Diagnostic Software for SMARTNet-2

Woodrow D. Willis and Louis J. Duchesneau

TRW Avionics & Surveillance Group
Dayton, Ohio

Abstract

TRW developed SMARTNet-2 to provide a high speed shared memory network for support of real-time parallel processing. Each node of the network communicates in real-time with the other nodes through a SMARTNet-2 board set consisting of two circuit boards, referred to as the Host Board and the Network Board. From the beginning of the SMARTNet-2 hardware design effort, incorporation of built-in test and fault isolation features were emphasized to an unprecedented level. These features were required to identify faulty components to the chip level on either board. This capability was achieved through innovative design techniques both in hardware and in software.

Development of the hardware and software began in parallel. The core software was written in Ada; it implements a powerful diagnostic command language which is used to directly access the hardware. During rapid prototyping the software was used as a powerful tool to support hardware development. The design allowed for commands to be stored in external files, read in, and executed. This capability provides the mechanism for the hardware specific fault isolation processing.

Rapid prototyping put early versions of the SMARTNet-2 Diagnostic Software on the various computers being used for hardware board development and testing. This provided quick feedback to the hardware design engineers and insured that all addressable hardware could be

accessed by the software. As the hardware matured and new parts were added it was fairly simple to update the software to accommodate the new hardware.

This paper details the unique aspects of the design and implementation of the SmartNet-2 Diagnostic Software. In addition, it discusses the command language and the methods used to achieve fault isolation testing.

This work was performed as part of the Avionics Integration Support Technology (AIST) program for the Air Force Wright Research and Development Center (WRDC) under contract number F33615-88-R-1700/0002.

Project Background

Hardware testability and hardware diagnostics have become a central topic of new electronic circuit design efforts. Increases in circuit density and circuit complexity have dramatically increased the difficulty of troubleshooting and repairing modern electronic circuits. This paper describes TRW's approach to providing improved testability and maintainability for a computer interface circuit designed for the Air Force.

As real-time simulation architectures evolve from use of a single, monolithic central processor to a distributed array of smaller processors, high-speed real-time communication among the distributed

processors becomes the central engineering challenge. Under contract to the Air Force Avionics Laboratory, TRW has developed a network device to meet this real-time challenge - The Shared Memory Advanced Real-Time Network, Version 2 (SMARTNet-2). To an individual processor, SMARTNet-2 looks like conventional memory. When a processor writes data to a location in SMARTNet-2 memory, that data value is transmitted by high-speed fiber optics to the other SMARTNet-2 nodes on the network and inserted in that same memory location for all of the processors. Maximum real-time response is assured because this process is completely automatic and involves no software or hardware overhead from any of the distributed processors.

The SMARTNET-2 card set consists of a "Host" board and a "Network" board. The Host Board provides the interface between the host bus and the SMARTNet-2 card set. The Network Board communicates with the Host Board and with the Network Boards of both adjacent nodes in the network to maintain network traffic and protocol. The Network Board also contains a unique system for capturing network traffic.

With the evolution from the earlier prototype, SMARTNet-1, the Air Force chose to emphasize hardware testability and maintainability in addition to increases in network performance and throughput. The issue of hardware testability has been addressed recently by the IEEE Joint Test Action Group (JTAG). Their work on the proposed IEEE standard 1149.1 will eventually provide standardized test capabilities at the chip level, but as our development began JTAG was just starting. We determined that our testability requirements could best be handled through a hardware design which contained intermediate registers which were accessible through software, and software specifically designed to test the board set.

Objectives

The SMARTNet-2 Diagnostic Software was written in Ada to provide the following capabilities:

1. Component level fault isolation testing of the SMARTNet-2 board set.
2. Network level burn in testing.
3. An automated means of testing the SMARTNet-2 hardware during development, test and production.

The contract required the Diagnostic Software to provide a powerful on-line tool to aid in the identification of faulty components on malfunctioning boards. By designing the software so that it could also be used as an engineering and production tool, its cost effectiveness increased dramatically.

In addition to the functional objectives established for the Diagnostic Software, portability was also desired. This would allow the software to run functionally the same on each type of SMARTNet-2 node.

Development Approach

The software design began when the hardware design was still in a transitory state. This necessitated a flexible Diagnostic Software design, which would allow fast adaptation to hardware design changes and easy incorporation of newly developed testing methodologies. To provide this flexibility the Diagnostic Software was designed as a Command Language Processor, with an emphasis placed on modularity. The modular design would allow new commands to be easily added to the command language. The resulting command language would be used to provide hardware development testing, and would also be used as the underlying language for the fault isolation and network performance monitoring functions. Commands could be invoked on an individual basis or groups of commands could be stored in external SMARTNet Command Files (.SNC).

Because the Diagnostic Software would be used as an engineering development tool which would be needed soon after software design began, a fast prototyping development style was adopted. A simple but functional prototype was produced for review and use by the hardware engineering team. This allowed proof of concept to be established and also provided feedback on the validity and usefulness of the command language. The resulting comments were reviewed and necessary changes made to subsequent versions. This process was repeated as the software was coded, resulting in a final product which was immediately used and understood by the hardware engineering team. The development of the Fault Isolation files began when the Diagnostic Software and SMARTNet-2 hardware reached a suitably stable state.

The knowledge that our software would ultimately execute on a variety of different CPUs, forced us to address the portability issue at the earliest stages of our design. The packaging scheme of the Ada language was particularly well suited to managing our portability problems. When writing bus dependent code we created packages with a common specification but bus specific package bodies. By using selective compilation we could create the desired executables.

Portability also influenced our decision to divide the Diagnostic Software into two separate modules, see Figure 1. The User Interface Software (UIS) handles all user interaction and all file access. The Diagnostic Execution Processor Software (DEPS) provides the interface to the SMARTNet-2 hardware. For the MicroVax version a single executable is created with communications between the two modules provided through a simulated communications link. For the VME version two executables are created, the UIS code resides on a MicroVax with the necessary external files, the DEPS code resides on the VME computer. All communication between the two modules is performed using a standard protocol, currently RS-232.

Figure 1 Software Structure

Fault Isolation Implementation

Although the development of the Diagnostic Software provided many new challenges and discoveries, the remainder of this paper will focus on the methods used to implement the fault isolation testing.

The SMARTNet Command Language provides a wide variety of commands. There are commands which provide basic read or write access to RAM locations and I/O registers, commands to control logic flow and even a sophisticated command which performs a set of complex verification algorithms on an I/O register or RAM. Many of our RAM testing algorithms were derived from research performed by David Jacobson. After a few rounds of updates the command language met the engineering development needs and provided the means for fault isolation file development. Figure 2 presents an overview of the available SMARTNet commands.

User access to the fault isolation test suite is provided through a set of predefined menus selectable from the Diagnostic Software top level menu. When a specific test is selected, the appropriate .SNC files are accessed by the UIS and sent to the DEPS for execution. This approach allows the details of the hardware configurations to be reflected in the external .SNC files. This allows handling of most hardware

changes as well as implementation of new test approaches without requiring changes to the Ada code. When the Ada code is not changed it does not need to be recompiled and relinked, thus reducing development overhead time.

Flow Control	File Handling	Exerciser	Arithmetic and Logic	
Loop	@filename	Exer_Cntl	Add	Sub
Endloop	Download	Remote	Mult	Div
Jump	Execute		And	Or
Labelname:	Cleanup		Xor	Not
Hardware Access	Monitor	Miscellaneous		
Init	Clear_Monitor	Echo		
Read	Address_Monitor	Noecho		
Write	Cad_Monitor	List		
Compare	Data_Monitor	Nolist		
Verify	Sid_Monitor	Nop		
Random_Fill	Load_Monitor	Output		
Ram_Compare	Enable_Monitor	Percent		
Ram_Copy	Disable_Monitor	Print		
Connect	Dump_Monitor	Fail		
Disconnect		Status		
Wait		Symbol		
		nosymbols		

Figure 2 SMARTNet Command Overview

When a user performs fault isolation, the menu suite presents the available tests in a logically structured order. The Host Board tests are presented before the Network Board tests because the Network Board cannot be tested without a correctly functioning Host Board. Within the Host Board test the first set of tests presented will be those which test the interface to the host bus, this is because if the Host Board is not accessible via the bus then there is no hope of operating it. As this logical procession of tests is traversed, it is assumed that all preceding tests have run successfully. The presented order of tests is recommended for isolating faults, however any test may be accessed and executed independently at the user's discretion.

The set of .SNC files written to provide fault isolation are considered part of the deliverable code (along with the executable, a file used to define the set of menus and various initialization and setup files). The fault isolation files represent approximately 30 percent of the total lines of software written for this effort. About 25 percent of the Fault Isolation files contain bus specific commands or references. For

these files, we maintain separate bus specific versions which functionally perform the same tests. Therefore no matter which type of node is being tested, our fault isolation menus are identical.

The most unique feature of our fault isolation system is its component level tracking capability. This allows the software to track the number of tests each component was involved in, and the number of tests which passed. As tests are run, statistics are stored which are used to calculate a confidence factor for each component. Statistics are kept for all chip level components on each board.

The component tracking is implemented using a section of memory on the CPU which executes the DEPS software. This memory is referred to as the Results Ram. Two locations in the Results Ram are reserved for each component in the given hardware configuration. The first location tracks the number of tests, the second location tracks the number of passes. The "Test" and "Pass" variables are identified by assigning a unique symbol to each, this symbol is translated as an address into the Results Ram. The symbol definitions are stored in a special .SNC file which is executed every time the Diagnostic Software is run.

Test Approach

As we began coding the Diagnostic Software, we intended to have a functional tool available to support hardware and continued development as soon as the prototype SMARTNet-2 board set was available. This presented some unique testing challenges because many of the commands were unique to the SMARTNet-2 hardware.

The strategy used to allow software testing before the prototype board set was available, was to run the software on a MicroVax where we could reserve a large section of memory to simulate the SMARTNet-2 resident Ram memory and register set. This method allowed

testing of approximately 80 percent of the commands. We could perform Reads, Writes, Verifies, Flow control commands, etc., however we could not test commands which wrote values to registers which should cause the actual hardware to perform specific functions. We also ran our software with a SMARTNet-1 card set to allow check out of our physical memory mapping and interrupt handling schemes. The result of this testing effort was that integrating the Diagnostic Software with the SMARTNet-2 cardset required less than one day to complete.

Testing the ability of the Fault Isolation files to detect hardware errors also provided a testing dilemma. We needed to create temporary hardware faults without causing damage to the boards or the components. This was especially critical due to the many high priced components utilized on the boards. As a solution to this problem, the lead hardware engineer provided realistic hardware faults through three basic methods.

1. Grounding address, data or control pins.
2. Disconnecting address, data or control pins.
3. Inserting bad versions of programable components on the boards.

We felt that these tests were representative of the types of faults that would occur in actual use. Using these fault generation methods, the fault isolation software was on average able to identify the faulty component within a list of three or four different components. Our least successful test isolated the bad component within a list of eight components.

These results validated our concept, however we realize that the fault isolation software has some limitations. We cannot isolate board damage, this includes corrupted traces and small scale parts such as transistors, resistors, and capacitors. These are

important parts of any hardware board, however we decided that the effort required to track such entities greatly outweighed the probability of their failure. If such faults occur more typical debug methods, such as logic probes and data analyzers, must be utilized. In such cases the Diagnostic Software's fault isolation capability is still useful to indicate the general area of the board where the fault is located.

Summary

Jon Gabay, a contributing editor for the industry publication Computer Design, describes an ideal development environment as follows:

"In this ideal environment, test engineers work side by side with design engineers, software programmers and printed circuit board designers. This permits test features to be incorporated at the time of implementation rather than later adding components that may effect critical delay paths or already tight printed circuit boards. A software guru working with the group can also make the hardware implementation more efficient for operation and test."

Our success supports his theory. Lines of communication between hardware and software personnel were established early in the development cycle. This allowed the hardware to be designed such that it could be tested through software. The Diagnostic Software was developed in 14 months, using approximately 40 man months of effort, and met or exceeded all design requirements and goals. We believe that the Diagnostic Software will require little maintenance due to its implementation as a command language processor and its modular design.

At this time the Diagnostic Software has not been fielded, however we are confident that it will perform its fault isolation functions as expected. The Diagnostic Software has been used extensively as a hardware development tool

and in operational checks of the production boards before delivery, the Diagnostic Software has performed these tasks exceptionally. During development testing a reasonable level of fault isolation was achieved, however we feel that an increase in the built in test features of the hardware and further sophistication of the fault isolation .SNC files could double our resolution. We hope to make such gains when the technology developed on this project becomes a TRW product.

References

1. Proposed IEEE Standard 1149.1, "Standard test access port and boundary-scan architecture," Draft D6, November 22, 1989.
2. David M. Jacobson, "A fast, Probabilistic Algorithm for Functional Testing of Random Access Memory Systems" 1985 IEEE International Testing Conference, pp. 169-175.
3. Jon Gabay, "How much can Design-for-Test reduce the need for testing?" Computer Design, September 1, 1990, pp. 94-112.

The Authors

Woodrow Willis obtained his B.S in Computer Systems Analysis from Wright State University in 1983. He served as an adjunct instructor at the University from 1983 to 1986, where he taught high level language and assembly language programming classes. Mr. Willis joined TRW in 1983 and is currently a Senior Member of the Technical Staff. At TRW he has been involved in many high level language development efforts including the INEWS real-time operating system, real-time operational flight program development for the B2 and the Advanced Multi-Purpose Support Environment (AMPSE) for F16 avionics. His general hardware knowledge and broad computer language experience position him for a variety of embedded real-time applications.

Louis Duchesneau obtained his B.S.E. in Computer Engineering from The University of Michigan in 1985. At the University of Michigan he was a lab instructor for Ada and Real-Time programming courses. He has worked on projects involving the review of the ACVC compiler validation tests, and Ada compiler benchmarking. At TRW he has been involved in Ada development efforts including the ICNIA Fleet Satellite Communications and the Advanced Multi-Purpose Support Environment (AMPSE) for F16 avionics. He is a strong supporter of the Ada language which allows him to apply his unique blend of Hardware and software background to embedded real-time applications.