(2)

ANNUAL REPORT
VOLUME 5
TASK 4, 5, & 7: SOFTWARE DEVELOPMENT AND GN&C PROCESSOR DEVELOPMENT

REPORT NO. AR-0142-90-001

July 22, 1990

GUIDANCE, NAVIGATION AND CONTROL
DIGITAL EMULATION TECHNOLOGY LABORATORY

Contract No. DASG60-89-C-0142

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

Contract Data Requirements List Item A005

Period Covered: FY 90

Type Report: Annual

DRAFT

ANNUAL REPORT
VOLUME 5
TASK 4, 5 & 7 SOFTWARE DEVELOPMENT AND GN&C PROCESSOR DEVELOPMENT

July 22, 1990

Author

W. S.Tan, Karsten Schwan

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

Eugene L. Sanders
USASDC
Contract Monitor

Cecil O. Alford
Georgia Tech
Project Director

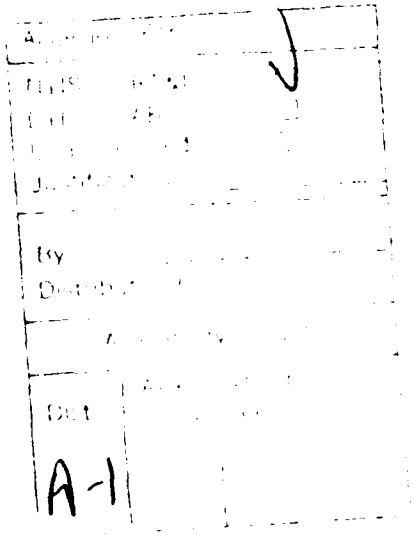
DISCLAIMER

DISCLAIMER STATEMENT - The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

DISTRIBUTION CONTROL

(1) DISTRIBUTION STATEMENT - Approved for public release; distribution is unlimited.

(2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227 - 7013, October 1988.



1. Introduction

1.1. Summary

Georgia Tech is designing a set of VLSI chips for the guidance, navigation, and control (GN&C) of KEW interceptors. The GN&C processor consists of three types of architectures: executive processor (EP), data processor (DP), and signal processor (SP). The EP is a general purpose processor with extensive general processing, numerical, and I/O capabilities. The DP is a numerical processor that is specifically designed to solve ordinary differential equations associated with closed–loop control systems. The SP is designed to process image data from a high resolution focal plane array. These processings include several filtering stages, an object clustering stage, and an object centroiding stage. Two VLSI chips are associated with the EP design. They are in the design stage. All the VLSI chips associated with the DP have been fabricated and tested. A prototype multiwire DP module has also been built and tested. Six VLSI chips are associated with the SP design. Three are in the design stage. Three are currently being fabricated.

The integrated parallel programming framework (IPPF) is a parallel programming environment for the development of parallel software applications targetted for the GN&C processor and the Georgia Tech real–time simulation testbed (Parallel Function Processor). The environment features a graphical programming front–end interface that allows system programmers to contruct application and system software based on block diagram programming approach. Most work is concentrated on the low level utilities that are necessary to construct the integrated framework. The front–end graphical interface is still in its early stages of development.

1.2. Objectives

The objective of the VLSI development effort is to devise a set of VLSI chips to form a parallel processing system with enough computational throughput to guide, navigate, and control advanced KEW interceptors. Specific objectives include the definition, development, integration, test, and evaluation of a set of functional GN&C modules which are implemented in semiconductor chips using VLSI technology. The mapping of the GN&C functions onto the VLSI chips are intended to be structured, modular, and flexible in nature. Once these modules are developed, system designers can use the VLSI chips as the building blocks to develop and construct a specific GN&C processor that is suitable for a particular class of KEW interceptors.

The objective of the IPPF is to provide a consistent, user–friendly, graphical environment for the development of parallel programming applications. This objective includes the Ada programming support to develop complex, real–time system software. The graphical environment includes a configuration editor that allows programmers to take advantage of the diverse hardware resources that are associated with parallel computer architectures.

1.3. Requirements

The GN&C processor is required to interface to a focal plane array (FPA) with 128x128 pixel resolution. The processing rate for the images from the FPA is 100 frames per second. At this rate, the GN&C

need to perform all necessary filtering operations to separate the targets from the background noise. These filtering operations include non–uniformity compensation, temporal filtering, spatial filtering, thresholding, clustering and centroiding. Once the targets are clustered, Kalman filtering is performed to track the movement and to extract the velocity of the targets. The targets are descriminated and one of them is designated for the purpose of computing the final aim point. All necessary processings are then performed to guide the interceptor to the designated target. Computations for the tracking and descrimination of the targets as well as control processing are carried out in IEEE 32–bit floating point numbers.

The GN&C software is required to be programmed in Ada high order language. The IPPT environment will need to provide the necessary programming tools to support the development of Ada parallel programs. A graphical interface is required to allow system programmers to hierarchically build the system software for the GN&C processor as well as the emulation software required to test the GN&C processor in a closed–loop system. A configuration editor is required to take advantage of the GN&C processor and the emulation testbed parallel resources. A performance monitoring system is required to allow system programmers to fine tune the overall system software to meet the performance requirement.

2. VLSI Development

In this section, the Georgia Tech VLSI development effort is presented. A detail description of the GN&C processor architecture is submitted under a separate final report under the old contract.

There are a total of 13 chips in the VLSI development plan. The chips are:

GT–VIAG : Instruction Address Generation chip for the EP.

GT–VDAG : Data Address Generation chip for the EP.

GT–VFPU : Fixed/Floting Point Arithmetic Logic Unit for the DP and the EP.

GT–VSEQ : Sequencer chip for the DP.

GT–VDR : Dataram chip for the DP.

GT–VSNI : Serial Network Interface chip for the DP and the EP.

GT–VSM8 : 8–point Switching matrix chip for the DP and the EP.

GT–VNUC : Non–Uniforminity Compensation chip for the SP.

GT–VTF : Temporal Filtering chip for the SP.

GT–VSF : Spatial Filtering chip for the SP.

GT–VTHR : Thresholding chip for the SP.

GT–VCLS : Clustering chip for the SP.

GT–VCTR : Centroiding chip for the SP.

2.1. Chips Completed

Five chip designs have been completed. They are the GT–VFPU, GT–VSEQ, GT–VDR, GT–VSNI, and GT–VSM8. These five chips combined comprises the DP processor module. A multiwire board had been built for these five chips. Several programs have been tested on the DP module.

There is one problem with the packaging of GT–VDR chip. The signal pin Pc[3] was bonded to a pin that directly connects to the back bias die substract of the chip. This significantly increases the input capacitance of the Pc[3] signal. As a result, the chip operates much slower than expected. The DP test has been carried out using a 1 Mhz. clock. The expected operating frequency is 6.6 Mhz.

Functionally, the DP chipset has been performing according to the design specification.

2.2. Chips in Fabrication and Testing

The GT–VCLS, GT–VCTR, and GT–VSF have just completed the fabrication at the HP 1.0 CMOS fab facility. The chips are currently queuing for packaging. Testing of the three chips are expected to follow suit.

The GT–VTHR is being prepared to be sent out for design verification (DV) at Mentor Graphics (formally Silicon Compiler System Corp.). All design stages have been completed. Only the final DV document remains to be completed.

2.3. Chips in Design

The GT–VIAG, GT–VDAG, GT–VNUC, and GT–VTF are in the various stages of the design cycle.

The GT–VDAG essentially has completed all phases of the design. The final DV document is almost complete. The chip is currently waiting for the GT–VIAG chip to reach the same design stage. The chip is used in a multichip simulation.

The GT–VIAG requires significant number of manufacturing test vectors before it can be sent out for DV. All functional design entry has been completed and tested. The chip is currently being extensively tested using multichip simulation at the EP system level.

The GT–VNUC has passed all functional test vectors generated by the SP simulator. Test converage is currently an issue. The design is being fine tuned to increase its observability at the chip level. Manufacturing test vector generation for the chip proceeds until a desirable coverage can be achieved.

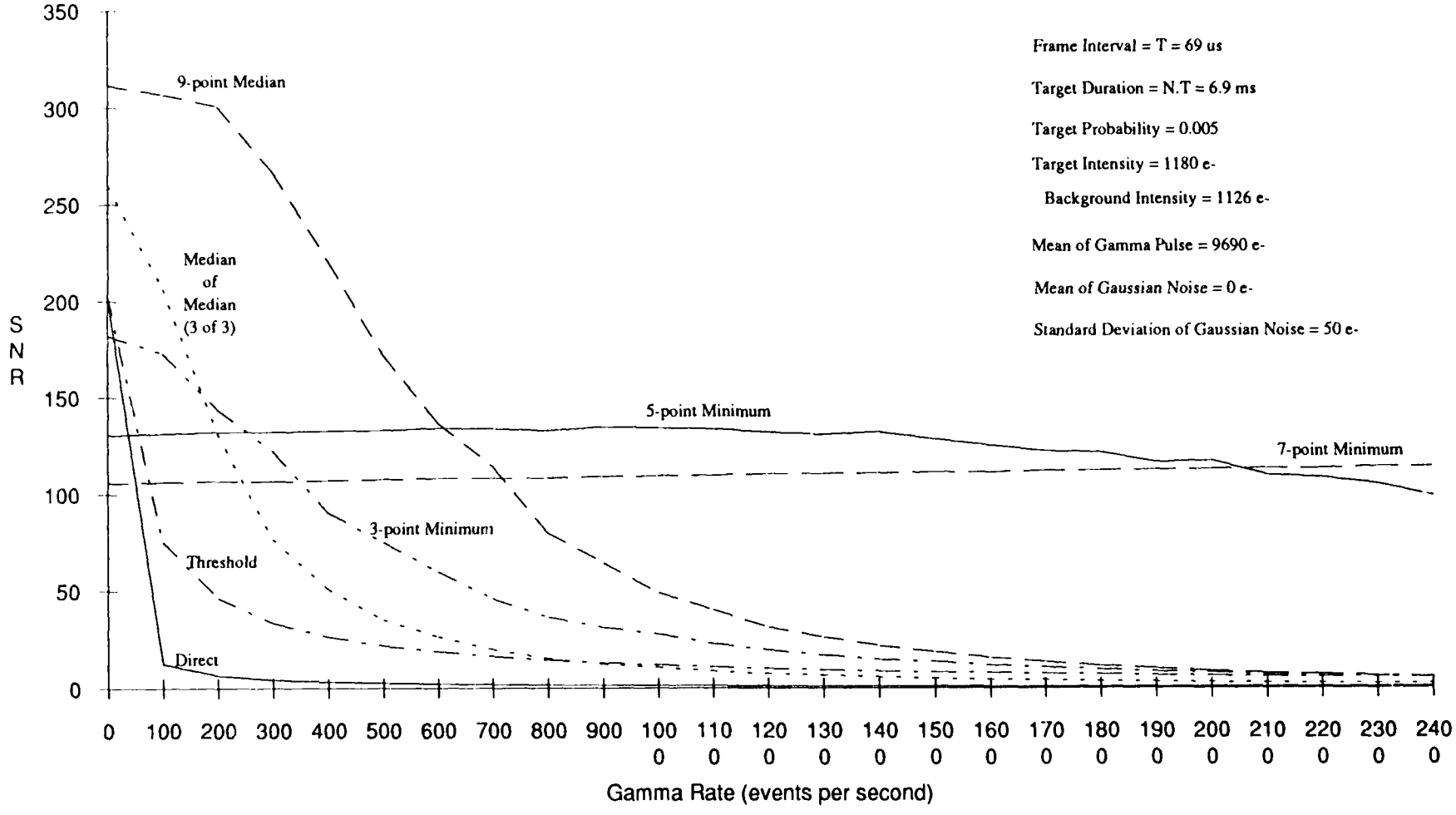
2.4. Chips in Consideration

A gamma suppression chip was considered for VLSI implemtation. A simulation study was initiated. The generation of the gamma events is based on statistical Poisson arrival distribution. The intensity of each gamma pulse is based on Exponential distribution. Gaussian signal noise generator is then added to the signal model. Several types of filter were studied. The result is shown in Figure 1. Each data point is taken from 1 millions sampled simulated signals with gamma and gaussian noise. A source code listing of the simulation program is shown in Appendix A.

As a result of several discussions with Lockheed Missile and Space Corporation, Georgia Tech has decided not to pursue the gamma suppression chip in digital technology for the following reasons. To be effective, the gamma suppression logic needs to operate at a very high frame rate (~ 10 k frames per second). Consequently high speed A/D convertors are needed to convert the analog signal from the focal plane array (FPA) to digital. Power consumption is propotionate with the speed of the A/D converter. Furthermore, Lockheed has indicated that gamma suppression can be accomplished in the analog domain. The rest of the SP filter algorithms only require a rate of 100 hz. This significantly reduce the size and power requirement on the A/D convertors.

A third version of the fixed/floating point arithmetic logic unit chip, designated as GT–VFPU/3, is now being considered. The GT–VFPU/3 will be matched closer to the EP design. The first GT–VFPU design was specifically tailored for the DP. The second version, GT–VFPU/2, was targetted for the Harris Gamma–3 Rad Hard process. GT–VFPU/2 never made it through fabrication because the Gamma–3 line was discontinued by Harris. GT–VFPU/3 is expected to be 50% faster than the GT–VFPU/1. Georgia Tech is considering on–chip double precision support for the GT–VFPU/3 design.

Figure 1. Signal to Noise Ratio of Various Gamma Suppression Algorithms



Several new features will be considered for the SP chips. These will include the ability to handle FPAs with staggered row and variable frame size. One feature for the GT–VCTR chip that might be useful is to compute the second order moment of the objects in the field of view. This feature is currently not supported.

The existing EP design requires very wide instruction memory. The code generated by the EP compiler shows that very few of the instruction bits are actually non zero. This leads into the investigation of compression algorithms which could significantly reduce the instruction memory width. This investigation will potentially evolve into the development of an instruction compression/decompression chip (GT–VICD).

Georgia Tech is beginning to investigate the applicability of neural network to enhance the capability of the target detection algorithm. Two neural network application development environments are currently being evaluated. One of them is the Explore Net 3000. The second is the Plexi. The Explore Net 300 runs under PC windows 3.0. The Plexi runs runs under Sunview.

2.5. DP Testing

In this section, the testing of the DP module will be presented. The DP module consists of 4 VLSI chips: GT–VSEQ, GT–VDR, GT–VFPU, and GT–VSNI. The DP module is constructed on a multiwire board. The board measures 1.75"x6.25". This module is connected to a Multibus I wirewrapped board for testing. A GT–VSM8 chip is wirewrapped on the Multibus I board to allow communication between two DPs.

2.5.1. Monitor

The monitor program provides a set of instructions to control the DP resources. Among the instruction supported are *stop system, start system, stop processor, start processor, memory write, memory read, test seq, test dr, test sni, and reset SM8*. The monitor serves as a basic debugging tool during the course of the test program development. Program listing of the monitor is inlcuded in Appendix B.

2.5.2. Instruction Memory

The instruction memory testing consists of two parts. The first part tests the GT–VSEQ instruction memory. There are 512 words of on–chip instruction memory. Each instruction word is 27 bits wide. The second part tests the GT–VDR instruction memory. GT–VDR has 512 words of memory. Each instruction word is 40 bits wide.

Each memory module is excercised with the data patterns 5555, aaaa, ffff, 0000, walking 1's, and walking 0's. Furthermore, to check that each memory word location is unique, the memory array was loaded with a unique data pattern on each word location. These data patterns were then read back for verification.

The source listing of the test program is included in Appendix B. The main routine calls two procedures to perform the actual testing. The procedures are listed in the utility module at the end of the Appendix B.

2.5.3. *Compiler*

A Pascal (subset) compiler is used to program the DP. The supported Pascal syntax supportted includes for–loop, while–loop, if–then–else–construct, procedure, function, and arbitratry boolean, arithmetic, and logical expressions. The compiler is written in Turbo Pascal 4.0. Due to the size of the compiler source code (~6500 lines), the source code is not included in this report. The source code listing will be available upon request.

A branch filter is used to collapse the ouput file of the compiler which produces branch lookahead pointers. The source code listing of the branch filter is listed in Appendix B.

2.5.4. *Loader*

The loader is used to down load a program to the DP module for execution. It consists of two parts: data constant loading and instruction loading. The data constant loading involves the loading of a program to the DP which allows the DP to fetch pointers and data constants from the host. The DPwrites the data constants to some memory location in the data memory designated by the pointers. Each time a data constant is written to the data memory, the DP sends the data constant back to the host for verification. After the data constants are loaded, the loader downloads the program instruction memory. Before completing the loading process, the loader reads all the instruction memory for verification. The source code listing of the loader can be found in Appendix B.

2.5.5. *I/O*

I/O test is used to verify that the DP is able to communicate with the host correctly. Data patterns 5555, aaaa, ffff, 0000, walking 1's, and walking 0's are used to verify the host communication channel. Appendix B contains a source code listing of the program.

2.5.6. *Mandelbrot*

Mandel_brot is a computational intensive algorithm that involves interactive computation of a complex number. A color is assigned to each pixel of an image depending on the converging point of the complex number in the Mandelbrot complex space. This result in a simulated image that mimics the fractal phenomena in natural scenes. This test provides a good exercise for the DP sequencer and floating point arithmetic unit. Appendix A contains a source code listing of the program. A copy of the ASCII object file is also provided for refrence.

2.5.7. *Satellite Attitude Control*

The Satellite Attitude Control is a closed loop system for the control of the attitude of a satellite. It is a 7^{th} order differential equation with a "bang–bang" controller and a pure time delay. This problem represents a class of problem targetted for the DP. Appendix A contains a source code listing of the program. A copy of the ASCII object file is also provided for refrence.

2.5.8. Future Tests

A test will be devised to allow multiple DP modules to communicate with each other through the GT–VSM8 crossbar switch. This will require the development of a crossbar compiler for the GT–VSM8 chip.

A second test involves the construction of a PFP crossbar interface which will allow the DP to communicate with the PFP testbed. This interface will be designed around a GT–VSNI serial network interface chip.

The third test will be running the flight software from the EXOSIM on the DP in a real–time, closed–loop simulation with the PFP.

2.6. Design Schedules and Milestone

The VLSI development schedules and milestones are shown in Figure 2.

Five DP VLSI chips have completed fabrication and chip testing. The five DP chips are currently being used in a DP processor module for testing. These chips are GT–VSEQ, GT–VDR, GT–VFPU/1, GT–VSNI, and GT–VSM8.

Three SP VLSI chips have also completed fabrication. The three SP chips are currently being tested. The chips are GT–VSF, GT–VCLS, and GT–VCTR. Two more SP chips, GT–VTF and GT–VNUC, are in the design stage.

The EP VLSI chips, GT–VIAG and GT–VDAG, are in the final stages. The GT–VDAG chip is about to go out for DV. The GT–VIAG is in the manufacturing test vector generation stage.

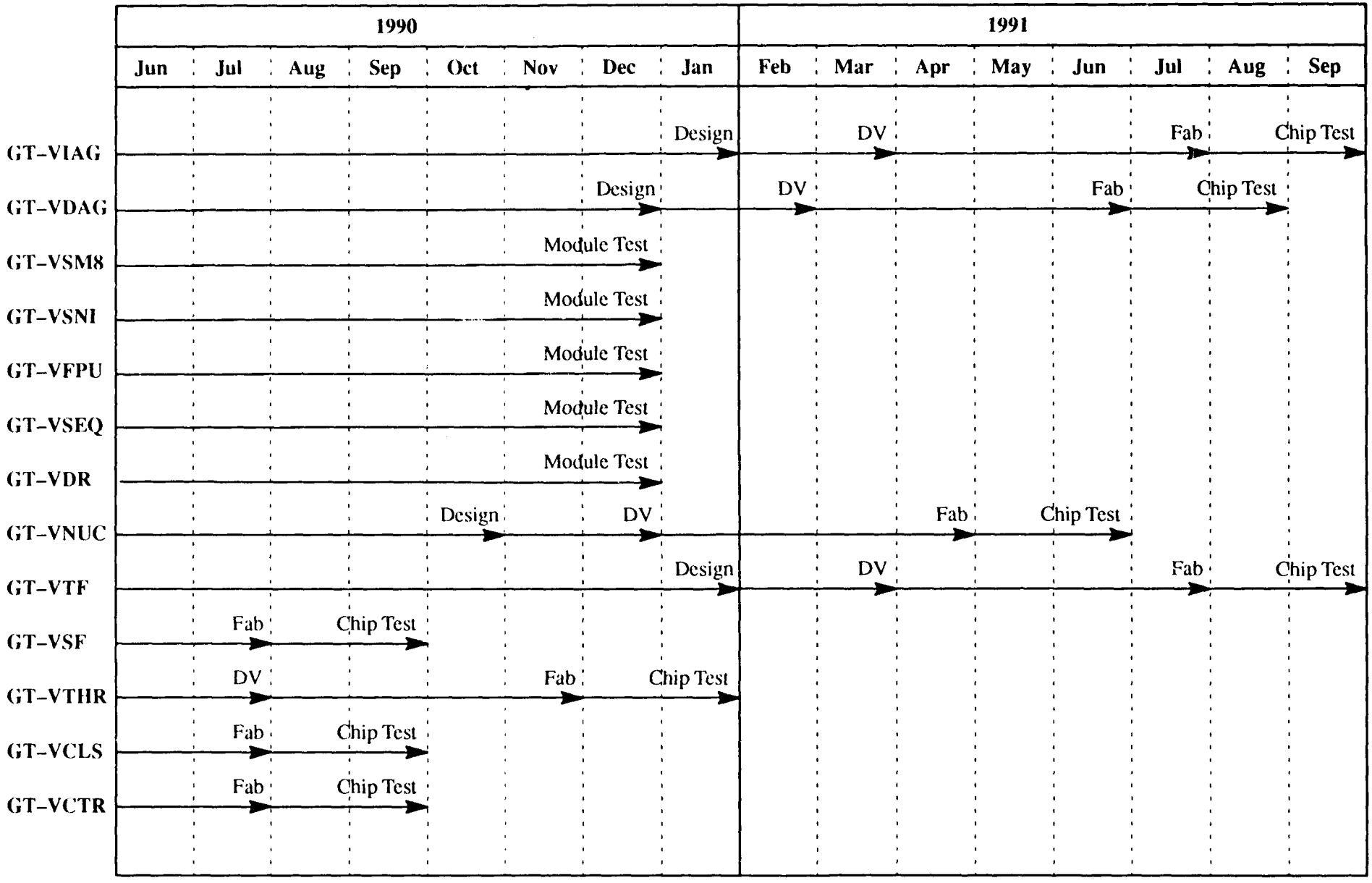


Figure 2. VLSI Development Design Schedules and Milestones

3. Software Development

3.1. Overview of Software Architecture

Georgia Tech is devising an integrated parallel programming framework (IPPF) for the development of software for the special purpose parallel processor architectures. The IPPF serves as an entity for the integration of diverse hardware components and provides a consistent interface for effective exploitation of hardware capabilities. The IPPF consists of four components: user interface, compilation and execution, operating and monitoring system, and database and tool integration. The overall software architecture of the IPPF is shown in Figure 3.

3.1.1. User Interface

The conventional programming method uses direct text editing of program source code. This mode of programming is still supported in the IPPF environment. However, the primary method of programming is through a block editor and block diagram editor. Using the block editor, the user creates and defines basic functional programming blocks. Each functional block is represented by a graphical block diagram. Data flowing into and out of the block are represented by input and output connection ports. The behaviour of the block is represented by a self–contained code segment with receive commands for information flowing into the block and send commands for information flowing out of the block. The block diagram editor allows a user to assemble predefined functional blocks and connect the blocks into a higher level system of functional blocks. Using this method, a complex application program can be graphically and hierarchically developed.

The configuration editor and monitoring specification allows a user to effectively control hardware resources and specify specific monitoring information for an application run. The default hardware configuration is automatically extracted from the application block diagram. Manual configuration is only needed if optimization around a particular hardware configuration is desired.

The text editor, block editor, block diagram editor, configuration editor, and monitor specification are incorporated into an integrated mouse–controlled, menu–driven, graphical environment.

3.1.2. Compilation and Execution

Regardless of the form of user interface, either through block diagram editing or direct program editing, the eventual output from the user interface is program source code. The source code is directed to an appropriate compiler to generate the target object code for execution. The programming languages supported are Ada, Pascal, C, and FORTRAN. A crossbar compiler is used to compile the specification of communication configuration into interconnection patterns between processing elements. Each processor type requires a separate compiler, linker, and loader. Processor specific low level utilities are incorporated into the environment database.

3.1.3. Operating and Monitoring System

The operating system provides efficient run–time system constructs for effective utilization of the underlying capability of hardware resources. It provides a concise interface between the various program-

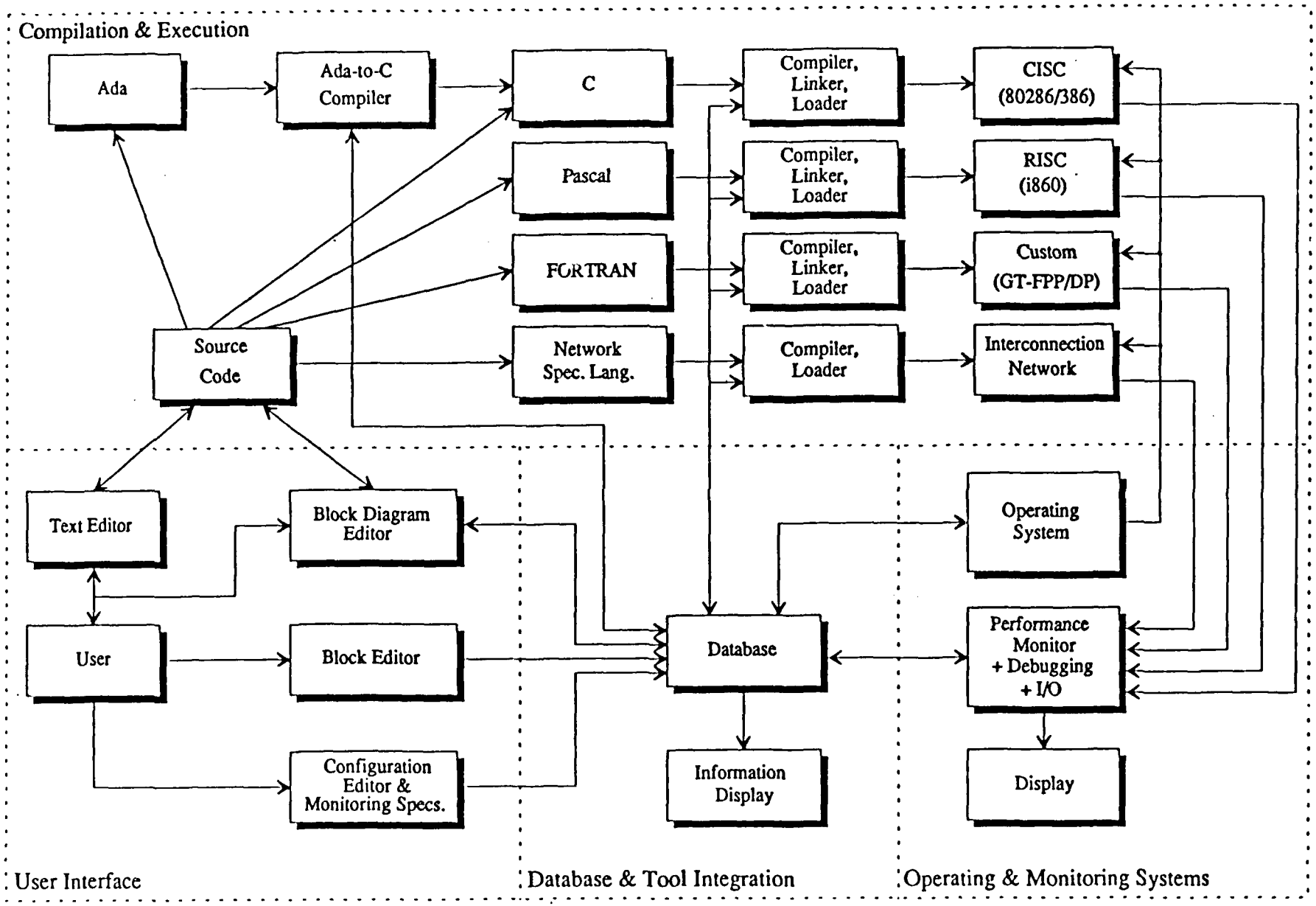


Figure 3. Integrated Parallel Programming Framework Software Architecture

ming languages (Ada, C, Assembly language, etc.) and the underlying architecture of each type of processor element. It also provides a facility for exception handling and error recovery mechanisms. The operating system gives each processor element the ability to execute multiple tasks. This is especially important when the number of functional blocks exceeds the number of available processor elements.
The monitoring system provides a useful mechanism to obtain feedback from an application run. It provides capability for a systematic display of application results and a structured facility for real–time data collection. The monitoring system also serves as the interface for system debugging.

3.1.4. Database and Tool Integration

The software database provides a repository for the block diagrams, configuration information, monitoring specification, language–specific package library, target–machine specific utilities, operating system constructs, and monitoring systems. The database serves as an integration tool for the other three components of the IPPF. A database interface provides constructs to create, access, update, and display the information in the database.

3.2. Characteristics of Parallel Programming Environments

Highly parallel architectures offer opportunities for significant improvements in program execution speeds and reliability. However, these opportunities cannot be realized unless effective program development tools are available. A parallel programming environment —— henceforth termed PPE —— differs from conventional program development systemsin several ways.

3.2.1. Explicit and implicit parallelism.

A program's parallelism must be expressed (1) explicitly by the programmer and/or (2) implicitly as part of a PPE's "compilation" of program source. Typically, larger to medium grain parallelism may be expressed explicitly. The effectiveness of implicit parallelism has been demonstrated for smaller grain parallelism and for specific application domains (e.g., for functional descriptions of real–time simulations).

For a PPE, (1) and (2) imply that explicitly expressed parallelism should be visible and easily modifiable by applications programmers, whereas implicit parallelism may be visible but need not be directly modifiable. For the real–time simulations addressed by the PFP project, we will use domain–specific representations of parallelism, in the form of graphical descriptions of functional decompositions.

3.2.2. Parallelism and target parallel machines.

Typically, programming models have to be specialized for specific target parallel machines and operating systems. For example, for many hypercube applications, synchronous styles of programming have been shown effective, whereas asynchronous (or chaotic) algorithms have been highly successful on non–uniform memory multiprocessors. Therefore, any PPE that attempts to provide support for parallel programming needs to be able to support multiple backend parallel machines and it must also support the use of multiple programming models. For the PFP multicomputer, a synchronous style of programming appears appropriate.In addition, a PPE must be able to use standard programming languages, such as Ada and C.

3.2.3. Parallelism and application domains.

For the explicit expression of parallelism, the programming model presented to the programmer should address the specific properties of the programmer's application domain. For example, in real–time simulations, low–level control functions are easily described as statically decomposed collections of communicating functional blocks. This suggests that a useful programming model is one that presents the functional (possibly replicated) building blocks in the application using graphical descriptions. This is the approach we will pursue for the PPE being constructed for PFP, in conjunction with visual illustrations of the performance effects of such decompositions.

More importantly, one of the environment's attributes will be its ability to exploit application domain–specific knowledge for assistance in parallel programming. For example, when performing resource allocation for PFP's real–time simulations (e.g., mapping functional blocks to processors), the system will use built–in mapping functions, thereby removing from programmers the responsibility of computing and enforcing such mappings.

3.2.4. Performance evaluation and improvement.

Since the primary objective of parallel computing is performance improvement, a PPE must assist the programmer in gaining and understanding of program performance on the target parallel machine. This implies (1) that tools for program monitoring, performance evaluation or prediction and for the visualization of performance information should be integral parts of the programming system and (2) that programmers should be assisted in making changes to their parallel applications in response to such evaluations or predictions — termed program tuning. We will address this issue in the context of the abstract information representation described next. Specifically, we will assume that the PPE should provide a general framework that makes effective use of a wide variety of performance display, evaluation, and visualization tools.

3.2.5. Abstract information representation.

Whether programs are evaluated and tuned by inspection and alteration of their parallel structure, of resource allocation decisions, or of specific program components, the representation and manipulation of the program required for making such changes should be straightforward. This requires that a PPE maintain abstract representations of the executable version of the parallel program and of its execution environment that are easily inspected and manipulated and contain sufficient information for program generation and alteration, as well as for the visualization and evaluation of its performance. This abstract representation must describe parallelism in a fashion that is "neutral" with respect to the program representation at runtime and the program description at compile–time. The representation developed as part of this research is termed program views. It describes a parallel program at two levels: (1) at the high level as a set of interacting and related entities with certain attributes using a data model similar to the entity–relationship database model and (2) at a lower level as a set of interacting processes, where process interactions are described as messages sent and received on abstract communication channels. In addition, since multiple components of (tools in) a PPE must jointly access or manipulate the information repository(ies) associated with the information model, the model is defined such that the repository(ies) offers the following function-

ality: (1) facilities for sharing information between cooperating tools, (2) facilities for tool control via the shared repository, and (3) facilities for tool observation schwantr. Topics (1)–(3) are discussed in 2 below.

3.2.6. Operating software and parallel machines.

Since high–performance parallel applications must make efficient use of the target system's hardware, runtime software actually consists of both application code and operating system code, and program tuning also concerns the performance of the operating system components used by the application. Specifically, tuning may concern operating system configuration, such as the selection among alternative operating system components, the determination of relevant OS parameters (e.g., setting buffer sizes for message communication), or even the synthesis or construction of appropriate OS components in conjunction with the generation of executable program code. This implies that the proposed PPE cannot be constructed with the assumption that all parallelism is mapped to a single set of runtime constructs. Instead, the "operating system" actually consists of a variety of runtime primitives configurable by the application programmer.

3.3. Technologies for Parallel Programming Environments

The successful development of a PPE requires the following technologies:

- Language and compiler technology for the support of multiple models of parallel programming (when explicitly describing parallelism) and for the automatic or semi–automatic parallelization of programs. Here, we are using and developing Ada and C compilers, linkers, and loaders, enhanced with libraries and special–purpose packages for inter–processor communication and coordination on the target parallel machine.

- For program generation, analysis, and improvement: programming environment, database, and visualization technology for the representation, sharing, and display of information about the parallel program, its execution environment, and its runtime performance. This is the PPE being developed in this research.

- Performance analysis, program specification techniques, and, perhaps, Artificial Intelligence technology (1) for performance modeling of the parallel program, (2) for expression of relevant performance attributes of parallel programs or of program invariants not to be changed during performance tuning, and (3) for relating models and measurements to actual program code as well as for suggesting and making changes to such code. Here, we will offer simple means of performance evaluation and program visualization, coupled to a graphical interface used for program development.

- Operating system technology for efficient performance monitoring and for the efficient execution of parallel programs on different target parallel machines. Here,

we have developed a configurable and portable operating system kernel presenting a "library" of primitives used by the programmer. Portability is essential in light of the multi–CPU nature of any PFP machine, which may contain both special–purpose processors (such as the FPP) and general–purpose processors like the Intel 386 or 860 boards.

It is beyond the scope of a single research effort to develop novel technologies in all of the areas listed above. Therefore, for purposes of this work, we are focusing on operating systems and programming environment technology:

Compiler technology and environment tools. Due to the special–purpose nature of the target hardware, we have been designing and implementing a variety of tools for hardware use, including low–level device drivers, system monitoring and configuration software, compilers and assemblers for the target parallel machine, loaders, linkers, etc. In addition, significant efforts have been expended on facilitating system installation, by use of Unix Makefiles.The environment is structured as a collection of tools sharing a common information store, called the abstract information representation.

Abstract information representation and tool integration. As a prerequisite for the support of multiple models of parallel programs, we have designed and implemented a high–level information model and associated information repository that can be used to represent relevant program information (compile–time and runtime information, information about the program and its execution environment). In addition, we have designed the model so that it and the repository may be used with multiple tools for information sharing, mutual observation, and mutual control.

Performance evaluation, improvement (tuning), and visualization. We have prototyped a graphical user interface to the system and are now using our experiences with this prototype to implement a framework for the graphical construction of parallel programs and for the visualization of their performance.

Operating system technology. We have developed a configurable kernel that has been ported to both the special–purpose processors (FPP) on PFP as well as the general–purpose CPU's (currently, the Intel 386). Kernel communication primitives offer substantially increased functionality compared to that offered by the basic hardware, yet are quite competitive in performance.

Future work may concern the use of graphical techniques for program tuning and an investigation of the role of precise specifications of program functionality or performance (e.g., performance models) in the tuning of program performance.

3.4. Current State of Related Technologies

It would not be possible to perform the research proposed above without the existence of two other substantial, current research efforts at Georgia Tech: (1) the CHAOS project, and (2) the PaCTech laboratory, which jointly address the development of operating system support for high–performance parallel machines, with a current emphasis (a) on operating systems for highly dependable real–time systems (on a BBN Butterfly

multiprocessor and a Sequent Symmetry) and (b) on domain–specific programming system support (e.g., for finite element applications) on the BBN Butterfly.

The technical developments in those efforts have immediate technical impact on our work with PFP.

In the CHAOS project, the programming environment technology currently being developed permits us to explore the representation of program information used for automatic parallelization, both in terms of the object model and using higher–level information models. The representations of such information are based on our past research in parallel programming environments. Such environments were constructed by layering them on top of existing sequential languages, and the program information shared between the environments' different tools consisted of program monitoring data, resource allocation information, and instructions concerning program tuning. Such research has used relational information models and compiler–oriented representations for information representation. The experimental information repository constructed as part of such research was successfully used for tool integration. Therefore, the prototype systems constructed with the concepts developed in our previous work explored the types of information to be shared between tools and experimented with the possible uses of such information, resulting in experimentation with program tuning on a network of SUN workstations, with program adaptation for an embedded real–time multiprocessor, and with the display of program performance information for an iPSC hypercube. We will use the examples of information to be represented and shared among tools, as well as the associated tool interactions experienced in our past work for the effort proposed here.

Current operating system developments in the CHAOS project relevant to this proposal include the customization of operating system kernels for high–performance or highly dependable parallel applications, on shared memory and on distributed memory multiprocessors. Such work has been based on the object model of software. Substantial experiences with large–scale engineering and scientific applications and with real–time applications have resulted in the development of a family of operating system kernels supporting multiple forms of the object model, the definition of a notion of "distributed objects" for distributed memory machines, and the development of algorithms for automatic program tuning for the real–time domain. We are using our experiences regarding the runtime representation of parallel software to ensure the implementability of the PPE's internal representations of parallel software. Ideally, no differences would exist between the PPE's and the operating system's object models of software. In practice, such differences appear unavoidable due to differences in the objectives of both model designs (e.g., reusability in the PPE vs. high performance in the OS).

Regarding parallel applications, in the PaCTech laboratory, we are studying a range of application programs jointly with domain experts. Current developments concern: the real–time simulations being developed as part of the PFP project and other real–time applications developed as part of the CHAOS project, parallel finite element methods, parallel computer graphics, parallel event simulation, and parallel command and control applications. These applications are at our disposal for testing the concepts of the PFP parallel programming environment being constructed.

3.5. Current Status of the Software Development

The current implementation of the Integrated Parallel Programming Framework (IPPF) consists of four primary tool sets:

- Parallel Program Construction System (PPCS),
- Parallel Program Monitoring System (PPMS),
- Parallel Program Tuning System (PPTS), and
- Tool Integration System (TIS).

A programmer uses the PPCS for the initial development of a parallel application. When the program is sufficiently complete, the programmer runs it and gathers performance data with the PPMS. Based on the performance measurements, more development with the PPCS might be necessary or the PPTS can be used to tune the completed application for the target execution environment. These three tool sets are supported by and coordinated by the TIS.

The PPCS encompasses all tools used during program development: a graphical user interface, recompilation drivers, compilers, assemblers, program libraries, linkers, loaders. Currently, the PFP PPCS has a prototype graphical user interface (called "bde" or "Block Diagram Editor") which runs under the X window system, a sophisticated system of Makefiles and shell scripts which drive the recompilation process, The Makefiles and scripts not only drive the recompilation of application programs but also the recompilation and installation of the PFP environment itself. a compiler which translates a subset of Ada to C, a C compiler for the FPP, assemblers for both the FPP and FPX, a linker for the FPP, and loaders for the FPP, FPX and Intel 386 processors. These tools were all designed and implemented in-house. Additionally, there are vendor-supplied compilers, assemblers, and linkers for the Intel 386 processor and vendor-supplied and publicly available compiler tools and text editors for the Sun host environment. Much work has been done in supporting the FPP: the C compiler, under development for over a year and a half, has been subjected to a C test suite (the so-called "C Torture Test") and appears to be stable. The compiler is currently being targeted to the FPX processor. We have purchased a commercial Ada to C translator This is part of a validated Ada compiler from Irvine Compiler Corporation. to replace our Ada-subset compiler and we are beginning to integrate it with the rest of the IPPF.

The PPMS is currently being designed based on experience with the CHAOS and PaCTech efforts described above. Other than this design work, a rudimentary implementation exists for verifying timing characteristics of real-time simulations, but it does not yet provide any hints on where performance problems exist within an application or how it might be modified to increase its performance. A less rudimentary implementation which indicates communication bottlenecks is underway.

The PPTS currently consists primarily of a compiler which produces process-to-processor mappings and interprocessor communication patterns. The language for this compiler and the compiler itself

are currently being redesigned. The PPTS will ultimately include a graphical user interface where a programmer can easily reconfigure and tune his parallel program for its target execution environment.

The TIS provides the basis for the PPCS, the PPMS, and the PPTS to work together effectively and efficiently. It consists of several tools: a number of device drivers to control the PFP hardware, a library supporting host program interaction with the hardware and the parallel programs it is executing, an automatic hardware configuration tool, The configuration tool also serves as an effective hardware testing and diagnostic device. and a newly–completed entity–relationship–set database system which implements the abstract information representation described above. Currently, each of these tools is implemented. We are in the process of refitting some of the pieces of the PPCS, PPMS, and PPTS to use the database system.

The current PFP environment and the associated PFP configurable kernel has supported the development of several real–time applications:

a Satellite Attitude Control Simulation
a 3–DOF Missile Simulation (Terminal Phase)
a Multiple Threat/KEW Interceptor Simulation enumerate

The implementation of the operating kernel is detailed in the *PFP Kernel Design* Document which is submitted under a separate techinical report. The structure of the environment database is described in the *Shared Persistent Data Structures* design document which is also submitted under a separe techinical report.

3.6. Future Plan

Future directions of the PFP IPPF development are described in the **PFP Software Development Plan (SDP)** submitted under a specifal technical report.

APPENDIX A

SOURCE CODE LISTING

GAMMA SUPRESSION ALGORITHMS

MONTE CARLO SIMULATION

```
{ generates normal distributed random numbers }
uses crt,{$U c:\tp4\graph.tpu} graph,
   minimum;

label start;
const debug = false;
    max_signal_amplitude = 1500.0;
    zero_y_axis = 479;
    no_lambda = 100;
    lambda_increment = 100;
    frame_interval = 89e-6;

var
   iset : byte;        { used exclusively by function gaussian }
   x_normal_1 : single; { used exclusively by function gaussian }
   target_counter : integer; { used exclusively by function target }
   median_store : array[1..9] of single; { for function median }
   lambda_array : array[1..no_lambda] of single; { for function gamma }
   lambda : single;
   gamma_events : integer;
   poisson0 : single; { for function poisson }
   plot_counter,plot_pointer : integer;  { for plot }
   plot_store : array[1..639] of integer;
   plot_p1,plot_p2 : pointer;
   plot_old_y : integer;
   image_size : word;

   x : longint;
   y1,y2,y3,y_scale : single;
   filter,option : integer;
   signal,noise : single;
   total_signal : double;
   total_direct_noise : double;
   total_median_noise : double;
   total_threshold_noise : double;
   total_median_of_median_noise : double;
   total_minimum3_noise : double;
   total_minimum5_noise : double;
   total_minimum7_noise : double;
   no_snr_samples : longint;
   i : integer;
```

```
  outfile : text;
  start_lambda_index : integer;

procedure initialize;
var graphdriver,graphmode : integer;
  i : integer;
  x1,x2,x3 : single;
begin
 graphdriver := vga;
 graphmode := vgahi;
 if not debug then
 begin
  initgraph(graphdriver,graphmode,'c:\tp4');
  setcolor(blue);
  line(0,0,0,479);
  line(0,zero_y_axis,639,zero_y_axis);
 end;
 randomize;
 iset := 0;
 y_scale := zero_y_axis/max_signal_amplitude;
 for i := 1 to 9 do median_store[i] := 0;
 target_counter := 0;
 plot_pointer := 639;
 plot_counter := 1;
 image_size := ImageSize(1,0,639,200);
 getmem(plot_p1,image_size);
 image_size := ImageSize(1,201,639,400);
 getmem(plot_p2,image_size);
 total_signal := 0;
 total_direct_noise := 0;
 total_median_noise := 0;
 total_threshold_noise := 0;
 lambda_array[1] := 0;
 for i := 2 to no_lambda do
  lambda_array[i] := lambda_array[i-1] + lambda_increment;
 lambda := frame_interval*lambda_array[4];
 poisson0 := exp(-lambda);
end;
```

```
procedure swap(var x,y:single);
var t : single;
begin
 t := x;
 x := y;
 y := t;
end;

function target:single;
const target_probability = 0.005;
    target_duration = 100;
    target_intensity = 1180;
    background_intensity = 1126;
begin
 if target_counter = 0 then
 begin
  if random <= target_probability then
   target_counter := target_duration;
 end;
 if target_counter > 0 then
 begin
  target_counter := target_counter-1;
  target := target_intensity;
 end
 else
  target := background_intensity;
end;

function gaussian_noise:single;
const sigma = 50;
var v1,v2,r,fac,x_normal:single;
   { iset and x_normal_1 are declared as global variables to retain
    the state from the previous function call }
begin
 if iset=0 then
 begin
  repeat
   v1 := 2.0*random-1.0;
   v2 := 2.0*random-1.0;
   r := v1*v1+v2*v2;
```

```
  until r < 1;
  fac := sqrt(-2.0*sigma*ln(r)/r);
  x_normal_1 := v1*fac;
  x_normal := v2*fac;
  iset := 1;
 end
 else
 begin
  x_normal := x_normal_1;
  iset := 0;
 end;
 gaussian_noise := x_normal;
end; { of gaussian }

function poisson:integer;
var t : single;
   x_poisson : integer;
begin
 t := 1.0;
 x_poisson := -1;
 while (t > poisson0) do
 begin
  t := t*random;
  x_poisson := x_poisson+1;
 end;
 poisson := x_poisson;
end;

function gamma_noise:single;
const mean = 9690.0;
     gamma_probability = 0.089;
var   x : single;
     i : integer;
     noise : single;
begin
{ if random < gamma_probability then
 begin
  repeat x := random; until x <> 1.0;
  gamma_noise := -ln(1-x)*mean;
 end
```

```
else
  gamma_noise := 0; }
 gamma_events := poisson;
 noise := 0;
 for i := 1 to gamma_events do
 begin
   repeat x := random; until x <> 1.0;
   noise := noise – ln(1–x)*mean;
 end;
 gamma_noise := noise;
end;

function median(x :single):single;  { 9 points }
var i,j : integer;
   gs : array[1..9] of single;
begin
  for i := 8 downto 1 do
  begin
    median_store[i+1] := median_store[i];
  end;
  median_store[1] := x;
  for i := 1 to 9 do
  begin
    gs[i] := median_store[i];
  end;
  for i := 1 to 8 do
    for j := i+1 to 9 do
      if gs[i] > gs[j] then swap(gs[i],gs[j]);
  median := gs[5];
 end;

 function median_of_median(x : single):single; { 3 of 3}
 var i,j : integer;
    gs : array[1..9] of single;
 begin
   for i := 8 downto 1 do
   begin
     median_store[i+1] := median_store[i];
   end;
   median_store[1] := x;
```

```
 for i := 1 to 9 do gs[i] := median_store[i];
 for i := 1 to 2 do
  for j := i+1 to 3 do
   if gs[i] > gs[j] then swap(gs[i],gs[j]);
 for i := 4 to 5 do
  for j := i+1 to 6 do
   if gs[i] > gs[j] then swap(gs[i],gs[j]);
 for i := 7 to 8 do
  for j := i+1 to 9 do
   if gs[i] > gs[j] then swap(gs[i],gs[j]);
 if gs[2] > gs[5] then swap(gs[2],gs[5]);
 if gs[2] > gs[8] then swap(gs[2],gs[8]);
 if gs[5] > gs[8] then swap(gs[5],gs[8]);
 median_of_median := gs[5];
end;

function GRIS(x:single):single;
begin

end;

function threshold(x : single):single;
const threshold_constant = 2000;
begin
 if x > threshold_constant then threshold := 0 else
                        threshold := x;
end;

procedure plot1(x:integer;ry:single;color:integer);
var y : integer;
begin
 ry := zero_y_axis-ry*y_scale;
 if (ry < 0) then
 begin
  ry := 0;
 end
 else
 if (ry > 479) then
 begin
  ry := 479;
 end;
 y := round(ry);
```

```
 setcolor(color);
 line(x,zero_y_axis,x,y);
end;

procedure plot2(ry:single;color:integer);
var y,x,xt1,xt2 : integer;
   sy,sx,sxt1,sxt2 : string;
begin
 ry := zero_y_axis-ry*y_scale;
 if (ry < 0) then
 begin
  ry := 0;
 end
 else
 if (ry > 479) then
 begin
  ry := 479;
 end;
 y := round(ry);
 if plot_counter < 640 then
 begin
  setcolor(color);
  line(plot_counter,zero_y_axis,plot_counter,y);
  plot_store[plot_counter] := y;
  plot_counter := plot_counter+1;
 end
 else
 begin
  if plot_pointer = 639 then xt1 := 1
                  else xt1 := plot_pointer+1;
  for x := 1 to 638 do
  begin
   if xt1 = 639 then xt2 := 1 else xt2 := xt1 + 1;
   setcolor(black);
   line(x,zero_y_axis,x,plot_store[xt1]);
   setcolor(color);
   line(x,zero_y_axis,x,plot_store[xt2]);
{    if (plot_store[xt2] <= plot_store[xt1]) then
    setcolor(color)
   else
```

```
      setcolor(black);
     line(x,plot_store[xt1],x,plot_store[xt2]); }
     xt1 := xt2;
    end;
    if y <= plot_store[plot_pointer] then setcolor(color) else setcolor(black);
    line(639,plot_store[plot_pointer],639,y);
    if plot_pointer = 639 then plot_pointer := 1
                  else plot_pointer := plot_pointer+1;
    plot_store[plot_pointer] := y;
  end;
{ moveto(10,300);
  setcolor(color+iset);
  outtext('exit'); }
end;

procedure plot(ry:single;color:integer);
var y,x,xt1,xt2 : integer;
    sy,sx,sxt1,sxt2 : string;
begin
  ry := zero_y_axis-ry*y_scale;
  if (ry < 0) then
  begin
    ry := 0;
  end
  else
  if (ry > 479) then
  begin
    ry := 479;
  end;
  y := round(ry);
  if plot_counter < 640 then
  begin
    setcolor(color);
    line(plot_counter,zero_y_axis,plot_counter,y);
    plot_store[plot_counter] := y;
    plot_counter := plot_counter+1;
  end
  else
  begin
    setcolor(yellow);
```

```
   getimage(1,0,639,200,plot_p1^);
   getimage(1,201,639,250,plot_p2^);
   putimage(0,0,plot_p1^,normalput);
   putimage(0,201,plot_p2^,normalput);
   if keypressed then exit;
   setcolor(black);
   line(639,zero_y_axis,639,plot_old_y);
   setcolor(yellow);
   line(639,zero_y_axis,639,y);
  end;
  plot_old_y := y;
end;

begin
  clrscr;
  writeln('Display options  : auto-shift        (1)');
  writeln('                   auto-refresh      (2)');
  writeln('                   manual-refresh <CR> (3)');
  writeln('                   calculate S/N ratio (4)');
  write  ('                   debugging         (5) ? ');
  readln(option);
  if option <> 4 then
  begin
    writeln('Filter options   : no filter       (1)');
    writeln('                   median          (2)');
    writeln('                   threshold       (3)');
    writeln('                   median of median (4)');
    writeln('                   minimum 3       (5)');
    writeln('                   minimum 5       (6)');
    write  ('                   minimum 7       (7) ?');
    readln(filter);
  end
  else
  begin
    write('number of data points ? ');
    readln(no_snr_samples);
    write('starting lambda array index ? '); readln(start_lambda_index);
  end;
  case option of
  1,2,3:
```

```
 begin
  initialize;
start:
  lambda := frame_interval*1300;
  poisson0 := exp(-lambda);
  for x := 1 to 639 do
  begin
   signal := target + gaussian_noise + gamma_noise;
   case filter of
    1: ;
    2: signal := median(signal);
    3: signal := threshold(signal);
    4: signal := median_of_median(signal);
    5: signal := minimum3(signal);
    6: signal := minimum5(signal);
    7: signal := minimum7(signal);
   end;
   case option of
    1:   if keypressed then exit else plot(signal,yellow);
    2,3: plot1(x,signal,yellow);
   end;
  end;
  case option of
    2: begin
        clearviewport;
        if keypressed then exit;
       end;
    3: begin
        moveto(10,10);
        readln;
        clearviewport;
       end;
   end;
   goto start;
  end;
  4:
  begin
   assign(outfile,'gamma.out');
   rewrite(outfile);
```

```
    initialize;
    closegraph;
    for i := start_lambda_index to no_lambda do
    begin
     total_direct_noise := 0;
     total_median_noise := 0;
     total_threshold_noise := 0;
     total_median_of_median_noise := 0;
     total_minimum3_noise := 0;
     total_minimum5_noise := 0;
     total_minimum7_noise := 0;
     total_signal := 0;
     lambda := frame_interval*lambda_array[i];
     poisson0 := exp(-lambda);
     for x := 1 to no_snr_samples do
     begin
      signal := target;
      noise := gaussian_noise + gamma_noise;
      total_signal := total_signal + signal;
      total_direct_noise := total_direct_noise + abs(noise);
      total_median_noise := total_median_noise + abs(signal-median(signal+noise));
      total_threshold_noise := total_threshold_noise + abs(signal-threshold(signal+noise));
        total_median_of_median_noise := total_median_of_median_noise + abs(signal-median_of_me-
dian(signal+noise));
      total_minimum3_noise := total_minimum3_noise + abs(signal-minimum3(signal+noise));
      total_minimum5_noise := total_minimum5_noise + abs(signal-minimum5(signal+noise));
      total_minimum7_noise := total_minimum7_noise + abs(signal-minimum7(signal+noise));
      if (x and $fff) = 0 then write('.');
     end;
     writeln;
     writeln(i);
    { writeln(outfile,'lambda = ',lambda_array[i]);
     writeln(outfile,'signal to noise ratio : direct = ',total_signal/total_direct_noise:8:4);
     writeln(outfile,'                      median = ',total_signal/total_median_noise:8:3);
     writeln(outfile,'                   threshold = ',total_signal/total_threshold_noise:8:4);
     writeln(outfile,'           median of median = ',total_signal/total_median_of_median_noise:8:4);
     writeln(outfile,'           minimum          = ',total_signal/total_minimum_noise:8:4);
     writeln(outfile,'              total signal = ',total_signal:8:4);
     writeln(outfile,'         total direct noise = ',total_direct_noise:8:4);
```

```
    writeln(outfile,'          total median noise = ',total_median_noise:8:4);
    writeln(outfile,'       total threshold noise = ',total_threshold_noise:8:4);
    writeln(outfile,' total median of median noise = ',total_median_of_median_noise:8:4);
    writeln(outfile,'         total minimum noise = ',total_minimum_noise:8:4); }
    write(outfile,lambda_array[i],chr(9));
    write(outfile,total_signal/total_direct_noise,chr(9));
    write(outfile,total_signal/total_median_noise,chr(9));
    write(outfile,total_signal/total_median_of_median_noise,chr(9));
    write(outfile,total_signal/total_threshold_noise,chr(9));
    write(outfile,total_signal/total_minimum3_noise,chr(9));
    write(outfile,total_signal/total_minimum5_noise,chr(9));
    write(outfile,total_signal/total_minimum7_noise,chr(9));
    writeln(outfile);

    if keypressed then
    begin
     close(outfile);
     exit;
    end;
   end;
   close(outfile);
  end;
  5:
  begin
   initialize;
   closegraph;
   lambda := frame_interval*2000;
   poisson0 := exp(-lambda);
   for i := 1 to 1000 do
   begin
    writeln('target     = ',target:6:2);
    writeln('gamma      = ',gamma_noise:6:2);
    writeln('gamma_events = ',gamma_events);
    writeln('gaussian   = ',gaussian_noise:6:2);
    readln;
   end;
  end;
  end; { of case option of }
end.
```

Monitor

```
Program GT_DP_monitor;
uses {$u e:\dp_comp\hex_conv} hex_conv,
    {$u e:\dp_comp\ieee_cnv} ieee_cnv,
    {$u e:\dp\utility} utility,
    crt;
var
  command : string;
  address,data : word;
  addr_off : word;

procedure rd;
begin
  write('address ? '); readln(address);
  writeln('Read data : ',mem[segment:address+addr_off]);
end;

procedure wr;
begin
  write('address ? '); readln(address);
  write('data ? '); readln(data);
  mem[segment:address+addr_off] := data;
end;

procedure mtest;
var device_no : word;
begin
  write('Choose a device : sni (1), dr (2), seq (3), sm8 (4) ?');readln(device_no);
  case device_no of
    1: addr_off := sni_off;
    2: addr_off := dr_off;
    3: addr_off := seq_off;
    4: addr_off := sm8_off;
  end;
  repeat
    write('<mtest>.'); readln(command);
    if command = 'rd' then rd else
    if command = 'wr' then wr
  until command = 'quit';
  command := '';
 end;
```

```
(******************  Main Program **********************)
Begin
 clrscr;
 initialize;

 while true do
 begin
  write('.'); readln(command);
  if command = 'start_system' then start_system else
  if command = 'stop_system' then stop_system else
  if command = 'start' then start_processor else
  if command = 'stop' then stop_processor else
  if command = 'test_sni' then test_sni else
  if command = 'test_dr' then test_dr else
  if command = 'test_seq' then test_seq else
  if command = 'test_sm8' then test_sm8 else
  if command = 'mtest' then mtest else
  if command = 'quit' then halt else
  if command = 'reset_sm8' then reset_sm8 else
  writeln('unknown command');
 end;
end.
```

Instruction test

```
Program test_inst;
uses {$u e:\compile\hex_conv} hex_conv,
    {$u e:\dp_comp\ieee_cnv} ieee_cnv,
    {$u e:\dp\utility} utility,
    crt;
var
  command : string;
  segment : word;
  address,data : word;
  addr_off : word;
  pattern : array[0..40] of word;
  no_data_pattern : word;
(*****************  Main Program **********************)
Begin
 clrscr;
 initialize;
 test_seq;
 Test_DR;
 writeln('Testing completed');
end.
```

Branch Filter

```
program load;
uses {$u e:\dp\dp_comp\hex_conv} hex_conv,
   {$u e:\dp\dp_comp\ieee_cnv} ieee_cnv,
   {$u e:\dp\utility} utility,
   crt;
const

 {SEQ opcode}
 CJ_ALU_T = 0; { conditional jump based on ALU condition with default of true }
 CJ_ALU_F = 1; { conditional jump based on ALU condition with default of false }
 CJ = 2;      { conditional jump based on on others }
 RTN = 3;     { return }
 JS = 4;      { jump subroutine }
 BF = 5;      { begin for }
 EF = 6;      { end for }

 {branch status}
 no_branch = 0; { force condition to be always false }
 if_counter_not_zero = 1;
 if_not_xrfi = 2; { if network is ready to receive data }
 if_not_xdav = 3; { if network has data available }
 if_not_hrfi = 4; { if host port is ready to receive data }
 if_not_hdav = 5; { if host port has data available }
 if_zero = 6; { if ALU result is zero }
 if_not_zero = 7; { if ALU result is not zero }
 if_carry = 8; { if ALU results in a carry }
 if_not_carry = 9; { if ALU does not result in a carry }
 if_negative = 10; { if ALU result is negative }
 if_not_negative = 11; { if ALU result is not negative }
 if_ALU_error = 12; { if error occurs }
 if_net_error = 13; { if network error }
 operand_not_available = 14; { operand is still in the pipeline }
 unconditional = 15; { unconditional branch }

 {ALU_opcode}
 Fix_add = 0;
 Fix_sub = 1;
 Fix_mult = 2;
 Fix_rsub = 3;
 Bit_shl  = 4;
```

```
Bit_shr   = 6;
Bit_and   = 8;
Bit_or    = 9;
Bit_xor   = $a;
Bit_not   = $b;
Pack_exp  = $d;
Inv_Seed  = $e;
Float_round  = $f;
Float_add = $10;
Float_sub = $11;
Float_mult = $12;
Float_rsub = $13;
fix_float = $15;
float_trunc = $16;
unpack_exp  = $18;
unpack_mant  = $19;
sqrt_exp   = $1A;
sqrt_mant   = $1B;
sine_sign   = $1C;
odd_neg    = $1d;
swap_sign   = $1e;
tan_sign    = $1f;
ALU_nop     = $14;
pass_R      = $14;
float_div   = $100; { emulation in software }

{ IO_opcode }
net_send = 1;
net_receive  = 2;
host_send = 3;
host_receive = 4;
IO_nop = 0;

{ R_bus_en }
 en_dataram = 0;
 en_network = 1;

{ idx_opcode }
idx_nop = 0;
load_idx = 1;
```

```
 decr_idx = 2;
 incr_idx = 3;

 buffer_size = 100;

type
  s_buffer_type = record
   pc : word;
   seq_op : word;
   branch_stat : word;
   branch_adr : word;
   idx_op : word;
   f_idx_adr : word;
  end;
  b_buffer_type = record
   pc : word;
   branch_adr : word;
  end;

var pc,seq_op,branch_stat,branch_adr,alu_op,io_op,
  rbus_en,idx_op,f_wr,f_adr_mode,r_adr_mode,s_adr_mode,
  f_idx_adr,r_idx_adr,s_idx_adr,f_off,r_off,s_off:longint;
  board_id : word;
  tmpfile,infile : text;
  filename : string[100];
  seq_inst,dr_inst1,dr_inst0 : longint;
  ch : char;
  i : integer;
  s_buffer : array[0..buffer_size] of s_buffer_type;
  b_buffer : array[0..buffer_size] of b_buffer_type;
  b_ptr,last_b_ptr,s_ptr,last_s_ptr : integer;

Procedure reset_instruction_field;
var i : integer;
begin
 SEQ_op := CJ;
 branch_adr := 0;
 branch_stat := no_branch;
 ALU_op := ALU_nop;
 IO_op := IO_nop;
 Rbus_en := en_dataram;
 idx_op := idx_nop;
```

```
  f_wr := 0;
  f_adr_mode := 0;
  r_adr_mode := 0;
  s_adr_mode := 0;
  f_idx_adr := 0;
  r_idx_adr := 0;
  s_idx_adr := 0;
  f_off := 0;
  r_off := 1;
  s_off := 1;
end;

procedure query_filename;
begin
  write('input file : ');
  if paramcount > 0 then
  begin
    filename := paramStr(1);
    writeln(filename);
  end
  else
    readln(filename);
end;

procedure initialize;
begin
  query_filename;
  reset_instruction_field;
  s_ptr := 0;
  last_s_ptr := 0;
  b_ptr := 0;
  last_b_ptr := 0;
  assign(infile,filename+'.dp');
  {$I-} Reset(infile) {$I+};
  if (IOresult <> 0 ) then
  begin
    writeln(filename,'.dp does not exist. ');
    halt;
  end;
  assign(tmpfile,filename + '.err');
```

```
 rewrite(tmpfile);
end;

Procedure output_instruction_field;
begin
 write(infile,'c ',pc:6);
 write(infile,SEQ_op:4,branch_stat:3,branch_adr:6);
 write(infile,ALU_op:4,IO_op:3,Rbus_en:3,idx_op:3,f_wr:3);
 write(infile,f_adr_mode:3,r_adr_mode:3,s_adr_mode:3);
 write(infile,f_idx_adr:3,r_idx_adr:3,s_idx_adr:3);
 writeln(infile,f_off:6,r_off:6,s_off:6);
end;

procedure sort_s_buffer;
var t : s_buffer_type;
   i,j : integer;
begin
 for i := 1 to last_s_ptr-1 do
 begin
  for j := i+1 to last_s_ptr do
  begin
   if s_buffer[i].pc > s_buffer[j].pc then
   begin
    t:=s_buffer[i];
    s_buffer[i]:= s_buffer[j];
    s_buffer[j]:=t;
   end;
  end;
 end;
end; { sort_s_buffer }

procedure sort_b_buffer;
var t : b_buffer_type;
   i,j : integer;
begin
 for i := 1 to last_b_ptr-1 do
 begin
  for j := i+1 to last_b_ptr do
  begin
   if b_buffer[i].pc > b_buffer[j].pc then
   begin
```

```
      t:=b_buffer[i];
      b_buffer[i]:= b_buffer[j];
      b_buffer[j]:=t;
     end;
   end;
  end;
end; { sort_b_buffer }

procedure load_buffers;
begin
 writeln('load branch buffers');
 while not eof(infile) do
 begin
  read(infile,ch);
  if (ch='s') then
  begin
   s_ptr := s_ptr+1;
   with s_buffer[s_ptr] do
   begin
    read(infile,pc);
    readln(infile,seq_op,branch_stat,branch_adr,idx_op,f_idx_adr);
   end;
  end
  else
  if (ch='b') then
  begin
   b_ptr := b_ptr+1;
   with b_buffer[b_ptr] do
   begin
    read(infile,pc);
    readln(infile,branch_adr);
   end;
  end
  else
  begin
   write(tmpfile,ch);
   while not eoln(infile) do
   begin
    read(infile,ch);
    write(tmpfile,ch);
```

```
    end;
    readln(infile);
    writeln(tmpfile);
   end;
  end;
  last_s_ptr := s_ptr;
  s_ptr := 1;
  last_b_ptr := b_ptr;
  b_ptr := 1;
  sort_s_buffer;
  sort_b_buffer;
  close(tmpfile);
end; { load_buffer }

procedure modify_code;
begin
  writeln('modify code');
  reset(tmpfile);
  rewrite(infile);
  while not eof(tmpfile) do
  begin
   read(tmpfile,ch);
   if ch='c' then
   begin
    read(tmpfile,pc);
    read(tmpfile,seq_op,branch_stat,branch_adr,alu_op);
    read(tmpfile,io_op,rbus_en,idx_op,f_wr);
    read(tmpfile,f_adr_mode,r_adr_mode,s_adr_mode);
    read(tmpfile,f_idx_adr,r_idx_adr,s_idx_adr);
    readln(tmpfile,f_off,r_off,s_off);
    if s_ptr <= last_s_ptr then
    begin
     if s_buffer[s_ptr].pc=pc then
     begin
      seq_op := s_buffer[s_ptr].seq_op;
      branch_stat := s_buffer[s_ptr].branch_stat;
      branch_adr := s_buffer[s_ptr].branch_adr;
      idx_op := s_buffer[s_ptr].idx_op;
      f_idx_adr := s_buffer[s_ptr].f_idx_adr;
      s_ptr := s_ptr + 1;
```

```
      end;
    end;
    if b_ptr <= last_b_ptr then
    begin
     if b_buffer[b_ptr].pc = pc then
     begin
      branch_adr := b_buffer[b_ptr].branch_adr;
      b_ptr := b_ptr + 1;
     end;
    end;
    output_instruction_field;
   end
   else
   begin
    write(infile,ch);
    while not eoln(tmpfile) do
    begin
     read(tmpfile,ch);
     write(infile,ch);
    end;
    readln(tmpfile);
    writeln(infile);
   end;
  end;
  close(infile);
end; { modify_code }

begin
 initialize;
 load_buffers;
 modify_code;
end.
```

Loader

```
program load;
uses {$u e:\dp\dp_comp\hex_conv} hex_conv,
   {$u e:\dp\dp_comp\ieee_cnv} ieee_cnv,
   {$u e:\dp\utility} utility,
   crt;
const
 {SEQ opcode}
 CJ_ALU_T = 0; { conditional jump based on ALU condition with default of true }
 CJ_ALU_F = 1; { conditional jump based on ALU condition with default of false }
 CJ = 2;      { conditional jump based on on others }
 RTN = 3;     { return }
 JS = 4;      { jump subroutine }
 BF = 5;      { begin for }
 EF = 6;      { end for }

 {branch status}
 no_branch = 0; { force condition to be always false }
 if_counter_not_zero = 1;
 if_not_xrfi = 2; { if network is ready to receive data }
 if_not_xdav = 3; { if network has data available }
 if_not_hrfi = 4; { if host port is ready to receive data }
 if_not_hdav = 5; { if host port has data available }
 if_zero = 6; { if ALU result is zero }
 if_not_zero = 7; { if ALU result is not zero }
 if_carry = 8; { if ALU results in a carry }
 if_not_carry = 9; { if ALU does not result in a carry }
 if_negative = 10; { if ALU result is negative }
 if_not_negative = 11; { if ALU result is not negative }
 if_ALU_error = 12; { if error occurs }
 if_net_error = 13; { if network error }
 operand_not_available = 14; { operand is still in the pipeline }
 unconditional = 15; { unconditional branch }

 {ALU_opcode}
 Fix_add = 0;
 Fix_sub = 1;
 Fix_mult = 2;
 Fix_rsub = 3;
 Bit_shl  = 4;
```

```
Bit_shr   = 6;
Bit_and   = 8;
Bit_or    = 9;
Bit_xor   = $a;
Bit_not   = $b;
Pack_exp  = $d;
Inv_Seed  = $e;
Float_round = $f;
Float_add = $10;
Float_sub = $11;
Float_mult = $12;
Float_rsub = $13;
fix_float = $15;
float_trunc = $16;
unpack_exp = $18;
unpack_mant = $19;
sqrt_exp   = $1A;
sqrt_mant   = $1B;
sine_sign   = $1C;
odd_neg     = $1d;
swap_sign   = $1e;
tan_sign    = $1f;
ALU_nop     = $14;
pass_R      = $14;
float_div   = $100; { emulation in software }

{ IO_opcode }
net_send = 1;
net_receive  = 2;
host_send = 3;
host_receive = 4;
IO_nop = 0;

{ R_bus_en }
en_dataram = 0;
en_network = 1;

{ idx_opcode }
idx_nop = 0;
load_idx = 1;
```

```
 decr_idx = 2;
 incr_idx = 3;

var pc,seq_op,branch_stat,branch_adr,alu_op,io_op,
   rbus_en,idx_op,f_wr,f_adr_mode,r_adr_mode,s_adr_mode,
   f_idx_adr,r_idx_adr,s_idx_adr,f_off,r_off,s_off:longint;
   board_id : word;
   infile : text;
   filename,infilename : string[100];
   seq_inst,dr_inst1,dr_inst0 : longint;
   ch : char;
   i : integer;

Procedure reset_instruction_field;
var i : integer;
begin
  SEQ_op := CJ;
  branch_adr := 0;
  branch_stat := no_branch;
  ALU_op := ALU_nop;
  IO_op := IO_nop;
  Rbus_en := en_dataram;
  idx_op := idx_nop;
  f_wr := 0;
  f_adr_mode := 0;
  r_adr_mode := 0;
  s_adr_mode := 0;
  f_idx_adr := 0;
  r_idx_adr := 0;
  s_idx_adr := 0;
  f_off := 0;
  r_off := 1;
  s_off := 1;
 end;

 procedure query_filename;
 begin
  write('input file : ');
  if paramcount > 0 then
  begin
    filename := paramStr(1);
```

```
   writeln(filename);
  end
  else
   readln(filename);
end;

procedure query_board_id;
var i : integer;
begin
  board_id := 0;
  if paramcount > 1 then
   val(paramStr(2),board_id,i)
  else
  begin
    write('hex switches on processor board [0] ? ');readln(board_id);
  end;
  set_board_id(board_id);
end;

procedure initialize;
begin
  query_filename;
  query_board_id;
  reset_instruction_field;
end;

procedure load_code;
begin
  seq_inst := (seq_op shl 23) +
          (branch_stat shl 19) +
          (branch_adr shl 10) +
          (alu_op shl 5) +
          (io_op shl 2) +
          (rbus_en shl 0);
   write_seq_inst(pc,seq_inst);
   dr_inst1 := (idx_op shl 5) +
          (f_wr shl 4) +
          (f_adr_mode shl 3) +
          (r_adr_mode shl 2) +
          (s_adr_mode shl 1) +
          (f_idx_adr shr 2);
```

```
 dr_inst0 := (f_idx_adr shl 30) +
          (r_idx_adr shl 27) +
          (s_idx_adr shl 24) +
          (f_off shl 16) +
          (r_off shl 8) +
          (s_off shl 0);
  write_dr_inst(pc,dr_inst1,dr_inst0);
end;

 procedure verify_code;
 begin
  seq_inst := (seq_op shl 23) +
           (branch_stat shl 19) +
           (branch_adr shl 10) +
           (alu_op shl 5) +
           (io_op shl 2) +
           (rbus_en shl 0);
   verify_seq_inst(pc,seq_inst);
   dr_inst1 := (idx_op shl 5) +
           (f_wr shl 4) +
           (f_adr_mode shl 3) +
           (r_adr_mode shl 2) +
           (s_adr_mode shl 1) +
           (f_idx_adr shr 2);
    dr_inst0 := (f_idx_adr shl 30) +
           (r_idx_adr shl 27) +
           (s_idx_adr shl 24) +
           (f_off shl 16) +
           (r_off shl 8) +
           (s_off shl 0);
    verify_dr_inst(pc,dr_inst1,dr_inst0);
  end;

  procedure load_program;
  begin
    while not eof(infile) do
    begin
      read(infile,ch);
      case ch of
        ';': begin
```

```
          readln(infile);
        end;
    'c': begin
          read(infile,pc);
          read(infile,seq_op,branch_stat,branch_adr,alu_op);
          read(infile,io_op,rbus_en,idx_op,f_wr);
          read(infile,f_adr_mode,r_adr_mode,s_adr_mode);
          read(infile,f_idx_adr,r_idx_adr,s_idx_adr);
          readln(infile,f_off,r_off,s_off);
          load_code;
        end;
    's': begin
          read(infile,pc);
          read(infile,seq_op,branch_stat,branch_adr,f_adr_mode,f_idx_adr);
        end;
  end; { of case ch }
 end;
end;

procedure verify_program;
begin
 reset(infile);
 while not eof(infile) do
 begin
  read(infile,ch);
  case ch of
   ';': begin
         readln(infile);
        end;
    'c': begin
          read(infile,pc);
          read(infile,seq_op,branch_stat,branch_adr,alu_op);
          read(infile,io_op,rbus_en,idx_op,f_wr);
          read(infile,f_adr_mode,r_adr_mode,s_adr_mode);
          read(infile,f_idx_adr,r_idx_adr,s_idx_adr);
          readln(infile,f_off,r_off,s_off);
          verify_code;
        end;
  end; { of case ch }
```

```
 end;
end;

procedure load_application;
var i : word;
begin
 writeln('loading application');
 stop_processor;
 stop_system;
 infilename := filename + '.dp';
 assign(infile,infilename);
 {$I-} Reset(infile) {$I+};
 if (IOresult <> 0 ) then
 begin
   writeln(infilename,' does not exist. ');
   halt;
 end;
 load_program;
 verify_program;
 writeln('program is downloaded (',pc,' lines)');
end;

procedure load_constant;
var ch : char;
   adr,data,rcv: longint;
begin
 writeln('loading constant');
 assign(infile,'\dp\cprog.dp');
 {$I-} Reset(infile) {$I+};
 if (IOresult <> 0 ) then
 begin
   writeln('\dp\cprog.dp constant program does not exist. ');
   halt;
 end;
 stop_processor;
 stop_system;
 load_program;
 verify_program;
 assign(infile,filename+'.hct');
 {$I-} Reset(infile) {$I+};
```

```
 if (IOresult <> 0 ) then
 begin
   writeln(filename,'.hct does not exist. ');
   halt;
 end;
 start_system;
 start_processor;
 while not eof(infile) do
 begin
  read(infile,ch);
  if (ch='v') then
  begin
   readln(infile,adr,data);
   send_longint(adr);
   send_longint(data);
   rcv := receive_longint;
   if data <> rcv then
   begin
    writeln('error in loading constant, sent     : ',data,' ',longint_to_hex(data));
    writeln('                          received : ',rcv,' ',longint_to_hex(rcv));
   end;
  end;
 end;
end;

begin
 initialize;
 load_constant;
 load_application;
end.
```

Mandelbrot Source Code

```
program mandalbrot;
var
 zr,zi,pixel_value,real_value,imaginary_value,size : real;
 cr,ci,delta_cr,delta_ci : real;
 x_out : integer;
 counter : real;
 cr_max,cr_min,ci_max,ci_min : real;
 selection : real;
 i,r : integer;
begin
 receive(host,selection);
 if selection = 0 then
 begin
  cr_min := -0.8485;
  cr_max := -0.8215;
  ci_min := 0.2165;
  ci_max := 0.2435;
 end
 else
 begin
  cr_min := -2.0 ;
  cr_max := 1.0  ;
  ci_min := -1.5 ;
  ci_max := 1.5  ;
 end;
 cr := cr_min;
 ci := ci_min;
 delta_cr := (cr_max-cr_min)*1.564945227e-3;
 delta_ci := (ci_max-ci_min)*2.865329513e-3;
 if delta_cr < 0 then delta_cr := -delta_cr;
 if delta_ci < 0 then delta_ci := -delta_ci;
 for i := 0 to 349 do
 begin
  for r := 0 to 639 do
  begin
   zr := 0;
   zi := 0;
```

```
    pixel_value := 0;
    real_value := 0;
    imaginary_value := 0;
    size := 0;
    while (pixel_value < 255) and (size < 4) do
    begin
      zi := 2*zr*zi + ci;
      zr := real_value – imaginary_value + cr;
      real_value := zr*zr;
      pixel_value := pixel_value + 1;
      imaginary_value := zi*zi;
      size := real_value + imaginary_value;
    end;
    x_out := round(pixel_value);
    send(host,x_out);
    cr := cr + delta_cr;
  end;
  cr := cr_min;
  ci := ci + delta_ci;
 end;
end.
```

Mandelbrot Host Program

```
program mandalbrot;
uses {$u e:/compile/hex_conv} hex_conv,
    {$u e:/compile/ieee_cnv} ieee_cnv,
    crt,
    {$u e:/dp/utility} utility;
var
 zr,zi,pixel_value,real_value,imaginary_value,size : real;
 cr,ci,delta_cr,delta_ci : real;
 x_out : integer;
 counter : real;
 cr_max,cr_min,ci_max,ci_min : real;
 selection : real;
 i,r : integer;
begin
 stop_processor;
 stop_system;
 start_system;
 start_processor;
 write('selection ? '); readln(selection);
 send_single(selection);
 if selection = 0 then
 begin
  cr_min := -0.8485;
  cr_max := -0.8215;
  ci_min := 0.2165;
  ci_max := 0.2435;
 end
 else
 begin
  cr_min := -2.0  ;
  cr_max := 1.0   ;
  ci_min := -1.5  ;
  ci_max := 1.5   ;
 end;
 cr := cr_min;
 ci := ci_min;
 delta_cr := (cr_max-cr_min)*1.564945227e-3;
```

```
 delta_ci := (ci_max–ci_min)*2.865329513e–3;
 if delta_cr < 0 then delta_cr := –delta_cr;
 if delta_ci < 0 then delta_ci := –delta_ci;
 for i := 0 to 349 do
 begin
  for r := 0 to 639 do
  begin
   zr := 0;
   zi := 0;
   pixel_value := 0;
   real_value := 0;
   imaginary_value := 0;
   size := 0;
   while (pixel_value < 255) and (size < 4) do
   begin
    zi := 2*zr*zi + ci;
    zr := real_value – imaginary_value + cr;
    real_value := zr*zr;
    pixel_value := pixel_value + 1;
    imaginary_value := zi*zi;
    size := real_value + imaginary_value;
   end;
   x_out := round(pixel_value);
   writeln(x_out,' rcv –> ',receive_longint);
   cr := cr + delta_cr;
  end;
  cr := cr_min;
  ci := ci + delta_ci;
 end;
end.
```

Mandelbrot Object Code

```
;
;          S  b    b   A  I  R  i  f   f  r  s  f  r  s     f     r     s
;          E  r    r   L  O  b  d  |   |  |  |  |  |  |     |     |     |
;          Q  a    a   U  |  u  x  w   a  a  a  i  i  i     o     o     o
;          |  n    n   |  o  s  |  r   d  d  d  d  d  d     f     f     f
;       p  o  c    c   o  p  |  o      r  r  r  x  x  x     f     f     f
;       c  p  h    h   p  c  e  p      |  |  |  |  |  |     s     s     s
;          c  |    |   c  o  n  c      m  m  m  a  a  a     e     e     e
;          o  s    a   o  d     o      o  o  o  d  d  d     t     t     t
;          d  t    d   d  e     d      d  d  d  r  r  r
;          e  a    r   e        e      e  e  e
;
;
;  0.00000000000000E+0000 ---> 1 (real constant)
; 0 ---> 1 (integer constant)
; 1 ---> 2 (integer constant)
; false ---> 1 (boolean constant)
; true ---> 2 (boolean constant)
;0: Nop
c       0    2 15      1  20  0  0  0  0  0  0  0  0  0  0      0      1      1
; size ---> 3 (real)
; imaginary_value ---> 4 (real)
; real_value ---> 5 (real)
; pixel_value ---> 6 (real)
; zi ---> 7 (real)
; zr ---> 8 (real)
; delta_ci ---> 9 (real)
; delta_cr ---> 10 (real,
; ci ---> 11 (real)
; cr ---> 12 (real)
; x_out ---> 13 (integer)
; counter ---> 14 (real)
; ci_min ---> 15 (real)
; ci_max ---> 16 (real)
; cr_min ---> 17 (real)
; cr_max ---> 18 (real)
; selection ---> 19 (real)
; r ---> 20 (integer)
; i ---> 21 (integer)
; main program
;1: receive(host,selection)
c       1    2  5      1  20  4  1  0  1  0  0  0  0  0  0     19      1      1
;2: #246 := selection -  0.00000000000000E+0000
;2: branch if not zero to 0
c       2    1  7      7  17  0  0  0  1  0  0  0  0  0  0    246     19      1
;  8.48500000000058E-0001 ---> 22 (real constant)
;3: cr_min :=  8.48500000000058E-0001 r- 0
```

```
c      3   2  0      0  19  0  0  0  1  0  0  0  0  0  0     17     22      1
;  8.21500000000015E-0001 ---> 23 (real constant)
;4: cr_max :=  8.21500000000015E-0001 r- 0
c      4   2  0      0  19  0  0  0  1  0  0  0  0  0  0     18     23      1
;  2.16499999999996E-0001 ---> 24 (real constant)
;5: ci_min :=  2.16499999999996E-0001 + 0
c      5   2  0      0  16  0  0  0  1  0  0  0  0  0  0     15     24      1
;  2.43500000000040E-0001 ---> 25 (real constant)
;6: ci_max :=  2.43500000000040E-0001 + 0
c      6   2 15     11  16  0  0  0  1  0  0  0  0  0  0     16     25      1
;6: branch out of else block
;  2.00000000000000E+0000 ---> 26 (real constant)
;7: cr_min :=  2.00000000000000E+0000 r- 0
c      7   2  0      0  19  0  0  0  1  0  0  0  0  0  0     17     26      1
;  1.00000000000000E+0000 ---> 27 (real constant)
;8: cr_max :=  1.00000000000000E+0000 + 0
c      8   2  0      0  16  0  0  0  1  0  0  0  0  0  0     18     27      1
;  1.50000000000000E+0000 ---> 28 (real constant)
;9: ci_min :=  1.50000000000000E+0000 r- 0
c      9   2  0      0  19  0  0  0  1  0  0  0  0  0  0     15     28      1
;10: ci_max :=  1.50000000000000E+0000 + 0
c     10   2  0      0  16  0  0  0  1  0  0  0  0  0  0     16     28      1
;11: cr := cr_min + 0
c     11   2  0      0  16  0  0  0  1  0  0  0  0  0  0     12     17      1
;12: ci := ci_min + 0
c     12   2  0      0  16  0  0  0  1  0  0  0  0  0  0     11     15      1
;  1.56494522700079E-0003 ---> 29 (real constant)
;13: #246 := cr_max - cr_min
c     13   2  0      0  17  0  0  0  1  0  0  0  0  0  0    246     18     17
;14: delta_cr := #246 *  1.56494522700079E-0003
c     14   2  0      0  18  0  0  0  1  0  0  0  0  0  0     10    246     29
;  2.86532951299989E-0003 ---> 30 (real constant)
;15: #246 := ci_max - ci_min
c     15   2  0      0  17  0  0  0  1  0  0  0  0  0  0    246     16     15
;16: delta_ci := #246 *  2.86532951299989E-0003
c     16   2  0      0  18  0  0  0  1  0  0  0  0  0  0      9    246     30
;17: #246 := delta_cr -  0.00000000000000E+0000
;17: branch if not negative to 0
c     17   1 11     19  17  0  0  0  1  0  0  0  0  0  0    246     10      1
;18: delta_cr := delta_cr r- 0
c     18   2  0      0  19  0  0  0  1  0  0  0  0  0  0     10     10      1
;19: #246 := delta_ci -  0.00000000000000E+0000
;19: branch if not negative to 0
c     19   1 11     21  17  0  0  0  1  0  0  0  0  0  0    246      9      1
;20: delta_ci := delta_ci r- 0
c     20   2  0      0  19  0  0  0  1  0  0  0  0  0  0      9      9      1
;21: i := 0 + 0
c     21   2 15     49   0  0  0  0  1  0  0  0  0  0  0     21      1      1
; 349 ---> 31 (integer constant)
```

```
;22: r := 0 + 0
c     22   2 15    45   0  0  0  0  1  0  0  0  0  0  0    20     1     1
; 639 ---> 32 (integer constant)
;23: zr := 0 + 0
c     23   2  0     0  16  0  0  0  1  0  0  0  0  0  0     8     1     1
;24: zi := 0 + 0
c     24   2  0     0  16  0  0  0  1  0  0  0  0  0  0     7     1     1
;25: pixel_value := 0 + 0
c     25   2  0     0  16  0  0  0  1  0  0  0  0  0  0     6     1     1
;26: real_value := 0 + 0
c     26   2  0     0  16  0  0  0  1  0  0  0  0  0  0     5     1     1
;27: imaginary_value := 0 + 0
c     27   2  0     0  16  0  0  0  1  0  0  0  0  0  0     4     1     1
;28: size := 0 + 0
c     28   2  0     0  16  0  0  0  1  0  0  0  0  0  0     3     1     1
; 255 ---> 33 (integer constant)
; 4 ---> 34 (integer constant)
;  2.55000000000000E+0002 ---> 35 (real constant)
;29: #247 := pixel_value -  2.55000000000000E+0002
;29: branch if not negative to 0
c     29   1 11    40  17  0  0  0  1  0  0  0  0  0  0   247     6    35
;  4.00000000000000E+0000 ---> 36 (real constant)
;30: #248 := size -  4.00000000000000E+0000
;30: branch if not negative to 0
c     30   1 11    40  17  0  0  0  1  0  0  0  0  0  0   248     3    36
; 2 ---> 37 (integer constant)
;31: #246 :=  2.00000000000000E+0000 * zr
c     31   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246    26     8
;32: #247 := #246 * zi
c     32   2  0     0  18  0  0  0  1  0  0  0  0  0  0   247   246     7
;33: zi := #247 + ci
c     33   2  0     0  16  0  0  0  1  0  0  0  0  0  0     7   247    11
;34: #246 := real_value - imaginary_value
c     34   2  0     0  17  0  0  0  1  0  0  0  0  0  0   246     5     4
;35: zr := #246 + cr
c     35   2  0     0  16  0  0  0  1  0  0  0  0  0  0     8   246    12
;36: real_value := zr * zr
c     36   2  0     0  18  0  0  0  1  0  0  0  0  0  0     5     8     8
;37: pixel_value := pixel_value +  1.00000000000000E+0000
c     37   2  0     0  16  0  0  0  1  0  0  0  0  0  0     6     6    27
;38: imaginary_value := zi * zi
c     38   2  0     0  18  0  0  0  1  0  0  0  0  0  0     4     7     7
;39: size := real_value + imaginary_value
c     39   2 15    29  16  0  0  0  1  0  0  0  0  0  0     3     5     4
;39: goto 29
;40: #38 := pixel_value round pixel_value
c     40   2  0     0  15  0  0  0  1  0  0  0  0  0  0    38     6     6
;41: x_out := #38 + 0
c     41   2  0     0   0  0  0  0  1  0  0  0  0  0  0    13    38     1
```

```
;42: send(host,x_out)
c     42   2  4    42  20  3  0  0  0  0  0  0  0  0  0     0    13    13
;43: cr := cr + delta_cr
c     43   2  0     0  16  0  0  0  1  0  0  0  0  0  0    12    12    10
;44: r := r + 1
c     44   2  0     0   0  0  0  0  1  0  0  0  0  0  0    20    20     2
;45: #246 := 639 - r
;45: branch if not negative to 23
c     45   0 11    23   1  0  0  0  1  0  0  0  0  0  0   246    32    20
; 22: goto 45
;46: cr := cr_min + 0
c     46   2  0     0  16  0  0  0  1  0  0  0  0  0  0    12    17     1
;47: ci := ci + delta_ci
c     47   2  0     0  16  0  0  0  1  0  0  0  0  0  0    11    11     9
;48: i := i + 1
c     48   2  0     0   0  0  0  0  1  0  0  0  0  0  0    21    21     2
;49: #246 := 349 - i
;49: branch if not negative to 22
c     49   0 11    22   1  0  0  0  1  0  0  0  0  0  0   246    31    21
; 21: goto 49
;50 :  goto 50
c     50   2 15    50  20  0  0  0  0  0  0  0  0  0  0     0     1     1
```

Satellite Attitude Control Source Code

```
program satelite;
{$i \fpp\io.lib}
const
 delta = 0.008;
 minus_delta = –0.008;
 a11 = 500.0; { FM/Tc }
 a12 = –20.0; { –1/Tc }
 a21 = 0.01; { 1/I }
 c1 = 7.1614583333e–03; { h(55/24); h= 0.003125 }
 c2 = –7.682291667e–03; { –h(59/24) }
 c3 = 4.817708333e–03; { h(37/24) }
 c4 = –0.001171875; { –h(9/24) }
 a31 = 4441.322018; { 0.5*w*w }
 a32 = –8882.644035; {w*w}
 a41 = –131.946892; { –2*zeta*w}
 a51 = 8882.644035; {w*w}
 a61 = –131.946892; {–2*zeta*w}
 h = 0.003125;
var
 f1n_0,f1n_1,f1n_2.f1n_3 : real;
 f2n_0,f2n_1,f2n_2,f2n_3 : real;
 f3n_0,f3n_1 f3n_2,f3n_3 : real;
 f4n_0,f4n_1,f4n_2,f4n_3 : real;
 f5n_0,f5n_1,f5n_2,f5n_3 : real;
 f6n_0,f6n_1,f6n_2,f6n_3 : real;
 f7n_0,f7n_1,f7n_2,f7n_3 : real;
 theta_e,theta_c : real;
 theta_n, theta_next : real;
 theta_s_n,theta_s_next : real;
 theta_s_dot_n,theta_s_dot_next : real;
 u_prime,u,u1,u2,u3,u4,u5,u6,u7 : real;
 Fc_n, Fc_next : real;
 Fc_prime, Fd : real;
 Theta_dot_n, theta_dot_next : real;
 x3_n,x5_n,x3_next,x5_next : real;
 time : real;
begin
```

```
  initialize;
 { initialization }
  u1 := 0;
  u2 := 0;
  u3 := 0;
  u4 := 0;
  u5 := 0;
  u6 := 0;
  u7 := 0;
  f1n_3 := 0;
  f1n_2 := 0;
  f1n_1 := 0;
  f2n_3 := 0;
  f2n_2 := 0;
  f2n_1 := 0;
  f3n_3 := 0;
  f3n_2 := 0;
  f3n_1 := 0;
  f4n_3 := 0;
  f4n_2 := 0;
  f4n_1 := 0;
  f5n_3 := 0;
  f5n_2 := 0;
  f5n_1 := 0;
  f6n_3 := 0;
  f6n_2 := 0;
  f6n_1 := 0;
  f7n_3 := 0;
  f7n_2 := 0;
  f7n_1 := 0;
  Fc_n :=  0;
  Theta_dot_n := 0;
  x3_n := 0;
  x5_n := 0;
  theta_s_dot_n := 0;
  theta_s_n := 0;
  theta_n := 0;
  Fd := 12.1875;
  Theta_c := 0.09;
```

```
time := 0.0;
While true do
begin
 { sample command pitch angle theta_c }
 { sample Disturbance Force Fd }

 { G1 – 8 cycles }
 theta_e := theta_c – theta_s_n – theta_s_dot_n;
 u_prime := 1;
 if theta_e < delta then u_prime := 0;
 if theta_e <= minus_delta then u_prime := –1;

 { G2 – 9 cycles }
 u  := u7;
 u7 := u6;
 u6 := u5;
 u5 := u4;
 u4 := u3;
 u3 := u2;
 u2 := u1;
 u1 := u_prime;

 { G3 – 11 cycles }
 f1n_0 := a11*u + a12*Fc_n;
 Fc_next := Fc_n + c1 *f1n_0 + c2*f1n_1 + c3*f1n_2 + c4*f1n_3;

 { G4 – 10 cycles }
 Fc_prime := Fc_n + Fd;
 f2n_0 := a21*Fc_prime;
 Theta_dot_next := theta_dot_n + c1*f2n_0+ c2*f2n_1 + c3*f2n_2 + c4*f2n_3;

 { G5 – 21 cycles }
 f3n_0:= a31*theta_dot_n + a32*theta_s_dot_n;
 f4n_0:= x3_n + a41*theta_s_dot_n;
 x3_next := x3_n + c1*f3n_0+ c2*f3n_1 + c3*f3n_2 + c4*f3n_3;
 theta_s_dot_next := theta_s_dot_n + c1*f4n_0+ c2*f4n_1 + c3*f4n_2 + c4*f4n_3;

 { G6 – 21 cycles }
 f5n_0:= a51*(theta_n – theta_s_n) ;
 f6n_0:= x5_n + a61*theta_s_n;
 x5_next := x5_n + c1*f5n_0+ c2*f5n_1 + c3*f5n_2 + c4*f5n_3;
 theta_s_next := theta_s_n + c1*f6n_0+ c2*f6n_1 + c3*f6n_2 + c4*f6n_3;

 { G7 –  9 cycles }
 f7n_0:= theta_dot_n;
 theta_next := theta_n + c1*f7n_0+ c2*f7n_1 + c3*f7n_2 + c4*f7n_3;

 send(host,theta_n);
 send(host,theta_dot_n);

 { move the time step – 29 cycles }
 f1n_3 := f1n_2;
 f1n_2 := f1n_1;
```

```
    f1n_1 := f1n_0;
    f2n_3 := f2n_2;
    f2n_2 := f2n_1;
    f2n_1 := f2n_0;
    f3n_3 := f3n_2;
    f3n_2 := f3n_1;
    f3n_1 := f3n_0;
    f4n_3 := f4n_2;
    f4n_2 := f4n_1;
    f4n_1 := f4n_0;
    f5n_3 := f5n_2;
    f5n_2 := f5n_1;
    f5n_1 := f5n_0;
    f6n_3 := f6n_2;
    f6n_2 := f6n_1;
    f6n_1 := f6n_0;
    f7n_3 := f7n_2;
    f7n_2 := f7n_1;
    f7n_1 := f7n_0;
    Fc_n := Fc_next;
    Theta_dot_n := theta_dot_next;
    x3_n := x3_next;
    x5_n := x5_next;
    theta_s_dot_n := theta_s_dot_next;
    theta_s_n := theta_s_next;
    theta_n := theta_next;
    time := time + 0.01;
  end;
end.
```

Satellite Attitude Control Host Program

```
program test3;
uses {$u e:/compile/hex_conv} hex_conv,
   {$u e:/compile/ieee_cnv} ieee_cnv,
   crt,
   {$u e:/dp/utility} utility;
const
 delta = 0.008;
 minus_delta = -0.008;
 a11 = 500.0; { FM/Tc }
 a12 = -20.0; { -1/Tc }
 a21 = 0.01;  { 1/I }
 c1  =  7.1614583333e-03;    { h(55/24); h= 0.003125 }
 c2  = -7.682291667e-03;     { -h(59/24) }
 c3  =  4.817708333e-03;     { h(37/24) }
 c4  = -0.001171875;         { -h(9/24) }
 a31 = 4441.322018; { 0.5*w*w }
 a32 = -8882.644035; {w*w}
 a41 = -131.946892; { -2*zeta*w}
 a51 = 8882.644035; {w*w}
 a61 = -131.946892; {-2*zeta*w}
 h = 0.003125;
var
 f1n_0,f1n_1,f1n_2,f1n_3 : single;
 f2n_0,f2n_1,f2n_2,f2n_3 : single;
 f3n_0,f3n_1,f3n_2,f3n_3 : single;
 f4n_0,f4n_1,f4n_2,f4n_3 : single;
 f5n_0,f5n_1,f5n_2,f5n_3 : single;
 f6n_0,f6n_1,f6n_2,f6n_3 : single;
 f7n_0,f7n_1,f7n_2,f7n_3 : single;
 theta_e,theta_c : single;
 theta_n, theta_next : single;
 theta_s_n,theta_s_next : single;
 theta_s_dot_n,theta_s_dot_next : single;
 u_prime,u,u1,u2,u3,u4,u5,u6,u7 : single;
 Fc_n, Fc_next : single;
 Fc_prime, Fd : single;
 Theta_dot_n, theta_dot_next : single;
```

```
  x3_n,x5_n,x3_next,x5_next : single;
  time : single;
begin
    stop_processor;
    stop_system;
    start_system;
    start_processor;
  { initialization }
    u1 := 0;
    u2 := 0;
    u3 := 0;
    u4 := 0;
    u5 := 0;
    u6 := 0;
    u7 := 0;
    f1n_3 := 0;
    f1n_2 := 0;
    f1n_1 := 0;
    f2n_3 := 0;
    f2n_2 := 0;
    f2n_1 := 0;
    f3n_3 := 0;
    f3n_2 := 0;
    f3n_1 := 0;
    f4n_3 := 0;
    f4n_2 := 0;
    f4n_1 := 0;
    f5n_3 := 0;
    f5n_2 := 0;
    f5n_1 := 0;
    f6n_3 := 0;
    f6n_2 := 0;
    f6n_1 := 0;
    f7n_3 := 0;
    f7n_2 := 0;
    f7n_1 := 0;
    Fc_n :=  0;
    Theta_dot_n := 0.
    x3_n := 0;
```

```
x5_n := 0;
theta_s_dot_n := 0;
theta_s_n := 0;
theta_n := 0;
Fd := 12.1875;
Theta_c := 0.09;
time := 0.0;
While true do
begin
 { sample command pitch angle theta_c }
 { sample Disturbance Force Fd }

 { G1 – 8 cycles }
 theta_e := theta_c – theta_s_n – theta_s_dot_n;
 u_prime := 1;
 if theta_e < delta then u_prime := 0;
 if theta_e <= minus_delta then u_prime := –1;

 { G2 – 9 cycles }
 u  := u7;
 u7 := u6;
 u6 := u5;
 u5 := u4;
 u4 := u3;
 u3 := u2;
 u2 := u1;
 u1 := u_prime;

 { G3 – 11 cycles }
 f1n_0 := a11*u + a12*Fc_n;
 Fc_next := Fc_n + c1 *f1n_0 + c2*f1n_1 + c3*f1n_2 + c4*f1n_3;

 { G4 – 10 cycles }
 Fc_prime := Fc_n + Fd;
 f2n_0 := a21*Fc_prime;
 Theta_dot_next := theta_dot_n + c1*f2n_0+ c2*f2n_1 + c3*f2n_2 + c4*f2n_3;

 { G5 – 21 cycles }
 f3n_0:= a31*theta_dot_n + a32*theta_s_dot_n;
 f4n_0:= x3_n + a41*theta_s_dot_n;
 x3_next := x3_n + c1*f3n_0+ c2*f3n_1 + c3*f3n_2 + c4*f3n_3;
 theta_s_dot_next := theta_s_dot_n + c1*f4n_0+ c2*f4n_1 + c3*f4n_2 + c4*f4n_3;
```

```
{ G6 – 21 cycles }
f5n_0:= a51*(theta_n – theta_s_n) ;
f6n_0:= x5_n + a61*theta_s_n;
x5_next := x5_n + c1*f5n_0+ c2*f5n_1 + c3*f5n_2 + c4*f5n_3;
theta_s_next := theta_s_n + c1*f6n_0+ c2*f6n_1 + c3*f6n_2 + c4*f6n_3;

{ G7 – 9 cycles }
f7n_0:= theta_dot_n;
theta_next := theta_n + c1*f7n_0+ c2*f7n_1 + c3*f7n_2 + c4*f7n_3;

writeln(theta_n,' ',receive_single);
writeln(theta_dot_n,' ',receive_single);

{ move the time step – 29 cycles }
f1n_3 := f1n_2;
f1n_2 := f1n_1;
f1n_1 := f1n_0;
f2n_3 := f2n_2;
f2n_2 := f2n_1;
f2n_1 := f2n_0;
f3n_3 := f3n_2;
f3n_2 := f3n_1;
f3n_1 := f3n_0;
f4n_3 := f4n_2;
f4n_2 := f4n_1;
f4n_1 := f4n_0;
f5n_3 := f5n_2;
f5n_2 := f5n_1;
f5n_1 := f5n_0;
f6n_3 := f6n_2;
f6n_2 := f6n_1;
f6n_1 := f6n_0;
f7n_3 := f7n_2;
f7n_2 := f7n_1;
f7n_1 := f7n_0;
Fc_n := Fc_next;
Theta_dot_n := theta_dot_next;
x3_n := x3_next;
x5_n := x5_next;
theta_s_dot_n := theta_s_dot_next;
theta_s_n := theta_s_next;
```

```
    theta_n := theta_next;
    time := time + 0.01;
  end;
end.end.
```

Satellite Attitude Control Object Code

```
;
;          S  b    b   A  I  R  i  f   f  r  s  f  r  s     f     r     s
;          E  r    r   L  O  b  d  |   |  |  |  |  |  |     |     |     |
;          Q  a    a   U  |  u  x  w   a  a  a  i  i  i     o     o     o
;          |  n    n   |  o  s  |  r   d  d  d  d  d  d     f     f     f
;      p   o  c    c   o  p  |  o      r  r  r  x  x  x     f     f     f
;      c   p  h    h   p  c  e  p      |  |  |  |  |  |     s     s     s
;          c  |    |   c  o  n  c      m  m  m  a  a  a     e     e     e
;          o  s    a   o  d     o      o  o  o  d  d  d     t     t     t
;          d  t    d   d  e     d      d  d  d  r  r  r
;          e  a    r   e        e      e  e  e
;
;
;  0.00000000000000E+0000 ---> 1 (real constant)
; 0 ---> 1 (integer constant)
; 1 ---> 2 (integer constant)
; false ---> 1 (boolean constant)
; true ---> 2 (boolean constant)
;0: Nop
c       0    2 15      1  20  0  0  0  0  0  0  0  0  0  0     0     1     1
; delta ---> 3 (real constant)
; minus_delta ---> 4 (real constant)
; a11 ---> 5 (real constant)
; a12 ---> 6 (real constant)
; a21 ---> 7 (real constant)
; c1 ---> 8 (real constant)
; c2 ---> 9 (real constant)
; c3 ---> 10 (real constant)
; c4 ---> 11 (real constant)
; a31 ---> 12 (real constant)
; a32 ---> 13 (real constant)
; a41 ---> 14 (real constant)
; a51 ---> 15 (real constant)
; a61 ---> 16 (real constant)
; h ---> 17 (real constant)
; f1n_3 ---> 18 (real)
; f1n_2 ---> 19 (real)
; f1n_1 ---> 20 (real)
; f1n_0 ---> 21 (real)
; f2n_3 ---> 22 (real)
; f2n_2 ---> 23 (real)
; f2n_1 ---> 24 (real)
; f2n_0 ---> 25 (real)
; f3n_3 ---> 26 (real)
; f3n_2 ---> 27 (real)
; f3n_1 ---> 28 (real)
; f3n_0 ---> 29 (real)
```

```
; f4n_3 ---> 30 (real)
; f4n_2 ---> 31 (real)
; f4n_1 ---> 32 (real)
; f4n_0 ---> 33 (real)
; f5n_3 ---> 34 (real)
; f5n_2 ---> 35 (real)
; f5n_1 ---> 36 (real)
; f5n_0 ---> 37 (real)
; f6n_3 ---> 38 (real)
; f6n_2 ---> 39 (real)
; f6n_1 ---> 40 (real)
; f6n_0 ---> 41 (real)
; f7n_3 ---> 42 (real)
; f7n_2 ---> 43 (real)
; f7n_1 ---> 44 (real)
; f7n_0 ---> 45 (real)
; theta_c ---> 46 (real)
; theta_e ---> 47 (real)
; theta_next ---> 48 (real)
; theta_n ---> 49 (real)
; theta_s_next ---> 50 (real)
; theta_s_n ---> 51 (real)
; theta_s_dot_next ---> 52 (real)
; theta_s_dot_n ---> 53 (real)
; u7 ---> 54 (real)
; u6 ---> 55 (real)
; u5 ---> 56 (real)
; u4 ---> 57 (real)
; u3 ---> 58 (real)
; u2 ---> 59 (real)
; u1 ---> 60 (real)
; u ---> 61 (real)
; u_prime ---> 62 (real)
; fc_next ---> 63 (real)
; fc_n ---> 64 (real)
; fd ---> 65 (real)
; fc_prime ---> 66 (real)
; theta_dot_next ---> 67 (real)
; theta_dot_n ---> 68 (real)
; x5_next ---> 69 (real)
; x3_next ---> 70 (real)
; x5_n ---> 71 (real)
; x3_n ---> 72 (real)
; time ---> 73 (real)
; main program
;1: u1 := 0 + 0
c       1   2  0      0  16  0  0  0  1  0  0  0  0  0  0     60      1      1
;2: u2 := 0 + 0
c       2   2  0      0  16  0  0  0  1  0  0  0  0  0  0     59      1      1
```

```
;3: u3 := 0 + 0
c       3    2   0       0   16   0   0   0   1   0   0   0   0   0   0      58       1       1
;4: u4 := 0 + 0
c       4    2   0       0   16   0   0   0   1   0   0   0   0   0   0      57       1       1
;5: u5 := 0 + 0
c       5    2   0       0   16   0   0   0   1   0   0   0   0   0   0      56       1       1
;6: u6 := 0 + 0
c       6    2   0       0   16   0   0   0   1   0   0   0   0   0   0      55       1       1
;7: u7 := 0 + 0
c       7    2   0       0   16   0   0   0   1   0   0   0   0   0   0      54       1       1
;8: f1n_3 := 0 + 0
c       8    2   0       0   16   0   0   0   1   0   0   0   0   0   0      18       1       1
;9: f1n_2 := 0 + 0
c       9    2   0       0   16   0   0   0   1   0   0   0   0   0   0      19       1       1
;10: f1n_1 := 0 + 0
c      10    2   0       0   16   0   0   0   1   0   0   0   0   0   0      20       1       1
;11: f2n_3 := 0 + 0
c      11    2   0       0   16   0   0   0   1   0   0   0   0   0   0      22       1       1
;12: f2n_2 := 0 + 0
c      12    2   0       0   16   0   0   0   1   0   0   0   0   0   0      23       1       1
;13: f2n_1 := 0 + 0
c      13    2   0       0   16   0   0   0   1   0   0   0   0   0   0      24       1       1
;14: f3n_3 := 0 + 0
c      14    2   0       0   16   0   0   0   1   0   0   0   0   0   0      26       1       1
;15: f3n_2 := 0 + 0
c      15    2   0       0   16   0   0   0   1   0   0   0   0   0   0      27       1       1
;16: f3n_1 := 0 + 0
c      16    2   0       0   16   0   0   0   1   0   0   0   0   0   0      28       1       1
;17: f4n_3 := 0 + 0
c      17    2   0       0   16   0   0   0   1   0   0   0   0   0   0      30       1       1
;18: f4n_2 := 0 + 0
c      18    2   0       0   16   0   0   0   1   0   0   0   0   0   0      31       1       1
;19: f4n_1 := 0 + 0
c      19    2   0       0   16   0   0   0   1   0   0   0   0   0   0      32       1       1
;20: f5n_3 := 0 + 0
c      20    2   0       0   16   0   0   0   1   0   0   0   0   0   0      34       1       1
;21: f5n_2 := 0 + 0
c      21    2   0       0   16   0   0   0   1   0   0   0   0   0   0      35       1       1
;22: f5n_1 := 0 + 0
c      22    2   0       0   16   0   0   0   1   0   0   0   0   0   0      36       1       1
;23: f6n_3 := 0 + 0
c      23    2   0       0   16   0   0   0   1   0   0   0   0   0   0      38       1       1
;24: f6n_2 := 0 + 0
c      24    2   0       0   16   0   0   0   1   0   0   0   0   0   0      39       1       1
;25: f6n_1 := 0 + 0
c      25    2   0       0   16   0   0   0   1   0   0   0   0   0   0      40       1       1
;26: f7n_3 := 0 + 0
c      26    2   0       0   16   0   0   0   1   0   0   0   0   0   0      42       1       1
;27: f7n_2 := 0 + 0
```

```
c     27   2  0     0  16  0  0  0  1  0  0  0  0  0  0    43     1     1
;28: f7n_1 := 0 + 0
c     28   2  0     0  16  0  0  0  1  0  0  0  0  0  0    44     1     1
;29: fc_n := 0 + 0
c     29   2  0     0  16  0  0  0  1  0  0  0  0  0  0    64     1     1
;30: theta_dot_n := 0 + 0
c     30   2  0     0  16  0  0  0  1  0  0  0  0  0  0    68     1     1
;31: x3_n := 0 + 0
c     31   2  0     0  16  0  0  0  1  0  0  0  0  0  0    72     1     1
;32: x5_n := 0 + 0
c     32   2  0     0  16  0  0  0  1  0  0  0  0  0  0    71     1     1
;33: theta_s_dot_n := 0 + 0
c     33   2  0     0  16  0  0  0  1  0  0  0  0  0  0    53     1     1
;34: theta_s_n := 0 + 0
c     34   2  0     0  16  0  0  0  1  0  0  0  0  0  0    51     1     1
;35: theta_n := 0 + 0
c     35   2  0     0  16  0  0  0  1  0  0  0  0  0  0    49     1     1
;  1.218?5000000000E+0001 ---> 74 (real constant)
;36: fd  =  1.21875000000000E+0001 + 0
c     36   2  0     0  16  0  0  0  1  0  0  0  0  0  0    65    74     1
;  9.000)0000000318E-0002 ---> 75 (real constant)
;37: the:a_c :=  9.00000000000318E-0002 + 0
c     3    2  0     0  16  0  0  0  1  0  0  0  0  0  0    46    75     1
;38: tim: :=  0.00000000000000E+0000 + 0
c     3?   2  0     0  16  0  0  0  1  0  0  0  0  0  0    73     1     1
;39: #24 6 := theta_c - theta_s_n
c     3:   2  0     0  17  0  0  0  1  0  0  0  0  0  0   246    46    51
;40: th:ta_e := #246 - theta_s_dot_n
c     4(   2  0     0  17  0  0  0  1  0  0  0  0  0  0    47   246    53
;41: #24 6 := 1 float 0
c     4    2  0     0  21  0  0  0  1  0  0  0  0  0  0   246     2     1
;42: u_rrime := #246 + 0
c     4.   2  0     0  16  0  0  0  1  0  0  0  0  0  0    62   246     1
;43: #2 6 := theta_e - delta
;43: br:nch if not negative to 0
c     4:   1 11    45  17  0  0  0  1  0  0  0  0  0  0   246    47     3
;44: u_prime := 0 + 0
c     44   2  0     0  16  0  0  0  1  0  0  0  0  0  0    62     1     1
;45: #246 := minus_delta - theta_e
;45: branch if negative to 0
c     45   1 10    48  17  0  0  0  1  0  0  0  0  0  0   246     4    47
;46: #246 := 1 float 0
c     46   2  0     0  21  0  0  0  1  0  0  0  0  0  0   246     2     1
;47: u_prime := #246 r- 0
c     47   2  0     0  19  0  0  0  1  0  0  0  0  0  0    62   246     1
;48: u  := u7 + 0
c     48   2  0     0  16  0  0  0  1  0  0  0  0  0  0    61    54     1
;49: u7 := u6 + 0
c     49   2  0     0  16  0  0  0  1  0  0  0  0  0  0    54    55     1
```

```
;50: u6 := u5 + 0
c     50   2  0     0  16  0  0  0  1  0  0  0  0  0  0    55    56     1
;51: u5 := u4 + 0
c     51   2  0     0  16  0  0  0  1  0  0  0  0  0  0    56    57     1
;52: u4 := u3 + 0
c     52   2  0     0  16  0  0  0  1  0  0  0  0  0  0    57    58     1
;53: u3 := u2 + 0
c     53   2  0     0  16  0  0  0  1  0  0  0  0  0  0    58    59     1
;54: u2 := u1 + 0
c     54   2  0     0  16  0  0  0  1  0  0  0  0  0  0    59    60     1
;55: u1 := u_prime + 0
c     55   2  0     0  16  0  0  0  1  0  0  0  0  0  0    60    62     1
;56: #246 := a11 * u
c     56   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246     5    61
;57: #247 := a12 * fc_n
c     57   2  0     0  18  0  0  0  1  0  0  0  0  0  0   247     6    64
;58: f1n_0 := #246 + #247
c     58   2  0     0  16  0  0  0  1  0  0  0  0  0  0    21   246   247
;59: #246 := c1 * f1n_0
c     59   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246     8    21
;60: #247 := fc_n + #246
c     60   2  0     0  16  0  0  0  1  0  0  0  0  0  0   247    64   246
;61: #248 := c2 * f1n_1
c     61   2  0     0  18  0  0  0  1  0  0  0  0  0  0   248     9    20
;62: #249 := #247 + #248
c     62   2  0     0  16  0  0  0  1  0  0  0  0  0  0   249   247   248
;63: #250 := c3 * f1n_2
c     63   2  0     0  18  0  0  0  1  0  0  0  0  0  0   250    10    19
;64: #251 := #249 + #250
c     64   2  0     0  16  0  0  0  1  0  0  0  0  0  0   251   249   250
;65: #252 := c4 * f1n_3
c     65   2  0     0  18  0  0  0  1  0  0  0  0  0  0   252    11    18
;66: fc_next := #251 + #252
c     66   2  0     0  16  0  0  0  1  0  0  0  0  0  0    63   251   252
;67: fc_prime := fc_n + fd
c     67   2  0     0  16  0  0  0  1  0  0  0  0  0  0    66    64    65
;68: f2n_0 := a21 * fc_prime
c     68   2  0     0  18  0  0  0  1  0  0  0  0  0  0    25     7    66
;69: #246 := c1 * f2n_0
c     69   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246     8    25
;70: #247 := theta_dot_n + #246
c     70   2  0     0  16  0  0  0  1  0  0  0  0  0  0   247    68   246
;71: #248 := c2 * f2n_1
c     71   2  0     0  18  0  0  0  1  0  0  0  0  0  0   248     9    24
;72: #249 := #247 + #248
c     72   2  0     0  16  0  0  0  1  0  0  0  0  0  0   249   247   248
;73: #250 := c3 * f2n_2
c     73   2  0     0  18  0  0  0  1  0  0  0  0  0  0   250    10    23
;74: #251 := #249 + #250
```

```
c     74   2  0     0  15  0  0  0  1  0  0  0  0  0  0   251   249   250
;75: #252 := c4 * f2n_3
c     75   2  0     0  18  0  0  0  1  0  0  0  0  0  0   252    11    22
;76: theta_dot_next := #251 + #252
c     76   2  0     0  16  0  0  0  1  0  0  0  0  0  0    67   251   252
;77: #246 := a31 * theta_dot_n
c     77   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246    12    68
;78: #247 := a32 * theta_s_dot_n
c     78   2  0     0  18  0  0  0  1  0  0  0  0  0  0   247    13    53
;79: f3n_0 := #246 + #247
c     79   2  0     0  16  0  0  0  1  0  0  0  0  0  0    29   246   247
;80: #246 := a41 * theta_s_dot_n
c     80   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246    14    53
;81: f4n_0 := x3_n + #246
c     81   2  0     0  16  0  0  0  1  0  0  0  0  0  0    33    72   246
;82: #246 := c1 * f3n_0
c     82   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246     8    29
;83: #247 := x3_n + #246
c     83   2  0     0  16  0  0  0  1  0  0  0  0  0  0   247    72   246
;84: #248 := c2 * f3n_1
c     84   2  0     0  18  0  0  0  1  0  0  0  0  0  0   248     9    28
;85: #249 := #247 + #248
c     85   2  0     0  16  0  0  0  1  0  0  0  0  0  0   249   247   248
;86: #250 := c3 * f3n_2
c     86   2  0     0  18  0  0  0  1  0  0  0  0  0  0   250    10    27
;87: #251 := #249 + #250
c     87   2  0     0  16  0  0  0  1  0  0  0  0  0  0   251   249   250
;88: #252 := c4 * f3n_3
c     88   2  0     0  18  0  0  0  1  0  0  0  0  0  0   252    11    26
;89: x3_next := #251 + #252
c     89   2  0     0  16  0  0  0  1  0  0  0  0  0  0    70   251   252
;90: #246 := c1 * f4n_0
c     90   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246     8    33
;91: #247 := theta_s_dot_n + #246
c     91   2  0     0  16  0  0  0  1  0  0  0  0  0  0   247    53   246
;92: #248 := c2 * f4n_1
c     92   2  0     0  18  0  0  0  1  0  0  0  0  0  0   248     9    32
;93: #249 := #247 + #248
c     93   2  0     0  16  0  0  0  1  0  0  0  0  0  0   249   247   248
;94: #250 := c3 * f4n_2
c     94   2  0     0  18  0  0  0  1  0  0  0  0  0  0   250    10    31
;95: #251 := #249 + #250
c     95   2  0     0  16  0  0  0  1  0  0  0  0  0  0   251   249   250
;96: #252 := c4 * f4n_3
c     96   2  0     0  18  0  0  0  1  0  0  0  0  0  0   252    11    30
;97: theta_s_dot_next := #251 + #252
c     97   2  0     0  16  0  0  0  1  0  0  0  0  0  0    52   251   252
;98: #246 := theta_n - theta_s_n
c     98   2  0     0  17  0  0  0  1  0  0  0  0  0  0   246    49    51
```

```
;99: f5n_0 := a51 * #246
c     99   2  0     0  18  0  0  0  1  0  0  0  0  0  0    37    15   246
;100: #246 := a61 * theta_s_n
c    100   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246    16    51
;101: f6n_0 := x5_n + #246
c    101   2  0     0  16  0  0  0  1  0  0  0  0  0  0    41    71   246
;102: #246 := c1 * f5n_0
c    102   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246     8    37
;103: #247 := x5_n + #246
c    103   2  0     0  16  0  0  0  1  0  0  0  0  0  0   247    71   246
;104: #248 := c2 * f5n_1
c    104   2  0     0  18  0  0  0  1  0  0  0  0  0  0   248     9    36
;105: #249 := #247 + #248
c    105   2  0     0  16  0  0  0  1  0  0  0  0  0  0   249   247   248
;106: #250 := c3 * f5n_2
c    106   2  0     0  18  0  0  0  1  0  0  0  0  0  0   250    10    35
;107: #251 := #249 + #250
c    107   2  0     0  16  0  0  0  1  0  0  0  0  0  0   251   249   250
;108: #252 := c4 * f5n_3
c    108   2  0     0  18  0  0  0  1  0  0  0  0  0  0   252    11    34
;109: x5_next := #251 + #252
c    109   2  0     0  16  0  0  0  1  0  0  0  0  0  0    69   251   252
;110: #246 := c1 * f6n_0
c    110   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246     8    41
;111: #247 := theta_s_n + #246
c    111   2  0     0  16  0  0  0  1  0  0  0  0  0  0   247    51   246
;112: #248 := c2 * f6n_1
c    112   2  0     0  18  0  0  0  1  0  0  0  0  0  0   248     9    40
;113: #249 := #247 + #248
c    113   2  0     0  16  0  0  0  1  0  0  0  0  0  0   249   247   248
;114: #250 := c3 * f6n_2
c    114   2  0     0  18  0  0  0  1  0  0  0  0  0  0   250    10    39
;115: #251 := #249 + #250
c    115   2  0     0  16  0  0  0  1  0  0  0  0  0  0   251   249   250
;116: #252 := c4 * f6n_3
c    116   2  0     0  18  0  0  0  1  0  0  0  0  0  0   252    11    38
;117: theta_s_next := #251 + #252
c    117   2  0     0  16  0  0  0  1  0  0  0  0  0  0    50   251   252
;118: f7n_0 := theta_dot_n + 0
c    118   2  0     0  16  0  0  0  1  0  0  0  0  0  0    45    68     1
;119: #246 := c1 * f7n_0
c    119   2  0     0  18  0  0  0  1  0  0  0  0  0  0   246     8    45
;120: #247 := theta_n + #246
c    120   2  0     0  16  0  0  0  1  0  0  0  0  0  0   247    49   246
;121: #248 := c2 * f7n_1
c    121   2  0     0  18  0  0  0  1  0  0  0  0  0  0   248     9    44
;122: #249 := #247 + #248
c    122   2  0     0  16  0  0  0  1  0  0  0  0  0  0   249   247   248
;123: #250 := c3 * f7n_2
```

```
c    123   2  0     0  16  0  0  0  1  0  0  0  0  0  0   250    10    43
;124: #251 := #249 + #250
c    124   2  0     0  16  0  0  0  1  0  0  0  0  0  0   251   249   250
;125: #252 := c4 * f7n_3
c    125   2  0     0  18  0  0  0  1  0  0  0  0  0  0   252    11    42
;126: theta_next := #251 + #252
c    126   2  0     0  16  0  0  0  1  0  0  0  0  0  0    48   251   252
;127: send(host,theta_n)
c    127   2  4   127  20  3  0  0  0  0  0  0  0  0  0     0    49    49
;128: send(host,theta_dot_n)
c    128   2  4   128  20  3  0  0  0  0  0  0  0  0  0     0    68    68
;129: f1n_3 := f1n_2 + 0
c    129   2  0     0  16  0  0  0  1  0  0  0  0  0  0    18    19     1
;130: f1n_2 := f1n_1 + 0
c    130   2  0     0  16  0  0  0  1  0  0  0  0  0  0    19    20     1
;131: f1n_1 := f1n_0 + 0
c    131   2  0     0  16  0  0  0  1  0  0  0  0  0  0    20    21     1
;132: f2n_3 := f2n_2 + 0
c    132   2  0     0  16  0  0  0  1  0  0  0  0  0  0    22    23     1
;133: f2n_2 := f2n_1 + 0
c    133   2  0     0  16  0  0  0  1  0  0  0  0  0  0    23    24     1
;134: f2n_1 := f2n_0 + 0
c    134   2  0     0  16  0  0  0  1  0  0  0  0  0  0    24    25     1
;135: f3n_3 := f3n_2 + 0
c    135   2  0     0  16  0  0  0  1  0  0  0  0  0  0    26    27     1
;136: f3n_2 := f3n_1 + 0
c    136   2  0     0  16  0  0  0  1  0  0  0  0  0  0    27    28     1
;137: f3n_1 := f3n_0 + 0
c    137   2  0     0  16  0  0  0  1  0  0  0  0  0  0    28    29     1
;138: f4n_3 := f4n_2 + 0
c    138   2  0     0  16  0  0  0  1  0  0  0  0  0  0    30    31     1
;139: f4n_2 := f4n_1 + 0
c    139   2  0     0  16  0  0  0  1  0  0  0  0  0  0    31    32     1
;140: f4n_1 := f4n_0 + 0
c    140   2  0     0  16  0  0  0  1  0  0  0  0  0  0    32    33     1
;141: f5n_3 := f5n_2 + 0
c    141   2  0     0  16  0  0  0  1  0  0  0  0  0  0    34    35     1
;142: f5n_2 := f5n_1 + 0
c    142   2  0     0  16  0  0  0  1  0  0  0  0  0  0    35    36     1
;143: f5n_1 := f5n_0 + 0
c    143   2  0     0  16  0  0  0  1  0  0  0  0  0  0    36    37     1
;144: f6n_3 := f6n_2 + 0
c    144   2  0     0  16  0  0  0  1  0  0  0  0  0  0    38    39     1
;145: f6n_2 := f6n_1 + 0
c    145   2  0     0  16  0  0  0  1  0  0  0  0  0  0    39    40     1
;146: f6n_1 := f6n_0 + 0
c    146   2  0     0  16  0  0  0  1  0  0  0  0  0  0    40    41     1
;147: f7n_3 := f7n_2 + 0
c    147   2  0     0  16  0  0  0  1  0  0  0  0  0  0    42    43     1
```

```
;148: f7n_2 := f7n_1 + 0
c    148   2  0     0  16  0  0  0  1  0  0  0  0  0  0    43    44     1
;149: f7n_1 := f7n_0 + 0
c    149   2  0     0  16  0  0  0  1  0  0  0  0  0  0    44    45     1
;150: fc_n := fc_next + 0
c    150   2  0     0  16  0  0  0  1  0  0  0  0  0  0    64    63     1
;151: theta_dot_n := theta_dot_next + 0
c    151   2  0     0  16  0  0  0  1  0  0  0  0  0  0    68    67     1
;152: x3_n := x3_next + 0
c    152   2  0     0  16  0  0  0  1  0  0  0  0  0  0    72    70     1
;153: x5_n := x5_next + 0
c    153   2  0     0  16  0  0  0  1  0  0  0  0  0  0    71    69     1
;154: theta_s_dot_n := theta_s_dot_next + 0
c    154   2  0     0  16  0  0  0  1  0  0  0  0  0  0    53    52     1
;155: theta_s_n := theta_s_next + 0
c    155   2  0     0  16  0  0  0  1  0  0  0  0  0  0    51    50     1
;156: theta_n := theta_next + 0
c    156   2  0     0  16  0  0  0  1  0  0  0  0  0  0    49    48     1
;  1.00000000000051E-0002 ---> 76 (real constant)
;157: time := time +  1.00000000000051E-0002
c    157   2 15    39  16  0  0  0  1  0  0  0  0  0  0    73    73    76
;157: goto 39
;158 :  goto 158
c    158   2 15   158  20  0  0  0  0  0  0  0  0  0  0     0     1     1
```

Utilities

```
unit utility;
Interface
uses {$u e:\dp_comp\ieee_cnv } ieee_cnv,
    {$u e:\dp_comp\hex_conv } hex_conv;
type
  longint_overlay = record
   case t:byte of
    0: (l:longint);
    1: (w:array[0..1] of word);
    2: (b:array[0..3] of byte);
    3: (s: single);
   end;
const
  segment = $d000;
  SM8_off = $0000;
  DR_off  = $0080;
  SNI_off = $0100;
  SEQ_off = $0180;
  longint_pattern_no = 67;
  byte_pattern_no = 19;
  word_pattern_no = 35;
var longint_pattern : array[0..longint_pattern_no] of longint;
    byte_pattern : array[0..byte_pattern_no] of byte;
    word_pattern : array[0..word_pattern_no] of word;
    Procedure reset_sm8;
    Procedure write_seq_inst(pc:word;data:longint);
    Procedure verify_seq_inst(pc:word;data:longint);
    procedure write_dr_inst(pc:word;data1:byte;data0:longint);
    procedure verify_dr_inst(pc:word;data1:byte;data0:longint);
    Procedure start_system;
    Procedure stop_system;
    Procedure start_processor;
    Procedure stop_processor;
    Procedure initialize;
    Procedure test_seq;
    Procedure test_dr;
    Procedure test_sm8;
```

```
  Procedure test_sni;
  Procedure verify_longint(rcv,expected:longint);
  Procedure verify_word(rcv,expected:word);
  Procedure verify_byte(rcv,expected:byte);
  Procedure set_board_id(board_id:byte);
  function receive_longint: longint;
  function receive_single : single;
  function rfi : boolean;
  procedure check_rfi;
  function dav : boolean;
  procedure check_dav;
  Procedure send_longint(data : longint);
  Procedure send_single(x : single);
  var error : boolean;

Implementation
const dav_timeout_count = 10000;
    rfi_timeout_count = 10000;
var
 verify_count : longint;
 i : longint;

function rfi : boolean;
begin
 if (mem[segment:sni_off+6] and 1) = 1 then rfi := true else rfi := false;
end; { of rfi }

function dav : boolean;
begin
 if (mem[segment:sni_off+6] and 2) = 2 then dav := true else dav := false;
end; { of dav }

Procedure check_rfi;
var count: longint;
begin
 count := 1;
 repeat
  count := count + 1;
 until ((mem[segment:sni_off+6] and 1) = 1) or (count = rfi_timeout_count);
 if count = rfi_timeout_count then
 begin
  write('processor is not ready for input');
```

```
   halt;
  end;
end; { of rfi }

Procedure check_dav;
var count : longint;
begin
  count := 1;
  repeat
    count := count + 1;
  until ((mem[segment:sni_off+6] and 2) = 2) or (count = dav_timeout_count);
  if count = dav_timeout_count then
  begin
    write('data not available ');
    halt;
  end;
end; { of check_dav }

function receive_sni_data:longint;
var rcv : longint_overlay;
    i : integer;
begin
  for i := 0 to 3 do
  begin
    rcv.b[i] := memw[segment:sni_off+(4+i)*2];
{   writeln('received : ',byte_to_hex(rcv.b[i])); }
  end;
  receive_sni_data := rcv.l;
end;

procedure send_sni_data(data : longint);
var snd : longint_overlay;
    i : integer;
begin
  snd.l := data;
  for i := 3 downto 0 do
  begin
{   writeln('sending : ',byte_to_hex(snd.b[i])); }
    memw[segment:sni_off+(4+i)*2] := snd.b[i];
  end;
end;
```

```
function receive_longint: longint;
var msw,lsw : word;
   data : longint;
begin
 check_dav;
 receive_longint := receive_sni_data;
end; { of receive_longint }

function receive_single: single;
var msw,lsw : word;
begin
 check_dav;
 receive_single := longint_to_single(receive_sni_data);
end; { of receive_single }

procedure send_longint(data : longint);
begin
 check_rfi;
 send_sni_data(data);
end; { of send_longint }

procedure send_single(x : single);
var data : longint;
begin
 data := single_to_longint(x);
 check_rfi;
 send_sni_data(data);
end; { of send_single }

Procedure reset_sm8;
begin
 mem[segment:sm8_off+$1e] := 0; { reset sm8 }
 mem[segment:sm8_off+$1a] := 0; { set the state machine counter to 0 }
 mem[segment:sm8_off+$18] := 0; { set it to non-test mode }
 mem[segment:sm8_off+$16] := 0; { set it to stop mode }
end;

Procedure start_system;
begin
 mem[segment:sm8_off+$16] := 1;
end;

Procedure stop_system;
begin
```

```
 mem[segment:sm8_off+$16] := 0;
end;

Procedure start_processor;
var x : byte;
begin
 x := mem[segment:sni_off+0];
 mem[segment:sni_off+0] := $80 + (x and $3f);
 if mem[segment:sni_off+0] <> ($80 + (x and $3f)) then
  writeln('Failed to start from sni');
end;

Procedure stop_processor;
var x : byte;
begin
 x := mem[segment:sni_off+0];
 mem[segment:sni_off+0] := $40 + (x and $3f);
 if mem[segment:sni_off+0] <> ($40 + (x and $3f)) then
  writeln('Failed to stop from sni');
end;

Procedure initialize;
var address : integer;
begin
{ writeln('  - Multibus adapter card is set to 64k window at $0D0000 '); }
 address := port[$200]; { initialize multibus adaptor card }
{ writeln('  - Multibus page register is set at IO address $202 '); }
 port[$202] := 4;  { corresponding to hex switch on the GT-WWDP board }
 reset_sm8;
end; { of initialize }

procedure set_board_id(board_id:byte);
begin
 port[$202] := board_id;
end;

procedure write_seq_inst(pc:word;data:longint);
var d,p : longint_overlay;
begin
 { write to pc register }
 p.w[0] := pc;
 mem[segment:SEQ_off+0] := p.b[0];
 mem[segment:SEQ_off+2] := p.b[1];
```

```
 { write to data register }
 d.l := data;
 mem[segment:SEQ_off+4] := d.b[0];
 mem[segment:SEQ_off+6] := d.b[1];
 mem[segment:SEQ_off+8] := d.b[2];
 mem[segment:SEQ_off+10] := d.b[3];
 { issue a memory write }
 mem[segment:SEQ_off+12] := 1;
end;

procedure verify_seq_inst(pc:word;data:longint);
var rd_data,ck_data : longint_overlay;
   msw,lsw : word;
   p : longint_overlay;
begin
  { write to pc register }
  p.w[0] := pc;
  mem[segment:SEQ_off+0] := p.b[0];
  mem[segment:SEQ_off+2] := p.b[1];
  rd_data.b[0] := mem[segment:seq_off+0];
  rd_data.b[1] := mem[segment:seq_off+2];
  rd_data.b[2] := mem[segment:seq_off+4];
  rd_data.b[3] := mem[segment:seq_off+6];
  data := data and $07ffffff;
  rd_data.l := rd_data.l and $07ffffff;
  if rd_data.l <> data then
  begin
    writeln('Sequencer error at location ',pc);
    writeln('  expected : ',longint_to_hex(data));
    writeln('  read     : ',longint_to_hex(rd_data.l));
    write('continue ? <CR> ');readln;
  end;
 end;

 procedure write_dr_inst(pc:word;data1:byte;data0:longint);
 var d,p : longint_overlay;
 begin
   { write to pc register }
   p.w[0] := pc;
   mem[segment:dr_off+0] := p.b[0];
```

```
  mem[segment:dr_off+2] := p.b[1];
  { write to data register }
  d.l := data0;
  mem[segment:dr_off+4] := d.b[0];
  mem[segment:dr_off+6] := d.b[1];
  mem[segment:dr_off+8] := d.b[2];
  mem[segment:dr_off+10] := d.b[3];
  mem[segment:dr_off+12] := data1;
end;

procedure verify_dr_inst(pc:word;data1:byte;data0:longint);
var rd_data1 : byte;
    rd_data0,p : longint_overlay;
begin
  { write to pc register }
  p.w[0] := pc;
  mem[segment:dr_off+0] := p.b[0];
  mem[segment:dr_off+2] := p.b[1];
  { read data }
  rd_data0.b[0] := mem[segment:dr_off+4];
  rd_data0.b[1] := mem[segment:dr_off+6];
  rd_data0.b[2] := mem[segment:dr_off+8];
  rd_data0.b[3] := mem[segment:dr_off+10];
  rd_data1      := mem[segment:dr_off+12];
  if (rd_data1 <> data1) or (rd_data0.l <> data0) then
  begin
    writeln('Dataram error at location ',pc);
    writeln('  expected : ',byte_to_hex(data1),' ',longint_to_hex(data0));
    writeln('  read     : ',byte_to_hex(rd_data1),' ',longint_to_hex(rd_data0.l));
    write('continue ? <CR> ');readln;
  end;
end;

Procedure Test_SEQ;
var addr : longint;
    i : integer;
begin
  writeln('Testing sequencer memory');
  for addr := 0 to 511 do write_seq_inst(addr,addr + (addr shl 9) + (addr shl 18) + (addr shl 27));
  for addr := 0 to 511 do verify_seq_inst(addr,addr+ (addr shl 9) + (addr shl 18) + (addr shl 27));
```

```
 for i := 0 to longint_pattern_no do
 begin
  write('.');
  for addr := 0 to 511 do write_seq_inst(addr,longint_pattern[i]);
  for addr := 0 to 511 do verify_seq_inst(addr,longint_pattern[i]);
 end;
 writeln;
 writeln('Sequencer memory test completed');
end;

Procedure Test_DR;
var addr,i : word;
   filler : longint;
begin
 writeln(' Testing dataram memory ');
 for addr := 0 to 511 do
  write_dr_inst(addr,addr and $ff,addr + (addr shl 9) + (addr shl 18) + (addr shl 27));
 for addr := 0 to 511 do
  verify_dr_inst(addr,addr and $ff,addr + (addr shl 9) + (addr shl 18) + (addr shl 27));
 for i := 0 to longint_pattern_no do
 begin
  write('.');
  for addr := 0 to 511 do write_dr_inst(addr,longint_pattern[i] and $ff,longint_pattern[i]);
  for addr := 0 to 511 do verify_dr_inst(addr,longint_pattern[i] and $ff,longint_pattern[i]);
 end;
 for i := 0 to byte_pattern_no do
 begin
  write('.');
  filler := byte_pattern[i];
  filler := filler + filler shl 8 + filler shl 16 + filler shl 24;
  for addr := 0 to 511 do write_dr_inst(addr,byte_pattern[i],filler);
  for addr := 0 to 511 do verify_dr_inst(addr,byte_pattern[i],filler);
 end;
 writeln;
 writeln('Dataram memory testing completed');
end;

Procedure Test_SM8;
const
 chip_reset = $f;
```

```
 state_mach_mask = $e;
 state_mach_write = $d;
 test_latch_write = $c;
 run_latch_write = $b;
 sel_latch_write = $a;
 dav_latch_write = $9;
 rfi_latch_write = $8;
var
 data : word;
begin
 { reset the sm8 }
 mem[segment:sm8_off+chip_reset] := 0;
 { setting the sm8 in test mode }
 mem[segment:sm8_Off+test_latch_write] := 1;
 { reset the state machine counter }
 mem[segment:sm8_off+state_mach_write] := 0;
 { set sm8 to run mode }
 mem[segment:sm8_off+run_latch_write] := 1;
 { checking rfi_latch }
 mem[segment:sm8_off+rfi_latch_write] := $55;
 data := mem[segment:sm8_off];
 if data = $55 then
  writeln('rfi latch verified : $55')
 else
  writeln('written : $55, read : ',word_to_hex(data));
end;

procedure verify_longint(rcv,expected: longint);
begin
 if rcv = expected then
  writeln('verified : ',longint_to_hex(rcv))
 else
 begin
  writeln('error');
  writeln('rcv      : ',longint_to_hex(rcv));
  writeln('expected : ',longint_to_hex(expected));
  write('continue ? <CR> ');readln;
 end;
end;
```

```
procedure verify_word(rcv,expected: word);
begin
 if rcv = expected then
  writeln('verified : ',word_to_hex(rcv))
 else
 begin
  writeln('error');
  writeln('send     : ',word_to_hex(rcv));
  writeln('received : ',word_to_hex(expected));
  write('continue ? <CR> ');readln;
 end;
end;

procedure verify_byte(rcv,expected: byte);
begin
 if rcv = expected then
  writeln('verified : ',byte_to_hex(rcv))
 else
 begin
  writeln('error');
  writeln('send     : ',byte_to_hex(rcv));
  writeln('received : ',byte_to_hex(expected));
  write('continue ? <CR> ');readln;
 end;
end;

Procedure Test_SNI;
var addr,i : word;
begin
  stop_system;
  write('Processor stopped ? <CR>'); readln;
  start_system;
  write('Processor started ? <CR>'); readln;
  { toggle host_dav line }
  mem[segment:sni_off+7] := $55;
  mem[segment:sni_off+6] := $55;
  mem[segment:sni_off+5] := $55;
  mem[segment:sni_off+4] := $55;
  write('Host dav toggled ? <CR>'); readln;
  for addr := 0 to 2 do
```

```
 begin
   for i := 0 to byte_pattern_no do
{     verify_longint(SNI_off,addr,byte_pattern[i]); }
   writeln;
 end;
 writeln('test completed');
end;

begin
 for i := 0 to 31 do longint_pattern[i] := 1 shl i;
 for i := 0 to 31 do longint_pattern[32+i] := not (i shl i);
 longint_pattern[64] := 0;
 longint_pattern[65] := $ffffffff;
 longint_pattern[66] := $55555555;
 longint_pattern[67] := $aaaaaaaa;
 for i := 0 to 7 do byte_pattern[i] := 1 shl i;
 for i := 0 to 7 do byte_pattern[8+i] := not (i shl i);
 byte_pattern[16] := 0;
 byte_pattern[17] := $ff;
 byte_pattern[18] := $55;
 byte_pattern[19] := $aa;
 for i := 0 to 15 do word_pattern[i] := 1 shl i;
 for i := 0 to 15 do word_pattern[16+i] := not (i shl i);
 word_pattern[32] := 0;
 word_pattern[33] := $ffff;
 word_pattern[34] := $5555;
 word_pattern[35] := $aaaa;
 error := false;
 verify_count := 0;
 initialize;
end.
```