JPL PUBLICATION 78-67

Fault-Tolerant Building-Block Computer Study

David A. Rennels

July 15, 1978

Prepared for
Naval Ocean Systems Center
by
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL PUBLICATION 78-67

Fault-Tolerant Building-Block Computer Study

David A. Rennels

July 15, 1978

Prepared for
Naval Ocean Systems Center
by
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

CONTENTS

Figures

Tables

ABSTRACT

The development of ultra-reliable core computers is a starting point for improving the reliability of complex military systems. Such computers can provide reliable fault diagnosis, failure circumvention, and, in some cases serve as an automated repairman for their host systems.

This report describes a small set of building-block circuits which can be implemented as single VLSI devices, and which can be used with off-the-shelf microprocessors and memories to build Self Checking Computer Modules (SCCM). Each SCCM is a microcomputer which is capable of detecting its own faults during normal operation and is designed to communicate with other identical modules over one or more Mil Standard 1553A buses. Several SCCMs can be connected into a network with backup spares to provide fault-tolerant operation, i.e. automated recovery from faults. Alternative fault-tolerant SCCM configurations are discussed along with the cost and reliability associated with their implementation.

EXECUTIVE SUMMARY

Reliability is a continuing problem in military electronic systems. The cost of electronic system failures shows up as reduced operational readiness and requirements for logistics supply chains and maintenance personnel. It is estimated that the cost of supporting an electronic system for its life is often more than its initial procurement cost.

Today both the knowledge and the technology exist to build highly reliable computers at only small penalties of size, weight and cost. The reliability of these computers is achieved through a redundant or fault-tolerant architecture. VLSI technology provides the capability of putting large amounts of circuitry in small and inexpensive packages. By using a standby redundant architecture in which unpowered elements of the computer are spare, computer system reliability can be improved in increments by adding spares. The long term potential result is systems which are sufficiently reliable, that they do not require technician or logistic support for the life of their mission. In the shorter term, systems can be built which utilize a highly reliable core computer which can significantly aid in the diagnosis and maintenance of the entire system.[1]

The purpose of this study is to define and characterize the VLSI building blocks required to combine existing microprocessors and memory into a wide variety of self-checking and fault-tolerant computing systems. Fault tolerance is the ability to continue correct operation in the presence of failures. Self-checking circuits are capable of detecting their own malfunctions. This study has resulted in the definition of four VLSI circuits which allow the construction of a single or a distributed fault-tolerant computer system. The four building-block circuits are: 1) an error detecting (and correcting) Memory Interface, 2) a programmable Bus Interface, 3) a Core Building Block, and 4) an I/O building block. These circuits interface with two commercial microprocessors and commercial memory to form a Self-Checking Computer Module (SCCM). This computer operates just like any regular microcomputer but additionally it signals its own malfunctions and can disable its outputs upon detection of an internal fault. It is designed to communicate with other identical modules over one or more Mil Standard 1553A buses.

The Self-Checking Computer Module (SCCM), which is constructed using the VLSI building-block circuits, is itself a much larger building block which is combined with other SCCMs to form a variety of fault-tolerant computing systems. Examples of fault-tolerant SCCM configurations are shown in Figure 1. The first is a standby redundant configuration. A single computer (SCCM) is backed up by one or more spares. Upon failure of the primary SCCM, a backup spare automatically takes over the ongoing computations. The second configuration represents a network of SCCMs operating as a distributed system. In this case, redundant spare modules can be employed to provide automated fault recovery in critical functions as designated by the system designer. This type of configuration is applicable to avionics and shipboard control systems. In the third configuration, a number of SCCMs perform the same computations simultaneously. Their outputs are voted in peripheral devices. This type of configuration is used for extremely high reliability applications which are human-life dependent, such as commercial aircraft control.[2]

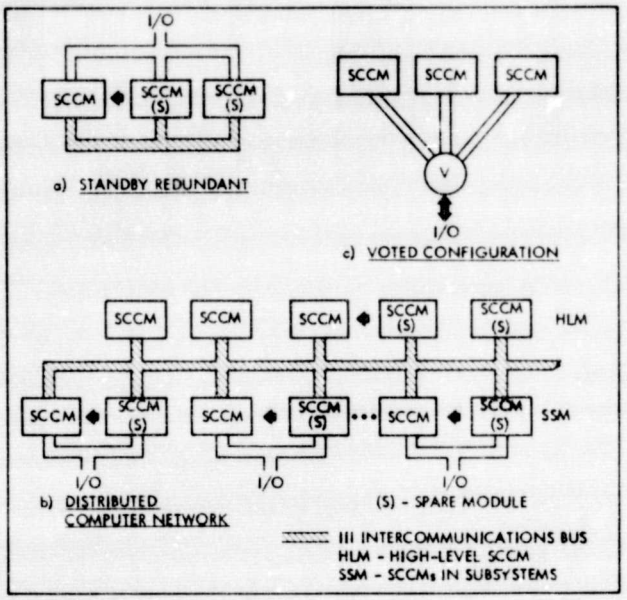


Figure 1. Fault-Tolerant SCCM Configurations

In summary, the important attributes of this building-block approach to fault-tolerant computing are:

(1) Using the four VLSI building blocks, Self-Checking Computer Modules can be constructed from a variety of commercial microprocessors and memories.

(2) The self-checking property of the SCCM allows these machines to instantly detect and signal internal faults, thus allowing straightforward implementation of automated recovery by backup spares.

(3) Using the SCCMs as building blocks allows the system designer to choose from a wide variety of system architectures. He is allowed full flexibility in the tradeoff between redundancy and performance in adding or deleting computers in the system.

The following report describes the individual VLSI building blocks, the resulting Self-Checking Computer Module (SCCM), fault-tolerant configurations of several SCCMs, and finally, an evaluation of the cost and effectiveness of this approach.

I. INTRODUCTION

Reliability and consequently maintenance are a continuing problem in complex military systems. The cost of failures shows up in many ways, including reduced operational readiness, and high dollar costs associated with the large number of personnel involved in maintenance and logistics. It is estimated that costs of ownership often exceed procurement costs in major electronic systems. It is likely that life-cycle costs can be significantly reduced by increasing system testability, maintainability, and adding self-repairing features in the early stages of a system design. A moderate increase of initial hardware costs can yield improved system reliability and reduce maintenance costs for a system's operational lifetime.

The computers within a system provide the starting point for automated maintenance. If computer reliability is assured, the computers can be used for 1) subsystem testing and failure diagnosis, 2) automatically replacing failed subsystems with spare parts, or 3) where no backup spares are available, modifying on-board processing to account for the degraded subsystem state. Stated in another way, the computer becomes an automated repairman which may bring within reach the ultimate goal of maintenance-free systems. To achieve the level of computer reliability required for this goal, fault-tolerant design techniques and extensive use of VLSI must be employed.

The use of VLSI circuitry enhances the basic component reliability, and combining reliable components with fault-tolerant design can lead to failure-free operation. Using fault-tolerance techniques, spare modules are included in a computer which are automatically substituted for faulty modules when a fault occurs. Discarded modules can be replaced by a repairman at regularly scheduled maintenance intervals without disruption in service. The number of spares can be adjusted to provide fault-free operation over various required time intervals. Fault-tolerant computer design is a mature discipline[3], and the use of VLSI technology makes the cost of such machines relatively inexpensive.[4,5]

The purpose of this study is to define and characterize the VLSI building-block circuits required to combine existing microprocessors and memories into a wide variety of fault-tolerant computing systems. The study has resulted in the definition of four VLSI building-block circuits which allow the construction of single or distributed (multiple) computer systems. These systems are fault-tolerant, and thus have the ability to continue correct computation in the presence of failures and transient malfunctions. Specifically, the building-block circuits are connected with existing ("off-the'shelf") microprocessors and memory devices to form Self-Checking Computer Modules (SCCM). Each SCCM contains a computer and the circuits necessary to communicate with other (SCCM) computer modules or with dedicated I/O circuits. Each SCCM is also Self-Checking in that it is capable of detecting internal hardware faults concurrently with normal operation. It generates fault-alarm signals so that recovery can be implemented and other, redundant, (SCCM) computer modules can take over in case of failure.

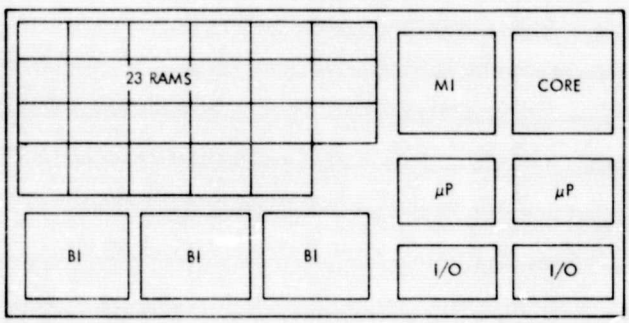
The four building-block circuits are designated the Memory Interface, Programmable Bus Interface, I/O Block, and Core Block. They provide the following important properties:

1. They can be used with a variety of existing microprocessors. This means that standard computers with existing software can be used in constructing building-block computer systems.

2. The building-block concept is oriented toward current and future standards through the use of standard interfaces and the ability to accept different microprocessors and memory.

3. Computers can be arranged in a distributed configuration. Modules can be added to augment performance or to provide redundancy for fault-tolerant operation.

A typical Self-Checking Computer Module is shown below in Figure 2. It provides a computer building block with a great deal of computing power. A 16-bit processor with instruction cycles in the range of 1-2 microseconds would be provided along with 32 thousand words of memory.



MI - MEMORY INTERFACE BUILDING BLOCK
BI - BUS INTERFACE BUILDING BLOCK
CORE - CORE BUILDING BLOCK
I/O - I/O BUILDING BLOCK
μP - MICROPROCESSOR

Figure 2. A Self-Checking Computer Module

Typical packaging would be a 50 square inch multilayer board containing 23 commercial RAMs and two commercial microprocessors. The building-block circuits are one Memory Interface (MI), one Core circuit (C), three Bus Interfaces (BI), and two I/O devices (IO).

A description of the individual building blocks and the resulting self-checking computer modules is presented in Section 2. This is followed by a description of their use in multicomputer networks, presented in Section 3. Upon definition of the building blocks, a study was conducted to estimate the cost, physical characteristics, and the reliability of the building-block computers. This is presented in Section 4 and summarized below.

In order to evaluate the cost and reliability of this approach, we postulated a hypothetical Baseline Computer which has performance equivalent

to the SCCM, but does not have any of its fault-tolerant properties. This baseline machine was estimated to cost between $8,000 and $10,000 when constructed with high-rel VLSI parts. The Self-Checking Computer Module was then compared with the baseline computer. The relative increase in cost for its fault-tolerant properties (over the baseline machine) was tabulated, along with the relative improvement in reliability. An example is shown in Table 1. For various design options, the expected percentage of failures is shown for an ensemble of machines.

	Relative Cost	Expected Percent Failures in Ensemble of Machines in:		
		6 Months	1 Year	2 Years
Baseline Computer	1	22%	39%	63%
Self-Checking Computer Module (SCCM)	1.5	6%	16%	40%
SCCM plus:				
1 Spare	3	0.5	2%	16%
2 Spares	4.5	<0.1%	0.5%	6%

Table 1. SCCM Cost and Reliability Example

To assess the incremental cost of an SCCM, we examine the additional features which must be included in the Baseline Computer. The additional logic for self-checking circuitry adds about 15% to the complexity of the Baseline Computer. This property is essential for fault-tolerant operation since prompt and dependable detection of faults is necessary for automated recovery. Each SCCM also contains error correcting (SEC/DED) codes in memory and a spare replacement bit such that most single faults in memory can be corrected. Since the memory is by far the least reliable portion of the SCCM (using current technology), single-fault recovery in this area significantly improves the reliability of the whole module. More common than hard failures are transient mistakes caused by erroneous bits stored in memory. Most of these errors are also corrected by this error correction capability which represents an addition of 25% to the complexity of the SCCM.

Thus the SCCM is about 1.5 times as expensive as a conventional computer of similar capability but, due to its internal memory fault correction, it is expected to have several times less failures in the field. The self-checking property allows the SCCM to be easily connected with one or more backup spares which provide automatic fault recovery. In this case, the expected failures can be reduced one or more orders of magnitude.

When spares are used, their number is chosen to assure reliable operation over some specified time interval at which scheduled maintenance is performed (e.g. 6 months, 2 years, etc.). At the time when the maintenance person arrives, the computer system indicates which SCCM modules have been discarded as faulty and automatically replaced with spares. He then substitutes new SCCMs for the faulty modules without disrupting computer service. In the future, when the cost of VLSI circuitry drops even farther, it is expected that sufficient redundancy can be included so that the core computers will function properly over the entire operational life of their host systems without any external maintenance.

The relative increase in internal gates and registers to provide a self-checking and fault-tolerant computer is relatively constant over various levels of circuit integration (i.e. SSI, MSI, LSI, VLSI). VLSI implementation makes this type of design more attractive for several reasons. First, packaging is more efficient. When a computer is partitioned into chips, pin limitations often do not allow full utilization of the potential gates inside. Extra space is often available for a nearly "free" implementation of self-checking logic. A side benefit is that the self-checking chips are more easily tested by the manufacturer. The general reduction in circuit costs associated with VLSI similarly reduces the incremental costs of fault tolerance, i.e. checking circuitry and spares. In past computers, fault tolerance cost an additional $100,000 or more and was only applied to one-of-a-kind problems such as space missions. With VLSI technology, this incremental cost is reduced by an order of magnitude making fault tolerance applicable to a much wider range of applications. This report is directed toward this area of LSI and VLSI devices which can currently be produced. Extrapolating to future VLSI developments, the cost of fault-tolerant computing will be reduced by an additional order of magnitude. It is expected that self-checking computers will be packaged on a single chip along with redundancy for local fault recovery at a cost of a few hundred dollars.

II. THE SELF-CHECKING COMPUTER MODULE (SCCM)

The Self-Checking Computer Module, as previously described, is a small computer which is capable of detecting its own malfunctions. It contains I/O and bus interface logic which allows it to be connected to other SCCMs to form fault-tolerant configurations. The SCCM contains commercially available microprocessors, memories, and four types of VLSI building-block circuits as shown in Figure 3. The building blocks are: 1) an error detecting (and correcting) Memory Interface (MI-BB), 2) a programmable Bus Interface (BI-BB), 3) a Core building block (Core-BB), and 4) an I/O building block (IO-BB).

The building-block circuits control and interface the various processor, bus interface, memory, and I/O functions to the SCCM's internal bus. Each building block is responsible for detecting faults in its associated circuitry and then signaling the fault condition to the Core building block by means of an internal fault indicator. The Memory Interface Building Block implements fault detection and correction in the memory, as well as providing detection of faults in its own internal circuitry. Similary, the Bus Interface and I/O Building Blocks provide intercommunications and input-output functions, along

with detecting faults within themselves and their associated communications circuitry. The Core Building Block checks the processing function by running two CPUs in synchronism and comparing their outputs. It is also responsible for fault collection and fault handling within the module.

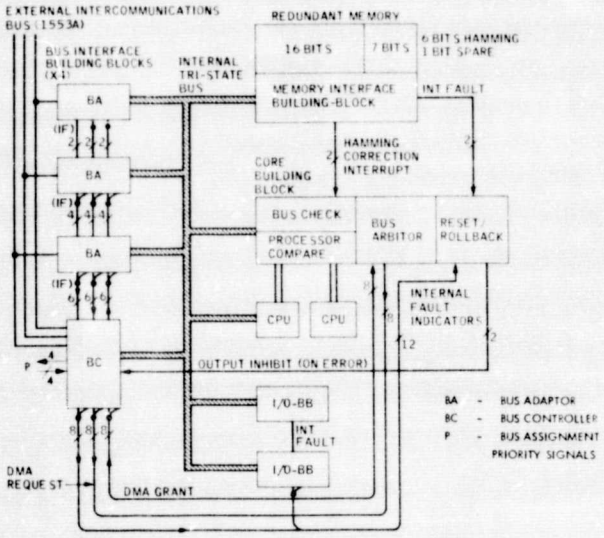


Figure 3. Self-Checking Computer Module Architecture

The Core Building Block receives fault indicators from the other building-block circuits and also checks internal bus information for proper coding. Upon detecting an error, the Core-BB disables the bus interface and I/O functions, isolating the SCCM module from its surrounding environment. The Core-BB can optionally 1) halt further processing until external intervention, or 2) attempt a rollback or restart of the processor, or when fitted, 3) initiate a memory reload from a local nonvolatile store and execute a program restart. Repeated errors result in the disabling of the faulty computer module by the Core Building Block. Recovery can be effected by an external SCCM which is programmed to recognize the lack of activity from an SCCM and take over the ongoing computations.

An important attribute of the building blocks is that they are interconnected via the internal processor-memory bus. All I/O, external bus transmissions, reconfigurations, and external diagnoses are commanded by reading from or writing into out-of range addresses. This use of "memory-mapped I/O" avoids dependence on processor-specific I/O operations, and thus allows use of a wide range of existing microprocessors in the SCCM.

The following is a brief description of the building-block circuits.

The Memory Interface Building Block (MI-BB)

The fault-detecting and correcting MI-BB interfaces a storage array (consisting of a redundant set of memory chips) to the SCCM internal bus. It provides Hamming correction to damaged memory data, replacement of a faulty bit with a spare, parity encoding and decoding to the internal bus, and detection of internal faults within its own circuitry.

The MI-BB needs only to be capable of detecting errors to satisfy the requirement of an SCCM. However, memory is the source of the largest number of failures within the SCCM, and single-fault repair in this area will greatly improve the reliability of the whole module, even though the basic computer module is treated as a replaceable (throw away) item with backup spares. A block diagram of the memory interface building block is shown in Figure 4.

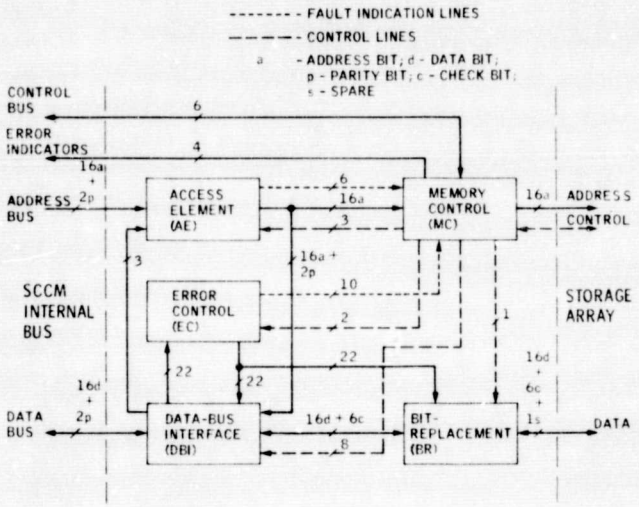


Figure 4. The Memory Interface Building Block

The Access Element (AE) provides the address parity checking and decoding required to select a memory module (storage array plus MI-BB). It names and validates the incoming address by means of a self-checking parity checker circuit.

The Error Control element is responsible for generation of Hamming code check bits and syndromes, byte-parity generation and checking (for the SCCM internal bus), and error analysis. The circuits used in the EC are also self-testing. A single-bit error is corrected by decoding the syndrome generated from the word read from memory, in order to localize the faulty bit. The correction is performed by complementing the fault bit.

The error analyzer collects various error indications such as single error, double error, and circuit error, which are recorded in an Error Status Word and which can be transmitted over the bus on system demand.

The Bit Replacement (BR) element performs the reconfiguration of the storage array. It contains a multiplexer circuit which can replace any one bit plane in the memory with a single standby spare. The bit to be replaced is specified by external command.

The Data Bus Interface (DBI) contains a Memory Data Register and the tri-state drivers and receivers used to interface with the SCCM internal data bus. Bit inversion for Hamming correction is performed in the Data Register.

The Memory Control (MC) element receives commands from the SCCM internal control bus which

specifies "read" and "write" operations. For addresses less than 61,440, the commands are interpreted as normal memory operations. "Read" and "write" instructions with addresses larger than 61,440 are reserved for memory-mapped I/O. A set of these out-of-range addresses is reserved for commands to the MI-BB. Among these commands are: 1) read error status word, 2) read error position of faulty word, 3) read address of last error, 4) reset, 5) disable correction, 6) read redundant check bits, 7) replace a bit plane with a spare, and 8) set soft name.

From preliminary designs, the complexity of the Memory Interface building block is estimated to be equivalent to 2000 gates. This represents a small failure rate with respect to the storage array and is readily implemented as a single LSI building block.

The Bus Interface Building Block (BI-BB)

The Bus Interface Building Block provides the mechanism by which information is transferred between SCCMs via an intercommunications bus system. This external bus system consists of several redundant buses and is being designed to utilize MIL STD 1553A communications formats. Each BI-BB can be microprogrammed to function as either a Bus Controller or Remote Terminal (designated Bus Adaptor) for a single 1553A bus. Several BI-BBs are employed in each SCCM so that each computer module can communicate over several buses simultaneously.

The Bus Controller and Adaptor functions provided by the BI-BBs are much more powerful than those normally implemented for 1553A controllers and terminals. These BI-BBs are capable of moving data directly between the memories of the SCCMs attached to a given data bus with a minimum of software support. The controller and adaptors on a given bus operate together in a relatively autonomous fashion similar to the data channels on much larger machines, as described below.

The SCCM which controls an intercommunications bus contains a BI-BB which is microprogrammed to be a Bus Controller. When it wishes to initiate a data transfer between the memories of the SCCM modules on its bus, it alerts the Bus Controller.

The Bus Controller reads a control table from its host SCCM's memory which specifies the source and destination of information required for the bus transfer along with the length of the transmission. The Controller then broadcasts the appropriate commands over the bus system to "set up" the transmitting and receiving adaptor circuits. It monitors the transfer of information, records status messages, and notifies the host SCCM upon completion of the transfer.

BI-BBs in other SCCMs connected to the same intercommunications bus are microprogrammed as Bus Adaptors. These Bus Adaptors serve as remote terminals on the bus. The Bus Controller, in setting up a transfer, specifies one adaptor as a data source. It then specifies one or more Bus Adaptors as data acceptors and names the data to be moved.

The source Adaptor then finds and extracts the specified information from its host SCCM's memory, using cycle-stealing, and places this information on the bus. Simultaneously, the acceptor Adaptor(s) takes this information off the bus and loads it into its host SCCM's memory, via cycle-stealing.

An SCCM can contain several bus adaptors to provide an interface to a number of redundant external buses. Communication can occur simultaneously over as many as three buses with an SCCM without conflict (time delays) seen on any bus. A Bus Adaptor cannot initiate a bus transfer, but only responds to the commands of a Bus Controller. Provision is made for sending discrete commands through Bus Adaptors such as, power on, power off, halt, interrupt, reconfigure, etc.

The functioning of the Intercommunications Bus System is further described in a following section. A block diagram of the BI-BB is shown in Figure 5. It consists of five major elements, a Manchester/NRZ translator, a Microprogram Control Unit, a Control ROM, a Data Path Element, and a DMA (direct memory access) Controller.

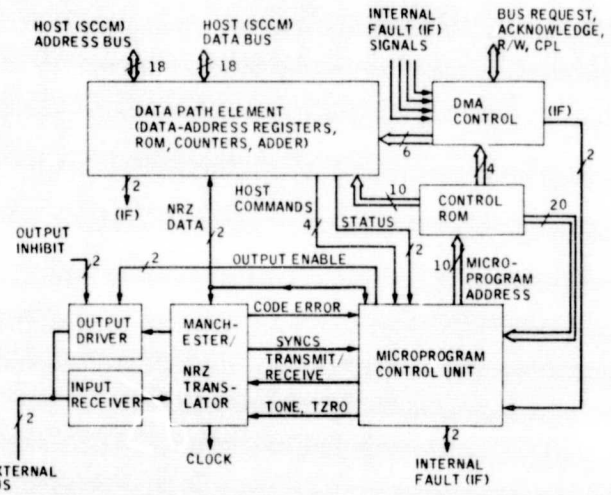


Figure 5. The Bus Interface Building Block

The Manchester/NRZ Translator translates incoming Biphase Manchester into commands and data by supplying a bus-synchronized clock, command and data word-sync indicators, NRZ data, and parity and Manchester-error detection signals. It will also accept NRZ data, encode it, and output Manchester data for bus transmission, along with the associated command and data sync signals.

The Microprogram Control Unit (MCU) is a microprogram sequencer. A microprogram location counter is started at one of several fixed addresses by command sync, data sync, or a host processor command (detection of an out-of-range address). The location counter proceeds through sequential addresses or branches on the basis of incoming data, internal flags, or other internal circuit conditions. The microprogram sequencer is programmed to generate a unique set of address sequences for each type of incoming bus command, data sequence, or computer command. This output sequence is then mapped through a Control ROM to generate the

detailed control signals required to drive the Data Path, MCU, and DMA Control Elements.

The Control ROM maps the microprogram address sequence into control signals for the various circuit elements.

The Data Path element contains 1) registers necessary to buffer addresses and data, 2) ROM to store memory protection bounds, data keys, and table addresses, and 3) an arithmetic logic unit for addressing computations. This circuit is not unlike existing bit slice processors, with the exception that serial-parallel conversion registers, ROM, and several holding registers are required for the unique bus interface and DMA functions.

The DMA Control element is responsible for obtaining control of the host SCCM's internal bus and transferring data between the BI-BB and the host SCCM's memory.

The fault detection techniques employed in the Bus Interface Building Block are based on parity coding to protect memory information and duplication with morphic comparison for most of the logic circuitry.[6] Preliminary designs indicate that this building block will have complexity equivalent to 7,000-10,000 gates.

The I/O Building Blocks

Input-output requirements of host systems vary widely in voltage ranges, currents, and timing parameters. The approach best suited to building-block development is to provide a standard set of functions which serve a majority of general applications. The user is required to supply any additional functions unique to his applications.

To be consistent with the SCCM design, all building blocks must provide memory mapped I/O. That is, each I/O building block must recognize its identification and the function being requested from an out-of-range address appearing on the host SCCM's internal address bus. Data for output or input is transferred over the data bus in response to a processor write or read to the specified address.

Candidate I/O functions are: 1) 16-bit parallel data in and out, 2) 16-bit serial data in and out, 3) a pulse sampling circuit, 4) a pulse counter, 5) a pulse generator, 6) an adjustable frequency generator, 7) an analog multiplexer with A/D converter, and 8) a high-rate DMA channel.

The density of VLSI technology is sufficiently high that a number of I/O functions can be supplied on a single chip. The specific function which is required can be activated by connecting pins. This approach can reduce the inventory of different I/O building blocks to one or two.

The Core Building Block (Core-BB)

The Core-BB is responsible for: 1) detecting CPU faults by synchronizing and comparing two duplex CPUs, 2) collecting fault indications from itself and other building blocks, and 3) disabling its host SCCM upon detection of a permanent fault. Three options are provided for attempted recovery from transient faults. These are: 1) Stop at first fault indication; wait for outside help; 2) Roll back at first fault indication; stop if the fault recurs; 3) Reload memory and restart; stop if fault recurs. In all cases, Bus Controller and I/O outputs are inhibited as long as the SCCM is suspect; e.g. before a rollback or restart has been successfully completed. Specific functions of the Core-BB are listed: 1) Compare two CPUs for disagreement; 2) Parity encode CPU output for internal bus transmission; 3) Check parity on the internal bus; 4) Recognize Core-BB commands which can be sent from an external module via a bus adaptor as out-of-range address (these are commands to halt and inhibit outputs, restart, and enable outputs of the receiving module); 5) Allocate the internal tristate bus among several DMA requests from the Bus Controller, Adaptors, and I/O-BBs; 6) Detect internal faults within the Core-BB; 7) Collect internal fault indications from all building blocks within the SCCM; 8) Disable SCCM output under fault conditions; 9) Provide optional rollback/restart compatibility for optional transient fault recovery; and 10) Halt computation on recurring faults.

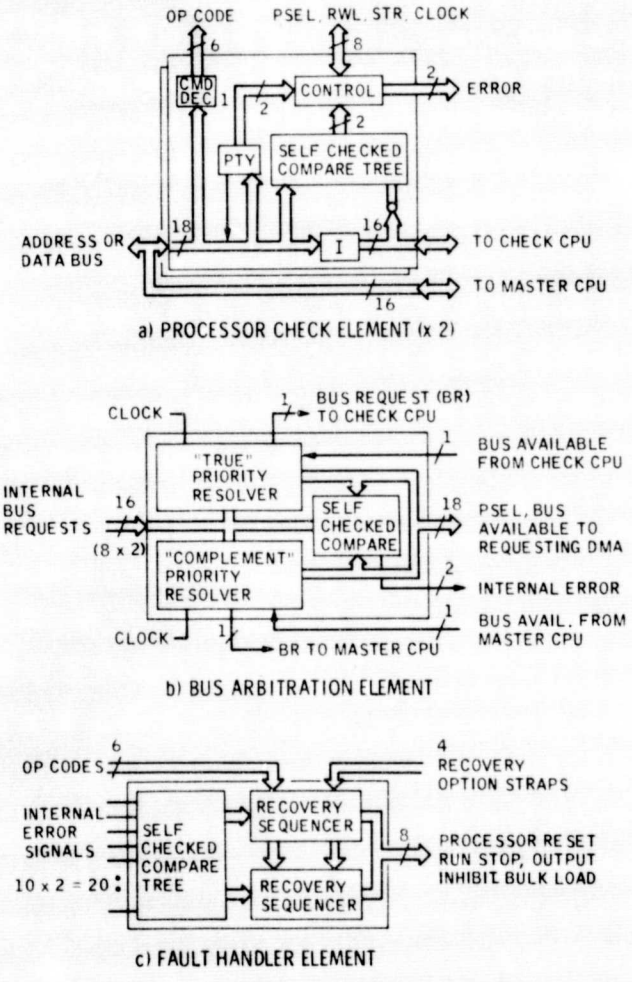


Figure 6. The Core Building Block

The Core-BB consists of three elements as shown in Figure 6. The Processor Check element serves three functions: 1) to compare the outputs of two synchronous CPUs, 2) to encode and check internal bus parity, and 3) to recognize and decode commands sent to the Core-BB through the internal bus. It contains self-checking parity checkers, a duplex command decoder, and morphic reduction trees.[6]

The Bus Arbitration element accepts two-wire bus request signals from the various DMA controllers on other BBs. It obtains release of the bus by the CPUs, and grants access to requesting BBs on the basis of hardware priority. The Bus Available signal is sent as a two-wire indicator to each requesting BB.

The Fault-Handler element accepts morphic fault indicators from the other BBs and from within the Core-BB. It reduces these to a single two-wire master fault indicator which indicates a fault somewhere in the SCCM. This fault indicator triggers the removal of power from bus controller and output drivers, isolating the computer module from the rest of the system. Duplex Recovery Sequencers are employed to implement optional transient recovery sequences. They are checked with a self-checking comparator.

The Building-Block Designs are all directed toward developing a computer module which removes itself from operation whenever a fault occurs inside. The next section deals with the use of these modules in a computer network.

III. SYSTEM CONFIGURATIONS

The SCCMs can be used in a number of different fault-tolerant configurations to support a variety of applications. This includes single computer applications which employ standby redundancy (i.e. a single computer protected with backup spares) and hybrid redundancy (i.e. several computers execute the same program and their outputs are voted; additional spares may also be employed). The SCCMs may also be combined into distributed computer networks in which the individual computing functions can be protected with standby or hybrid redundancy.

A distributed computer network architecture has been developed at the Jet Propulsion Laboratory for control and data handling applications, which allows SCCMs to be connected into a variety of configurations. This architecture, designated the Unified Data System (UDS), has been constructed in the form of a breadboard system. Software has been developed, and the results have been widely reported.[6,7,8] This distributed computing architecture provides a framework for a number of computers to work together, performing a collection of different dedicated computing tasks. Individual computer modules may be protected by standby redundancy or voting, and thus the architecture provides a model for single computer configurations by considering a single internal computer with its redundant protection, or a model for distributed computer fault tolerance by implementing the complete processing ensemble.

The next section describes the hardware and intercommunications structure of the UDS architecture. This is followed by a brief description of the fault-tolerant configurations that can be constructed.

The UDS Architecture Using Self-Checking Computer Modules

A fault-tolerant UDS architecture consists of a set of Self-Checking Computer Modules connected by a redundant set of intercommunications buses as shown in Figure 7. There are two types of modules: Terminal Modules (TM) and High-Level Modules (HLM).[9]

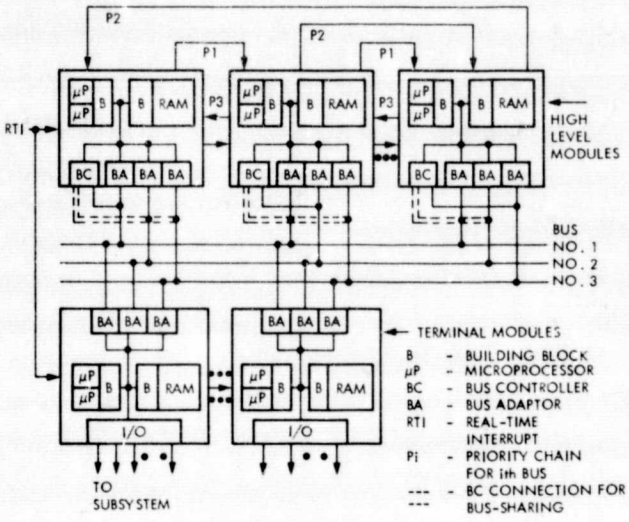


Figure 7. A Distributed SCCM Architecture

Terminal Modules (TM) are SCCMs located within various subsystems which are responsible for control and data gathering within their associated subsystem. The TM contains two microprocessors (μP), memory (RAM) with an MI-BB, I/O-BBs, a Core-BB, and several BI-BBs configured as Bus Adaptors (BA). The TM interfaces with the other modules in two ways: 1) It receives a single Real-Time Interrupt (RTI) which is common to all modules and which is used for timing and synchronization, and 2) Each TM contains a Bus Adaptor interface to each of several intercommunications buses. Data words can be entered or extracted from the memory of the TM computer using Direct Memory Access (DMA) techniques. Each BA can be commanded over its bus to fetch or deposit data into the TM memory. A TM cannot initiate bus communications but it actively supports DMA transactions into and out of its memory. An external High-Level Module enters commands, data, and timing information into the memory of the TM. The TM delivers information to the system by placing outgoing messages into predetermined locations of its memory, which can then be extracted by the HLM over the bus. The TM can be accessed through several buses simultaneously. The associated BAs provide hardware conflict resolution between competing DMA requests from different buses.

High-Level Modules (HLM) are SCCMs which are responsible for coordinating the processing which is carried out in the remote TMs, for control of intercommunications over the bus system, and for high-level processing such as data compression and decision making. They contain the same internal components as the TMs, augmented by an additional BI-BB which configured as a Bus Controller (BC).

Each BC, which is unique to an HLM, can move blocks of data between the memories of all computer modules connected to its bus via commands to their Bus Adaptors. The BC specifies a source module, one or more destination modules and identifies the data to be moved. The BAs in the addressed modules then move the specified data over the bus between their host SCCM memories.

The computer in the HLM activates its BC by presenting it with the address of a Bus Control Table in the HLM memory. This table specifies the source module, destination modules, data names, and the length of the requested information transfer. The BC initiates and controls the specified transmission, monitors status messages to verify a correct transfer of information, and notifies the HLM computer when it is completed. The BC is the mechanism by which the HLM can coordinate the processing in other computer modules by entering commands into their memories and reading out information to monitor ongoing processes.

The Intercommunication Bus System (IBS) consists of several independent serial buses, each of which provides a bandwidth of approximately 1 megabit. Each bus is connected to one Bus Adaptor in each of the computer modules (HLMs or TMs) to which it is connected. To each bus is assigned a primary Bus Controller whose HLM has complete control over that bus. It can relinquish control over its bus to another HLM under two conditions: 1) it is not powered, or 2) its processor commands release of the bus to lower priority Bus Controllers for a designated time interval. Thus, the set of buses may be operated simultaneously with each bus controlled by a different HLM or with individual buses time-shared between several such modules.

Access to each bus by the various HLMs is based on a fixed hardware priority assignment between Bus Controllers. A daisy-chain structure is utilized for each bus to establish this priority assignment as shown in Figure 7. Modules of higher priority signal release of a bus via its "daisy-chain," which then activates the hardware necessary to allow bus access within modules of lower priority. Thus spare modules can gain access to a bus whose controlling HLM has failed, or if a bus fails, another bus can be shared between two controllers. The individual buses are physically independent; each has its own set of hardware bus access control circuits and a daisy-chain for priority assignments. Therefore, no central bus system controller exists as a potential catastrophic failure mechanism. Similarly, there is no common clock. Each bus uses clock signals generated by the transmitting module. The number of buses within the IBS may be selected to meet requirements of data throughput and redundancy. The concept facilitates reconfiguring throughout a mission if failures occur. In the extreme, a single remaining bus can support essential functions.

The UDS design is oriented toward removing "hard core" items whose failure can cause catastrophic system failure. Intercommunications and clocks have often presented significant problems in this area. These are dealt with in the UDS in the following ways. The buses are made independent to avoid any common failure mechanisms. Each HLM or TM uses its own internal clock, and the buses use the clock of whatever module is transmitting. With independent clocks in each computer module there is also a distributed mechanism for protecting against failure of the common RTI. If two or more independent RTI signals are generated, each module can decide whether the RTI is correct by comparing this signal with their own internal clocks. If an RTI generator fails, the modules will automatically switch to a backup, and if an individual computer clock fails, damage is contained to the faulty module.

Each SCCM, as previously described, is capable of detecting its own faults concurrent with normal operation and will disable itself upon detecting a permanent internal fault before damaged information can propagate beyond the faulty SCCM. This allows automated recovery to be implemented with backup spares.

Fault-Tolerant Configurations

A number of fault-tolerant configurations can be derived from the UDS architecture using SCCMs as HLMs and TMs. Several are described below.

a) Standby Redundant Uniprocessor. This configuration consists of a set of HLMs. One HLM is designated the "primary" module, another is designated a "hot" spare, other HLMs may be employed as additional spares which are maintained in a dormant state. The primary module performs the required computations, generates outputs, and collects inputs for both itself and the "hot" spare module via the intercommunications bus system. The "hot" spare performs identical computations to the primary module but does not produce outputs. At each Real-Time Interrupt these two modules perform cross-checks via the bus to see if the other is functioning properly. Failure of either module triggers recovery by the "good" module. Should either active module detect an internal fault and disable its outputs, the other machine continues the ongoing computations. At a later scheduled time, the surviving machine diagnoses the faulty module and activates a new "hot" spare if additional spares are available. Under special conditions two or more "hot" spares can be assigned to back up the primary module and provide a greater degree of redundant protection.

b) Voted Uniprocessors. For applications demanding extremely high reliability, three HLMs can be assigned to perform identical computations, and each communicates with peripheral devices with a separate dedicated bus. Outputs are voted in the peripheral devices. Backup spare HLMs may optionally be employed to provide hybrid redundancy. This type of configuration provides voting in addition to the self-checking features of the HLMs. It has primarily been applied to systems in which the cost of failure is clearly unacceptable, such as flight control of commercial aircraft and the Space Shuttle.[2]

c) Distributed Systems. Since the computer modules in a distributed system are performing a variety of different functions, the requirements for fault recovery vary in the different modules of the system. Some functions are deemed more critical than others and redundancy is employed in a selective fashion. Some computer modules,

embedded in fully redundant subsystems, may not require fault recovery; others performing critical functions may be very heavily protected.

The HLM which serves as the system executive, and in some cases other HLMs, must survive a fault without interruption in computations. These critical modules are protected by standby redundancy with "hot" spares as described above. There are often other HLMs assigned to functions which are deemed noncritical and which can tolerate a program interruption of up to several seconds. These HLMs do not have dedicated "hot" spares. When one of these modules develops an internal fault, the system executive HLM replaces the faulty module with a "blank" spare, loads the spare's memory, and restarts the internal programs. Since the HLMs have nondedicated connections, a common set of spare modules can back up a number of HLMs performing different critical and noncritical computations.

The Terminal Modules are attached by a number of wires to the specific subsystem in which they are located so they must have dedicated spares which are also connected to the same subsystem. The number of spares is determined by the criticality and failure rate of its associated subsystem. Since a Terminal Module does not have the ability to initiate a bus communication, it can only halt and signal an error. Recognition of a failed TM and commands for reconfiguration are the responsibility of a High-Level Module.

Each controlling HLM is responsible for polling the various modules under its control to determine if a module has isolated itself due to a failure. This polling process can be carried out nearly automatically, using the bus system every few (10-100) milliseconds. (Bus system failures are determined by rerouting suspect messages through a different HLM-bus combination.) The HLM is then responsible for issuing the reload and reconfiguration commands necessary to replace a faulty module with a spare or reinitialize a transient-disturbed module.

These commands are sent to the various UDS modules through one of several redundant Bus Adaptors, which provide the following functions with respect to their associated HLM or TM: 1) power or unpower the internal computer, 2) load or read out memory, 3) halt or start the processor at specified locations, 4) sample error status from the BBs, and 5) send reconfiguration commands to the BBs. Since there are several Bus Adaptors in each UDS module which are connected to independent bus systems, there are redundant paths for carrying out reconfiguration. The Bus Adaptors are powered at all times.

IV. COST AND EFFECTIVENESS OF BUILDING-BLOCK FAULT-TOLERANT COMPUTERS

In order to establish the cost and effectiveness of fault-tolerant SCCM configurations, it is first necessary to examine the properties of individual SCCM modules. The user determines how many are used for a given system and the number of spares which are to be employed, and thus his cost and reliability results are quite system-specific.

Appendix I is a characterization of the cost, weight, power, volume, and reliability of a single SCCM. Also included is a hypothetical baseline computer module which does not have any of the error detection and memory-fault correction capabilities of the SCCM. This baseline computer is presented so that the relative increase in cost for the SCCM can be determined. The cost increase is what one pays for increased reliability and fault tolerance.

The estimates in Appendix I are made for an SCCM containing 1) two 16-bit microprocessors, 2) 32,000 words of RAM, 3) three bus interface building blocks, 4) one Core building block, and 5) one Memory Interface building block. This configuration corresponds to a High-Level Module previously described. (A Terminal Module would typically contain one less Bus Interface and two additional I/O building blocks and is of similar complexity.) The technology is assumed to be hi-rel CMOS/SOS circuits with up to 10,000 gates on a single chip. The results are summarized below in Table 2.

	Cost	Power	Weight	Volume
Baseline Computer Module...	$ 8.6K	6W	3 lbs	18 in^3
Self-Checking Computer Module...	$13.6K	8W	2.2 lbs	23 in^3

Table 2. Computer Module Physical Properties

Absolute estimates of SCCM properties are, at best, an approximation since the VLSI devices have not been developed. Our basis of estimation lies in projection of past and current experience. The relative estimates between the properties of the Baseline and SCCM modules should be more accurate, since they are less dependent upon specific assumptions. In a relative sense, the SCCM is 40% to 50% more expensive, more power consuming, and more voluminous than a nonredundant baseline module. This represents the "front-end" price which is paid in order to reduce life-cycle maintenance costs.

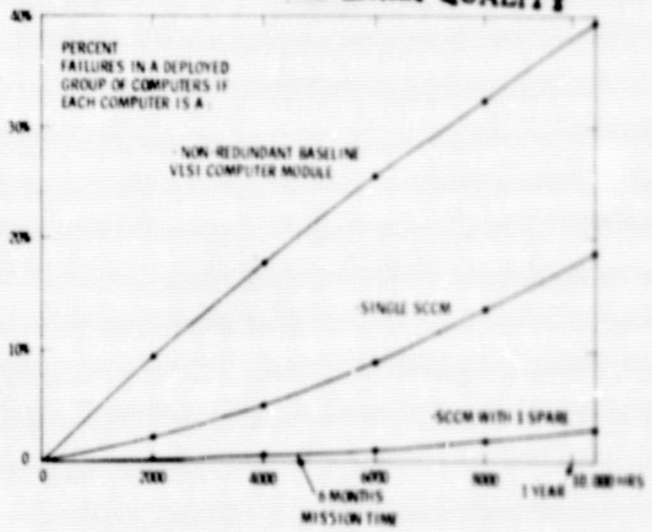


Figure 8. Reliability Improvement Using SCCMs

Figure 8 shows the comparative reliability of the nonredundant baseline computer, a single SCCM, and a duplex pair of an SCCM backed up by a spare.

If a large number of nonredundant computers are deployed in the field, the user can expect a large percentage of them to fail in a year's time. He must then support maintenance personnel and their associated test equipment to correct these failures. At an extra 50% initial investment, SCCMs can be employed which, without backup spares, reduce this failure rate by a factor between 2 and 3. At 3 times the price (an extra $10-$20K per computer) he can employ backup spares and reduce the failure rate 10 to 20 times. Additional spares can be employed to reduce in-field failures to extremely rare occurrences.

Future Trends

Through the use of LSI, the cost of hardware has dropped to the point where a fault-tolerant computer costs much less than a non fault-tolerant computer did only a few years ago. Using LSI circuits, the intrinsic reliability of computer systems has improved greatly, but not enough to provide fault-free operation. This is achieved by fault-tolerant, i.e. self-repairing, architectures which offer fault-free operation of a year or more with current technology. The current cost of fault tolerance is on the order of an extra $10,000-20,000 per computer. This is largely due to the cost of the additional high reliability parts which are many times more expensive than commercial quality devices. Current costs may be significantly reduced in two ways: 1) the use of self-testing logic should make them much more easily tested, thus reducing a large production cost, and 2) using fault tolerance less component screening is required since an occasional failure is automatically corrected.

VLSI circuit development is not static and by the mid-1980s there should be major improvements in this fault-tolerant computer technology. For example, we can expect an SCCM to be packaged on one or two chips at a cost of less than $1000. Components can be expected to be several times more reliable, producing an equivalent increase in the reliable life of the fault-tolerant configurations. One can project conservatively that fault-tolerant machines can be built in a few years which provide 5-10 years of maintenance-free operation at less than $1000 in increased costs.

V. CONCLUSIONS

This study has resulted in the definition and preliminary design of a small set of VLSI building-block circuits, which, if implemented, would allow the user to construct fault-tolerant computing systems out of existing processors in a straightforward fashion. VLSI circuitry is the key to this implementation. The redundant circuitry required for fault tolerance, which was once expensive, can now be obtained in small inexpensive packages.

The VLSI building-block circuits are used to construct small (SCCM) computers which are capable of detecting their own internal faults (and correcting some faults which are most likely to occur). These Self-Checking Computer Modules can then be combined with backup spares to provide automatic recovery from faults. A standard busing system allows these SCCMs to be connected into a variety of fault-tolerant distributed computing networks.

The ultimate goal of this work is maintenance-free systems. By providing an ultrareliable core computing system, it can act as an automated repairman, to provide automated testing and redundancy management in complex systems.

The next phase of this effort will be to do a detailed logic design of the building-block circuits and implement a breadboard consisting of several self-checking building-block computers interconnected by the redundant bus system. This step will be completed in 1979 and will provide the detailed experience necessary for a subsequent VLSI implementation of the building-block circuits.

When this is accomplished, we hope that the system engineer will have in hand the tools for the immediate and routine use of fault-tolerant computing. This building-block development has a near term goal of developing the enabling technology (circuits) for the off-the-shelf use of fault-tolerant computers. It is a first step in a long term goal of using the increasing density and reliability of new VLSI circuit technologies to provide fault-free computing at lower and lower costs.

APPENDIX 1.

CHARACTERIZATION OF BASELINE (NONREDUNDANT) AND SCCM COMPUTER MODULES

There are two approaches to evaluating the fault-tolerant building-block systems which are of value to the potential user. First, there is the absolute approach: "How much does it cost and what are its reliability characteristics?" And secondly, there is the relative measure: "How much more does it cost than an equivalent nonredundant structure, and what is the relative improvement in reliability?" Both approaches will be examined.

The cost and reliability of each computer system is highly dependent on the number of integrated circuits which are required for implementing the building blocks. Two levels of integration are considered in estimating cost and reliability. These correspond to the current and projected state-of-the-art in device and technology as listed below:

1. SSI MSI - Current breadboard technology

2. 10,000 gates/chip - Next generation custom VLSI

It should be recognized that absolute cost and reliability estimation results, at best, in a rough approximation. This is because the VLSI devices have not been developed, and thus our basis of estimation lies in a projection of past and current experience.

The following is an examination of the characteristics of a Building-Block Self-Checking Computer Module. These computers make up the basic modules from which fault-tolerant networks are constructed. In order to evaluate the additional costs of self-checking building-block circuitry, estimates are also included for a

nonredundant computer module of similar capability and constructed out of similar technology.

The parameters to be studied are power, weight, volume, cost, and reliability. Since the computer circuits have yet to be constructed, estimates of these parameters must be regarded as only approximations projected from current experience. Underlying assumptions are given in the following discussion in order to aid the reader in evaluating the validity of the results.

Complexity of Functional Elements

Table A-1 examines the logical complexity of the various component elements which make up 1) a building-block self-checking computer, or 2) an equivalent nonredundant machine. Complexity is determined in terms of equivalent logic gates e.g. 1RAM cell = 1 gate, 1ROM cell = 1/4 gate, 1 bit of parallel-serial shift register = 6 gates) for each element which makes up the computers. It is assumed that each computer has a memory of 32,000 16-bit words. (This is admittedly small for VLSI implementations, but it is expected that memory development will not proceed as rapidly in low power, hardenable technologies required by the military.) An estimated component count is tabulated for each of two implementations. The first assumes that all circuitry surrounding the CPU and Memory will be constructed out of existing SSI and MSI circuits. The second implementation assumes a VLSI technology with on the order of 10,000 gates per chip.

Complexity of Computer Modules

Table A-2a is a tabulation of the component requirements for self-checking high-level and terminal (computer) modules along with equivalent nonredundant computers.

Two patterns become obvious in these results. First, as the level of integration is increased in the building-block circuits, memory becomes the dominant cost in a number of devices. Second, as

ELEMENT	QUANTITY	COMPLEXITY (equivalent gates)	IMPLEMENTATION SSI-MSI	IMPLEMENTATION VLSI (10^4 g/chip) 128 pin pkg.
1. BUILDING-BLOCK, SELF-CHECKING COMPUTER				
a. BUS INTERFACE	2-4	2500-3000 gates 12,000 Bits ROM	200 Chips 3 ROMs (512x8)	1 VLSI
b. CPU	2	8000 gates	2 CPU	2 CPU
c. CORE BUILDING BLOCK	1	1000 gates	150 Chips	1 VLSI
d. MEMORY INTERFACE BUILDING BLOCK (for 32K module)	n	1700 gates	150 Chips	1 VLSI
e. MEMORY	8Kx16 (in)	1 gate/bit	184 RAMs (@4Kx1)	23 RAMs (@ 32Kx1)
f. I/O BUILDING BLOCKS	k	100-600 gates each (10 types)	10-40 Chips (each type)	1 VLSI for 3 or 4 I/O functions concurrently
2. NONREDUNDANT COMPUTER - of similar technology and capability				
a. BUS INTERFACE	2-4	1700-2000 gates 10,000 Bits ROM	130 Chips 3 ROMs (512x8)	1 VLSI
b. CPU	1	8000 gates	1 CPU	1 CPU
c. MEMORY (32Kx16)		1 gate/bit	136 RAMs (@4Kx1)	17 RAMs (@ 32Kx1)
d. BUS ARBITER	1	100 gates	16 Chips	Included in VLSI Bus Interface
e. I/O STANDARD CIRCUITS	k	75-400 gates each (10 types)	8-32 Chips each	1 VLSI for several functions (1 type)

Extra discrete logic folded into custom LSI circuits. Common elements are clock, power supply, and bus drivers.

Table A-1. Complexity of Component Elements

the level of integration is increased, the relative cost of a self-checking computer (over a nonredundant computer) decreases dramatically.

If memory fault detection (but not correction) is employed in the self-checking computer, the cost of self-checking is negligible over the nonredundant machine. If error correction is employed in the memory of the self-checking computer (an additional feature to improve reliability), the increment in cost between machines is primarily related to the cost of SEC/DED codes.

Several authors have demonstrated that self-checking circuitry is relatively inexpensive if used. One reason for this is because system partitioning, i.e., partitioning of a computer into VLSI chips, does not occur on the basis of gate count alone. It is based on separation of computing functions (I/O Processor, Memory Intercommunications) which results in breaking up the computer at points where the number of interconnections and the control interfaces are reasonable. When partitioning is completed, the VLSI chips are seldom used to full capacity, and enough area remains to implement checking circuits at negligible cost on each chip.

Packaging and Cost

The methodology for estimating packaging density is to assume the use of flat packs on multilayer boards. The area for each circuit type is shown in Table A-2b, and the resulting board area is computed for each of the computers which are being examined. The circuit boards require about 1/2 inch of depth, and thus volume is estimated at 0.5 times the circuit area requirements.

Cost is determined on the basis of three items--parts costs, board costs, and assembly costs--as shown in Table A-2c. Assembly costs are related to board area, which is in turn related to the number of pins to be soldered. From our experience, a technician can assemble and inspect on the average of one square inch/hour.

A "TYPICAL" BUILDING-BLOCK COMPUTER MODULE

COMPONENT	NUMBER	AREA (in^2)	COST $K (ICs only)	
1. SSI/MSI VERSION: (14 lbs)				
SSI/MSI	900	300	31.5	
ROMs	9	7	1.4	
CPU	2	3	.4	70.1
RAM	184	138	36.8	+ 20.1*
LSI	0	0	0	90.1K
	1095	448	70.1	
2. VLSI VERSION: (1.4 lbs)				
SSI/MSI	0	0	0	
ROMs	0	0	0	
CPU	2	3	0.4	11.6
RAM	23	17	9.2	2.0*
VLSI	5	25	2	13.6
	30	45	11.6	

NONREDUNDANT COMPUTER MODULE W/ SIMILAR CAPABILITY AND TECHNOLOGY

COMPONENT	NUMBER	AREA (in^2)	COST $K	
1. SSI/MSI VERSION: (8 lbs)				
SSI/MSI	406	135	14.2	
ROMs	9	7	1.4	
CPU	1	2	.2	43.0
RAM	136	102	27.2	11.1*
LSI	0	0	0	54.1K
	552	247	43.0	
2. VLSI VERSION: (1.1 lbs)				
SSI/MSI	0	0	0	
ROMs	0	0	0	
CPU	1	2	0.2	7.0
RAM	17	13	5.2	1.6*
VLSI	4	20	1.6	8.6
	22	35	7.0	

* Board & Fabrication Costs

Table A2-a. Characteristics of Computer Modules

1. VLSI 128 pin pkg. - 5 in^2
2. LSI 64 pin pkg. - 1.5 in^2
3. ROM, RAM some MSI 20-24 pin pkg. - .75 in^2
4. SSI 14-18 pin pkg. - .25 in^2

(avg. area for SSI/MSI = 1/3 in^2)

Table A-2b. Packaging Density

VLSI $400/chip
LSI $200/chip
RAM $200/chip (4Kx1, or 8Kx1)
RAM $400/chip (32Kx1)
ROM $150/chip (512x8)
SSI/MSI $35/chip, avg.

Multilayer board $25/$in^2$

Assembly and Inspection $20/$in^2$

Table A-2c. Cost Model

Note: These estimations do not include the power supply, which is estimated at 4 lbs, 102.5 in^3, $10,000, and 75% efficiency.