AN ORGANIZED APPROACH TO HARDWARE DIAGNOSTIC DESIGN

Submitted by
Frank Murphy
Aydin Computer and Monitor Division
700 Dresher Road
Horsham, PA 19044

ABSTRACT

The built-in diagnostic test has taken on an increased role as a maintenance tool in today's complex electronic systems. While the ultimate diagnostic would exercise all of the major functions in a system and instantly isolate and identify any fault down to the specific part, many practical problems stand in the way. Using the diagnostic facility installed in a recent frame synchronizer/decommutator for the Jet Propulsion Laboratory (JPL) in Pasadena, the author attempts to show the logical approach, considerations, and compromises necessary to design the best possible diagnostic routine in a telemetry processor.

INTRODUCTION

In recent years, the built-in diagnostic test has progressed from being almost a novelty to being virtually an essential element of many complex electronic systems. While the ultimate goal of the diagnostic is to exercise the system's circuitry and to detect and isolate any system problem down to the specific part at fault, achieving this goal from a completely automated standpoint is nearly impossible. Why? By examining the real case of the development of a diagnostic routine for a frame synchronizer/decommutator recently designed for the Jet Propulsion Lab (JPL) the author will show the considerations and compromises necessary to develop a viable diagnostic facility in a telemetry processor.

THE JPL FRAME SYNCHRONIZER/DECOMMUTATOR

The diagnostic routine under discussion was needed for a frame synchronizer/ decommutator designed for the Deep Space Network of the Jet Propulsion Lab for use in the Magellan Mission. The function of the unit was to break up serial frames of data into blocks of parallel words and to add header words into each block with status information concerning frame synchronization state, block count, frame sync pattern errors, etc. Previously done in software, the real-time function now called for a full-function

synchronizer merged with two FIFO buffers, one for the data, and one for the header information. An sequencer would combine the two at the output. Aside from the timing complexities of matching the proper status information with its status, the data flow was basically simple. Each unit had three circuit cards: two synchronizer/decommutators and one microprocessor control card. Control was provided by either a remote RS-232 port or front panel keystroke input, aided by a gas plasma display.

JPL had a good track record with the diagnostic abilities of their other equipment. Technicians liked the simple checkout procedures and management liked the reassurance they provided. It was no surprise then that one of the requirements of the new design was an automated diagnostic routine that would isolate any fault down to the IC level. Though this requirement sounded very simple, the obstacles to its implementation were formidable.

THE LIMITATIONS

First, the system would have to exhibit a fairly high level of viable operation just to run the diagnostic. Many possible failures concerning the controlling microprocessor/PROM networks, control interfaces, or power supplies would leave the unit "braindead" and incapable of even delivering any useful information much less gathering it. Second, even in a simple, serial, data processing path like this one, many faults could be so fundamental that a processor-driven diagnostic looking at the output would not be able to derive any information about the problem.

What if a main clock driver died and data could not get past the input buffer? In cases like these the only possible solution would be to get out an oscilloscope and poke around a logical list of signals. The overlapping of IC gate usage between functional blocks could also help to make fault isolation difficult. One gate in an IC might belong to a very specific, detectable function while another might leave the technician in the "fog" of a problem that is acknowledged, but impossible to isolate by test results. Also, while a given parameter setup could exercise the unit very well, it could not be expected to exhaustively check every bit of every setup parameter the way a good acceptance test procedure could.

THE COMPROMISE

So while perfection could not be attained, the task of engineering a solution has always been to find the best possible compromise. We decided that even though we could not achieve a completely automated test that would tell the operator just which IC was bad, we could design an overall step-by-step procedure that, if followed, would lead the operator to detect and isolate most failures to an IC or group of ICs. The procedure involved automated tests and technician investigation in an organized plan. Would the plan detect and find every single problem? Maybe not, but then how could one even simulate

the myriad of possible random breakdowns to test the diagnostic? We felt confident that the procedure would find the overwhelming majority of flaws, and we planned to test it by removing all of the IC's, one by one, to verify the diagnostic's operational effectiveness.

Our initial plan to acquire diagnostic information was to implement an output port, parallel to the real output, but accessible by the setup processor. We thought that we might improve the fault isolation process by adding special circuitry to the board to acquire intermediate circuit information. However, as the main design progressed, we had to abandon these plans as the circuit cards became completely full.

THE FINAL PRODUCT

What we termed the "diagnostic facility" was a logical flowchart designed to find and repair any malfunction. The whole facility involved nine automated tests that ran in sequence on power-up. Based on where the tests failed the operator would either replace an IC or isolated group of Ics, or perform some test check such as a voltage check, or run a self-contained simulator stream and compare real signals to ideal traces in the manual. The best way to describe the diagnostic is to follow the chart from the most fundamental failures, and on through complete system verification.

The flowchart began with the unit powering up completely dead or operating erratically. The technician was led through a series of power supply voltages checks, followed by checking various critical signals on the microprocessor board, such as clocks, resets, interrupts, and memory reads. When the network of the microprocessor, memory, the front panel, and the frame synchronizer/decommutator setup were finally working, the automated tests would begin to provide answers.

There were three types of automated tests which ran in the following order: (1) memory checking tests that confirmed PROM checksums or wrote to and read back all RAM locations, (2) register tests that wrote to and read back all setup registers, and (3) frame sync/decom tests that ran on-board, canned simulator patterns through unique parameter configurations and compared the output header words and data to a table of known, good results.

Of the nine automated tests, the initial memory and register tests were very effective at pinpointing a failure, usually to a specific IC. After checking individual PROM checksums and RAM viability, each hardware register on the frame sync/decom boards was checked. These registers stored the control bits that told the synchronizer which mode to operate in and what count values to give variable parameters. The diagnostic test wrote and read back all 5's and all A's to these registers. Any individual failed register was detected quickly and reliably. We often wished that all of the ICs could have been tested that easily.

The last group of automated diagnostic tests checked all of the frame synchronizer/ decommutator functions. The only “window” we had into the unit’s main data reformatting function was a port to the final parallel output of the synchronizer/decommutator. This port was completely parallel to the normal output of the unit, so it allowed the control microprocessor to read back the exact data that, in normal operation, would go out the rear panel. All of the frame sync/decom tests consisted of setting up the parameters, running a canned format, and then checking the contents of the output. The user could then compare these results to tables of known good results in the manual. In this way the raw information was available to the user with all of its sometimes subtle, but important, clues to the failure.

Aside from the fact that it required most of the processing circuitry to work, this proved to be an excellent diagnostic vehicle. With access to this information, the operator would be able to tell not only what was wrong, but how it was wrong. For example, the headers contained frame sync errors and status, so the operator would be able to see in the case of an incorrect sync state advance if the sync state logic was at fault or if the error detector had reported the wrong number of errors. Even though we had designed a simple repair-by-numbers procedure that anyone could follow, I emphasized the value of this raw information during all design reviews with JPL.

Because these tests relied heavily on the overall ability of the circuitry to move data through the entire unit there was a large grey area where a fault could leave the “pipeline” blocked and no useful information would be available. In these cases the operator would be instructed to turn on a canned simulator pattern and, using an oscilloscope, compare a series of test points to known good signal signatures. The assumption was that the failure would be found when the real signals deviated from the expected. We agreed that what we were actually trying to do was to systematically design, and document the instinct and intelligence of a skilled troubleshooter.

Once the basic operation of the frame synchronizer was achieved, meaning that the synchronizer could lock up and that reasonably correct data could flow through the unit, then the final group of frame sync tests would be run. As before, a canned simulator pattern would be run through a specific setup, but now all of the various header words in the data output would be checked, one by one. These tests proved very effective since each header word was built from a very specific group of circuitry. About half were counters such as the large block counter or the valid bits counter. The other half were status bits, such as frame sync state, bit slip, and data polarity. Intricate parameter setups and data patterns were required to completely exercise these complex functions. Another complication was that in normal operation of the unit, the synchronizer could sync up on the MSB, LSB, normal, or inverted form of the data. Since separate hardware supported each of these possibilities, each test was actually run four times, once for each type.

CONCLUSION

We verified the completeness and effectiveness of the diagnostic test by pulling out all of the ICs, one by one, and found the results very interesting. Over 95% of the ICs were actually recognized by the diagnostic, meaning that it would not pass with them out. The only ICs invisible to it were the extreme input and output drivers surrounding our test input and output points. Around 35% of the missing ICs could be detected either specifically or as belonging to a small group of ICs. These included the read-back registers which were detected individually and the ICs directly and completely responsible for a specific header word. The remaining 60% of the ICs were detected as missing, but would require the operator to get out an oscilloscope and follow the trace procedure. We were all pleased with the final results of the diagnostic, and we felt that we had made the best compromise possible. We all knew that the perfect diagnostic was not realistically attainable, but within the time and physical restraints that we had, we felt we had designed the best diagnostic facility possible.