D43

AUTOMATIC SOFTWARE FOR CONTROLLING CRYOGENIC SYSTEMS

James W. Rudolph*
Martin Marietta Corporation
Kennedy Space Center

ABSTRACT

This will be a technical discussion of the lessons learned during the seven years of software development/testing which occurred on the Liquid Oxygen System for the Space Shuttle at KSC. Problems which were solved during these years came into four distinct phases: design/debug before simulation runs, verification using simulation with models up through STS-1 launch, hardware usage from first launch to STS-5 launch, future use (integrated automation, VAFB). Each problem/solution will describe the apparent problem requirements/constraints, usable alternatives, selected action, results.

INTRODUCTION

Seven years ago, NASA contracted the Martin Marietta Corporation to hire systems engineers to work in the Liquid Oxygen (LOX) storage area of Launch Complex 39 for the purpose of servicing the hardware required to support the Space Shuttle. The commonly accepted definition of system engineering duties outlined a rather clear guideline for management as they went about hiring the finest available talent. Cryogenic experts, field engineers, eager college graduates, Saturn veterans, all were interviewed. Both interviewer and prospect didn't need to discuss task definition and, therefore, concentrated more on job qualifications. After all, a LOX systems engineer was to write procedures to maintain and operate the LOX hardware.

This was no simple task but by the same token this hardware was 15 years old by design and not as complex as some state-of-the-art facilities. A working system of valves, pipes, pumps, transducers, controllers, and pneumatics was inherited from the Saturn program. The task seemed even easier considering the 10,000 GPM pumping system was not to be used, only one LOX stage had to be filled instead of three and the existing procedures were made available by the Government from a previous contractor. The Space Shuttle philosophy made promises of being more operational by reducing the amount of maintenance and redundancy of less critical components, thus reducing the day-to-day work load of the systems engineer. The people selected for the original refurbishment of the facility were a dedicated group and long hours became commonplace as the seaside environment had taken its toll on the unused hardware. It took two years to complete the site activation work. There were more major modifications than expected to accommodate the Shuttle's loading requirements and procedures from the previous program were scrapped entirely by both the modifications and a new style of procedural writing. The end result of the refurbishment was a staff of systems engineers specialized in the operation of the hardware and ready to move into the operational phase.

It was soon obvious that the job of system engineering would have to expand. Interfaces with the local facility contractors and the NASA design group were suddenly expanded to the designers of the flight hardware, along with their respective NASA counterparts located in Houston, Huntsville and Kennedy Space Center. Cryogenic engineering was increased to include the related engineering fields of mechanical, electrical, pneumatic, and thermal as the decision was made for one console to have loading responsibility. Routine operations were being moved to the Firing Room consoles to take advantage of the new Launch Processing System (LPS) for daily tests which used to be run locally using switches. And, most significantly, a complex software set designed to use LPS for all operations was provided to the systems man as his primary 'tool' for future work.

These new areas of responsibility triggered a new definition of systems engineer. The resulting months of preparation for the STS-1 launch were spent exercising the software set against a high fidelty math model, using a simulated loading environment to establish the appropriate man-to-machine interface. The support of a new breed of software engineers was needed to develop, test, and demonstrate the new tool's capabilities and limitations to the systems engineer. Training under fire for a chosen few seemed to be the only method timely enough to safely satisfy the major objectives of a LOX cold flow, tanking test, engine firing, TPS retest load and the first launch. In subsequent launch flows, this broadened definition of a system engineer continued to increase as the mechanically and hydraulically operated GOX arm was added to the LOX console, along with the responsibility for a purge panels on the mobile launchers, and a new understanding of geysers with the removal of the ET anti-geyser line.

*Senior System Engineer, MMC-KSC

Training continued, in parallel with the launch activities, to bring the new assignments of the LOX engineer to a clear cut definition and to expand the talents and experience of the entire group. This did not happen easily. Software engineers had a different perspective on what was efficient than did a field engineer and quite often the two differed as to how the tool was to be used. Some systems engineers likened the Firing Room CRT to a glorified video game and perferred to do only the original tasks first defined as system work. A third generation of system engineer developed - the console operator, a combination of both the software and the hardware expertise.

These are the words this author has used to answer the question, "What does a systems engineer do?" This paper will answer the question, "What is the system engineer's tool?" In order to avoid confusion, the liquid oxygen (LOX) system at Kennedy Space Center will serve as the example cryogenic system. Actually the Liquid Hydrogen system was developed in a similar manner and contains many of the same philosophies. The lessons learned in the development of this automatic software set will be discussed in three phases - design and debug, verification against a mathematical model, and modifications due to hardware testing. This includes the first five developmental flights of the Space Shuttle. Then, a fourth phase, dealing with present and future modifications, will be discussed. Each of these phases will present several examples of problems which were solved by meeting the requirements and constraints of the LPS system and the hardware. Each item was, in itself, a discovery, since no one had previously used a computer to control the LOX system.

DESIGN AND DEBUG OF LOX SOFTWARE SET

The original LPS system was designed with the LOX and LH_2 systems in mind. NASA felt that these two systems were the most hazardous and complicated and therefore would serve as the pilot system for the general architecture of an automatic software set. The design contractor was selected to do the task and began creating the structure in 1976.

The LPS system restricted the use of a Firing Room console minicomputer to 3 CRTS, 3 keyboards, 6 concurrent programs and 4 sub-calling levels for an application program. A brand new language was devised, GOAL, which was close enough to English that an engineer without a software background was supposed to be capable of reading and understanding the programs. A Common Data Buffer, several Front End Processors and Hardware Interface Modules connected the LPS system to the hardware. NASA asked that a modular programming concept be used to reduce maintenance and increase reliability and further restricted the number of concurrent programs to two, so that up to three systems could be loaded into one console. It did not take very long to adopt a concept whereby all monitoring takes place in one concurrency and all control and decision-making takes place in the other.

In the monitor concurrency, an overall system display was developed using the proto-type LPS character set which provided symbols for valves, pipes, transducers, controllers, wires, switches and various components which could easily be recognized on a 7x9 dot matrix character and could occupy any position on a 34 line-by-73 column CRT. This overall page was to include any item which was essential to the loading operation and eventually depicted all lines in the Ground Support Equipment (GSE), Mobile Launcher Platform (MLP), Fixed Service Structure (FSS), Tail Service Mast (TSM), Orbiter, Main Engines (SSME) and External Tank (ET) which were subject to cryogenic fluids (see Figure 1). Gaseous purges and other support functions were left off the overall and were added to other displays which were divided into the three areas of the system - storage area, MLP, FSS/ET (see Figure 2-4). Also during the debug phase, a fifth display was added, power distribution, to complete the original display set (see Figure 5). All of these displays were called as a sub-level program by a display scheduler which tied up one entire concurrency. The schedular is seldom executing as its main function is a switcher between the other displays, but it is always active as a top-level program. The LPS system allows this display concurrency to be seen on any of the three CRTs at the LOX console, but only one display can be seen at any one time by any position. This did not seem to be a problem at the time since only one operator in a loading configuration would normally be watching the display and his interests would be on the overall display most of the time.

An entire paper could be written on the human engineering involved in these displays. Many hours were spent looking at all facets of update rate, color, size, location, identification, familiarization, and standardization of each character on the display. These displays are one half of the man-to-machine interface and, as such, take on a critical role in an automatic control system because this is the only way to see if the operation is proceeding as planned. Virtually 80% of the operating procedure is devoted to describing exactly what the operator should see happening on the displays.

One other important feature was baselined in the displays to increase reliability and speed in the daily operations and to give operator flexibility in loading operations - cursor control. This allowed an operator, using the display page, to command a valve open or to adjust a pump setting simply by moving the console cursor (X-Y type) to the component and executing a single command. In reality, the command was not sent from the display concurrency, but from the control concurrency. A sim-

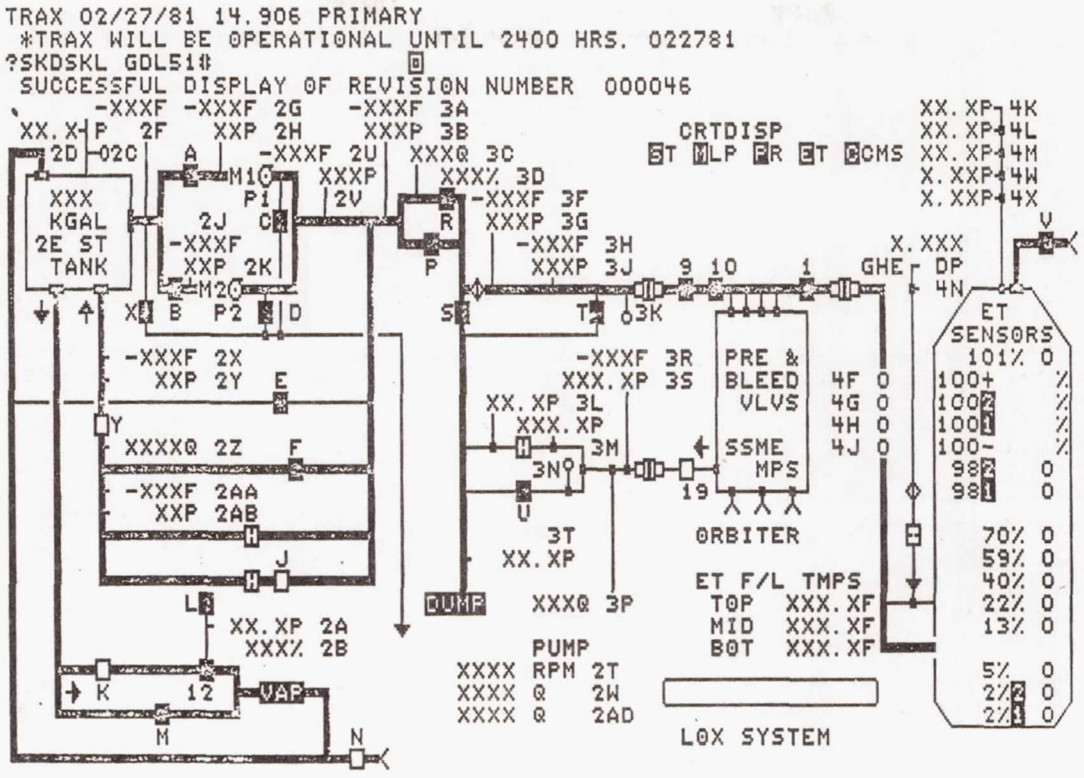

Figure 1

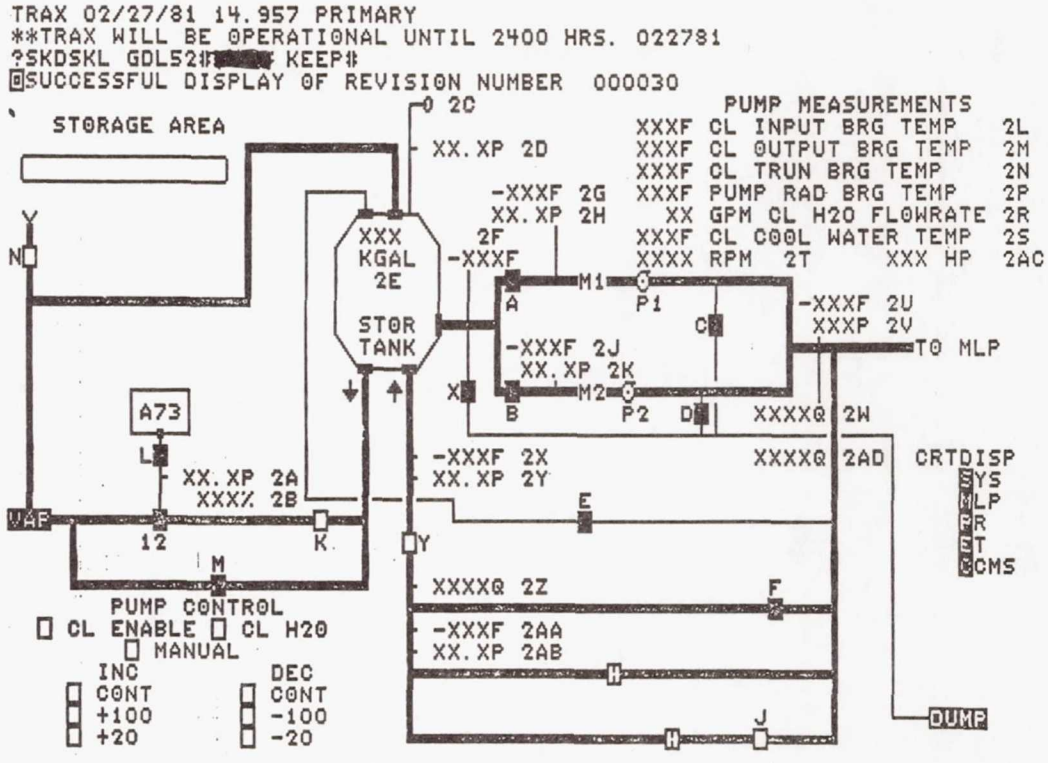

Figure 2

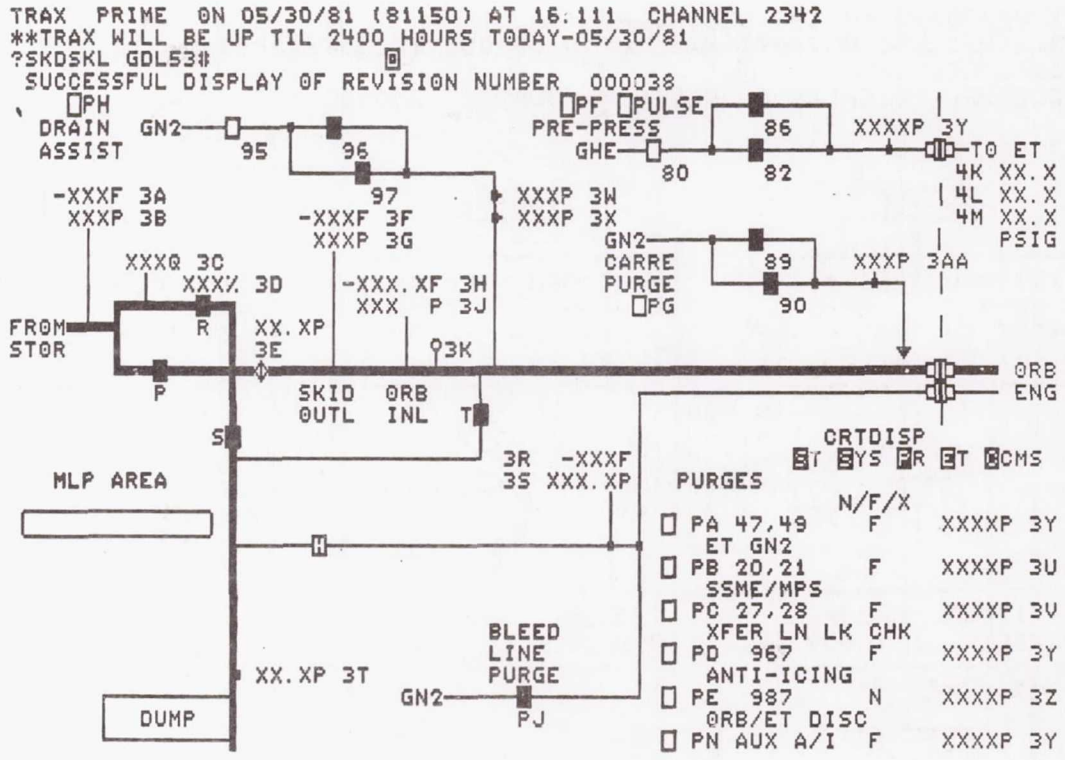

Figure 3

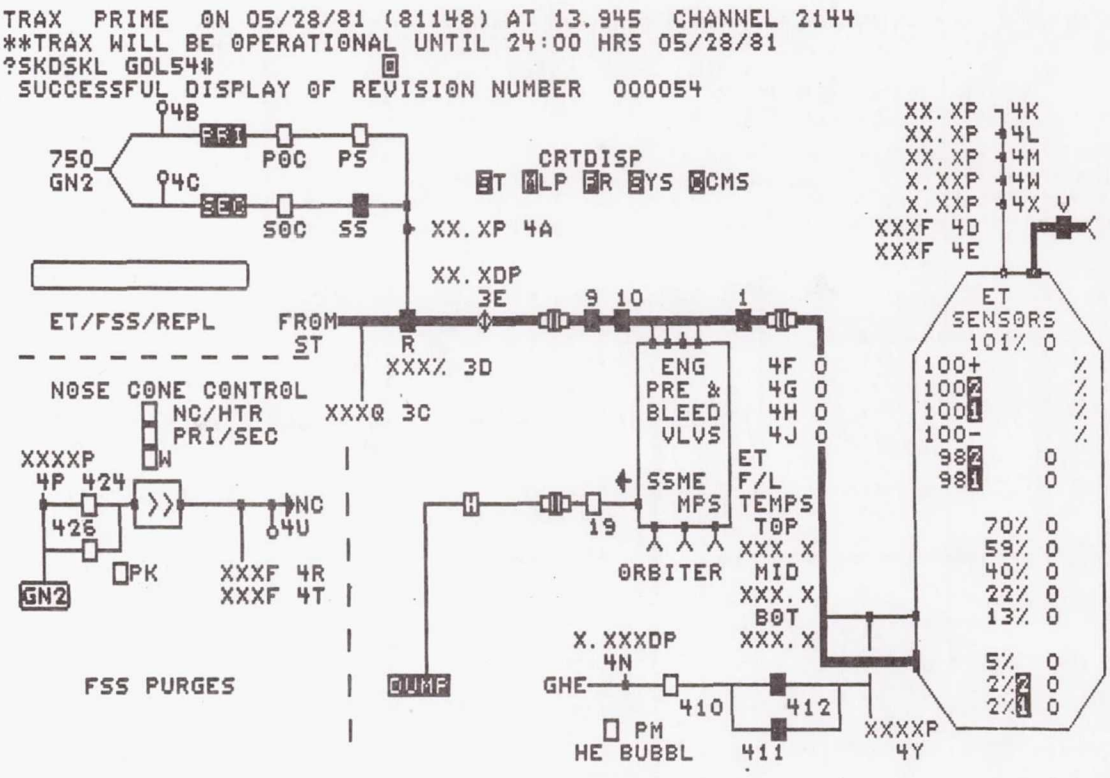

Figure 4

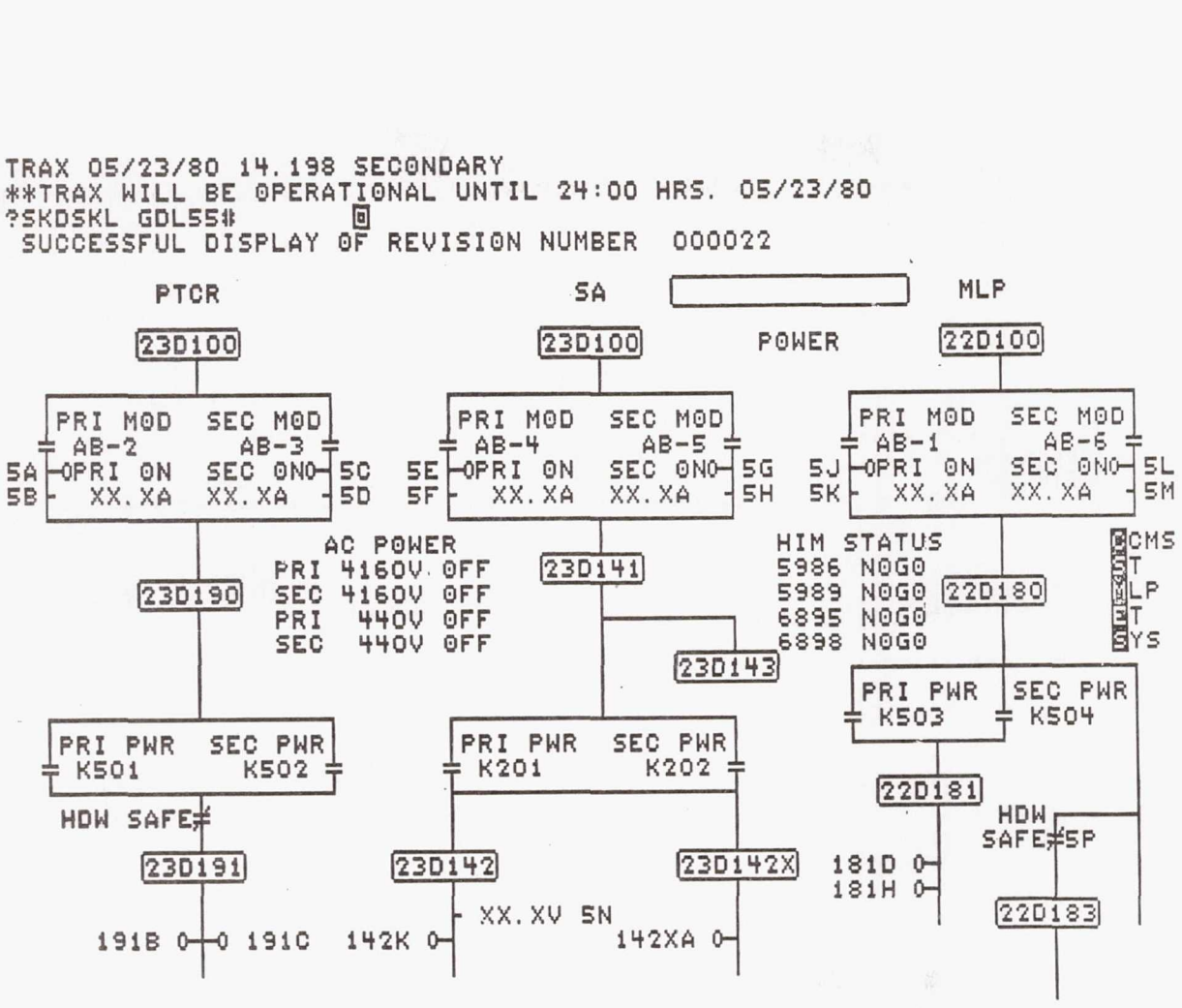

Figure 5

ple communications interrupt and parameter passing scheme allowed the control concurrency to be master of all commands whether they be issued from the automatic software or from an operator input. This allowed the display to continue updating all components rather than dedicating a fixed amount of time during the commanded valve travel.

Over in the control concurrency, the actual commands to the end item were issued by a component program. This is one of five types of programs in the control concurrency structure, defined as follows:

Schedular - Allows Operator to select the desired test he wants to perform.

Sequencer - Controlling program for a particular test. Calls lower level programs in proper sequence.

Component programs - Called by sequencer to perform a particular function, i.e., open a valve, close a valve, purge on, purge off, power on, power off.

Task programs - Usually called by sequencer to perform software chores or special functions.

Interrupt programs - Called by sequence when a measurement deviates from a pre-determined state or value. Provides message on CRT regarding faulty measurement and status of other measurements pertinent to that component.

The calling heirarchy of these programs is shown in Figure 6. Calling is done automatically by the sequencers for any lower level program, whereas calling for a sequencer is done manually by the operator through a Programmable Function Panel (PFP). This panel is located immediately below the CRT and provides six arm-and-fire buttons which are LED-labeled to prevent operator error. The PFP options are updated by the manual standby sequencer in the manner shown in Figure 7. Note that two actions are required to select a sequencer. This is now a NASA standard for all systems.

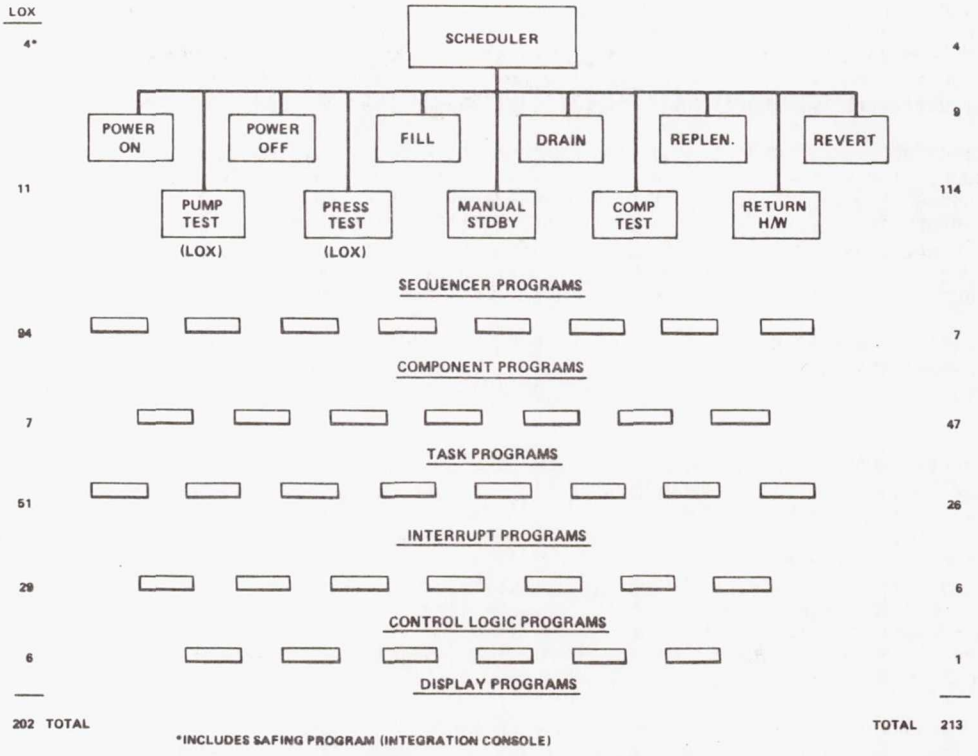

Figure 6.- Software structure and element count.

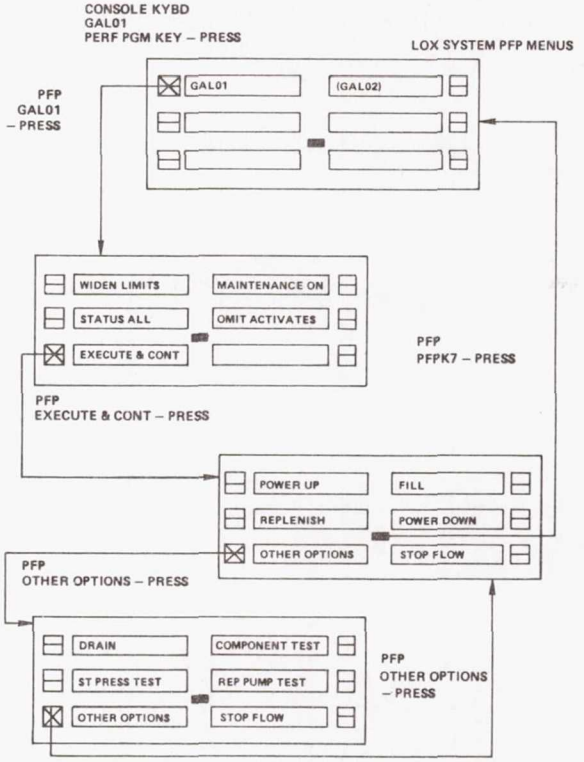

Figure 7

Although component programs are most directly responsible for the command/response or an individual end item, it is the sequencers which perform the most critical functions of timing, sequencing, safing, and redundant operations. The four sequencers used during a loading operation are fill, replenish, drain, and revert. These four programs control and set automatic monitoring for the more than twenty different configurations which are defined in the LOX Load/Drain criteria document.

The need for each program to know the current phase of the system gave rise to global flags in the data bank. These came to be known as pseudo-function designators (FD) and could be read universally throughout the Firing Room so that other systems (e.g., LH_2, Integration) could also find the necessary information about the LOX system status without disturbing the operators. This same type of pseudo-FD was used for flagging failed measurements or commands (bypasses), for sending communication interrupts from console-to-console, for creating duplicate measurements, and for communicating with the control logic (described later) programs resident in the console.

All together, pseudo discretes (500) and pseudo analogs (2) play an important part in automatic software communication. As evidence of the necessity of these pseudos, NASA now reports weekly on the limited remaining spares and as every other system increases automation, LO_2 system pseudos have actually gone down.

The fourth important piece of the LOX automatic software structure is a group of GOAL-like programs called control logic. These programs reside in the console on a high speed, high priority basis to act as a double-check that no command is issued from the console nor does any primary condition exist in the hardware that would cause an immediately hazardous situation. These two distinctly different types are called prerequisite and reactive control logic respectively. Prerequisite control logic is seldom violated, although always active, in an automatic system since the sequencers activities are preplanned and have been tested many times before hardware use. Its primary function is then to prevent the operator from manually commanding an incorrect action. Reactive sequences are the backbone of automatic safing, usually sending an initial salvo of commands within 50 milliseconds and then triggering a safing sequencer to bring the system to a totally secure configuration.

This software set totaled 202 programs with over 64,000 lines of code. The majority of this code was written within the timespan of one year. Obviously, the job of debugging was considerable. Some help was gained in the fact that these same concepts were used in the liquid hydrogen automatic software set and had been tested against a Math Model simulator in the Firing Room. Debug became little more than a successful compile and load as the schedule drove into the next phase of verification. The classic definition of debug was not complete and as such debug continued long into the verification process. However, the basic structure described so far still exists as the structure standard for today's LOX operations as well as the standard for several other high energy systems as the automation process at Kennedy Space Center continues.

VERIFICATION USING A MATHEMATICAL MODEL

This phase of the LOX software development lasted almost three years as engineers struggled against an ever-changing maze of LPS system software, LOX system hardware and requirements, and a sliding launch schedule. Engine and TPS development problems which slowed the entire Shuttle program made it difficult to predict when this new set first would be used against the hardware. Firing Room time was hard to get as everyone wanted to checkout the new LPS capabilities. Manpower was limited, at first, because the complications of an automated control system required software engineers to aid systems engineers in creating the automatic set.

Despite these complications a great deal was accomplished during this phase. The greatest 'aid' of all, the SGOS Math Model, was in the final stages of development and provided a high-fidelity simulator to respond to the application set. This high-fidelity modeling is essential to the verification of any hazardous software set. A great deal of timing problems was found simply because some commands responsed in milliseconds and others would see no appreciable change for up to one minute. Operator training became important and if a perspective operator watched the model respond incorrectly to a given situation, it would be sheer guesswork to estimate the effect of an incorrect real-time reaction. Details of the LOX model will not be discussed in this paper, but can be found in "Mathematical Models For Space Shuttle Ground Systems" by E. G. Tory.

The second major accomplishment of this phase was the establishment of software verification procedures (SVP). This was a configuration - controlled document which listed the exact steps required by an engineer to effectively test every line of code against the Math Model. Both NASA and contractor management could not justify the risk of leaving out unlikely branches of code from the verification, so rather than accepting an improbable failure, a 12-volume, 7,000-page set of SVPs was developed and executed. This, of course, required numerous informal runs before running the official test with the signature authorities present. To this day, maintenance of the SVP is a task equal to

the effort required to maintain the code. Many problems were found as the SVP sought to run all code over and over again in different configurations to really provide all aspects were covered.

The third major accomplishment was the development and use of simulated loadings. Over thirty simloads were performed before the first Shuttle tanking test at KSC, enough to try over 100 different hardware 'failures'. This procedure, with over twenty different operators working together in the same Firing Room configuration, using the same consoles and communications channels as a normal launch day, was also used for almost 1000 man-hours of informal engineering runs. Once again, the Math Model proved to be the tool capable of simulating realistic failures or anomalies, and then responding to the workarounds initiated by the operator. Many a vehicle was "damaged" as the simloads exercised the developing software set.

The verification process also brought some significant changes to the actual coding design. A new flag for maintenance operations was invented as a way to short-circuit control logic which was always active but was designed with a loading day configuration in mind. An emergency button was added to the PFP that could call a revert and bypass all control logic so that an operator would have total cursor control capability. A hardwire panel was added to the console to provide control in the event of a LPS failure and a hardwire recovery sequencer was coded to provide a faster recovery from such an event. Every component program was rewritten to suspend interrupt processing during 'valve in transition' when the first valve cycling test against hardware showed two interrupts for every valve indicator in motion. Measurement sampling rates were increased from 1 sample per second to 100 sps while a valve was in transition and then returned to 1 sps so that accurate data was automatically recorded.

A significant change to the control logic design had to be found when ten prerequisite sequences exceeded the size limitation of 256 bytes. The coding had already been written in the most streamlined fashion so the only alternative seemed to be a streamlining of the requirements. Deleting some requirements seemed too risky since control logic is the last line of defense and the impact to the LPS to upgrade the size limitation was severe. The solution seemed to be to find a way to consolidate a block of 'pump off' code which was common to all ten prerequisite sequences. Since subroutines were not possible in control logic, this block of code was made into a reactive control logic sequence which was always active and whose primary function was simply to set the correct state of a 'pump off' pseudo discrete flag. Now the ten prerequisite sequences only had to status the one flag thus reducing all the programs to less than 256 words. This became known as a "pseudo reactive sequence" because it issues no commands and performs no safing and, as such, is treated differently than all other reactive control logic programs.

Control logic also provided the best technique for a new vent valve cycling requirement which came into effect a few months before the first launch. Simply stated, the ullage pressure in the External Tank (ET) must be maintained above 1.7 psig when the liquid level is greater than 2%. A new set of two low pressure measurements with a range of 0-5 psig were installed on the ET along with the three original 0-30 psig high range instruments. Since this requirement applied to four different sequencers and must be active during all transitions between loading and drain phases, interrupt processing alone was determined to be risky in view of the likelihood of tank damage as a result of a pressure undershoot. So three control logic programs were created as follows: one to open the vent valve on the high limit of 8.0 psig based on any of the high range transducers, another to close the vent valve when the primary low range transducer reached 2.2 psig using the primary command, and the third to close the vent if the secondary low range transducer reached 2.0 psig using the secondary command. These trigger points were selected to allow adequate response time for the pneumatically operated vent valve (2 seconds) and the control logic sequence (50 milliseconds). This concept featured a pair of totally redundant circuits on the low side which precluded any single-point failure from threatening vehicle damage. Math model testing, SVP, and simloads were used to verify the process and the decision was made to use this technique for the first loading with no other hardware testing.

MODIFICATIONS DUE TO FIRST FIVE LAUNCHES

The LOX loading for the first space Shuttle launch on April 12, 1981 went extremely well. This turned out to be the sixth loading that would be performed using the baseline STS-1 software set as a launch scrub and two TPS tanking tests were added to the planned loadings. These loadings were far from 'automatic' though as operators learned to manually adjust their new tool to the first-time anomalies as they occurred.

The two biggest problems were two long-term control loops which had to be done manually and in parallel by the same operator. The nose cone purge was designed for a hardware controller to maintain 55 to 110 degrees in the nose cap, but failed to respond fast enough to the cooling effects of the vent valve cycling. This required the operator to manually adjust both the set point and the heater panel output temperature and to constantly monitor for further adjustments throughout the six hour

countdown. Also, the replenish control loop coded into the software failed to achieve the stabilized values which had been seen during testing at the National Space Technology Laboratories. So the same operator for the final three hours of the countdown had to manually monitor and adjust the replenish valve to maintain a steady flight mass on-board.

The other operator was constantly occupied with the steady stream of interrupts, exception monitoring and anomaly messages generated as a result of several blueline measurements being set too tight to allow momentary spikes or glitches to pass through the wary eye of the computer. Limit setting during debug and verification had not seen the kind of 'noise' and transducer inaccuracies which were within specifications and acceptable for a loading environment.

Other annoyances came from the slower than expected update rate of the displays. It took fifteen seconds for an initialization each time the operator changed pages and then ten seconds for each subsequent pass through the loop. Some measurements unexpectedly, needed to be constantly monitored such as the nose cone temperatures and the pump bearing temperatures, but were not available on the overall display. When the operator switched to a detailed area, say the storage area to check a pump reading, there was a total loss of visibility to the rest of the entire system. Application programs had no way of knowing when control logic programs had interceded, forcing the operator to presume that his invisible safeguard was in operation. Communications between LOX automatic software and Ground Launch Sequencer (GLS) automatic software were operational for a normal countdown, but incomplete as far as handling some abort and recycle conditions. Interfaces with GOX arm and ET power operators cluttered the communications net and seemed to indicate a need for greater software automation in those areas.

Most of these items were corrected by the STS-2 launch in November, 1981, however two major projects were put in motion on the second software set which would take until the last developmental flight (STS-5) to complete. Several years had passed since the original displays were built and the addition of new items late in the development left some areas terribly crowded and other areas totally blank. Display and color standards had been adopted by NASA, but had been waived to avoid impacting existing displays, so there were several obvious items of non-conformance. Update rates, limited visibility, and the status of related systems became a concern as described in the previous paragraph. All these became of more concern when the decision was made to move the GOX arm system into the LOX console and to absorb their software and displays. Clearly, a new display structure was needed. Secondly, a top level decision to remove the LOX anti-geyser line, as a weight-saving measure, created a much tighter set of temperature requirements in all phases and meant a totally new loading procedure.

Returning first to the display problems, it seemed certain that the learning experience of the first six loadings should allow the console operators to redesign the overall layout so that it could be made more compact. If this was done so as to create enough room to have a breakdown area included in a split-screen effect, then each area could be seen without loss of visibility to the overall and redundant displaying of items on more than one screen would be unnecessary. Pursuing this path soon led to a single display with three sections - the top half was the overall, the bottom right quadrant was the GOX arm system, and the bottom left quadrant was a selectable area which could show either the storage area, MLP, FSS, Power, or the new LPS status page (see Figure 8). By adding in the new color and display standards, the LOX loading displays took on an entire new look on the outside, but that did not complete the job.

On the inside, a brand new structure was required to cyclically update the selectable quadrant as a sub-level program from the overall, rather than directly from the schedular. Voting logic from each valve with multiple indicators and bypasses was streamlined to be unanimous or show an undetermined state rather than projecting a best guess to the operator. Code to determine significant change from the previous reading on analogs was dropped as bit toggling was handled by a simple formatting change. Console disk files were used as an innovative method for saving valve states while bouncing between displays. Initialized displays stayed in an undetermined state until one pass through the update loop to prevent the operator from quickly reporting an erroneous state. If several measurements could be interpreted and condensed to one signal, this was done in the software and shown on the display as one data point. New coding techniques got the loop update rate down to three seconds and initialization time down to three seconds on sub-levels, with no initialization time for the overall since it was displayed constantly. This entire project was phased in over three loadings (STS-2 to STS-4), and resulted in a total achievement of the previously mentioned objectives.

During the same loadings, the loading criteria was rewritten for Orbiter chilldown and slow fill phases. Test data from STS-1 showed sufficient cooling of the facility to allow the anti-geyser line to take away any excess heat which may cause a geyser in the 120-foot long 17" feedline during slow flowrates. But with the anti-geyser line removed for STS-5, temperatures were several degrees above saturation. At first, a 2-minute hi-speed flush of the transfer line was tried. This took enough heat out of the facility but could not relieve the latent heat of the Orbiter and engines. On the next loading, the flush was followed by a low flow load to the engine cut-off sensors (ECO) with a

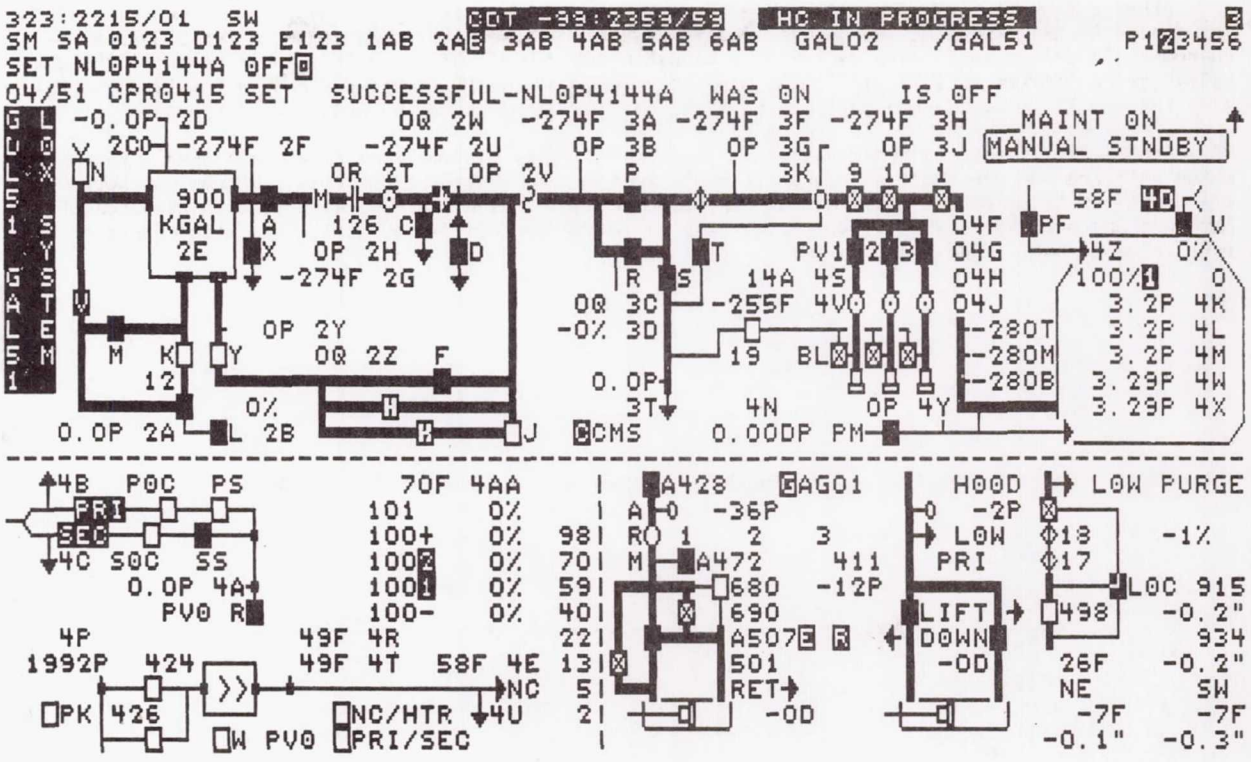

Figure 8

5-minute drain back through the engine bleeds. Each loading changed the valve actions, timers, and redlines such that the same software was never used, yet each time the automated sequencer had the flexibility to safely provide the required LOX on time to support launch for STS-5. Several modifications had to be made to the terminal count, abort and recycle operations to support the warmer LOX coming down the main feedline prior to engine start. Good planning and the conscientious efforts of the entire propulsion community, made the STS-5 tanking test and the subsequent loading for launch the first truly automatic loadings of the LOX system at KSC. All of the procedures, requirements, criteria, software, displays, and hardware modifications had come together in a way as to allow the auto fill sequencer to be initiated at T-7 hours and automatically continue through post-launch securing.

THE FUTURE OF AUTOMATIC SOFTWARE

The future of automatic software is now! Based largely on the remarkable success of the LOX and LH_2 systems, NASA and the operations contractors have formed an automation sub-panel which has established the automation goals of every console in the Firing Room. It is important to realize that while every system suffers the risk and effort that is required to leap from a cursor-controlled, manually-operated software set to a fully automated one, the LOX system must continue to improve. Software decays! It becomes outdated as the world of hardware, requirements and LPS changes around it.

For STS-6, for example, a new lightweight ET, a new Orbiter, and a new MLP meant the operation of configuration flags which a single sequencer could use so that it was capable of loading to any configuration. Representatives from Vandenberg Air Force Base are part of the automation sub-panel and, as part of developing their LOX software set, have submitted an idea for parameter-passed component programs which this author believes will save maintenance costs over the life of the Shuttle program. LPS is scheduled to provide more coding capabilities which will allow the continued development of fault isolation and correction programs. The integration console is designed for the day when a top-level manager program, communicating through linkers to all consoles, can kick-off the entire 72-hour countdown with one button.

Other changes to software, not driven by improving automation, but by improved hardware, will be required. Already approved and in the working stages are a new chilldown sequence for STS-7, a requirement to delete vent valve cycling and maintain sub-cooled LOX for STS-8, an increase of the vent valve stroke limiter to 1.5" on STS-10, and the removal of the 2% liquid sensors on STS-13. Eventually there will be two launch pads, three MLPs, and four Orbiters being processed in parallel.

Controlling these cryogenic systems and other hazardous systems must be accomplished with automated software for the Shuttle program to meet the turnaround time objectives. Systems engineers will continue to be essential to perform the classic functions of design, test, and maintenance of the hardware and procedures, but we must also take an active part in design, test, and maintenance of our most valuable tool - the software set.