers rigid, exhaustive, non-programmable data addressing, with programmable data. The Programmable Logic Array (PLA), on the other hand, provides more flexibility, with not-necessarily exhaustive, separately programmable data addressing and data.

The difference between the ROM and the PLA is in the address decoding. In the PLA, the address decoding is programmable, as is the data matrix. Each address line (and usually its complement) has an intersection with each of the word lines in the address matrix. Each of those word lines has an intersection with each of the output lines in a data matrix. The PLA might have  $n$  input lines,  $m$  output lines, but  $X$  word lines (with  $X$  not constrained by  $n$  or  $m$ ). A given word line may be selected by any one input or by minterms of inputs. Because each word line can be selected in general by more than one minterm, word lines are sometimes termed product lines.

Because each word line may be selected on a general product term basis rather than on just a minterm basis, it is possible to select two or more output words at the same time. This concurrency occurs when two or more word lines have product terms which share at least one minterm. Such concurrency can not occur in the ROM.

Typically PLA's are available as mask-programmable TTL or MOS LSI chips. Typical parts may also contain flip-flops which may receive inputs from the data matrix and output to the address matrix. Some typical available parts and characteristics are indicated in Table 3-15.

TABLE 3-15  
Typical Available PLA's

<u>Characteristic</u>	<u>Part A</u>	<u>Part B</u>	<u>Part C</u>
Inputs	14	17	13
Outputs	8	18	10
Word Lines	96	60	72
Flip-Flops	0	8	10
Pins	24	40	28
Power (mw)	550	300	300

Note that as each input variable is added the size of a ROM is doubled. This is because the decoding is exhaustive. All the product terms are generated and this soon becomes prohibitive. For example, part B in Table 3-15, if implemented with a ROM would take over 218 million bits, which is a ludicrous size for any ROM (if the PLA uses its full capacity).

PLA's are programmed by writing the outputs in terms of the inputs, expressed as sums of minterms. One PLA of the sizes indicated in the table can replace hundreds of gates and a great many MSI/SSI parts.

Another LSI logic replacement approach is the Configurable Gate Array (CGA). This has been developed by TRW as a universal array of gates which features quick turn around time from design to hardware and low initial design cost. Discretionary means for configuring the logic gates at the component level in addition to the intraconnection has been provided.

The TRW CGA consists of 120 4-input gates located internally within the chip, and 60 2-input gates and 60 inverters located around the periphery at the 60 pad locations. The logic power of this assembly is greatly enhanced

by the gate configuration, i.e., the gate is not of a fixed type but may be made into NAND, NOR, etc., to fit the specific logic task at hand. The CGA uses the high yield 3D process and EFL circuitry, but the gate functions emulate series 54 and series 54L TTL. Therefore, the logic designer need only use familiar methods to design with the CGA.

To expand upon this simple scheme, the configuration of the gates is carried out by discretionary connections made at the component level so that the gates may each be configured into a whole host of useful types. Intra-connection between the gates and the pads is also retained as discretionary. The term "discretionary" is used here in the sense that it is at the option of the designers through computer aids to define the intraconnection paths and also the type of gates which are connected. The gates used internally within the chip can be configured to form any noncomplemented Boolean function of four (or fewer), variables. One hundred and two such configurations exist within the sets:

- NOR
- NAND
- AND-OR-INVERT
- AND-OR-AND-INVERT
- OR
- AND
- AND-OR
- AND-OR-AND.

Each gate also has additional options for:

- High level logic
- Low level logic
- Totem-pole outputs
- Open collector output
- Tri-state control output for inverted output logic gates
- Resistor terminated outputs
- Low power option, low speed, series 54L emulation
- High power option, high speed, series 54 emulation
- Passthrough or passive option wherein the gate space is used to facilitate intraconnection within the chip
- Unpowered, unused gate option.

All options, configurations, and intraconnections operate on the principle of defining where metal to silicon contacts are made. Therefore one mask, the contact mask, contains all information necessary to configure the entire LSI array to the application. All other masks are fixed and not necessary to change from application to application. What this results in is a rapid turnaround time for an LSI with low nonrecurring cost. However, this LSI is more limited in electronic function capability than a custom LSI design and uses somewhat more power per gate. The limit on electronic function capability can be estimated from the following list of available pads and gates:

- 120 4-input gates
- 60 2-input gates
- 60 Inverters
- 60 Input-output leads or pads
- 4 Power and ground pads.

Fewer gates are needed in the CGA than in fixed gate types. CGA wafers are processed and stockpiled up to the contact mask. Once the unique contact mask is received in the fabrication area, the requisite number of wafers are removed from stock and the processing is completed within four working days. Variability in the time required to package, test, and deliver LSI chips is based on inspection and test requirements, but these are known routines subject to reliable scheduling.

Use of either PLA's or CGA's in the COPE ACU could serve to reduce the current IC count from approximately 110 parts to 10 to 15 parts, with a small power saving. This assumes no change in topology (micro-programming processing, etc.). Lesser savings could also be achieved in the IOU.

### 3.8.3 Microprocessors

A microprocessor is a processor organized around the use of LSI chips. Most microprocessors are general-purpose digital computers designed as replacements for minicomputers or for use in instruments or a variety of consumer products.

The "one-chip" microprocessors put both control and arithmetic on one chip, then use other chips for memory and I/O. The larger, multiple-chip microprocessors use one chip for control and break down the arithmetic into modules, each on a chip.

The one-chip microprocessors are usually 4 to 8 bits in word length. Most 16-bit microprocessors use 5 or more LSI chips for arithmetic and control. (One chip per 4-bit byte, plus a control chip). The control chip may be organized with microprogramming. (See Section 3.8.4).

Existing microprocessors now contain such features as:

- Compatible peripheral interfaces
- Powerful, large (>50) instruction sets
- Ability to address large memories (>32K words)
- Priority interrupts
- Subroutine nesting
- Direct memory addressing
- Large I/O addressing capability
- Up to 16 bit word length
- Multiple general-purpose registers
- Microprogramming

The faster microprocessors have speed capabilities suitable for use on spacecraft control systems. For many of the devices restrictions on pin (lead)count often lead these microprocessors to use serial data transfer

and/or various tricks for addressing or memory/ACU address and data exchange. These tend to slow the operational speed of the processor down. Both parallel address and read/write (or I/O) buses are advantageous. The processor may be byte organized, however.

Microprocessor chips are also available, wired together onto cards, in a variety of configurations. Most of the currently available microprocessor LSI is in MOS, but bipolar versions are being developed.

It is interesting to note that just as Polaroid builds cameras so they can sell film, and Mattel produces dolls so they can sell their clothes; most microprocessor chips have been developed by vendors so they can sell more memory chips (because that is where the money is). No such impetus apparently exists in the aerospace microprocessor field.

All available microprocessor chips (that the authors are aware of) are designed for the commercial field and do not have the high environmental resistance, broad temperature range, or qualified status needed for spacecraft usage. Some companies (including RCA, TRW and Honeywell) have built ACU's completely from LSI, although not necessarily for control system use.

The TRW microprocessor is called Multi-Purpose Processor (MPP). It was developed primarily for signal processing, real-time data processing and secure communications systems (and not for control systems). It is microprogrammed and has considerable I/O and memory flexibility. It is quite fast, using triple diffused bipolar LSI, is 16 bits parallel in arithmetic organization and has short instruction execution times of 300 nsec. The MPP uses 11 chips of 5 types for its ACU. The Honeywell microprocessor has about the same number of chips and other capabilities, but is much slower.

No existing microprocessors can be applied directly to spacecraft control system processing (with or without fault tolerance) for a variety of reasons:

- Reliability organization. No existing micro-processor is organized for a sufficiently low level of redundancy. (Which after all, does not have to be as low as for a "normal" processor because of the lower failure rates per function.) None of the micro-processors use cross-strapping circuits for inter-block fault isolation (although this might be done external to the LSI chips).
- Environmental Considerations. Design for spacecraft environmental extremes is usually lacking.
- Quality. Control of processes, qualification, etc. for high-reliability programs is missing in the part production.
- Fault Tolerance. No consideration has been given to fault tolerance in either hardware or software.

The preceding does not mean that microprocessors could not be designed to meet all of the criteria necessary for a spacecraft control system fault-tolerant processor. It only means that this has not yet been done. If sufficient money were available, it should be done. Such a processor would:

- Require about 10 chips for the ACU and 3-5 for the IOU.
- Use the same memories as a "conventional" processor (although the non-LSI parts of the RAM, ROM and PWM could also be made using LSI PLA's, CGA's, etc.).
- Reduce the size, weight, and volume of the processor considerably.
- Increase the processor reliability.

No particular unique design changes appear to be necessary in the software or hardware organization to convert the COPE-type processor to an LSI micro-processor. All aspects of the preceding discussions of Section 3 appear to remain valid. It would be desirable to also convert the RCU and HCU to LSI. This might be done in each RCU for from 3 to 7 chips and in the HCU for 1 to 4 chips, depending on partitioning, etc.

Note that there are no technical restrictions to building a spacecraft control system fault-tolerant LSI microprocessor. The only restrictions are money and desire.

#### 3.8.4 Microprogramming

Microprogramming may be used in either a microprocessor or a "conventional" processor, or it may not. Microprogramming refers to all of the following specific features:

- Use of registers in the ACU that are general-purpose.
- Ability to "easily" change the micro-instruction set.
- Ability of a single instruction from the program to call up complex ACU operations, involving smaller programs or sub-routines, all intrinsic in the instruction, and totally residing in the ACU. For example, a multiply, or  $\sin X$  or  $\arctan Y$ , etc. may be called up by a single program instruction.

Some processors that are not microprogrammed contain some of these features. For example, COPE uses single instruction multiply and divide. All of these features can also be differentiated by degree (number of registers, ease of microprogram change, number and complexity of operations, etc.).

Microprogramming can offer some or all of the following advantages:

- Reduction in software programming time and effort. Complex arithmetic computations can be done with fewer instructions. More and more-general-purpose ACU registers can simplify data manipulations and reduce access to RAM. This can also increase processor effective speed.
- The instruction set can be changed relatively easily from program to program. Actually this is not much of an advantage and has little usefulness relative to its cost (new LSI chips).
- It may save parts in the ACU. This is really a savings from use of LSI and not just from going to microprogramming.

For purposes of a controls processor, the replacement of the control logic in the ACU could be done (relative to COPE) with LSI at a considerable savings in parts. At the same time, the changeover to general-purpose ACU registers (and their possible increase in numbers) would be a good idea. Consistent with op. code quantity restrictions, some more operational instructions could be added at the same time with no hardware penalty. All of the advantages could thus be obtained.

The COPE ACU now has the following relatively special-purpose registers:

- ⊙ A register
- ⊙ Q register
- ⊙ Memory register
- ⊙ Index register
- ⊙ Data base register
- ⊙ Instruction register
- ⊙ Address register

All of these but the last two could be made general purpose. An expansion in the number of these registers could also be made (which is like moving some of RAM into the ACU). All general-purpose registers would require access to:

- ⊙ Adder (arithmetic element) input and output
- ⊙ Memory bus input
- ⊙ Memory bus output
- ⊙ Any other register input or output
- ⊙ I/O bus (if used).

The use of PLA's or CGA's for the control logic replacement is one possible approach. The logic could also be replaced by custom LSI chips in a microprocessor approach. Again, money is the restriction to such changes.

#### 4.0 SYSTEM DESCRIPTION

This section of the report summarizes the results of the previous sections in the form of a recommended system approach. The concentration in this recommended system is on the electronics, both processor and peripheral.

This section first treats the resulting system as if it were developed today, without the infusion of a large amount of money into development of custom LSI parts. The use of LSI is then treated, pointing out the changes (improvements) that might be made. As previously stated, the limitations to the reduction in system electronics size, weight and power are economic, not technological.

All of the conclusions of the previous sections will not be repeated here. The reader is advised to read those sections to determine the "why" of the choices made. Sections 2.0, 3.2 and 3.3 particularly point up the mission requirements and constraints, the control system needs and equipment, including modes, accuracies, etc. The other portions of Section 3 provide tradeoffs and recommendations on a variety of pertinent subjects.

#### 4.1 Current Technology

Current technology here refers primarily to the integrated circuits used in the electronics. In both this and the succeeding section, all other parts of devices are also assumed to be those already developed and available, (i.e. gyros, reaction wheels, sensors, etc.)

The circuitry considered in this category are those parts available now, that can be bought in quantity, made under high-quality programs, and that have been/or could be) qualified for spacecraft use. This includes existing (non-custom) LSI parts.

This section will start from the baseline COPE processor and show those changes necessary (or recommended) as a result of the study, to produce a good, viable fault-tolerant control system design for long-life interplanetary missions.

The concentration will be on the processor, with a consideration of the peripherals (sensors, actuators and supporting electronics) only from the standpoints of:

- o their effect on the processor (I/O, speed, etc.)
- o their effect on the system reliability
- o their effect on the system requirements (accuracy, etc.)

The peripheral size, weight, power, etc. will not be discussed. On the other hand, all of these characteristics of the processor will be summarized.

#### 4.1.1 Performance Requirements

Some of the requirements developed in previous sections are again summarized below.

The control implementation and processing requirements are assumed as discussed in Section 3.2.1. This results in the functional requirements of Section 3.3.1. Important conclusions of this section and Section 3.3.2 are that the processor needs:

- o A moderate speed of 100 - 150 kops (>64 kops required)
- o An instruction set similar to COPE with double precision operations.
- o 16 bit accuracy
- o Memory requirements of
  - Program Memory (not including executive) > 2255 words
  - Scratch pad memory > 452 words
- o Short instruction time < 8  $\mu$ sec.

The input/output requirements of the processor were developed in Section 3.3.3 and are:

- o Analog Inputs (12 bit A/D) - 14
- o Serial Digital Inputs - 34
- o Serial Digital Outputs - 18

- Bi-level Inputs - 8
  - Bi-level Outputs - 20
  - Command - Single, 32 bit buffer
  - Telemetry - Dual, 256 bit buffers
- } also see Section 3.4.5.3

Section 3.4 discussed the tradeoffs in the processor (hardware) design. Some of the conclusions of that section were:

- 4-bit byte-organized arithmetic is optimum
- The 16-bit word length of COPE is adequate.
- Fixed-point, fractional two's complement arithmetic is satisfactory.
- The COPE instruction set is adequate and quasi-optimum (see 3.4.2)
- The only interrupts needed are internal (i.e. cycle timing and fault interrupts)
- Discrete event timing can be accomplished using the cycle timing interrupts. (No IOU timers needed.)
- No IOU discrete inputs need to be provided.
- Analog IOU outputs are unnecessary.
- Serial I/O interfaces should be accomplished as described in Section 3.4.5.2, with the (peripheral) circuitry of Figure 3-38, 3-39 or 3-40.
- The cross-strapping should be done by:
  - Bussing for all processor internal paths, using the circuit of Figure 3-16, with power gating.
  - Bussing for clock distribution and serial I/O lines, using the circuit of Figure 3-15, without power gating.
  - Conventional means, using the circuit of Figure 3-14 for all peripherals, commands, telemetry, etc.
  - Duplication (Figure 3-37) for all other I/O.
  - No analog cross-strapping is needed.

- A need for three memory types was defined:
  - ROM for executive program storage
  - RAM for read/write storage
  - PWM for protected, variable program read/write storage.
- Solid-state memory using existing LSI is much preferable to other memory types.
- Static MOS or CMOS memory is to be preferred for low power consumption for RAM or PWM.
- Addressing, read and write of memories should be parallel.
- RAM and PWM should be semi-or non-volatile.
- The ROM, ACU, IOU, RCU, HCU, etc. are currently best accomplished using bipolar TTL parts.
- The topological architecture of the COPE ACU is adequate, but might be improved by:
  - making the registers more general
  - adding registers
  - adding microprogramming

Section 3.5 discussed the tradeoffs in the processor software design, particularly from the standpoint of organization for fault tolerance. It was concluded that a modular program structure with an executive was essential, with sequential program flow. (This was also discussed in Section 3.7). Each of the applications modules is essentially independent of the others, easing development, testing, and augmenting fault detection and roll back processes.

The executive should be organized synchronously with an asynchronous overlay as discussed in Section 3.5.3.4. A representative executive structure was described in Section 3.5.4 and sized as requiring 593 words in ROM, 133 words in RAM, 115 words in PWM (in Section 3.5.6).

Software fault detection techniques were suggested in Section 3.5.5. Section 3.6.2.1 covered hardware fault detection (in the processor). The former are sized within the words (and speed) requirements listed. The latter are seen to be unnecessary, even to the limited extent provided in COPE. Section 3.6.2.2 covered hardware self-test (BITE) in the peripherals. This was also seen to be virtually unnecessary.

Section 3.6.1 discussed the use of error detection coding in both data and instruction words. Although use of a check byte is attractive, it is not necessary and so is not recommended. Retention of the parity bit in RAM (and its use in PWM) is probably advisable. The parity bit should also be added to ROM. Parity checking should be moved to the ACU, with the parity bit brought across the data bus. This should aid instruction error detection.

The redundancy management should be accomplished by using the bootstrapped distributed hardcore system discussed in Sections 3.1.4 and 3.7.5 and shown in Figures 3-22 and 3-73. This results in significant hardware savings, as discussed (together with the new RCU and HCU requirements) in Section 3.7.6. Figures 3-67 and 3-68 show the new RCU and HCU block diagrams.

The redundancy management would result in control as follows:

- Centralized control (in a power switching electronics ala Figure 3-7) for all peripherals.
- Control through the bilevel outputs of the IOU for RAM, PWM and PDB.
- Control from the RCU for ACU, IOU, ROM and DBS.
- Control of the RCU's from the HCU.

All processor elements would use the (decentralized) power control circuit of Figure 3-6. All switching would be done using latching relays.

The requirements can now be expressed in terms of the equipment.

#### 4.1.2 Processor

The processor requirements summarized in Section 4.1.1 can now be translated into hardware requirements. Using COPE as a baseline, these requirements can be translated into design changes for each element type, as listed below. The aggregate of these changes gives birth to a new, improved processor, called COPE -2. The changes are:

- ACU    - Add parity checking in instruction decoder.
  - Go to full parallel memory read-write logic & bus.
  - Delete detection of illegal micro-timing states, instructions.
  - Divide ACU into halves (AU & CU) for redundancy
- IOU    - Delete all (28) discrete inputs
  - Delete the program cycle timer
  - Change the I/O requirements as follows
    - Analog inputs, from 32 to 16 (needs 14)
    - Serial input word gates, from 32 to 40 (needs 34)
    - Serial output word gates, from 16 to 24 (needs 18)
    - Bilevel inputs, from 64 to 16 (needs 8)
    - Bilevel outputs, from 80 to 32 (needs 18, plus 8 for primary processor power control)
  - Divide IOU into halves for redundancy.
- ROM    - Add parity bit generator (increases to 17 bit word)
  - Modularize ROM as:
    - ROM-1, 1024 words for executive
    - ROM-2, 2048 words for AM's, etc.
  - Use 512 x 4 chips for ROM-2, 256 x 4 for ROM-1.
- RAM    - Use CMOS design (as developed for DPA-2)
  - Add parallel read/write bus interfacing
  - Add semi-volatility circuitry
  - Modularize RAM as one 1024 word module
  - Use 1024 x 1 chips

- PWM - Modify the above RAM design by adding write protection as discussed in Section 3.4.7.2.
  - Only one 512 word PWM required, but use 1024 words to utilize 1024 x 1 chips (same as RAM).
- RCU - Use new RCU requirements as presented in Section 3.7.6 and Figure 3-67.
- HCU - Use the HCU requirements as presented in Section 3.7.6 and Figure 3-68.
- Software - Redesign the software to provide the proper executive and modular organization, incorporating the suggested fault detection features.

These changes result in processor elements having the characteristics shown in Table 4-1: (All parameters are per element (module).)

In this table, the assumption is made of the following failure rates: ANA (analog IC), 50 bits; LSI, 40 bits; MSI, 15 bits; SSI, 10 bits; and DIS (discretes), 2 bits average. The volumes are based on a modular packaged configuration of 6 x 8 inches by a module (quantized) thickness based on the number of boards required. Multi-layer boards have been assumed (and flat-paks for the IC's).

There is a choice for the ACU and IOU as to whether they should not be split in half (and use dual standby redundancy (1/3) ) or whether they should be split in half (with approximately a 15% penalty in parts), and use single standby redundancy. A comparison of the overall processor parameters for both configurations is given in Table 4-2. The "unsplit" case is worse on volume and weight, but better on power, part count, reliability, and cost. As always, the choice depends on what is important to the chooser. (All other redundancy is 1/2, except for the HCU, which is TMR.)

TABLE 4-1  
COPE -2 Characteristics of Elements

Element	Volume (in <sup>3</sup> )	Weight (lb)	Power (w)	Parts Count						$\lambda$ (bits)
				ANA	LSI	MSI	SSI	DIS	TOTAL	
<u>ACU</u>	77	1.6	6.4	-	-	70	168	13	251	2756
or AU	38	0.9	4.0	-	-	39	99	10	148	1595
and CU	38	0.9	3.3	-	-	37	92	10	139	1495
<u>IOU</u>	48	1.0	5.5	11	2	39	89	154	295	2413
or IOU-1	38	0.7	3.2	6	-	22	53	30	111	1055
and IOU-2	38	0.7	3.2	5	2	23	51	130	211	1280
<u>ROM 1(1K)</u>	38	0.8	0.5	-	16	1	21	40	78	945
<u>ROM 2(2K)</u>	38	0.8	0.6	-	16	1	23	45	85	975
<u>RAM (1K)</u>	38	0.8	1.0	-	17	3	33	40	93	1135
<u>PWM(1K)</u>	38	0.9	1.2	-	17	9	37	45	108	1275
<u>RCU</u>	38	0.7	2.5	1	-	8	27	10	46	460
<u>HCU</u>	38	0.7	1.0	-	-	6	29	-	35	120*

\* - per TMR section

TABLE 4-2  
COPE-2 OVERALL PARAMETERS

<u>Configuration</u>	<u>Volume (in<sup>3</sup>)</u>	<u>Weight (lb.)</u>	<u>Power(W)</u>	<u>Total Parts</u>	<u>Reliability</u>	
					<u>7.5 years</u>	<u>10 years</u>
ACU, IOU unsplit 1/3 redundancy	793	16.5	18.7	1947	0.977	0.959
ACU, IOU split 1/2 redundancy	722	15.1	20.5	2073	0.958	0.928

As a comparison, the existing COPE design (with 1/3 unsplit ACU and IOU and comparable memory size would have the following parameters:

o	Volume	-	863 in <sup>3</sup>		
o	Weight	-	14.6 lb.		
o	Power	-	34.3 W		
o	Parts	-	2916		
o	Reliability	7.5 years	-	0.968	
		10 years	-	0.941	

It should be noted that COPE is not fault tolerant or autonomous to the same degree as the projected COPE -2.

#### 4.1.3 Peripherals

The peripherals are relatively little affected by the fault-tolerance requirements, but they must be designed for a small enough element size to meet the reliability requirements, with the proper interfacing to/from the IOU, and with fail-safe features designed into the electronics of the RWA and thruster valve drivers. Except for the latter requirement, this should little change the design of any existing equipment.

The inclusion of back-up electronics is optional and has not been considered in the reliability calculations here. It is considered that the FSS, STA and IRU are functionally redundant and this has been considered in the reliability calculations.

The following failure rate assumptions have been made:

- o WASS - Four portions, each 1 of 2 redundant, each at 200 bits.
- o Accelerometers - 1 of 2 redundant, at 1,000 bits.
- o Other Sensors - Functionally redundant to each other.
  - Gyro - 7,000 bits per axis, 3 of 4 redundant
  - FSS - 2,000 bits, 1 of 2 redundant
  - STA - 3,600 bits, 1 of 2 redundant

- RWA - 1,000 bits per axis, 3 of 4 redundant
- Actuators (scan and TVC) - 1,000 bits per axis, 4 axes, each 1 of 2 redundant
- Valve Drive Electronics - 1,000 bits, 1 of 2 redundant  
(This is very conservative, since the valves driven are also redundant.)
- Power Switching Electronics - 500 bits, 1 of 2 redundant.

Using these assumptions, the reliabilities of the peripherals is 0.976 for 7.5 years and 0.957 for 10 years. Combining this with the processor reliabilities (the unsplit ACU/IOU case is assumed) gives overall control system reliabilities of 0.954 for 7.5 years and 0.918 for 10 years. Further redundancy could improve these figures at the expense of extra volume, weight and cost. Note that (for this example) the processor and peripherals have approximately equal reliability.

#### 4.2 Advanced Technology

Again, to repeat what was said earlier, what can be accomplished by the incorporation of custom LSI is limited by the money one wants to spend. The benefits of reduced size, weight, power, parts count, etc. (and improved reliability) are achieved at an increasing cost to develop the necessary new parts. The benefits are almost proportional to the additional dollars spent.

There is nothing apparent at this time that would drastically affect the nature of the organization of the system by the incorporation of generally applied LSI. The  $\lambda$ 's of the elements would come down, improving the system reliability (or alternatively, the number of elements could be reduced, with the same result).

The arithmetic structure, byte-size, memory requirements, basic architecture, program modularity, executive organization, etc. would probably not change. The same interfaces would exist between elements, whether in the processor or outside it.

The peripherals would be affected less than the processor. LSI circuits could be developed for the receivers/senders for the serial IOU interface. Power limitations would probably prevent major changes/improvements in the actuator electronics. Certainly the sensor electronics might be improved with LSI.

An ordering of the incorporation of further LSI into the system design (in order of approximate improvement vs. cost ratio) might be useful. This is very subjective, but is approximately correct. The LSI improvements (from the COPE-2 processor configuration described in the preceeding section, are:

- Use of PLA's or CGA's to reduce the logic in the ACU (and incorporate microprogramming). Approximately 110 IC's could be replaced by 10 to 20 LSI parts, also saving about 1 watt.
- Development of an LSI part for power-gated bus sending and ceiving.
- Development of an LSI part for peripheral sending/receiving registers.
- Incorporation of those portions of the ROM, RAM and PWM that are not now LSI into LSI chips.
- Development of micro-processor-type LSI chips for the ACU. Experience (such as on MPP) indicates this would require 10-12 chips of about 5-6 designs.
- Development of IOU LSI chips, which could drastically cut this part count. Note that this can only work to a point, however since the lead restrictions of the LSI are not compatible with the large number of inputs and outputs of an IOU.

- Development of an RCU set of LSI chips. The clock might be one and the remainder of the RCU 3-4 more.
- Development of a single-chip TMR LSI HCU.

Now if all these new LSI chips were developed (30 to 40 of them), we would have a processor (call it COPE- $\infty$ ) that might have the following parameters:

- Volume - 200 in<sup>3</sup>
- Weight - 6 lb
- Power - 7 w
- Parts Count - 200
- Reliability - 0.98 for 10 years

The question remains - is it worth it? The authors believe that in the limited-funding aerospace environment of today the answer will be no.

What about the possibility of "someone else" paying for the development of parts that can be used?

Commercial LSI part development that can be used in a fault tolerant control system are not too likely, beyond the memories already developed. The requirements that "we" need are too special and are not needed in the commercial applications.

The military offers better hope, particularly with regard to micro-processor LSI. Although such parts do not now offer the necessary fault-tolerant/redundancy, etc. features, it is hoped that they will eventually, since the military needs are quite similar, even though their spacecraft are earth-orbiting.

One last possibility should be mentioned. Someday a single-chip microprocessor LSI part of extremely high speed (allowing serial input/output/addressing, etc.) will be developed. If such a part were available, the basic structure of the system, as discussed in this report, could change. The most likely changes would involve use of many of these microprocessor chips, distributed throughout the system in a multiprocessor-like arrangement, operating mostly independently of each other, but redundant to each other. Such a development could effect a quantum improvement in system parameters, without a corresponding cost increase.

The trend is already away from the use of a single computer on board a spacecraft to handle all systems to the use of optimized computers in each system, asynchronously tied into each other (as was a ground-rule of this study). This evolution will undoubtedly continue until eventually each system uses several micro-computers in an analogous manner. The future is always hard to predict and we must act now with what we have available now.

## 5.0 SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

This section of the report views the study in perspective, providing a brief summary of the conclusions reached and recommendations for future studies.

### 5.1 Summary and Conclusions

The subject is a very dynamic one and progress is being made very rapidly in both the hardware and non-hardware arenas. For the hardware, the direction is fairly predictable; that of providing more and better LSI parts, increasing the capability of each chip, and reducing size, weight and power, while increasing reliability.

For the non-hardware activities, the possibilities are not so predictable. Non-hardware here implies not just the software, but particularly the system design - the architecture and organization of the system.

In the modern design of computers, or any other complex system, the distinctions between the system, the hardware, and the software are becoming ever more diffuse. If the system is to be fault tolerant, then the overlap is almost total.

It is of the utmost importance that such systems be designed as systems, from the beginning. The requirements of fault tolerance must be considered from the very start. The retrofitting of fault tolerance in an existing design is very difficult, if not impossible.

Yet, from the standpoint of spacecraft attitude control systems, we have seen that there is much in the "old" systems that does survive, almost unchanged. The sensors, actuators and their supporting "peripheral" electronics can generally remain the same if certain rules are observed:

- The size of the redundancy elements must be suitably small (see Section 3.1.1)
- The redundancy blocks must be designed for independent failures.
- The redundancy blocks should be controllable by the processor.
- The interfaces should preferably be serial digital with the processor-I/O (but do not have to be).
- The actuators (and their electronics) must be designed for fail-safe operation (see Sections 2.2.5 and 3.2.4).

- The interfaces should be asynchronous.
- Fast servo loop closures should be performed in the peripheral electronics, not in the processor.
- No particular peripheral BITE needs to be added.
- The use of a back-up electronics mode/equipment is optional.

One cannot, however, just take an existing computer and make it do the job. Neither in its redundancy organization, nor in its program organization is it suited for the task. Existing computer/processors are characterized by a monolithic reliability structure not suited to long life and by a hardware/software organization not suited to fault tolerance.

The processor, therefore, is the key element. Whether one starts from "scratch" or uses existing "good tries", such as STAR or COPE, redesign and optimization will be necessary. Section 4 provided some alternatives, based on the TRW COPE processor.

It was seen that the necessary reliability criteria could be met by the system. That is, reliabilities  $> 0.9$  could be achieved for 10 years for reasonable size, weight and power. This can be done now, with existing "off-the-shelf" parts, components, sensors and actuators, etc. The infusion of more money could create new LSI parts, saving size, weight, power and improving the reliability.

The most important part of the study and the most important conclusions are in Section 3.7. Some of those conclusions are:

- General fault tolerance can be achieved relatively economically.
- Recovery times of  $< 1$  second can be obtained.
- The system must be structured so that the number of faulty behavior patterns is limited.
- Adjoined processes (such as timing) are the best indicators of faulty operation of a primary process.
- The testing for fault correction must be exhaustive, but this can only be accomplished if the number of fault patterns is small and all faults can be proved to result in one or more of these fault patterns.

- The feasibility of validation must be a key element of the design process.
- The system must be structured as an aggregate of processes, each independent from the others and all sequential in operation.
- To minimize recover time, rollback should occur only to the last successful program portion.

Other important conclusions relate to the redundancy structure of the processor and the heirarchy of control of the failure correction process.

Summarizing:

- Failure correction by switching of standby, redundant elements is most economical.
- Passive redundancy is not needed except in the hardcore portion of the processor.
- The hardcore portion should be reduced to the minimum possible. It should control other parts (which use standby redundancy), which control other parts, etc. This provides the greatest simplicity, lowest power, etc.

In summary of the entire study, what was required by the statment of work (see Section 1.3), has been done.

- Using an existing programmable digital processor (COPE), it has been applied to the control system electronics for deep space missions.
- The criteria for hardware optimization have been considered and analyzed with respect to all pertinent parameters.
- The use of LSI in the design has been extensively studied.
- The architecture, software and system design has been treated in detail.

The authors believe that this report is the first comprehensive document on the subject, treating all aspects of the problems in a coordinated manner. To others, or to ourselves from the viewpoint of a later perspective, this coordination may seem less than it should have been and some of our work may seem naive. None-the-less, it is a milestone on a complex and rocky path.

We would welcome the opportunity to discuss our conclusions and our trades with any interested parties.

## 5.2 Recommendations for Further Studies

In closing, it is appropriate to suggest paths that might be productively followed in future studies. Some possibilities for studies are:

- Fault Tolerant Design - The recommended design (see Section 3.7.7) is based on the use of secondary failure detection processes at various structural and functional levels. Recovery management is distributed between the HCU, the RCU's and the IOU's, with common paths through the software. The degree of fault tolerance attainable with a multi-structured configuration like this is not evident unless satisfactory proofs that all failure modes of the primary system (real or virtual) map into subsets of failure patterns of the associated secondary processes. Since secondary processes introduce additional failure modes (not corresponding to any primary ones), the recovery process must be shown to be exempt from singular cases (where recovery fails) and cyclic loops.

Systematic procedures are required to provide visibility and conclusive verification of fault tolerant properties in a multi-process structure. Graph-theoretic approaches based on connectivity considerations have been successfully used for similar purposes. Extension of the work of Ramamoorthy to attack the problems discussed above should be further explored. It is believed that proofs may be found that proper structuring will yield systems that are conclusively fault-tolerant.

- Software Design Methodology - There are several aspects of the structured software design problem which need to be defined in greater detail. General criteria for partitioning the software into non-interacting processes must be adapted and extended to the particular cases of the attitude control applications programs and overlaid isosequential executives. The methodology developed by Dijkstra and his followers is based on objectives which are different from those of the present application. Component processes in a fault tolerant environment must meet other requirements in addition to

those which have been established to enhance testability. For instance, desirable qualities are diagnosability and fault tolerance at the interfaces.

The software verification process usually involves a significant amount of duplicated effort since testing personnel must retrace the steps followed by the programmers in order to understand the program operation and be able to design relevant test cases. In a well-structured software organization, test procedures are an integral part of the design criteria and facilities are provided initially to facilitate and expedite testing.

- Software Design and System Testing - There may be machine idiosyncracies, timing problems and software incompatibilities which are difficult to identify unless a real machine is operated in real time and with realistic test cases. Also, design ground rules and/or criteria may need to be revised to accommodate implementation/programming constraints which are not obvious until detailed execution phases are completed.

In addition to validating the design approach recommended in the study, a test program with actual hardware can provide the following valuable information:

- Frequency of usage of the various machine instructions. This provides a measure of processor performance and an indication of the adequacy of the current instruction repertoire.
- Diagnostic overhead. Relative qualities of alternative diagnostic approaches can be compared with realistic sets of simulated fault patterns.
- Recovery times. Recovery mechanisms can be exercised at system, sub-system and unit levels. Effects of switching times of hardware elements can be realistically assessed.
- Interactions with testing routines. Testing and debugging routines included in the system software can be exercised and effects on normal system operations can be determined.
- Input/output signal flow. The input/output interfaces of the processor can be evaluated under realistic simulated conditions. Fail-safe circuits and mechanisms can be exhaustively tested.

- o Failure Modes, Effects and Criticality Analysis. Such an analysis can proceed from the hardware or from the software aspects, or preferably, both. An actual processor should be analyzed in detail to determine the effect of each class of failures. This should be conducted first on the lowest level of hardware (the parts) and then extended to the structure of the processor (by the use of computer aided emulation) as discussed earlier in this section.
- o Hardware Development. Relative to COPE, the design changes suggested in this report should be completed to the breadboard stage, to prove their feasibility. This includes development of the new RCU and HCU and operating them with the existing COPE breadboard.
- o LSI Studies. Some or all of the further possible LSI applications (see Section 4.2) could be further evaluated for feasibility, cost, etc. Some of these might then be further developed to the point of building parts and incorporating them into existing breadboards.
- o The Future. More thinking might be done on the application of and requirements for true, low-chip-count microprocessors. This would include the possibilities of distributing such elements throughout the system and studying the problems/opportunities this affords relative to the architecture, redundancy, fault tolerance, etc. of the system.

## BIBLIOGRAPHY

- [1] A. Hopkins, A. Green, W. Weinstein, et al, "A Fault-Tolerant Information Processing System for Advanced Control, Guidance, and Navigation", MIT Charles Stark Draper Lab., Rept. No. R-659, May 1970.
- [2] A. Avizienis, "Fault-Tolerant Computing: An Overview", Computer, Jan/Feb 1971.
- [3] P. G. Neumann, J. Goldberg, K. N. Levitt, and J. H. Wesley, "A Study of Fault-Tolerant Computing: Final Report", Stanford Res. Inst., 31 July 1973, NTIS Access No. AD-766974.
- [4] E. W. Dijkstra, "The Structure of the 'THE' Multiprogramming System", Comm. of the ACM, v. 11, n. 5, May 1968, pp.341-346.
- [5] W. C. Carter, "Experiments in Proving Design Correctness for Microprogram Controlled Computers", Digest of Papers of the 4th Annual Symposium on Fault-Tolerant Computing, Champaign, Illinois, June 19-21, 1974, pp. 5-22 to 5-27.
- [6] M. R. Paige, "Software Testing: An Overview", Digest of Papers of the 4th Ann. Symposium on Fault-Tolerant Comp., Champaign, Illinois, June 19-21, 1974, pp 5-18 to 5-21.
- [7] J. A. Rohr, "Starex Self-Repair Routines - Software Recovery in the JPL Star Computer", Digest of Papers of the 2nd Annual Symposium on Fault-Tolerant Computing.
- [8] K. M. Chaudy and C. V. Ramamoorthy, "Rollback and Recovery Strategies for Computer Programs", IEEE Trans. on Computers, v. C-21, n. 6, June 1972.
- [9] R. E. Forbes, D. H. Rutherford, C. B. Stieglitz, and L. H. Tung, "A Self-Diagnosable Computer", Proceedings of the 1965 Fall Joint Computer Conference, pp. 1073 - 86.
- [10] F. P. Preparata, G. Metze, and R. T. Chien, "On the Correction Assignment Problem of Diagnosable Systems", IEEE Trans. on Elect. Comp., v. EC-16, n.6, Dec. 1967, pp. 848-54.



ONE SPACE PARK • REDONDO BEACH, CALIFORNIA

CODE IDENT 11892

TITLE	
SPECIFICATION FOR A DIGITAL PROCESSOR ASSEMBLY (COPE)	
DATE	NO.

SUPERSEDING: \_\_\_\_\_  
\_\_\_\_\_

PREPARED BY: A. A. Sorensen

APPROVAL SIGNATURES:

\_\_\_\_\_ DAYE

\_\_\_\_\_ DAYE

\_\_\_\_\_ DAYE

\_\_\_\_\_ DAYE

\_\_\_\_\_ DAYE

\_\_\_\_\_ DAYE

## TABLE OF CONTENTS

- 1.0 GENERAL
- 2.0 SYSTEM ORGANIZATION
- 3.0 ARITHMETIC AND CONTROL UNIT (ACU)
  - 3.1 Data Word Format
  - 3.2 Instruction Execution Times
  - 3.3 Description of Instructions & Instruction Formats
    - 3.3.1 Memory Paging
    - 3.3.2 Indexing
    - 3.3.3 ACU Initialization Entry Point
    - 3.3.4 Instructions
    - 3.3.5 Double Precision Operations
    - 3.3.6 Multiply Algorithm
    - 3.3.7 Divide Algorithm
  - 3.4 ACU Internal Organization
  - 3.5 ACU/I-O Operations and Interface
  - 3.6 ACU BITE
- 4.0 DPA MEMORY SYSTEM
  - 4.1 Read Only Memory Units (ROM)
  - 4.2 Random Access Memory Units (RAM)
  - 4.3 Alterable Program Memory Units (APM)
  - 4.4 Plated Wire Memory Units (PWM)
- 5.0 INPUT/OUTPUT UNIT (IOU)
  - 5.1 Functional Requirements
    - 5.1.1 Control
    - 5.1.2 A/D Converter
    - 5.1.3 Serial Data
    - 5.1.4 Telemetry
    - 5.1.5 Commands
    - 5.1.6 Program Cycle Timing
    - 5.1.7 Discrete Inputs
    - 5.1.8 Bilevel Inputs
    - 5.1.9 Bilevel Outputs

## 5.2 Peripheral Interface Definition

- 5.2.1 Analog Inputs
- 5.2.2 Discrete Inputs
- 5.2.3 Input Bilevels
- 5.2.4 Serial Input Data
- 5.2.5 Serial Output Data
- 5.2.6 Commands
- 5.2.7 Output Bilevels
- 5.2.8 Telemetry

## 6.0 RECONFIGURATION CONTROL UNIT (RCU)

### 6.1 Internal Organization

### 6.2 Functions

### 6.3 Functional Requirements

- 6.3.1 Reconfiguration
- 6.3.2 Clock Generation
- 6.3.3 RCU Communication

### 6.4 Tester Interfaces

## 7.0 PACKAGING

## 8.0 POWER

## 1.0 GENERAL

The Digital Processor Assembly (DPA) is a small, general-purpose spacecraft flight control computer of moderate speed and memory capabilities. It is designed to be modular in terms of memory size, input-output channels, packaging, and reliability (achieved by variation of the amount of redundant standby modules). It also is absolutely fault-tolerant insofar that any single failures will only momentarily suppress its capability, until it performs a self-reconfiguration.

A summary of the DPA capabilities and features is shown in Table 1.1.

## 2.0 SYSTEM ORGANIZATION

The DPA is subdivided into modules or units. Each module is a separate sub-assembly or package (see Section 7.0). Each module is fully cross-strapped with all other modules so that it functions as a redundant element of the DPA.

The five main types of units of the DPA are shown in Figure 2.1. These units (together with other units, not shown in the Figure), are described briefly below and are specified more fully in Sections 3.0 through 6.0.

- o Arithmetic and Control Unit (ACU) This unit provides the central arithmetic processing, logic processing and machine control for the DPA. It communicates with all other units via the data bus. The ACU provides memory and input/output addressing; cycle, program and micro-timing; instruction decoding; and the necessary computational registers, counters and logic.

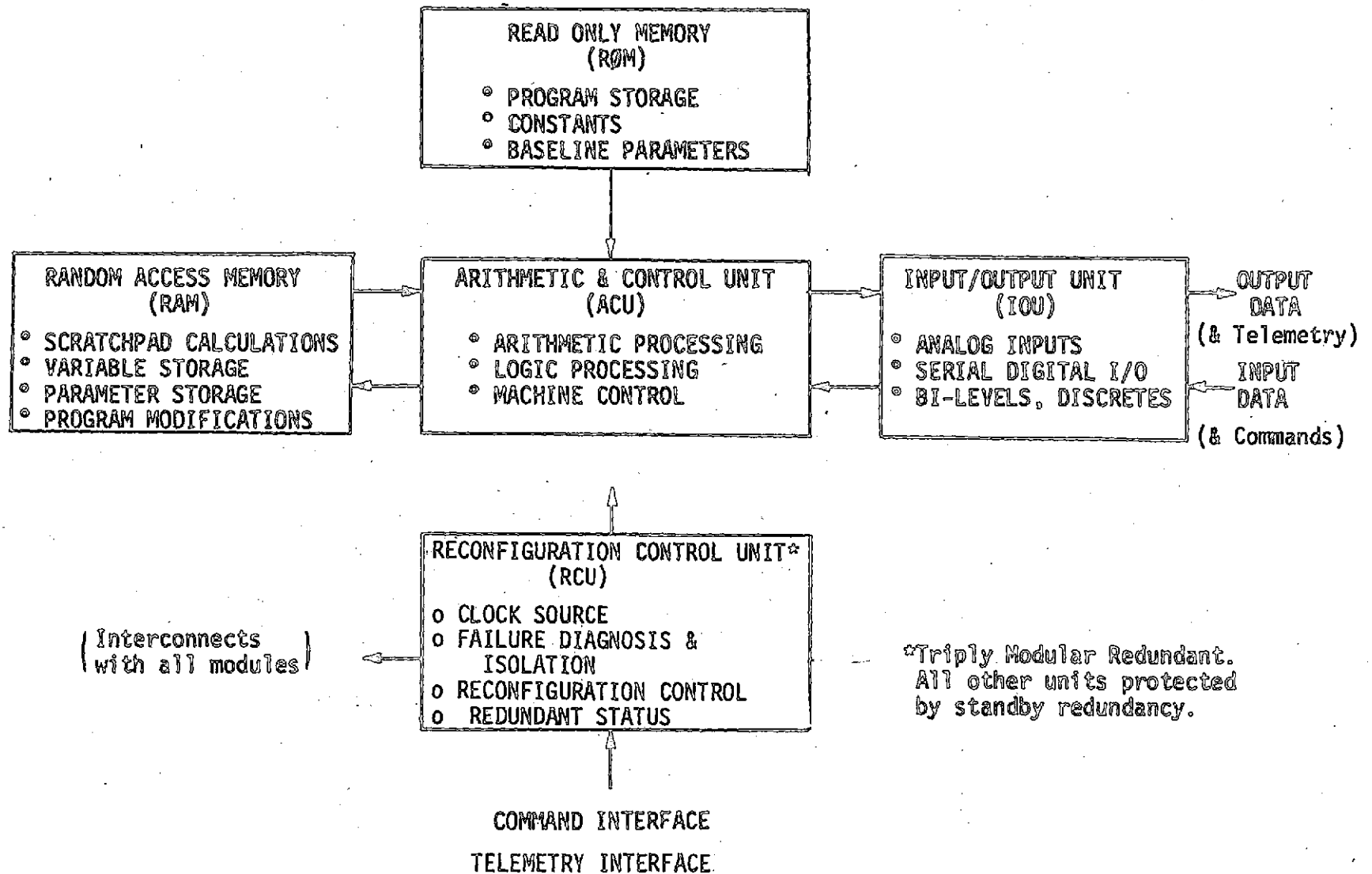
Table 1.1

DPA CAPABILITIES AND FEATURES SUMMARY

- o 4 bit Byte-organized arithmetic
- o Fixed point, two's complement, fractional words
- o 16 bit data and instruction word length
- o 38 basic instructions, including multiply, divide and double precision add and subtract
- o 120 k ops speed (typical short instruction time of 7.2  $\mu$ sec)
- o Up to 64k of memory addressing capability
- o RAM memories in 256 and 512 word modules
- o ROM memories in 1024 and 2048 word modules
- o APM memory available to replace ROM for ground test
- o Compatible with plated-wire memories
- o No unconditional interrupts
- o Conditional interrupts available as programmed
- o Fully buffered, asynchronous I/O interfaces
- o Self-contained redundant clock
- o Maximum I/O capability of (in modular steps of 8):
  - 32 serial input channels
  - 16 serial output channels
  - 64 bilevel inputs
  - 80 bilevel outputs
  - 28 discrete inputs
  - 32 analog input channels (12 bit A/D conversion)
  - 32 bit (or less) command word
  - Variable format telemetry interface
- o Reliability selectable by number of redundant spares included
- o Self-contained failure detection
- o Automatic or manual reconfiguration
- o Automatic program restart upon power turn-on or reconfiguration
- o Self-contained internal and external program timing
- o Modular packaging

Figure 2.1

DPA MODULAR ELEMENTS



- o Input/Output Unit (IOU) This unit provides the primary interface between the DPA and the external world. It accepts serial, bilevel, discrete, and analog inputs and commands and produces serial and bilevel outputs and telemetry. All input and output interfaces with the exterior are asynchronous and fully buffered. The IOU is designed to be internally modular so that it may be optimized to its requirements for minimum cost, power and part count.
- o Read Only Memory (ROM) This unit provides the program memory and storage of constants and baseline parameters. It is a solid-state, programmable (at DPA assembly) memory using power gating for minimum power consumption. Interchangeable module sizes of 1024 and 2048 words may be used, with a total sizing of up to 32k words.
- o Random Access Memory (RAM) This unit provides the scratch pad memory for variable storage, storage of intermediate results of operations, parameter storage and program modifications. It is a solid-state memory and includes parity bit generation and checking. Interchangeable module sizes of 256 and 512 words may be used, with a total sizing of up to 32k words.
- o Reconfiguration Control Unit (RCU) This unit provides the clock generation for the DPA and also provides for the reconfiguration of the (redundancy of the) other modules of the DPA, should faults occur. It confirms the faults and reconfigures the powered and standby modules of each type, through all combinations, until a viable configuration is found. The RCU is an optional unit and is not used when automatic reconfiguration is unnecessary. In that case, the clock must be supplied external to the DPA.
- o Data Bus (DBS) This is not a physical unit, but is a redundant element of the DPA. It includes the lines and cross-strapping which performs the inter-communication between all units.

- o Alterable Program Memory (APM) This memory element is a direct functional, electrical and physical replacement for the ROM element in every respect except that the APM is able to be programmed by electrical signal inputs and is a volatile memory. The APM is used to replace the ROM for ground testing and situations where program changes are frequent. It is not used in flight.
- o Plated Wire Memory (PWM) This memory element can be used to replace either the ROM or RAM elements to provide low-power non-volatile memory capability.

Each module contains its own power control circuitry and may be turned on or off by bilevel signals. Since all modules are passively cross-strapped, this power control controls the redundancy. All electrical power is generated and regulated external to the DPA. The DC voltages required are +5V, +15V, -15V, and +28V. Power consumption depends on the numbers of modules used.

### 3.0 ARITHMETIC AND CONTROL UNIT (ACU)

The arithmetic and control unit (ACU) contains the central control and arithmetic processing for the DPA. The ACU addresses all memory and inputs and outputs, executes all instructions while controlling the DPA timing. All communication with the memories and IOU is via a multi-wire data bus.

This section is organized to first define the DPA word format, then describe the instructions and their functions, and then detail the ACU organization and requirements.

### 3.1 DATA WORD FORMAT

Data words shall be in binary form using two's complement fixed point fractional arithmetic.

Single precision words shall include a sign bit and 15 fractional magnitude bits as shown in Figure 3.1.

Double precision words shall have the format shown in Figure 3.2. The most significant half word, normally operated on in the accumulator, shall contain a sign bit and the 15 most significant fractional magnitude bits. The least significant half word, normally operated on in the Q-register, shall contain the 15 least significant fractional magnitude bits. Bit number 15 having an equivalent weight  $2^{-31}$  is undefined for signed sixteen bit double precision arithmetic, but is still carried in the machine as a working bit. Refer to Section 3.3.5, "Double Precision Arithmetic", for a description of how this bit is handled.

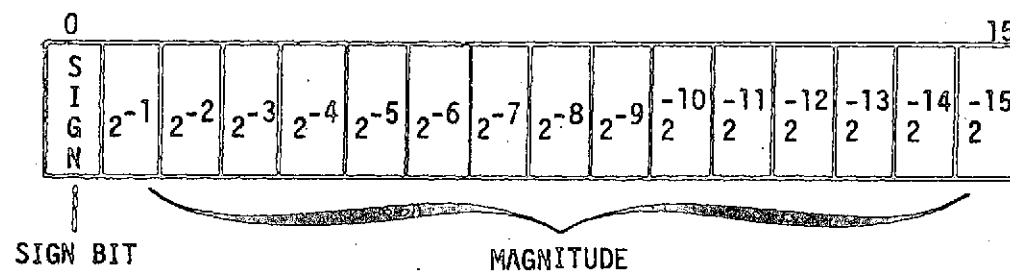
### 3.2 INSTRUCTION EXECUTION TIMES

Basic execution times for the DPA instruction family are specified in Paragraph 3.3, "Description of Instructions". The speeds given are for the condition where instructions are fetched from the solid state read-only memory (ROM) when no indexing is specified.

When instructions are fetched from the solid state random access memory (RAM), a time penalty of 3.6  $\mu$ seconds must be added to the basic instruction execution time.

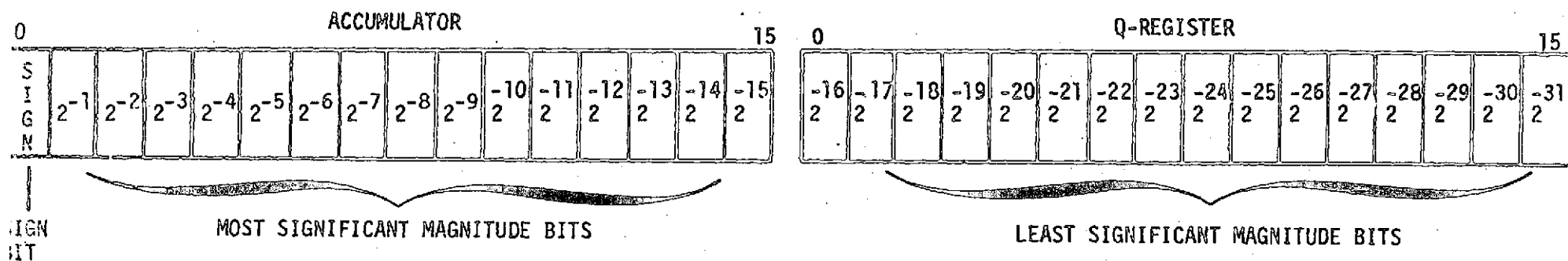
There is no time penalty experienced for instruction fetches when operating with a plated wire memory system.

Operand Fetches and Stores - There will be an additional 1.2  $\mu$ second time penalty experienced whenever an operand is fetched from the solid state read-only memory. When operating with a plated wire memory system,



### DATA WORD FORMAT, SINGLE PRECISION

FIGURE 3.1



DATA WORD FORMAT, DOUBLE PRECISION

FIGURE 3.2

a time increment of 1.2  $\mu$ seconds must be added to any instruction requiring an operand fetch or store.

Indexing - Independent of the memory system employed, a time penalty of 2.4  $\mu$ seconds is experienced whenever indexing is specified.

### 3.3 DESCRIPTION OF INSTRUCTIONS AND INSTRUCTION FORMATS

The following abbreviations are used throughout this section.

DA	=	Device Address
DISP	=	Displacement Field for Relative Jumps
EA	=	Effective (Memory) Address
EDA	=	Effective Device Address
EO $\emptyset$	=	Extended Operation Code
I	=	Index Tag Bit
IM $\emptyset$	=	Immediate Operand
IND	=	Discrete Indicator Bits
N	=	Number of Shifts
$\emptyset$	=	Operand Source Tag Bit
$\emptyset$ P	=	Operation Code
S	=	Sector Tag Bit
X	=	Operand
Y	=	Operand Memory Address Field

Register abbreviations are defined in Section 3.4, "ACU Internal Organization".

The generic instruction format for the ACU is shown in Figure 3.3. Bits 0-4 of the instruction word contain a five bit operation code defining the particular instruction to be executed. Bits 5-7 are tags having the following significance:

GENERIC INSTRUCTION FORMAT (for instructions listed in Table 3.1)

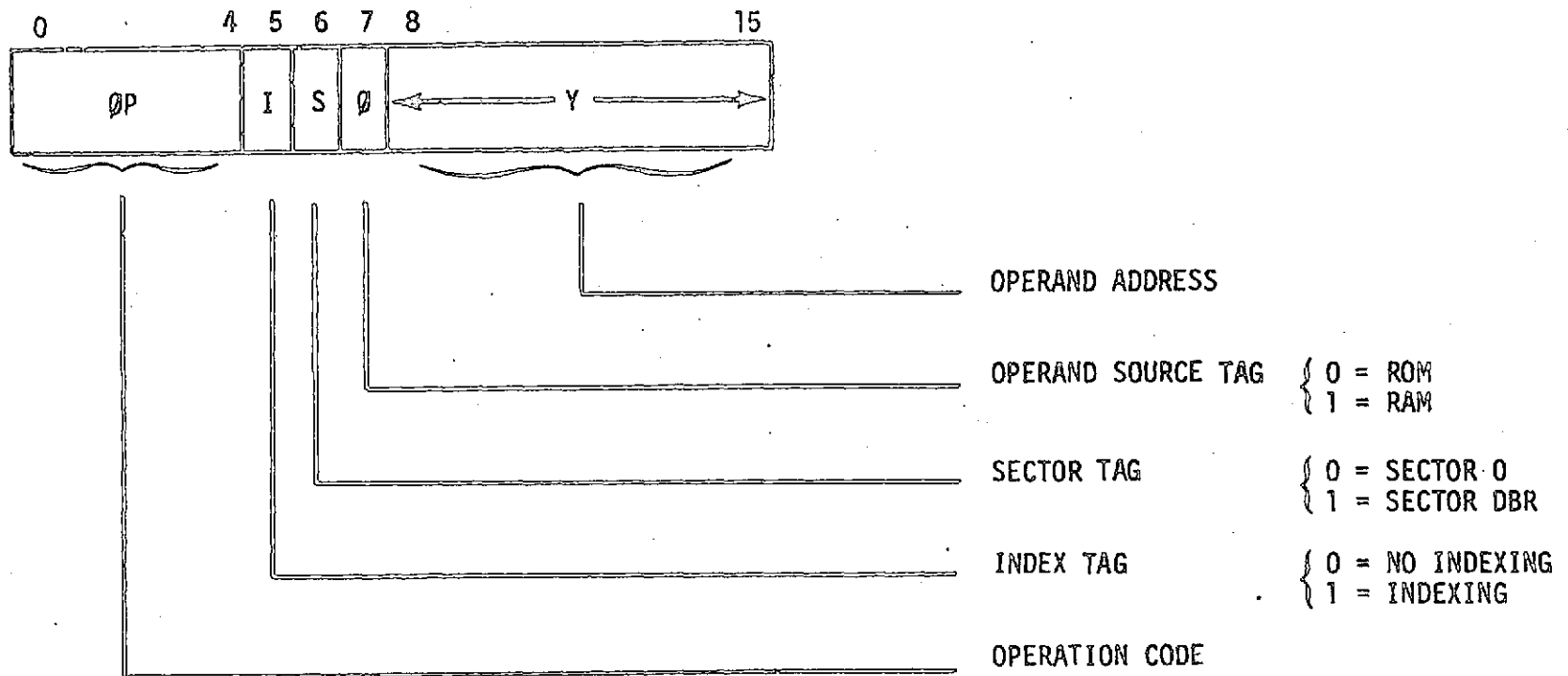


FIGURE 3.3

- I = Index Tag. This bit is set when indexing is desired.
- S = Sector Tag. This bit specifies whether or not the Data Base Register (DBR) is to be appended to the address field. A sector is defined as 256 words of memory.
- Ø = Operand Source Tag. This bit defines the memory source from where the operand will be retrieved; i.e., ROM or RAM. For instructions requiring a store operation, this bit will be set to a one by the assembler. The Ø tag is used in a similar manner for a system using a plated wire memory (refer to Section 4.4).

The effect of these tag bits are further described in Table 3.1.

Bits 8 - 15 contain the unmodified address for the operand. This field permits an operand to be accessed from any location within a given memory sector (256 addresses).

### 3.3.1 Memory Paging

Because of the limited operand address field of the instruction word, the ACU must perform a memory paging (sectorization) function to access operands throughout the memory system. To accomplish this task, the ACU contains a DBR that is used to specify the current sector from which operands will be retrieved. A memory sector contains 256 words of storage. The DBR operates in conjunction with the sector tag bit of the instruction word. If the sector tag bit is set to zero, the operand is retrieved from sector zero (i.e., locations 0 - 377<sub>8</sub>). If the sector tag bit is set to one, the DBR defines the sector from which operands will be retrieved. The DBR is seven bits wide. Therefore, appending the DBR to the eight bit address field of the instruction word results in a memory address field that is 15 bits wide. This allows addressability to a maximum of 32,768 words of memory. Note that addresses in both ROM and RAM memories have identical values running from 0 - 7777<sub>8</sub>. The operand source tag defines the memory source from which operands will be retrieved.

### 3.3.2 Indexing

When indexing is specified, the eight bit index register is added to the eight bit address field of the instruction word.

TAGS INSTRUCTION	S = 0		S = 1		$\emptyset = 0$	$\emptyset = 1$
	I = 0	I = 1	I = 0	I = 1		
LDA, LDX *, LDQ, ADD, SUB, MPY, DVD, $\emptyset$ RA, ANA, $\emptyset$ R, ADQ, SUQ	EA = Y	EA = $(XR + Y) \bmod 2^8$	EA = $Y + 2^{DBR}$	EA = $\left[ (XR + Y) \bmod 2^8 + 2^{DBR} \right]$	ROM	RAM
STA, STQ, STX *	EA = Y	EA = $(XR + Y) \bmod 2^8$	EA = $Y + 2^{DBR}$	EA = $\left[ (XR + Y) \bmod 2^8 + 2^{DBR} \right]$	NOT DEFINED	RAM

NOTES: DBR = Data Base Register

EA = Effective Address

For STA, STQ, and STX instructions, the condition where  $\emptyset = 0$  is not defined.

\*LDX and STX are non-indexable, therefore, the condition where I = 1 is undefined.

TABLE 3.1

The index addition is performed modulo 256; therefore, indexing across a sector boundary is not permitted. Instructions that are indexable are denoted by an (I) following the operation mnemonic, e.g., LDA (I) indicates that the load accumulator instruction is indexable.

### 3.3.3 ACU Initialization Entry Point

Upon initialization, the ACU executes the first instruction from ROM location 00000<sub>8</sub>.

Special Note: For many system applications, the actual requirements for random-access memory will be within 512 words of storage. Thus by setting the DBR to the first sector, all operands may be retrieved by simple manipulation of the sector bit. Because of the facility for the ACU to simply access the first two sectors of memory, it is desirable to reserve the first two sectors of the ROM for the storage of frequently used constants and jump tables. If this is the case, the programmer should immediately transfer the program to sector two via an indirect jump order.

### 3.3.4 Instructions

#### Register Operations

LDA (I)      Load Accumulator      (7.2  $\mu$ sec)

$X \rightarrow A$  ;  $PC+1 \rightarrow PC$

The accumulator is cleared and replaced by the operand X.

STA (I)      Store Accumulator      (7.2  $\mu$ sec)

$A \rightarrow EA$     $PC+1 \rightarrow PC$

The contents of the accumulator are stored in the memory cell specified by the effective address EA. The accumulator is not disturbed.

LDA (I)      Load Q-Register      (7.2  $\mu$ sec)

$X \rightarrow Q$  ;  $PC+1 \rightarrow PC$

The Q-register is cleared and replaced by the operand X.

STQ (I)      Store Q-Register      (7.2  $\mu$ sec)

$Q \rightarrow EA$  ;  $PC+1 \rightarrow PC$

The contents of the Q-register are stored in the memory cell specified by the effective address EA. The Q-register is not disturbed.

LDX      Load Index Register      (4.8  $\mu$ sec)

$X \rightarrow XR$  ;  $PC+1 \rightarrow PC$

The index register is cleared and replaced by the least significant eight bits of the operand X. This instruction is not indexable.

STX      Store Index Register      (4.8  $\mu$ sec)

$XR \rightarrow EA$  ;  $PC+1 \rightarrow PC$

The contents of the index register are stored in the eight least significant bits of the memory cell specified by the effective address EA. The index register is not disturbed. This instruction is not indexable.

#### Arithmetic Instructions

ADD (I)      Add to Accumulator      (7.2  $\mu$ sec)

$A+X \rightarrow A$  ;  $PC+1 \rightarrow PC$

The operand X is added to the contents of the accumulator and the result is placed back into the accumulator. If an overflow results, the overflow flip-flop is set.

SUB (I)      Subtract from Accumulator      (7.2  $\mu$ sec)

$A - X \rightarrow A$  ;  $PC+1 \rightarrow PC$

The two's complement of the operand X is added to the accumulator and the result is placed back into the

accumulator. If an overflow results, the overflow flip-flop is set.

ADQ \*(I)      Add to Q-Register      (7.2  $\mu$ sec)

$Q + X \rightarrow Q$  ;  $PC+1 \rightarrow PC$

The operand X is added to the contents of the Q-register and the result is placed back in Q. Overflow indication is not provided for Q-register operations.

SUQ \*(I)      Subtract from Q-Register      (7.2  $\mu$ sec)

$Q - X \rightarrow Q$  ;  $PC+1 \rightarrow PC$

The operand X is subtracted from the contents of the Q-register and the result is placed back in Q. Overflow indication is not provided for Q-register operations.

\* If an ADQ order is followed immediately by an ADD (to accumulator) order, the operations are automatically handled by the DPA to produce a double precision arithmetic sum. The same is true for SUQ followed by SUB to produce a double precision arithmetic difference.

MPY (I)      Multiply      (66  $\mu$ sec)

$A \cdot X \rightarrow AQ$  ;  $PC+1 \rightarrow PC$

The contents of the accumulator is multiplied by the operand X and the double length product is placed in AQ.

DVD (I)      Divide      (115.2  $\mu$ sec)

$AQ/X \rightarrow A$  ;  $PC+1 \rightarrow PC$

The contents of AQ are divided by the operand X and the rounded quotient is placed in the accumulator. No overflow will occur if  $-1 \leq (A,Q)/X < 1$ , i.e., if dividend/divisor are greater to or equal to minus one, but strictly less than plus one. If overflow occurs due to improper scaling, the results will be incorrect. The overflow indicator will not be set if this condition occurs. If quotient round-off causes an overflow, the overflow flip-flop is set. The Q-register is zeroed at the end of a divide operation.

CMP

Two's Complement

(7.2  $\mu$ sec)

$-A \rightarrow A$  ;  $PC+1 \rightarrow PC$

The accumulator is replaced by the two's complement of its present value. (The format for this instruction is shown in Figure 3.10.)

Logical Instructions

ANA (I)

And (Logical Product)

(7.2  $\mu$ sec)

$A \cap X \rightarrow A$  ;  $PC+1 \rightarrow PC$

The logical product (inclusive or) of the accumulator and the operand X is placed back into the accumulator.

ORA (I)

Or (Logic Sum)

(7.2  $\mu$ sec)

$A \cup X \rightarrow A$  ;  $PC+1 \rightarrow PC$

The logical sum of the accumulator and the operand X is formed and placed back into the accumulator.

XOR (I)

Exclusive Or

(7.2  $\mu$ sec)

$A \oplus X \rightarrow A$  ;  $PC+1 \rightarrow PC$

The logical exclusive or of the accumulator and operand X is formed and placed back into the accumulator.

## Jump Instructions

Two types of unconditional jump instructions are provided for in the processor.

The relative jump order (Figure 3.4) allows an unconditional program transfer to be made within a limited area relative to the current position of the program counter. This instruction contains an immediate displacement field, therefore, no additional memory fetches are required.

The indirect jump order (Figure 3.5) permits an unconditional program transfer to be made to any location within 32,768 words of memory. The jump address is obtained indirectly by fetching a 16-bit word from the location specified by the instruction address field. Indexing is permissible with this instruction. The indexing function is to the address portion of the instruction, i.e.,  $(Y+XR) \bmod 2^8$ . The seven most significant bits of the program counter are then appended to the instruction address field if the sector tag bit is set to a one. If the sector tag bit is set to a zero, the indirect jump address is retrieved from sector zero of memory. The most significant bit of the indirect address word defines the memory source for following instruction fetches.

JMR	<u>Jump Relative (Unconditional)</u>	(7.2 $\mu$ sec)
-----	--------------------------------------	-----------------

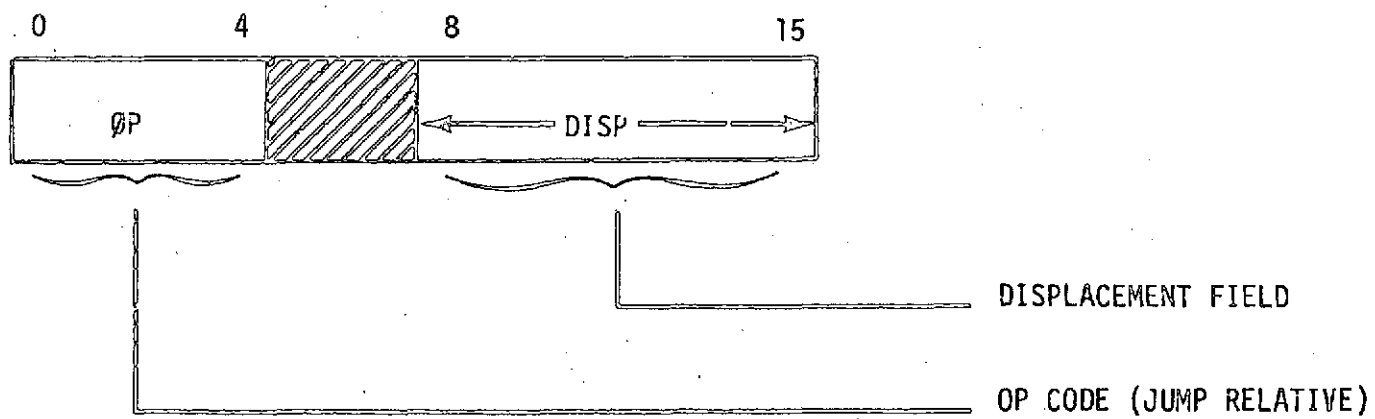
PC+1+DISP  $\rightarrow$  PC

The displacement field (DISP) of the jump instruction is added to the program counter plus one and the next instruction is taken from there. The displacement field is an 8-bit two's complement number; therefore, the relative jump may be between +128 and -127 places with respect to this instruction.

JMI (I)	<u>Jump Indirect (Unconditional)</u>	(9.6 $\mu$ sec)
---------	--------------------------------------	-----------------

(EA)  $\rightarrow$  PC

The contents of the memory cell defined by the EA is placed into the program counter and the next instruction is taken from there. EA is 16 bits in length where the msb identifies the instruction source and remaining 15 bits specify the address in that memory system from where the next instruction is to be taken.

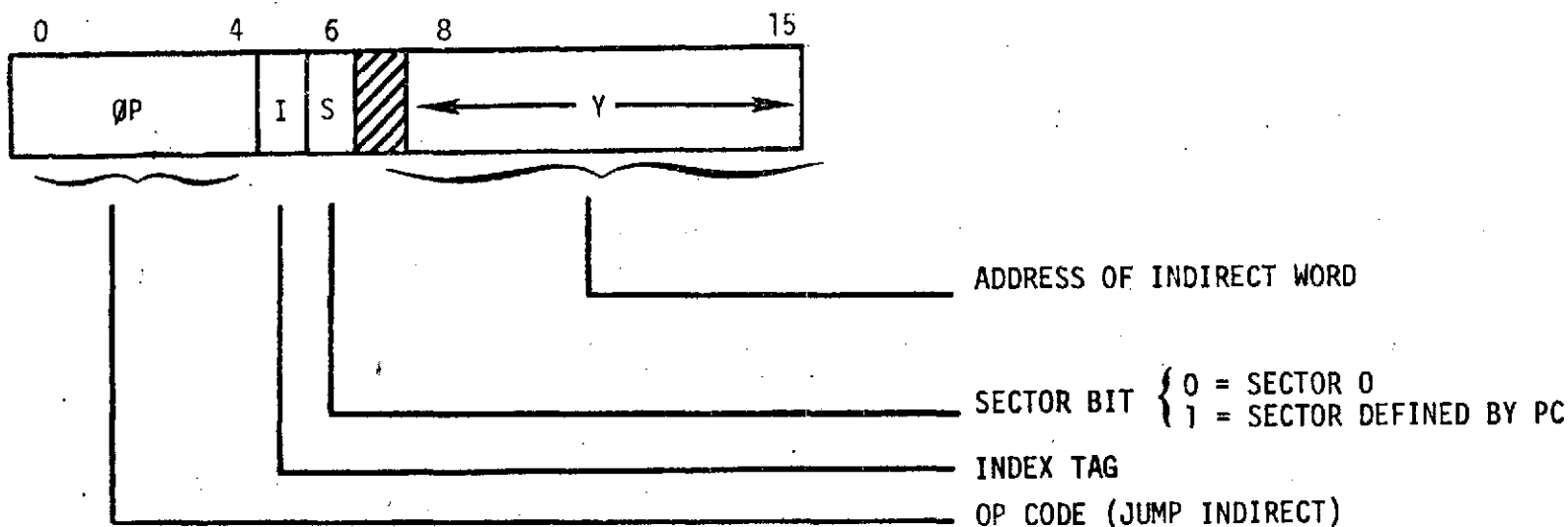


NOTES:

$PC+1+DISP \rightarrow PC$

DISP is an eight bit signed two's complement number

FIGURE 3.4. JUMP RELATIVE (JMR).

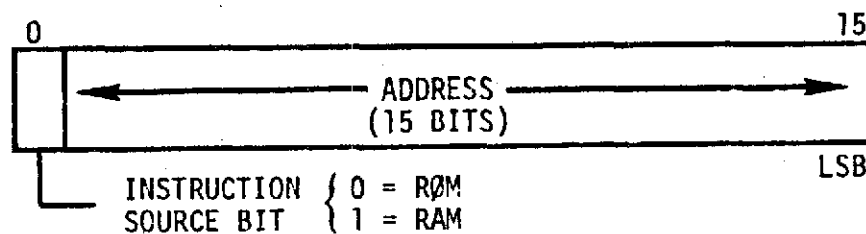


#### NOTES:

If  $S = 0$ :  $EA = Y$   $(EA) \rightarrow PC$   
 If  $S = 1$ :  $EA = Y + 2^{MSBPC}$ ,  $(EA) \rightarrow PC$

THE INDIRECT WORD IS RETRIEVED FROM THE MEMORY CURRENTLY SPECIFIED AS THE INSTRUCTION SOURCE.

THE MSB OF THE INDIRECT WORD SPECIFIES THE NEW INSTRUCTION SOURCE AND THE REMAINING FIFTEEN BITS SPECIFY THE ADDRESS IN THAT MEMORY SYSTEM. THE FORMAT OF THE INDIRECT ADDRESS IS SHOWN BELOW.



INDIRECT ADDRESS FORMAT

FIGURE 3.5. JUMP INDIRECT (JMI).

## Shift Instructions

The instruction format for shift and serial transfer orders is shown in Figure 3.6.

All operations within this class are identified by a single operation code. The specific order to be executed by the ACU is defined by the extended operation code (EOP). The number of places that the register(s) will be shifted is defined by the number N. During the assembly of symbolic instructions into machine code, the assembler places a 5-bit unsigned number N-1 in bit positions 11-15 of the instruction. Note, therefore, that a shift of zero places (e.g., ALS 0) is undefined.

ALS                      Accumulator Left Shift                      (1.2+1.2N  $\mu$ sec)

$N-1 \rightarrow CC$  ;  $A2^{-i} \rightarrow A2^{-(i+n)}$ ,  $0 \rightarrow A15$  ;  $PC+1 \rightarrow PC$

The accumulator is shifted left N places. Bits shifted out of the sign position are lost. Zeros are shifted into the least significant end of the accumulator. A change in the sign position is defined as an overflow and the overflow flip-flop is set.

ARS                      Accumulator Right Shift                      (1.2+1.2N  $\mu$ sec)

$N-1 \rightarrow CC$  ;  $A2^{-i} \rightarrow A2^{-(i+n)}$ ,  $A00 \rightarrow A00$  ;  $PC+1 \rightarrow PC$

The accumulator is shifted right N places. Bits shifted out of the lower end are lost. The sign bit is stretched into the most significant positions. Overflow is not defined.

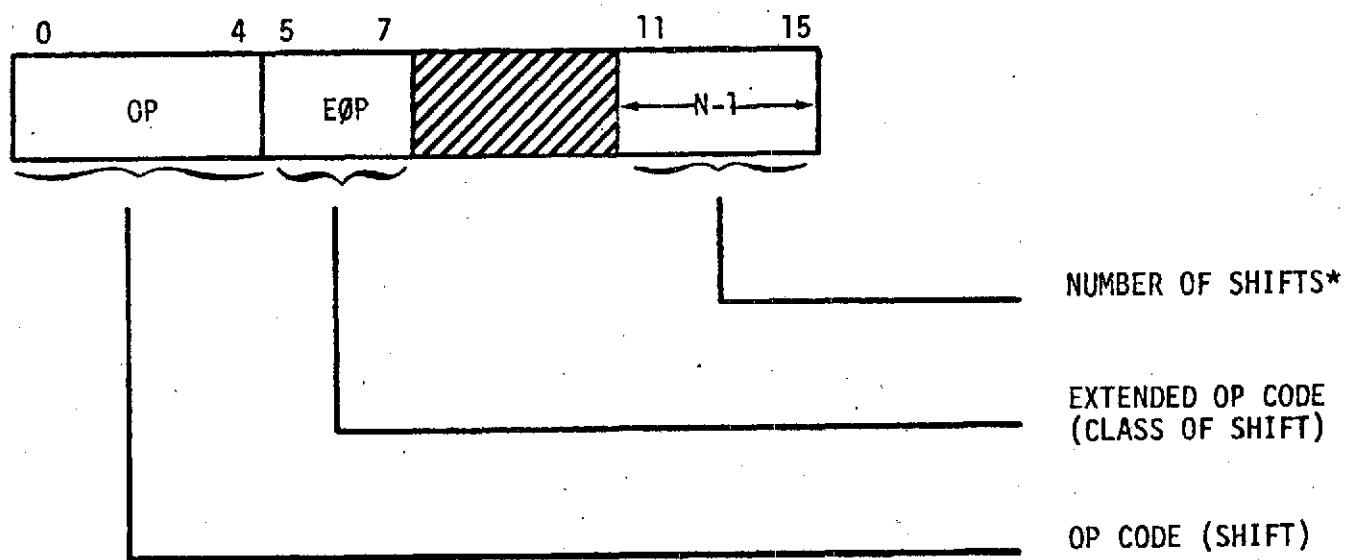
ARC                      Accumulator Right Cycle                      (1.2+1.2N  $\mu$ sec)

$N-1 \rightarrow CC$  ;  $A \cdot 2^{-i} \rightarrow A \cdot 2^{-(i+n)}$ ,  $A15 \rightarrow A00$  ;  $PC+1 \rightarrow PC$

The accumulator is shifted right in a circular manner such that bits shifted out of the least significant position are inserted into the sign position. Overflow is not defined.

LLS                      Long Left Shift                      (1.2+1.2N  $\mu$ sec)

$N-1 \rightarrow CC$  ;  $AQ2^{-i} \rightarrow AQ2^{-(i+n)}$ ,  $Q00 \rightarrow A15$ ,  $0 \rightarrow Q15$  ;  
 $PC+1 \rightarrow PC$



NOTES: \*Number of places shifted = N  
A shift of zero places is undefined.

FIGURE 3.6. SHIFT AND SERIAL TRANSFERS ALS, ARS, ARC, LLS, LRS, LRC, TRC, INS.

The accumulator and Q-register are considered as a single register and both registers are simultaneously shifted left N places. Bits shifted out of the sign of A are lost and zeros fill the least significant bits of Q. Overflow is defined if the sign of A changes.

LRS Long Right Shift (1.2+1.2N  $\mu$ sec)

$N-1 \rightarrow CC : AQ2^{-i} \rightarrow AQ2^{-(i+n)}, A15 \rightarrow Q00, A00 \rightarrow A00 ;$   
 $PC+1 \rightarrow PC$

The A and Q-registers are connected together as in the LLS instruction and shifted right N places. The sign of A is stretched into the most significant bit positions. Bits shifted out of the lower end of Q are lost. Overflow is not defined.

LRC Long Right Cycle (1.2+1.2N  $\mu$ sec)

$N-1 \rightarrow CC : AQ \cdot 2^{-i} \rightarrow AQ \cdot 2^{-(i+n)}, A15 \rightarrow Q00, Q15 \rightarrow A00 ;$   
 $PC+1 \rightarrow PC$

The A and Q-registers are shifted right N places in a circular manner such that bits shifted out of the l.s.b. of A are inserted into the sign of Q and bits shifted out of the l.s.b. of Q are inserted into the sign of A. Overflow is not defined.

Index Test, Decrement, and Skip Instruction (3.6  $\mu$ sec)

SXD Skip on Index Zero or Decrement

if  $XR \neq 0 : XR-1 \rightarrow XR, PC+1 \rightarrow PC$

if  $XR = 0 : PC+2 \rightarrow PC$

If the contents of the index register is not equal to zero, the index register is decremented by one and the next instruction in sequence is taken. If the contents of the index register is equal to zero, the next instruction in sequence is skipped and the index register remains undisturbed. The format of this no address instruction is shown in Figure 3.10.

Subroutine Linkage Instructions

The format for these instructions is shown in Figure 3.7.

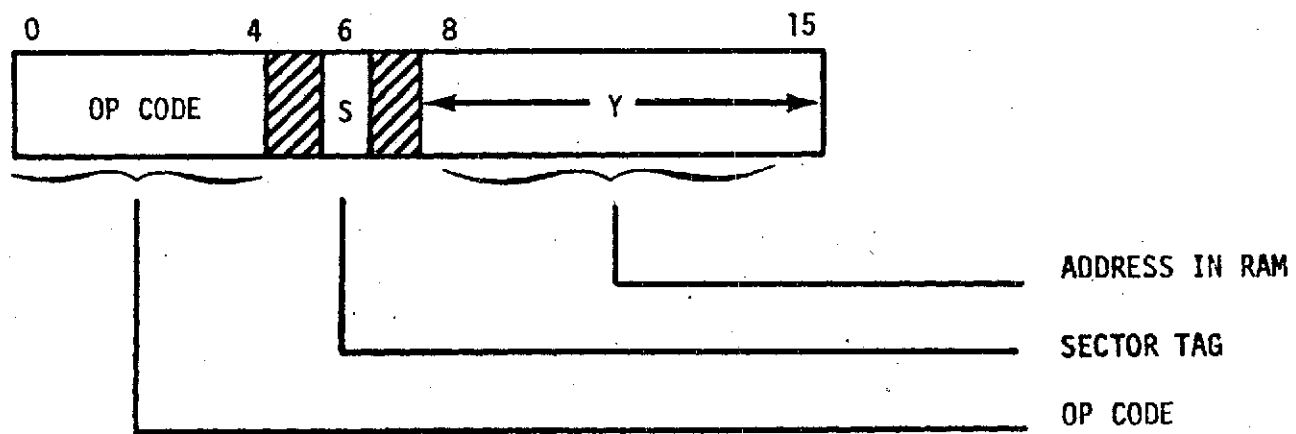


FIGURE 3.7. LINK INSTRUCTIONS SAV, RSR.

SAV	<u>Save Return Link</u>	(8.4 $\mu$ sec)
	PC+1 $\rightarrow$ PC ; PC $\rightarrow$ EA	
	The program counter is incremented by one and stored in the memory cell specified by the effective address EA.	

RSR	<u>Return from Subroutine Call</u>	(9.6 $\mu$ sec)
	(EA) $\rightarrow$ PC ; PC+1 $\rightarrow$ PC	
	The contents of the memory cell defined by the effective address (EA) is transferred to the program counter which is then incremented by one.	

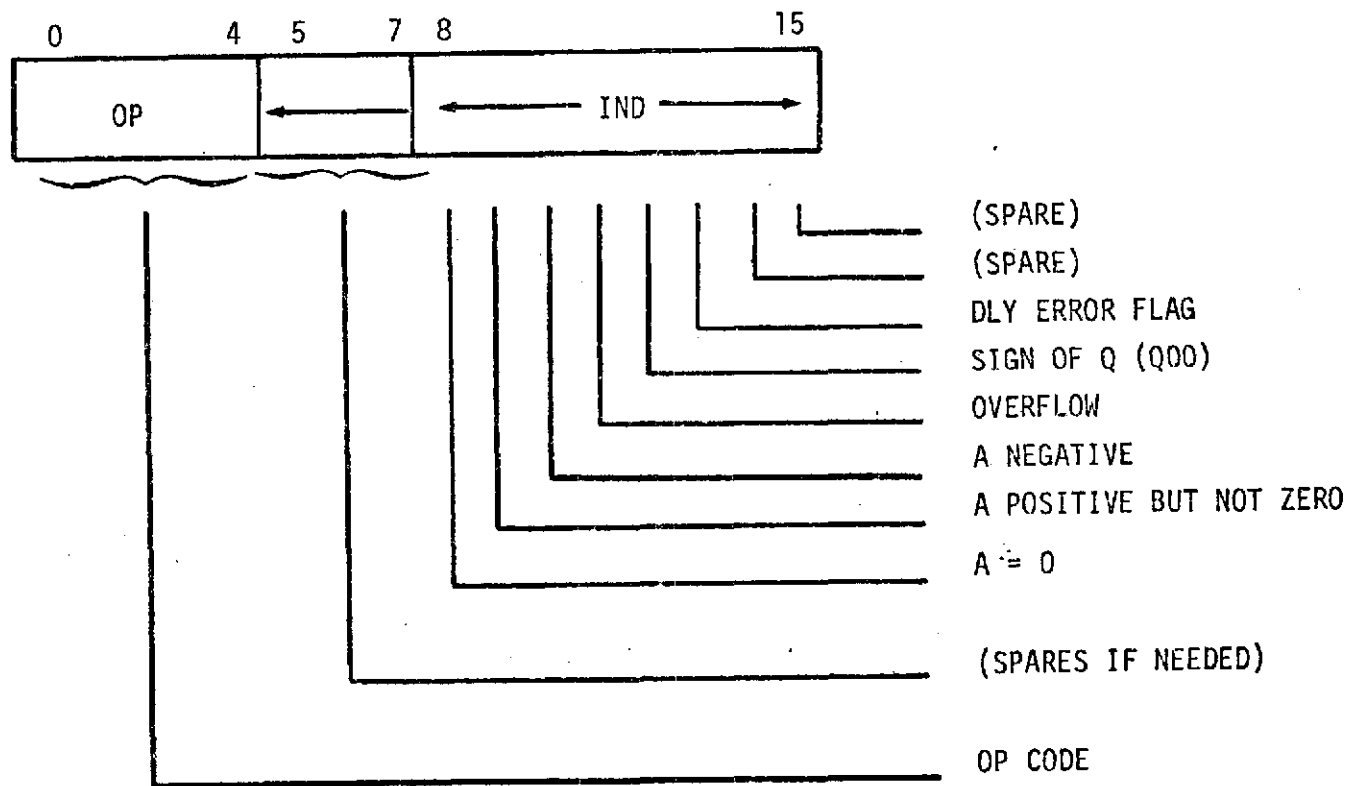
### Internal Machine Indicator Tests

Internal machine indicators are tested by means of a SKI instruction. Indicators that may be tested via this instruction are shown in Figure 3.8. The DLY error flag indicator is set to one by the hardware if a delay advance signal is received when the processor is not executing a DLY instruction. The other indicators are self-explanatory.

SKI	<u>Skip on Internal False</u>	(3.6 $\mu$ sec)
	If internal is true PC+1 $\rightarrow$ PC	
	If internal is false PC+2 $\rightarrow$ PC	
	If any of the internal machine indicators specified by the associated discrete in the IND field are true, the following instruction in sequence is taken. The overflow bit and DLY error flag indicator are reset to zero once they have been tested.	

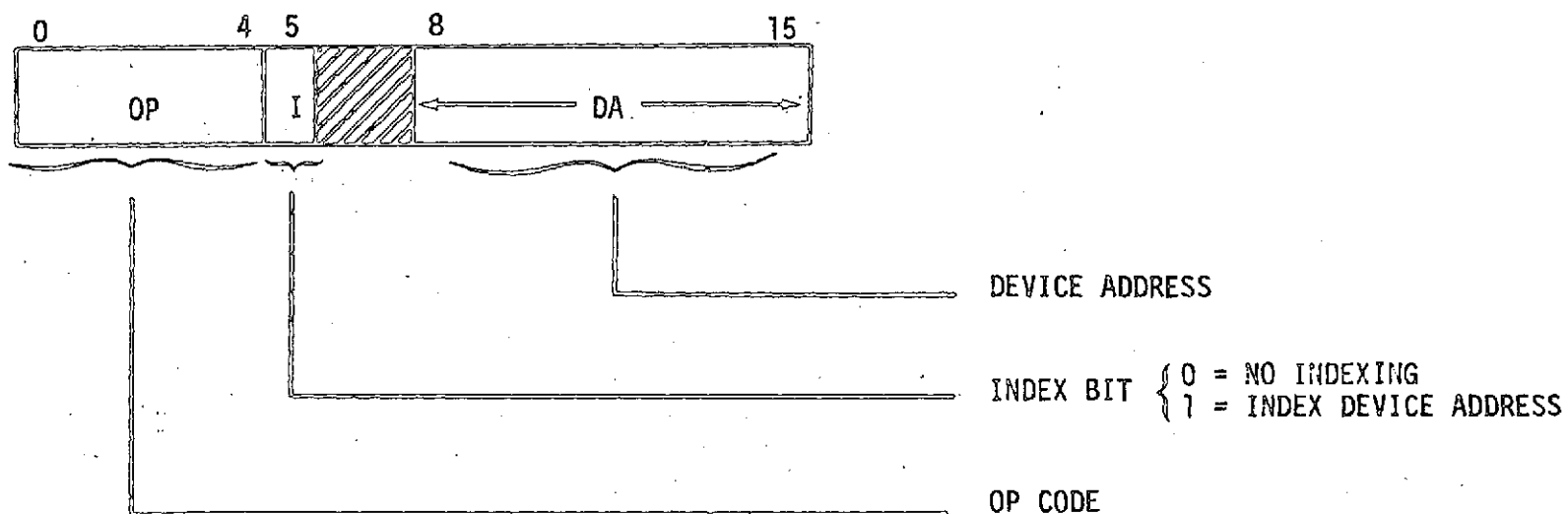
### Input/Output (I/O) Instructions

The instruction format for the following I/O oriented instructions is shown in Figure 3.9. These instructions are used to implement various operations in the IOU. A complete discussion of their significance and method of utilization is provided in Section 3.5, "ACU I/O Interface". It should be noted that the indexing operation performed here is different than for the other machine instructions. For I/O instructions, the indexing operation is applied to the device address field, i.e., if the index tag bit is set  $EDA = [DA + XR] \text{ modulo } 256$ .



NOTE: The overflow bit is reset to zero after being tested.  
The DLY error flag is reset to zero after being tested.

FIGURE 3.8. SKIP ON INTERNAL.



NOTES:

DA = Device Address

EDA = Effective Device Address

If I = 0: EDA = DA

If I = 1: EDA = DA + 1

FIGURE 3.9. I/O INSTRUCTIONS INP, OUT, SBL, RBL, SKE.

### External Device Test

SKE (I)      Skip on External False      (8.4  $\mu$ sec)

If external is true,  $PC+1 \rightarrow PC$

If external is false    PC+2 → PC

If the state of the external device specified by the effective device address (EDA) is false, a "zero" level is returned to the ACU on the common response line and the next instruction in sequence is skipped. If the state of the external device is true, a "one" level is returned to the ACU on the common response line and the next instruction in sequence is taken.

SBL (I)      Set Bi-Level and Skip if Ready      (8.4  $\mu$ sec)

If ready:  $PC+1 \rightarrow PC$

If busy: Set bi-level  $2^{EDA}$ ,  $PC+2 \rightarrow PC$

This instruction is used to set an external bi-level. The bi-level specified by the EDA is set to a logical "one" state if the external device is ready and the next instruction in sequence is skipped. If the external device is busy, no action is taken and the next instruction in sequence is executed.

RBL (I)      Reset Bi-Level and Skip if Ready      (8.4  $\mu$ sec)

If ready:  $PC+1 \rightarrow PC$

If busy: Reset bi-level  $2^{EDA}$ ,  $PC+2 \rightarrow PC$

This instruction is used to reset an external bi-level. The bi-level specified by the EDA is reset to a logical "zero" state if the external device is ready and the next instruction in sequence is skipped. If the external device is busy, no action is taken and the next instruction in sequence is executed.

INP (I)      Input to Accumulator and Skip if Ready

If busy: PC+1  $\rightarrow$  PC (7.2  $\mu$ sec)

If ready: (EDA)  $\rightarrow$  A, PC+2  $\rightarrow$  PC (12.0  $\mu$ sec)

Data contained in the external device specified by the EDA is transferred into the accumulator in 4-bit bytes if the device is ready and the next instruction in sequence is skipped. If device EDA is busy, the next instruction in sequence is executed and the accumulator remains undisturbed.

OUT (I)      Output from Accumulator and Skip if Ready

If busy: PC+1 → PC (7.2 μsec)  
If ready: A → Device EDA; PC+2 → PC (12.0 μsec)

Data contained in the accumulator is transferred in 4-bit bytes to the external device specified by EDA if the device is ready and the next instruction in sequence is skipped. If device EDA is busy, the next instruction in sequence is executed. In either case, the accumulator remains undisturbed.

INS      Serial Input to Accumulator      (1.2+1.2N  $\mu$ sec)

$N-1 \rightarrow CC$  : Serial Data  $\rightarrow A00$ ,  $A \cdot 2^{-i} \rightarrow A \cdot 2^{-(i+n)}$ ,  $PC+1 \rightarrow PC$

N data bits are accepted and transferred serially into the accumulator. External data is accepted l.s.b. first. A transfer gate is provided at the I/O interface to indicate when data is to be transferred. The format for this instruction is shown in Figure 3.4.

Other Instructions (See Figures 3.10 and 3.11)

LDB	Load Data Base Register Immediate	(4.8 $\mu$ sec)
-----	-----------------------------------	-----------------

IMO  $\rightarrow$  Base register ; PC+1  $\rightarrow$  PC

LXR	Load Index Register Immediate	(4.8 $\mu$ sec)
-----	-------------------------------	-----------------

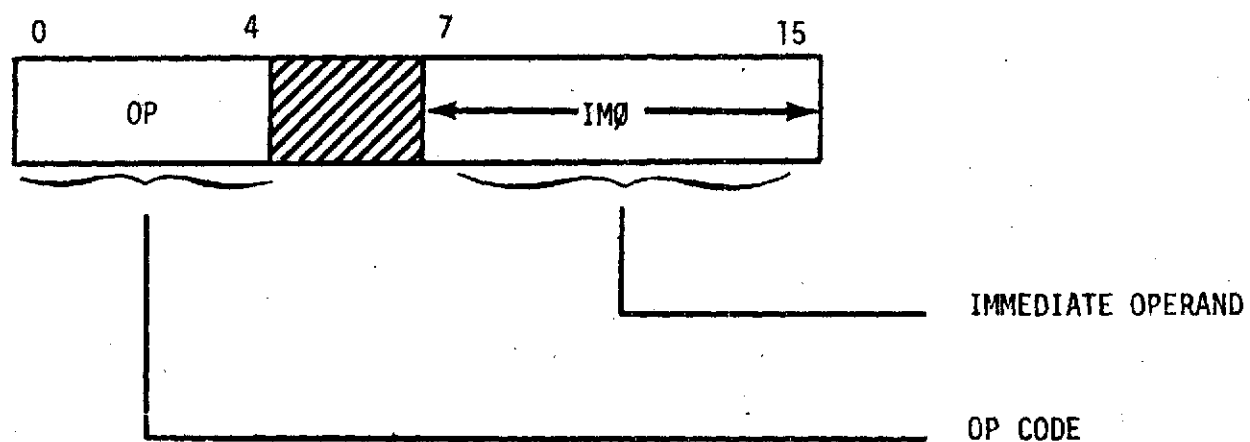
IMO  $\rightarrow$  XR : PC+1  $\rightarrow$  PC

DLY	Delay
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33
34	34
35	35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	48
49	49
50	50
51	51
52	52
53	53
54	54
55	55
56	56
57	57
58	58
59	59
60	60
61	61
62	62
63	63
64	64
65	65
66	66
67	67
68	68
69	69
70	70
71	71
72	72
73	73
74	74
75	75
76	76
77	77
78	78
79	79
80	80
81	81
82	82
83	83
84	84
85	85
86	86
87	87
88	88
89	89
90	90
91	91
92	92
93	93
94	94
95	95
96	96
97	97
98	98
99	99

This instruction causes the machine to halt instruction execution until

- (1) a delay advance timing signal is received from the real time clock.
- (2) an advance signal is received from the programmer/maintenance console.

The next instruction in sequence will commence execution within 1.2  $\mu$ seconds of the leading edge of the delay advance signal.

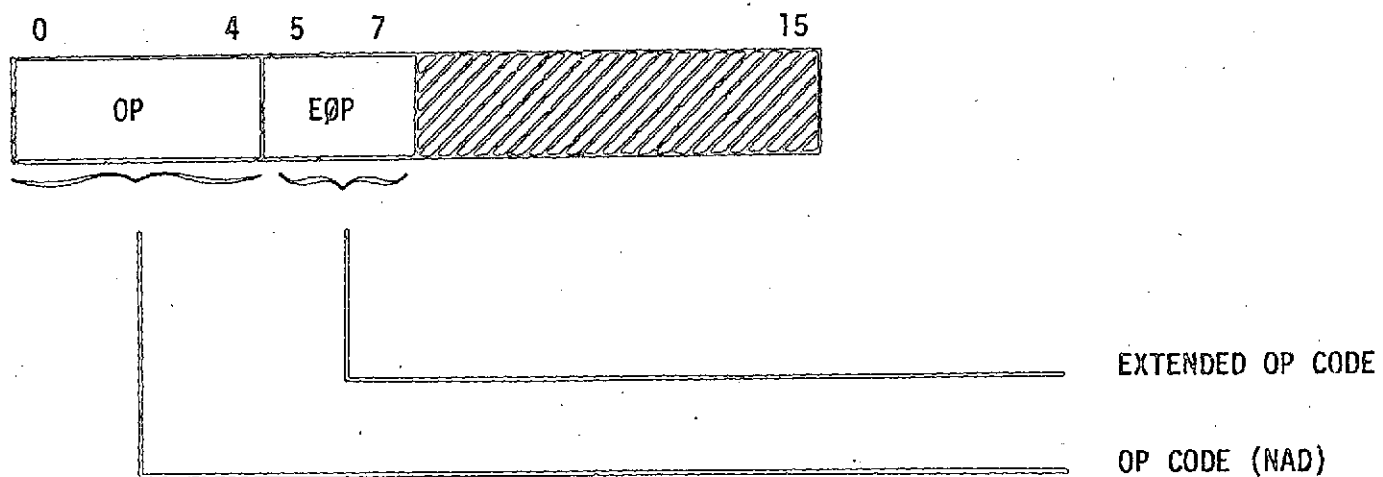


NOTES:

For LXR  $IMØ \rightarrow XR$

For LBR  $IMØ \rightarrow DBR$

FIGURE 3.10. IMMEDIATE INSTRUCTIONS LBR, LXR.



NOTES: The instruction is defined by the EOP code.

FIGURE 3.11. OTHER INSTRUCTIONS (NO ADDRESS) DLY, SXD.

TRC

Transfer Reconfiguration Control

(1.2+1.2N  $\mu$ sec)

N-1  $\rightarrow$  CC; RCU Data  $\rightarrow$  A00  
Q15  $\rightarrow$  RCU,  $AQ \cdot 2^{-1} \rightarrow AQ \cdot 2^{(i+n)}$ ;  
A15  $\rightarrow$  Q00; PC+1  $\rightarrow$  PC

This instruction is used to serially exchange information between the RCU and the A,Q registers of the ACU. The length of the shift performed is specified in the instruction word (see Figure 3.4). During the shifting operation, Q15 is placed on the l.s.b. of the output bus and the serial data stream from the RCU is input to A00 via the l.s.b. of the input bus.

### 3.3.5 Double Precision Operations

Double precision words are carried in the machine as is shown in Figure 3.2. The least significant half of the word is carried in the Q-register and the most significant half is carried in the accumulator. Double precision additions (or subtractions) are automatically produced by the processor when the programmer places ADQ and ADD (or SUQ and SUB) operations in contiguous sequence. Based upon the results of the Q-register operation (i.e., ADQ or SUQ), the hardware automatically prepares the proper carry for the following accumulator operation (i.e., ADD or SUB) so as to produce a double precision result. The time required to implement double precision sums or differences by this technique is 14.4  $\mu$ seconds (19.2  $\mu$ seconds if indexing is specified).

Programming restrictions are as follows:

An ADQ (SUQ) operation will never be followed immediately by a SUB (ADD) instruction.

An ADD (SUB) instruction may immediately follow an ADQ (SUQ) operation only if a double precision result is intended.

There are no other restrictions.

### 3.3.6 Multiply Algorithm

A single precision multiply order produces a double length product having a sign and 30 magnitude bits. A flow chart describing the ACU two-bit-at-a-time multiply algorithm is shown in Figure 3.12. The multiplier is taken from A and placed into Q as part of the hardware implemented micro-routine. The multiplicand is obtained from the memory cell defined by the effective operand address. The resulting product ends up in A, Q as is shown in Figure 3.2. For this operation, the least significant bit of the Q-register (i.e.,  $Q_{15} = 2^{-31}$ ) is undefined. Because of the hardware mechanization of the multiply order, the sign of the multiplier will end up in this position at the end of the operation. (Currently no provision is being incorporated in the hardware to zero this bit for case when a negative multiplier is used. During double

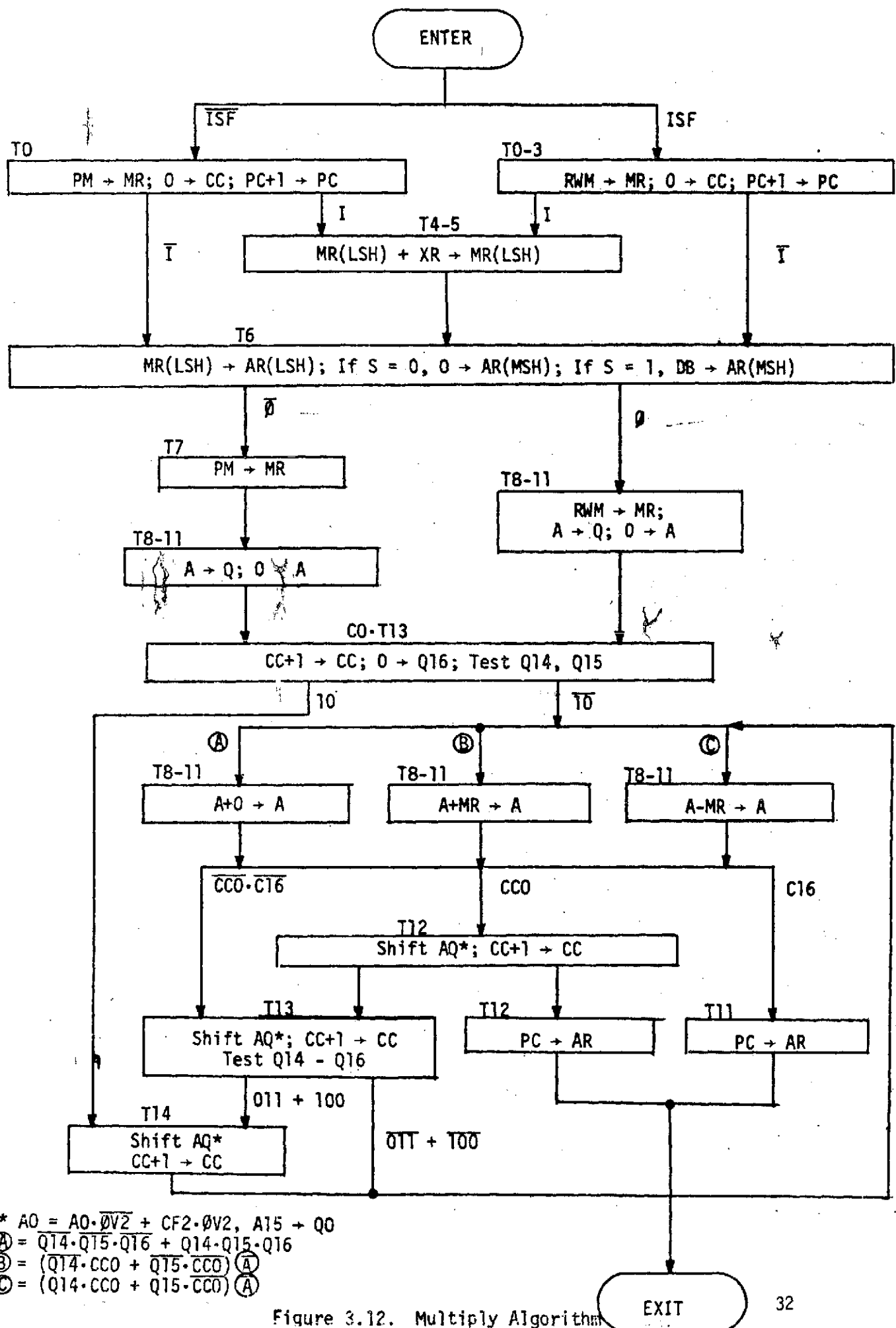


Figure 3.12. Multiply Algorithm

precision addition, a carry out of Q15 is possible if both augend and addend have "ones" in the position.)

The resulting product for several example cases of multiplier and multiplicand is shown in Table 3.2.

TABLE 3.2

Multiplier (A) (Octal)	Multiplicand (X) (Octal)	Product (AQ) (Octal)
A	MR	A   Q
000000	xxxxxx	000000000000
100000	000000	000000000001
100000	100000	100000000001
100000	000001	177777000001
000001	100000	177777000000
177777	077777	177777000003
077777	177777	177777000002
077777	077777	077777177776
000001	077777	000001777776
001216	000001	00000002434

No overflows can be produced from multiplication.

### 3.3.7 Division Algorithm

The DPA uses a non-restoring division algorithm shown in Figure 3.13 to produce a 16-bit quotient from the division of AQ by the operand retrieved from memory. The quotient is generated one bit-at-a-time by this algorithm. The basic process is as follows:

Examine the sign of the dividend (remainder) and divisor. If they are the same, generate a quotient bit of "1" and subtract the divisor from twice the remainder. If they are different, generate a quotient bit of "0" and add the divisor to twice the remainder. Repeat the process until 16 quotient bits have been produced.

Flip-flop DCF holds the result of the remainder-divisor sign comparison test ( $DCF = 0$ , signs agree;  $DCF = 1$ , signs disagree), and therefore, represents the new quotient bit and specifies the next arithmetic operation to be performed. The equation for the quotient bit is  $\overline{C0} (C1 + DCF)$ . The first meaningful bit of the quotient produced is the sign bit which introduced into Q15 when the cycle counter equals one (C1). This particular bit is inserted into Q15 as the inverse of DCF in order for the sign of the quotient to end up in the proper state at the end of the divide operation.

To implement quotient round-off, the seventeenth bit of the quotient is determined by the same process. If it is a "one",  $2^{-15}$  is added to the quotient as it is transferred into the accumulator. A quotient round-off resulting in an accumulator overflow will set the overflow indicator. The Q register is zeroed at the end of the divide operation and the rounded quotient appears in the machine's accumulator.

Divisor Smaller than Dividend - If scaling is incorrect such that the divisor is smaller than the dividend, an incorrect quotient will be produced. There is no special hardware included in the DPA to test for a division over-capacity condition. The overflow flip-flop will not necessarily be set if scaling is incorrect.

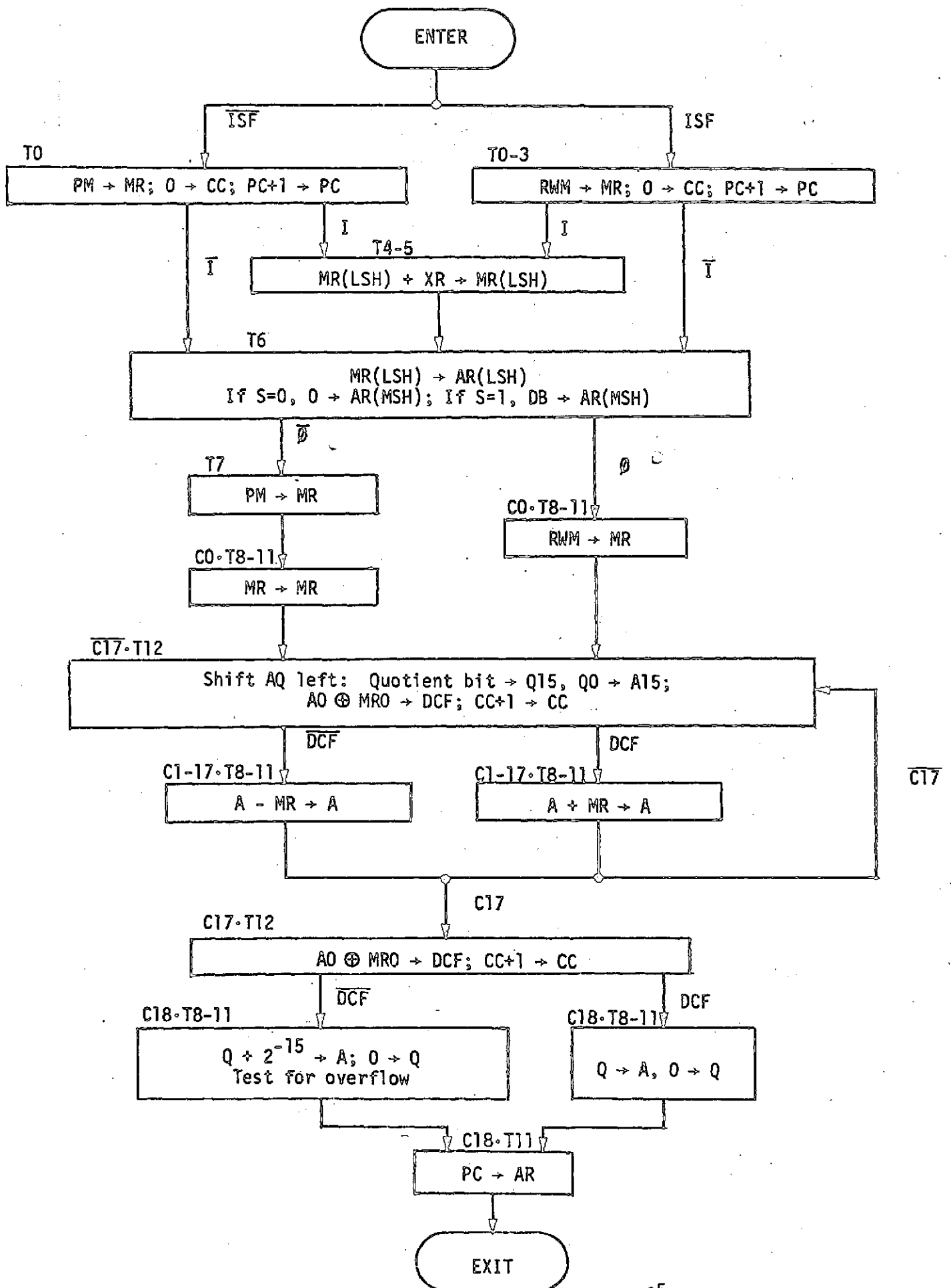


Figure 3.13. Divide Algorithm

Divisor Equal in Magnitude to Dividend - Division of numbers of equal magnitude should result in a quotient of either plus or minus one. A quotient of "plus one" is not representable in the machine and will cause the overflow indicator to be set during quotient round-off. A quotient of "minus one" will be correctly represented in the machine as 1.000....00.

### 3.4 ACU INTERNAL ORGANIZATION

The ACU is internally organized as shown in the block diagram, Figure 3.14. Each functional block shall operate as described in the following paragraphs.

In this section the following abbreviations are used:

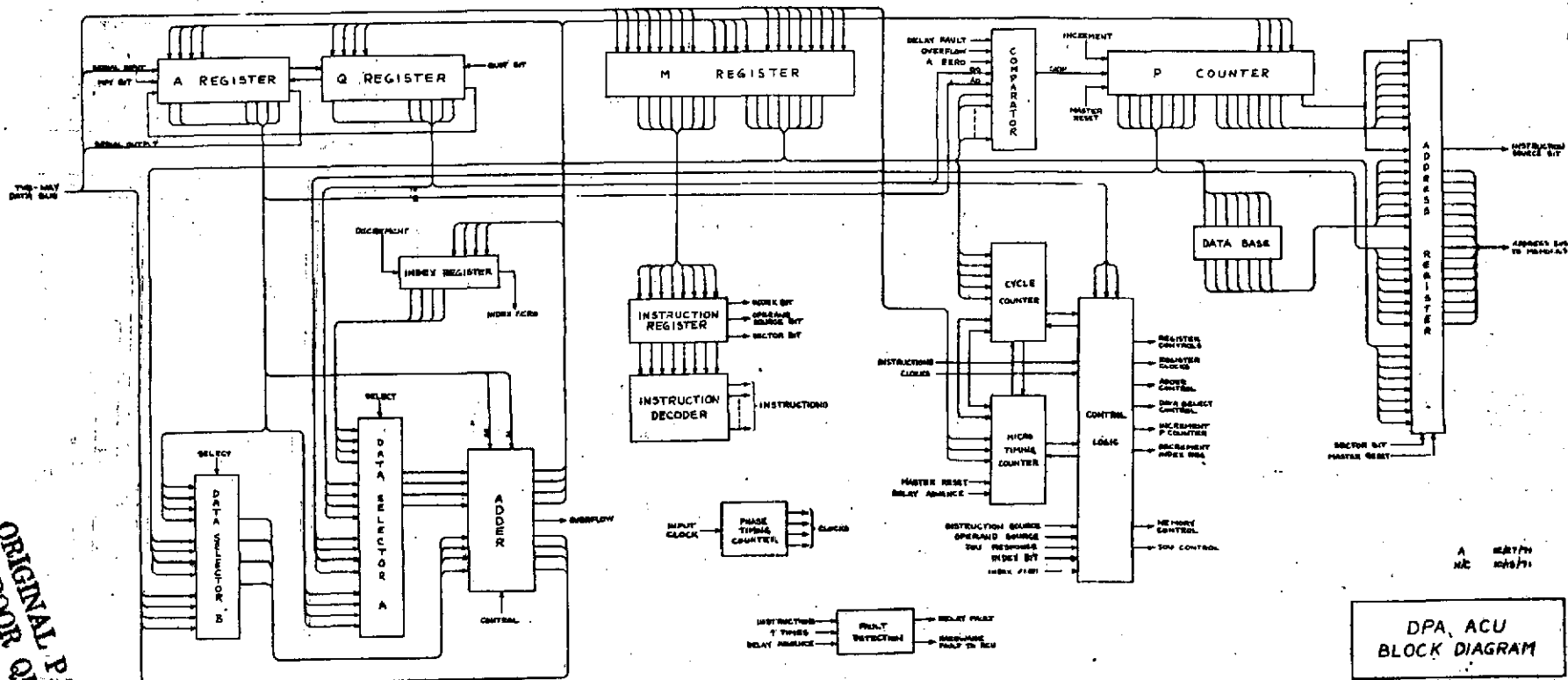
A	=	A Register	IR	=	Instruction Register
AR	=	Address Register	M	=	Memory Register
CC	=	Cycle Counter	MTC	=	Micro-Timing Counter
DB	=	Data Base Register	P	=	Program Counter
DSA	=	Data Selector A	PTC	=	Phase Timing Counter
DSB	=	Data Selector B	Q	=	Q Register
ID	=	Instruction Decoder	XR	=	Index Register

#### Adder

The ACU contains a 4-bit Adder that performs the following operations on two's complement numbers: add, subtract, exclusive-OR, inclusive-OR and AND. The Adder is used not only for arithmetic and logical operations on data, but also for address modification (indexing), for performing relative jumps and for all byte-organized data transfers between registers, the RWM and the IOU. The Adder receives its two inputs from the two data selectors and provides a buffered output to the various ACU registers, and to the RWM and IOU on the output data bus. In subtraction the output of DSB is subtracted from the output of DSA.

The Adder contains two carry flip-flops and two overflow flip-flops. One flip-flop is used only during indexing additions. The other carry flip-flop saves the carry bit between data additions as well as the carry from the last byte of the operation. The first overflow flip-flop is used during ADD, SUB, ALS and LLS, where an overflow condition sets the flip-flop and can be sensed by SKI. The overflow is reset only by a SKI instruction that tests for overflow. The second overflow flip-flop is used only during MPY when overflows normally occur, and it cannot be sensed by SKI.

ORIGINAL PAGE IS  
OF POOR QUALITY



DPA ACU  
BLOCK DIAGRAM

Figure 3.14

The Adder facilitates double-precision addition and subtraction by the following feature: when an ADQ or SBQ is immediately followed by an ADD or SUB, the carry bit is properly handled to execute the double-precision operation.

#### Data Selector A and B

Data Selector A (DSA) has four 4-bit inputs; A, Q, XR and P. Data Selector B (DSB) has three 4-bit inputs; A, M and the Data Bus. DSB selects A only during CMP; all other instructions requiring A as an operand use DSA. Either selector may be disabled when the Adder is used as a data path for transferring data between registers, memory and IOU.

#### A Register

This 16-bit register is the primary arithmetic register of the ACU and consists of a sign bit and 15 magnitude bits. It is capable of shifting in three modes: right one bit, left one bit and right four bits (one byte). All inputs are clocked into the register with CLKC and six of the bits, A0, A1, A12, A13, A14 and A15 are dual ranked and change at CLKD.

A contains the multiplier at the beginning of MPY, and it is transferred to Q while A is cleared by the MPY instruction. At the end of MPY, A contains the MS half of the product. At the beginning of DVD, A contains the MS half of the dividend and the rounded quotient at the end. The only byte input to A is the Adder output. Serial inputs to A are from the serial I/O interface, the RCU, Q, and the input determined by the MPY control logic.

#### Q Register

This 17-bit register is the secondary arithmetic register of the ACU. For single-precision operations, Q consists of a sign and 15 magnitude bits, but consists of 16 magnitude bits for double-precision operations. The 17th bit is used only during MPY to determine the manner of forming each new partial product. Q has the same three shift modes as A. Six bits of Q (Q0, and Q12 through Q16) are dual ranked as in A. It receives the multiplier from A in the first step of MPY, and contains the LS half of the product at the end. It contains the LS half of the dividend at the beginning of DVD. The only byte input to Q is the Adder output. Serial inputs consist of A0 and the quotient bit.

### Index Register (XR)

This 8-bit register contains a number used to modify memory and I/O device addresses. When indexing is specified by an instruction, XR is added to the 8 LSB of M and the result placed in the 8 LSB of M. The register operates in two modes: as a down-counter with a zero detector and as a two-byte shift register. It shifts in bytes and recirculates when indexing and can be loaded from M or RAM via the Adder. The down-counter is clocked by SxD.

### Memory Register (M)

This 16-bit register receives instructions or data from ROM in parallel or in bytes from RAM. The MS half is single ranked, and is loaded by CLKC. The LS half is dual ranked, the first rank is loaded by CLKC and the second rank loaded from the first by CLKD. The LS half of MR is provided in parallel to the Address Register (AR).

M contains the multiplicand during MPY and the divisor during DVD. During these two operations, M is loaded from memory and recirculated during the execution of the instruction.

In a JMR instruction the LS half of M is considered to be a signed, two's complement number that is added to P. The JMR shifts the LS half of M four bytes. During this shift the sign bit is written into all 4 bit positions of M input in order to stretch the sign for the addition to the 16-bits of P.

### Instruction Register (IR)

The IR is an 8-bit single-ranked register that is loaded with the operation code, index bit, sector bit, and operand source bit, from the MS half of M by CLKD when M has been loaded with an instruction. This register is not clocked when data is being handled in M. The register output feeds the ID.

### Instruction Decoder (ID)

The input to the ID is the eight-bits stored in the IR. Five of these bits are decoded to determine the appropriate operation to be performed by the ACU. The ID also decodes the remaining three bits to indicate the type of shift or the type of no-address operation to be performed. Depending on the class of instruction, the ID generates: 1) an output that causes the AR to select sector 0 or the DB in forming addresses and 2) an output that indicates whether indexing is to be performed or not. The details of the instruction format are given in Paragraph 3.3.

### Program Counter (P)

P is a 16-bit up-counter that is used to provide the memory address for retrieving instructions. In normal operation, P is incremented each time an instruction is executed in order to execute the instructions in sequence. Fifteen of the bits are used for the address, giving a limit of 32,768 words of memory. The MSB of P is used as the instruction source flip-flop (ISF) that indicates whether instructions are being read from ROM or RWM. The P output is provided in parallel to the AR.

The P also behaves as a byte shift register when it is being loaded, stored or modified by a JMR. When shifting, its input is the Adder output, and its output goes to DSA. During SAV, P is effectively recirculated by loading it with the Adder output. The master reset pulse resets P in order to initiate proper operation.

### Data Base (DB)

This 7-bit register provides the 7 MSB of operand addresses to the AR while the 8 LSB are provided by M. It can be loaded in parallel from M by LDB.

### Address Register (AR)

The AR provides a 15-bit parallel address bus to both memories. The Control Logic causes the AR to select the proper address source or forces zeros for the seven MS address bits. When fetching instructions, the AR selects all 15 bits from P. When fetching operands, the AR selects the 8 LSB from M and either selects the 7 MSB from the DB or forces zeros, depending upon the sector bit of the instruction. The address for JMI is formed from the 8 LSB of M and either the 7 MSB of the PC or forced zeros, as dictated by the sector bit.

### Phase Timing Counter (PTC)

This two-bit counter with its output gating circuits generates the four-phase clock. The input frequency is eight times the output frequency. In most cases data is clocked into a register or written into memory with CLKC, and register or counter outputs change with CLKD.

### Micro-Timing Counter (MTC)

This 4-bit counter generates a sequence of T times that are used to control each of the events that occurs in the execution of an instruction. Except for shift instructions, and DLY, each T time is one clock-time long. The counter begins at T0 and advances from one state to another as instruction execution progresses and returns to T0 when it is completed. The counter either counts up in a binary sequence or is loaded in parallel with certain hard-wired numbers, depending on the sequence required for the particular instruction. The counter is reset by the master reset pulse.

Typical operations that occur during each T time are given below:

- T0 Read instruction from ROM into M and IR if ISF = 0, Increment P.
- T0-3 Read instruction from RWM into M and IR in bytes if ISF = 1, Increment P in T0.
- T4 Perform shift instructions, delay or serial input.
- T4-5 Index memory address or send I/O device address to IOU.
- T6 Select operand address and decode it in the RWM.
- T7 Load M from PM if so specified by operand source bit.
- T8-11 Perform byte-organized arithmetic and data transferring operations.

T12-14 Perform shifts required during MPY and DVD.

T15 Not used; entry is a fault condition.

### Cycle Counter (CC)

The CC is a 5-bit binary up-counter that is used to count the number of operations performed in MPY and DVD, and to count the number of clocks applied during the shift and serial input instructions. Some of the states are decoded to provide control signals for MPY and DVD. A comparator is used by shift instructions to compare the counter contents with 5 LSB of the M and causing the MTC to return to T0 when a comparison is found. The CC is not used by the majority of the instructions.

### Control Logic

From the decoded instruction, the MTC, CC, PTC and various other signals the Control Logic generates control signals to shift registers, select data, operate the Adder, etc., as shown in the block diagram. The Control Logic supplies signals to the memories and IOU as well as the ACU.

### Comparator

This circuit is used only by the SKI instruction. Each of the status bits that are tested by SKI are ANDed with a corresponding bit in M and the results ORed to determine if the next instruction is to be skipped.

### Fault Detection

Three fault conditions are detected by this circuit:

- Execution of an unused instruction code.
- Entry of the MTC to state T15.
- Receipt of a Delay Advance signal when the ACU is not executing DLY.

The output of this circuit is provided to the RCU. Detection of a delay fault can be sensed by the SKI instruction.

### 3.5 ACU INPUT/OUTPUT (I/O) INTERFACE

#### General

This section provides a general description of the ACU I/O mechanization. All communications with the central arithmetic processor are via this interface. The design of IOU's shall be compatible with the ACU interface mechanization described herein. A basic signal interface diagram of the ACU I/O is shown in Figure 3.15. System timing specifications are provided for each particular I/O instruction that may be executed by the ACU.

#### ACU I/O Signal Interfaces

Common Output Bus - All information transfers from the ACU are performed over a 4-bit wide common output bus. This bus is used to transfer address or data information from the ACU to the peripheral hardware. Eight bit address transfers are mechanized by sequentially sending two 4-bit address bytes over this interface. Sixteen bit data transfers are implemented by sequentially sending four 4-bit data bytes over this interface. Serial data outputs from the ACU are transmitted to the peripheral hardware via the least significant bit line of the 4-bit wide interface. Control signals are provided to inform the peripheral hardware as to the particular class of information on the output bus.

Common Input Bus - All information transfers to ACU are performed over a 4-bit wide common input bus. Communication over this bus is to the accumulator of the ACU. Sixteen bit data transfers are implemented by externally sending four 4-bit data bytes to the ACU in sequence. Serial data inputs are transmitted to the ACU using the l.s.b. line of the 4-bit wide interface. Control signals are provided to the external hardware to inform the sender when new data should be placed on the input bus.

Common Response Line - The common response line is used by the peripheral hardware to inform the ACU of the conditional response to an

ACU

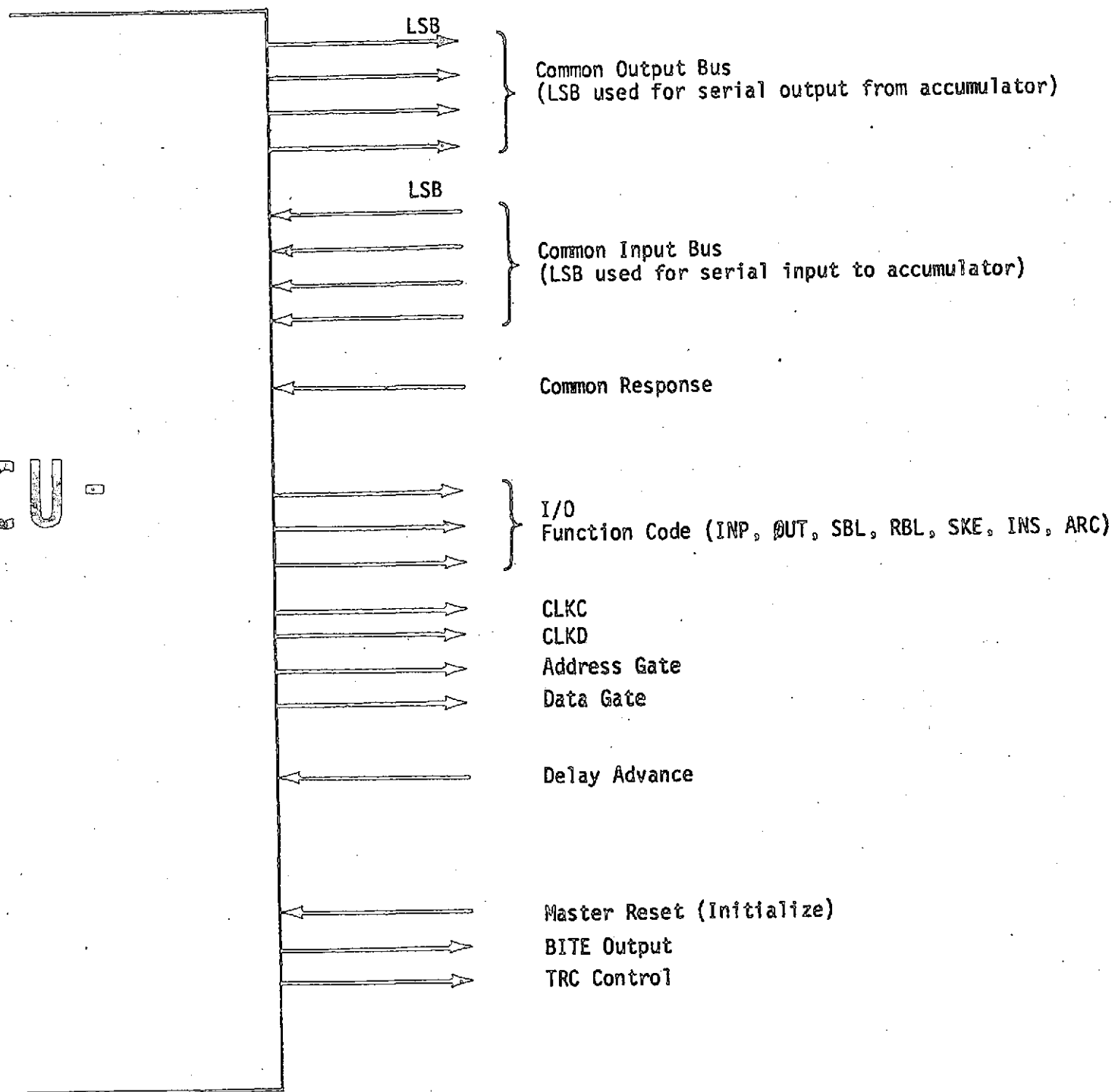


Figure 3.15. ACU I/O Signal Interface Diagram

executed instruction. In the case of an I/O data transfer or bilevel set/reset command this line notifies the ACU whether or not the peripheral is ready to accept (or transmit) the information. In the case of an external indicator test, this line informs the ACU of the specified indicators state (true or false). Timing for the common response signal is shown on the I/O instruction timing diagrams.

I/O Function Code - The I/O function code (3 bits) is used to inform the peripheral hardware of the particular I/O instruction currently being executed by the ACU. A unique code is available for each ACU initiated I/O order, viz., INP, ØUT, SBL, RBL, SKE, INS, ARC. The codes are shown in Table 3.3. The state of these lines are maintained throughout the execution of the instruction.

Table 3.3

<u>Operation</u>	<u>Function Code</u>
ARC	0 0 1
INS	0 1 0
INP	0 1 1
ØUT	1 0 0
SBL	1 0 1
RBL	1 1 0
SKE	1 1 1

Clocks - Basic 833 kHz ACU generated phase C and D clock pulses (CLKC, CLKD) are supplied to the peripheral hardware over separate lines. All external logic must use these clock pulses in conjunction with appropriate data and/or control lines to initiate logical functions.

Address Gate - The address gate signal is used to inform the external hardware that address information is being placed on the common output bus.

Data Gate - The data gate signal is used to inform the external hardware that (1) data is being placed on the ACU common output bus

in the case of byte or serial data output transfers, or that (2) data should be placed on the common input bus in the case of byte or serial data input transfers.

Delay Advance Signal - This line, when placed in the logical zero state, will cause the ACU to advance to next instruction if it is in the process of executing a DLY instruction.

Master Reset (Initialize) - This line, when placed in the logical "one" state, will cause logic in the ACU to be maintained in a reset state (instruction execution will be inhibited). When the reset line changes from the "one" to "zero" state, the ACU will begin executing instructions from ROM location (0000)<sub>8</sub>.

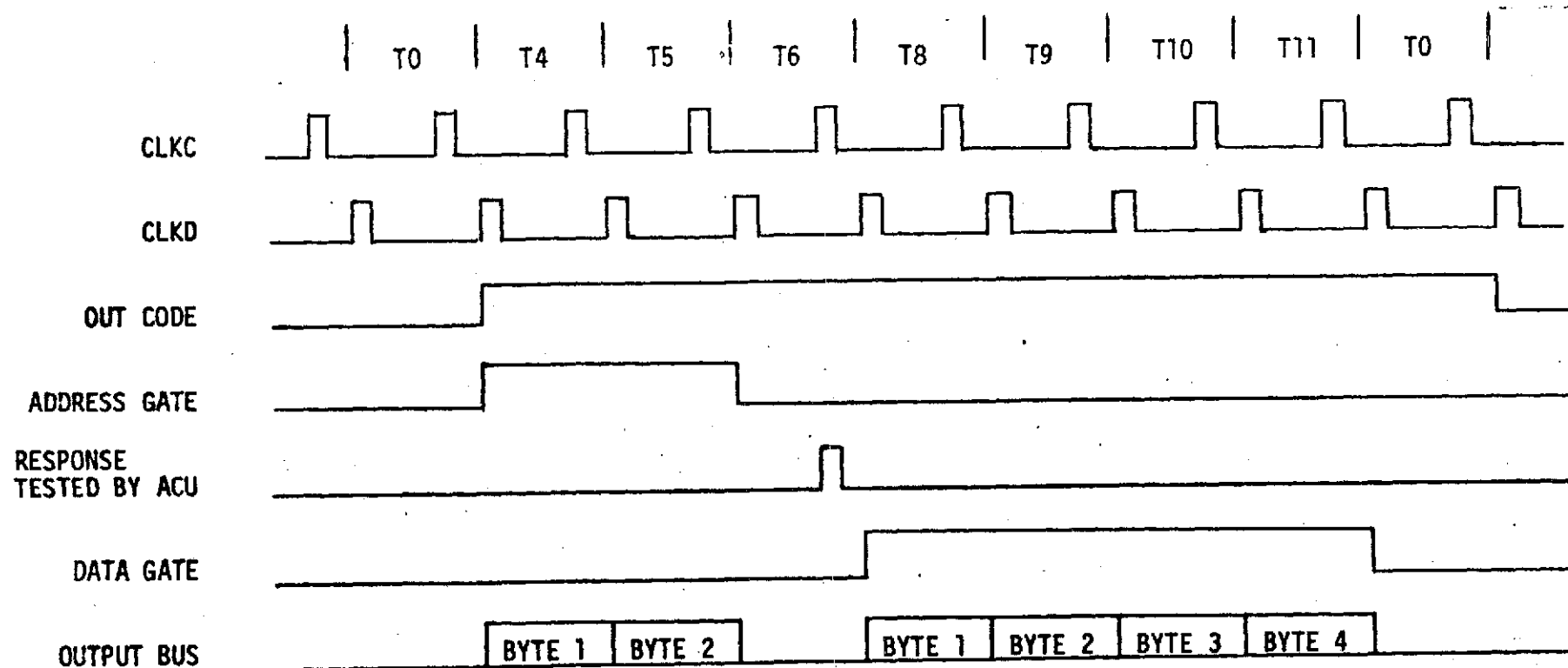
BITE Output - This line is used to inform the RCU or peripheral hardware of the general "health" of the ACU. Built-in test equipment internal to the ACU will cause this line to assume a logical zero state if a fault has been detected. If the ACU is operating properly, this line will be maintained in a logical "one" state. The details of the ACU BITE logic are described in Section 3.6.

TRC Control - The TRC control line is used to inform the RCU that a data exchange is to be initiated between units.

#### Output Instruction

The output (OUT) instruction is used to transfer data from the accumulator of the ACU to an external device. Data transfers are implemented in byte fashion. A device address is supplied to the external hardware in conjunction with this instruction. Timing is as shown in Figure 3.16.

Shortly after the ACU fetches and decodes this instruction, an OUTput code is forced on the I/O function control lines. The peripheral hardware decodes this information to develop an "OUTput in progress" control signal.



BYTE 1 = L.S. BYTE  
 RESPONSE: IF READY  
 NO RESPONSE: IF BUSY  
 IF ISF = 1, OUT WILL REMAIN HIGH THRU T3

Figure 3.16. Output Timing

The OUT instruction contains an 8-bit immediate device address field that identifies the particular external unit to receive the information. The immediate device address (DA) may be modified by the ACU to form an effective device address  $EDA = (XR + DA) \bmod 2^8$  if indexing is specified. During the time interval when the ACU is forming the EDA, the output of the operator (adder) is gated onto the common output bus. An address gate control signal is simultaneously generated by the ACU hardware during this two byte interval. This gate is used by the external device to clock the EDA into the appropriate peripheral logic.

The external hardware initiates a test of the addressed device during the next clock interval. If the device is "ready", a response is sent back to the ACU over the common response line. A ready response causes the ACU to place four sequential 4-bit data bytes on the common output during the interval shown in Figure 3.16. A data gate is simultaneously established during this interval to control and initiate the clocking of data bytes into the specified peripheral register.

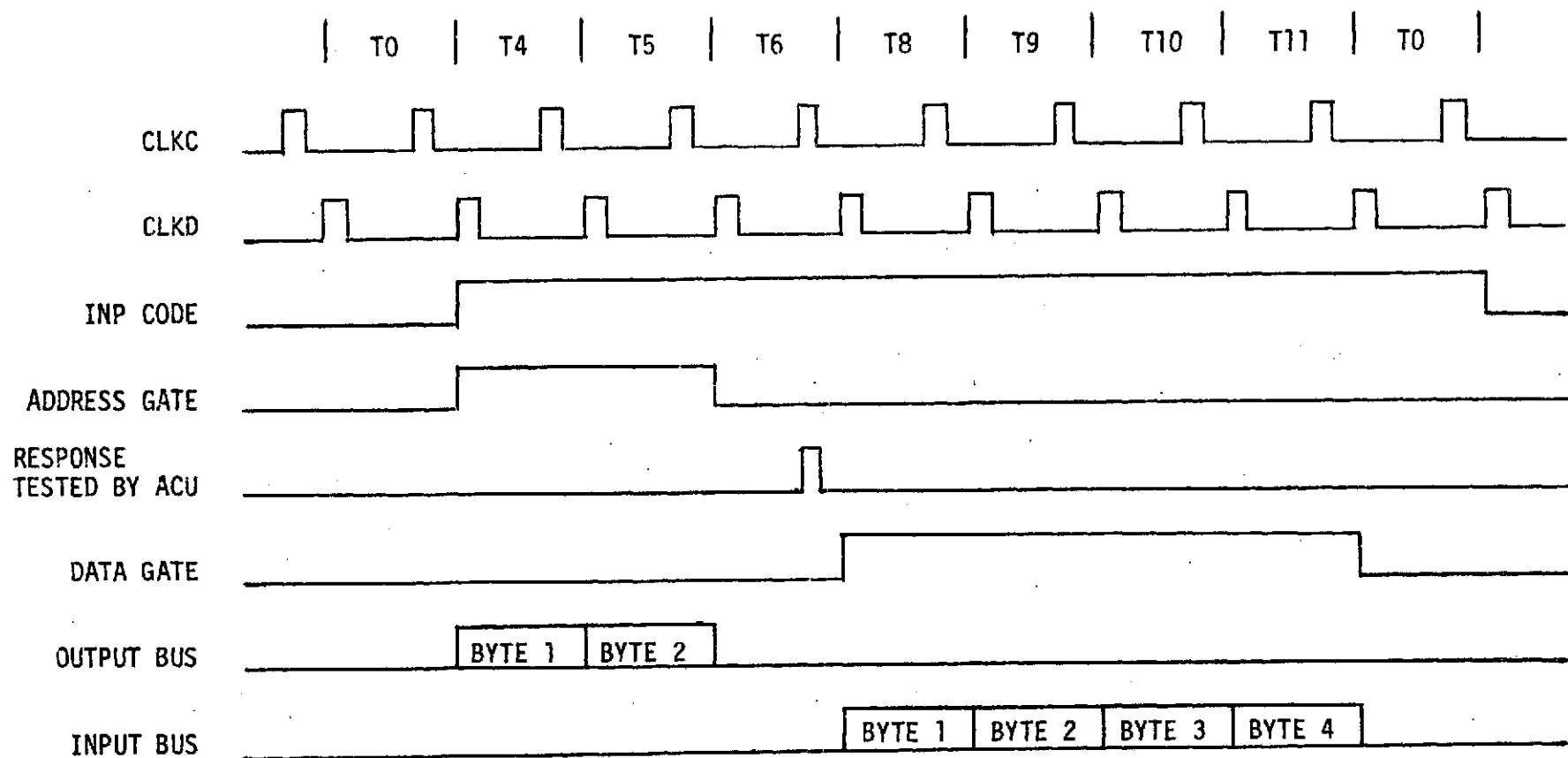
If the device specified is busy, no response is sent to the ACU and the next instruction in sequence is executed. In this case, no data transfers are made. In either case, the contents of the accumulator remains undisturbed.

#### Input Instruction

The Input (INP) instruction is used to transfer data from an external device into the accumulator of the ACU. Data transfers are implemented in byte fashion. A device address is supplied to the external hardware in conjunction with this instruction. Timing is as shown in Figure 3.17.

Shortly after the ACU fetches and decodes this instruction an INP code is forced on the I/O function control lines. The peripheral hardware decodes this information to develop an "INPut in progress" control signal.

The INP instruction contains an 8-bit immediate device address field that identifies the particular external unit to furnish the information



BYTE 1 = L.S. BYTE  
 RESPONSE: IF READY  
 NO RESPONSE: IF BUSY  
 IF ISF = 1, INP WILL REMAIN HIGH THRU T3

Figure 3.17. Input Timing

to the ACU. The immediate device address (DA) may be modified by the ACU to form an effective device address  $EDA = (XR + DA) \bmod 2^8$  if indexing is specified. EDA is gated onto the common output bus during the interval when address formation is mechanized. An address gate control signal is simultaneously developed during this period of time and is used by the peripheral hardware to clock the EDA into the appropriate logic.

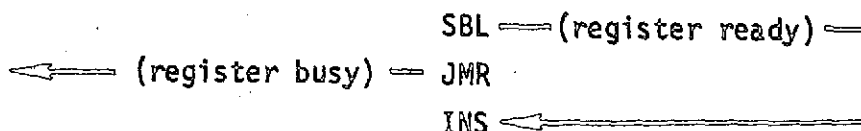
The peripheral device initiates a test of the addressed device during the next clock interval to determine if it is ready to input data to the ACU. If the device is "ready", a response is sent back to the ACU over the common response line. A ready response causes the ACU to initiate a data gate signal that is used by the peripheral device to sequentially gate four 4-bit data bytes onto the common input bus.

If the device specified is busy, no response is sent to the ACU and the next instruction in sequence is executed. In the latter case, the accumulator remains undisturbed.

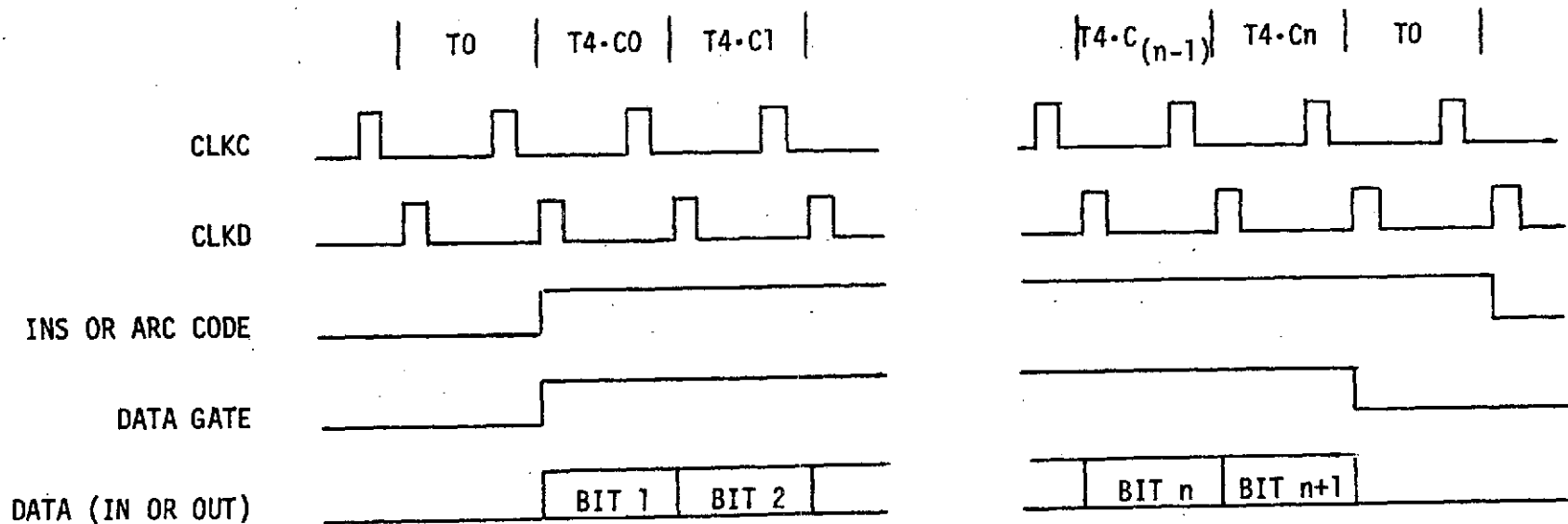
#### Serial Input

The serial input instruction is used to transfer NRZ serial information from an external device into the accumulator of the ACU. Timing is as is shown in Figure 3.18.

It should be noted that no device address is provided in conjunction with this instruction. Therefore, all serial input data transfers must be preceded by a SBL (set bi-level) order to select a particular external device or register as the data source. This instruction would be followed by an INS order as is indicated below:



As soon as the ACU fetches and decodes the serial input instruction, an INS function code is forced to the interface informing the external device that this order has started. A data gate is supplied to the I/O during the interval when the accumulator is ready to accept the NRZ input stream. The first data bit is transferred into the accumulator



DATA GATE ENCLOSES  $n+1$  CLOCKS  
 BIT 1 = LSB  
 IF ISF = 1, CODE WILL REMAIN HIGH THRU T3

Figure 3.18. Serial I/O Timing

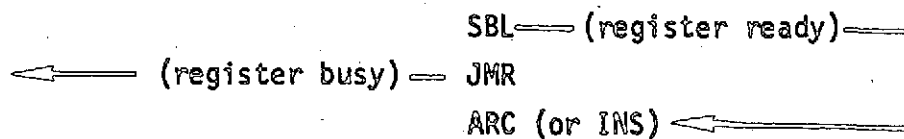
by the first ACU CLKC-pulse following the rise of the data gate. Other sequential bits are accepted as long as the data gate remains true. Upon completion of the transfer, the bi-level should be reset. As an option, this operation may be performed by executing a RBL instruction.

It should be noted that the INS instruction format is identical to other accumulator shift orders. This feature gives the system user added flexibility in that partial word transfers can be mechanized simply by specifying the correct number of shifts (N) in the INS instruction field.

### Serial Output

NRZ serial data may be transferred from the accumulator of the ACU to an external device through the execution of an accumulator right cycle (ARC) instruction. Additionally, the output of the accumulator is made available on the common output bus during the execution of an INS instruction. This feature allows the system user to perform an exchange operation between the contents of an external register and the machines accumulator if so desired. Timing is as shown in Figure 3.18.

It should be noted that no device address is provided in conjunction with this instruction. Therefore, all serial output data transfers must be preceded by a SBL (set bi-level) order to select a particular device or register as the data sink. This instruction would be followed by an ARC (or INS) order as indicated below:



Note that there is no special operation code in the ACU dedicated to a serial output instruction. Whenever an ARC or INS order is executed, the accumulator output is serially placed on the l.s.b. of the common output bus. Since the ARC instruction is frequently used for internal machine operations, it is important that logic in the IOU reset any bi-level that specifies a serial output transfer immediately following the completion of the transfer.

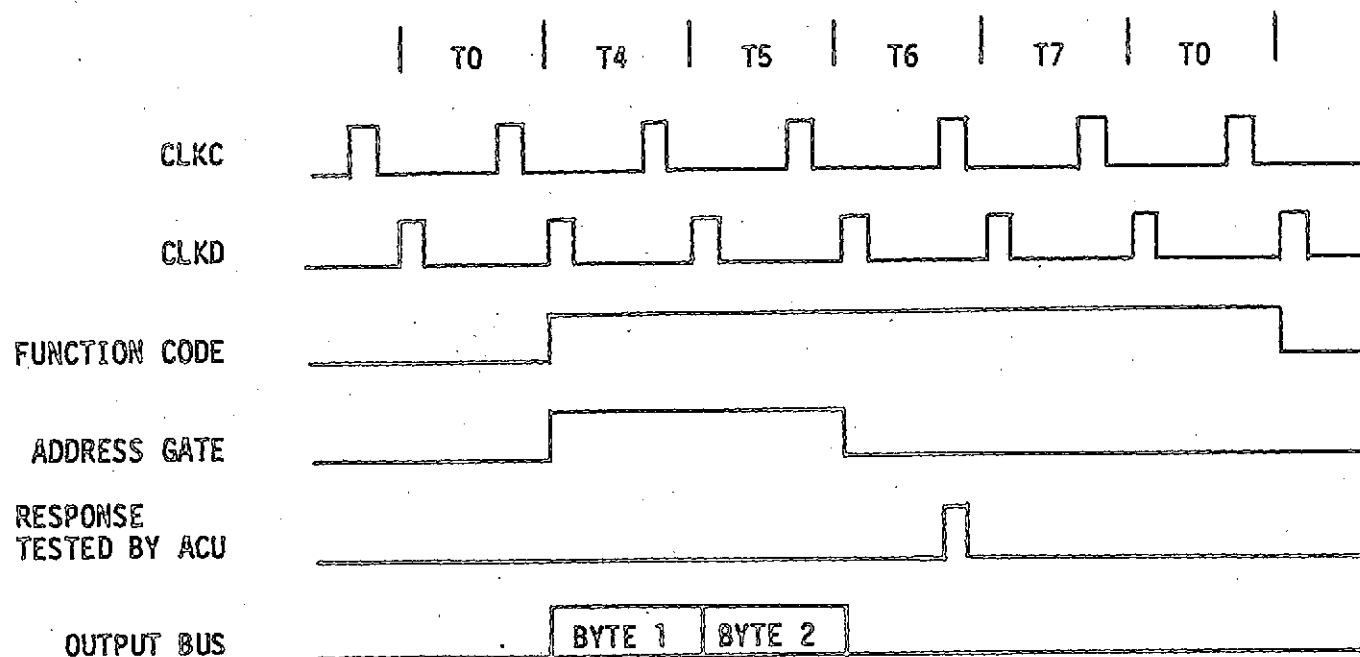
As soon as the ACU fetches and decodes the accumulator right cycle instruction, the ARC code is forced to the interface informing the external device that this order is being executed. A data gate is supplied to the I/O during the interval when the ACU is placing the NRZ output stream on the common output bus. Transfer is performed l.s.b. (A15) first. The first bit is transferred into the external register using the first CLKC pulse after the start of the data gate. Other sequential bits are accepted as long as this gate remains true. Upon completion of the transfer, logic in the IOU should automatically reset the bi-level in question. As an option, the operation may be performed by executing a RBL instruction. The number of bits transferred to the external device is a function of the number of shift positions (N) specified in the ARC instruction field.

#### Skip on External Instruction

The Skip on External (SKE) instruction is used to test the state of indicators (bi-levels or discretes) external to the ACU. Signal timing is as shown in Figure 3.19. Operation is as follows:

Shortly after the ACU fetches and decodes this instruction, an SKE code is forced on the I/O function control lines. These lines are maintained in this state until the instruction execution is completed. The I/O unit or peripheral hardware attached to the ACU decodes this information to develop an "SKE in progress" control signal.

The SKE instruction contains an 8-bit immediate device address field that specifies the particular indicator to be tested by the external hardware. The immediate device address (DA) may be modified by the ACU to form an effective address  $EDA = (SR + DA) \bmod 2^8$  if indexing is specified. During the time interval when the ACU is forming the EDA, the output of the ACU operator (adder) is gated onto the common output bus. An address gate control signal is simultaneously generated by the ACU hardware during this two byte interval. This gate is used by the external device to control the clocking of the EDA into the appropriate peripheral logic.



BYTE 1 = L.S. BYTE  
 RESPONSE: IF READY OR INDICATOR TRUE  
 NO RESPONSE: IF BUSY OR INDICATOR FALSE  
 IF ISF = 1, CODE WILL REMAIN HIGH THRU T3

Figure 3.19. SBL, RBL, and SKE Timing

The external hardware initiates a test of the specified indicator during the next clock interval. If the indicator is in a "true" state, a response signal is transmitted to the ACU over the common response line.

If the indicator being tested is in the false state, no response is transmitted to the ACU. The ACU examines the response line at the appropriate time to determine whether the next instruction in sequence should be executed (external true = response received) or skipped (external false - no response received).

#### Set Bi-level and Reset Bi-level Instruction

The Set/Reset Bi-level instruction is used to set or reset a discrete or bi-level in the peripheral hardware. Signal timing is as shown in Figure 3.19.

These instructions operate similarly to the SKE instruction with the exception that an external bi-level or discrete is controlled (set or reset) rather than being tested. The SBL/RBL instructions may best be thought of as a test, control, and skip order. That is to say, some external device is first tested to ascertain its availability. Based upon this test, the device is either commanded to a new state (if ready) or left undisturbed (if busy). The ACU is notified whether or not the command has been accepted by examining the response sent from the peripheral unit.

Immediately after the ACU fetches and decodes one of these instructions, the proper I/O function code is forced to the interface, i.e., SBL or RBL. Decoding of these lines at the peripheral device effectively form a SBL or RBL "in progress control signal".

As in the SKE instruction, the immediate device address field of the instruction word may be modified to form an effective address  $EDA = (XR + DA) \bmod 2^8$  if indexing is specified. During the time interval when the ACU is forming the EDA, the output of the operator (adder) is gated onto the common output bus. This information is gated into the appropriate peripheral logic during the period identified by the address gate control signal.

As mentioned previously, this instruction may be used to initiate the transferral of data into an external register. For this class of operation, a response is needed to inform the ACU whether or not external register is ready to accept the command. For bi-level or discrete set/reset operations, the busy/ready test is superfluous since the bi-level register is always ready to accept new commands. Because of the mechanization of the instruction, it is still necessary to return a positive response signal to the ACU.

The response signal is transmitted to the ACU during the clock period immediately following the fall of the address gate. If a positive response is received, the next instruction in sequence is executed else the next instruction in sequence is executed.

#### Program Restart Following Reconfiguration

After a reconfiguration takes place, the following actions are performed:

- o The DPA will be automatically sequenced to take the first instruction from ROM memory location 00<sub>H</sub>. This is performed by hardware.
- o The program may be written to utilize the "state" of the newly reconfigured DPA. (Which units are on or off of the redundant set.) This information may be input to the A and Q registers by performing a TRC instruction. The configuration of the message in the registers following completion of the data transfer is shown in Figure 3.20.
- o The software must be programmed such that the DLY instruction is executed no earlier than 97.60 msec or later than 100.04 msec after initialization. Failure to do so will result in a timing fault being registered in the RCU.
- o There are no other requirements or restrictions.

The program will then restart with initial conditions and proceed until new fault messages are generated and further reconfiguration occurs.

"1" indicates module is selected

"A"

0

15

ACU A	ACU B	ACU C	IOU A	IOU B	IOU C	DBS A	DBS B	R01 A	R01 B	R02 A	R02 B	RAM A	RAM B	RAM C	RAM D
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

"Q"

0

15

RAM E	RAM F	DSTM1	DSTM2	øABC	øSCAF	øSCBF	DSTA1	DSTA2	DSTB1	DSTB2	—	—	—	—	—
-------	-------	-------	-------	------	-------	-------	-------	-------	-------	-------	---	---	---	---	---

} Differencing  
Status

1 = CLKB Failure

1 = CLKA Failure

0 = CLKB On

1 = CLKA On

} Differencing  
Status

RCU STATUS WORD IN A-Q

FIGURE 3.20

ORIGINAL PAGE IS  
OF POOR QUALITY

### 3.6 ACU BITE

The ACU BITE (Built-in Test Equipment) features included as hardware to facilitate failures to the ACU (or ACU/IOU combination) are as follows:

- o Power failure. When the ACU is turned on by the RCU, logic power should be applied to all of the circuit elements in the unit. If the correct voltages are not being applied to the logic in the modules, the BITE signal assumes a logical zero condition denoting an inoperable unit. This type of failure could be caused by a faulty switch/regulator device or due to an excessive current drain of a failed circuit element.
- o Illegal Instruction Decode - Execution of any unused instruction code is a fault condition. Receipt of a Delay Advance signal when the ACU is not executing DLY. (Also see Incorrect Program Timing, below):
- o Illegal Microtiming Counter State - Entry of the MTC to state T15 is a fault condition.
- o Incorrect Program Timing - The system program may be operated in a free running mode (Mode 1) or be forced into external synchronization by the hardware (Mode 2). The hardware program timer will be wired as follows:
  - o Program Cycle Time = 100.04 milliseconds per loop
  - o "Window" location, 97.60 msec - 100.04 msec

The effect of this is as follows:

- o The program is expected to have executed an ADVance instruction no earlier than 97.60 msec and no later than 100.04 msec after initialization. If the DLY instruction is executed outside of these limits, a fault will be registered in the system.

- o If the DLY instruction is executed prior to 100.04 msec, the program will be transferred out of the ADVance state at the 100.04 msec point.
- o If the DLY instruction is executed late, the program will be transferred out of the ADVance state concurrently with the execution of the instruction.

Thus, if the software is properly timed (within the tolerances stated) the system will cycle exactly at the 100.04 msec rate and no timing faults will be registered.

In addition, there may be system faults, detected by software. The extent of these fault detection methods are limited only by the ingenuity of the programmer and the memory capacity and system cycle time they consume. The fault signal to the RCU for software initiated (detected) faults is SBL 01<sub>H</sub>.

#### 4.0 DPA MEMORY SYSTEM

The DPA is designed to operate with either solid state or plated wire memory units. Four different types of memory units are available for use with the DPA, namely:

- Solid State Read Only Memories (ROM)
- Solid State Random Access Memories (RAM)
- Solid State Alterable Program Memories (APM)
- Plated Wire Memories (PWM)

The organization and operation of each of these memory modules are described herein.

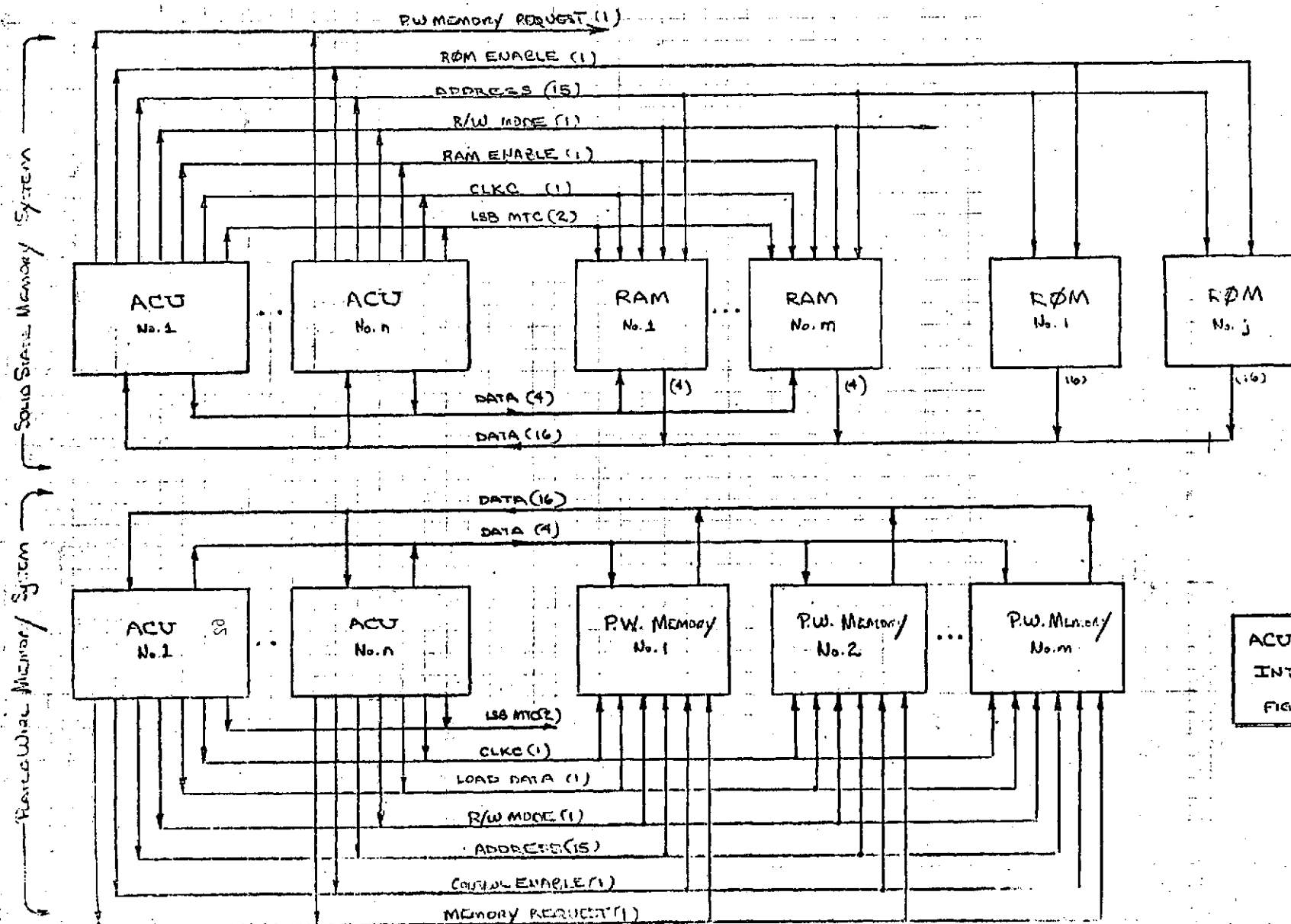
##### ACU-Memory Bus Structure

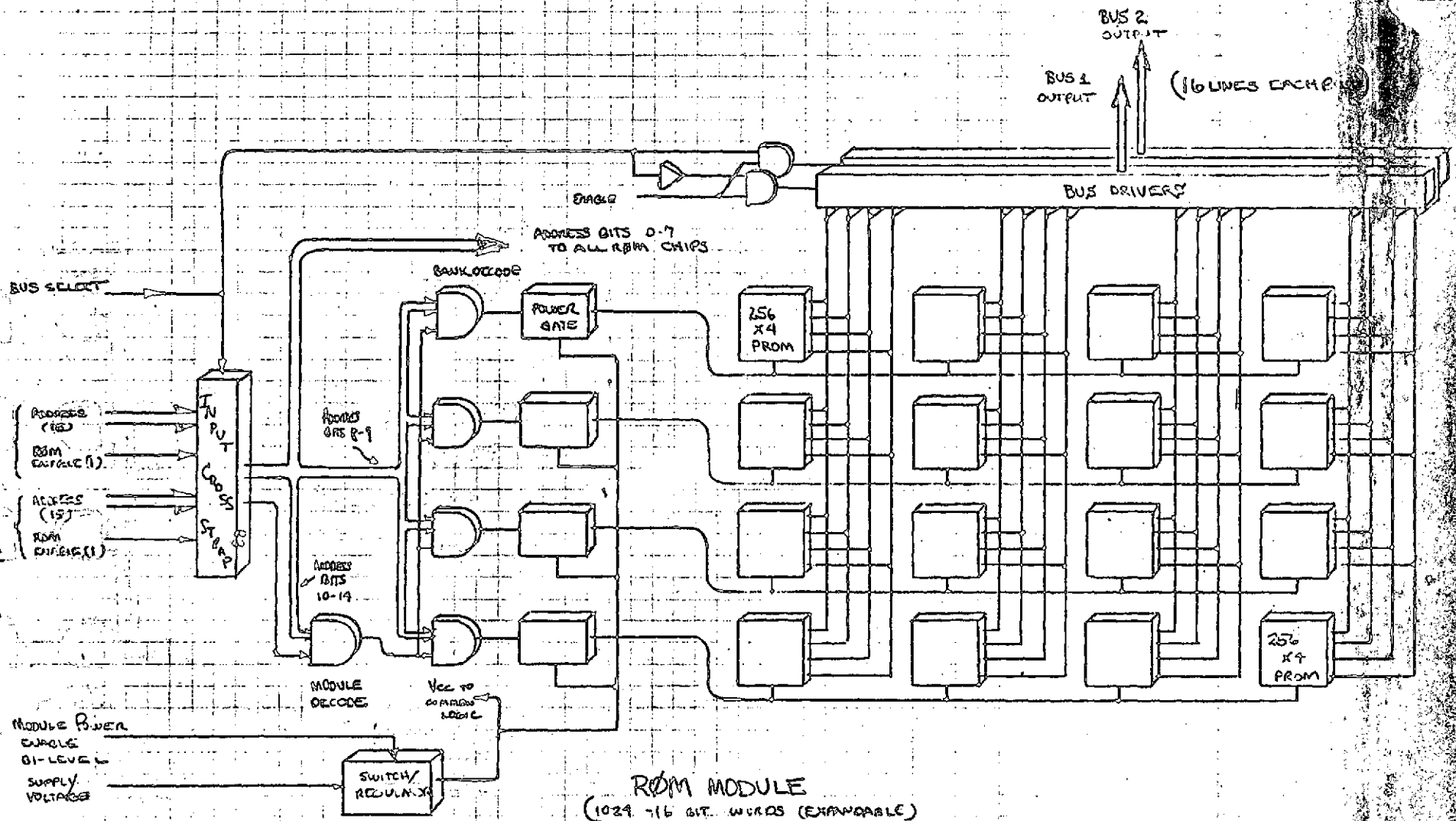
Communication between the system ACU's and memory modules is via a redundant bus system. All address, data, and control lines are compatible with all of the modules cited above. A block diagram showing the interconnection of the system ACU's and memory modules is provided in Figure 4.1. Like the other redundant elements in the DPA, the dual redundant memory bus system is handled as a switchable element. Only one of the buses is assigned an active status at any point in time. The other is held in standby redundancy. A complete description of the signal flow and timing over this interface is provided for in the detailed paragraphs to follow.

#### 4.1 Read Only Memory Units (ROM)

The ROM's provide the user desiring an all solid state system with an element to store the DPA program as well as other fixed parameters and system constants. Like the other elements in the DPA, the ROM's are modularized to facilitate simple expansion and/or replication for redundancy purposes. Each basic ROM module is designed to accommodate between 1024 and 4096 words of storage. Through minor redesign the module can be expanded to a capacity of 8192 words.

All ROM modules communicate with the system ACU's over a common dual redundant bus. Memory data output from the ROM modules is via a 16-bit wide communication interface. A block diagram showing the interconnection between the ROM's and ACU's is provided in Figure 4.1. The number of ROM modules attached to the bus for any particular system application is a function of the required system storage capacity and the system reliability/redundancy requirements. The ACU's are capable of addressing up to 32,768 words of active read-only memory. As currently designed, the RCU is capable of controlling up to four separate ROM modules. Power gating is employed in these devices to minimize the overall system power consumption.





ROM MODULE  
(1024 x 16 BIT. WORDS (EXPANDABLE))

FIGURE 4.2

### Internal Organization

The internal organization of a typical 1024 word module is shown in Figure 4.2. Each 1024 word module contains sixteen 256 x 4 bi-polar field programmable LSI ROM chips. The module is organized into four 256 word x 16-bit banks of storage. Each 256 word bank is mechanized from four chips that are addressed and power switched as a group. Memory data outputs from each of the four banks are wire-or'ed internally and sent to the memory data output bus as a common 16-bit word. Provisions are included in the design to allow the storage capacity of a basic ROM module to be expanded to 2048 or 4096 words. This is accomplished by replicating all elements within the module with the exception of the input bus/cross strap circuits, output bus drivers, and switch/regulator device.

### Addressing and Control

Address and control information is transferred from the ACU's to the ROM's over a common dual redundant bus interface. Cross strap circuits located within each module maintain the system integrity in event of a component or single bus failure. Only one of the buses is assigned an active status by the RCU at any point in time. Receiving elements connected to the active bus are placed in a powered on status. The other receivers are powered off. Each ROM module connected to the bus receives fifteen address lines and a ROM enable/read request signal that originates in the powered on operating ACU.

The most significant bits of the fifteen memory address lines are decoded internally to provide for the first level module and sub-module selection. Connection of memory address signals to the module and sub-module decode gates are made via jumper wires on the multilayer boards based on the module size (word capacity) and the assignment location of the particular module within the memory system. Final address selection is accomplished by the bank decoders which select a specific set of LSI ROM chips as the memory source.

The ROM enable signal serves two distinct functions. First, it is used as a control signal to power gate the selected 256 word ROM bank to the on condition. Secondly, it is used to enable the transmitting tri-state elements of the selected module to the low impedance (active) state. This causes the selected ROM module to gain control of the memory

data output bus for the memory cycle of interest. Sixteen data bits are placed on the memory output data bus during the duration of the ROM enable signal.

#### ROM Cycle Timing

A diagram showing the signal timing relationships for a typical ROM memory cycle is provided in Figure 4.3. The ROM enable line is switched to the "enable" condition concurrently with the transition of the memory address lines to the selected address. This causes the selected memory chips to be power gated to the on condition. A sixteen bit memory data word is placed on the memory data output bus within TBD nanoseconds after the rising edge of the ROM enable signal. It is maintained until the fall of the ROM enable signal at which time the bus is released.

#### RCU ROM Assignment

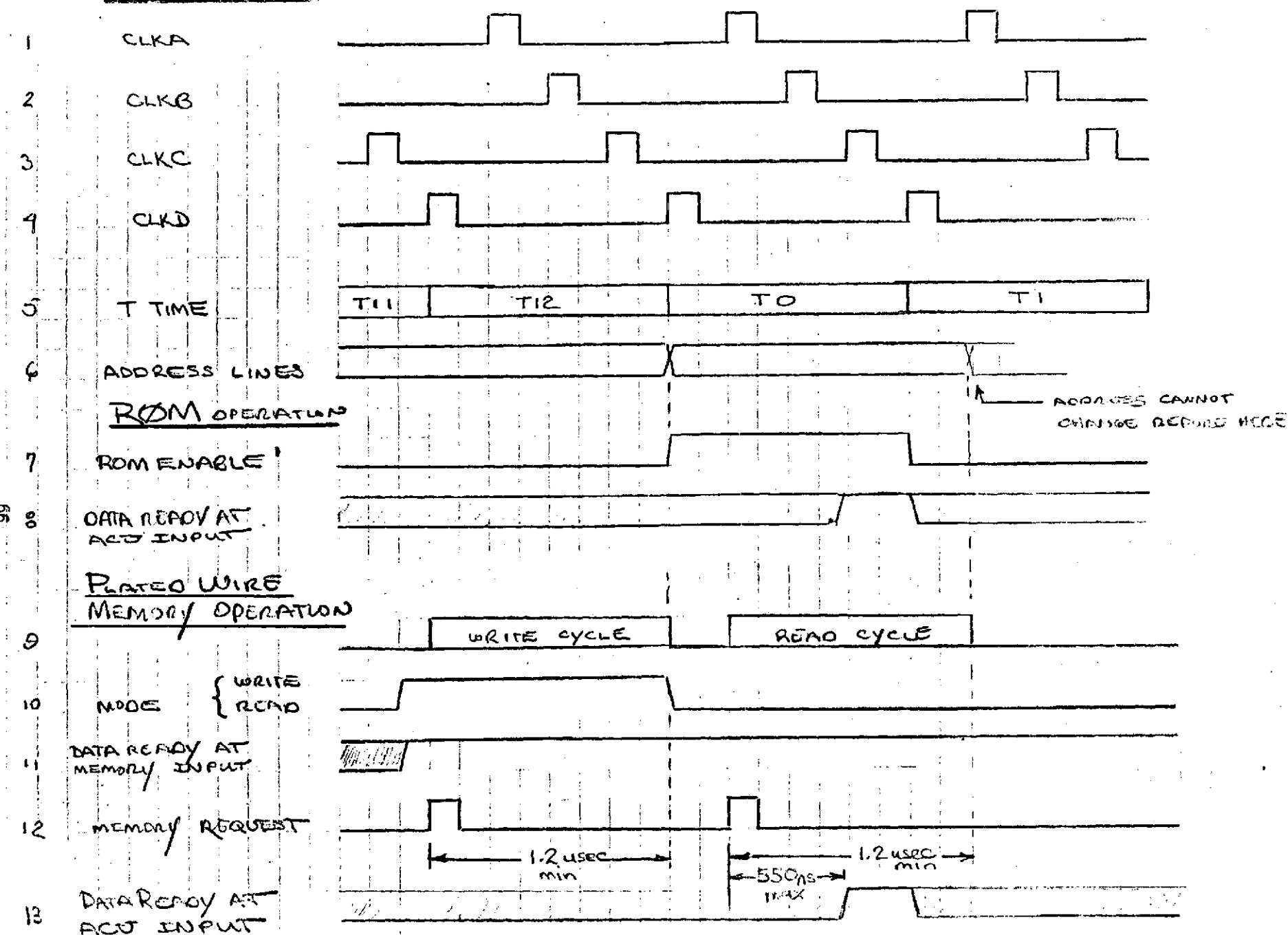
Because of the nature in which the ROM's are utilized in the system, each module must be minimally duplicated for a system requiring standby redundancy. Each module contains program code for a specific segment of the total ROM. RCU assignment of the ROM modules is best explained by an example. Suppose the system requirement is for 8192 words of program memory. Further, suppose that the modules must be divided into 4096 word groups and have single standby redundancy to meet the reliability objectives of the program. Thus, for the particular application, the DPA would contain a total of four 4096 word ROM modules. Two of the modules, A1 and A2, would contain redundant program code for program addresses 0 - 4095. The other two modules, B1 and B2, contain redundant program code for memory locations 4096 - 8191. Thus, the RCU can configure a working program memory using the following module combinations:

A1, B1  
A1, B2  
A2, B1  
A2, B2

RCU assignment is accomplished simply by sending a power enable signal to one of the A elements and one of the B elements. The only requirement is that power enable signals A1 and A2 and B1 and B2 be mutually exclusive. This requirement is handled by the logical mechanization of the RCU.

LIVE No ACU TIMING

← 1.2 usec →



AS SPEC'D AS SHOWN AND PIN CHANGES FOR ROM EDM/PWM/ACU TIMING FIGURE 43

## ROM Built-In Test Equipment (BITE)

Two BITE features are included to facilitate the isolation of failures in a ROM module. Whenever the module is turned on by the RCU, logic power should be supplied to all of the non-power gated elements in the unit. A BITE signal, available from the ROM module, notifies the RCU that the correct voltage is being applied to the circuits within the module. A failure in the switch/regulator device, or an excessive current load due to a failed circuit element, would cause this signal to assume a logical zero condition denoting an inoperable unit.

An additional technique for failure isolation is achievable by means of a special ACU initiated diagnostic test. The final memory word in each ROM module (i.e., address location  $2^n - 1$ , where  $2^n$  is the size of the basic module) is dedicated to a zero check sum function. This cell contains the proper data word such that the arithmetic sum of the entire module contents are equal to zero. A check sum of the ROM resulting in an answer other than zero indicates a failed unit.

### 4.2 Random Access Memory Units (RAM)

The solid state RAM units function as the electrically alterable working scratchpad for the DPA. Additionally, these memories may be used to store program instructions that represent modifications to the ROM program. For a satellite application, this feature allows ground commanded program patches to be input to the system after launch into orbit.

Each RAM module is designed to accommodate either 256 or 512 words of storage. Data communication between the RAM and ACU is via a four-bit byte interface. All RAM modules communicate with the system ACU's over a common dual redundant bus system. A block diagram showing the basic interconnect structure is provided in Figure 4.1.

The number of RAM modules attached to the bus for any particular system application is a function of the required scratchpad storage

capacity and reliability/redundancy requirements. The system ACU's are capable of addressing up to 32K words of random access memory.

As currently designed, the RCU is capable of controlling up to six RAM modules, any four of which may be assigned as an active element during any point in time.

It should be noted that the RAM devices are constructed of volatile storage elements. Removal of system power destroys the contents of these memories; therefore, the use of these devices to store program instructions should be limited to those instances where ground initiated changes are absolutely mandatory.

#### Internal Organization

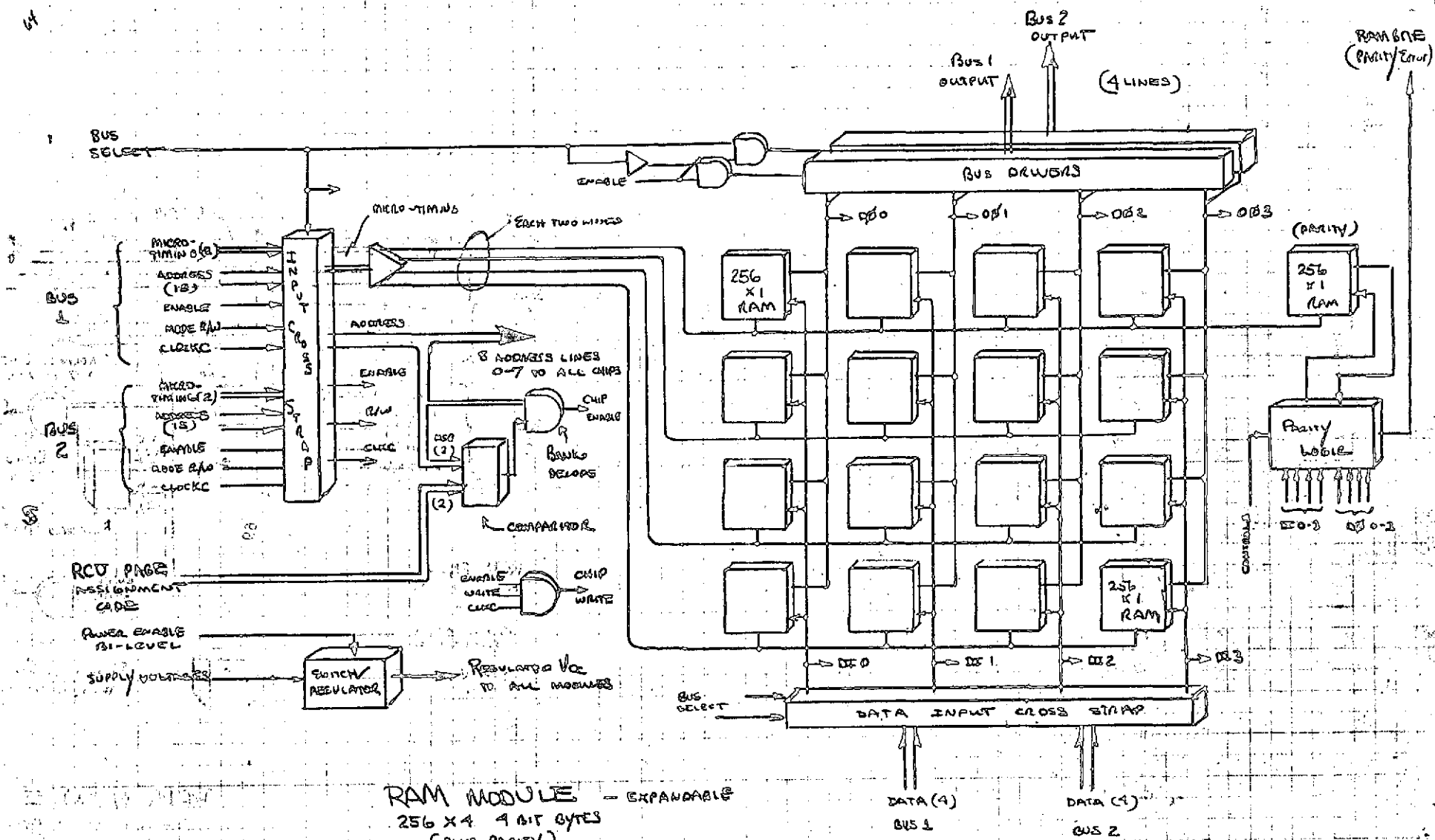
The internal organization of a typical 256 word RAM module is shown in Figure 4.4. A 256 word memory unit is constructed of seventeen 256 x 1 LSI RAM chips. Sixteen of the chips are used for the storage of a basic DPA word and the remaining chip is used for parity. Essentially the memory is organized in a 1024 x 4 array. A sixteen bit ACU operand is written into or read out of the RAM in four 4-bit bytes. The eight least significant bits of the memory address lines are distributed to all chips in the module and function as the word address. Four-bit byte selection is mechanized by the decoding of the two-bit micro-timing signals that are supplied from the operating ACU. The memory parity bit is read from, or written into, the seventeenth chip during byte four time.

Each group of four LSI memory chips comprising a single bit of the four bit wide interface have their collector outputs wire-or'ed internally. Expansion of the memory element to a capacity of 512 words is accomplished by adding seventeen additional RAM chips to the basic module. Each 256 word bank of memory is selected by unique chip enable signal.

#### RCU Page Assignment

The RAM modules are unlike the ROM modules from the standpoint of locatibility within the memory system. Any RAM module can equally well serve as the storage element for any segment of this class of memory.

64



RAM MODULE - EXPANDABLE  
256 x 4 4 BIT BYTES  
(PLUS PARITY)

FIGURE 4.4

This capability permits one to achieve an equivalent system reliability with fewer total standby modules than would be required for the ROM devices. Consider a system containing six 256 word RAM modules where four such modules are required to mechanize a working DPA. There are a total of  $\binom{6}{4} = \frac{6!}{2!4!}$  or 15 ways of configuring the six modules into a working arrangement. Assignment of each RAM to a particular location in the scratchpad memory system is performed by the RCU. For this particular example, the RCU selects and powers on four working modules to make up the system scratchpad. Each of the four modules are sent a unique two-bit page assignment code by the RCU to specify their particular placement in the memory system. Upon being addressed by the ACU, the most significant bits of the memory address field are compared with the page assignment code to perform the module select function. As currently designed, the RCU can service up to six RAM modules, any four of which can be powered on as active storage elements at any particular instance in time.

#### Addressing and Control

Address and control information is transferred from the ACU to the RAM's over a common dual redundant bus interface. Cross strap circuits located within each module maintain the system integrity in event of a component or single bus failure. Only one of the buses is assigned an active status by the RCU at any point in time. Receiving and transmitting elements connected to the active buses are placed in a powered on status. The other bus interface elements are powered off. Each RAM module connected to the bus receives fifteen address lines, two micro-timing lines, a four-bit data byte, CLKC signal, read/write mode signal and a RAM enable signal from the powered on operating ACU. The functional significance of each of these signals are described below.

Address Lines - Fifteen address lines are supplied to the memory system to select the particular memory word to be operated upon. The most significant bits of these lines are compared with the RAM page assignment code and provide an internal module (first level chip select) function. Connections of the proper memory address

signal to the comparator network are made by jumper wires on the multilayer boards based upon the module storage capacity. Second level chip selection to a 256 word group is made internally by the bank decoders. The eight least significant bits of the address field are sent to all chips within the module to select the particular word of interest.

Micro-timing Address Lines - The MTC lines, in essence, provide the two least significant address bits for the RAM memory which is organized in a 1024 x 4 array. These lines are decoded internally to perform final chip selection to the byte level.

Read/Write Mode Control - This signal specifies whether the memory operation is to be a write or read function. A write operation is signified by a true level for the duration of the write cycles. At all other times, it resides in the false or read state.

RAM Enable - This line serves the function of signaling the RAM modules that they are being selected as working memories for this particular cycle. Additionally, it permits the selected RAM to connect to the memory data output bus during transfer of information to the ACU.

CLKC - ACU timing signal CLKC is sent to the RAM modules to assist in two internal functions. During write operations, the CLKC signal is used to generate the write enable signal for the RAM chips. During read operations, it is used as a clock for the parity checking operation.

ACU Data Byte - Four bits of data are supplied to the RAM module during each write operation. Data is made available to the RAM module nanoseconds prior to the rising edge of the CLKC signal.

Memory Data Output - During read operations, the selected RAM module places a four-bit data byte on the memory data output bus nanoseconds after the change in the micro-timing portion of the address. The transmitting bus drivers of the selected module are switched to the low impedance (active) state and gain control of the bus only during the period of time when the RAM

enable signal is present. All other transmitting elements physically connected to the bus are maintained either in the high impedance (non-active) state or are powered off. Note that the four-bit RAM data byte is placed on the least significant four lines of the sixteen line memory data output bus.

#### RAM Cycle Timing

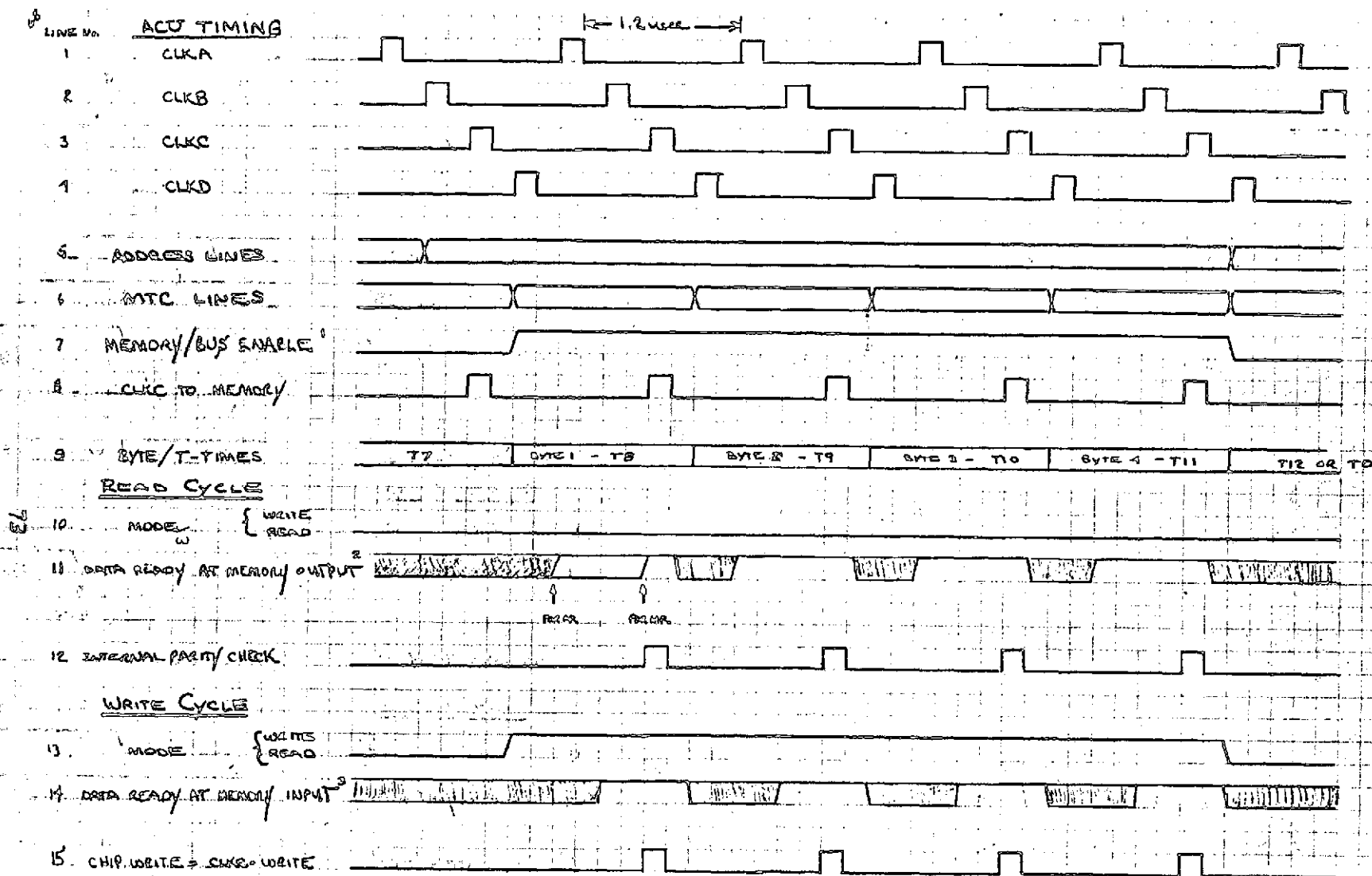
A diagram showing the timing relationships for a typical RAM read or write cycle is provided in Figure 4.5. Information is always transferred into or out of the RAM as four 4-bit bytes. The RAM enable and read/write mode control signals are maintained in the same state during the entire four byte cycle. The byte being operated on is controlled by the state of the micro-timing counter lines.

#### RAM BITE

Two built-in test features are included to facilitate the isolation of failures in a RAM module. A common built-in test signal available from the RAM is used to notify the external hardware (RCU) of either a power failure internal to the module or of a memory parity check failure.

When the module is turned on by the RCU, logic power should be applied to all of the circuit elements in the unit. If the correct voltages are not being applied to the logic in the module, the BITE signal assumes a logical zero condition denoting an inoperable unit. This type of failure could be caused by a faulty switch/regulator device or due to an excessive current drain of a failed circuit element.

Memory parity is checked each time a sixteen bit DPA word is read from memory. This operation is performed at the end of the byte time four. Whenever this parity check fails, the BITE signal assumes a logical zero condition. This signal is maintained in the fault condition until the next memory cycle is initiated at which time the parity error flip-flop in the RAM module is cleared. The momentary parity error indication from the RAM BITE lines are held in latches located in the RCU. For a system not having an RCU, this error indication would be sent to a latch located in the telemetry system.



- NOTES:
1. POWER GATE AND BUS ENABLE LOAD GATE FOR P.W. MEMORY
  2. DATA NEEDED AT ACU INPUT
  3. DATA AVAILABLE AT ACU OUTPUT

RANDOM ACCESS MEMORY / ACU TIMING

FIGURE 4.5

### 4.3 Alterable Program Memory Units (APM)

The APM units are provided for use during the system testing and program debug stages of a developmental program. They serve the same function as the ROM units, that being the storage of the system program and constants. Unlike the ROM's, these units are constructed of electrically alterable volatile storage elements that may be reprogrammed or changed as desired. Programs are loaded into the APM's via a paper tape reader. Program patches may be made to the contents of the APM via controls on the programmer/maintenance console. The APM and ROM modules are physically interchangeable.

The APM module consists of a 2048 word (or 1024 word) by 16-bit dynamic random access memory. The memory element used is a 1024 word by 1-bit MOS-LSI circuit. The memory elements are considered dynamic due to the fact that each bit of storage consists of a capacitor which is subject to information loss due to leakage currents. Therefore, the memory must be periodically refreshed.

Each time the APM module is powered on, a program must be loaded from an external source. This is accomplished by using the same 4-bit data bus that is used to write into the random access memory. The 16-bit word is loaded in four 4-bit bytes under control from the ACU. The two MTC lines are decoded to determine which byte is being loaded into memory. The load timing diagram is shown by Figure 4.6. The same write request signal that goes to the RAM is given in order to enable a load operation, except that the ROM enable signal goes true rather than the RAM enable signal. The information is actually written into the memory during the write strobe. The address lines must be set up 500 nsec prior to the write strobe in order to insure that all internal decoding is accomplished to prevent any data from being written into an incorrect address.

Refreshing of the memory can be accomplished in several ways. Refreshing of a row of words occurs whenever there is not a read cycle. The selected row is determined by the five least significant bits of address. Therefore,

in a 1024 word, 16-bit memory, there are a total of 16,384 bits consisting of 32 rows with 512 bits per row. Each refresh operation refreshes one row of 512 bits. Thirty-two refresh operations are required to refresh the memory at least every 2 msec. During a load cycle, the row to be refreshed is determined by the five least significant bits of the address bus. Otherwise, the row is selected by a 5-bit refresh counter. This counter is incremented only at the completion of a row refresh operation which was under control of the counter. If, for any reason, the memory were to be loaded continuously, and in sequence, a new row would be refreshed each time a complete, 4 byte word is written in. This means a row would be refreshed every 4.8  $\mu$ sec and that all 32 rows would be refreshed every 153.6  $\mu$ sec. Ordinarily, loading will be from the paper tape, which is relatively slow. This means that there will be plenty of time between words for many complete refresh operations to occur, since only 38.4  $\mu$ sec are required to refresh 32 rows under control of the refresh counter. During read operations, only 38.4  $\mu$ sec is needed out of 2 msec for refreshing.

Each APM module also decodes all 15 bits of the address bus, permitting up to 16 modules to be used for a total of 32,760 words. The five most significant bits of address are decoded along with the ROM enable to enable the module for a write or read operation. The ten least significant bits of address are connected to all of the memory elements and are internally decoded in each chip. The output of each memory element consists of 1-bit driven by an open collector circuit. This, in turn, is connected to a sense amplifier which converts the signal to a TTL level for driving the bus drivers. Figure 4.7 is a timing diagram for a read operation. The timing criteria is the same as for the ROM in that the valid data must be on the bus 300 nsec prior to the trailing edge of CLKC.

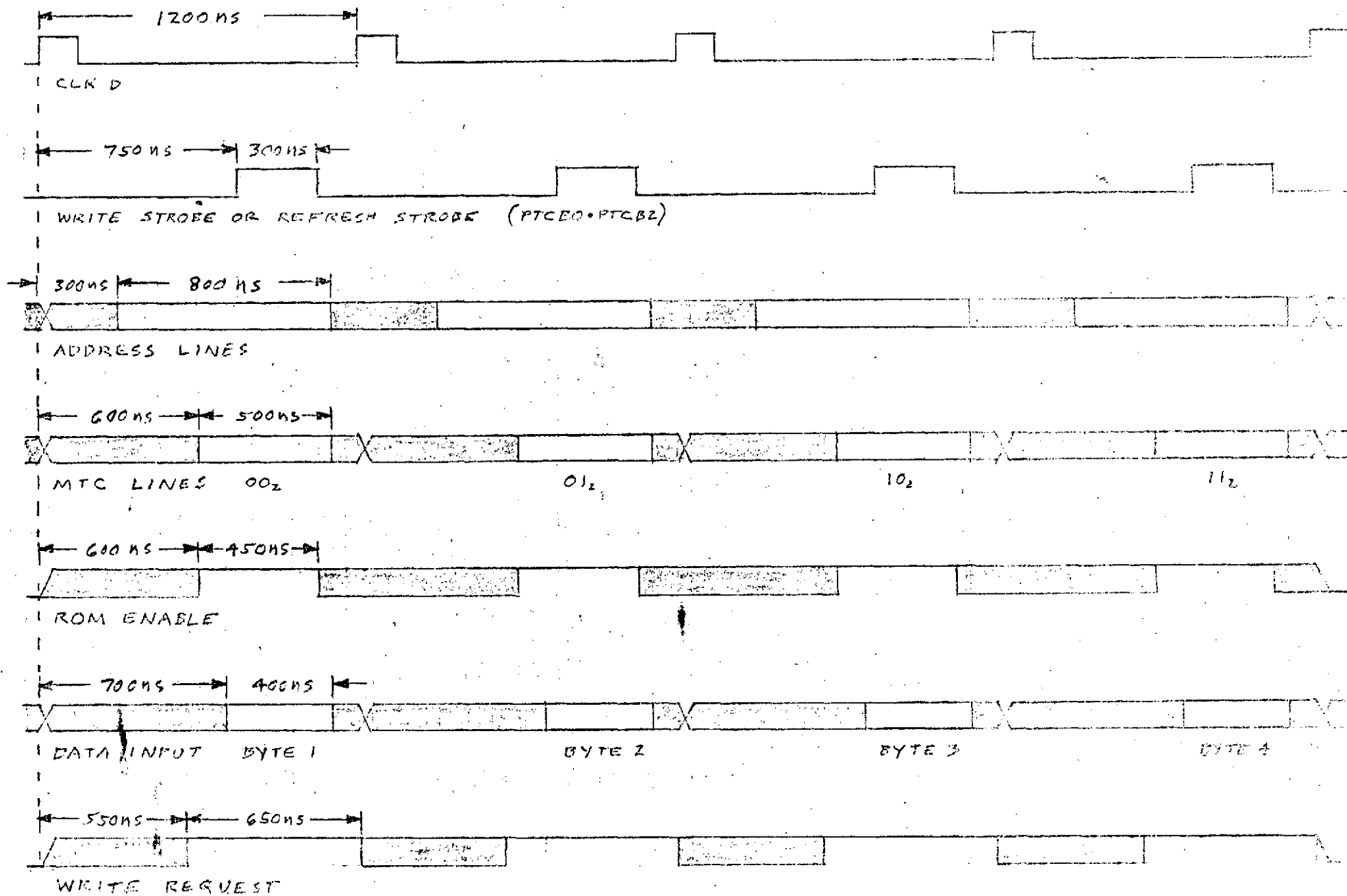


FIGURE 4.6  
AFM LOAD TIMING DIAGRAM

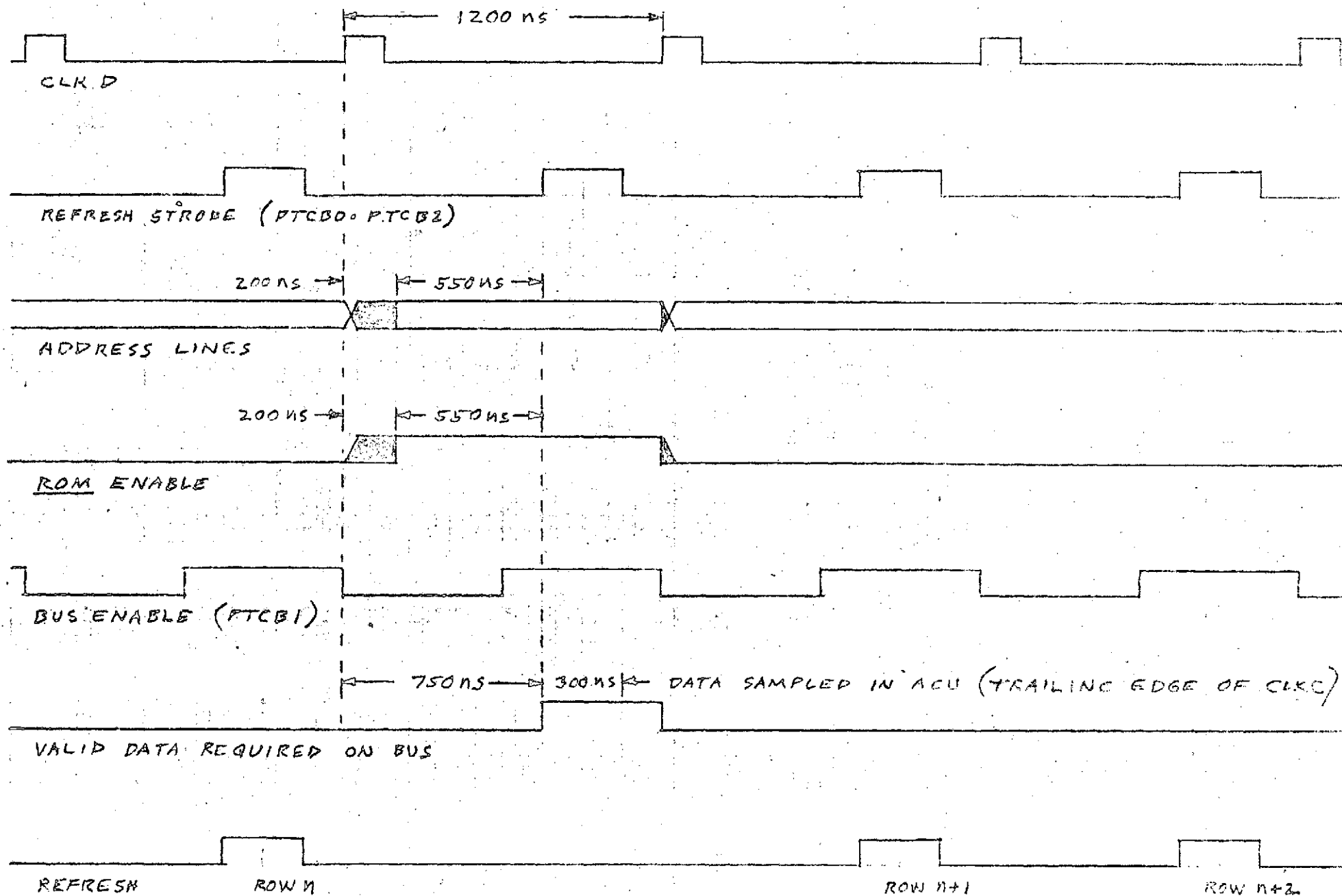


FIGURE 4.7  
APM READ TIMING DIAGRAM

#### 4.4 Plated Wire Memory Units (PWM)

The PWM's provide the user with a memory system which may be used as either a read-only or a read-write memory and thus may replace any of the other memory types for either ground operation or spacecraft use. The PWM's are interchangeable with the other memory types in both a physical, functional and electrical sense. The PWM's are modularized to allow expansion for memory size and/or replication for redundancy purposes.

Each PWM module is designed to accommodate either 1024 or 2048 words of storage. In 256 word increments, the PWM is designed to either allow or prevent write into memory from the DPA. This is accommodated by external jumpers in a connector. The same connector is used for memory load, which can occur into any cells.

The communication with the system ACU's occurs over a common dual redundant bus, as with the other memory types. All interfaces are also identical to those for the other memories. The PWM contains the features of both a RAM (page number addressing, parity check write/read/compare and of a ROM.

Note that (unlike a RAM) the memory storage in the PWM is non-volatile. This means that removal and reinstatement of system power will not cause either program or read-write scratch pad memory to be lost. This feature can be used to advantage in many systems. On the other hand, since the program memory is stored as magnetic domain states and not as physically altered conditions (as in a ROM), severe nuclear or electro-magnetic pulse events could alter the program words. These features must be considered in the application of the PWM, along with the power, reliability and other trade-offs.

## 5.0 INPUT/OUTPUT UNIT (IOU)

The input/output unit (IOU) functions to provide the interfaces between the DPA and all peripheral equipment (external world). The only exceptions are the direct interfaces with the RCU (command, telemetry and clock source), and the electrical power, which interfaces with all units of the DPA directly.

The IOU is designed so that all input/output functions are under central processor (program) control. There are no program interrupts through the IOU. All functions must wait for recognition by the software.

Figure 5.1 is an overall block diagram of the IOU showing all of the functional blocks and their interrelationships. All input and output interfaces are modular so that the IOU may be relatively optimally suited to its particular application.

The maximum capabilities have been selected relative to estimated needs for spacecraft attitude control subsystems.

### 5.1 Functional Requirements

The functional requirements of the various IOU blocks are specified by block in this section.

#### 5.1.1 Control

Figure 5.2 is a block diagram of the IOU control logic. The control generates clocks, instructions, and addresses for the internal IOU functions. Figure 5.3 is a timing diagram for the control functions.

The master clocks, PTCB0-PTCB3, are supplied from the RCU. These clocks have a period of 1.2  $\mu$ sec, which is equal to one T time in the ACU. These clocks are decoded to form the CLKC, CLKCW, and CLKDW functions. The wider clock pulse widths (CLKCW and CLKDW) are necessary for clocking 54L/74L series flip-flops. CLKC is divided by a factor of 8 to form the shift clocks for the peripheral serial interfaces. These clocks have a period of 9.6  $\mu$ sec, or

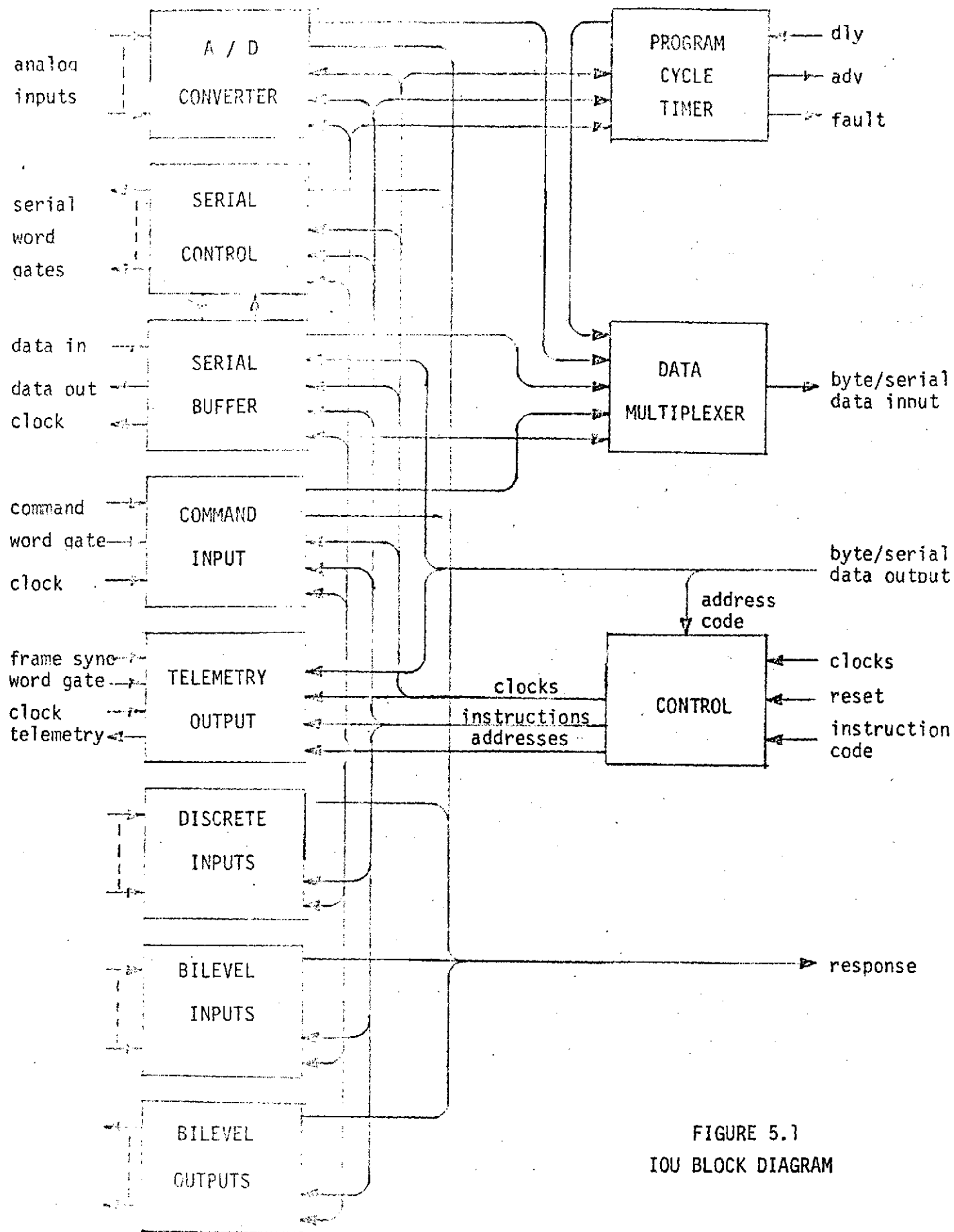


FIGURE 5.1  
IOU BLOCK DIAGRAM

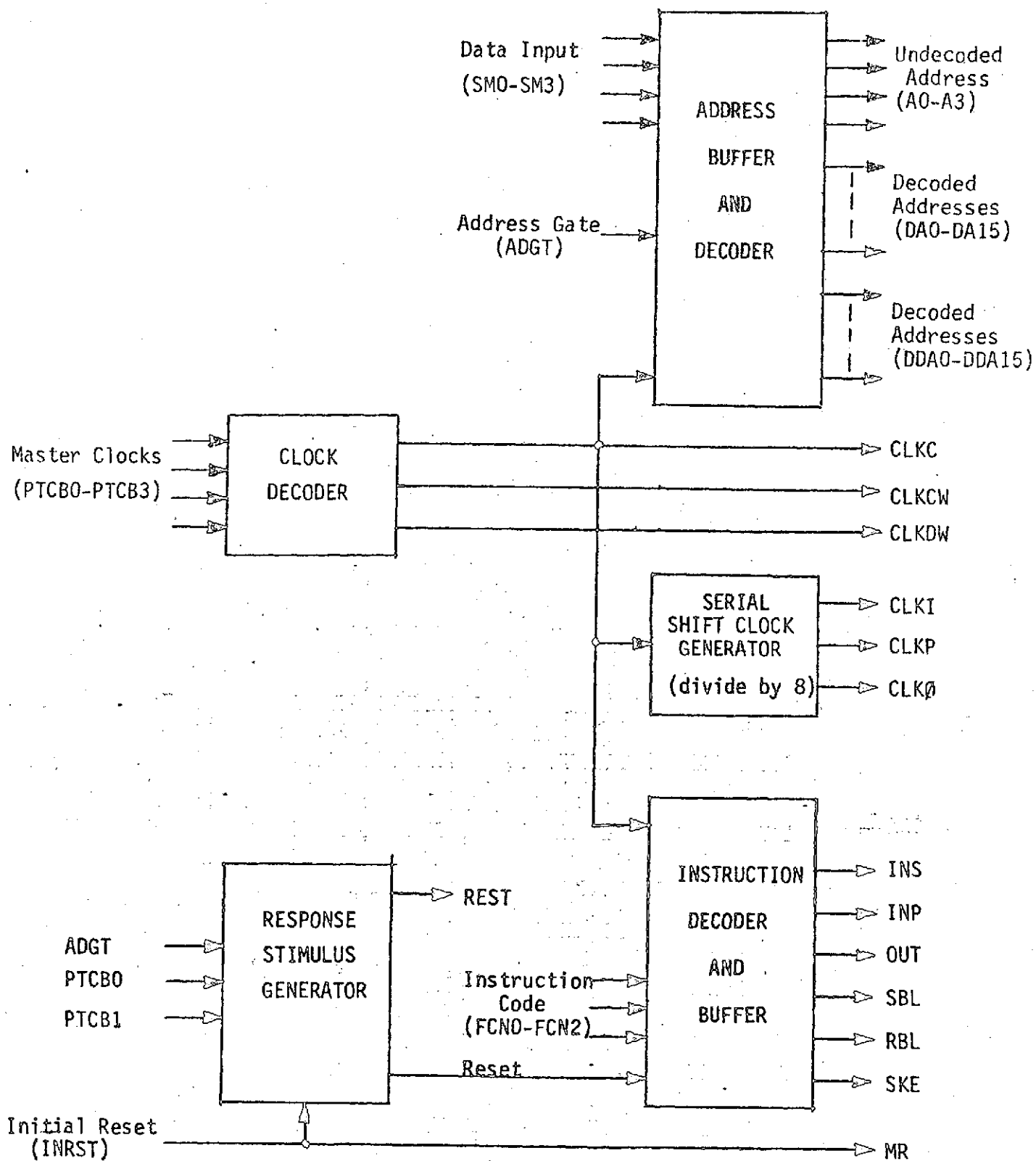


FIGURE 5.2  
CONTROL BLOCK DIAGRAM

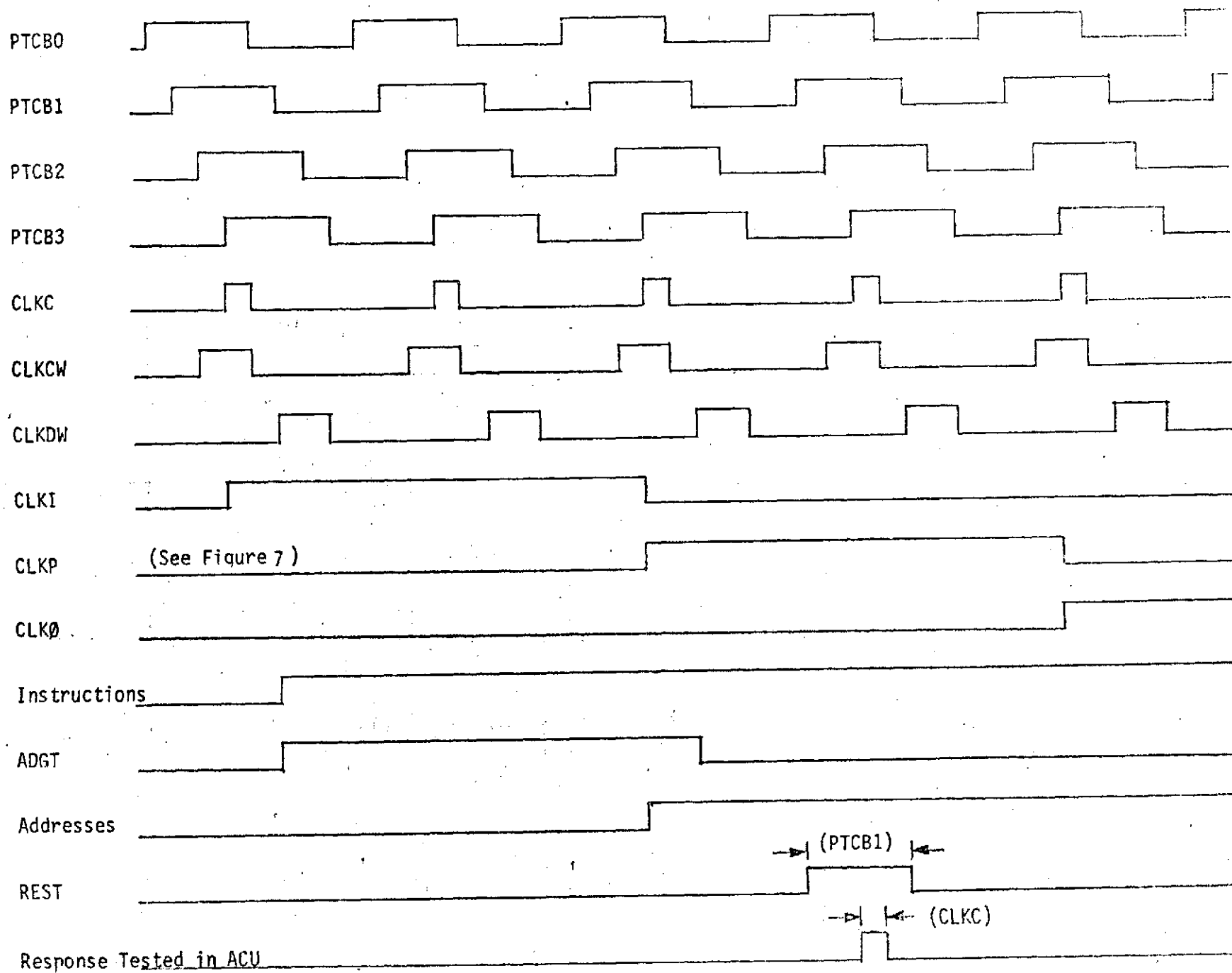


FIGURE 5.3 CONTROL TIMING DIAGRAM

a frequency of approximately 104 KHz. CLKP is the clock that drives all peripheral serial transfers, while CLKI is used in the IOU for serial input, and CLKØ is used in the IOU for serial output.

An 8-bit address is transferred to the IOU from the ACU over the internal data bus (SM0-SM3) in two 4-bit bytes. A separate address gate (ADGT) from the ACU defines the presence of a valid address on the bus and is also used to initialize the execution of instructions in the IOU. This is done by having the address gate generate a response stimulus (REST) 600 nsec after the trailing edge of the address gate. The response stimulus is gated with the appropriate instructions and decoded addressee to execute the instructions and also enable any response that may be sent back to the ACU. The four most significant bits of address (A4-A5) are decoded into 16 direct addresses (DA0-DA15), while the four least significant bits (A0-A4) are also decoded into 16 direct addresses (DDA0-DDA15). By ANDing the DAX addresses with the DDAX addresses, any one of 256 addresses may be selected. The address assignments and associated functions are summarized in Table 5.1.

The instructions are decoded from three lines (FCN0-FCN2) from the ACU. Their functions are defined in section 3.3.4, and also in the descriptions of the remaining IOU functions.

### 5.1.2 A/D Converter

Figure 5.4 is a block diagram of the A/D Converter, while Figure 5.5 is a timing diagram of the basic timing functions.

The converter is a successive approximation type with a maximum of 32 semi-differential analog inputs. These inputs are selected by IOU address. The conversion resolution shall be 12 bits and accuracy shall be  $\pm 0.1\%$ . Conversion time is 140-150  $\mu$ sec. Signal range is  $\pm 5.0$  v with negative values represented by two's complement notation. Overvoltage range is  $\pm 15$  v.

The conversion process is initiated when one of the 32 analog inputs is addressed and a set bilevel (SBL) instruction is given. If the converter is not busy, a response is sent to the ACU, the address is stored, and the

Table 5.1 - IOU Addresses

<u>Address</u>	<u>Function</u>	<u>Command</u>	<u>Response</u>
MSB    LSB			
00xxxxxx	80 Bilevel Outputs	SBL/RBL	Yes
0100xxxx			
0101xxxx	16 Serial Outputs	SKE/OUT	Yes
011xxxxx	32 Serial Inputs	SBL/INP	Yes
10xxxxxx	64 Bilevel Inputs	SKE	Yes
110xxxxx	32 Discrete Inputs	SKE/OUT/INS	Yes/Yes/No
111xxxxx	32 Analog Inputs	SBL/INP	Yes
<u>Dedicated Discretes</u>			
11000000	Command Input	INS	No
11000001	Telemetry Maj. F.S.	SKE	Yes
11000010	Telemetry Min. F.S.	SKE	Yes
11000011	Telemetry Buf. Status	SKE	Yes
	Telemetry Load	SBL/RBL	Yes
<u>Dedicated Bilevels</u>			
00000000	Program Counter		
	Window Test	SKE	Yes
	Sync Bilevel	SBL	Yes
0000xxxx	TOD Input	INP	No
00000001	ACU Detected Fault	SBL	Yes
00000010	Peripheral Bus Select	SBL/RBL	Yes
	0 = Bus A 1 = Bus B (External jumper)		

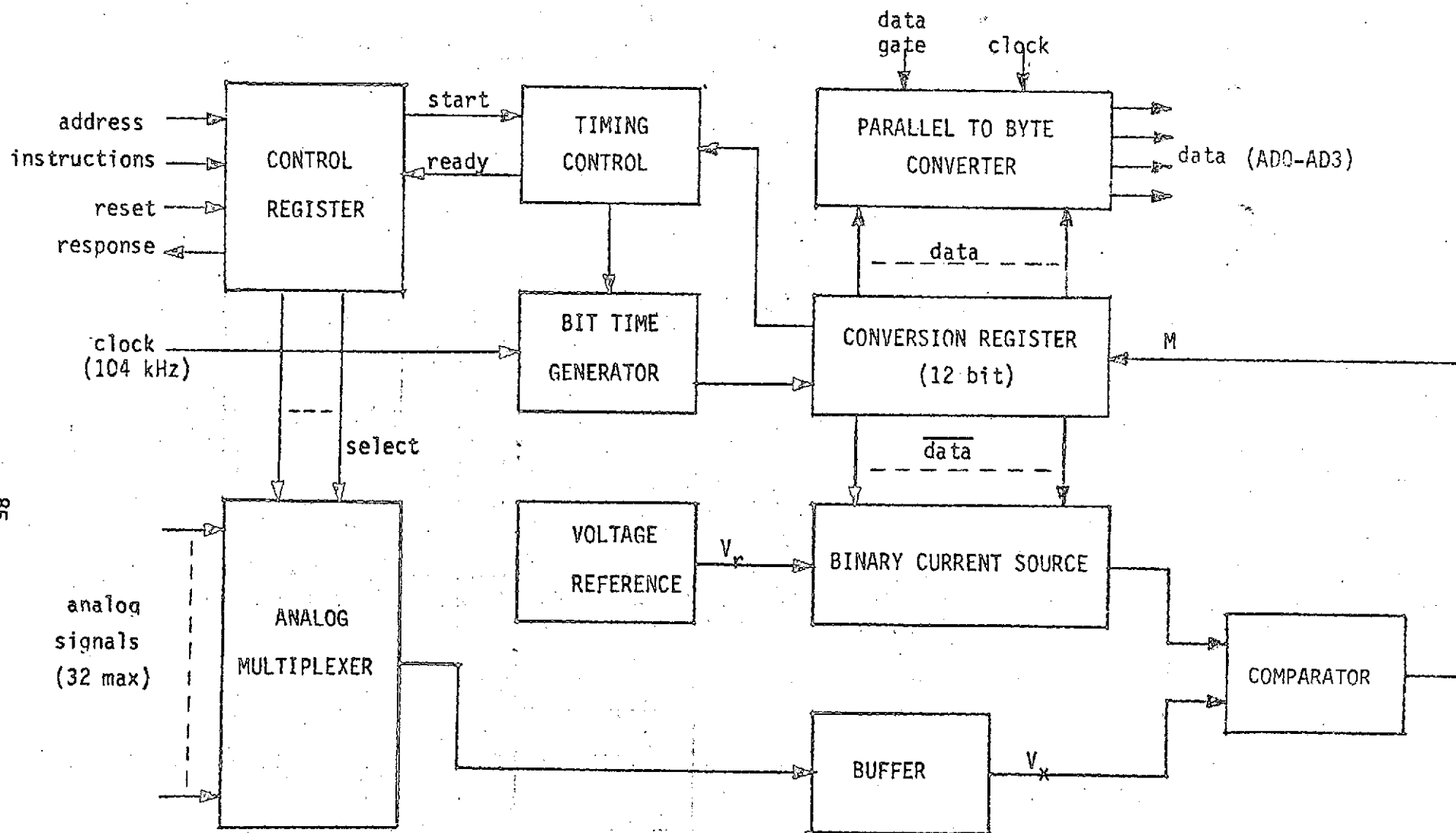


FIGURE 5.4  
A-D CONVERTER  
BLOCK DIAGRAM

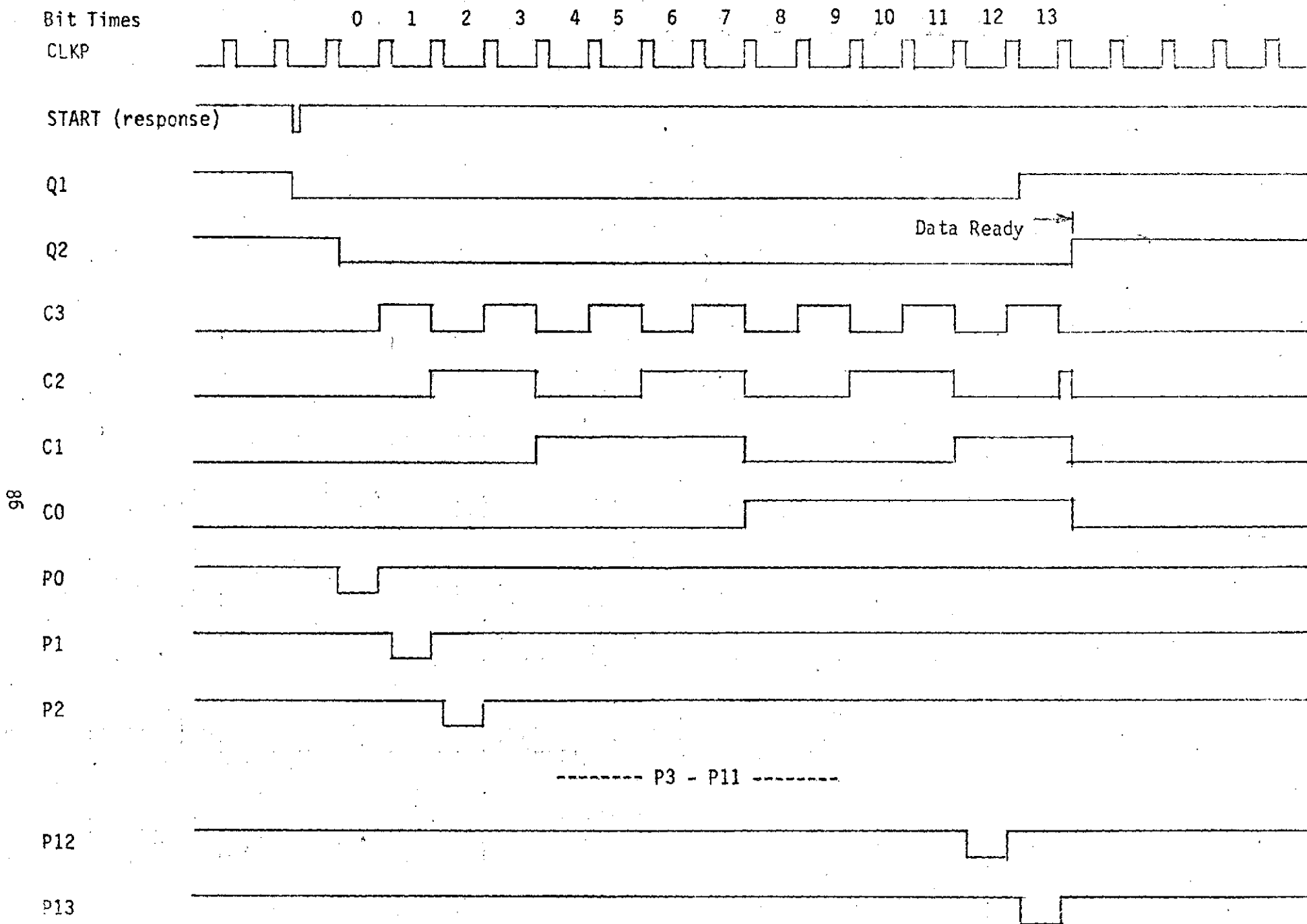


FIGURE 5.5 A-D CONVERTER TIMING DIAGRAM

converter logic is initialized. During the conversion process, the IOU is free for performing other functions. When the conversion process is completed and the ACU is ready to accept the data, an INP (data input) instruction is given along with the address of any analog channel. If the conversion is completed and the data ready, a response is given to the ACU and the data is shifted to the ACU in 4-bit bytes, least significant byte first. The 12-bit word is left justified to form a 16-bit word with zeros filling the least significant end of the word. The byte data transfer is shown in Figure 5.8.

### 5.1.3 Serial Data

Figure 5.6 is a block diagram of the serial data control function. Figure 5.7 is an overall timing diagram for serial data transfers, while Figure 5.8 is an expansion of Figure 5.7 showing the byte data transfer between the ACU and IOU. The serial interface between the IOU and the peripheral equipment is defined in Section 5.2.

A dual, redundant bus system is used between the IOU and all peripherals for serial data transfer and the shift clock. One common bus is used for all serial input data with the appropriate bus driver being power gated on by the selected serial input bilevel gate. A separate common bus is used for all of the serial output data. The data is routed to the peripheral which is selected by the appropriate serial output bilevel gate. All serial data transfers are 16-bit words, least significant bit first, with a clock rate of 104.167KHz. The serial bilevel gates are used to address the peripherals and also act as word gates which are 16 clock periods in duration.

A serial output is initiated by addressing one of the 16 serial output bilevel gates (SBL00-SBL015), and giving an output (OUT) instruction. If the serial buffer is not busy, a response is sent to the ACU, and the address is stored in a buffer. The ACU next loads the byte/serial buffer in four 4-bit bytes. This is controlled by a data gate (DTGT) from the ACU which is 4.8  $\mu$ sec wide (4T times). After the data gate goes false, the OUT instruction goes false. This causes the serial bit counter to begin counting at the 104.167 KHz

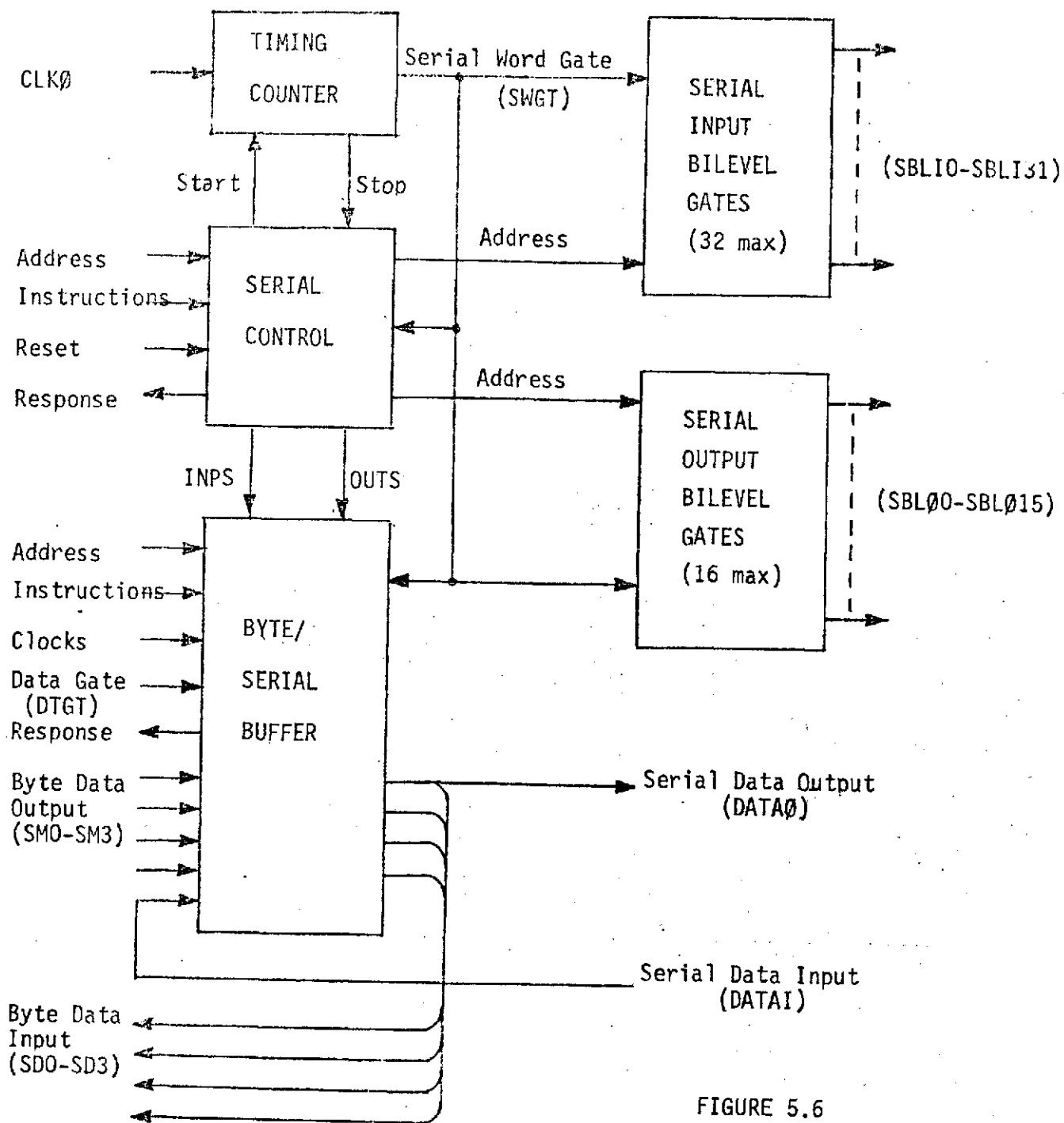


FIGURE 5.6  
SERIAL DATA  
BLOCK DIAGRAM

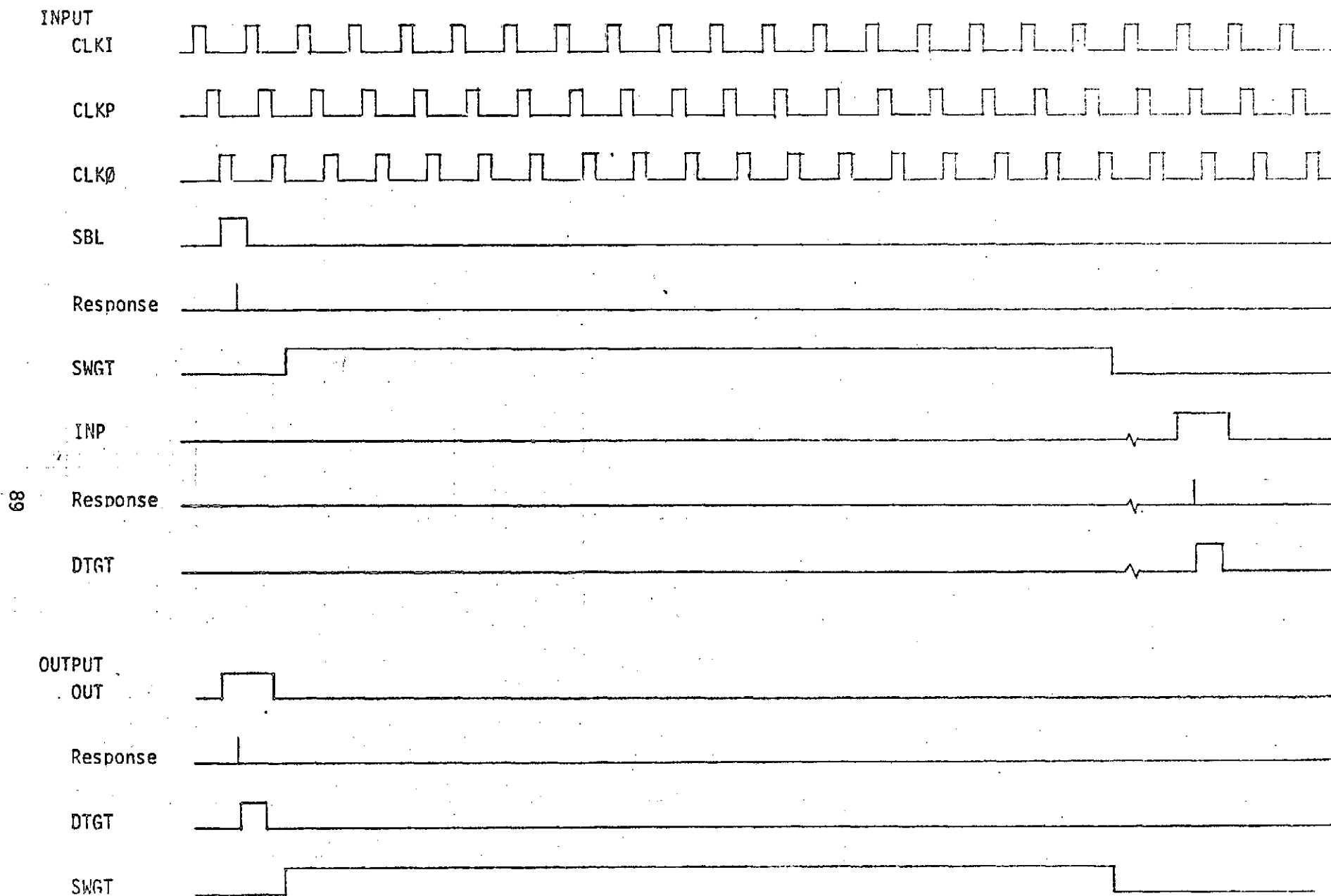


FIGURE 5.7 PERIPHERAL SERIAL DATA TRANSFER TIMING DIAGRAM

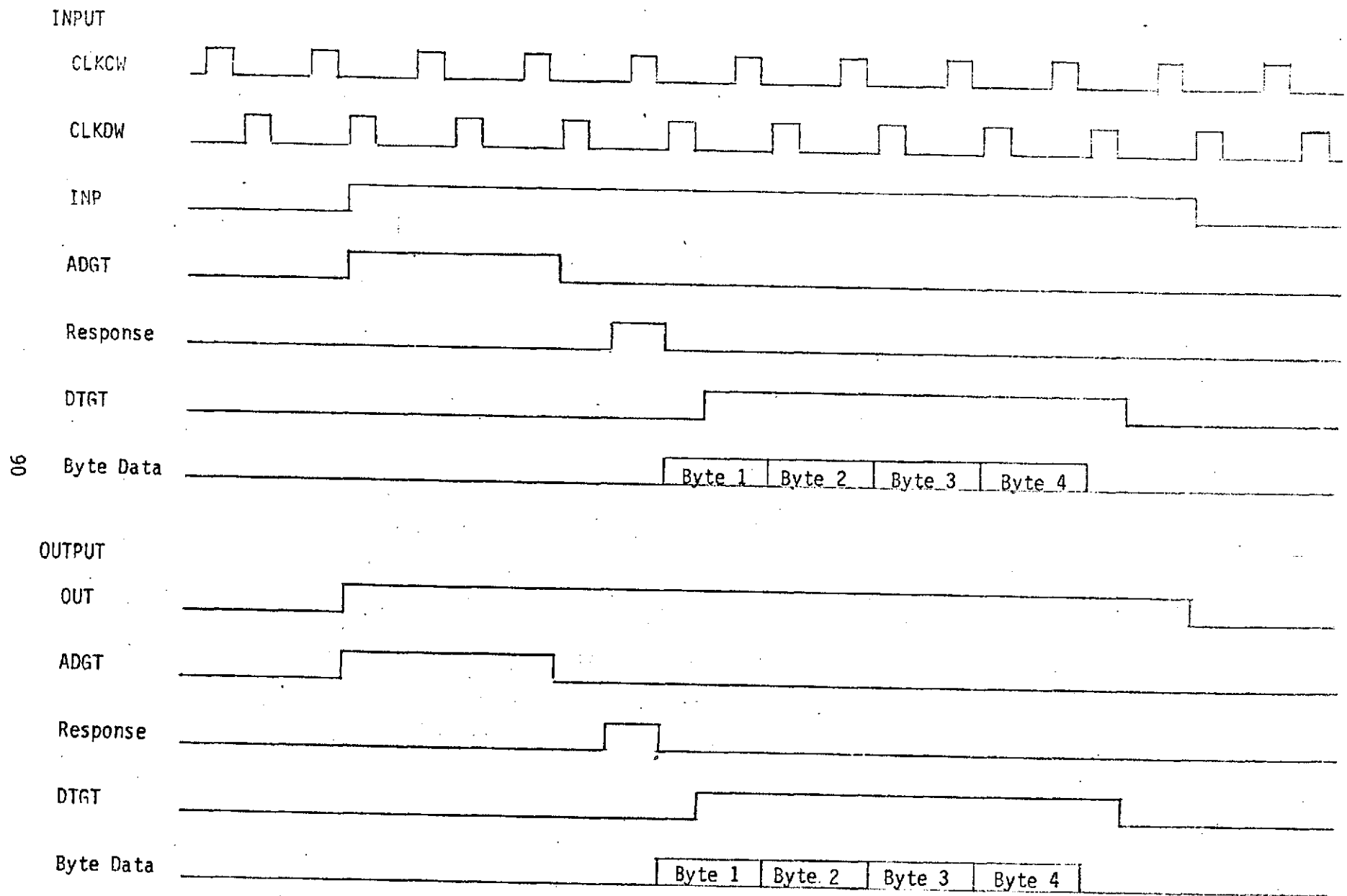


FIGURE 5.8 INTERNAL BYTE DATA TRANSFER TIMING DIAGRAM  
(Expanded from Figure 5.7)

clock rate. The counter enables a serial word gate which in turn enables the addressed serial output bilevel gate and also switches the byte/serial buffer from byte to serial operation, and gates the serial output clock (CLK $\emptyset$ ) to the buffer. During this time, the IOU is free to perform other functions not associated with serial data transfers. The status of the serial buffer may be tested at any time by addressing any serial output channel and giving a skip on external (SKE) instruction. If the buffer is not busy, a response is given to the ACU.

A serial input is initiated by addressing one of the 32 serial input bilevel gates (SBLI0-SBLI31), and giving a set bilevel (SBL) instruction. If the serial buffer is not busy, a response is sent to the ACU, the address is stored in a buffer, and the serial bit counter begins counting. A serial word gate is generated by the counter which enables the addressed serial input word gate, switches the byte/serial buffer to serial operation, and gates the serial input clock (CLKI) to the buffer. During this time, the IOU is free to perform other functions not associated with serial data transfers. When the ACU is ready to accept the input data from the byte/serial buffer, it is tested to see if the buffer is filled by addressing the first serial bilevel gate (SLI0, address 01100000<sub>2</sub>), and giving an input (INP) instruction. If the buffer is filled and ready, a response is sent to the ACU. The ACU then sends the 4.8  $\mu$ sec wide data gate (DTGT) and unloads the buffer in four 4-bit bytes. The byte data is routed through the data input multiplexer which selects the data from the byte/serial buffer and routes it to the IOU to ACU data bus (IOU0-IOU3).

#### 5.1.4 Telemetry

Figure 5.9 is a block diagram of the telemetry output function. Figures 5.10, 5.11, 5.12, and 5.13 are timing diagrams for four different options of telemetry applications.

The timing diagram in Figure 5.10 shows a case where the telemetry data (TMDI $\emptyset$ ) is a continuous, serial bitstream. The word gate (WDGT) is held in the one state at all times. Whenever the telemetry empties one of the 256-bit, random access buffers, the terminal count (TC1) from the address counter

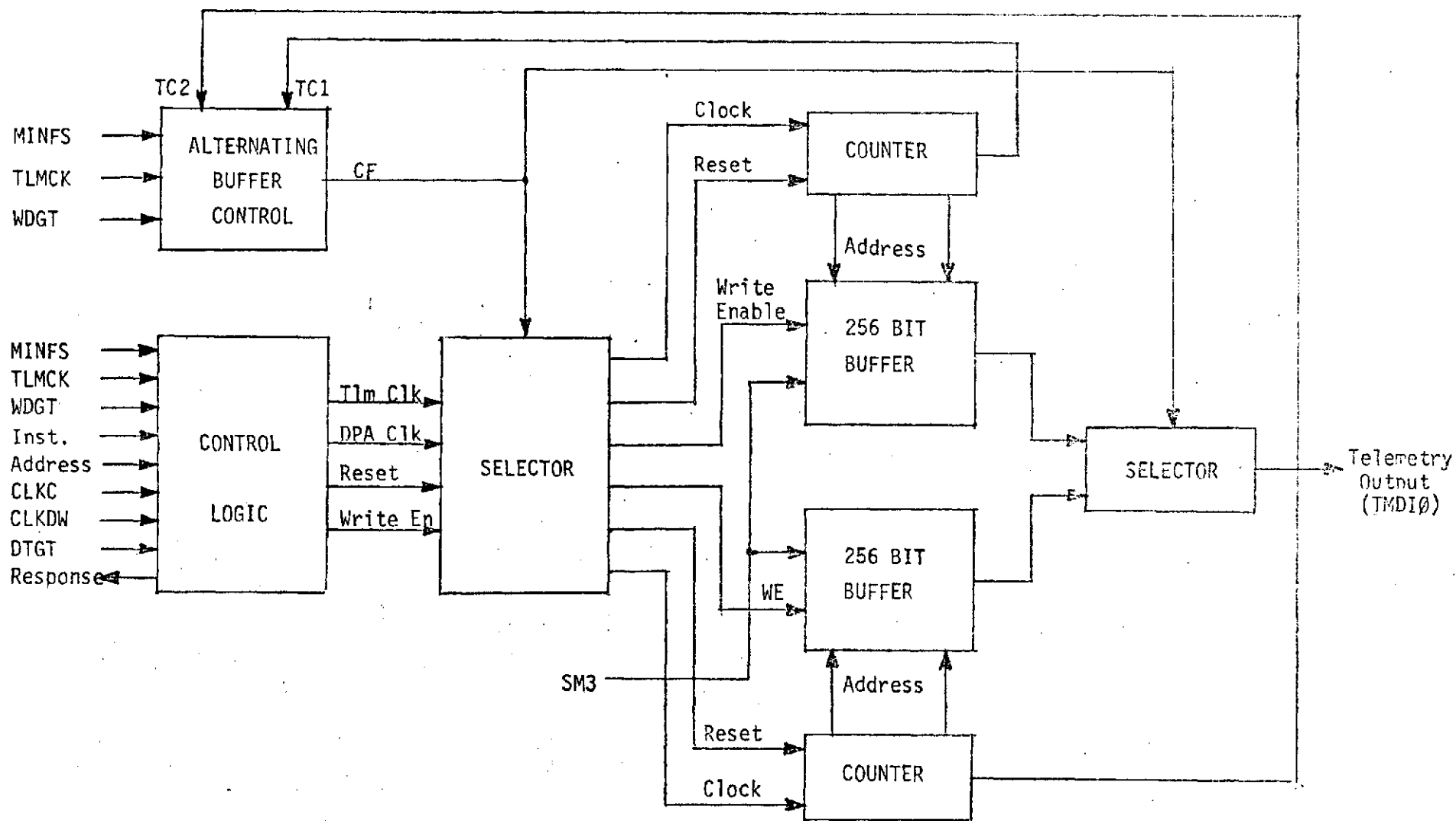
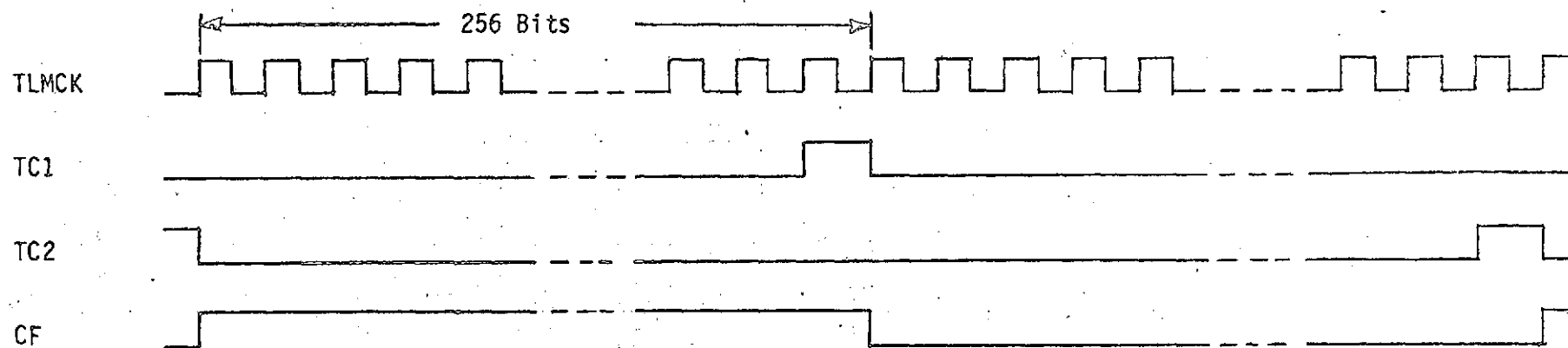


FIGURE 5.9

TELEMETRY BLOCK DIAGRAM

# TELEMETRY OUTPUT



# BUFFER LOAD

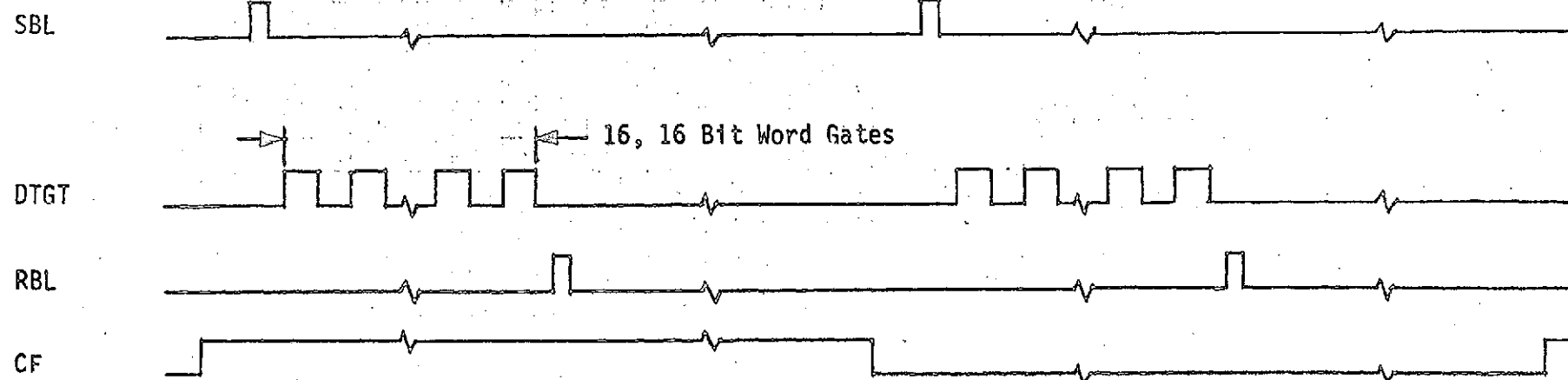
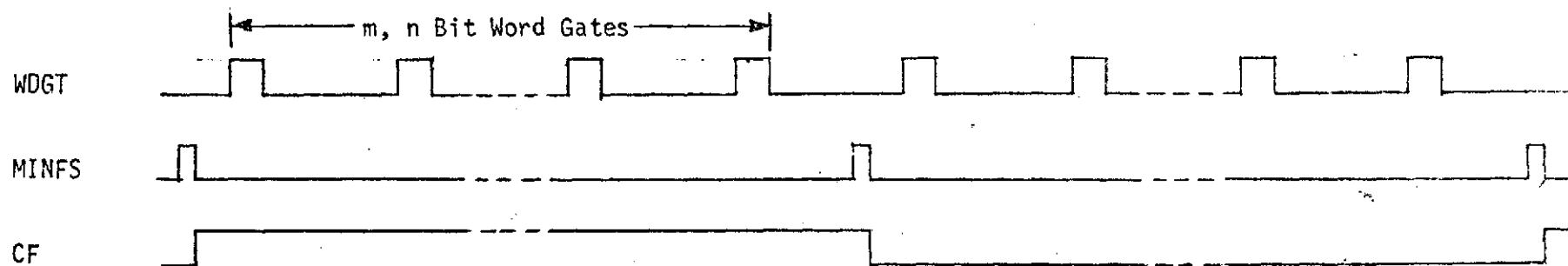


FIGURE 5.10

TELEMETRY TIMING DIAGRAM  
COMSAT BREADBOARD

TELEMETRY OUTPUT



BUFFER LOAD

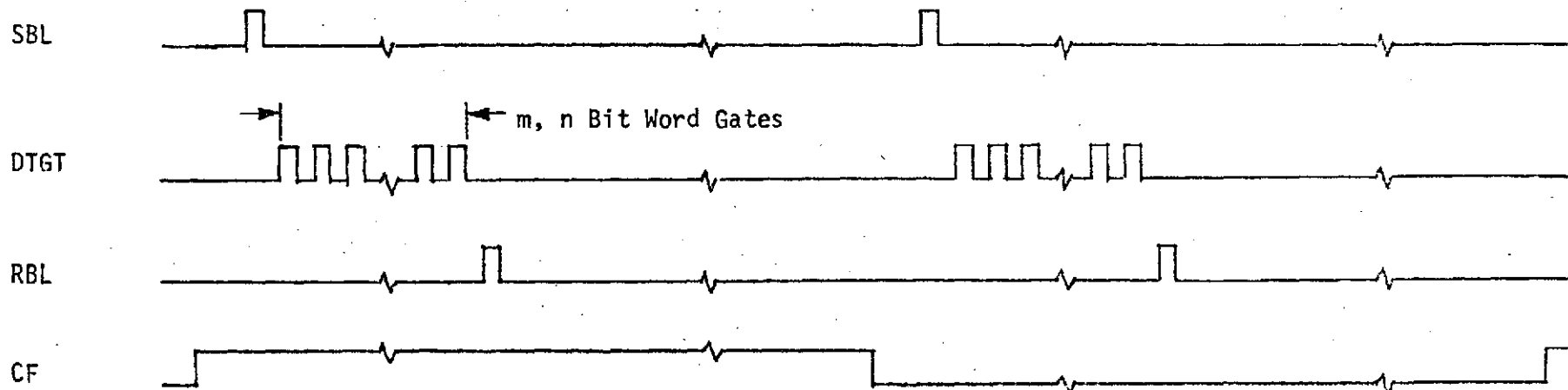
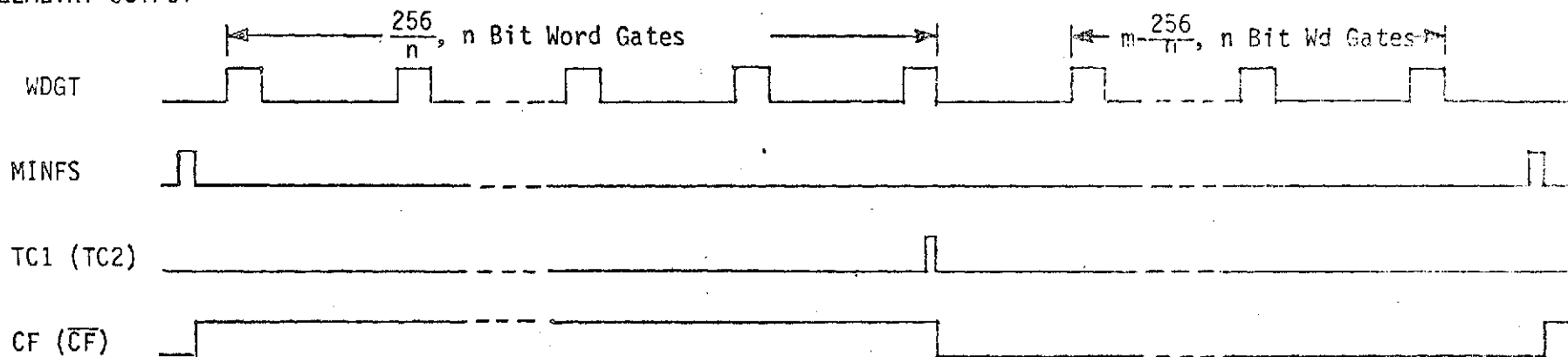


FIGURE 5.11

TELEMETRY TIMING DIAGRAM  
GENERAL CASE

TELEMETRY OUTPUT



BUFFER LOAD

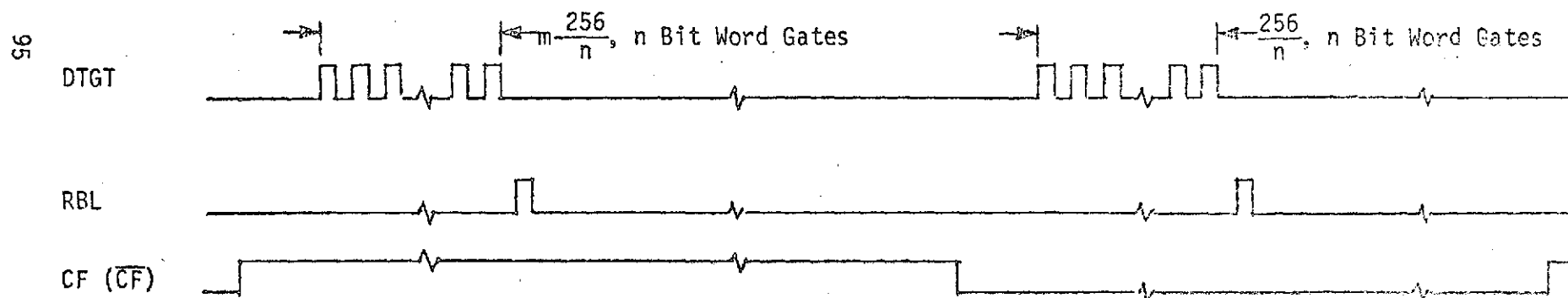
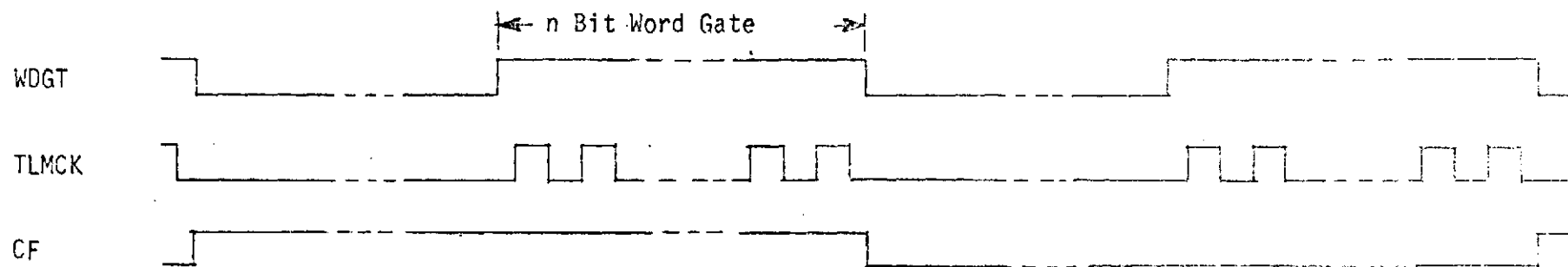


FIGURE 5.12  
TELEMETRY TIMING DIAGRAM  
SPECIAL CASE  
 $m \times n > 256$

# TELEMETRY OUTPUT



## BUFFER LOAD

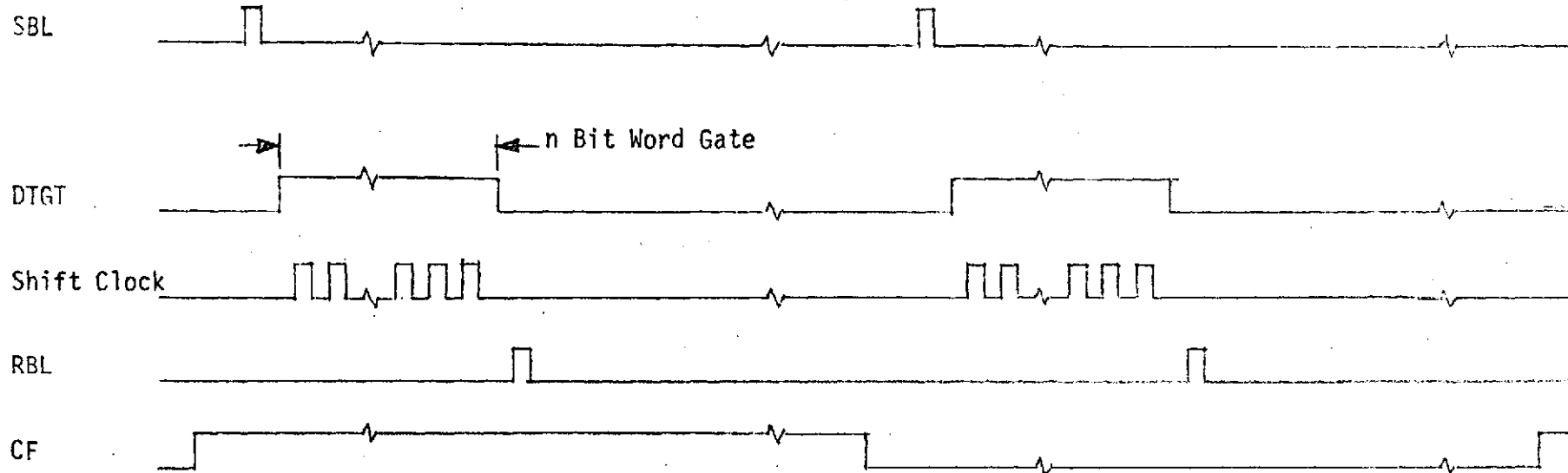


FIGURE 5.13

TELEMETRY TIMING DIAGRAM

SPECIAL CASE

$n \leq 16$

for that buffer causes the telemetry output to switch over to a second identical buffer. The ACU tests to see if a buffer is empty by addressing a buffer status discrete (address  $11000010_2$ ), and giving an SKE instruction. If either one of the buffers has been emptied, as the first one has been in the above example, a response is sent to the ACU. The ACU may then load the buffer by addressing the telemetry buffer (address  $11000011_2$ ), and giving a set bilevel (SBL) instruction. This resets the buffer empty discrete, resets the address counter for the buffer to be loaded, and sends a response to the ACU. The buffer is then loaded serially from the ACU, 16 bits at a time, with a word gate (DTGT) accompanying each 16-bit word. After the buffer has been filled with 16 16-bit words, a reset bilevel (RBL) instruction is given which disables DTGT to the telemetry buffers, again resets the buffer address to zero, and gives a response to the ACU.

Figure 5.11 is a timing diagram of a more general case where telemetry words are generally short compared to 256 bits and are accompanied by word gates (WDGT) and a shift clock (TLMCK) from the telemetry unit. The total number of bits required for one telemetry minor frame is less than 256 bits, therefore, the minor frame sync (MINFS) is used to alternate between buffers and set the buffer empty discrete. The loading of the buffers by the ACU is the same process as the first example except that less than 256 bits may be loaded.

Figure 5.12 is a timing diagram of a special case where a telemetry frame may consist of more than 256 bits. This is a combination of the cases in Figures 5.10 and 5.11 where either the minor frame sync (MINFS) or the address counter terminal count (TC1 or TC2) are used to alternate between buffers and set the buffer-empty discrete. These conditions are set up by connecting jumper wires on the circuit board.

Figure 5.13 is a timing diagram for a special case where the telemetry word lengths are 16 bits or less and the data rate is slow enough to permit the ACU to respond and refill a buffer in the time between alternate words. In this case, the 256-bit RAM's and address counters are each replaced by a simple, 16-bit shift register. The trailing edge of the telemetry word gate (WDGT) is used to alternate between buffers and set the buffer empty discrete.

A simpler case would exist where there is enough time between every telemetry word for the ACU to respond and refill the buffer. In this case, only one buffer is required, and the word gate trailing edge is used only to set the buffer-empty discrete. If the telemetry word is less than 16 bits, it is right justified in a 16-bit word from the ACU. The remaining bits are ignored.

#### 5.1.5 Commands

Figure 5.14 is a block diagram of the serial command input function. Figure 5.15 is the associated timing diagram. The command input is relatively simple since the serial commands consist of a single 32-bit word (CMDIN) accompanied by a 32-bit long word gate (CMWG). The trailing edge of CMWG sets a discrete that indicates the command buffer is full. The ACU tests the status of the discrete by addressing the command buffer (address  $11000000_2$ ) and giving a set bilevel (SBL) instruction. If the buffer is full, a response is given to the ACU, and a buffer bilevel is set which enables the buffer to be emptied. The ACU then gives a serial input (INS) instruction along with a word gate (DTGT) and the buffer contents are shifted serially into the ACU. The ACU then gives a reset bilevel (RBL) instruction which sends a response back to the ACU and resets the buffer bilevel which enables it to be loaded by the next command word.

#### 5.1.6 Program Cycle Timing

Figure 5.16 is a block diagram of the program cycle timer, while Figure 5.17 is a timing diagram of its operation in two different modes. There are three possible alternate implementations for the counter:

- o Program Cycle Fault Time-Out - The function of the timer here is to verify proper software execution of a program major cycle by performing a simultaneous hardware check on the cycle time. If the software cycle is out of bounds in either direction from an expected norm, a DPA fault is indicated. This function is central to the fault tolerant machine concept. This is defined as mode 1 operation.

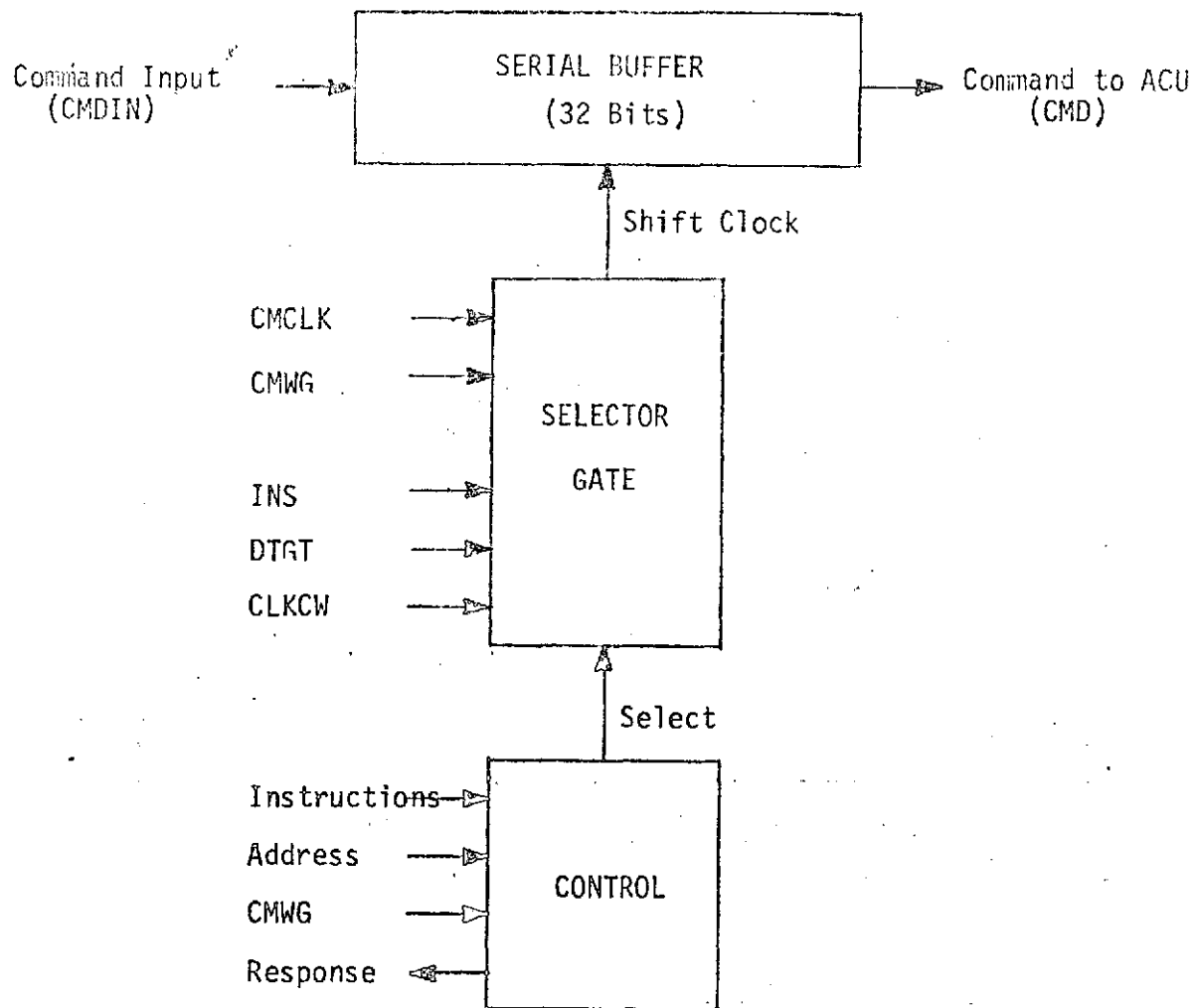


FIGURE 5.14  
COMMAND INPUT  
BLOCK DIAGRAM

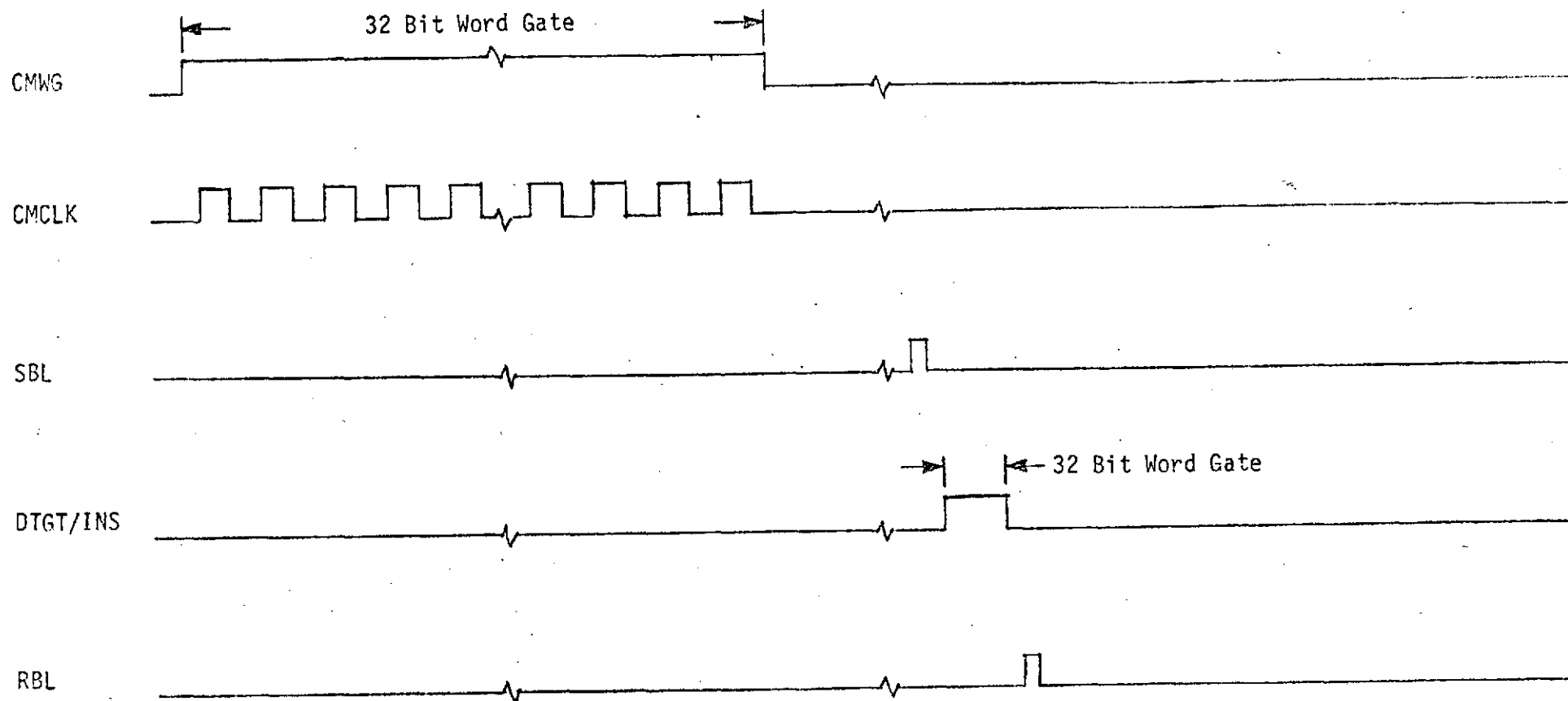


FIGURE 5.15  
COMMAND TIMING DIAGRAM

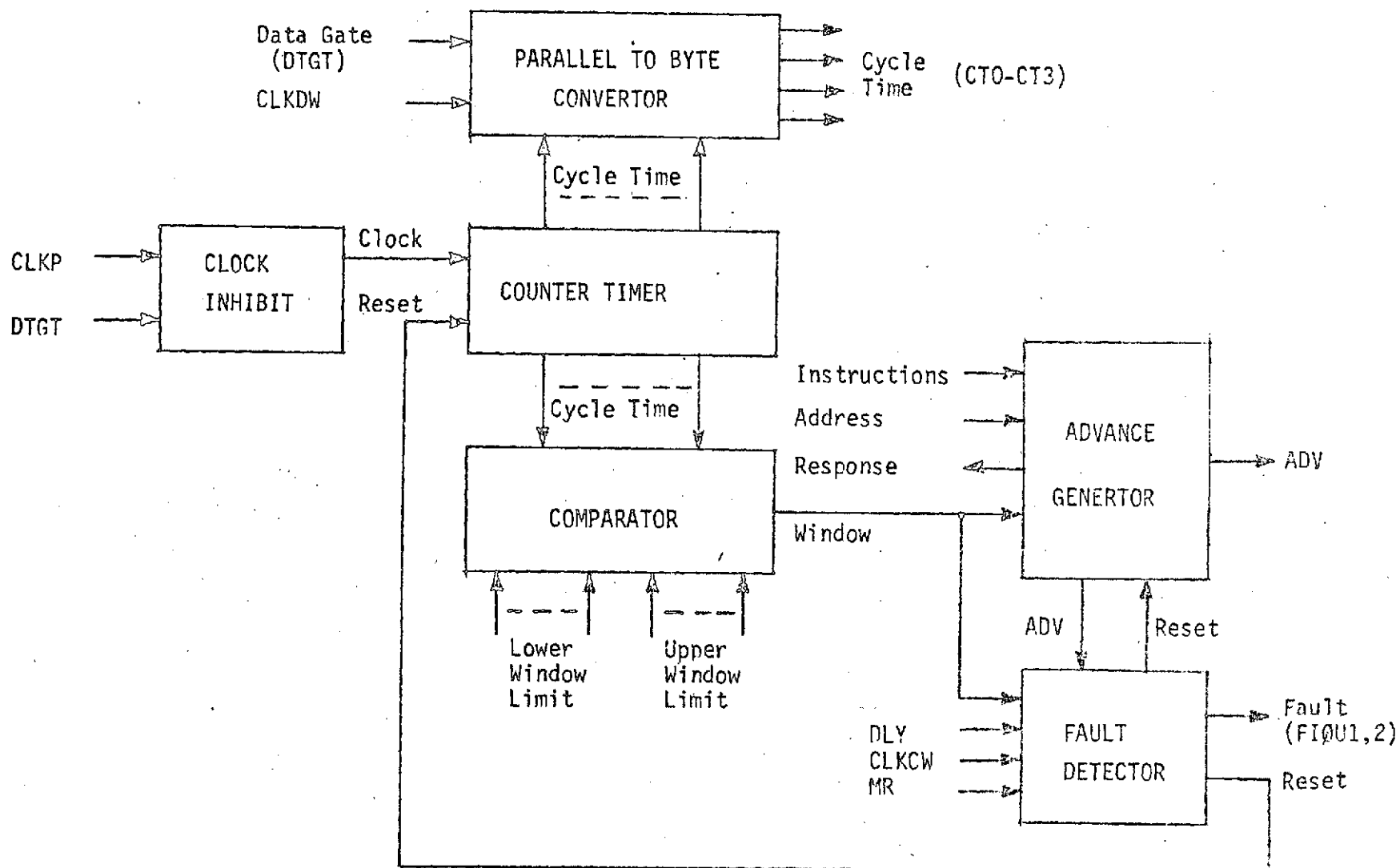
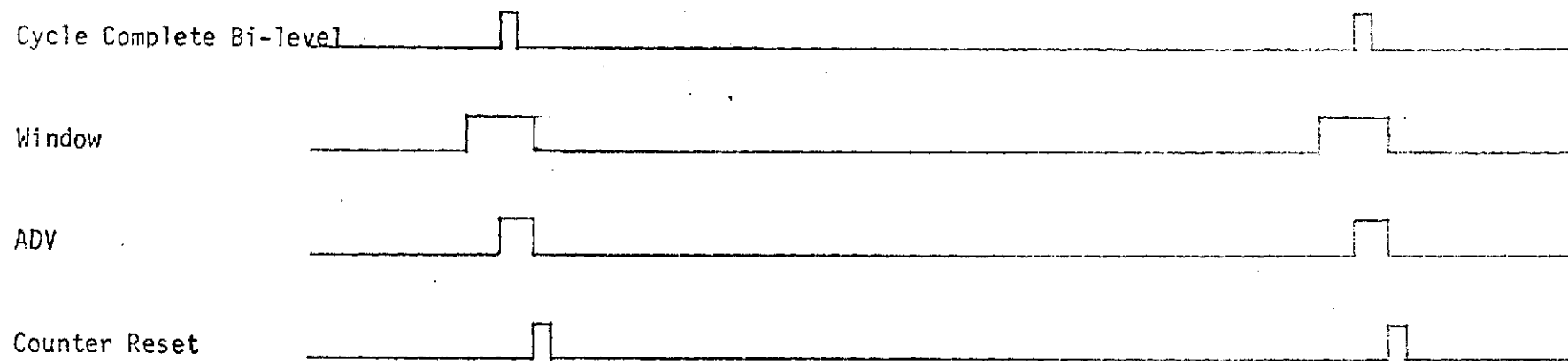


FIGURE 5.16

PROGRAM CYCLE TIMER  
BLOCK DIAGRAM

MODE 1



MODE 2

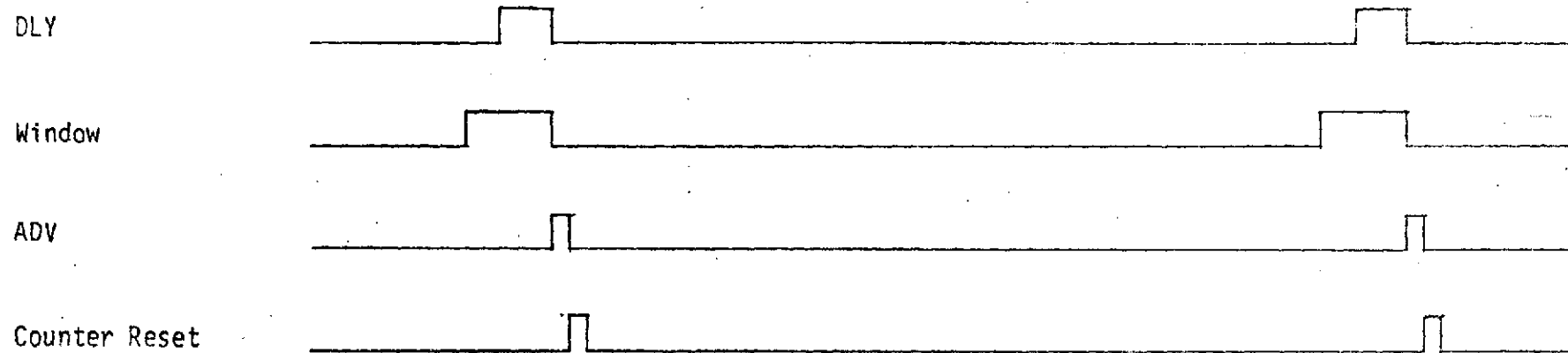


FIGURE 5.17

PROGRAM CYCLE TIMER

TIMING DIAGRAM

(Normal Operation)

- o Program Cycle External Synchronization - For many applications it is desirable for a program major cycle loop to be repeated at precise intervals. Since program major cycle times may vary to some limited degree based on the particular paths taken during the execution of a given cycle, it is not possible to precisely time-out each cycle based on the program code alone. For such applications, the counter/timer provides an external synchronization signal to the ACU to ensure a precision fixed cycle rate. This is defined as a mode 2 operation.
- o Real Time Clock - The counter/timer serves the function of providing the programmer with a real time clock source. Hardware is included to allow the program to read the state of the counter on demand.

The following signal definitions apply to the counter operation:

- DLY - This is a signal generated by the ACU when it has executed or resides in the delay condition.
- ADV - This is a signal generated by the IOU counter/timer logic to signal the ACU to advance from the delay mode. If it occurs before DLY, it is maintained in the advance state until the ACU enters the delay mode. This signal is enabled at the end of the established window and is released upon recognition of DLY.
- TMØT - This is a signal that is generated in the RCU by a re-triggerable one-shot and has a period equal to  $1-1/2 T$ , where  $T$  is the expected program major cycle time. TMØT is triggered by the trailing edge of ADV. Whenever this signal falls, the RCU forces a reconfiguration.
- WND - This is a signal generated by the counter/timer and defines the region where the ACU is expected to complete a major cycle.

The function of the program cycle time logic is to check for the following fault conditions:

- o Program major cycle time too short
- o Program major cycle time too long, including the condition where a cycle never completes due to a non-exited loop or other gross failure.
- o Failure of the program cycle fault timer itself.

The basic counter consists of an 8-bit binary counter which is clocked at either 1.2288 msec or 2.4576 msec intervals, depending on a circuit board jumper connection. The maximum count time would then be

314.5728 msec or 629.1456 msec, respectively. The counter output is fed to a digital comparator which generates the window period (WND) during which the completion of an ACU major cycle is expected to occur. The lower and upper window limits are determined by two 8-bit words fed into the comparator which are jumpered on the circuit board. At any time, the state of the counter may be read by the program by addressing the counter (address 00000000<sub>2</sub>) and giving an input (INP) instruction along with a 4.8  $\mu$ sec wide word gate (DTGT). The state of the counter will then be read into the ACU in four 4-bit bytes with the 8-bit counter state right justified, and the remaining 8 bits filled with zeros.

In mode 1 operation, a cycle complete bilevel is expected from the ACU during the window period. This is accomplished by addressing the counter (address 00000000<sub>2</sub>), and giving an SBL instruction. This tests the window, gives a response back to the ACU, causes the counter to be reset, and sends an ADV signal to the RCU. The RCU requires this signal within a fixed period of time, otherwise it determines that a fault has occurred. If the window is not present when this bilevel occurs, a fault (FIOU-1) indication is sent to the RCU. If the window falls (trailing edge of WND), and the bilevel has not occurred, FIOU-1 is also enabled. The ACU has the capability of testing the window by addressing the counter and giving an SKE instruction. If the window is present, a response is sent back to the ACU. If the software determines from this test that the cycle counter is in error, it can give an SBL instruction at address 00000001<sub>2</sub>, and a fault (FIOU-2) indication will be sent to the RCU.

In mode 2 operation, a delay (DLY) signal from the ACU is expected to go true during the window period. This signal remains true and the ACU is inoperative until the end of the window period, at which time the counter is reset, and an ADV pulse is sent to the ACU and RCU. If the leading edge of DLY fails to occur during the window, or if the window falls without DLY having occurred, then a fault (FIOU-1) indication is sent to the RCU. The ACU has the same capability of testing the window and generating a fault (FIOU-2) indication as in mode 1 operation.

### 5.1.7 Discrete Inputs

Figure 5.18 is a block diagram of the discrete input function, while Figure 5.19 is the related timing diagram. A discrete is defined as a function which is set by an input pulse and reset when its status is tested by the ACU.

There are a maximum of 28 discrete inputs (DI4-DI31) available to the peripheral equipment. Each discrete is set by the trailing edge of a 400 nsec minimum wide pulse applied to the appropriate input. The status of a discrete is tested by addressing the desired discrete (addresses  $11000100_2$  through  $11011111_2$ ), and giving an SKE instruction. If the addressed discrete is in the one state, it is reset and a response is sent to the ACU. All discrettes are reset during the initialization process (INRST). INRST generates the master reset (MR) for all functions in the IOU.

### 5.1.8 Bilevel Inputs

Figure 5.20 is a block diagram of the bilevel input function, while Figure 5.21 is the related timing diagram. A bilevel input is defined as a function that is set or reset solely by the peripheral source, and is not affected by having its status tested by the ACU. There are a maximum of 64 bilevel inputs (BLI0-BLI63) available. The status of a bilevel input is tested by addressing the desired bilevel ( $10000000_2$  through  $10111111_2$ ) and giving an SKE instruction. A response is sent to the ACU if the addressed bilevel is in the one state.

### 5.1.9 Bilevel Outputs

Figure 5.22 is a block diagram of the bilevel output function, while Figure 5.23 is the related timing diagram. A bilevel output is defined as a function which is separately set and reset by instructions from the ACU. There are a maximum of 78 bilevel outputs (BL02 through BL079) available to the peripheral equipment. This includes BL02 which is used to select between the two redundant peripheral buses for serial data transfer. A bilevel output is set to a one by addressing the desired bilevel (addresses

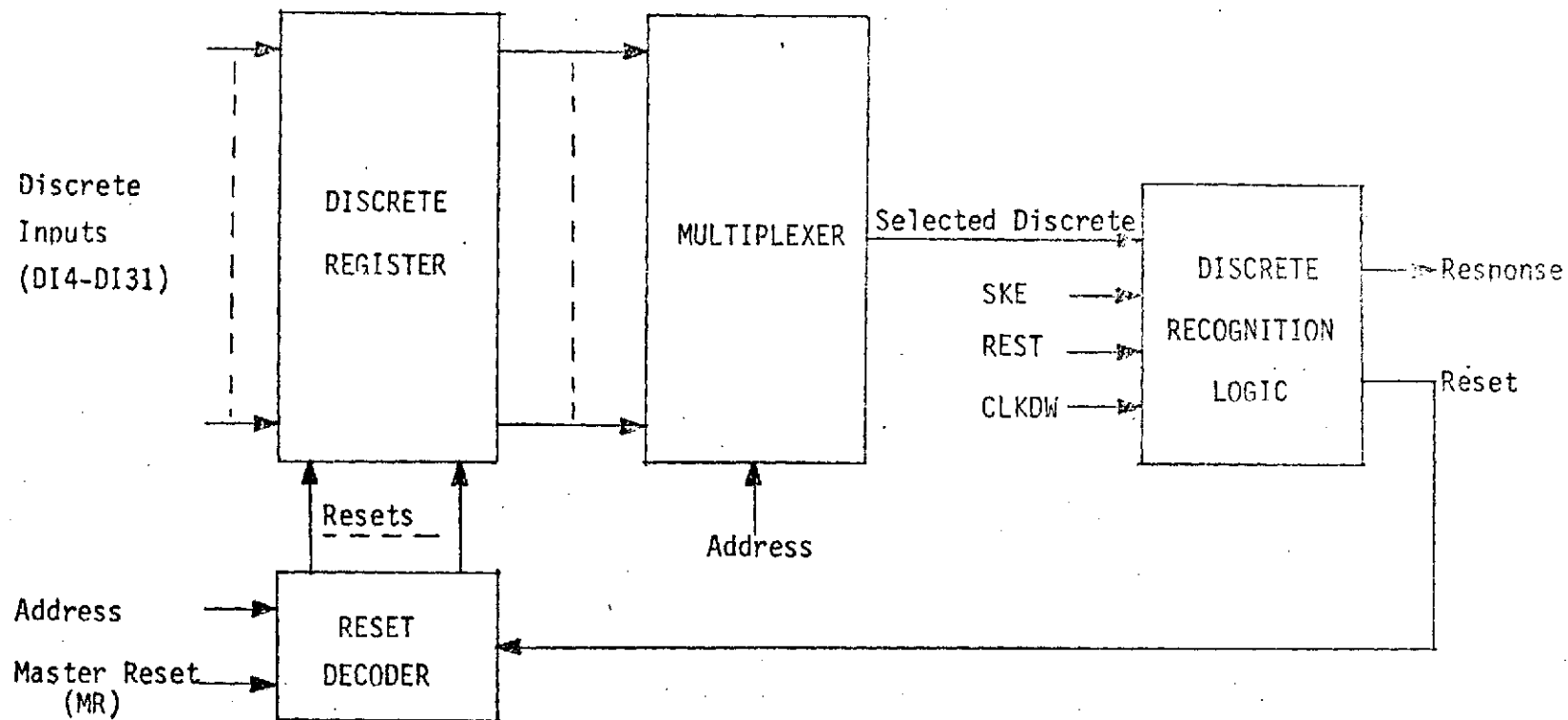


FIGURE 5.18

DISCRETE INPUT  
BLOCK DIAGRAM

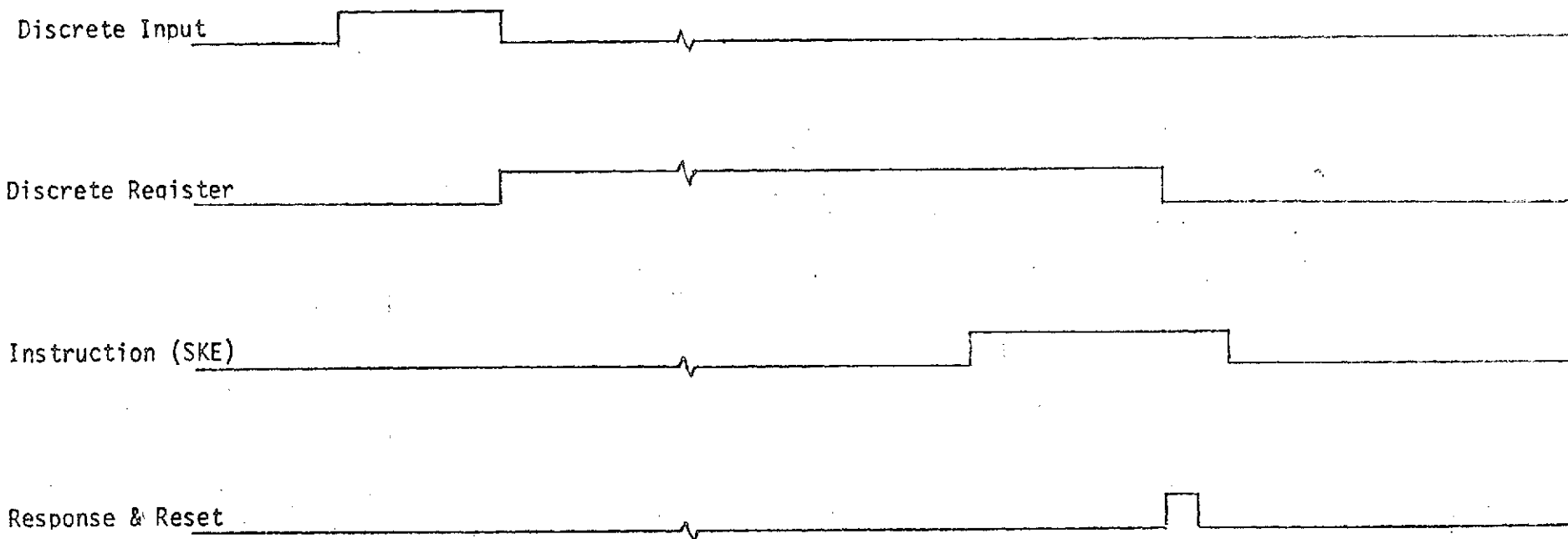


FIGURE 5.19  
DISCRETE INPUT  
TIMING DIAGRAM

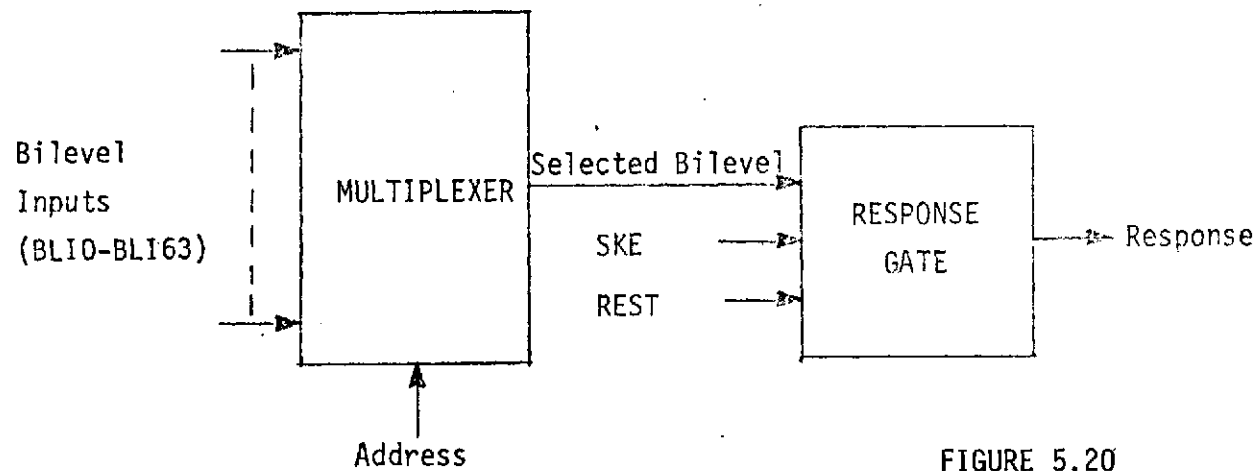


FIGURE 5.20  
BILEVEL INPUT  
BLOCK DIAGRAM

108

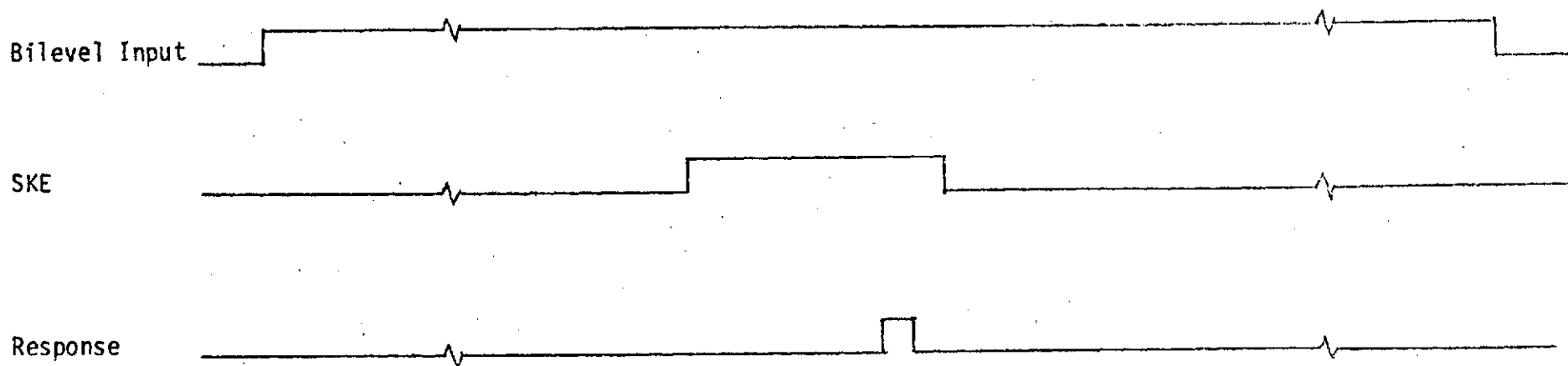


FIGURE 5.21  
BILEVEL INPUT  
TIMING DIAGRAM

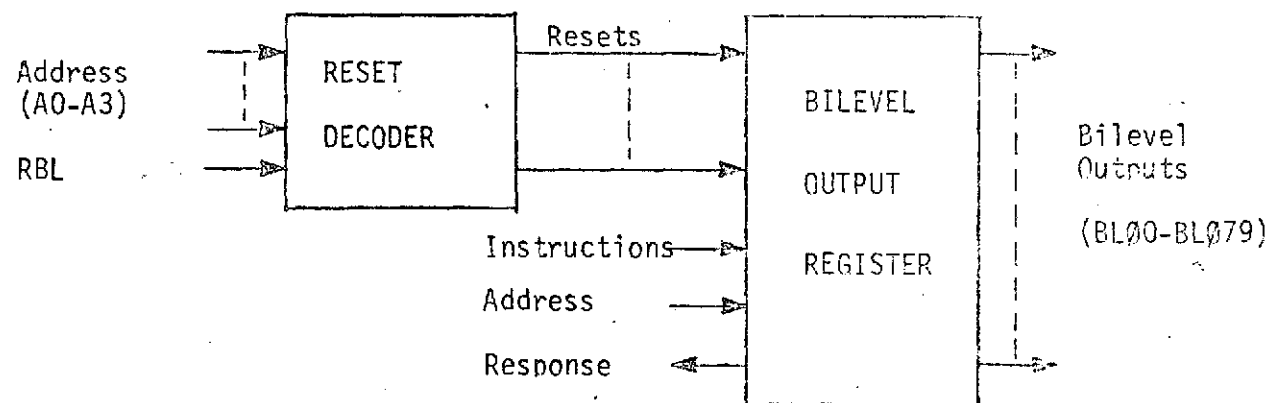


FIGURE 5.22  
BILEVEL OUTPUT  
BLOCK DIAGRAM

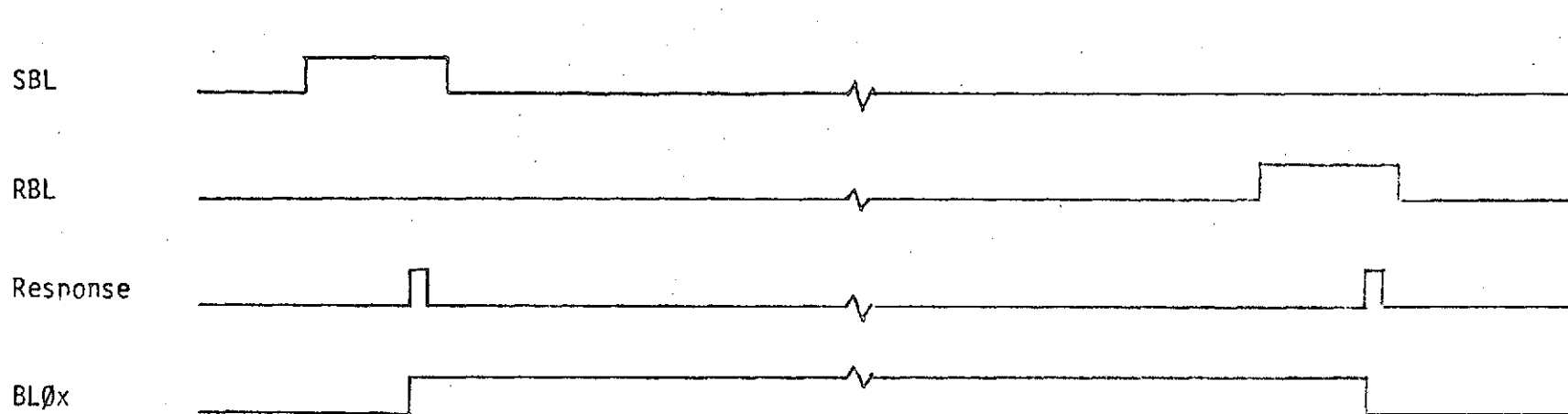


FIGURE 5.23  
BILEVEL OUTPUT  
TIMING DIAGRAM

00000010<sub>2</sub> through 01001111<sub>2</sub>), and giving an SBL instruction. It is reset by giving the same address and an RBL instruction. A response is always given to the ACU in either case. All bilevel outputs are reset by the master reset (MR).

## 5.2 Peripheral Interface Definition

This section defines the electrical characteristics of all interfaces between the IOU and the peripheral equipment outside the DPA. Note that the direct interfaces between the RCU and peripheral equipment is specified in Section 6.0.

Except for the analog signal inputs, all signal levels are positive logic, TTL compatible, where the levels are defined as follows:

Logic 0: 0 v to +0.4 v  
One unit load = -400  $\mu$ a

Logic 1: +2.4 v to +5.0 v  
One unit load = 20  $\mu$ a at +2.4 v

Three different types of cross-strapping are used in the IOU-peripheral interface. These consist of conventional cross-strapping, dual redundant bus, and duplication and selection. Briefly, they consist of the following:

- o Conventional Cross-Strapping - This consists of sender and receiver conventional gates with pull-up resistors on the receivers. This interface is used for the command and telemetry lines.
- o Dual Redundant Bus - This will be referred to as the peripheral bus and consists of dual redundant buses, A and B. Power to either bus is controlled by a bilevel called "BUSBL". When in the "0" state, Bus A is enabled, and when in the "1" state, Bus B is enabled. Power to the sender in any peripheral equipment is also gated by the bilevel which address that equipment. This would be the serial word gate input bilevel (SBLix) in the case of serial data.
- o Duplication and Selection - In this case, each peripheral element has a separate line running to/from the IOU for each interface. The IOU selects which line is to be interrogated. The software keeps track of which of the peripherals are in use and which are in standby. This interface is used for bilevels, discretes and analog inputs.

The following sections detail the various interfaces.

### 5.2.1 Analog Inputs

The A/D converter accepts up to 32 analog input channels. An internal reference voltage is available which may be externally connected to any number of inputs for self-test purposes. The selection of the input channels and sample rate is controlled by the ACU. Each input is a semi-differential configuration where the return side is internally isolated from ground by approximately 10K $\Omega$  for the selected channel and essentially open circuit for the non-selected inputs.

Specifications:

- Voltage range:  $\pm 5.0$  v
- Overvoltage tolerance:  $\pm 15$  v
- Resolution: 12 bits
- Accuracy:  $\pm 0.1\%$
- Source Impedance:  $10K\Omega$  max
- Frequency response: dc to 160 Hz from  $0\Omega$  source  
dc to 80 Hz from  $10K\Omega$  source
- Signal name designation: ANGOS and ANGOR through  
ANG30S and ANG30R where S  
is the signal line and R  
is the return line
- Number of inputs: 32 max
- Input modularity: 8
- Cross-strapping: duplication and selection

### 5.2.2 Discrete Inputs

Discrete inputs are defined as those inputs which are set by a pulse from the peripheral equipment and are reset by the hardware in the IOU whenever the status of the selected discrete has been tested by the ACU. All discrettes are automatically reset during power turn on to the IOU.

Specifications: Discrete set: Trailing edge of pulse  
Minimum pulse width: 400 nsec  
Loading: Logic 0, 0.9 unit load  
Logic 1, 1 unit load  
Number of inputs: 28 max\*  
Modularity: 8  
Signal name designation: DI4-DI31\*  
Cross-strapping: duplication and selection

\*NOTE: There is internal provision for 32 discretes, but the first 4 (DIO-DI3) are dedicated to internal IOU functions.

### 5.2.3 Input Bilevels

Input bilevels are defined as those inputs which are continuous as long as the represented condition exists. They are not affected by having their status checked by the ACU.

Specifications: Loading: Logic 0, 1 unit load  
Logic 1, 1 unit load  
Number of inputs: 64 max  
Modularity: 8  
Signal name designation: BLIO-BLI63  
Cross-strapping: duplication and selection

### 5.2.4 Serial Input Data

Figure 5.24 is a timing diagram for the serial input data. The same clock (CLKP) is used for both serial input and output data shifting. All data is clocked on the leading edge of the clock. The word gate bilevel (SBLIx) is used to enable the bus drivers in the peripheral equipment for serial data input. Therefore, the power gates for the drivers should have a fast turn on time (<100 nsec). The bilevel that selects between the two redundant buses would ordinarily come from BL02 in the I0U. This will be provided by a jumper wire on the connector to permit other sources to control the bus select for some applications.

Specifications: Serial Input Bilevel  
Drive Capability: Logic 0, 8 unit loads  
Logic 1, 16 unit loads  
Duration: 16 clock periods  
Number of bilevels: 32 max  
Modularity: 16  
Signal name designation: SBLIO-SBLI31  
Cross-strapping: sender, duplication and selection  
receiver, conventional

#### Peripheral Shift Clock

Drive Capability: Logic 0, 12 unit loads  
Logic 1, 24 unit loads  
Powered off loading: Logic 1, 1.5 unit loads  
Frequency: 104.167 KHz  
Period: 9.6  $\mu$ sec

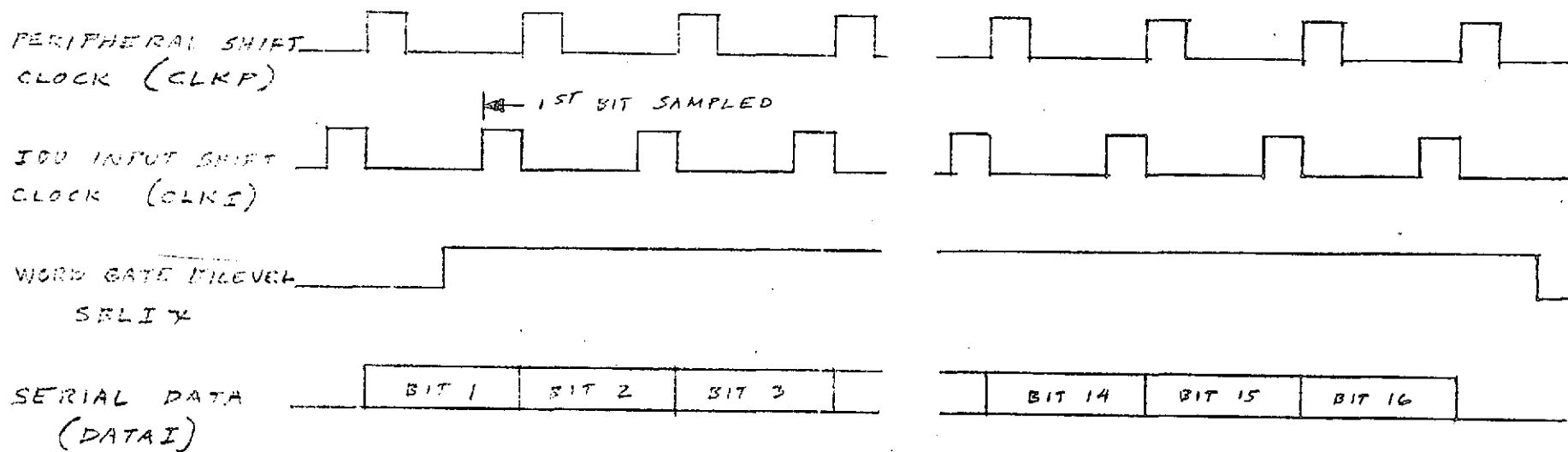


FIGURE 5.24 SERIAL INPUT TIMING

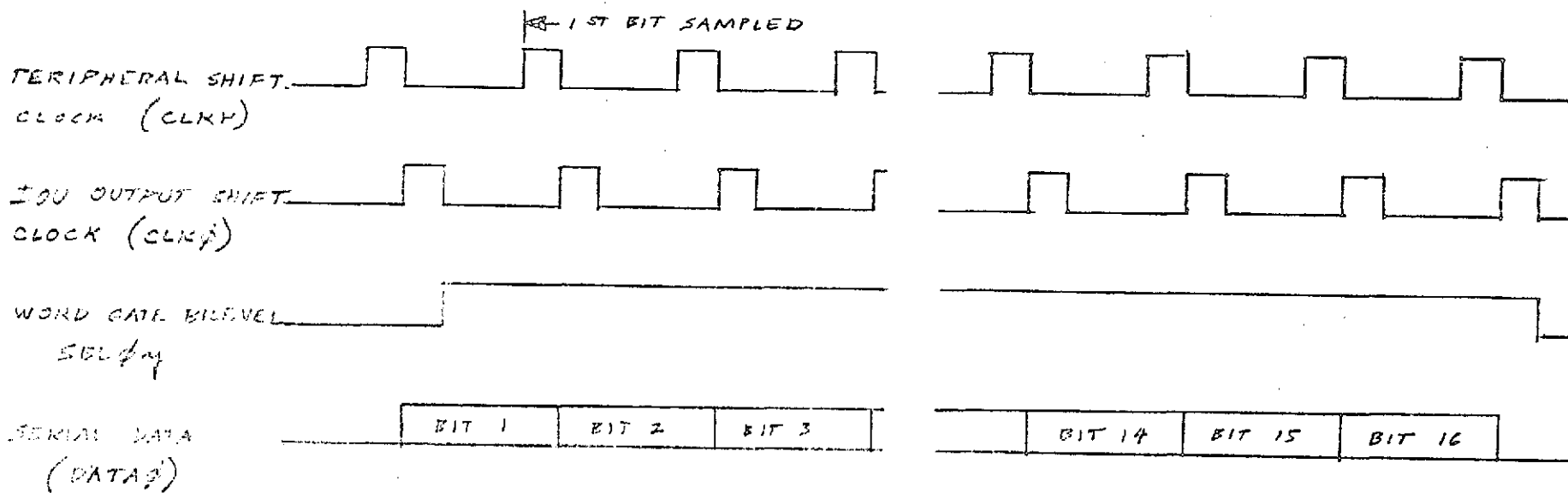


FIGURE 5.25 SERIAL OUTPUT TIMING

#### Peripheral Shift Clock (continued)

Duty cycle: 25% (pulse width = 2.4  $\mu$ sec)

Signal name designation: CLKP

Cross-strapping: peripheral bus

#### Serial Data Input

Loading: Logic 0, 0.45 unit load

Logic 1, 0.5 unit load

Powered off, logic 1, 0.25 unit load

Word Length: 16 bits, LSB first

Bit rate: CLKP rate

Signal name designation: DATAI

Cross-strapping: peripheral bus

### 5.2.5 Serial Output Data

Serial output data transfer is exactly the same as serial input data transfer except a separate bus line is used for a data line. Figure 5.25 is the timing diagram for serial output.

#### Specifications: Serial Output Bilevel

Drive capability: Logic 0, 8 unit loads

Logic 1, 16 unit loads

Duration: 16 clock periods

Number of bilevels: 16 max

Modularity: 16

Signal name designation: SBL00-SBL015

Cross-strapping: sender, duplication and selection  
receiver, conventional

#### Peripheral Shift Clock

Drive capability: Logic 2, 12 unit loads

Logic 1, 24 unit loads

Powered off loading: Logic 1, 1.5 unit loads

Frequency: 104.167 KHz

Period: 9.6  $\mu$ sec

Duty cycle: 25% (pulse width = 2.4  $\mu$ sec)

Signal name designation: CLKP

Cross-strapping: peripheral bus

#### Serial Data Output

Drive capability: Logic 0, 8 unit loads

Logic 1, 16 unit loads

Powered off loading: Logic 1, 1 unit load

Word length: 16 bits, LSB first

Bit rate: CLKP rate

Signal name designation: DATA0

Cross-strapping: peripheral bus

### 5.2.6 Commands

The IØU contains a single command input buffer. Therefore, only one command word can be loaded in at a time. The ACU must empty the buffer before the next word can be loaded in. The falling edge of the command word gate (CMWG) sets an internal bilevel which is tested by the ACU to determine when a command word is in the buffer. This bilevel blocks any further commands from being loaded into the buffer. It is reset by the ACU after the buffer has been emptied. The command source supplies the word gate and shift clock to the IØU for loading commands into the IØU.

#### Specifications: Command Word

Loading: Logic 0, 2.5 unit loads  
          Logic 1, 0 unit load  
Word length: 32 bits max, LSB first  
Modularity: 8 bits  
Signal name designation: CMD  
Cross-strapping: conventional

#### Command Shift Clock

Loading: Logic 0, 2.5 unit loads  
          Logic 1, 0 unit load  
Clock pulse width: Logic 1, 400 nsec min  
                    Logic 0, 300 nsec min  
Clock edge: leading edge  
Signal name designation: CMCLK  
Cross-strapping: conventional

#### Command Word Gate

Loading: Logic 0, 2.5 unit loads  
          Logic 1, 0 unit load  
Length: command word length  
Signal name designation: CMWG  
Cross-strapping: Conventional

### 5.2.7 Output Bilevels

Output bilevels are defined as continuous outputs which are set and reset under ACU control. All output bilevels are automatically reset during power turn on to the IØU.

Specifications: Drive capability: Logic 0, 7 unit loads  
Logic 1, 14 unit loads  
Number of outputs: 78 max\*  
Modularity: 4  
Signal name designation: BL02-BL079\*  
Cross-strapping: sender, duplication and selection  
receivers, conventional

\*NOTE: There is internal provision for 80 output bilevels but the first 2 (BL00, BL01) are dedicated to internal IOU functions. BL02 will ordinarily be dedicated for peripheral bus selection.

#### 5.2.8 Telemetry

The telemetry interface may be implemented with long or short buffers. The digital telemetry unit (or equivalent peripheral equipment) will supply the shift clock, word gate, and minor frame sync. A major frame sync, if required, may be handled by a conventional discrete input.

Specifications: Minor Frame Sync

Loading: Logic 0, 2.5 unit loads  
Logic 1, 0 unit load  
Minimum pulse width: 600 nsec  
Signal name designation: MINFS  
Cross-strapping: conventional

Word Gate

Loading: Logic 0, 2.5 unit loads  
Logic 1, 0 unit load  
Length: Telemetry word length  
Signal name designation: WDG  
Cross-strapping: conventional

Telemetry Data

Drive capability: Logic 0, 32 unit loads  
Logic 1, 16 unit loads  
Word length: Determined by telemetry unit  
Signal name designation: TMDI0  
Cross-strapping: conventional

Telemetry Clock

Loading: Logic 0, 2.5 unit loads  
Logic 1, 0 unit load  
Frequency: 500 KHz max  
Signal name designation: TLMCK

## 6.0 RECONFIGURATION CONTROL UNIT (RCU)

The RCU is an optional module of the DPA. It is used for those applications which require automatic on-board DPA reconfiguration in the event of faults in the DPA. The RCU controls the turn on and turn off of power to the other DPA units, and thus controls the redundancy of the DPA.

When the RCU is not used, an external four-phase clock must be supplied to the DPA. Also, the signals for turn on and off of power to the DPA units must be supplied externally.

### 6.1 Internal Organization

The RCU is designed using triple modular redundancy (i.e., using three identical decision-making sections, with voting to determine the outputs). There shall be no single failures in the RCU that will cause failures in the DPA to go undetected or uncorrected.

The RCU consists of four portions: the three identical sections (A, B and C) and a fourth section providing voting, differencing, clock oscillators, clock fault detection and switching. All portions of the RCU are continuously powered (and are controlled by commands processed and originating outside of the DPA). See Figure 6.1.

### 6.2 Functions

The primary functions of the RCU are two: 1) Redundantly generate the four-phase 819 kHz clocks for the DPA, and 2) control the manual or automatic reconfiguration of the DPA by the processing of fault signals and control of unit power turn-on.

Secondarily, the RCU provides: 1) DPA initialization and reset at power turn-on, after a reconfiguration, or by command, 2) Assignment of unique page numbers to each RAM module in use, 3) The program cycle fault detection, 4) Its own command processing and telemetry processing for communication from and to the world outside the DPA, and 5) The communication of its status (and which modules of the DPA are on) to the ACU.

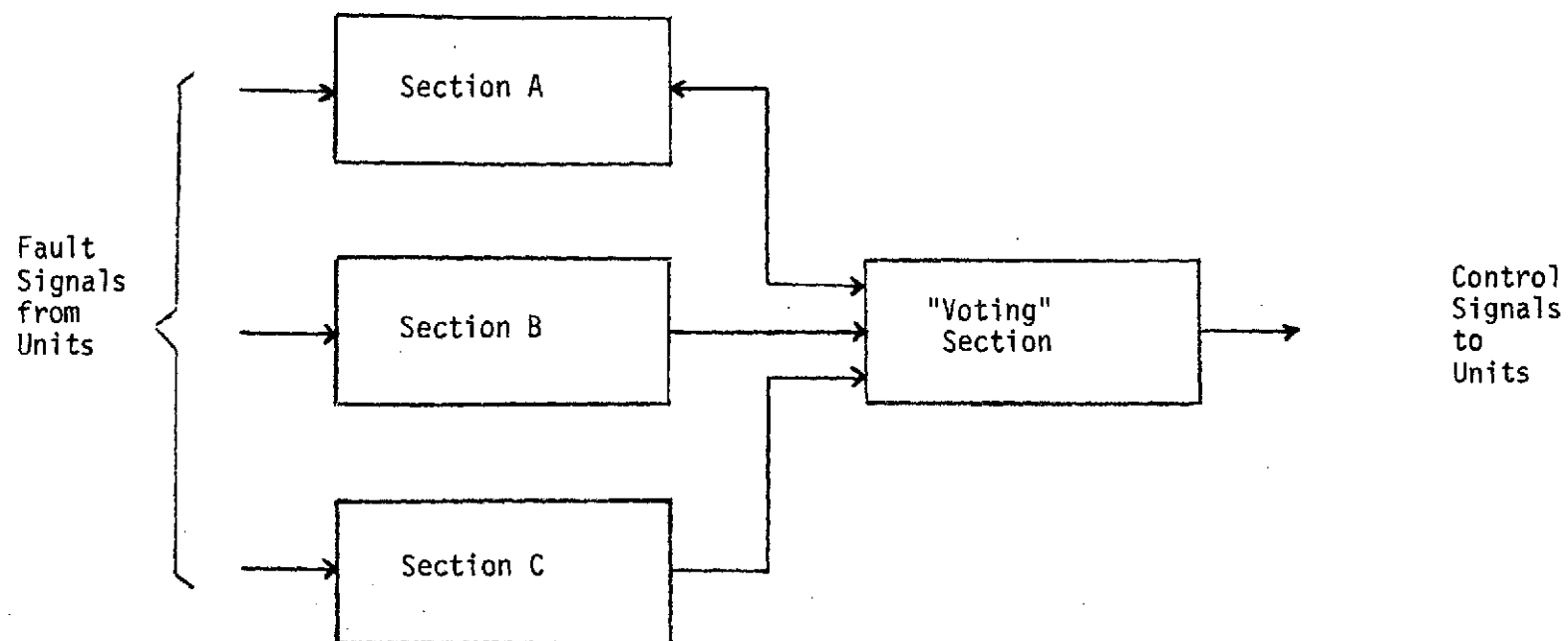


Figure 6.1  
RCU OVERALL BLOCK DIAGRAM

The RCU is designed to control up to the following quantity of modules of the other DPA units:

ACU	3
IOU	3
DBS	2
ROM 1	2
ROM 2	2
RAM	6

It shall also be designed so that it may control lesser quantities of any or all of these modules, with correspondingly reduced reconfiguration time.

The reconfiguration process is started when two fault signals are received by the RCU within a time period. This reconfiguration process is (except for RAM faults) enumerative. This means that starting with the current configuration, one unit at a time is replaced by a standby unit until the fault does not re-occur. For RAM faults, only the RAM units are reconfigured, also enumeratively, until the fault signal does not occur again. No record is kept of previously failed units. The first attempts at reconfiguration are made with the ACU, then IOU, etc., in the order given in the above table. This is done since the ACU has the highest failure probability, IOU next, etc.

### 6.3 Functional Requirements

#### 6.3.1 Reconfiguration

Reconfiguration shall begin after two sequential fault signals have been received within 220 to 320 msec, when the RCU is in the automatic mode. The receipt of two sequential RAM fault signals within this time period results in a confirmed RAM fault, which initiates stepping of the RAM unit power control. The receipt of two sequential fault signals of any other type within this time period results in a confirmed general fault, which initiates stepping of all of the units power control.

The power control signals to each unit consist of voted bilevel signals, one for each module (18 in all) which are high when that module is to be on and low at all other times. A bilevel signal is delivered back to the RCU from each module confirming the actual power (on-off) status of that unit. The power control signal and corresponding power confirmation signal for each module are compared. If any unit should be on, but is not, this is treated as a fault signal (RAM or general, as appropriate).

If the RCU is in a reset condition, no reconfiguration occurs due to the fault messages. When in the manual mode sequential stepping of any particular type of unit may occur by command. All units may be turned off by the off command to the RCU. The initialize command may be used to turn on the "initial combination of units.

6.3.1.1 Fault Signals. The possible fault signals are:

FIOUA-1	}	General Faults	FRAMA	}	RAM Faults
FIOUB-1			FRAMB		
FIOUC-1			FRAMD		
FIOUA-2			FRAME		
FIOUB-2			FRAMF		
FIOUC-2					
FACUA	}	not now assigned			
FACUB					
FACUC					
AUXA					
AUXB					

6.3.1.2 Program Cycle Fault Detection

The RCU shall contain timers, which must be reset each 97.6 to 100.04 msec by the ADV (advance) signal from the ACU, or a fault signal will be generated, when in the automatic mode. This detects program cycle time faults, resulting either from hardware failures or software faults.

### 6.3.1.3 Power Turn-on Initialization

When power is turned-on to the RCU, or if power is turned off for more than 10  $\mu$ sec and then turned on again, the RCU shall initialize the unit power control to the "initial" combination of units in the automatic mode. This prevents power interruptions from placing the DPA in an "off" condition. If the initial combination is faulty, reconfiguration will then automatically occur.

### 6.3.1.4 Sequence of Reconfiguration

The sequence of reconfiguration is the order in which the enumerative reconfiguration occurs, while the confirmed fault signals persist.

#### 6.3.1.4.1 RAM Sequence

For a confirmed RAM fault, only the RAM units shall be sequenced (stepped). Since the RCU can control a maximum of 6 RAM modules, the following cases of number of RAM modules available and needed can occur:

<u>Case</u>	<u>Available</u>	<u>Needed</u>	<u>Initial Combination</u>	<u>Total Combination</u>
1	2	1	A	2
2	3	1	A	3
3	4	1	A	4
4	5	1	A	5
5	6	1	A	6
6	3	2	AB	3
7	4	2	AB	6
8	5	2	AB	9
9	6	2	AB	15
10	4	3	ABC	4
11	5	3	ABC	10
12	6	3	ABC	20
13	5	4	ABCD	5
14	6	4	ABCD	15

Also shown are the "initial" combination of RAM modules and the number of combinations\* of RAM units which can meet the "needed" requirements. A complete cycle of the sequence of RAM stepping will test all combinations

---

\*Given by  $\frac{A!}{N!(A-N)!}$ , where A = no. available and N = no. needed

prior to repeating. For example, for the 6 available, 2 needed case (case 9) the sequence will be:

```

Initialize → AB → BC → CD → DE → EF → FA → AC → BD
              ↑      CF ← BE ← AD ← FB ← EA ← DF ← CE

```

The stepping may begin at any point in this sequence, that point being determined by the current combination at the time of the confirmed RAM fault. This tends to put previously tried combinations at the end of the sequence.

All cases shall be selectable in the RCU by changes in the terminal-to-terminal hardwiring internal to the RCU.

#### 6.3.1.4.2 General Sequence

For a confirmed general fault, all units shall be sequenced (stepped). This sequencing shall begin by stepping each unit type by one step until each of the six types have been stepped. This is intended to most quickly correct single failures, as they should be avoided in this first pass. If this is unsuccessful (the confirmed general fault persists), then all combinations of units are stepped through. The How of this stepping is shown in Figure 6.2.

The maximum number of combinations which are stepped through (assuming the maximum number of modules of all types and case 12 for the RAM's) is:

$$6 + (3 \times 3 \times 2 \times 2 \times 2 \times 20) = 1446$$

The sequence shall repeat indefinitely if no viable combination is found.

#### 6.3.1.5 System Reset

Following each reconfiguration attempt, the RCU shall generate a nominal 12 msec reset to the ACU (and to the RCU fault confirmation logic) to cause a reset and initialization of the program. The program must then run through two cycles, and be faulty in both, before another reconfiguration step is made. This DPA reset may also occur by command.

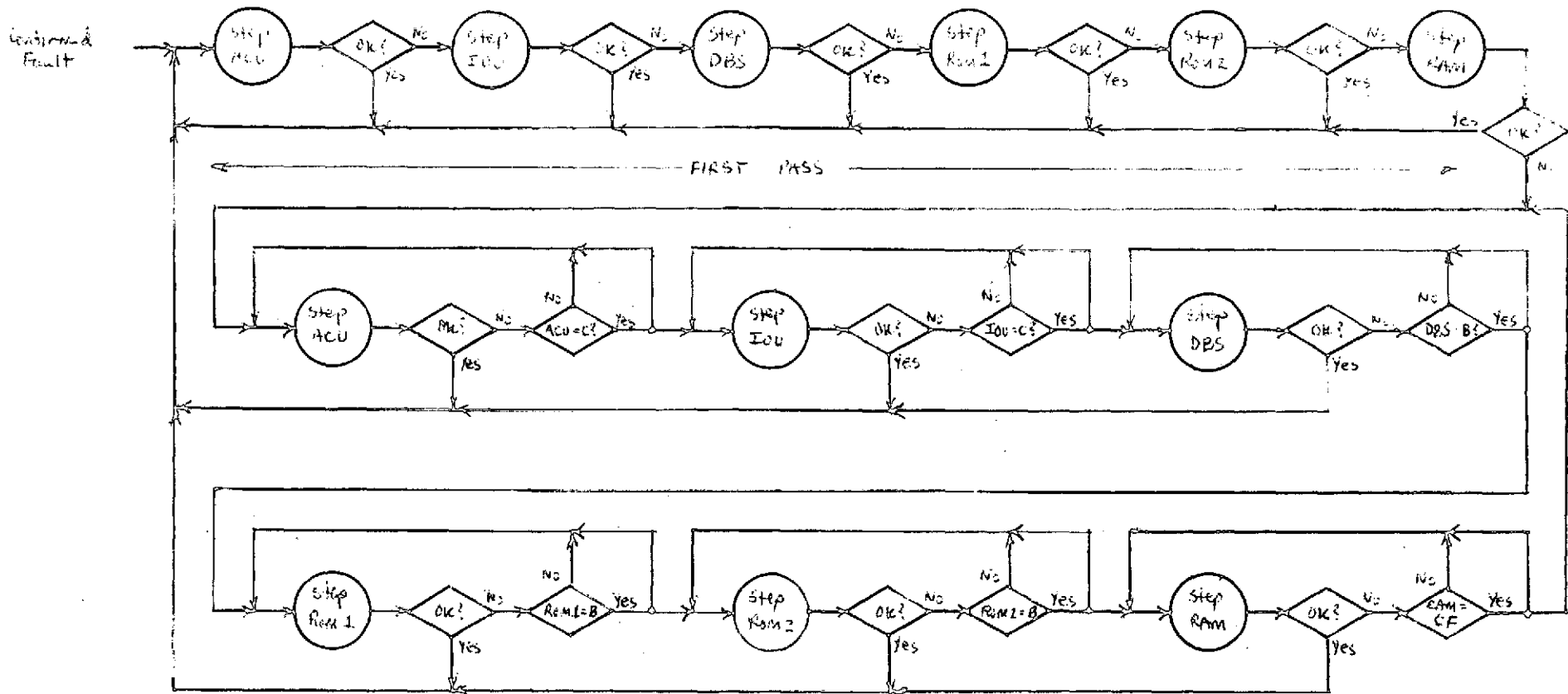


Figure 6.2 RCU Reconfiguration Flow

#### 6.3.1.6 RAM Page Assignments

Since from 1 to 4 RAM modules can be on at a time, for all cases of 2 or more on at a time, it is necessary for the RCU to assign each on RAM module a unique program address, termed a page number. This page number is a two-bit code, indicating which of the four possible page numbers is assigned to a particular RAM module. The RCU shall assign these page numbers so that they are unique, in the order 00, 01, 10, 11.

#### 6.3.2 Clock Generation

The DPA clocks shall be four-phase 819 kHz signals, generated in a completely redundant fashion so that no single failures can change the frequency by more than 10 kHz or prevent any or all of the clock phases from being present. These phases shall be derived from redundant 6.554 MHz crystal oscillators. All redundancy shall be passive and automatic without command interaction. Clock faults shall not cause any effect on the DPA other than to cause the timing to "hold" for up to 300 msec, at which time operation is resumed.

#### 6.3.3 RCU Communication

RCU communication can occur between the RCU and the DPA (ACU) or between the RCU and the non-DPA external world. The former allows for communication to the DPA of the reconfiguration status (which units are on) and (through the IOU), telemetry to the external world. The latter provides for those commands to the RCU that are needed (and which cannot go through the IOU, because the "on" IOU may be failed), and for the RCU status telemetry (which includes the reconfiguration status) (and which also cannot go through the IOU).

##### 6.3.3.1 ACU Communication

The ACU communication consists of a serial exchange of data between the RCU communication register and the ACU A and Q registers upon a TRC instruction. The format of this data is shown in Table 6.1. A "1" indicates the module is selected (on).

Table 6.1  
RCU Status Word in ACU

Bit No.	Register	Module Selected	Other Status
0	A	ACU A	
1		ACU B	
2		ACU C	
3		IOU A	
4		IOU B	
5		IOU C	
6		DBS A	
7		DBS B	
8		ROM1 A	
9		ROM1 B	
10		ROM2 A	
11		ROM2 B	
12		RAM A	
13		RAM B	
14		RAM C	
15		RAM D	
0	Q	RAM E	
1		RAM F	
2			DSTM1, Voting Difference Status, Unit Control*
3			DSTM2, " " " " " "
4			OABC, Clock On (1 = Clock A)
5			OSCAF, Clock A failure (if = 1)
6			OSCBF, Clock B failure (if = 1)
7			DSTA1, Voting Difference Status, System*
8			DSTB1, Voting Difference Status, System*
9			DSTA2, Voting Difference Status, Timing*
10			DSTB2, Voting Difference Status, Timing*
11			Not Used
12			
13			
14			
15			

\*For the two bits:  
(in each pair)

A	B	
0	0	= No RCU faults
1	0	= Section A failure
0	1	= Section B failure
1	1	= Section C failure

The voting difference status bits indicate failures in the redundant sections of the RCU, as indicated.

#### 6.3.3.2 Commands

The RCU command interface with the external world is serial digital. This interface is cross-strapped for redundant sources and consists of three types of signals. They are: 1) 4-bit serial NRZ data word (CMDR), 2) Data clock (CMCLKR) synchronized with the data word, and non-synchronous with the DPA. The frequency of clock and data may be any desired, and 3) A word gate, (CMWGR) enclosing the data word.

Upon the fall of the word gate, the new command will be executed and will be stored until a subsequent command replaces it. A power reset will set the command register to the automatic mode state.

The command decoding is shown in Table 6.2. The reset DPA or unit step commands are executed once per command, only when in the manual mode.

#### 6.3.3.3 Telemetry

The RCU telemetry interface with the external world is serial digital. This interface is cross-strapped for redundant destinations and consists of three types of signals. They are: 1) 16-bit serial data word (TMDR), 2) Telemetry clock (TLMCKR), which will synchronize the bits of the data word, and 3) A word gate (WDGTR), which must enclose the 16 bits of the data word.

The telemetry bit rate is non-synchronous with the DPA and may occur at any frequency. Telemetry data changes are blocked out while the word gate is present. The encoding of the telemetry data is shown in Table 6.3. There is a completely separate telemetry interface with each of the three redundant sections of the RCU.

Table 6.2  
RCU COMMAND CODING

<u>Bits</u>				<u>Function</u>	
<u>*MSB</u>			<u>LSB</u>		
0	0	0	0	DPA Off	
0	0	0	1	Initialize	
0	0	1	0	Manual Mode	
0	0	1	1	Automatic Mode	
0	1	0	0	-	
0	1	0	1	-	
0	1	1	0	-	
0	1	1	1	-	
1	0	0	0	Reset DPA	} Accepted only for manual mode
1	0	0	1	Step ACU	
1	0	1	0	Step IOU	
1	0	1	1	Step DBS	
1	1	0	0	Step ROM 1	
1	1	0	1	Step ROM 2	
1	1	1	0	Step RAM	
1	1	1	1	-	

\* Note - MSB arrives first

Table 6.3  
RCU TELEMETRY CODING

	<u>Bit No.</u>	<u>Indication*</u>	
First	0	RCUDS	(Voting difference)
	1	RAM F	
	2	RAM E	
	3	RAM D	
	4	RAM C	
	5	RAM B	
	6	RAM A	
	7	ROM 2	} "1" = A unit on "0" = B unit on
	8	ROM 1	
	9	DBS	
	10	IOUC	
	11	IOUB	
	12	IOUA	
	13	ACUC	
	14	ACUB	
Last	15	ACUA	

\*Note - a logical "1" indicates on

## 6.4 Tester Interfaces

The RCU interfaces with the DPA tester allow stimulation and monitoring of the RCU during test. The inputs (stimuli) are listed in Table 6.4 and the outputs are listed in Table 6.5.

Table 6.4  
RCU TEST INPUTS

<u>Name</u>	<u>Function</u>
GREST-T/	Provides RCU (&DPA) reset
GENF-T/	Simulates general power configuration
RAMF-T/	Simulates RAM power confirmation
OSCAFC-T/	Simulates A clock failure
OSCBFC-T/	Simulates B clock failure

Table 6.5  
RCU TEST OUTPUTS

<u>Name</u>	<u>Function</u>
RAUTØ-T/	Indicates RCU in automatic mode
TRTM-T/	Indicates transient fault registered
CFAU-T/	Indicates confirmed general fault
RFAU-T/	Indicates confirmed general fault
RØFF-T/	Indicates DPA in off condition
OSCAF-T/	Indicates Clock A failed
OSCBF-T/	Indicates Clock B failed
CLKAØNT	Indicates Clock A selection
CLKBØNT	Indicates Clock B selection

Note that both inputs and outputs are bilevels

## 7.0 PACKAGING

The DPA will be packaged as a series of modules, each of which plugs into the adjacent module. These "internal" connectors provide the data bus and RCU to/from other unit intercommunication within the DPA. In addition, some units will have test connectors and the IOU will, of course, have the several connectors necessary for inputs and outputs. The RCU will also have a connector for inputs and outputs and for system power input (See Section 8.0).

All modules have the standard dimensions of 6 in. x 8 in. (15 x 20 cm). They vary in thickness and weight according to function. These figures are summarized in Table 7.1.

Table 7.1

DPA UNIT SIZE AND WEIGHT (ESTIMATES)

<u>Unit Type</u>	<u>Thickness (in/cm)</u>	<u>Weight (lb/kg)</u>
ACU	1.6/4.1	1.6/0.72
IOU (MAX I/O)	1.2/3.0	1.2/0.54 (1.0/0.45 for min)
RCU	3.2/8.1	3.2/1.45
RAM (256)	0.8/2.0	0.8/0.36
RAM (512)	0.8/2.0	0.9/0.41
ROM (1024)	0.8/2.0	0.8/0.36
ROM (2048)	0.8/2.0	0.9/0.41
PWM (1024)	1.2/3.0	1.4/0.63
PWM (2048)	1.6/4.1	1.8/0.82
APM (1024)*	0.8/2.0	0.8/0.36
APM (2048)*	0.8/2.0	0.9/0.41

\*Ground use only

The modules (slices) are stacked together to form the DPA. Each slice has mounting feet on one of its 8 in. (narrowest) surfaces. The slices are held together by through bolts and end covers. These add 0.8 in (2.0 cm) to the length and 0.6 lb plus 0.05 lb/in of stack length (0.27 kg plus 0.009 kg/cm to the weight of the DPA.

Using these figures, one can calculate the weight and size of any particular DPA configuration.

## 8.0 POWER

The DPA requires the following voltages at a composite regulation of +5% from an external source:

+5 VDC	-	logic (all units)
+15 VDC	-	IOU, PWM
-15 VDC	-	IOU, RCU, APM, PWM
+28 VDC	-	Relay switching (all units)

The power requirements per unit are as follows: (note that only one ACU, IOU or RCU may be on at a time, but more than one of each type of memory unit may be on simultaneously).

<u>Unit Type</u>	<u>Power Consumption (watts)</u>
ACU	6.4
IOU	7.0 max (6.0 min)
RCU	18.7
RAM (256)	0.9
RAM (512)	1.0
ROM (1024)	0.5
ROM (2048)	0.6
PWM (1024)	5.0
PWM (2048)	7.5
APM (1024)*	9.1
APM (2048)*	15.6

\*Ground use only

The APM power (all but ~ 4 watts) is from -15 VDC. The IOU power includes 0.3 W of +15 VDC and 0.7 W of -15 VDC. The RCU uses 0.2W of -15VDC. The PWM power is estimated at 1/3 from each of +5 VDC, +15 VDC and -15 VDC. Power consumption from +28 VDC is essentially nil, except when units are being reconfigured. The max and min figures for the IOU refer to maximum and minimum input/output capabilities. All power figures are averages. Peak consumption may be up to 15% higher.

G L O S S A R Y

ABS	ATA Bootstrapping Test Subroutine*
ACS	Attitude Control Subsystem
ACU	Arithmetic and Control Unit
A/D	Analog-to-Digital (converter)
ADD	Add to A-register #
ADQ	Add to Q-register #
APM	Alterable Program Memory
AM	Applications Module *
AMS	Application Module Scheduler *
ASM	Asynchronous Module*
ATS	Asynchronous Task Scheduler*
AVS	AMS Verification Subroutine*
AU	Arithmetic Unit (or Astronomical Unit)
BITE	Built-in Test Equipment
CCS	CER Check Subroutine*
CER	Control Executive Routine*
CGA	Configurable Gate Array
CM	Center-of-Mass
CMG	Control Moment Gyro
CMOS	Complementary MOS
CMP	Complement #
COPE	Control Processing Electronics
CS	Control Subroutine*
CTI	Cycle Timing Interrupt
CU	Control Unit
D/A	Digital-to-Analog (converter)
DBS	Data Bus
DPA	Digital Processor Assembly (See COPE)
DSN	Deep Space Network
DVD	Divide #
EBR	Executive Bootstrap Routine*
EMI	Electro-Magnetic Interference
ERP	Equivalent Received Power
ETS	Executive Tracer Subroutine*
EXEC	Executive *

\* - Software

# - Instruction

## GLOSSARY (continued)

FI	Fault Interrupt
FMECA	Failure Mode, Effects and Criticality Analysis
FOV	Field-of-View
FSS	Fine Sun Sensor
FVS	Fault Signal Verification Subroutine*
HCU	Hardcore Unit
IC	Integrated Circuit
I/O	Input/Output
IOU	Input/Output Unit
IRU	Inertial Reference Unit
LDA	Load A-register #
LDQ	Load Q-register #
LDT	Linear Differential Transformer
LDX	Load X-register #
LED	Light-Emitting Diode
LSB	Least-Significant Bit
LSI	Large Scale Integration
MCS	Memory Checking Subroutine*
MeV	Million Electron-Volt
MOS	Metal-Oxide Silicon
MPP	Multi-Purpose Processor
MPY	Multiply #
MSB	Most-Significant Bit
MSI	Medium Scale Integration
PBS	Primary Bootstrap Subroutine*
PDB	Peripheral Data Bus
PLA	Programmable Logic Array
PROM	Programmable ROM
PS	Priority Scheduler *
PWM	Protected Write Memory
PWS	PWM Write Subroutine*
RAM	Random Access Memory
RCS	Reaction Control System
RCU	Reconfiguration Control Unit
RF	Radio Frequency
RIG	Rate Integrating Gyro
ROM	Read-Only Memory
RTG	Radioisotope Thermo-electric Generator
RWA	Reaction Wheel Assembly

### GLOSSARY (continued)

SBS	Secondary Bootstrap Subroutine*
SH	Suspension Handler*
STA	Star Tracker Assembly (or Store A-register #)
STQ	Store Q-register #
STX	Store X-register #
SSI	Small Scale Integration
SUB	Subtract from A-register #
SUQ	Subtract from Q-register #
TMR	Triple-Modular Redundant
TTL	Transistor-Transistor Logic (also T <sup>2</sup> L)
TVC	Thrust Vector Control
UHF	Ultra-High Frequency
VDE	Valve Drive Electronics
WASS	Wide Angle Sun Sensor
$\Delta V$	Delta-Velocity (correction)