CONTINUATION OF RESEARCH INTO LANGUAGE CONCEPTS FOR THE MISSION SUPPORT ENVIRONMENT

FINAL REPORT

NASA Grant No. NAG 9-435
SwRI Project No. 05-3531

Prepared for:

NASA
Johnson Space Center
Houston, TX 77058

September 5, 1991

SOUTHWEST RESEARCH INSTITUTE
SAN ANTONIO
DETROIT
HOUSTON
WASHINGTON, DC

SOUTHWEST RESEARCH INSTITUTE

6220 CULEBRA ROAD • POST OFFICE DRAWER 28510 • SAN ANTONIO, TEXAS, USA 78228-0510 • (512) 684-5111 • TELEX 244846

September 10, 1991

Ms. Maryland Edwards
NASA-Johnson Space Center
Building 12, DJ23
Houston, Texas 77058

Subject: Delivery of Final Report for the Continuation of Research Into Language Concepts; NASA Grant No. NAG9-435; SwRI Project 05-3531

Dear Ms. Edwards,

Enclosed is the final report for the Continuation of Research Into Language Concepts research grant, NASA Grant 9-435. The final report contains a summary of the efforts performed on the grant. The final report contains the following sections:

A general description of the efforts performed on the grant is contained in the first section of the final report. This section provides information about the background for the research that was performed and also describes the major efforts of the grant.

Several documents which were generated on the grant are included in the middle sections of the final report. The Graphical Comp Builder Prototype documents and the Ada investigation documents are included.

A copy of the source code of the MOTIF version of the Graphical Comp Builder Prototype is included in the last section of the final report.

If you have any questions or comments, please feel free to call Tim Barton at (512) 522-3540, or Dr. Steven W. Dellenback at (512) 522-3914.

Sincerely,

Susan B Crumrine

for Melvin A. Schrader
Director
Data Systems Department

TJB:vc

Enclosures

cc: Susan B. Crumrine SBC
Timothy J. Barton
Steven W. Dellenback
NASA Scientific and Information Facility (2 copies)

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

CONTINUATION OF RESEARCH INTO LANGUAGE CONCEPTS FOR THE MISSION SUPPORT ENVIRONMENT

FINAL REPORT

NASA Grant No. NAG 9-435
SwRI Project No. 05-3531

Prepared by:
Timothy J. Barton
Jeremiah M. Ratner

Prepared for:
NASA
Johnson Space Center
Houston, TX 77058

September 5, 1991

Approved:

Melvin A. Schrader, Director
Data Systems Department

Table of Contents

1.0 Introduction

Southwest Research Institute (SwRI) was awarded a grant by NASA-Johnson Space Center (JSC) to perform research in the area of Specification Driven Languages (NASA Grant NAG 9-339). The purpose of the research was to investigate alternative programming techniques/concepts which could be utilized in control center software development environments. At the conclusion of the Specification Driven Language Grant, a follow on grant (NASA Grant NAG 9-435) was awarded to further pursue Specification Driven Languages and to prototype the concepts investigated during the previous grant.

This final report is a summary of the work completed under the follow on grant. In summary, the grant has focused on the following areas of research:

- A concept for a more intuitive and graphically based Computation (Comp) Builder was developed. This concept is the result of knowledge gained during the previous grant and research performed on the current grant. This concept is described in a document entitled "Graphical Comp Environment Concepts and Prototype Design Specifications" and is contained in Appendix A.
- The Graphical Comp Builder Prototype was developed to demonstrate the concepts which were researched during the early phases of this grant. The Graphical Comp Builder Prototype is an X Windows based graphical tool which allows the user to build Comps using graphical symbols. The prototype allowed NASA users to become familiar with the new concepts and allowed the research team to gain feedback on the viability of the new concepts.
- Investigation has been conducted to determine the availability and suitability of the Ada programming language for the development of future control center type software. The Space Station Freedom Project (SSFP) has identified Ada as the desired programming language for the development of Space Station Control Center (SSCC) software systems. The Department of Defense (DoD) has mandated Ada as the programming language for all new DoD software. Due to these mandates and related directions within private industry, an investigation of Ada was necessary.

The results of the research areas described above are contained in the following sections and appendices of this final report.

2.0 Graphical Comp Environment Concepts

One of the main goals of the research grant was the development of concepts for a more graphical means to represent comps. Most comps are constructed from engineering diagrams and flow charts which describe the system to be monitored. These engineering diagrams and flow charts are very graphical in nature. Using existing control center software, these engineering diagrams and flow charts must be manually converted into textual algorithms. These textual algorithms are then manually entered into the comp builder software.

A concept of a software system was developed as part of this research grant which would allow comps to be entered into the comp builder software in a manner which more closely matches the engineering diagrams and flow charts used to describe the systems. This allows comps to be constructed in a more natural and direct fashion, and more accurately due to the removal of the manual conversion from a graphical representation to a textual representation. The Graphical Comp Environment Concepts and Prototype Design Specifications are included as Appendix A of this final report. The Graphical Comp Environment Concepts and Prototype Design Specifications contain a concept for a Comp development, maintenance, and operations environment which is more graphically based than existing textually based environments.

A prototype demonstration of portions of the Graphical Comp Environment Concepts has been completed. The Graphical Comp Builder portion of the Graphical Comp Environment has been prototyped and demonstrated to NASA. The Graphical Comp Builder portion of the concept determines how the formally textual comps will be graphical represented. The Graphical Comp Builder portion of the environment is also the least dependent on the facilities provided by the underlying hardware and operating system software. The Graphical Comp Execution Environment is very dependent on the underlying hardware and operating system software, so this portion of the concept was not prototyped.

2.1 Graphical Comp Builder Prototype Specifications

The Graphical Comp Builder Prototype Specifications were developed to guide the development of the demonstration prototype. The knowledge gained on the prior grant during the review of the Computation Development Environment (CODE) influenced the development of the Graphical Comp Builder Prototype Specifications. This knowledge was used to ensure that the specifications identify a software system that is relevant and applicable to a control center environment.

The concepts demonstrated by the Graphical Comp Builder Prototype are a departure from the textually represented comps. The initial phase of the research grant included an investigation into the various methods available to graphically represent information similar in nature to the data contained in the engineering diagrams and flow charts used at NASA.

2.1.1 Graphically Represented Comps

A portion of the research performed on the grant has been in the area of graphically representing data. Numerous software packages were investigated to determine the currently available strategies for representing data similar to the engineering diagrams and flow charts used by control center personnel. Each software package was installed at SwRI and exercised for several days so a thorough understanding of the mechanisms used to graphically represent data could be assessed. Once the package had been exercised and evaluated, a review of the graphical representation mechanisms was developed.

Numerous software packages were obtained and exercised during the investigation period. The software packages ranged from very complex and expensive UNIX work station software, to inexpensive IBM PC programs. These software packages were often very different in nature and function, but all utilized a graphical means to represent data. Some of the packages investigated included:

- flow chart generation packages
- microwave circuitry design/simulation packages
- schematic design/drawing packages
- PC board layout packages
- graphics drawing packages

Each package was thoroughly investigated and a review of each package was developed. Packages which stood out from the others, either because of their effectiveness in representing graphical data or because of their gross ineffectiveness, were identified. These packages were demonstrated to NASA during a visit to SwRI on August 3, 1990.

2.1.2 Graphical User Interface

Another important part in the development of the Graphical Comp Builder Prototype was the selection of the Graphical User Interface (GUI) and the use of the GUI within the Prototype. The previous grant had determined that even though CODE used a mouse and graphics, it was not always intuitive or easy to use. The X Windows prototypes developed on the previous grant improved the GUI of CODE so it was much easier and faster to use. The development of the Graphical Comp Builder Prototype Specifications attempted to ensure that an intuitive and efficient user interface was specified.

The review of the various software packages, performed to identify graphical representation strategies, also provided information into the effective use of a GUI and the mechanisms of each GUI which would benefit the Graphical Comp Builder Prototype. The strengths and weaknesses of each package were recorded so the most desirable features of each package could be specified for use in the Graphical Comp Builder Prototype if appropriate.

The various elements of the GUI utilized by the various software packages were also demonstrated to NASA during the August 3, 1990, visit to SwRI.

2.1.3 MOAL Language

The prior grant focused extensively on control center language issues. As a result of the research on the preceding grant, a Backus-Naur Form (BNF) representation of the Mission Operations Application Language (MOAL) was developed. The specification of the Graphical Comp Builder Prototype identifies the use of the MOAL language for compatibility with comps developed with the various versions of CODE. The use of the MOAL language within the new Graphical Comp Builder Prototype is an example of integration of research performed during the prior grant into the existing grant.

3.0 Graphical Comp Builder Prototype Development

As stated previously, the goal of the research grant was to investigate language concepts and implementation strategies which could be used to improve control center computation development and execution environments. The Graphical Comp Environment Concepts and Prototype Design Specifications were developed to identify the technologies to use to improve comp environments. Once the Graphical Comp Builder Prototype concepts were formulated, implementation of the Graphical Comp Builder Prototype was begun.

The Graphical Comp Builder Prototype is a user friendly tool which allows flight controllers or other users to graphically represent comps. The Graphical Comp Builder Prototype's graphical representation of a comp is very similar to the engineering diagrams and flow charts used to describe the systems being monitored in control center environments. The development of the Graphical Comp Builder Prototype utilized the technologies identified in the Graphical Comp Builder portion of the Graphical Comp Environment Concepts and Prototype Design Specifications.

The Graphical Comp Builder Prototype was developed in the C language on UNIX-based Sun workstations. The first version of the Graphical Comp Builder Prototype used the C language XView toolkit implementation of the OpenLook GUI. A partially completed OpenLook version of the Graphical Comp Builder Prototype was demonstrated to NASA on two occasions. Once experience had been gained with the XView toolkit and the capabilities of the XView toolkit had been assessed, the Prototype was ported to the MOTIF GUI. The Graphical Comp Builder Prototype was completed using the MOTIF GUI. The MOTIF and OpenLook GUI's are currently competing to become the industry standard. The research team was able to assess the strengths and weaknesses of both GUI's during the development of the Graphical Comp Builder Prototype.

The MOTIF version of the Graphical Comp Builder Prototype has been completed and demonstrated to NASA-JSC. A complete listing of the C language source code for the MOTIF version of the Graphical Comp Builder Prototype is included in Appendix F of this final report.

3.1 Graphical Comp Builder Prototype Documentation

A discussion of the implementation of the MOTIF version of the Graphical Comp Builder Prototype is contained in the Graphical Comp Builder Prototype Documentation. The Graphical Comp Builder Prototype Documentation was developed to provide information about the following aspects of the Graphical Comp Builder Prototype:

- The major concepts of the Graphical Comp Builder Prototype are identified and their implementation within the Prototype are discussed in the first section of the Graphical Comp Builder Prototype Documentation. This section discusses the graphical nature of the Prototype and also discusses the Comp hierarchy, report generation, and automatic code generation features of the Graphical Comp Builder Prototype.
- A more complete list of the various features contained within the Graphical Comp Builder Prototype are contained in the second section of the Graphical Comp Builder Prototype Documentation.
- The third section of the Graphical Comp Builder Prototype Documentation contains a brief discussion of the implementation of the Prototype. This section identifies the module hierarchy, data files, and data structures of the Graphical Comp Builder Prototype.

The Graphical Comp Builder Prototype Documentation is contained in Appendix B of this final report.

4.0 Investigation of Ada for Control Center Software

As mentioned in the Introduction, NASA has selected Ada to be used in the development of any new software for SSCC and the DoD has mandated Ada as the programming language for any new DoD software. Therefore, a preliminary investigation into an Ada version of the Graphical Comp Builder was conducted. Initial investigation into Ada's acceptance at NASA and the performance of a program written in Ada are the two areas of investigation to date.

4.1 Ada Investigation

The first step in the investigation into Ada was to determine Ada's acceptance and usage. A member of the research team attended the Third Annual NASA Ada User's Symposium to determine Ada's acceptance within the NASA community. A summary of the information gathered at the NASA Ada User's Symposium is contained in Appendix C.

The second step in the investigation of Ada focused on the relative performance of Ada programs on two workstations often used in control center environments. The Hartstone Benchmark was used to provided information regarding the performance of the executables produced by various Ada compilers. A summary of the findings from the Hartstone Benchmark investigation is also contained in Appendix C.

4.2 Investigation and Comparison of Ada and C

NASA is about to embark on the development of millions of lines of software for the SSCC and the Space Station Training Facility (SSTF). Most of the programs written for JSC have been written in the C or FORTRAN programming languages during the last 5 years. NASA had originally specified the use of Ada in the development of the SSCC and SSTF software, but in recent years Ada's acceptance has seemed to dwindle while the acceptance of the C programming language has accelerated rapidly. To investigate this further, SwRI performed a comparison of the C and Ada programming languages for control center environments. The results of the investigation were presented to NASA-JSC on March 8, 1991. A copy of the results of the investigation are contained in Appendix D of this final report.

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

GRAPHICAL COMP ENVIRONMENT CONCEPTS AND PROTOTYPE DESIGN SPECIFICATIONS

NASA Grant No. NAG 9-435
SwRI Project No. 05-3531

Prepared by:
Timothy J. Barton
Jeremiah M. Ratner

Prepared for:
NASA
Johnson Space Center
Houston TX 77058

January 28, 1991

Approved:

Melvin A. Schrader, Director
Data Systems Department

Table of Contents

1.0 Purpose

This document represents a concept for the Graphical Computational Program (Comp) Environment. A summary of the investigations made concerning the design and implementation of the Graphical Comp Builder Prototype (GCB) and its supporting compiler and execution environment are presented. These discussions will provide the basis from which the Graphical Comp Environment Prototypes will be built. They will also serve as one of the measures by which the completed Graphical Comp Environment concept and prototypes will be evaluated.

The prototype Design Specifications will be broken into the following major sections:

- Graphical Comp Builder Prototype (GCB)
- Graphical Comp Compiler Prototype (GCC)
- Graphical Comp Execution Environment Prototype (GCEE)

The Graphical Comp Environment will require all three of the above identified parts to be a complete and functional system. Each part will be a separate and independent program, but each will be designed to operate with the other two.

The Graphical Comp Builder Prototype will allow users to graphically build and maintain fault detection algorithms (Comps). The Graphical Comp Compiler will input information from the Graphical Comp Builder Prototype and produce a representation that is suitable for execution by the Graphical Comp Execution Environment Prototype. The Graphical Comp Execution Environment Prototype will control the execution of the fault detection algorithms produced by the compiler.

In the NASA "test bed" environment, the Graphical Comp Execution Environment Prototype will be used during flight operations, while the Graphical Comp Builder Prototype and Compiler will be used off-line during development mode operations. The Graphical Comp Builder Prototype and Compiler each can be used without the other two parts of the environment, but the Execution Environment will require executable Comps produced by the Compiler Prototype.

2.1 GCB Feature Specifications

The Graphical Comp Builder Prototype will provide a number of features to allow users, such as flight controllers and engineers, to very easily design and document fault detection algorithms. The number one goal of the Graphical Comp Builder Prototype is to be a tool which is easy to use, yet contains the elements which are necessary to build fault detection Comps. Many of the features specified address the ease of use goal.

The Graphical Comp Builder Prototype will also contain several features designed to aid the Comp implementor in the development of Comps and their supporting documentation. Some of these features include: a Comp Executer, path generator, and a graphical means to view logical expressions.

2.1.1 Graphical Interface

The Graphical Comp Builder Prototype will use a set of graphical symbols to represent the algorithms which are needed to perform fault detection and notification. The graphical nature of the Graphical Comp Builder Prototype will make the tool easier to use than existing Comp builders. The graphical interface specifications are discussed in their own section due to their importance to the success of the entire Graphical Comp Environment. The graphical interface specifications are contained in Section 2.2 on page 6.

2.1.2 MOAL Support

The Graphical Comp Builder Prototype will allow the user to construct Comps which are based on the MOAL (Mission Operations Application Language) constructs as identified in the Comp Builder / Comp Manager Level B Requirements (JSC-23459). The MOAL language has been specified by JSC flight controllers. The MOAL contains the language elements required by flight controllers to perform fault detection algorithms.

2.1.3 Report Generation

The Graphical Comp Builder Prototype will provide several reports to document the developed graphical Comps. The Graphical Comp Builder Prototype will generate a listing of all Comp inputs, outputs, and possible test cases. The Graphical Comp Builder Prototype will also generate a report detailing the algorithms contained in the Comps. The algorithm descriptions will be provided by the user during the specification of DECISION symbols.

The Graphical Comp Builder Prototype will generate a report identifying all possible execution paths through each Comp. This report may grow to be very large depending on the size of the Comp, so the user will be notified during the path generation process as to the number of paths. The user will be allowed to abort the path generation process.

2.1.4 Graphical Representation of Logical Expressions

The Graphical Comp Builder Prototype will allow the user to select a DECISION symbol to be graphically represented. This function will allow the user to see the logical expression of the DECISION symbol to be graphically represented. The logical operators will be represented as graphical symbols. This function will allow the logical expressions of DECISION symbols to be viewed in terms of AND gates and OR gates in a graphical manner. The graphical representation of logical expressions may be viewed or printed only. The logical expression cannot be modified or maintained by the graphical representation.

2.1.10 Command Line Comp Specification

The Graphical Comp Builder Prototype may be initially executed with a Graphical Comp name specified as an argument to the Graphical Comp Builder Prototype program. In the event that a Comp filename is specified, the Graphical Comp Builder Prototype will load the Comp into the Work Area on start-up.

2.2.2 Expression Builder

There will be two separate methods available for expression building within the Graphical Comp Builder Prototype when the user selects a DECISION symbol or a SET symbol. Upon selecting either a DECISION or SET symbol, the user will have two options: 1) typing the expression into a popup using the keyboard and having the expression parsed after it is completed, or 2) using a mouse-driven menu and having the choices for operator and operand constrained at each step in the input process. The second method will be almost identical to the mouse-driven expression builder in the various versions of the Comp Builder. The graphical expression builder will be called by the Expression Builder. The first action performed when building an expression will determine how the expression must be completed. If the user selects a token from the Expression Builder using the mouse, then the expression must be completed using the mouse and the Expression Builder. If the user begins typing in an expression, then the expression must be completed using the keyboard. The user will be given the option of aborting the building of an incomplete expression. After the expression building is aborted, the user will then have the option of building the expression with the other mechanism or of starting a new expression with the same mechanism.

The benefit of this approach is that the user who does not need the guidance of the editor may input expressions quickly, but without immediate error checking, while the novice user will be able to build both Comps and the expressions within the Comps using mouse driven menus. Both means of expression building will ensure that only logically sound and syntactically correct expressions are entered. The Expression Builder will step the user through the selection of the expression operands and operators in a way that is syntactically correct (See the MOTIF Comp Builder for more details.) The keyboard method will also ensure that syntactically correct Comps are built by parsing the expression after the user has completed typing the expression into the popup.

The user will also have the option of entering the logical form of the expression as well as the Comp expression. The logical form, if it is included, will appear in the schematic in place of the Comp form. If the logical form is not entered by the user, then the Comp expression will be displayed in the DECISION symbol. The logical form will not be parsed nor can it be constructed using the Expression Builder. The logical form is provided so the user can build graphical comps which are easier to understand and are independent of the language used to construct the executable Comp. This will allow Comps to be constructed with the logic and hiearchial structure required to perform the desired fault detection, but the execution dependent portion of the fault detection algorithm does not need to be completed.

Documentation about the expression for report generation or other purposes may also be entered at this point, although this text will not appear in the schematic. The Documentation field will be a much larger text field than either the logical or comp expression. The Documentation field will be available so the Comp builder can document the algorithms in a completely textual form. The Documentation field will be included in the documentation produced by the Graphical Comp Builder Prototype.

2.2.3 Graphical Symbol Placement Model

The Graphical Comp Builder Prototype will require the user to select a graphical symbol from a menu template for placement in the work area. The user will choose components from a menu template, and drag or place them in the desired location in the graphical Comp. The graphical placement model, i.e., the way in which components are placed and connected on the screen, will

2.2.5 Popups and Text Fields

The Graphical Comp Builder Prototype will use popups wherever necessary to input or display information to the user. User input text fields will implement the standard OpenLook text field editing functions to provide a user-friendly and consistent interface. The following text editing functions will be available in all text edit fields:

- Home and End keys
- Insert and Delete keys
- All other traditional OpenLook text edit functions, including mouse controlled positioning of the text cursor

2.2.6 Symbol/Work Area Manipulation Functions

The Graphical Comp Builder Prototype will utilize several modes of operation during the course of Comp creation and modification. During the different operating modes, the mouse pointer will be modified to indicate to the user that a different mode is in effect and a status area will also indicate the current mode.

If the user is in any mode other than Edit Mode, the status area will contain a CANCEL button which will abort the current function and return the user back to Edit Mode. The following are two typical sequences which illustrate the user completing a function and aborting a function:

Edit Mode -> Add Symbol -> Edit Mode -> Move Symbol -> Edit Mode
Edit Mode -> Add Symbol -> CANCEL -> Edit Mode

In both cases, the user is returned to Edit Mode.

The following sections specify the operations that can be performed on the graphical symbols. The functions allow the user to add symbols, connect symbols, and in general maintain the symbols in the active Comp.

2.2.6.1 Select Symbol

The Select Symbol function will allow the user to select a graphical symbol to be added to the current Comp and will allow the user to place the selected symbol in the work area. The user will select a symbol from the available symbols by placing the mouse pointer over the symbol and clicking with the left mouse button. The selected symbol will then be highlighted. As the pointer is moved into the work area, an outline of the symbol will be highlighted, allowing the user to place the symbol. The symbol will be placed when the user clicks the left most mouse button again. The user can abort the symbol placement by either clicking the left button outside the work area or by hitting the ESC key. The user can also abort the Select Symbol function by clicking the left mouse button on the CANCEL button in the status area. If the placed symbol requires additional information from the user, a popup will be displayed in the work area after the new symbol has been placed by the user.

2.2.6.2 Connect Symbols

The Connect Symbol function will allow the user to connect two graphical symbols. The Connect Symbol function will be activated by clicking the center mouse button in the starting symbol. If

2.2.6.5 Delete Symbol

The Delete Symbol function will allow the user to delete a graphical symbol and all lines entering and exiting the symbol. The Delete Symbol function will be available from the Edit Mode menu. After selecting Delete Symbol from the Edit Mode menu, the user can select a symbol for deletion by clicking the left mouse button on any cell allocated to a symbol.

The user may abort the Delete Symbol function at any step by clicking the CANCEL button in the status area or by hitting the ESC key.

2.2.6.6 Delete Line

The Delete Line function will allow the user to delete an existing logical connecting line. The Delete Line function will be available from the Edit Mode menu and will also be available during Edit Mode by clicking the middle mouse button while the shift key is depressed.

If the user selects a line that represents more than one logical line, the user will be asked to select an unambiguous segment of the line to be deleted.

The user may abort the Delete Line function at any time by clicking the CANCEL button in the status area or by hitting the ESC key.

2.2.6.7 Undo

The Undo function will undo the last n operations. The Undo function will be available from the Edit Mode menu (See the Mouse Function Summary in the next section.) The Undo function will store the last n operations in a stack. The initial value for n will be 1. This setting may be configurable by the user.

2.2.6.8 Print

The Print function will print the work area to any PostScript printer. The Graphical Comp Builder Prototype will initially only print to PostScript devices, but the Graphical Comp Builder Prototype will be designed with the intention of supporting additional printers. The default paper size will be 8 1/2" x 11". However, the tool will allow the user to dynamically alter the page size, so that the output may appear on a plotter or other device that uses larger paper size.

The Graphical Comp Builder Prototype work area will be a WYSIWG (What You See Is What You Get) window similar to applications on the NeXT computer and will operate similar to the sheet window in FrameMaker. If the default paper size is selected, the work area will represent a standard 8 1/2" x 11" sheet of paper and the resulting printed output will match the on screen window. A larger paper size will not affect the appearance of the work area; rather, the virtual work area will extend beyond the work area window, and the user may pan and zoom around the work area to view different sections of the schematic in the work area window.

2.2.6.9 Select Font and Symbol Size

The user will be able to dynamically change the sizes of symbols and the fonts of text. The function that changes font size will include the options to change all symbol font sizes, all new symbol font sizes, or only the font size of the currently selected symbol. In addition, the font sizes of label text, text not contained in a symbol, will be dynamically configurable. The user will select the font size menu option and then click on the symbol whose text is to be changed or anywhere on the label text to be changed, and a popup window will accept the new font size.

2.3 GCB Environment Specifications

The Graphical Comp Builder Prototype must be available to as many potential users as possible. These potential users have a varied assortment of computer hardware available on which to run the Graphical Comp Builder Prototype, so the Graphical Comp Builder Prototype will be written in as portable a way as is reasonably possible.

2.3.1 Software Environment

The Graphical Comp Builder Prototype will be written initially in the C programming language using X Windows to supply and control the graphical interface. The Graphical Comp Builder Prototype will be implemented eventually in the Ada programming language when support for X Windows is available for Ada. OpenLook will be the target X Windows widget set, but the design of the Graphical Comp Builder Prototype will not preclude a conversion to the MOTIF widget set to meet NASA's current and future usage requirements.

2.3.2 Hardware Environment

The Graphical Comp Builder Prototype will be implemented initially on UNIX-based, Sun SPARC Stations. As with any X Windows based application, the client portion of the application and the X server may reside on very different hardware platforms as long as a network connects the server and client. Any hardware platform that supports an X Windows server will be sufficient for the graphical interface portion of the Graphical Comp Builder Prototype. Any UNIX-based computer will be sufficient for the client portion of the Graphical Comp Builder Prototype. The client and server portions of the application may reside within the same host, or they may reside on very different hardware platforms as long as both are connected via an X Windows compatible network. This will allow any bitmapped display connected to a PC compatible, workstation, or mainframe to execute the Graphical Comp Builder Prototype.

```
        comp_create_date[],
    comp_create_time[],
    comp_update_date[],
    comp_update_time[],
    comp_purpose[];
}
```

2.4.1.2 Work Area Cell Structure

The Graphical Comp Builder Prototype will divide the work area into a number of small cells which will not be visible to the user. These cells will be used internally to manage the work area. The Graphical Comp Builder Prototype will maintain a bitmap of the cells in the work area. This bitmap will identify which type of construct is contained within each cell. Each cell can be either: empty, contain a line, contain a portion of a graphical symbol, or contain text. The work area cell map will not be dynamically allocated, so the structure size will be kept as small as possible.

The following is the cell map structure which will be used to identify each cell in the work area:

```
struct Cell {
    int     cell_type;              /* if -1, free cell, otherwise type indicator */
    union   {
            Symbol *symbol;         /* if -1, free cell, else index into symbolmap*/
            LineList *lines;        /*linked list of lines - cell may be part of >1 line*/
            } cell_entry;
}
```

The following stactically sized and allocated array of cell map strucutres will be maintained:

```
struct Cell  cell_map[MAX_ROWS*MAX_COLS};
```

2.4.1.3 Symbol Structure

The Graphical Comp Builder Prototype will manage an array of symbol structures which will identify all the structures contained in the work area. The symbol structure will identify the symbol type, its location in the work area, any connectivity to other symbols, and any other supporting information. The symbol structures will be dynamically allocated during the execution of the Graphical Comp Builder Prototype.

The symbol structure will contain a C "union" structure to identify the information that is specific to the different symbol types. The DECISION symbol will contain the logical expressions and TRUE and FALSE paths, whereas the PRINT symbol will contain the output text. The following structure will be used to identify each work area symbol:

```
struct Symbol {
    int     symbol_type;            /*identifies the type of symbol*/
    int     height, width;          /*size of symbol in cells*/
    int     ulcx, ulcy;             /*upper left corner x and y coordinates*/
    LineList *from_lines;           /*list of lines entering symbol*/
```

2.4.1.4 Line List Structure

The Line List Structure will identify a list of lines. The Line List will be used in two instances. One Line List Structure will be used to identify the list of lines which occupy a work area cell. A second Line List Structure will be used to identify the list of lines which enter each symbol. The dynamically allocated linked list structure of the Line List will be used in place of a statically sized array. This will reduce the size of the Cell Map Structure and Symbol Structure.

```
struct LineList {
    LineMap     *line;          /* index into line map */
    LineList    *next;          /* next line for same cell or symbol */
}
```

2.4.1.5 Line Map Structure

The Line Map Structure will identify each logical line. The Line Map Structure will contain the symbols connected by the line, and will point to a Line Cell Map Structure which will list all the work area cells used by the line.

```
struct LineMap {
    LineCellMap *line;          /* list of line cells */
    int     from, to;           /* indices to symbol map */
}

struct linestruct     linemap[MAXLINES];
```

2.4.1.6 Line Cell Map Structure

The Line Cell Map Structure will identify all the work area cells which are utilized by a single logical line. The Line Cell Map will be a NULL terminated linked list of cell indexes.

```
struct LineCellMap {
    int     cell_row, cell_col;     /* cell map indicies */
    LineCellMap *next;              /* pointer to next cell map entry */
}
```

2.4.1.7 UNDO Stack Structure

The UNDO Stack Structure will maintain a list of the last operations performed by the user. This stack will allow the user to "undo" certain operations.

This directory may exist anywhere in the directory hiearchy, but must be named CompLib. The initial CompLib directory will be created in the GCB directory in the user's home directory.

2.4.2.4 WorkStation Global Table

The Graphical Comp Builder Prototype will maintain a WorkStation Global Table. This file will contain the names and data types of the values which will be output to other workstation applications by the Graphical Comps.

2.4.2.5 MSID Table

The Graphical Comp Builder Prototype's main goal is the retrieval of data values and the execution of fault detection algorithms on those data values. The Graphical Comp Builder Prototype will initially implement the Space Shuttle data acquisition mechanisms, and the Space Station Freedom data acquisition mechanisms will be incorporated in the Graphical Comp Builder Prototype at a later date. The MSID table is a file that is used in the Space Shuttle data acquisition mechanisms. The MSID table identifies the available data points and their corresponding data types.

The Graphical Comp Builder Prototype will maintain an MSID table. The Graphical Comp Builder Prototype will maintain a field in the MSID table to identify the MSID's that are used most frequently by the user.

2.4.2.6 User Macro Files

The user will be able to store named macro files. These files will be created by the Graphical Comp Builder Prototype and will be stored in the user specified macro directory. The default directory for macro files initially will be the GCB directory in the user's home directory. The macro directory path may be modified and the new path may be recorded in the user's configuration file. Macro files will have the following extension: *.MAC.

2.4.2.7 User Configuration File

The Graphical Comp Builder Prototype will have a number of features that are user configurable. The user will have the option of saving their desired configuration to a file which is accessed every time the Graphical Comp Builder Prototype is executed. The user configuration file will be stored in the GCB directory in the user's home directory and will have the following name: User.CFG.

The following options will be stored in the User Configuration File:

- The directory path containing the Graphical Comp Files last accessed by the user will be stored. If a Graphical Comp File is not supplied on the command line at execution, then the Graphical Comp Builder Prototype will default to the last directory that was accessed by the user. If the user is executing the Graphical Comp Builder Prototype for the first time, then the GDB directory of the user's home directory will be used as the default. The default directory will be displayed to the user during Graphical Comp file selection.
- The Comp Library directory path will be one of the options stored in the user's configuration file. This will allow the user to place the library directory where needed.
- The path to the user's macro files can be specified in the User Configuration File. This will allow the user to place their macro files where needed.

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

GRAPHICAL COMP BUILDER PROTOTYPE DOCUMENTATION

NASA Grant No. NAG 9-435
SwRI Project No. 05-3531

Prepared by:
Timothy J. Barton
Jeremiah M. Ratner

Prepared for:
NASA
Johnson Space Center
Houston TX 77058

August 28, 1991

Approved:

Melvin A. Schrader

Melvin A. Schrader, Director
Data Systems Department

Table of Contents

1.0 Purpose

This document describes the functionality and implementation of the Graphical Comp Builder Prototype. The Graphical Comp Builder Prototype allows users to graphically build and maintain fault detection algorithms (Comps) for control center environments.

The Graphical Comp Builder Prototype was designed and implemented as the result of investigation and experience in the following areas:

- existing NASA Comp Builders
- existing NASA high-level languages, such as UIL, GOAL, etc.
- existing applications which manipulate graphical symbols

Knowledge gained from investigations of these three areas formed the basis for the specification and subsequent implementation of the Graphical Comp Builder Prototype.

This document describes the features provided by the Graphical Comp Builder Prototype and also describes the implementation of these features. A description of the main concepts of the Graphical Comp Builder Prototype is contained in the following section, Section 2.0. A list of the features provided by the Graphical Comp Builder Prototype is contained in Section 3.0, and a description of the implementation of the features is contained in Section 4.0.

2.0 Graphical Comp Builder Prototype Concepts

The Graphical Comp Builder Prototype was designed around several basic concepts to make the process of developing and maintaining Comp algorithms an easier and more intuitive task. This section describes the key concepts of the Graphical Comp Builder Prototype.

2.1 Graphical Representation of Comp Algorithms

One of the most important aspects of the Graphical Comp Builder Prototype is evident in the Prototype's name. The Graphical Comp Builder Prototype allows the user to graphically represent Comp algorithms instead of using traditional text based methods. Comp algorithms are often designed and documented using graphical representations. Comp algorithm documentation often looks much like a flowchart. The Graphical Comp Builder Prototype was designed to support the development and maintenance of Comp algorithms using flowchart type constructs. This feature makes it very easy to implement Comp algorithms from design documentation and allows the Graphical Comp Builder Prototype user to maintain Comp algorithms using a representation that is familiar and comfortable.

The graphical Comp algorithm representation supported by the Graphical Comp Builder Prototype is based on a set of graphical symbols and logic flow lines. The user builds the graphical Comp algorithm from the graphical symbols. The order of execution, or algorithm flow, is determined by the way in which the symbols are connected by the user.

2.1.1 Graphical Symbols

The Graphical Comp Builder Prototype uses a predefined set of graphical symbols to represent the various actions that are performed by Comp algorithms. These symbols form the basic building blocks of each Comp algorithm and represent the basic operations which are performed in every Comp. The set of graphical symbols is available in a palette menu in the lower left-hand corner of the Prototype's screen. The user selects the desired symbol and then places it in the Comp Work Area (see the Work Area Section on page 4 for more information about the Work Area) using the mouse. The user may then connect the symbols in the Work Area to define the Comp algorithm flow during Comp execution. The Graphical Comp Builder Prototype allows the user to construct Comps from the following graphical symbols:

BEGIN	oval in shape, no entry point, single exit path. This symbol represents the beginning of the Comp.
END	circle in shape, multiple entry points, no exit path. This symbol indicates the end of the Comp.
IF	modified diamond shape, multiple entry points, two exit paths. This symbol is used to enter logical expressions into the Comp.
SET	rectangle shaped, multiple entry points, single exit path. This symbol represents the "setting" of a variable or signal, similar to a programming language assignment statement.
PRINT	hollerith card (punched card) shaped, multiple entry points, single exit path. This symbol represents the output of a formatted string during the execution of the Comp.
CALL	rounded rectangle shape, multiple entry points, single exit path. This symbol indicates that the current Comp is to be suspended, and the Comp named

within the symbol is to be started. When the named Comp completes its execution (reaches its END symbol), execution resumes in the current Comp at the next symbol. This symbol functions much like a subroutine "call" in a C or FORTRAN language program.

ACTIVATE modified rectangle shape, multiple entry points, single exit path. This symbol represents the start of execution of an asynchronous Comp in parallel with the current Comp. This symbol functions much like a fork() call in a C language program.

STOP octagon (stop sign) shaped, multiple entry points, single exit path. This symbol indicates the termination of a parallel Comp. This symbol is used to "stop" a Comp which was spawned by an ACTIVATE symbol.

PAUSE alarm clock shaped, multiple entry points, single exit path. This symbol causes the Comp to pause for a specified time period.

It is from this collection of graphical symbols that the user constructs a Comp algorithm.

The shape of the IF, PRINT, and SET symbols were chosen for their expandability. These symbols often may have to expand to allow the Comp designer to enter long expressions or text strings. The squared sides of these symbols may expand or shrink depending on the size of the expression entered by the Comp designer.

2.1.2 Logical Connecting Lines

Once two or more graphical symbols have been selected and entered in the Work Area, then the user may logically connect the symbols to define the control flow between the symbols during execution. The user may very easily introduce looping or recursion simply by connecting one symbol to another.

2.1.3 Expression Builder

The IF and SET graphical symbols may contain textual logical expressions. It is through these two symbols that much of the work of a Comp algorithm is performed. The IF symbol is used to represent a logical expression which returns a true or false value. The result of the logical expression determines which connecting line is traversed out of the symbol. The SET symbol is used to assign values to local variables, global variables, and to Work Station Globals. The SET symbol may be used to put values into Object Access (data acquisition) for retrieval by other control center applications.

The logical expressions in the IF and SET symbols are called Comp Expressions. Comp Expressions may be entered via the keyboard or the expressions may be constructed using the Expression Builder. The Expression Builder is a collection of logical expression building blocks contained in a menu. The Expression Builder steps the user through the building of Comp Expressions by activating only the expression building blocks that are valid for the current state of the logical expression. The Graphical Comp Builder Prototype provides the Expression Builder for novice users and also allows the more proficient user to enter expressions directly via the keyboard if desired.

Each IF and SET graphical symbol also allows the user to enter two levels of supporting documentation for each expression. The user may enter a short description of the expression and a much longer textual description if desired. The short description may be used to aid the user in the

building of the Comp algorithm, whereas the larger description field is designed to provide supporting documentation in the printed reports generated by the Graphical Comp Builder Prototype. These two documentation fields allow the user to logically design and document Comp algorithms. The logical design of the Comp algorithm may be constructed and documented by one user, and the Comp Expressions may be entered at a later date, possibly by another user. The documentation support provided by the Graphical Comp Builder Prototype aids the Comp design and development process by maintaining the Comp documentation with the Comp itself.

2.1.4 Work Area

The Graphical Comp Builder Prototype allows the user to construct Comp algorithms using graphical symbols and logical connecting lines. The building of the Comp algorithm takes place within the Work Area. The Work Area is a large, scrolled area in which the graphical symbols are placed and connected. A majority of the time spent building a Comp with the Graphical Comp Builder Prototype is spent in manipulating the graphical symbols and the connecting lines in the Work Area. The Graphical Comp Builder Prototype has been designed to provide the user with powerful, yet easy to use functions to manipulate the graphical symbols and connecting lines in the Work Area.

2.1.4.1 Graphical Symbol Placement Model

The Graphical Comp Builder Prototype allows the user to select a graphical symbol from a palette menu and then place the symbol in the Work Area. The user may then position the graphical symbol in any unoccupied place in the Work Area by "dragging" the symbol using the mouse. The graphical placement model, i.e., the way in which components are placed and connected on the screen, is a hybrid of the two styles used in most software packages that involve the manipulation of graphical symbols. These two methods are:

- A grid-based approach
- A cell-based approach

In the grid-based approach, the drawing area is defined by a point grid. A symbol is "snapped" to the defining set of grid points closest to the desired location; a drawn line is snapped to the grid points that conform most closely to its path. The benefit of grid-based approach is the precision with which components can be placed is configured by the granularity of the grid. The grid-based approach suffers due to the fact that it is harder to place and connect symbols than in other methods.

In the cell-based model, the drawing area is defined by a grid of cells. Each component occupies exactly one cell. The advantage of this approach is that symbol placement is easy; the user doesn't have to be concerned with precise placement or uniform spacing. The disadvantage is the flexibility of arbitrary placement is lost. In addition, connectivity becomes problematic: a line is a component, and so no more than one line may occupy a cell. Thus, all lines must be a cell width apart. This severely limits the line drawing capabilities of cell-based applications and negates their ease of use.

The Graphical Comp Builder Prototype uses a combination of these two approaches. The Work Area drawing region is composed of a grid of small, unseen cells, and symbols may occupy more than one cell. Connecting lines may also occupy more than one cell if the line is longer than one cell in length. Connecting lines are one cell in width but may be many cells in length. Each logical cell may contain only one of the following:

- a portion of a graphical symbol

- a portion of a connecting line

Symbol shapes may expand in one-cell gradations to contain long expressions.

The advantages of this hybrid approach are:

- Ease of symbol placement that is comparable to the basic cell-based approach but more flexible and efficient in its use of Work Area real estate -- there is less "external fragmentation."
- Flexibility of connecting line placement that approximates that of the grid-based approach, but because of the restriction that only one line may appear in a cell, gives the user more structured line placement.
- Arbitrarily large symbols may be constructed which contain long expressions. This is due to the fact that a single graphical symbol may occupy many logical cells.
- As with the grid-based approach, it is easier to place arbitrary label text.

The Graphical Comp Builder Prototype provides a number of functions to assist the user in the development of graphical Comp Elements. These functions allow the user to manipulate the graphical symbols and connecting lines in the Work Area.

2.2 Graphical Comp Hiearchy

The Graphical Comp Builder Prototype allows the user to construct and maintain Comp algorithms in a very logical and powerful fashion. The Graphical Comp Builder Prototype contains features which allow the user to group Comps according to function to aid the Comp maintenance process. The Graphical Comp Builder Prototype also contains features which allow the Comp algorithm designer to decompose algorithms into a number of smaller, more manageable components. The Graphical Comp Builder Prototype implements a hiearchy of three main components to allow the user to decompose algorithms into smaller, more manageable pieces:

- Position
- Comp
- Element

Each of these components of the Graphical Comp Builder Prototype are described in following sections.

2.2.1 Multiple Position Support

The Graphical Comp Builder Prototype was designed to allow the user to group Comp algorithms according to their function. In a control center environment, Comp algorithms are usually grouped together based on the flight control position for which they were designed. The Graphical Comp Builder Prototype contains and supports the concept of grouping and maintenance of Comp algorithms based on a flight control position. In the Graphical Comp Builder Prototype, the flight control position is the highest level in the Comp hiearchy.

2.2.2 Comp Structure

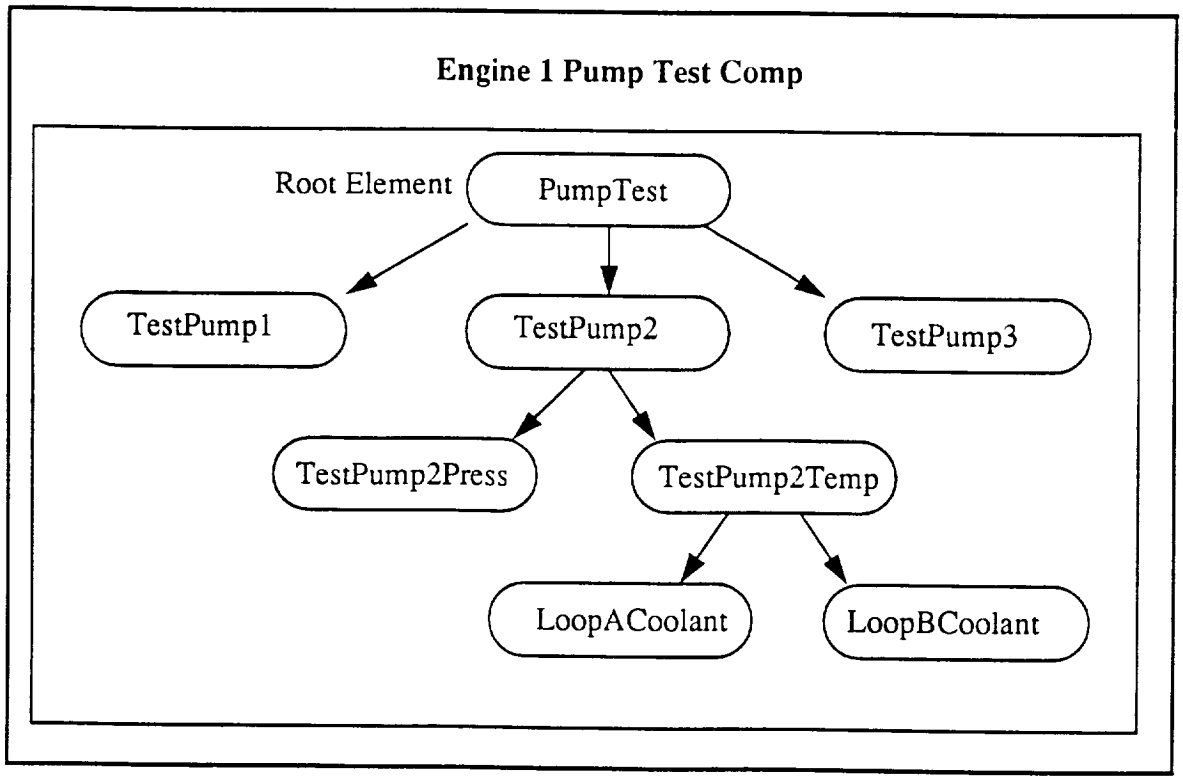
A number of logical Comp algorithms may be necessary for each flight control position and the Graphical Comp Builder Prototype allows the user to maintain a number of Comps for each position. The second level in the Graphical Comp Builder Prototype hiearchy is the Comp. Each Comp may become a stand-alone executable and may be managed by the Comp Manager

application during control center operations. Each Comp is primarily a logical entity and does not perform the actual work of the Comp algorithm. The Comp component of the hiearchy does not contain the graphical symbols or connecting lines.

Each Comp is composed of one or more Comp Elements. The Comp Elements contain the graphical symbols and connecting lines which contain the logic and control flow of the Comp. Comp Elements form the lowest level of the Graphical Comp Builder Prototype hiearchy.

Comp Elements are analogous to subroutines of a main program. Each Comp contains one or more Comp Elements and the Comp serves as a container of its Comp Elements. The Comp is the only entity which may be managed outside the Graphical Comp Builder Prototype. Comp Elements may only be maintained using the Graphical Comp Builder Prototype and serve only as the building blocks from which Comps are composed.

Each Comp contains a root Comp Element. The root Comp Element is the start point of the Comp's Element hiearchy. The root Comp Element may contain the entire Comp algorithm or the root Element may call other Comp Elements. As in traditional programming language subroutines, Comp Elements may transfer program control flow to another Comp Element. When a Comp Element reaches its END symbol, control is returned to the parent Comp Element. In the event the root Element reaches its END symbol, the Comp executable is terminated. Using this concept, the user may build a Comp of practically unlimited depth and breadth by adding calls to other Comp Elements. The user may decompose complex Comp algorithms into logical subunits and then decompose individual logical subunits into even more subunits. The following diagram displays an example Comp which is composed of eight Comp Elements. This example Comp hiearchy displays the flexibility available for decomposing complex Comp algorithms into smaller, modular, more manageable components.



2.3 Report Generation

The Graphical Comp Builder Prototype was designed to assist the Comp algorithm developer by automatically generating supporting documentation. Comp algorithm developers often spend more time producing and maintaining supporting documentation then they do actually developing Comp algorithms. Two features of the Graphical Comp Builder Prototype assist in the development of supporting documentation:

- An important aspect of the Graphical Comp Builder Prototype is to allow the user to maintain supporting documentation in the same tool as the Comp. It is much easier to maintain supporting documentation when it is readily available in the tool used to build the Comp algorithm. The Graphical Comp Builder Prototype allows the user to document Comps at three different levels. The user can document the high level design of the Comp, the high level design of each Element within a Comp, and the user can document each logical expression within a Comp Element. The Graphical Comp Builder Prototype user may also document each Comp Element by placing label text at any location in the Work Area.
- The Graphical Comp Builder Prototype automatically generates a printed report which contains much of the information needed in Comp algorithm supporting documentation. The report generated by the Graphical Comp Builder Prototype contains information used to generate the Comp executables (lists of variables and their data types, etc.) and also contains the supporting documentation entered by the user.

The combination of these two features make the development of supporting documentation much easier for the Comp algorithm designer.

2.4 Automatic Code Generation

The Graphical Comp Builder Prototype's main purpose is to allow the user to specify a fault detection algorithm which can be performed during control center operations. The Graphical Comp Builder Prototype captures information from the user and then generates an executable program which can be run during operations. The executables produced by the Graphical Comp Builder Prototype are built automatically within the Graphical Comp Builder Prototype. The Graphical Comp Builder Prototype performs the following process to produce an executable which can be performed during operations:

- The Graphical Comp Builder Prototype first audits each Comp Element to ensure the user has properly connected the graphical symbols. The Graphical Comp Builder Prototype then audits each graphical symbol to ensure the logical expressions are properly completed.
- If errors are not detected during the audit of the Comp Element, then a C language source file is generated for the Comp Element. The Comp Element's C language source file is then compiled using the resident C compiler. An object file is produced for each Comp Element C language source file.
- Once all of the C language source files for each Comp Element have been generated and successfully compiled, then a C language source file is produced for the Comp. The Comp's C language source file contains initialization code and also contains the call to the root Element. The Comp's C language source is then compiled to produce an object file.

- After the object files are produced for the Comp and for each of the Comp's Element files, then all of the object files are linked together to produce a Comp executable.
- The executable produced from this process can be run during control center operations. The executables produced by the Graphical Comp Builder Prototype are dependent on several other applications in the control center:

  The executables produced by the Graphical Comp Builder Prototype are usually controlled by a master program called the Comp Manager. The Comp Manager application is used to start and stop the Comp executables. The Comp executables are not designed to be run directly from the command line. The Comp Manager is used to control and monitor the execution of the executables produced by the Graphical Comp Builder Prototype.

  The executables produced by the Graphical Comp Builder Prototype display messages on the flight controller's screen to report the status of the system being monitored by the Comp and to report the status of the Comp itself. The Comp designer specifies what types of messages to display and when to display the messages during execution by placing PRINT symbols in the Comp's Elements. The Comp executables do not display these messages on the flight controller's screen directly. The messages are sent to another application program which controls the flight controller's screen.

  The executables produced by the Graphical Comp Builder Prototype retrieve data from the control center's data acquisition system. The Comp executables automatically are built with the proper interfaces to the control center's data acquisition system.

2.5 User Interface

The Graphical Comp Builder Prototype is an X Windows and MOTIF based tool. The Graphical Comp Builder Prototype screen is composed of four main areas:

- MOTIF menu bar
- Comp Element Work Area
- Comp status area
- Graphical symbol palette menu

The MOTIF menu bar contains most of the Graphical Comp Builder Prototype menus. The Comp Element Work Area is a large scrolled window area in which the graphical Comp is constructed. The Comp status area contains various information to identify the current Comp Element which is active in the Work Area and also contains information about the Comp. The graphical symbol palette menu contains bitmapped icons which represent the available Comp Element building blocks. The graphical symbol palette menu is overlaid with the Expression Builder during the completion of IF and SET symbols.

The Graphical Comp Builder Prototype uses popups wherever necessary to input or display information to the user. User input text fields are implemented using the standard MOTIF text fields to provide a user-friendly and consistent interface. Each popup uses a consistent button layout to provide a consistent interface:

- each popup contains a CANCEL button which is located at the bottom left of the popup form.
- each popup contains a HELP button which is located at the bottom right of the popup form.

- most popups contain a DONE button which is located at the bottom of the popup form just to the right of the CANCEL button.

3.0 Graphical Comp Builder Prototype Features

The Graphical Comp Builder Prototype provides a number of features to allow users, such as flight controllers and engineers, to very easily design and document fault detection algorithms. This section details the various features and capabilities of the Graphical Comp Builder Prototype. Section 4.0 of this document discusses the implementation of these features.

3.1 Graphical Representation of Comp Algorithms

The Graphical Comp Builder Prototype allows the user to build and maintain Comp algorithms using graphical symbols much like a flow chart. The flow chart type representation of the Comp algorithm is easier to understand than existing textually based methods. The flow chart type representation of the Comp algorithm is constructed in the Work Area portion of the Graphical Comp Builder Prototype's screen. The Graphical Comp Builder Prototype provides a number of features which allow the user to very easily build and maintain the Comp algorithms in the Work Area.

3.1.1 Work Area

The Work Area has several features to make the building and maintenance of graphical Comps easier. The Work Area is a large, scrolled window. The user may construct graphical Comps which are larger than the size of the Work Area. The Work Area scroll bars allow the user to move within the graphical Comp. The user may also "zoom out" to view the Work Area in a reduced scale view. The reduced scale view allows the user to view the entire Comp algorithm in the Work Area. The user may edit the Comp algorithm in the Work Area while working in the reduced scale view. All functions which are available in the full scale view are also available in the reduced scale, or zoomed view.

The Work Area has two "speed" menus which provide both Work Area specific funtions and general file functions. The Work Area specific functions are only available via the speed menus. The general file functions are available via other menus within the Graphical Comp Builder Prototype.

3.1.2 Add Symbol

The user may add a new graphical symbol to the Comp algorithm in the Work Area. To add a new symbol, the user clicks the left mouse button in one of the symbols in the symbol palette menu. If the symbol to be added is a BEGIN or END symbol, the symbol will appear in the Work Area and the user may place the symbol using the mouse. Otherwise, a popup will be displayed which allows the user to complete the information required for the symbol. Once the information has been entered in the popup, the new symbol will appear in the Work Area and the user may place the symbol using the mouse.

3.1.3 Move Symbol

The user may move a single graphical symbol within the Work Area by placing the mouse pointer in the symbol and then dragging it within the Work Area while holding down the left mouse button. If a symbol contains connecting lines, its connecting lines will be deleted after the symbol is moved. The Graphical Comp Builder Prototype currently does not attempt to maintain line connectivity after a symbol has been moved. This limitation is mitigated by the ease with which the user can connect symbols and by the Move Block function which allows the user to move a collection of symbols while maintaining their line connectivity.

The Move Symbol function can be "undone" by the UNDO function. The UNDO function will restore the symbol to its original location and will restore any connecting lines if any existed at the symbol's original location.

3.1.4 Edit Symbol

The user may edit the contents of any symbol in the Work Area by placing the mouse pointer in the symbol and then clicking the right mouse button. The same popup which allowed the user to originally define the symbol will then allow the user to modify the symbol's contents and attributes. The UNDO function will not recover changes made by the Edit Symbol function.

3.1.5 Symbol Implode

The user may "implode" into CALL and ACTIVATE symbols. The Implode function allows the user to edit the Comp Element specified in the symbol. Implode is a faster method of selecting a Comp Element to edit. The Implode function could be performed by Select Element and then choosing the proper Comp Element. Implode allows the user to point to a symbol and then implode into that symbol. The user may Implode into a symbol by placing the mouse pointer in a CALL or ACTIVATE symbol and then clicking the right mouse button while holding down the shift key.

3.1.6 Delete Symbol

The user may delete a single graphical symbol within the Work Area by placing the mouse pointer in the symbol and then clicking the left mouse button while the shift key is held down. The graphical symbol and its connecting lines are deleted from the Work Area.

The Delete Symbol fucntion can be "undone" by the UNDO function. The UNDO function will restore the symbol to its original location and will restore any connecting lines if any existed at the symbol's original location.

3.1.7 Connect Symbols

The user may logically connect two symbols within the Work Area. To connect two symbols, the user places the mouse pointer within the first symbol, clicks the middle mouse button, then the user moves the mouse out of the symbol in the direction in which to draw the connecting line. Orthogonal anchor points may be placed at any point during the construction of the line by clicking the middle mouse button. To conclude the line, the user places the mouse pointer within the end symbol and clicks the middle mouse button.

Connecting lines may not go thrcugh symbols other than the start or end symbol. Connecting lines have an associated direction and the connecting lines are drawn with arrow heads to indicate the direction.

The Connect Symbols function can be "undone" by the UNDO function. The UNDO function will remove the connecting line which was just added to the Work Area.

3.1.8 Delete Connecting Line

The Delete Line function allows the user to delete an existing logical connecting line. The user may delete a connecting line by placing the mouse pointer on the line and clicking the middle mouse button while the shift key is depressed.

The Delete Line fucntion can be "undone" by the UNDO function. The UNDO function will restore the connecting line which was deleted.

3.1.9 Move Block

The Move Block function allows the user to mark a set of symbols and their connecting lines using a rubber-banded box and to move them via the mouse to another location within the Work Area. The Move Block function is available via the right mouse button speed menu in the Work Area. The user selects the rectangle anchor point by clicking the left mouse button, then the user can expand the rectangle by moving the mouse. The user can anchor the size of the bounding box by clicking the left mouse button a second time. The user will see a rectangular box which indicates the area to be moved and can place this area by moving the mouse. The selected area can then either be anchored in the new location by clicking the left mouse button, or the Move Block function can be aborted by clicking the CANCEL button in the Status area.

The Move Block fucntion can be "undone" by the UNDO function. The UNDO function will move the symbols and their connecting lines to their original position in the Work Area.

3.1.10 Copy Block

The Copy Block function allows the user to mark a block of symbols and their connecting lines using a rubber-banded box and to copy them to a new location in the Work Area. The Copy Block function is available via the right mouse button speed menu in the Work Area. The user selects the Copy Block rectangle anchor point by clicking the left mouse button, then the user can expand the rectangle by moving the mouse. The user can anchor the size of the bounding box by clicking the left mouse button a second time. The user will see a rectangular box which indicates the size of the area to be copied and can place this area by moving the mouse. The selected area can then either be anchored in a new location by clicking the left mouse button, or the Copy Block function can be aborted by clicking the CANCEL button in the Status area.

The Copy Block function can be "undone" by the UNDO function. The UNDO function will remove the new symbols and their connecting lines from the Work Area.

3.1.11 Delete Block

The Delete Block function allows the user to mark a block of symbols and their connecting lines using a rubber-banded box and to delete them. The Delete Block function is available via the right button speed menu in the Work Area. The user selects the rectangle anchor point by clicking the left mouse button, then the user can expand the rectangle by moving the mouse. The user can anchor the size of the bounding box by clicking the left mouse button a second time. The user will see a rectangular box which indicates the area to be deleted. The Delete Block function can be aborted by clicking the CANCEL button in the Status area.

The Delete Block function can be "undone" by the UNDO function. The UNDO function will restore the symbols and their connecting lines.

3.1.12 UNDO and CANCEL

The Comp Status Area contains a dual function push button which may be used to cancel or "undo" functions which are performed within the Graphical Comp Builder Prototype. The push button is a CANCEL button if the user is in the midst of performing any of the above mentioned functions, otherwise, the push button will be an UNDO button. The push button label changes during execution of the Graphical Comp Builder Prototype to indicate which function is active.

The CANCEL function allows the user to cancel the current operation. Almost every multi-step function can be canceled. The following are examples of how the CANCEL button may be used within the Graphical Comp Builder Prototype:

- The user may abort the addition of a new symbol at any point by selecting the CANCEL button. The user may abort during symbol placement and while editing a symbol's information in the symbol specific popup.
- The user may abort the connection of two symbols at any point by selecting the CANCEL button.

Many of the functions performed in the Work Area are recorded so the user can "undo" the function after it is completed. The UNDO function restores the Work Area to the state it was in before the user performed the last function. The effects of an UNDO depend on the function last performed by the user. An UNDO may perform something as simple as moving a single symbol, or it may perform a more complex operation such as copying a group of symbols and their connecting lines. The UNDO function is also available from the left mouse button speed menu.

3.2 Position Management

The Graphical Comp Builder Prototype allows the user to organize Comps into different directories for different flight control positions. This allows the user to maintain Comps for multiple flight control positions on a single workstation or Network File System (NFS) disk. The Graphical Comp Builder Prototype contains several functions to support multiple Positions.

3.2.1 Select Position

The Select Position function allows the user to select an existing Position directory. The Select Position function allows the user to traverse the file system hiearchy in the event the Position directories are not located in a single subdirectory.

3.2.2 Create Position

The Create Position function allows the user to create a new Position directory. The Create Position function prompts the user for the new Position name and then displays the name and path of the directory which will be created.

3.3 Comp Management

The Graphical Comp Builder Prototype allows the user to maintain multiple Comps within a specified flight control position. The Graphical Comp Builder Prototype contains a number of functions to assist the user in the maintenance of Comps.

3.3.1 Select Comp

The Select Comp function allows the user to select from a list of existing Comps within the specified Position.

3.3.2 Create Comp

The Create Comp function allows the user to create a new Comp within the specified Position. The Create Comp function creates a new directory within the Position directory. All of the files for a single Comp are maintained in the Comp directory. The Create Comp function prompts the user for the Comp's purpose and the name of the Comp Element which will be the root Element of the Comp.

3.3.3 Select Comp Root Element

The Select Comp Root Element function allows the user to select the root Comp Element of a Comp's hiearchy. The Select Comp Root Element function allows the user to select the root Element from a list of Comp Elements which comprise the Comp.

3.3.4 Edit Comp Purpose

The Edit Comp Purpose function allows the user to edit and save the Comp Purpose text which is displayed in the Status area of the Graphical Comp Builder Prototype. The Edit Comp Purpose function displays the Comp's current Purpose text, and then allows the user to modify the text.

3.3.5 Display Comp CallFlow

The Display Comp CallFlow function displays the hiearchy of the Comp Element calls within a Comp. The Display Comp CallFlow starts at the root Element of the Comp and recursively traverses the list of Element calls. The hiearchy of Element calls are presented to the user in the Displayer.

3.3.5.1 Displayer

The Displayer is a large, output only window that allows the user to monitor the progress of several operations within the Graphical Comp Builder Prototype. The Displayer allows the user to monitor the progress of operations which may take more than a couple of seconds to complete. The Displayer is used during the following operations:

- The Displayer is used during the Installation of an Element. The Displayer allows the user to view the different stages of Element Installation and also displays status information.
- The Displayer is used during the Installation of a Comp. The Displayer allows the user to view the different stages of Comp Installation and also displays status information.
- The Displayer is used during the Comp Validation process.

The Displayer popup window is not modal. The user may leave the Displayer popup window on the screen even after the operation which caused the Displayer to be displayed has completed. This allows the user to refer to information within the Displayer during the editing of Comp Elements. The output of the Displayer is recorded into a disk file.

3.3.6 Install Comp

The Graphical Comp Builder Prototype's ultimate mission is the generation of a Comp executable which can be managed by the Comp Manager. There are two main steps in the generation of a Comp executable from the Comp entered by the user:

- The graphical Comp is converted into C language source files.
- The C language source files generated by the Graphical Comp Builder Prototype are compiled and linked to produce a Comp executable.

These two steps are implemented via the "Install Element" menu button and the "Install Comp" menu button. C language source files are generated via the "Install Element" menu button and an executable Comp is produced via the "Install Comp" menu button. See the Install Element section on page 16 for more details about the Install Element function.

The Install Comp function first ensures that an object file has been produced for each graphical Comp Element in the Comp. The Install Comp function then generates several C language source

files for the Comp. These source files are compiled, and then the resulting object file and the Comp Element object files are linked together to produce a Comp executable. The results of the Install Comp process are presented to the user in the Displayer.

3.3.7 Validate Comp

The Validate Comp function is designed to verify the data types and sizes of the values retrieved and stored in Object Access match the definitions contained in the Object Access and Work Station Global tables. The Validate Comp function presents its results to the user in the Displayer.

3.3.8 Comp Report Generation

The Graphical Comp Builder Prototype provides several printed reports to document the developed Comps and graphical Comp Elements. The printed reports are available via the Print Comp and Print Element menu buttons. The Print Comp function will generate a multipage printed report which contains the information for the current Comp. The Print Comp report contains the following information:

- Comp name
- Comp purpose
- List of Comp Elements which comprise the Comp
- List of global variables used in the Comp
- An Element level report for each Comp Element in the Comp

An example report generated by the Print Comp function is contained in Appendix A.

The Graphical Comp Builder currently only supports PostScript printers during the generation of printed reports.

3.4 Element Management

The Graphical Comp Builder Prototype allows the user to maintain multiple Comp Elements within a specified Comp. The Graphical Comp Builder Prototype contains a number of functions to assist the user in the maintenance of Comp Elements.

3.4.1 Select Element

The Select Element function allows the user to select from a list of existing Comp Elements for the specified Comp.

3.4.2 Create Element

The Create Element function allows the user to create a new Comp Element within the specified Comp. The Create Element function prompts the user for the new Element's name and purpose text.

3.4.3 Delete Element

The Delete Element function allows the user to delete a Comp Element from the specified Comp. The user may select from a list the Comp Element to delete.

3.4.4 Save Element

The Save Element function will save all information for the current Comp Element to disk. The Save Element function will also update the Comp file and its related information.

3.4.5 Copy Element

The Copy Element function allows the user to copy the Comp Element in the Work Area to a new Comp Element with a different name.

3.4.6 Edit Element Purpose

The Edit Element Purpose function allows the user to edit and save the Comp Element's Purpose text which is displayed in the Status area of the Graphical Comp Builder Prototype. The Edit Element Purpose function displays the Comp Element's current Purpose text, and then allows the user to modify the text.

3.4.7 Print Element

The Print Element function allows the user to print the Comp Element active in the Work Area. The user may print the Comp Element in one of two modes:

- The user may print a Comp Element in the "normal" mode. The normal mode report prints the Comp Element in the Work Area using the same size symbols and fonts as is used to display the Comp Element in the Work Area. The normal mode report requires as many as 4 pages to print the entire Element due to the fact that the Work Area is larger than a single sheet of 8 1/2" by 11" paper.
- The user may also print a Comp Element in "reduced" mode. The reduced mode report prints the entire Comp Element on a single 8 1/2" by 11" page. The reduced mode report uses a smaller symbol size and font to fit the entire Work Area on a single page.

An example report generated by the Print Element function is contained in Appendix B.

3.4.8 Audit Element

The Audit Element function allows the user to verify the Comp Element has been properly constructed and is ready for Installation. The Audit Element function operates on the graphical Comp Element active in the Work Area. The Audit Element function is composed of two functions which perform the following checks:

- The Audit Lines function will check the logical connectivity of each of the graphical symbols in the Work Area. The Audit Lines function will ensure that each symbol has at least one line entering the symbol and at least one line leaving the symbol. In the case of an IF symbol, the Audit Lines function will ensure the IF symbol has both a TRUE and FALSE logical connecting line leaving the symbol.
- The Audit Expressions function will check the Comp Expression of each of the graphical symbols in the Work Area. The Audit Expressions function will ensure the Comp Expression exists and that it is syntactically correct. The Audit Expressions function will parse and type check the Comp Expression to verify it is syntactically and semantically correct.

The Audit Lines and Audit Expressions functions can be performed separately or they may be performed together.

The Audit Element function highlights the symbols which have failed the specified Audit tests. The symbol highlighting may be turned off via the Clear Audit function.

3.4.9 Install Element

The Install Element function generates a MOAL or C language source file for the graphical Comp Element in the Work Area. If a C language source file is generated for the Element, then the C language source file is compiled to produce an object file. If an object file is successfully produced, the Install Element function marks the Comp Element as up-to-date. If the user makes and saves any changes to the graphical Comp Element, the Comp Element is marked as being out-of-date. This feature will ensure the user's object files are consistent with their corresponding graphical Comp Element. The Install Element process is presented to the user in the Displayer.

3.5 Options Management

The Graphical Comp Builder Prototype has several user configurable options. Some of these options may be configured during the execution of the Prototype. The following sections describe the options which are user configurable during the execution of the Graphical Comp Builder Prototype.

3.5.1 Display Options

The Display Options function displays a popup which identifies the different user configurable options and each option's current value. The Display Options popup contains the following information:

- The name of the flight control Position in which the user is working is displayed.
- The name of the current Comp is displayed.
- The name of the Comp Element which is active in the Work Area is displayed.
- The path and filename of the Displayer output file is displayed.
- The path and filename of the Error Log file is displayed.
- The state of the Symbol Snap toggle is displayed.
- The state of the Comp Element Audit toggle is displayed.
- The target language for Installation operations is displayed.
- The path and filename of the Object Access Table is displayed.
- The path and filename of the Work Station Global Table is displayed.
- The path to the User Defined Functions is displayed.
- The user's name, as determined from the /etc/password file, is displayed.
- The current time and date, as determined from the computer's system clock is displayed. The time and date displays are not updated while the Display Options popup is displayed.

3.5.2 Symbol Display

The Graphical Comp Builder Prototype allows the user to control the display of the expressions in the graphical symbols. The Symbol Display function allows the user to specify whether the Comp Expression or Logic Description text should be displayed in the symbols in the Work Area. The Symbol Display function is available in both the Options menu and the right mouse button speed menu.

3.5.3 Symbol Snap

The Graphical Comp Builder Prototype allows the user to place and move the graphical symbols in the Work Area using either a snap grid or free hand. If the symbols are placed or moved while snap is turned on, the symbol will be "snapped" so that the center of the symbol will be on a snap line. The symbols are snapped so that the center of the symbol is on the snap grid. This allows the user to line up the connecting lines which exit symbols.

The user may also place symbols free hand. The Work Area is bordered by a ruler bar which may be used to place symbols or connecting lines. The ruler bar contains a moving pointer which displays the location of the mouse pointer. The ruler bar pointer indicates the center of the symbol during symbol movement functions and indicates the end of the current line segment during connecting line functions.

3.5.4 Audit Toggle

The Audit Toggle controls the operation of the Audit Element function. The user may elect to turn the Audit Element function continuously on or the user may wish to explicitly select the Audit Element function as desired. If the Audit Element function is continuously turned on, the Comp Element in the Work Area will be Audited every time an operation is perfomed in the Work Area.

3.5.5 Set Colors

The Set Colors function allows the user to set the foreground and background colors of the graphical symbols in the Work Area. The Set Colors function affects only the current Comp Element. The specified colors are saved with the Comp Element and are restored every time the Comp Element is read into the Work Area. The Set Colors function allows the user to set the color of individual symbol types, or the user can set all symbol types to the same color. For example, the user can set all BEGIN symbols to a blue background and all END symbols to a yellow background.

3.5.6 Set Target Language

The Graphical Comp Builder Prototype was designed to automatically produce source code for several different languages. The Set Target Language function of the Graphical Comp Builder Prototype allows the user to select which language to use during the automatic generation of the source files:

- The Graphical Comp Builder Prototype can automatically generate C source files.
- The Graphical Comp Builder Prototype can also automatically generate MOAL source files. A compiler currently does not exist for MOAL language files, so a Comp executable can not be produced if MOAL is chosen as the target source language.
- The Graphical Comp Builder Prototype was also initially designed to support UIL source files and the Set Target Language popup contains a UIL option. Due to the unavailability of UIL, the UIL source code generation functions were not completed in the Graphical Comp Builder Prototype. The Set Target Language popup will inform the user that UIL source code generation has not been implemented if the user selects UIL in the Set Target Language popup.

3.6 Help System

The Graphical Comp Builder Prototype contains two levels of online help text to assist the user during execution of the Prototype. The Graphical Comp Builder Prototype allows the user to select Help from the main menu bar. Within the main menu bar help, there are three categories of help text:

- The user can view help text which describes the conventions of the Work Area. This set of help text identifies the different mouse button conventions and their function.
- The user can view help text which describes the graphical symbol palette menu. This set of help text describes the function of the different graphical symbols. The user may use the mouse to select the graphical symbol for which to display help text.
- The user may also browse through the entire collection of help text. All of the help text for the Graphical Comp Builder is maintained in a single disk file. The user may browse through the entire file if desired.

Each of the Graphical Comp Builder Prototype popup windows contains a HELP button in the lower right hand corner of the popup. This HELP button allows the user to view the help text for the current popup window.

4.0 GCB Implementation Notes

This section provides the details concerning the implementation of the Graphical Comp Builder Prototype. The Implementation Notes are contained in three sub-sections:

- Data Files - this sub-section details the various disk files maintained or accessed by the Graphical Comp Builder Prototype.
- Data Structures - this sub-section details the most important data structures which are maintained by the Graphical Comp Builder Prototype.
- Module Hiearchy - this sub-section describes the general layout and program flow of the major modules within the Graphical Comp Builder Prototype.

These sub-sections provide the user with information specific to the implementation of the Graphical Comp Builder Prototype. To gain a thorough understanding of the Graphical Comp Builder's Module Hiearchy and its implementation, it is important to first understand the nature of X Windows and Motif event driven applications. Refer to X Windows documentation for information about event driven applications.

4.1 Data Files

The Graphical Comp Builder uses a number of data files in several different directories during execution. The following subsections describe the purpose of the various data files and also identifies the directories where the Graphical Comp Builder Prototype expects to locate these data files.

4.1.1 Position Directory

The Graphical Comp Builder Prototype allows the user to maintain different flight control positions. Each logical flight control position is maintained in a separate directory. The Position Directories can be identified by their ".POS" extension. The ".POS" extension is searched for by the Select Position functions within the Graphical Comp Builder Prototype. The Create Position functions in the Graphical Comp Builder Prototype will automatically create a Position Directory with the correct extension. The Position Directories may reside at any place within the Unix file system. The Position Directory basename will correspond to the Position name. For example, the Position Directory for the INCO Position would be: INCO.POS.

4.1.2 Comp Directory

The Graphical Comp Builder Prototype creates a subdirectory for each Comp which is created during execution. The Comp Directories are created as subdirectories of the Position Directory. Each Comp Directory must end with a ".DIR" extension. The Select Comp functions within the Graphical Comp Builder Prototype will search for the ".DIR" extension. The Create Comp functions will automatically create Comp Directories with the proper extension. The Comp Directory basename will correspond to the Comp name. For example, the Comp Directory for the PumpSwitch Comp would be: PumpSwitch.DIR.

4.1.3 Comp File

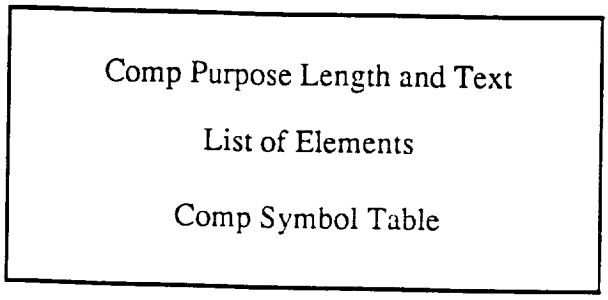
Each Comp Directory will contain a Comp File. The Comp File serves several purposes:

- The Comp File contains the Comp Purpose text.

- The Comp File contains a list of the Elements which are maintained in the corresponding Comp Directory.
- The Comp File contains the Comp's symbol table.

The Comp File can be identified by its ".CMP" file extension. Only one Comp File exists for each Comp and only one Comp File should exists within a Comp Directory. The Comp File basename will correspond to the Comp name. For example, the Comp File name for the PumpSwitch Comp would be: PumpSwitch.CMP.

The Comp File is automatically maintained by the Graphical Comp Builder Prototype and each Comp File is composed of three logical parts in the following order:



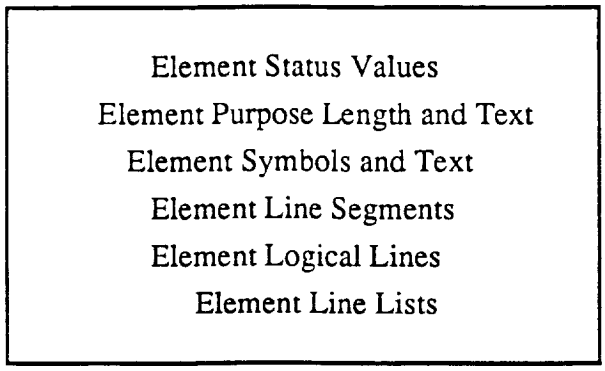
4.1.4 Graphical Element File

Each Comp is composed of one or more Comp Elements. Each Element is maintained in a Graphical Element File. The Graphical Element files serve several purposes:

- The Graphical Element File contains the Element purpose text and several status indicators including: Element Create Date, Element Update Date, and Element Author.
- The Graphical Element File contains the data which identifies the location and type of each of the graphical symbols within the Element.
- The Graphical Element File contains the expressions and supporting text for each of the graphical symbols within the Element.
- The Graphical Element File contains the data which identifies the connecting lines which logically connect the graphical symbols.

The Graphical Element files of a Comp are maintained in the Comp Directory. Graphical Element files may be identified by their ".GEF" extension. The Graphical Element File basename will correspond to the Element name. For example, the Graphical Element File name for the MainPump1 Element would be: MainPump1.GEF.

Each Graphical Element File is automatically maintained by the Graphical Comp Builder Prototype and each file is composed of six logical parts in the following order:



4.1.5 Library Graphical Element Directory

The Graphical Comp Builder Prototype allows the user to maintain a Library of Graphical Element Files. The Library of Graphical Element Files is maintained in a directory specified by the user via the User Configuration File (see the User Configuration File section on page 24 for information on the User Configuration File).

4.1.6 Comp Installation Files

The Graphical Comp Builder Prototype uses a number of files during the generation and linking of the Comp executable. These files have different functions and reside in several different locations.

4.1.6.1 Comp Header File

During the Comp Installation, a header file ("*.h") is created in the Comp Directory containing an "extern" for each of the Comp's global variables. The Comp Header File basename will correspond to the Comp name. For example, the Comp Header File name for the PumpSwitch Comp would be: PumpSwitch.h. The Comp Header File is used during the compilation of the Element C Language Source Files.

4.1.6.2 skeleton_element.o

The skeleton_element.o file is linked into the Comp executable during the Comp Installation process. This file resides in the Graphical Comp Builder executable directory. The skeleton_element.o file contains the object routines which are linked into every executable Comp. The skeleton_element.o file contains the following routines:

- process table initialization and maintenance routines
- data acquisition interface routines
- matrix manipulation routines

4.1.7 Element Installation Files

The Graphical Comp Builder generates a number of files during the Installation of an Element. These files have different functions but all files are created in the Comp Directory.

4.1.7.1 Element C Language Source File

During an Element's installation, a C language or MOAL language source file is generated from the Graphical Element File. The C Language Source File basename will correspond to the Element

name. For example, the C Language Source File name for the MainPump1 Comp Element would be: MainPump1.c

4.1.7.2 Element C Language Object File

During an Elements's installation, a C language object file is generated from the C language source file. This file is generated automatically by the Graphical Comp Builder Prototype during the Installation process. This file is produced by executing the workstation's resident C compiler on the Element's C language source file. The C Language Object File is linked into the Comp Executable which is produced during the Comp Installation process.

4.1.8 PostScript Files

The Graphical Comp Builder Prototype currently supports only PostScript compatible printers. The PostScript print functions access and create several PostScript files during Element printing.

4.1.8.1 PostScript Template File

The Graphical Comp Builder Prototype uses a template file which contains the functions needed to generate a PostScript file during the printing of Elements. This PostScript Template File is named: ps_template and is located in the Graphical Comp Builder Prototype executable's directory. The PostScript Template File is not printed by the Graphical Comp Builder Prototype. It is only used to build the PostScript File which is printed.

4.1.8.2 PostScript File

During the printing of an Element, the PostScript Template File is copied to the Comp Directory. The Comp Directory copy of the PostScript Template file is then modified to include the Element specific information. The modified PostScript file is renamed ps_file when it is copied to the Comp Directory. The PostScript File in the Comp Directory is the file which is printed during report generation.

4.1.9 Help Text File

The Graphical Comp Builder Prototype help system extracts help text from a disk file called GCBDoc during execution. This disk file is located in the same directory as the Graphical Comp Builder Prototype executable. This disk file is a standard ASCII file and may be modified with an editor. The GCBDoc help text file contains keywords which are used by the Graphical Comp Builder Prototype to locate the desired section of help text. The keywords in the GCBDoc help text file correspond to the keywords in the source file: tokens.h. Any change to the GCBDoc help text keywords or to the list of keywords in the file tokens.h, must be updated in both locations or the help text may not be selected properly during execution. The keywords in the GCBDoc help text file are identified by the asterisk in column 1. The GCBDoc help text file should not contain tab characters.

4.1.10 Error Log File

The Graphical Comp Builder Prototype will generate an Error Log File if errors are detected which should not occur during normal operation of the Graphical Comp Builder Prototype. The Error Log File is an ASCII file which contains the date and time the error occurred and a short description of the error condition. The Graphical Comp Builder Prototype will append new error messages to the end of the Error Log File as the errors are detected. The user may specify the location of the Error

Log File via the User Configuration File (see the User Configuration File section on page 24 for more information about the User Configuration File).

4.1.11 User Configuration File

The Graphical Comp Builder Prototype contains a number of features that are user configurable during execution. The state or value of the various options will be automatically saved to a user specific configuration file when the user exits the Graphical Comp Builder Prototype. The next time the user executes the Graphical Comp Builder Prototype, the user's defaults will be restored to the state or value which the user last selected.

The User Configuration File is saved as an ASCII text file. Each option is saved on a separate line in the configuration file. Each line contains an option name followed by a corresponding value. The Graphical Comp Builder Prototype will search several directories at the start of execution to locate the User Configuration File. The following directories will be searched in the following order to locate the User Configuration File:

- the current directory
- the user's home directory

The name of the User Configuration File is always: .Defaults.GCB The following options and corresponding values are stored in the User Configuration File:

- The directory containing the Library Element GEF files.
- The type of Element which was last edited by the user. This option may have one of two values: ELEMENT or LIBRARY_ELEMENT.
- The path and name of the disk file in which to write the Displayer's output.
- The name of the Comp last edited by the user.
- The name of the Element last edited by the user.
- The path and name of the Error Log file.
- The level of error log reporting. This option may be one of the following: "1", "2", or "3". Error log level "3" is used to specify the most verbose error reporting. Error log level "1" is the default setting and should be used during normal operations.
- The path and name of the Object Access table.
- The path and name of the last Position in which the user was working.
- The name of the last Position in which the user was working.
- The target language for code generation. This option may be one of two values: MOAL or C.
- The path and name of the directory containing the User Defined Functions object files.
- The path and name of the Work Station Global table.
- The state of the Display Symbol toggle. This option may be one of two values: "1", or "0". A value of "1" indicates the Logic Description text should be displayed in the graphical symbols. A value of "0" indicates the Comp Expression text should be displayed in the graphical symbols.

Directory path values which are saved to the User Configuration File should not end in a terminating backslash ("/").

4.1.12 Displayer Output File

The Graphical Comp Builder Prototype uses a standard "Displayer" to allow the user to view several operations during execution. See the Displayer section on page 14 for more information on the function of the Displayer. Each time the Displayer is presented to the user, a copy of the text which is displayed on the screen is also written to a disk file. This file may be printed, copied, or edited by the user. This file is created each time the Displayer is presented to the user, and the previous copy of the file is deleted. The user may specify the location and name of the Displayer Output File via the User Configuration File.

4.1.13 User Defined Functions Directory and Files

The Graphical Comp Builder Prototype allows the user to make calls to C language object files which were created outside of the Graphical Comp Builder Prototype. These User Defined Functions are located in a directory specified by the user via the User Configuration File. The User Defined Function names must begin with "FN_". The Graphical Comp Builder Prototype will display to the user a list of the C language object files which begin with the proper format during the construction of IF and SET symbol expressions. The Comp Installation process will locate the User Defined Function object files during the linking of the Comp executable.

4.1.14 Object Access Table

The Graphical Comp Builder Prototype will access an Object Access Table during the Comp Validation process. The Object Access Table should contain a list of available Objects and their associated data types.

4.1.15 Work Station Global Table

The Graphical Comp Builder Prototype will access a Work Station Global Table during the Comp Validation process. The Work Station Global Table should contain a list of available Work Station Globals and their associated data types.

4.2 Data Structures

The Graphical Comp Builder Prototype maintains a number of different data structures during execution. The following subsections will describe the more important data structures maintained by the Graphical Comp Builder Prototype.

4.2.1 Symbol Array

Each graphical symbol of an Element is maintained in the Symbol Array while an Element is active in the Work Area. The Symbol Array is a fixed length array of Symbol structures. The Symbol structure and Symbol Array are defined in gcb.h. All the information for a graphical symbol is available via fields or pointers contained in the Symbol structure, including the following:

- The expressions contained in each graphical symbol are available via pointers in the Symbol structure. The data space for the expressions is malloc()'ed as needed. The Symbol structure does not contain any data space for the expressions within the Symbol structure.
- Pointers to the connecting lines which enter and exit the Symbol are contained in the Symbol structure. The line information is not contained within the Symbol structure. The Symbol structure contains only pointers to Line Lists and Line structures which are maintained separate from the Symbol structure.
- The coordinates relative to the Work Area and dimensions of the Symbol are contained in the Symbol structure.

Almost all of the functions which manipulate the graphical symbols in the Work Area maintain the Symbol Array of structures due to the fact that almost every aspect of each graphical symbol is specified in the Symbol structure.

4.2.2 Cell Map

The Graphical Comp Builder Prototype utilizes a logical grid of cells as an efficient and powerful method of maintaining the Work Area. See the Graphical Symbol Placement Model section on page 4 for more information about the Cell Map concept.

The Work Area Cell Map is implemented as a two dimension array of Cell Structures. Each Cell Structure contains a type flag and a pointer. The type flag indicates if the Cell is occupied and may have one of three values: Symbol Cell, Line Cell, or vacant. If the Cell is not vacant, the pointer will point to the graphical symbol structure or line structure which occupies the cell. The Cell Structure definition and the double dimensioned array of Cell Structures is contained in gcb.h.

4.2.3 Line Structures

The Graphical Comp Builder Prototype allows the user to logically connect the graphical symbols in the Work Area. The Element builder determines the program flow of the Element by the manner in which the graphical symbols are connected. A collection of structures are maintained by the Graphical Comp Builder Prototype to record the logical lines which connect symbols. There are three main Line Structures which are maintained by the Graphical Comp Builder Prototype software:

- The LineSeg structure is the most basic element of the connecting lines between symbols. A LineSeg structure maintains the information for a single line segment from one point to another point. A logical connecting line between two graphical symbols may be composed of multiple line segments. Each orthogonal change in direction starts another line segment.

Each LineSeg structure contains a pointer to the next segment in the logical line if one exists. Each LineSeg structure contains the information needed to draw the line, including the information to draw the line segment's arrow head if one is at the end of the line segment.

- The Line structure represents a logical connecting line between two graphical symbols. Each Line structure contains a pointer to the first line segment of the logical line. The remaining segments of the logical line are available via a pointer in the LineSeg structure which points to the next line segment in the logical line. Each Line structure also contains a pointer to the two graphical symbol entries in the Symbol Array which the line connects.
- The LineList structure represents a list of separate logical lines. The LineList structure is the highest level line structure. Each LineList structure contains a linked list of pointers to logical Line structures. LineList structures are used to record the list of lines which enter a graphical symbol. Each symbol may have only one or two lines which exit the symbol, but many logical lines may enter a symbol, and the LineList structure is used to record each of these lines.

Each of the three Line Structures also includes a key field. The key field is used to record the line information of an Element File to disk and to restore an Element's lines during the reading of an Element File from disk. The key field of each structure is set to a unique number before an Element File is written to disk. The unique numbers in the key fields are then used to reconstruct the interwoven network of pointers during the reading of an Element File from disk.

The various Line Structures are dynamically allocated as needed to maintain the Element in the Work Area. The Line Structures are defined in gcb.h.

4.2.4 Symbol Table

The Graphical Comp Builder Prototype maintains a Symbol Table of the identifiers and variables which comprise the expressions within the graphical symbols of a Comp. The following entities are maintained in the Symbol Table for each Comp:

- The name of each Element which comprises the Comp is maintained in the Symbol Table.
- The name of each local variable within an Element is maintained in the Symbol Table.
- The name of each global variable within a Comp is maintained in the Symbol Table.
- The name of each Object and each Work Station Parameter is maintained in the Symbol Table.
- The name of each User Defined Function and the name of each intrinsic function (cos, tan, sqrt, etc.) is maintained in the Symbol Table. The Symbol Table contains an entry for every intrinsic function, even if it is not referenced in the Comp. Only the names of the User Defined Functions which are referenced within a Comp are maintained in the Symbol Table.

The Symbol Table contains the following information about each entry:

- The name of each symbol is maintained in the Symbol Table.
- The use count of each symbol is maintained within the Symbol Table. The use count is a count of the number of times each variable or Element name is referenced within all of the Elements of the Comp.

- The attributes of each symbol are maintained within the Symbol Table. The attributes field is implemented as a collection of bit masks. The bits in the attributes field of a Symbol Table entry indicate various information including: variable data type and variable scope. The Installation status of Elements is also maintained in the attributes field of the Symbol Table entry.
- The number of rows and columns of non-scalar variables is also maintained in the Symbol Table.

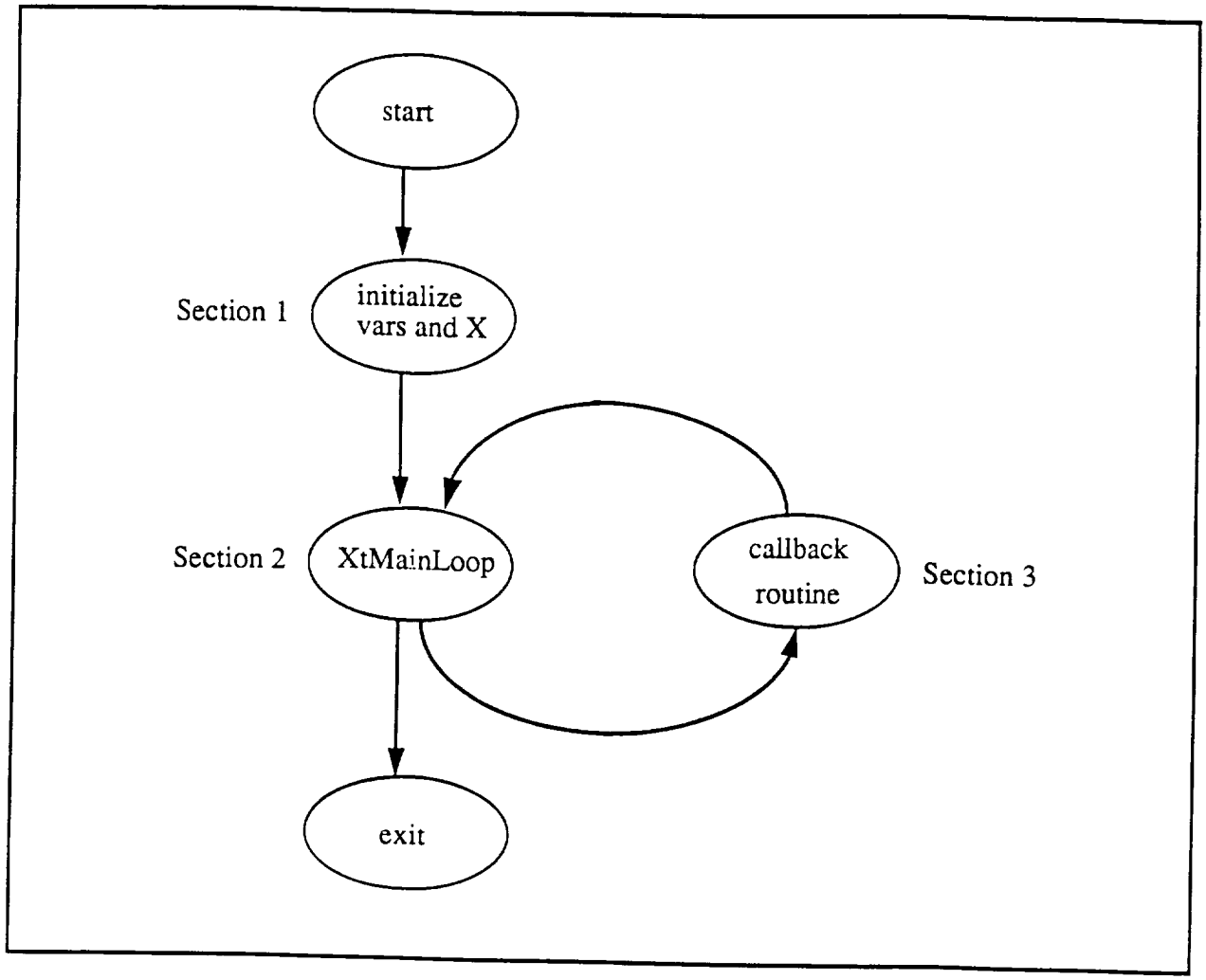
A list of local variables is maintained for each Element name. The local variables of an Element are maintained as a list of children of the Element. In this way, the local variables of an Element are tied to the Element. Different Elements within a Comp may have local variables of the same name. Each time a local variable is accessed in the Symbol Table, the name of the Element in which the local variable is defined is supplied to ensure the correct local variable is accessed.

Global variables are maintained in the root of the Symbol Table and have no parent due to the fact that global variables can be accessed from any Element within a Comp.

The Comp Symbol Table is implemented as a linked list of structures. The Symbol Table Structure definition is contained in symbol.h.

4.3 GCB Program Flow

A high level program flow diagram for the Graphical Comp Builder Prototype is contained in the figure below:



At this level, the program flow appears to be very simple due to the program structure that is typical of X Windows and Motif based applications. There are only three main areas of program flow within most X Windows programs and the Graphical Comp Builder Prototype is no exception. The three main areas of program flow and their function are:

- The first section of a typical X Windows program contains the establishment of the connection to the X Windows server and the definition of the X-based user interface.
- The second section of a typical X Windows program is the main X Windows event loop. This code is usually linked into the application and is not written as part of the application. The X Windows event loop sends and receives events to and from the X Windows server. The X Windows event loop is called at the start of the application and remains in control of the application until the program is terminated. The X Windows event loop will call various callback routines based on the events which occur in the X server.

- The third section of a typical X Windows program contains the callback routines. This section of code is where most application specific processing is performed. The majority of the Graphical Comp Builder specific software is contained within callback routines or is called by the callback routines.

Each X Windows program usually contains these three sections of code. The relative size and complexity of these three sections varies between applications. In the case of the Graphical Comp Builder Prototype, the callback routines section of the application comprises approximately 90% of the software in the Prototype. The following two sections will describe in greater detail the first and third sections (initialization and callback routines). The second section is generic to all X Windows applications and is not specific to Graphical Comp Builder Prototype. The reader is referred to the X Windows documentation for more information about the second section.

4.3.1 Section 1 - Initialization

The Initialization Section of the Graphical Comp Builder Prototype comprises approximately 10% of the code in the Prototype. The initialization routines perform the following functions in the following order:

- The Initialization routines contained in gcb.c setup a collection of signal handlers to trap desired Unix OS events. The Control-C signal is an example of the signals which are trapped.
- The Initialization routines contained in gcb.c establish a connection to the X server and open the X Display.
- The Initialization routines contained in init_X.c construct the X Windows interface. The Graphical Comp Builder Prototype builds all main windows and popup windows during initialization.
- The Initialize routines contained in init_vars.c and utils.c initialize all global variables, including the graphical symbol array and the Work Area cell map. The Initialize routines also read the user's Configuration File and set the variables identified in the User Configuration File to the specified values.

4.3.2 Section 3 - Callback Routines

The X Windows callback routines and the routines called by the callback routines, comprise about 90% of the code in the Graphical Comp Builder Prototype. Due to the event driven nature of an X Windows based application, these callback routines are only called as the result of a user's action within the X Windows interface. Almost every trace of a sequence of events within the Graphical Comp Builder Prototype begins in the source file: init_X.c. It is within this file that the majority of the interface is defined and the event handlers installed.

The following is an example of the sequence of events and the typical program flow that occurs during most operations performed by the user within the Graphical Comp Builder Prototype. The following example shows the sequence of events and the program flow that occurs when the user selects and reads in an existing Element file.

Event	Resulting Program Flow	Source File
User selects the Element menu.	The Element pulldown menu is displayed. This menu was defined during the building of the user interface.	init_X.c
User selects the "Select Element" menu button.	The callback routine cbr_elem_popup() was installed during the building of the user interface for this event and is called.	init_X.c
	The callback routine cbr_elem_popup() makes a call to load_element_list() to load the Element selection list of the "Select Element" popup before the popup is displayed.	element_file.c
	The load_element_list() routine makes a call to read_comp_list() to read the list of Element names from the Comp file.	comp_file.c
	The callback routine cbr_elem_popup() makes a call to display the "Select Element" popup to the user.	element_file.c
	The "Select Element" popup is displayed to the user. The "Select Element" popup was defined during the building of the user interface in init_X.c. A call was made from init_X.c to build_sel_elem_popup() in element_file.c to build the "Select Element" popup during the initialization of the user interface.	popup built in element_file.c
	The callback routine for the Element selection list is called. The callback routine cbr_el_selected() was installed when the "Select Element" popup was built in build_sel_elem_popup().	element_file.c
User selects an Element name from the list.	The callback routine cbr_el_selected() determines the name of the Element the user wants to read and then makes a call to read_element_file() to read the specified Element file from disk into the Graphical Comp Builder Prototype. Once the Element file has been read from disk, control returns to the X Windows main event loop and the cycle begins again.	element_file.

The preceding scenario is typical of each operation the user performs within the Graphical Comp Builder Prototype. For each operation the main program flow is basically the same.

Appendix A

Extract

This comp is performed during the extraction of the HAB satellite from the payload bar. This comp will monitor the status of the RMS pumps to make sure they remain within nominal and critical limits.

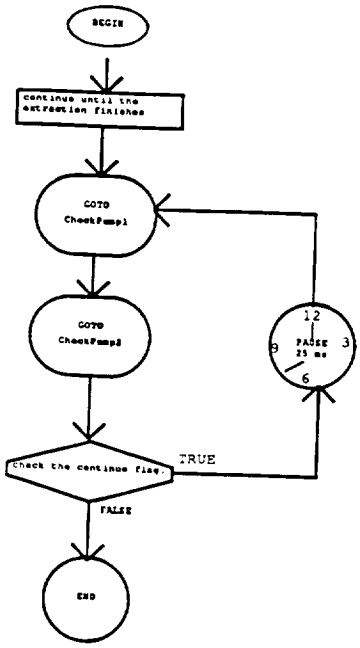
Elements	Installed
CheckPump1	yes
CheckPump2	no

Global Variables	Type	Dimensions	Use Count
V873457E	int	1	2
GV_cont	int	1	2

Position: RMS
Comp: Extract
Element Name: RootElement
Element Type: ELEMENT
Author: Timothy J. Barton
Created: 08/25/1991
Last Update: 08/27/1991
Status: Complete

Purpose:

This element sets up a loop which tests the two main pumps which drive the RMS. The loop continue flag is not altered within this Comp. The user can control the Comp from the Comp Manager.



continue until the
extraction finishes

Logical Description

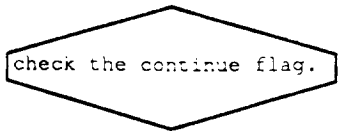
continue until the
extraction finishes

Comp Expression

GV_cont := 1

Comment

Initialize the loop control variable to be TRUE. This loop should run forever. This comp is stopped by the user via the Comp Manager once the extraction has been completed.



gical
scription

check the continue flag.

mp
xpression

GV_cont > 0

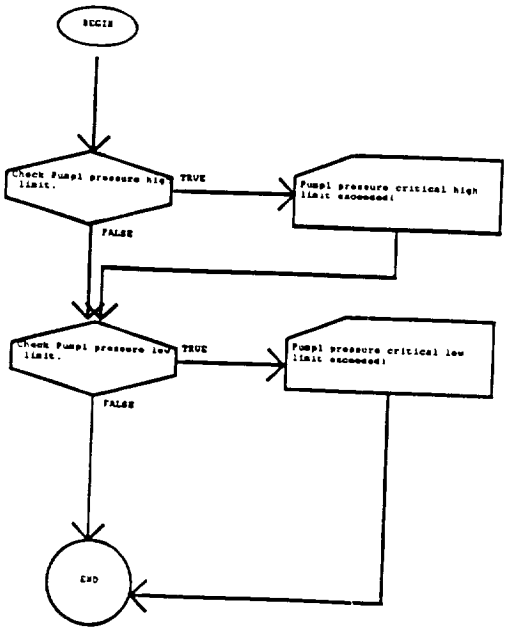
mment

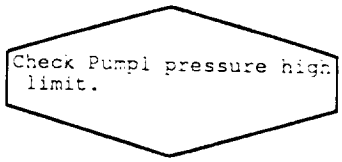
Check to make sure we should 'go around' and perform the tests again.

Position: RMS
Comp: Extract
Element Name: CheckPump1
Element Type: ELEMENT
Author: Jerry Ratner
Created: 08/27/1991
Last Update: 08/27/1991
Status: Complete

Purpose:

This element checks the Pump1 pressures. This element checks the critical high and critical low limits. This element is set up to check for Rev. 2 type pumps





Logical Description

Check Pump1 pressure high limit.

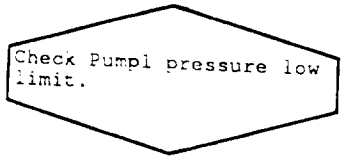
Comp F ression

V873457E > 125

Comment

Check to see if the Pump1 pressure has exceeded the critical high limit. The high limit for the Rev. 2 pumps used on Columbia and Discovery is 125 psi.

The Rev. 1 pumps have critical high limit of 115 psi.



Logical Description

Check Pump1 pressure low limit.

Comp Expression

V873457E < 40

Comment

Check to see if the Pump1 pressure has gone below the critical low limit. The low limit for the Rev. 2 pumps used on Columbia and Discovery is 40 psi.

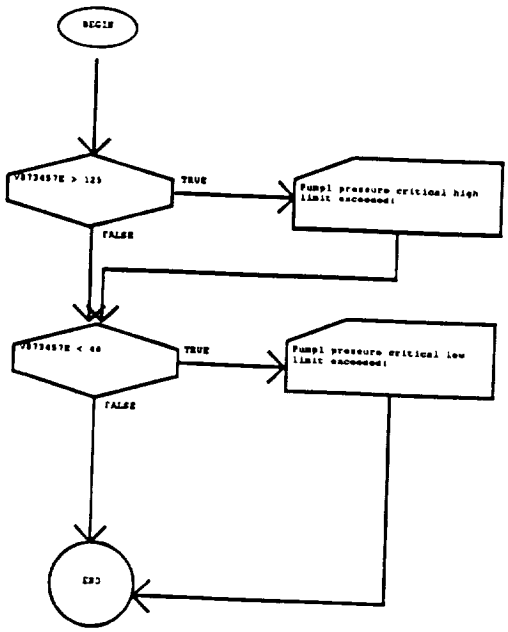
The Rev. 1 pumps have critical low limit of 60 psi.

Appendix B

Position: RMS
Comp: Extract
Element Name: CheckPump1
Element Type: ELEMENT
Author: Jerry Ratner
Created: 08/27/1991
Last Update: 08/27/1991
Status: Complete

Purpose:

This element checks the Pump1 pressures. This element checks the critical high and critical low limits. This element is set up to check for Rev. 2 type pumps



SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

ADA INVESTIGATION

NASA Grant No. NAG 9-435
SwRI Project No. 05-3531

Prepared by:
Timothy J. Barton

Prepared for:
NASA
Johnson Space Center
Houston TX 77058

September 5, 1991

1.0 Investigation of Ada for Control Center Software

Investigation has been conducted to determine the availability and suitability of the Ada programming language for the development of future control center type software. The Space Station Freedom Project (SSFP) has identified Ada as the desired programming language for the development of Space Station Control Center (SSCC) software systems. The Department of Defense (DoD) has mandated Ada as the programming language for all new DoD software. Due to these mandates and related directions within private industry, an investigation of Ada was necessary.

1.1 Ada User's Symposium

The first step in the investigation into Ada was to determine if the Ada programming language was receiving acceptance within NASA. The Ada programming language had a well publicized birth and was purported to be the High Order Language (HOL) of the 1980s and 1990s. During the last several years, Ada has received less publicity. It was important to determine if Ada was slipping into obscurity or quietly gaining acceptance, before significant effort was expended using Ada. The first step was to determine if it was still appropriate to develop an Ada version of the Graphical Comp Builder.

The NASA Ada User's Symposium was the ideal event to determine NASA's commitment to Ada. The presenters' experiences with current Ada compilers also provided data on the ability of Ada compilers to produce a Graphical Comp Builder Prototype with acceptable performance.

The Third Annual NASA Ada User's Symposium was held at NASA-JSC's Gilruth Center on November 6, 1990. The Ada Symposium directly answered the two main questions about Ada:

- Is Ada gaining acceptance?
- Will Ada executable's performance be acceptable?

The first question was answered immediately. The first hour and a half was a discussion about the Ada projects at the various NASA centers. Representatives from Goddard, JPL, Langley, JSC, and LeRC (Lewis Research Center) discussed the various projects using Ada and the Ada development labs at their respective center. A short summary of these projects is included.

The second question was answered through the course of the entire day. The symposium was not just a group of Ada fanatics who are oblivious to the merits of C and FORTRAN and who think that Ada should be used in every piece of software that is written. Almost all of the presenters discussed the development and performance impacts caused by using Ada in their project instead of FORTRAN. In most cases, a performance comparison table to FORTRAN was presented which clearly indicated the performance of the Ada version. The audience and the presenters were very objective and very honest in their discussions about Ada.

1.1.1 NASA Center Status Reports

The first part of the symposium was a status report from the following NASA centers:

- Johnson Space Center
- Goddard Space Flight Center
- Langley Research Center
- Lewis Research Center
- Jet Propulsion Laboratory

- JSC

A short description of the projects at each center is included.

1.1.1.1 JSC

Most of the Ada work being done at JSC is in the SSCC and SSF Training Facility. The SSE which is being done by Lockeed is all in Ada and they are projecting that 1 million lines of Ada will be written before SSE is completed. OADP and OPAS are two projects within the SSCC that are using Ada. Some other projects at JSC are:

- STA- Shuttle Training Aircraft
- COMPASS - Computer Aided Scheduler
- SMSS - F16 Stores Management System Simulator
- JAEL - JSC Avionics Engineering Lab

The JSC campus HVAC system is computer controlled using Ada.

1.1.1.2 GSFC

The Flight Dynamics Division at Goddard is using Ada very extensively. In 1984 less than 1% of all software developed at Goddard was in Ada. They estimate that more than 10% of all software developed in 1990 at Goddard will be in Ada. Goddard is probably the most committed of the NASA centers to Ada and they have extensive statistics to prove their move to Ada. The big projects at Goddard are:

- FTS - Flight Telerobotic Servicer - all software in Ada
- STGT - Second TDRS Ground Terminal - all software in Ada
- HST - Hubble Space Telescope interface simulator
- EUVE - Extreme Ultraviolet Explorer - spacecraft flight software

1.1.1.3 LaRC

Some of the projects at Langley are:

- CSI- Control Structures Interaction - shuttle bay instrumentation platform
- AFE- Aeroassist Flight Experiment

Langley is very big on Ada because they usually have very short turn-around times for their projects and they have found that Ada is faster to develop in than FORTRAN due to high reuse of software. It takes a software engineer longer to write an Ada module than a FORTRAN module, but the Ada module is often reused on other projects, whereas the FORTRAN modules are often rewritten due to the tighter coupling of the module to the particular program.

1.1.1.4 LeRC

The Electrical Systems Division of Lewis Research Center is building the Power Management and Distribution controller for SSFP. All the code in the Power Management system is in Ada and they are very pleased. As an example, the power on the SSF recently changed from AC to DC, and the Lewis software developers were very pleased with the maintainability of their Ada software.

1.1.1.5 JPL

JPL is one NASA center where very little Ada work is going on. JPL is not using Ada due to several factors. The staff at JPL feel there are not enough people with Ada experience so they see Ada as a risk item. JPL probes have had great success and they are afraid to abandon FORTRAN and assembler. The developers at JPL view Ada as big and slow, and their probes are small and have limited resources.

1.1.2 Symposium Speakers

The rest of the day, was devoted to speakers from the various NASA centers discussing their particular project in detail. A summary of some of the projects that were discussed is included.

1.1.2.1 STGT - Second TDRS Ground Terminal

This is a very large real-time project (12 VAX 63XX and 28 workstations) being done by General Electric (GE). They estimate that 490,000 lines of Ada will be written before STGT is completed. GE is very impressed with Ada's ability to aid in the management and development of a large software project. STGT is being completed ahead of schedule and it works. One of Ada's strengths in GE's opinion, is the reusability of Ada code. During development of one portion of STGT, 30,000 lines out of 90,000 total lines of Ada were reused from another portion of STGT. For one of the simulators, 16 were developed in all, Ada required 5,000 lines of code whereas 40,000 lines of FORTRAN were needed to write the same exact simulator.

1.1.2.2 STA - Shuttle Training Aircraft

NASA-JSC has a Gulfstream airplane which can be computer controlled to fly like the shuttle below 35,000 feet. STA is used by pilots to practice shuttle landings and descent. STA uses Sperry and Motorola processors in its Guidance Control Computer (GCC) to control the wing surfaces and engines to enable the Gulfstream to exhibit shuttle-like flight characteristics. The STA was originally programmed completely in assembly language. Most of the software has been converted to Ada, and soon all of the code will have been converted to Ada. This project has extensive timing data comparing FORTRAN to Ada. Not surprisingly, FORTRAN is faster than Ada, but only about 15% faster. The group which is responsible for STA feel that their Ada programs are definitely much easier to develop and maintain. Their theory is: software is expensive, hardware is cheap, buy fast enough hardware to support the proper software development environment (Ada) and you'll save money. The STA group uses Ready Systems ARTX Real-Time Executive and extensions to Ada.

1.1.2.3 JAEL

The JSC Avionics Engineering Lab (JAEL) is responsible for testing the shuttle's avionics hardware:

- GPC- General Purpose Computers
- MDM- Multiplexor Demultiplexor
- MTU- Master Time Unit
- MCDS- Multi-purpose Display System

Even though this group is shuttle related and has no Ada mandate, they now use Ada for all their development because they have had "good past experiences" with Ada. They feel Ada has helped them deliver their systems on-time as opposed to FORTRAN.

1.1.3 General Highlights

The following are some short excerpts from the symposium:

- The SSFP has been losing a lot of money for various projects through "scrubs" lately, but Ada and projects using Ada have fared very well.
- Interestingly, Ada was always compared to FORTRAN during the symposium. The C language was only mentioned once or twice.
- Many of the embedded system projects utilize the military's 1750A microprocessor. There were several very positive comments regarding the Ada development environments available for the 1750A microprocessor.
- Many of the real-time projects used run-time environments to improve real-time performance and these environments provide additional interprocess communication mechanisms other than the Ada rendezvous.

1.1.4 Non-NASA Highlights

The following are several short excerpts from the symposium regarding projects outside of NASA:

- There are over 500 validated Ada compilers.
- The B2 (stealth bomber) trainer was developed entirely in Ada by Link, and the Boeing 747-400 contains approximately 500,000 lines of Ada in its fly-by-wire control systems.
- Much of the software in the Sea Wolf ASW missile was written in Ada.
- Volvo has completed several projects using Ada including all of the software to control their assembly plant Automatic Guided Vehicles (AGV).
- On Monday, November 5, 1990, Congress signed legislation which states that the Secretary of Defense must now sign waivers if Ada is not going to be used on a military project. Waivers will only be granted if it can be demonstrated that another language will be more cost effective over the entire life-cycle of the project. Ada has done very well in the past in software life-cycle cost studies, and the new legislation may increase the number of DOD projects which use Ada.

1.2 Hartstone Benchmark

The second step in the investigation of Ada focused on the relative performance of Ada programs on two workstations often used in control center environments. The Graphical Comp Builder Prototype is a very user interactive program. The ability of the program to respond quickly to user actions is paramount to the program's acceptance by users. The performance of the Graphical Comp Builder Prototype is most critical during graphical symbol placement in the large work area. The C language version of the Graphical Comp Builder Prototype has been designed and implemented to ensure smooth response to mouse movements in the work area. An Ada version of the Graphical Comp Builder Prototype will have to perform symbol placement operations with equally acceptable performance.

The Hartstone Benchmark is being utilized to provide information regarding the performance of the executables produced by various Ada compilers. The Hartstone (Hard Real Time) Benchmark is an Ada program designed by the Software Engineering Institute (SEI) for the USAF which is used to measure a computer system's ability to support a collection of real-time tasks. The

Hartstone Benchmark is primarily intended to provide information about a system's ability to perform real-time tasks and about the performance of the executables produced by different compilers for the same system. The Hartstone Benchmark can also provide information regarding the ability of two different systems (for example, a Masscomp and a Sun) to perform the same system load. To date, the latter type of information has been gathered due to the availability of only one Ada compiler for either the Sun or Masscomp. Should a second Ada compiler become available for either the Sun or Masscomp, then comparison data of executables on the same system will be provided.

1.2.1 Hartstone Benchmark Design

The Hartstone Benchmark is comprised of a series of 4 experiments which are designed to test a different aspect of a real-time system. Each experiment utilizes a minimum of 5 Ada tasks which must perform a specified amount of computation within a given time period. The priority and time period of each of the 5 tasks and the number of tasks are varied in the different experiments to measure a different aspect of system performance. Each of the experiments uses a variant of the Whetstone benchmark to produce and measure system load during the tasks time period. The 4 experiments which comprise the benchmark are numbered 1-4 and they are designed to test the following aspect of the system:

- Experiment 1 gradually increases the frequency of the highest priority task. Experiment 1 is an indicator of a system's task switch overhead. As the time period of the high priority task is decreased, the amount of task switches increases and consequently the task switch overhead.
- Experiment 2 gradually increases the frequency of all 5 tasks uniformly. Each of the 5 tasks must complete a set amount of computational work in an ever decreasing time period. Experiment 2 is an indicator of a system's ability to task switch a collection of tasks.
- Experiment 3 gradually increases the computational work (number of Whetstones) which must be completed within a set time period. Experiment 3 is an indicator of a system's compute performance for a set time period.
- Experiment 4 gradually adds new tasks to the system. Experiment 4 is an indicator of a system's ability to support an increasing number of tasks.

Each experiment is terminated when any of the tasks begin to miss their deadlines (allotted time period) for completing their assigned computational work load. In each Experiment, either the frequency or computational work load of the tasks is increased until a deadline is missed.

1.2.2 Sun vs. Masscomp Comparison

NASA-JSC control center operations and development environments are dominated by either Concurrent Computer Corporation (Masscomp) or Sun Microsystems workstations. The Hartstone Benchmark suite was tested against these two machines to provide information towards whether either of the Ada environments on these machines was suitable for implementation of an Ada version of the Graphical Comp Builder Prototype. The X Windows bindings which are needed to implement the Graphical Comp Builder Prototype are not currently available to the research team. The Hartstone Benchmark is being used to determine on a general scale the performance characteristics of Ada executables on a Masscomp and a Sun.

The Hartstone Benchmark was executed on a Masscomp 6350 and a Sun 4/65 (SPARC Station 1+). Both machines are mid-level Unix workstations from the product lines of their respective manufacturers. The two workstations which were benchmarked share many of the same features:

- Unix-based workstation
- 16 megabytes of memory
- X Windows support
- SCSI peripherals (hard drive, tape drive, floppy drive)
- networked via ethernet (NFS and NIS client)

The two workstations are also very different in some respects. Due to the nature of the real-time Hartstone Benchmark, the Real-Time Unix (RTU) of the Masscomp would typically be a benefit when compared to the Sun's version of Unix (SunOS). The Hartstone Ada source code has been written with portability as a prime objective so the RTU real-time directives available to an Ada program were not employed. Only the real-time features implicitly utilized in RTU were employed. The following are some of the other differences between the two workstations:

- RISC based Sun (SPARC) vs. CISC based Masscomp (68030)
- desktop Sun (single system board employing high integration, i.e. memory) vs. deskside Masscomp (multiboard system, i.e. separate memory and graphics boards)
- Verdix Sun Ada compiler vs. Masscomp C3Ada compiler

The resulting benchmark results have been interpreted in a very general fashion to make allowances for the differences in the two workstation's hardware and operating system software. The results given in this report will be general statements based on the interpretation of the average results of the benchmark. The authors of the Hartstone Benchmark are quick to point out the fact that a large number of independent variables affect the results of the benchmark, and that the data produced by the benchmark should be used to demonstrate the variations that are possible. The specific Hartstone Benchmark results should only be directly compared in situations where most independent variables can be controlled. For this report, a direct comparison is not feasible. For both workstations, the results could be either improved or degraded with optimization and tuning for the particular workstation, use of the RTU real-time features within the C3Ada compiler are an example.

1.2.3 Hartstone Benchmark Results

The complete reports produced by the Hartstone Benchmark for both the Sun and the Masscomp are included in Appendix E of this report and a summary of the results is included in Figures 1, 2, 3 and 4. Where possible, the machines were configured as similarly as possible. Each machine was configured as follows during the execution of the benchmark:

- multiuser mode
- benchmark was executed as "root"
- disconnected from ethernet

A single copy of the Hartstone Benchmark source code was used to build each respective executable. The complete set of experiments (1-4) was executed 3 times for each machine. The results of each 3 runs are included in the table in the order in which they were run.

The Hartstone Benchmark was executed on both machines using the same baseline test; in each Experiment the same amount of computational workload was requested during the first test and subsequent increments in workload were also the same for the two machines. Several general observations can be made which are valid for all four of the experiments:

- Due to the controlled environment in which the experiments were conducted, the results for each three runs of each experiment were very close if not exactly the same. Slight variations are expected due to occasional system functions such as system clock and "cron" tasks. The very narrow variations exhibited in most experiment results is an indication of very few external factors affecting the experiment results.
- The more recently developed RISC microprocessor used in the Sun delivers better raw compute performance than the older CISC in the Masscomp. In the tables included in Figures 1-4, the right most column identifies the number of Whetstones which were calculated by one task in one second at the start of the experiment. This number is identified as "raw KWIPS". The number of KWIPS completed by a collection of tasks should never exceed this raw value due to task switch overhead. The Masscomp consistently computed approximately 2000 Whetstones a second. The Sun consistently computed approximately 4400 Whetstones a second, more than double the number computed by the Masscomp. The "raw KWIPS" value is used as the basis for 100% utilization. Each experiment compares the performance of multiple tasks against this single task baseline.
- The greater compute performance of the Sun ensures a finer scale in reporting of performance for the Sun than the Masscomp. This is evidenced by the "step factor" which is identified for each experiment in Figures 1-4. The "step factor" identifies at which rate the system load is increased in each subsequent experiment test. During the execution of the benchmarks, the initial load and all subsequent compute loads added during the experiments was the same for both the Sun and the Masscomp. Due to the Masscomp's lower performance, the new loads added to the Masscomp were a greater burden than to the Sun. The "step factor" identifies the ratio of loading between the Masscomp and the Sun. The "step factor" is computed by dividing the "raw KWIPS" (100% utilization) by the compute load "KWIPS". The "step factor" for the Sun was consistently 1/2 that of the Masscomp due to the greater raw compute performance.
- Due to the differences in compute performance and the resulting differences in "step factor", the number of tests completed by each machine may not be directly compared in some cases. The Sun is expected to complete more tests due to the greater raw compute performance and the finer loading factor. In cases where the number of tests completed in an experiment are very close for the Sun and Masscomp, it can be suggested that the Masscomp's software (Ada compiler and/or OS) is more efficient at performing the type of test targeted by the experiment. This is due to the fact that it has less raw compute performance available to perform the test. In cases where the variation in the number of tests completed is great, other factors may be studied to determine the nature of the test. The "percent CPU" loading indicator may be used in these cases to determine if the Masscomp is performing as expected when compared to the Sun.

These factors should be kept in mind when studying the Hartstone Benchmark results. Given the preceding backdrop, the following observations have been made:

- Experiment 1 is a good indicator of a system's ability to task switch five tasks with one task increasingly adding task switch overhead. The fifth task gradually increases its frequency as its time period decreases. In this experiment, the Masscomp delivered similar performance to the Sun even though it has less raw compute performance. This may indicate that for the task set in Experiment 1 the Masscomp's Real-Time Unix is an advantage over Sun OS and other BSD Unix implementations even when RTU's explicit real-time features are not utilized. Investigation of Experiment 3 would indicate the opposite is true. For the task set in Experiment 3, the Sun delivered much better performance when compared to the Masscomp.
- The results of Experiment 1 for the Masscomp and the Sun are consistent with the findings often observed at the SEI; the highest priority task (task number 5) misses deadlines before the lower priority tasks. Both the Masscomp and Sun missed deadlines in task 5 in Experiment 1. Due to the priorities assigned by the Hartstone Benchmark, the opposite scenario should be the case. The lower priority tasks should miss their deadlines first as they are increasingly preempted by higher priority tasks. The SEI is currently researching the cause of the "Inverted Task Set Breakdown Pattern" which was exhibited by both the Sun and Masscomp.
- The Masscomp and its RTU operating system produced very good CPU utilization figures when compared to the Sun for the Experiment 1 task set. The Masscomp consistently delivered approximately 17% CPU utilization as compared to the Sun which delivered approximately 8% CPU utilization.
- The Masscomp also performed very well in Experiment 2. Once again, the Masscomp delivered a much higher CPU utilization before deadlines were missed. Although the Masscomp delivered higher CPU utilization, the Sun delivered higher performance given its greater compute performance from its RISC microprocessor.
- The Sun performed exceptionally well during Experiment 3 when it achieved 4102 KWIPS, or approximately 90% of CPU utilization. The Sun also displayed a very even distribution of tasks which missed their deadlines as the system was loaded. In Experiment 1, both the Masscomp and the Sun missed deadlines in the highest priority task. This is not the behavior which is expected or desired. As the load on the Sun increased during Experiment 3, the lower priority tasks began to miss deadlines while the higher priority tasks missed very few or no deadlines. The task set used in Experiment 3 appears to be very well suited for the Sun.
- There were few surprises in the results of Experiment 4. Once again, the Masscomp delivered very good CPU utilization. Once again, the faster Sun CPU delivered higher performance than the Masscomp. As in Experiment 3, the Sun delivered very good distribution of missed deadlines across lower priority tasks as the system was loaded. As the Masscomp CPU was loaded, it did very poorly in distributing missed deadlines across lower priority tasks. In the last test attempted by the Masscomp, the highest priority task was the only task to miss deadlines.

The Hartstone Benchmark has provided insights into the relative strengths and weaknesses of the two hardware and software systems tested. The Hartstone Benchmark clearly showed the compute

performance of the Sun and the effectiveness of the real-time features implicitly available in the Masscomp's RTU. The Hartstone Benchmark also clearly showed how the two systems are very effective for certain experiments and less effective for other experiments. The Hartstone Benchmark is an excellent foundation for research into the real-time performance of Unix-based workstations.

Experiment 1 - Last test with no missed/skipped deadlines

	Test #	Percent CPU	Task5 Freq.	Step Size	KWIPS(raw)
Masscomp	1	16.32%	32Hz	1.63%	1960.79
	1	16.32%	32Hz	1.63%	1960.79
	1	15.31%	32Hz	1.53%	2090.62
Sun	2	7.86%	48Hz	.71%	4476.28
	1	7.17%	32Hz	.72%	4464.29
	1	7.15%	32Hz	.71%	4476.28

Experiment 1 - First test with missed/skipped deadlines

	Test #	Percent CPU	Task5 Freq.	Step Size	KWIPS(raw)
Masscomp	2	17.95%	48Hz	1.63%	1960.79
	2	17.95%	48Hz	1.63%	1960.79
	2	16.84%	48Hz	1.53%	2090.62
Sun	3	8.58%	64Hz	.71	4476.28
	2	7.88%	48Hz	.72	4464.29
	2	7.86%	48Hz	.71	4476.28

Experiment 1 - Test with 50 or more missed/skipped deadlines

	Test #	Percent CPU	Task5 Freq.	Step Size	KWIPS(raw)
Masscomp	2	17.95%	48Hz	1.63%	1960.79
	2	17.95%	48Hz	1.63%	1960.79
	2	16.84%	48Hz	1.53%	2090.62
Sun	3	8.58%	64Hz	.71	4476.28
	3	8.60%	64Hz	.72	4464.29
	3	8.58%	64Hz	.71	4476.28

Figure 1

Experiment 2 - Last test with no missed/skipped deadlines

	Test #	Percent CPU	KWIPS	Step Size	KWIPS(raw)
Masscomp	2	17.48%	352	1.59%	2013.42
	3	19.58%	384	1.63%	1960.79
	3	18.50%	384	1.54%	2076.11
Sun	7	11.73%	512	.73%	4364.91
	6	10.72%	480	.71%	4476.28
	6	10.72%	480	.71%	4476.28

Experiment 2 - First test with missed/skipped deadlines

	Test #	Percent CPU	KWIPS	Step Size	KWIPS(raw)
Masscomp	1	15.89%	320	1.59%	2013.42
	4	21.22%	416	1.63%	1960.79
	2	16.95%	352	1.54%	2076.11
Sun	2	8.06%	352	.73	4364.91
	1	7.15%	320	.71	4476.28
	3	8.58%	384	.71	4476.28

Experiment 2 - Test with 50 or more missed/skipped deadlines

	Test #	Percent CPU	KWIPS	Step Size	KWIPS(raw)
Masscomp	4	20.66%	416	1.59%	2013.42
	4	21.22%	416	1.63%	1960.79
	4	20.04%	416	1.54%	2076.11
Sun	8	12.46%	544	.73	4364.91
	8	12.15%	544	.71	4476.28
	8	12.15%	544	.71	4476.28

Figure 2

Experiment 3 - Last test with no missed/skipped deadlines

	Test #	Percent CPU	KWIPS	Step Size	KWIPS(raw)
Masscomp	7	35.29%	692	3.16%	1960.79
	6	31.29%	630	3.08%	2013.42
	9	39.03%	816	2.97%	2090.60
Sun	55	84.84%	3668	1.43%	4323.39
	58	86.10%	3854	1.39%	4476.28
	58	86.33%	3854	1.39%	4464.29

Experiment 3 - First test with missed/skipped deadlines

	Test #	Percent CPU	KWIPS	Step Size	KWIPS(raw)
Masscomp	3	22.64%	444	3.16%	1960.79
	3	22.05%	444	3.08%	2013.42
	10	42.00%	878	2.97%	2090.60
Sun	2	8.84%	382	1.43%	4323.39
	10	19.61%	878	1.39%	4476.28
	8	16.89%	754	1.39%	4464.29

Experiment 3 - Test with 50 or more missed/skipped deadlines

	Test #	Percent CPU	KWIPS	Step Size	KWIPS(raw)
Masscomp	9	41.62%	816	3.16%	1960.79
	8	37.45%	754	3.08%	2013.42
	12	47.93%	1002	2.97%	2090.60
Sun	62	94.88%	4102	1.43%	4323.39
	62	91.64%	4102	1.39%	4476.28
	62	91.88%	4102	1.39%	4464.29

Figure 3

Experiment 4 - Last test with no missed/skipped deadlines

	Test #	Percent CPU	Tasks	Step Size	KWIPS(raw)
Masscomp	3	22.25%	7	3.18%	2013.42
	8	39.17%	12	3.26%	1960.79
	13	52.04%	17	3.06%	2090.60
Sun	33	52.90%	37	1.43%	4476.28
	34	54.33%	38	1.43%	4476.28
	34	54.33%	38	1.43%	4476.28

Experiment 4 - First test with missed/skipped deadlines

	Test #	Percent CPU	Tasks	Step Size	KWIPS(raw)
Masscomp	4	25.43%	8	3.18%	2013.42
	6	32.64%	10	3.26%	1960.79
	2	18.37%	6	3.06%	2090.60
Sun	4	11.44%	8	1.43%	4476.28
	1	7.15%	5	1.43%	4476.28
	8	17.16%	12	1.43%	4476.28

Experiment 4 - Test with 50 or more missed/skipped deadlines

	Test #	Percent CPU	Tasks	Step Size	KWIPS(raw)
Masscomp	10	44.50%	14	3.18%	2013.42
	15	62.02%	19	3.26%	1960.79
	17	64.29%	21	3.06%	2090.60
Sun	36	57.19%	40	1.43%	4476.28
	36	57.19%	40	1.43%	4476.28
	36	57.19%	40	1.43%	4476.28

Figure 4

2.0 Bibliography

Donohoe, Shapiro, and Weiderman, Hartstone Benchmark User's Guide, Version 1.0, Carnegie-Mellon University Software Engineering Institute, March 1990.

Donohoe, Shapiro, and Weiderman, Hartstone Benchmark Results and Analysis, Carnegie-Mellon University Software Engineering Institute, June 1990.

Proceedings of the Third Annual NASA Ada User's Symposium, November 1990.

Weiderman, Nelson H., Ada Adoption Handbook: Compiler Evaluation and Selection, Version 1.0, Carnegie-Mellon University Software Engineering Institute, March 1989.

Weiderman, Nelson H., Hartstone: Synthetic Benchmark Requirements for Hard Real-Time Applications, Carnegie-Mellon University Software Engineering Institute, June 1989.

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

An Investigation and Comparison of the C and Ada Programming Languages for Use In the Space Station Control Center and Space Station Training Facility

NASA Grant No. NAG 9-435
SwRI Project No. 05-3531

Prepared by:
Timothy J. Barton
Steven W. Dellenback, Ph. D.

Prepared for:
NASA
Johnson Space Center
Houston TX 77058

Presented on:
March 8, 1991

Introduction

NASA is about to embark on the development of millions of line of software for the Space Station Control Center (SSCC) and the Space Station Training Facility (SSTF). The C programming language was used in the development of the Mission Control Center Upgrade (MCCU). Reuse of the MCCU software is a goal if possible, but the long life cycle of the Space Station will also influence the development of Space Station software. Ada is a language which was designed for long life cycles and would appear to be ideal for Space Station type applications, but general perceptions have been that Ada is a risk factor when compared to C for several reasons. This paper has been written to provide input to the question of which language to use for the SSCC and SSTF.

Due to the brief nature of this investigation, an in-depth performance benchmarking of the languages was not possible. These same time constraints did not permit an in depth study of all SSCC and SSTF applications which would be affected by the C versus Ada question. This paper will instead focus on more general discussions of the two languages and will focus on the applications with which the researchers have the most "hands on" experience. These applications are specifically the MCCU workstation applications.

The information presented in this paper is mostly the result of case studies. These case studies provide excellent information because they are the results of projects with experience using the two languages. The following case studies have provided significant input into the recommendations of this paper:

- Lockheed in Austin, Texas, is currently completing a large (30 programmers, 240,000 Ada statements) Ada effort. This effort is of special significance because it is being completed on Masscomp Unix workstations. This effort is also significant because approximately only 1-in-5 programmers actually knew Ada at the start of the project. This effort incorporates a large number of COTS products. This effort is especially significant because a related comparable project is being done in C concurrent to the Ada effort.
- SwRI is currently using Ada in the implementation of the Combined Arms Service Trainer (CAST). This project is relevant due to the number of experienced C programmers which are now using Ada in the development of CAST. This effort is using 80386 PCs as their development and target hardware platform.
- The experiences of the Flight Dynamics Division (FDD) at Goddard Space Flight Center have also provided information for this paper. The experiences of the FDD are of special significance because they have been gathered over almost 6 years and they are mostly from projects implemented on mainframes and minicomputers.

Various other case studies have provided information for this paper. The experiences of the Shuttle Training Aircraft (STA) and the Second TDRSS Ground Terminal (STGT) are some of the other case studies used in the preparation of this paper.

These case studies and information gathered from recent publications provide the basis for this discussion concerning Ada and C. The following ten questions have been identified by NASA in regard to the Ada versus C question. Each of these questions is addressed within this paper.

Questions

1. Does the amount of MCCU C code available for reuse justify the use of C for new SSCC code?

   [Less complex operating (run-time) environment then would be required to handle C, Ada, and possibly Fortran in TCATS System]

2. Can the productivity improvement of 2-lines of code per PYE per day claimed by Loral be justified when going from Ada to C language?

   [8-line/day for Ada versus 10-line/day for C]

3. Can coding in C be done in a way (i.e., using up front software engineering practices and tools) that results in a product that is as easily maintained as an Ada code product, i.e., is the life cycle cost for C code naturally higher than Ada code?

4. Does the control center and trainer systems requirements differ sufficiently to justify C language for SSCC and Ada language for SSTF (i.e., real-time operation of trainer systems, mostly new code required by Trainer; however, some Fortran reuse is expected, etc.)?

5. Does the future of code development suggest limitations on resources (people and tools) for C programming versus Ada programming (next 10-20 years)?

6. Are the up front costs for Ada (tools, training, and lack of experienced programmers) significant compared to that for C?

7. What are the risk assessments for completion of the SSCC and SSTF on schedule and within cost considering use of C versus Ada languages for new code in each facility?

8. Are COTS products that will be available in the next 1-10 years more likely to be in C or Ada language?

9. Will COTS products in the next 1-5 years be available that will support multiple language applications (C, Ada, Fortran)?

   [i.e., will COTS tools likely have bindings for Ada, Fortran, and C, such as X windows and other system services tools]

10. What would be your recommendation for language use in the SSCC and SSTF given the desire to maximize use from Shuttle, maximize portability across SSFP and within the SSCC/SSTF facilities, independence from hardware constraints, and significant budget pressures to cut up front costs and reduce run out costs with minimum risk to delivery capability and schedule?

    [Consider resources availability, etc.]

1. Does the amount of MCCU C code available for reuse justify the use of C for new SSCC code? [Less complex operating (run-time) environment then would be required to handle C, Ada, and possibly Fortran in TCATS System]

This is very dependent on two principle factors, the amount of MCCU C code which can actually be reused in SSCC and the benefits of Ada compared to C. If a majority of all Delivery 1 SSCC software is directly reused from MCCU, then C might be the best selection for all SSCC code. If it is demonstrated that Ada offers significant benefits over C, then Ada might be the best selection for all new SSCC code given the expected long life cycle of the SSCC.

The first factor in determining whether new SSCC code should be written in Ada is to ensure that Ada offers benefits over C for the life cycle of the SSCC. Everyone would agree that it does not make sense to use Ada if C is going to provide more benefits over the life cycle. It does not make sense to use Ada just because it has been widely associated with the SSFP. For the sake of argument for this question, it will be assumed that Ada provides benefits over C which are desired by NASA. If this is not the case, then the selection of C would be obvious. The Ada versus C argument will be addressed within other questions.

Given the assumption that Ada provides benefits over C which are desired, the decision to use Ada for new SSCC software is dependent on two factors:

- The amount of MCCU software which will be reused could justify the continued use of C in new SSCC software. If 90% of MCCU software could be reused without modification, then it would make sense to continue to use C. A completely C based SSCC environment would be less complex than a mixed language environment and might allow compatibility between SSCC and MCCU applications. Compatibility with MCCU is clearly a goal of the SSCC in an effort to reduce development costs for both centers. Differences in mission, scope, and data acquisition strategies, may make this compatibility goal impossible to attain. If this goal can be realistically attained, it makes sense to reuse the existing base of MCCU C code and to develop new SSCC code also in C.
- The differences in mission, scope, and data acquisition strategies will probably mean compatibility between MCCU and SSCC will not be maintained beyond Delivery 1 of the SSCC software. The amount of MCCU software which can be directly inserted into the SSCC without modification is probably very small in light of the modifications which are required to meet the SSCC requirements. The following are examples of the MCCU workstation software which have been selected for integration into the SSCC from MCCU:

  Display Manager

  Display Builder

  Comp Manager

  Comp Builder

  Each of these applications will require modifications before they can be integrated into the SSCC. The potential for reuse varies for each of these applications based on the new SSCC requirements for each application. The Display Builder and Display Manager applications are tentatively scheduled to receive new interfaces (GKS to X Windows), new data acquisition strategies, and additional functionality to meet SSCC require-

ments. The new interfaces and data acquisition will require extensive modifications to both applications and may limit reuse to as little as 10% of existing MCCU Display Manager and Display Builder software. In the case of the Display Manager and Builder, it may be reasonable to rewrite these applications from scratch using Ada if Ada is determined to be more desirable than C due to the projected long SSCC life cycle.

The potential for reuse is much greater for the MCCU Comp Manager and Comp Builder applications. These applications require less extensive medications to their user interfaces to meet SSCC requirements. Modifications are still required to their respective user interfaces, but the potential for reuse is probably about 60-70% in the user interface portions of these applications. The Comp Manager and Comp Builder applications will require extensive modifications for the new data acquisition strategies used in the SSCC and to meet new SSCC requirements. These required modifications, and the much smaller size of these two applications when compared to the Display Manager and Display Builder, also makes it reasonable to rewrite them from scratch in Ada if Ada can be shown to provide important benefits over the long life cycle which is projected for these applications.

One important aspect of converting the Comp Builder to Ada is the translation of comps into executables by the Comp Builder. All versions of the Comp Builder use the workstation resident C compiler to produce machine executable comps for the Comp Manager. An Ada based Comp Builder will either require the Ada compiler to be resident on the Comp Builder workstation, or the executable comps may continue to be generated via the C compiler. Converting the Comp Builder to use Ada as the target language from which executable comps are produced will have a sizable impact on the Comp Builder's translation functions.

Extensive modifications will be made to the four existing MCCU applications identified above before they are integrated into the SSCC. These modifications have been shown to significantly impact the percentage of code which can be reused. The relatively low percentage of existing code which can be reused coupled with the new software which must be developed makes it reasonable to consider Ada for use in the SSCC if Ada can be shown to provide benefits over C.

An important option is the integration of both languages into the SSCC. NASA-Goddard and other commercial vendors (Lockheed-Austin as one example) have had very good success utilizing a mixture of C and Ada. The Flight Dynamics Division (FDD) at Goddard has committed to Ada as their first language of choice. Even with this commitment to Ada, the FDD still reserves the right to use C in portions of their systems where C is more appropriate than Ada. The FDD is currently starting the development of a large software development effort estimated to be a 300 man-year effort. Most of the software will be developed in Ada, but some portions of the system will be developed in C because they feel C is a better choice than Ada for some portions of the entire system. The FDD feels that C is a better language for low level graphics operations.

The Comp Builder may be an excellent application in which to use both Ada and C. The Comp Builder application may be written in Ada while the comps which it generates may continue to be complied C. This would utilize Ada where it is most appropriate, but would still utilize C for a portion of the application where it is well suited due to the transparent compilation of comps which must occur. The runtime impacts of the port to Ada would be minimized for the Comp Manager and a larger portion of existing Comp Manager code could be reused. This would allow 95% of the

Comp Builder/Manager software to be written in Ada, and C could be used for the 5% of software where its benefits over Ada could be realized.

2. Can the productivity improvement of 2-lines of code per PYE per day claimed by Loral be justified when going from Ada to C language? [8-line/day for Ada versus 10-line/day for C]

The different objectives and philosophies of the Ada and C languages make an 8 lines per day versus 10 lines per day comparison difficult. It is reasonable to expect that in one day a C programmer will write more lines of code than an Ada programmer. For smaller projects this has been demonstrated and is widely accepted. Over the course of longer projects, the benefits of the Ada language compensate for its more complex and larger source files.

The experiences of developers on the SwRI CAST program would indicate that it will take slightly more time to develop Ada source code than C source code. This is attributed to the fact that the Ada language is more verbose than C and it is widely acknowledged that Ada is a better self documenting language than C. C was designed from its beginning to be the HOL replacement for assembly language. According to the fathers of C, "C is a relatively 'low level' language."

> "C is a general-purpose programming language which features economy of expression ... C is not a 'very high level' language, nor a 'big' one, and is not specialized to any particular area of application." [8]

Ada is a much larger and more complex language than C and its source files are as a result typically more verbose than C. Many Ada functions are often written in two source files: the package specification and the package body, whereas the same function in C would only require 1 source file. There is a price to pay for the additional documentation provided by the more verbose Ada source code.

Is the additional price for Ada source code 2 lines per day? For a smaller project, it is reasonable to expect development in Ada to take longer.

> "The same strong typing and documenting that make Ada programs easy to understand and cost-effective to maintain, can sometimes make it awkward to write simple test programs." [6]

Over the course of larger projects, the same features which make Ada awkward to use on small projects, provide benefits which reduce development time. The experiences of SwRI and Lockheed indicate that the entire development cycle for a large Ada program is not significantly longer than similar C projects. The experiences at Lockheed are an excellent example of the relative productivity of Ada and C programmers. Two similar projects are currently completing final testing at Lockheed. One project was written in C and one project was written in Ada. Almost all of the C programmers were trained in C at the start of the C project, while only 1 in 5 programmers out of a total of 30 were trained in Ada at the start of the Ada project. The other Ada programmers received mostly "on-the-job" training as the project progressed. Programmers on the C project developed more code initially, but the Ada project completed integration about the same time as the C project. Project managers at Lockheed acknowledge that it took longer to get started in Ada,

but over the entire development cycle, especially the testing and integration phases, Ada compared very similarly to C.

The FDD at Goddard has collected extensive statistics comparing development in Ada to FORTRAN. These statistics compare Ada to FORTRAN instead of Ada to C, but the similar complexity of FORTRAN and C makes the information useful for comparison. The FDD has collected comparison data over 5 years during 10 projects, 5 FORTRAN projects and 5 Ada projects. They have found that the actual cost per line of code is either the same for FORTRAN and Ada, or that Ada is actually cheaper than FORTRAN. These costs do not factor in the significant development costs saved through the reuse of existing Ada software.

For a smaller scale project, most everyone agrees that given a trained Ada programmer and a trained C programmer, the C programmer will be able to complete more new code sooner. For a larger, more complex project, most everyone agrees that there will be little difference in time-to-completion between Ada and C. Given this theory, and given the fact that most of the applications which will be in the SSCC will be fairly large and complex, it would follow that contractors should be able to complete Ada versions of the SSCC applications in the same PYE. Unfortunately, few if any contractors have the level of Ada expertise available which will be required. The 25% difference estimated by Loral is assumed to cover training time as C programmers transition to Ada programmers. This 25% penalty should only be absorbed once. Thereafter, the cost for an Ada project should compare very closely to the cost for the same project written in C.

3. Can coding in C be done in a way (i.e., using up front software engineering practices and tools) that results in a product that is as easily maintained as an Ada code product, i.e., is the life cycle cost for C code naturally higher than Ada code?

Software maintenance is the process of altering computer source code after the initial version of the system is placed into production. Software maintenance is a broad term and is used in the following context, that is, software maintenance includes:

- Modification to correct errors and design defects
- Modification of existing features to improve the software design
- Adaptation of the software to coexist with new hardware and software (i.e. software will need to be retested and possibly modified when new versions of operating systems are installed)
- Modification to basic data sources (such as files, databases, real-time data sources, etc.)
- Implementation of new features within the basic design

Any robust and heavily used system will have tremendous requirements for change, particularly after the first major release of the system. It has been estimated by many authors, through case studies and theoretical evaluation of the software life cycle, that most organizations will devote up to 80% of their computer resources to the maintenance of software (the balance of the resources is utilized for development of software systems).

With the proliferation of software systems over the last twenty years, the software industry is struggling to maintain the many systems currently in production. The major hurdle for most organizations is that the systems developed over the last twenty years lack appropriate documentation as well as established (or utilized) procedures to perform software maintenance. As a result, many organizations rarely reuse software from previous efforts because of the lack of understanding of what exactly exists.

During the late 1980s, a proliferation of software development environments have been marketed by various organizations. These software development environments allow software requirements, actual source code, test procedures and user documentation to be integrated into a single environment so that the necessary information about the source code is available in a controlled, complete manner.

When selecting a computer programming language for a system to be developed which will need to be maintained over a number of years, a careful evaluation of the programming language features needs to be made to assure that the selected language will cost effectively serve the software life cycle.

The C programming language was designed to provide system level programming services for the UNIX operating system. The environment was initially targeted for a research environment where software life cycles were not a concern. The initial design goal of C and UNIX was to provide an environment for a researcher to solve research problems. The language is a level of abstraction above assembly language which provides the user a language which has the flexibility to fully exploit the host computer while not providing many semantic constraints (implying that is easy to develop C source code files which behave erratically). The UNIX operating system has matured

and evolved tremendously from its initial design, the C language has essentially not changed since its introduction.

The Ada programming language was conceived by the Department of Defense to solve the problem of providing a language for development of embedded systems that could be developed and maintained by many people over many years. Ada was designed as an environment, that is, one of the primary design goals was to provide a language which encouraged the reuse of software. This was achieved by introducing rigid standards on the development of software routines so that the difficulty of transporting software between applications is minimized. The Ada language has many explicit syntactic and semantic constraints which do not allow a programmer to develop ill-behaved source code.

Many programming environments have been developed for Ada, these environments provide an excellent foundation on which long term software programs can be developed. Ada was designed from the start to support long term project development (it was not a design goal of the C language), the Ada language provides many features which greatly simplify as well as encourage a structured software development environment.

SwRI believes that in the long-term, if correctly utilized, Ada will require fewer software maintenance resources than a similar C program. Initial development costs for the C program may be less than for an Ada program (particularly for smaller applications) but over the course of the entire life cycle it is felt that costs will be less in the case of Ada. Due to the personnel and contract turnover present at NASA, it is obvious that any long term project will have numerous personnel assigned to the effort. The surest way for NASA to develop a good software product would be to choose a development environment in which NASA can establish maximum control of the software development process -- this environment exists for Ada.

Additionally, it might be possible to establish an environment for C which would be as rigorous as most Ada environments -- however that environment is not commercially available today. If this C environment were to be established, many manual controls and procedures would have to be instilled into the environment in an equivalent manner to many existing Ada environments. While these manual controls and procedures may be initially followed, it is difficult to predict whether or not they could be maintained over many years. In contrast, many of the Ada controls and procedures are natural to the development environment and if they are not followed -- application programs will not be completed.

The answer to this question is both yes and no. Software written in C can be developed in a manner similar to Ada. MIT's X Windows and OSF's Motif are two examples of programs written in C which are designed with some of the Ada mentality. Ada will provide a more natural mechanism for this type of decomposition and will enforce certain aspects which are important to the software development life cycle.

4. Does the control center and trainer systems requirements differ sufficiently to justify C language for SSCC and Ada language for SSTF (i.e., real-time operation of trainer systems, mostly new code required by Trainer; however, some Fortran reuse is expected, etc.)?

The SwRI researchers who generated this paper have not received as much exposure to the SSTF as they have the MCCU workstation applications. There was not enough time to investigate the SSTF applications, so a recommendation has not been made specifically for the SSTF applications. A blanket endorsement across as large a center as the SSTF for either C or Ada is not recommended. Each language has its strength and weaknesses. A generic set of metrics should be determined which can be applied against any application to determine which language should be used. Once the metrics are determined, then they can be applied against applications such as the trainer systems.

One of Ada's strengths is the support within the language which encourages reusability of code. Experiences at Goddard and within industry support the claim that Ada encourages reusability of code. On the surface, a training system would seem like the ideal type of system in which to reuse software.

5. Does the future of code development suggest limitations on resources (people and tools) for C programming versus Ada programming (next 10-20 years)?

There are many indications that both Ada and C will continue to be supported for at least the next 10 years. The large base of C software and programmers will ensure C's continued widespread use. Ada support and usage is growing steadily, and it is reasonable to assume an increasing availability of Ada tools and programmers.

C has achieved widespread acceptance across a variety of hardware platforms. In the Unix workstation arena, C is the dominate language for software development due to the fact that Unix itself was written in C. In the PC arena, the popularity of C is evidenced by the sheer number of available C compilers, C tool chests, and COTS written in C. There is a large and ever expanding pool of trained C programmers due to the demand for applications where C is typically used. Most universities in the country offer a C class, and many base their core curriculum around the use of C.

The next 10 years will see continued widespread support and use of C. University curriculum, and the growing PC and workstation markets will ensure C's continued success.

Although C has achieved widespread success and support, and this support can not be overlooked or under estimated, the projections for C are not without some questions. The recent popularity of C++ indicates that "vanilla C" does not fulfill every objective. It is interesting to note that the features which are being added to C++ are many of the features which were the basis for the design of Ada.

C++ is a relatively new language when compared to C or even Ada. C++ was developed at AT&T by Bjarne Stroustrup. The first C++ translator was completed in 1985. C++ adds the object oriented data encapsulation and abstract data types features of Ada to C. C++ also incorporates other important Ada features such as function inlining to improve performance. C++ is evidence of the fact that object oriented design and object oriented programming (OOP) are powerful tools in the development of complex applications.

C has always been regarded as a slightly higher level assembly language with the power and responsibilities which come with that power. Some even regard C as a more "dangerous" tool than assembly language because of the power and freedom permitted within C. The power and freedom which are C's strengths require responsibility and organization by the programmer for even modest sized programs. C++ is trying to address these issues:

> "Whereas the C programming language tests a programmer's inner strength and builds character by following an 'anything goes' philosophy, C++ is the programmer's friend, providing compile-time error messages that enforce data encapsulation." [9]

Products which are coming to market indicate C++'s popularity. More and more compilers are becoming available for C++ for both the PCs and the workstations. Even the MOTIF bindings from OSF now support C++. It appears that the computer industry has acknowledged the power of OOP. The well publicized "vaporwares" and the failures of software companies to deliver products ontime may instigate a rapid move toward OOP and C++.

The 1960's were dominated by assembly language. The 1970's and 1980's were dominated by structured programming and the popularity of HOL languages such as FORTRAN, C, Pascal, and Lisp. The 1990's may be dominated by OOP and languages such as Ada and C++.

Ada did not catch on and dominate the industry as it was projected to do during the last five years of the 1980's. The Ada language was developed with much fanfare and was touted as the DoD's answer to its mounting software costs. Unfortunately, too much was expected of Ada before it had matured. C language compilers were given the opportunity to mature over several years before C became widely used. The fanfare and the expectations placed on Ada did not allow the language several years to mature, and when the language did not meet the industry's expectations, it was regarded as unusable. This perception of Ada still exists today in the mind of many software developers and system designers.

The Ada language is more complex than C and Ada compilers are as a result necessarily more complex than C compilers. The Ada ANSI standard also requires that the complete Ada language is implemented before the compiler receives validation. Most of the original Ada compilers were expensive, of poor quality, and not very efficient. As a result, the Ada language itself was criticized for being inefficient when the implementation of the language was the culprit. This perception still exists today.

Ada compiler technology has improved significantly and continues to improve.

> "Technical problems with Ada still exist, although experts say they're decreasing steadily in significance, and could virtually disappear over the next three to five years." [6]

The experiences at Cray, the supercomputer manufacturer, are an excellent example of the improvements in Ada compiler technology. The Cray Ada compiler currently produces code which outperforms the same code written in FORTRAN and compiled using Cray's FORTRAN compiler. The Cray Ada code for the Whetstone and Dhrystone benchmarks outperforms the same code written in FORTRAN.

There are numerous examples of the advancements in Ada compiler technology. It is important to remember that the source language has little impact on many operations performed within an executable. When two variables are subtracted in a HOL language, it is the compiler's job to convert that subtraction into the CPU's machine code which performs two register loads from memory and then a subtraction of the two registers. The CPU should perform the same task independent of the HOL language which was used to build the source code.

> "On similar benchmarks I have seen straightforward scalar Ada come very close to what C can do on the same code. Indeed I have seen claims that for this kind of code, Ada has done better than C. I am very surprised that anyone who halfway understood compilers would claim that unadorned, numerical, scalar code from a language A program would necessarily be slower than the same code from language B. I'd be really delighted to see these claims debunked forever. On a level playing field, there are only X many ways to evaluate a scalar expression. ... The growing maturity of compilers, users, and managers is really gratifying! How long do you think it'll take for the bulk

> of the industry to throw away their outdated stereotypes about slow, clunky Ada? It's good to see intelligent debate about what really makes programs slow. It ain't the source language, folks." [11]

The performance of a language should not be an issue. The language's ability to represent these operations should be the issue. Ada may in fact offer advantages over C for some applications:

> "... hardware vendors are discovering that Ada may offer an inherent advantage for programming digital signal processors over its arch rival, C." [7]

A tremendous amount of effort is being spent on the investigation and improvement of Ada performance. Ada has undergone a maturing process during the last five years and it is now struggling to dispel the many adverse opinions which were formed by software developers during Ada's formative years. Support for Ada is steadily increasing as more quality Ada compilers become available to universities and industry.

It should be noted that it took C about 10 years to gain wide acceptance after it was written. Just as the legacy and installed base of FORTRAN and assembly language systems slowed the acceptance of C, so also has the large installed base of C and FORTRAN slowed the acceptance of Ada.

Ada has always been expected to succeed due to the DoD's sponsorship of its development and subsequent mandate of use in 1983.

> "The birth of a new programming language is rarely celebrated far beyond the immediate family. But, when the sponsor of the language is the largest consumer of computers in the world, then it becomes a major event." [10]

The DoD's 1983 mandate allowed waivers to be granted if Ada compilers were not available or when using Ada would be more expensive or might prevent developers from meeting their schedules. The poor performance of Ada code and lack of trained Ada programmers resulted in many waivers being granted since 1983. The relative ease with which waivers were obtained allowed C and FORTRAN to prosper and Ada's problems to persist.

The DoD has recently added teeth back into its Ada mandate by extending its mandate to include automated data processing in addition to embedded real-time applications. Congress further strengthened the DoD's mandate by adding language to the 1991 Defense Appropriations Act which states that after June 1, DoD programs that do not embrace Ada will be breaking the law. As before, Congress has left a loophole, but at this time it will be much harder to demonstrate the need for an Ada waiver.

The experiences of the FDD at Goddard are also of special interest. Frank McGarry, the Division Chief of the FDD at Goddard has seen the maturing of Ada since the FDD started using Ada in 1985. The FDD has committed to making Ada their main language by 1995. The FDD has seen an increase in the quality of Ada compilers and professionals, and the FDD has also seen an increase in the number of contractors bidding Ada in proposals. Mr. McGarry thinks the most encouraging

sign is the number of contractors bidding Ada on their own initiative when Ada was not identified as a requirement.

The next 10 years looks promising for both C and Ada. C's large base of applications and trained programmers will ensure its continued use for many years. Recent developments surrounding C++ seem to indicate a shift towards C++ on a large scale over the next few years. C++ may be a better option than C itself when the long life cycle of the SSCC and the complexity of the SSCC applications is considered.

The next 10 years also look very promising for Ada. The next 5 years may see a more significant growth of Ada than the last 10 years, due to the availability of quality compilers, the new DoD mandate, the increasing interest in OOP, and the increasing complexity of DoD applications.

This paper has not attempted to project the state of C or Ada past 10 years. The computer industry has proven to be too fast paced and uncertain. In 1980, the great language scholar Ellis Horowitz predicted that Pascal would be the language of the 1980's. Pascal did indeed enjoy a large measure of success during the 1980's, but the later part of the 1980's saw the emergence of C as the language of choice for engineering type applications. It is almost impossible to determine whether C or Ada will receive greater support 20 years from now. The trends we are observing today are the interest in OOP, the emergence of C++, the US Congress continued endorsement of Ada, and the increasing quality of Ada compilers.

6. Are the up front costs for Ada (tools, training, and lack of experienced programmers) significant compared to that for C?

The up front costs for Ada will be higher than for C. The size and total cost of the project will once again determine whether Ada is a viable option. Ada tools and training are more expensive and are fewer in number than C tools or training because Ada is a more complex language with less maturity and therefore less industry support. There is a lack of experienced Ada programmers when compared to C. The size and nature of the project may mitigate the deficiencies of Ada or enhance them.

The lack of trained programmers is always more of a concern for Ada project managers than C project managers, and Ada training often confirms a project manager's bias against Ada. A survey of Ada training courses compared to C training shows that Ada training courses are longer and more expensive than C training. A number of video tapes are available for self-paced C training, while very few Ada video tapes are available. Ada is definitely a more complex language than C and to use the language well typically requires a different design philosophy than is typically used with C or FORTRAN. Ada is acknowledged to have a steeper "learning curve" because the language is more complex and because Ada incorporates OOP practices which are less prevalent than the traditional structured programming techniques. C is a smaller language which is less imposing than Ada. It is widely believed that programming in Ada requires much more extensive and expensive training.

NASA Goddard has noticed a change in the nature of Ada training during the last several years. In 1985 the FDD committed to the use of Ada where possible and as a first step towards Ada the FDD programmers enrolled in Ada training. The Ada training at that time was a very intensive training in both OOP methodologies and the application of OOP using Ada. During the last several years, the FDD has noticed a change in the nature of Ada training toward less intensive training with less emphasis placed on "grandiose OOP techniques." The FDD has noticed a trend toward "C like" training, and they feel the effectiveness of this training is comparable to past training methods. This trend is one the FDD is watching closely as they progress toward using Ada for 70% - 80% of all projects by 1995.

Interviews with three Ada project managers have discovered a surprising but consistent theme. The experiences at SwRI, Lockheed, and NASA-Goddard indicate that the extensive training that was generally associated with Ada may not be required for the whole project team, and that the type of training now being offered is sufficient for a segment of the programmers on larger projects. This recent change in approach to training may lessen the cost and impacts of a lack of skilled Ada programmers.

Many project teams are using a tiered approach to Ada training. If an Ada project was to be composed of 10 programmers, 2 programmers would be skilled Ada programmers. Three programmers would be moderately trained, and the other 5 programmers would have little or no Ada experience at the start of the project. The layout of the software and the design of the interfaces would be recorded in Ada package specifications. The majority of the implementation would be completed in Ada package bodies by the 5 novice Ada programmers based on the Ada package specifications developed by the more experienced/trained programmers.

This tiered approach allows the more experienced developers to do the overall system design, the mid-level programmers to do the remaining interface design, and the inexperienced programmers to do the implementation. This allows the more experienced programmers to have a greater impact

on the design of the entire system, limiting the scope of the inexperienced programmers to implementation. Individual package bodies can then be recoded by a more experienced programmer if it becomes necessary. The recoding of a package body is less likely to have a "ripple effect" when the software is in Ada as opposed to C. The project managers at both SwRI and Lockheed feel the tiered approach helps keep projects out of trouble in the area of interfaces.

This tiered approach has been used at both SwRI and Lockheed with very good results. This approach alleviates some of the concerns about a lack of trained Ada programmers. Both Lockheed and the SwRI CAST group feel that using a tiered approach, even given a lack of skilled Ada programmers, is LESS of a risk factor on a large program then using C. The project managers of the SwRI CAST project feel the potentially greater benefits of Ada outweigh the initial training which is required and the lack of experienced programmers.

Ada compilers are in almost all cases more expensive than C compilers. There are several good Ada compilers which are comparable in price to C compilers for the same hardware platform, the AdaZ compiler for the PC is an example. There are several factors which usually cause Ada compilers to cost more than C compilers. The Ada language is more complex than the C language, and the resulting Ada compilers are more complex to develop. Ada compiler writers must implement the entire ANSI/MIL-STD-1815A language to become validated, whereas the C language has only recently become an ANSI standard. The lack of a defined standard allowed C compiler writers more freedom in the implementation of their compilers. A standard programming environment is also specified for Ada, the Ada Programming Support Environment (APSE), and the Ada compiler writer must provide these additional tools which may not be bundled with C compilers.

The up front costs for Ada are a very real concern. There is generally a lack of trained Ada professionals. This is usually perceived as a very important risk factor. Projects at NASA, SwRI, and Lockheed have shown this is not as big a risk factor as previously expected. The more complex Ada language dictates that Ada training is more expensive than C training and there are fewer sources of Ada training. NASA Goddard has seen a change in emphasis in Ada training which may indicate that an intensive study of both the Ada language and OOP design methodologies is not needed. NASA Goddard has noticed a trend during the last several years toward C language style training. The more complex Ada language and requirement for full implementation of the language for validation dictates that Ada compilers are more expensive than C compilers and there are fewer Ada compilers available. Smaller scale, less complex projects will not be able to offset the additional costs of Ada, but experience has demonstrated that larger projects can absorb the up front costs of Ada. The nature of the project will determine if the additional costs for Ada training and compilers can be offset by the good software engineering principles that are embodied in Ada which allow the management of complex programming tasks.

7. What are the risk assessments for completion of the SSCC and SSTF on schedule and within cost considering use of C versus Ada languages for new code in each facility?

There is greater support for C in environments similar to the SSCC and SSTF; C is established and has been proven to work. Ada does have risks associated with it when compared to C, but Ada also offers benefits in areas that C is suspect. The following table lists many of the factors which will be involved in the development and maintenance of the SSCC and SSTF. These factors have been grouped according to each language's strengths. A "+" indicates the language is uniquely superior, a "-" indicates the language is clearly more of a risk factor.

Factor	Ada	C
Programmer productivity	+	+
Software development support tools	+	+
Future Usage		+
Widespread acceptance, programmer availability		+
Small program risks for completion, integration		+
Interfaces to COTS		+
Compiler cost, quality		+
Existing MCCU software reuse		+
Training costs, availability		+
DoD Mandate, Congressional support	+	
Software error rates, reliability	+	-
Future software reuse within SSCC, SSTF, SSFP	+	
Large program risks for completion, integration	+	
Portability	+	
Standard language, language features (ANSI - ISO)	+	
Maintainability	+	

8. Are COTS products that will be available in the next 1-10 years more likely to be in C or Ada language?

For at least the next five years, more COTS products will be written in C than in Ada. There are several important reasons why C will dominate COTS product development in both the Unix workstation and PC arenas.

Ada is currently used primarily for large DoD contracts. Some work is being done outside of DoD and on smaller scale projects, but for the most part, Ada is used for large scale projects for either the military or NASA. These large projects often apply to a very specialized problem domain and are not applicable to the general market place. Ada is currently used by large government contractors such as GE, Lockheed, and Link. GE is currently working on the TDRSS satellite terminal for NASA and Link wrote the B2 trainer entirely in Ada. Both of these Ada systems are success stories, but there is little general need for a stealth bomber trainer or satellite terminal. There are several exceptions, the STARS Xlib bindings is one example, but for the most part the results of most Ada development projects do not become COTS products due to their nature.

Most COTS products are developed by Independent Software Vendors (ISV) and smaller software companies. These companies are most concerned with up-front development costs and quick market delivery. There are a number of factors which lead ISVs and smaller software companies to develop in C instead of Ada.

The compiler is the most important tool of software developers and C compilers are traditionally less expensive than Ada compilers. In the Unix workstation arena, the C compiler is often bundled with the OS or is readily available at a reasonable cost from the OS supplier. A prime example is Sun Microsystems workstations. Sun workstations come bundled with a C compiler at no additional cost, and the excellent GNU C compiler is also available free of charge for Sun workstations. An Ada compiler for the same workstation will cost approximately $5,000 dollars and may cost significantly more.

The compiler situation is even more clear-cut in the PC arena. There are a number of readily available excellent C compilers available for the PC. Good C compilers are available for less than $100. There are far fewer good Ada compilers available for the PC and the compilers which are available are more expensive. As in many areas of comparison between C and Ada, the number and quality of Ada compilers for the PC is improving while the cost is declining. Meridian's AdaZ compiler is an example of a good quality Ada compiler for the PC.

The larger pool of C software developers and the lack of good Ada software developers also causes more COTS products to be written in C. The costs associated with the additional hardware resources (memory, disk space, processing power) which may be necessary for Ada software development also influences COTS developers to use C.

These factors and others indicate that more COTS products will continue to be available written in C than Ada. The way which COTS products are used may reduce the importance of the language in which COTS packages are written. The language used to develop shrink wrapped applications or OS software which do not contain an Application Program Interface (API) is of little or no importance. If Lotus 1-2-3 or "vi" is written in Ada or C, does not matter. In both cases, an API is not required, so the language used to develop the application does not matter.

In cases where the API provided by the COTS package is utilized, then the languages supported natively by the API become important. This issue is addressed in the next question.

It is very hard to predict if C's popularity with ISVs and software companies will continue five years from now. The large pool of C programmers, the growth in the Unix based workstation market, and the increasing use of C (as opposed to assembly, Pascal, or FORTRAN) on PCs will definitely assure the development of more COTS packages in C than Ada.

The 1990's appear to be the decade of Object Oriented Programming (OOP) in the way that structured programming and the use of High Order Languages (HOL) dominated the late 1970's and 1980's. The number of COTS developed in C++, Ada, or even Smalltalk in the next five years may increase as OOP is taught in the universities and migrates into the commercial sector. C++ has already shown phenomenal growth within the last two years. Examples of this are evident in the number of C++ compilers which have recently become available and the recent release of the Motif bindings from OSF which support C++. The use of the C++ language may surpass "true" C in the development of COTS products during the next 5 years.

9. Will COTS products in the next 1-5 years be available that will support multiple language applications (C, Ada, Fortran)? [i.e., will COTS tools likely have bindings for Ada, Fortran, and C, such as X windows and other system services tools]

Currently, more COTS APIs are available for C than for Ada. This trend will continue for at least the next 5 years. In the workstation arena, Ada bindings are available for most popular applications. If Ada bindings are not available, the Ada pragma construct can be used to interface to the C bindings. This approach has been used very successfully by Lockheed with a number of COTS packages, including database applications. The following APIs are good examples of the availability of APIs for C and Ada:

- The POSIX standard's API was originally written in C due largely in part to Unix's large influence on POSIX. The POSIX bindings are now also available in FORTRAN and Ada.
- The X Window X11R4, MOTIF, and XView APIs were originally written in C. Ada bindings are now available for all three. C++ bindings are also now available for MOTIF, another indication of the growing popularity of C++.

Computer Aided Software Engineering (CASE) is one field which has embraced Ada extensively. Most leading CASE tools operate well in an Ada environment and many generate Ada code from information entered into the CASE tool. CADRE's Teamwork product is currently being used by GE on the STGT project and the results have been very encouraging.

More COTS bindings will continue to be available for C than Ada. The number of Ada bindings which are available is steadily increasing and the Ada pragma interface is available when bindings are not available.

10. What would be your recommendation for language use in the SSCC and SSTF given the desire to maximize use from Shuttle, maximize portability across SSFP and within the SSCC/SSTF facilities, independence from hardware constraints, and significant budget pressures to cut up front costs and reduce run out costs with minimum risk to delivery capability and schedule? [Consider resources availability, etc.]

The Ada language was designed for the large, complex, long life-cycle, application domain. The C language was designed for freedom, compactness and efficiency. The applications which will be written for the SSCC more closely match the goals of Ada than C. From a language viewpoint, Ada is the better choice. From an implementation viewpoint, C has traditionally been the best choice due to the quality of C compilers and the relative poor quality of Ada compilers.

The quality of Ada implementations varies even today. Depending on the target hardware platform, C may still be the only reasonable choice for large applications because a suitable Ada implementation may not exist. Preliminary information may indicate this is still the case for large IBM mainframes. Good Ada environments do exist, the DEC environment is one example, and more and more quality Ada compilers are becoming available. In the case of the military standard 1750A microprocessor, Ada and FORTRAN are the only choice.

The availability of a good Ada environment is the first question which must be asked. If a good Ada environment is not available, then C is the only choice. If a good Ada environment is available, then which language should be used? It is conceivable that both languages should be used. Ada and C were designed from their beginnings for different applications and they should be applied to the applications for which they were appropriate. C was designed by Dennis Ritchie as a "relatively 'low level' language" which is ideally suited for systems programming. Ada was designed through an international review process for large, real-time, embedded applications. It appears that most applications which will be integrated into the SSCC and SSTF fit the latter category and as a result Ada should be used if a good environment is available.

The SSCC and SSTF should not select a single language unless all applications which will be developed for those facilities are similar in nature or unless one of the languages is not a viable option due to a lack of quality compilers. Ada and C should be applied in each facility where appropriate to meet the priority of requirements identified in question number 10. A characterization of the type of applications should be made for each facility, and then the appropriate language applied to that type of application which best meets the priorities and requirements.

BIBLIOGRAPHY

[1] Booch, Grady. 1987. Software Components with Ada. Menlo Park, California,: Benjamin/Cummings.

[2] Horowitz, E. and J. Munson, Sept. 1984. An Expansive View of Reusable Software, IEEE Transactions of Software Engineering, vol. SE-10 (5), p. 477,479.

[3] Nise, N., C. McKay, D. Dillehunt, N. Kim, and C Giffin. Oct. 1985. A Reusable Software System, Proceedings of the AIAA/ACM/NASA/IEEE Computers in Aerospace V Conference, p. 492. Long Beach, California.

[4] Bryne, Dan, Richard Ham, Winter 1990. Ada Versus FORTRAN Performance Analysis Using the ACPS, ACM Ada Letters - Ada Performance Issues Volume X, Number 3, p. 139-145. New York, New York.

[5] Proceedings of the Third NASA Ada User's Symposium, November 1990, NASA-JSC.

[6] Keller, John, "When Avoiding Ada Is The Right Thing To Do", Military and Aerospace Electronics, February 1991.

[7] Keller, John, "Ada Is Law, So What Else Is New?", Military and Aerospace Electronics, January 1991.

[8] Kernighan, Brian W. and Ritchie, Dennis M., The C Programming Language, Prentice Hall, 1978

[9] Anderson, Paul and Anderson, Gail, Moving on to C++, UnixWorld, January 1991

[10] Horowithz, Ellis, Fundamentals of Programming Languages, Computer Science Press, 1984

[11] Mike Feldman, George Washington University

Experiment: EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.26 %
2	4.00	16	64.00	3.26 %
3	8.00	8	64.00	3.26 %
4	16.00	4	64.00	3.26 %
5	32.00	2	64.00	3.26 %
			320.00	16.32 %

Experiment step size: 1.63 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.26 %
2	4.00	16	64.00	3.26 %
3	8.00	8	64.00	3.26 %
4	16.00	4	64.00	3.26 %
5	48.00	2	96.00	4.90 %
			352.00	17.95 %

Experiment step size: 1.63 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	20.833	240	120	120	5.584

HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

Experiment: EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.26 %
2	4.00	16	64.00	3.26 %
3	8.00	8	64.00	3.26 %
4	16.00	4	64.00	3.26 %
5	32.00	2	64.00	3.26 %
			320.00	16.32 %

Experiment step size: 1.63 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Last test with no missed/skipped deadlines:
See preceding summary of test 1

Test when deadlines first missed/skipped:

Experiment: EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.26 %
2	4.00	16	64.00	3.26 %
3	8.00	8	64.00	3.26 %
4	16.00	4	64.00	3.26 %
5	48.00	2	96.00	4.90 %
			352.00	17.95 %

Experiment step size: 1.63 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	20.833	240	120	120	5.584

Final test performed:
See preceding summary of test 2

Benchmark : Hartstone Benchmark, version 1.0
Compiler : Masscomp 6350 - C3Ada version 1.0
Target : Masscomp 6350 - dual 33 MHz 68030

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

Test 1 of Experiment 1

Raw (non-tasking) benchmark speed in KWIPS: 1960.79

Full task set:

Total Tasks	Deadlines Per Second	Task Set Utilization	Total KWIPS
5	62.00	16.32 %	320.00

Highest-frequency task:

Period (msec)	Deadlines Per Second	Task Utilization	Task KWIPS
31.250	32.00	3.26 %	64.00

Experiment step size: 1.63 %

END OF HARTSTONE BENCHMARK SUMMARY RESULTS

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
			320.00	15.89 %

Experiment step size: 1.59 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	316	2	2	19.257

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.20	32	70.40	3.50 %
2	4.40	16	70.40	3.50 %
3	8.80	8	70.40	3.50 %
4	17.60	4	70.40	3.50 %
5	35.20	2	70.40	3.50 %
			352.00	17.48 %

Experiment step size: 1.59 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	454.545	23	0	0	0.000
2	227.273	45	0	0	0.000
3	113.636	89	0	0	0.000
4	56.818	177	0	0	0.000
5	28.409	353	0	0	0.000

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 3 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.40	32	76.80	3.81 %
2	4.80	16	76.80	3.81 %
3	9.60	8	76.80	3.81 %
4	19.20	4	76.80	3.81 %
5	38.40	2	76.80	3.81 %
			384.00	19.07 %

Experiment step size: 1.59 %

Test 3 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	416.667	25	0	0	0.000
2	208.333	49	0	0	0.000
3	104.167	97	0	0	0.000
4	52.083	193	0	0	0.000
5	26.042	383	1	1	4.700

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 4 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.60	32	83.20	4.13 %
2	5.20	16	83.20	4.13 %
3	10.40	8	83.20	4.13 %
4	20.80	4	83.20	4.13 %
5	41.60	2	83.20	4.13 %
			416.00	20.66 %

Experiment step size: 1.59 %

Test 4 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	384.615	26	0	0	0.000
2	192.308	52	0	0	0.000
3	96.154	104	0	0	0.000
4	48.077	196	6	6	0.214
5	24.038	290	63	63	1.537

HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
			320.00	15.89 %

Experiment step size: 1.59 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	316	2	2	19.257

Last test with no missed/skipped deadlines:

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.20	32	70.40	3.50 %
2	4.40	16	70.40	3.50 %
3	8.80	8	70.40	3.50 %
4	17.60	4	70.40	3.50 %
5	35.20	2	70.40	3.50 %
			352.00	17.48 %

Experiment step size: 1.59 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	454.545	23	0	0	0.000
2	227.273	45	0	0	0.000
3	113.636	89	0	0	0.000
4	56.818	177	0	0	0.000
5	28.409	353	0	0	0.000

Test when deadlines first missed/skipped:
See preceding summary of test 1

Final test performed:

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 4 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.60	32	83.20	4.13 %
2	5.20	16	83.20	4.13 %
3	10.40	8	83.20	4.13 %
4	20.80	4	83.20	4.13 %
5	41.60	2	83.20	4.13 %
			416.00	20.66 %

Experiment step size: 1.59 %

Test 4 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	384.615	26	0	0	0.000
2	192.308	52	0	0	0.000
3	96.154	104	0	0	0.000
4	48.077	196	6	6	0.214
5	24.038	290	63	63	1.537

Benchmark : Hartstone Benchmark, version 1.0
Compiler : Masscomp 6350 - C3Ada version 1.0
Target : Masscomp 6350 - dual 33 MHz 68030

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

Test 2 of Experiment 2

Raw (non-tasking) benchmark speed in KWIPS: 2013.42

Full task set:

Total Tasks	Deadlines Per Second	Task Set Utilization	Total KWIPS
5	68.20	17.48 %	352.00

Highest-frequency task:

Period (msec)	Deadlines Per Second	Task Utilization	Task KWIPS
28.409	35.20	3.50 %	70.40

Experiment step size: 1.59 %

END OF HARTSTONE BENCHMARK SUMMARY RESULTS

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.26 %
2	4.00	16	64.00	3.26 %
3	8.00	8	64.00	3.26 %
4	16.00	4	64.00	3.26 %
5	32.00	2	64.00	3.26 %
			320.00	16.32 %

Experiment step size: 3.16 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	33	66.00	3.37 %
2	4.00	17	68.00	3.47 %
3	8.00	9	72.00	3.67 %
4	16.00	5	80.00	4.08 %
5	32.00	3	96.00	4.90 %
			382.00	19.48 %

Experiment step size: 3.16 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 3 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	34	68.00	3.47 %
2	4.00	18	72.00	3.67 %
3	8.00	10	80.00	4.08 %
4	16.00	6	96.00	4.90 %
5	32.00	4	128.00	6.53 %
			444.00	22.64 %

Experiment step size: 3.16 %

Test 3 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	11.963

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 4 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	35	70.00	3.57 %
2	4.00	19	76.00	3.88 %
3	8.00	11	88.00	4.49 %
4	16.00	7	112.00	5.71 %
5	32.00	5	160.00	8.16 %
			506.00	25.81 %

Experiment step size: 3.16 %

Test 4 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 5 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	36	72.00	3.67 %
2	4.00	20	80.00	4.08 %
3	8.00	12	96.00	4.90 %
4	16.00	8	128.00	6.53 %
5	32.00	6	192.00	9.79 %
			568.00	28.97 %

Experiment step size: 3.16 %

Test 5 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 6 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	37	74.00	3.77 %
2	4.00	21	84.00	4.28 %
3	8.00	13	104.00	5.30 %
4	16.00	9	144.00	7.34 %
5	32.00	7	224.00	11.42 %
			630.00	32.13 %

Experiment step size: 3.16 %

Test 6 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 7 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	38	76.00	3.88 %
2	4.00	22	88.00	4.49 %
3	8.00	14	112.00	5.71 %
4	16.00	10	160.00	8.16 %
5	32.00	8	256.00	13.06 %
			692.00	35.29 %

Experiment step size: 3.16 %

Test 7 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 8 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	39	78.00	3.98 %
2	4.00	23	92.00	4.69 %
3	8.00	15	120.00	6.12 %
4	16.00	11	176.00	8.98 %
5	32.00	9	288.00	14.69 %
			754.00	38.45 %

Experiment step size: 3.16 %

Test 8 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	314	3	3	11.617

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 9 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	40	80.00	4.08 %
2	4.00	24	96.00	4.90 %
3	8.00	16	128.00	6.53 %
4	16.00	12	192.00	9.79 %
5	32.00	10	320.00	16.32 %
			816.00	41.62 %

Experiment step size: 3.16 %

Test 9 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	241	40	39	0.488

HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.26 %
2	4.00	16	64.00	3.26 %
3	8.00	8	64.00	3.26 %
4	16.00	4	64.00	3.26 %
5	32.00	2	64.00	3.26 %
			320.00	16.32 %

Experiment step size: 3.16 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Last test with no missed/skipped deadlines:

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 7 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	38	76.00	3.88 %
2	4.00	22	88.00	4.49 %
3	8.00	14	112.00	5.71 %
4	16.00	10	160.00	8.16 %
5	32.00	8	256.00	13.06 %
			692.00	35.29 %

Experiment step size: 3.16 %

Test 7 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Test when deadlines first missed/skipped:

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 3 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	34	68.00	3.47 %
2	4.00	18	72.00	3.67 %
3	8.00	10	80.00	4.08 %
4	16.00	6	96.00	4.90 %
5	32.00	4	128.00	6.53 %
			444.00	22.64 %

Experiment step size: 3.16 %

Test 3 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	11.963

Final test performed:

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1960.79

Test 9 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	40	80.00	4.08 %
2	4.00	24	96.00	4.90 %
3	8.00	16	128.00	6.53 %
4	16.00	12	192.00	9.79 %
5	32.00	10	320.00	16.32 %
			816.00	41.62 %

Experiment step size: 3.16 %

Test 9 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	241	40	39	0.488

Benchmark : Hartstone Benchmark, version 1.0
Compiler : Masscomp 6350 - C3Ada version 1.0
Target : Masscomp 6350 - dual 33 MHz 68030

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

Test 7 of Experiment 3

Raw (non-tasking) benchmark speed in KWIPS: 1960.79

Full task set:

Total Tasks	Deadlines Per Second	Task Set Utilization	Total KWIPS
5	62.00	35.29 %	692.00

Highest-frequency task:

Period (msec)	Deadlines Per Second	Task Utilization	Task KWIPS
31.250	32.00	13.06 %	256.00

Experiment step size: 3.16 %

END OF HARTSTONE BENCHMARK SUMMARY RESULTS

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
			320.00	15.89 %

Experiment step size: 3.18 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
			384.00	19.07 %

Experiment step size: 3.18 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 3 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
			448.00	22.25 %

Experiment step size: 3.18 %

Test 3 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 4 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
8	8.00	8	64.00	3.18 %
			512.00	25.43 %

Experiment step size: 3.18 %

Test 4 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	312	4	4	0.763
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 5 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
8	8.00	8	64.00	3.18 %
9	8.00	8	64.00	3.18 %
			576.00	28.61 %

Experiment step size: 3.18 %

Test 5 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	296	12	12	2.075
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 6 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
8	8.00	8	64.00	3.18 %
9	8.00	8	64.00	3.18 %
10	8.00	8	64.00	3.18 %
			640.00	31.79 %

Experiment step size: 3.18 %

Test 6 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	295	13	12	1.282
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 7 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
8	8.00	8	64.00	3.18 %
9	8.00	8	64.00	3.18 %
10	8.00	8	64.00	3.18 %
11	8.00	8	64.00	3.18 %
			704.00	34.97 %

Experiment step size: 3.18 %

Test 7 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	301	10	9	0.488
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 8 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
8	8.00	8	64.00	3.18 %
9	8.00	8	64.00	3.18 %
10	8.00	8	64.00	3.18 %
11	8.00	8	64.00	3.18 %
12	8.00	8	64.00	3.18 %
			768.00	38.14 %

Experiment step size: 3.18 %

Test 8 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	71	1	8	941.345
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	62	2	2	941.345
12	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 9 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
8	8.00	8	64.00	3.18 %
9	8.00	8	64.00	3.18 %
10	8.00	8	64.00	3.18 %
11	8.00	8	64.00	3.18 %
12	8.00	8	64.00	3.18 %
13	8.00	8	64.00	3.18 %
			832.00	41.32 %

Experiment step size: 3.18 %

Test 9 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	302	9	9	2.075
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 10 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
8	8.00	8	64.00	3.18 %
9	8.00	8	64.00	3.18 %
10	8.00	8	64.00	3.18 %
11	8.00	8	64.00	3.18 %
12	8.00	8	64.00	3.18 %
13	8.00	8	64.00	3.18 %
14	8.00	8	64.00	3.18 %
			896.00	44.50 %

Experiment step size: 3.18 %

Test 10 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	259	31	30	0.244
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000

HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

==========

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
			320.00	15.89 %

Experiment step size: 3.18 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

==========

Last test with no missed/skipped deadlines:

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 3 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
			448.00	22.25 %

Experiment step size: 3.18 %

Test 3 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000

Test when deadlines first missed/skipped:

==========

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 4 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
8	8.00	8	64.00	3.18 %
			512.00	25.43 %

Experiment step size: 3.18 %

Test 4 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	312	4	4	0.763
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000

==========

Final test performed:

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 2013.42

Test 10 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	3.18 %
2	4.00	16	64.00	3.18 %
3	8.00	8	64.00	3.18 %
4	16.00	4	64.00	3.18 %
5	32.00	2	64.00	3.18 %
6	8.00	8	64.00	3.18 %
7	8.00	8	64.00	3.18 %
8	8.00	8	64.00	3.18 %
9	8.00	8	64.00	3.18 %
10	8.00	8	64.00	3.18 %
11	8.00	8	64.00	3.18 %
12	8.00	8	64.00	3.18 %
13	8.00	8	64.00	3.18 %
14	8.00	8	64.00	3.18 %
			896.00	44.50 %

Experiment step size: 3.18 %

Test 10 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	259	31	30	0.244
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000

Benchmark : Hartstone Benchmark, version 1.0
Compiler : Masscomp 6350 - C3Ada version 1.0
Target : Masscomp 6350 - dual 33 MHz 68030

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

Test 3 of Experiment 4

Raw (non-tasking) benchmark speed in KWIPS: 2013.42

Full task set:

Total Tasks	Deadlines Per Second	Task Set Utilization	Total KWIPS
7	78.00	22.25 %	448.00

Highest-frequency task:

Period (msec)	Deadlines Per Second	Task Utilization	Task KWIPS
31.250	32.00	3.18 %	64.00

Experiment step size: 3.18 %

END OF HARTSTONE BENCHMARK SUMMARY RESULTS

Experiment: EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
			320.00	7.15 %

Experiment step size: 0.71 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	48.00	2	96.00	2.14 %
			352.00	7.86 %

Experiment step size: 0.71 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	20.833	480	0	0	0.000

Experiment: EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 3 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	64.00	2	128.00	2.86 %
			384.00	8.58 %

Experiment step size: 0.71 %

Test 3 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	15.625	477	81	82	1.988

HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

Experiment: EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
			320.00	7.15 %

Experiment step size: 0.71 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

==========

Last test with no missed/skipped deadlines:

Experiment: EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	48.00	2	96.00	2.14 %
			352.00	7.86 %

Experiment step size: 0.71 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	20.833	480	0	0	0.000

Test when deadlines first missed/skipped:

Experiment: EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 3 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	64.00	2	128.00	2.86 %
			384.00	8.58 %

Experiment step size: 0.71 %

Test 3 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	15.625	477	81	82	1.988

Final test performed:
See preceding summary of test 3

==========

Benchmark : Hartstone Benchmark, version 1.0
Compiler : Verdix 6.0 -> Sun SPARC
Target : Sun SPARC Station 1+ (25 MHz) - multiuser mode

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

Test 2 of Experiment 1

Raw (non-tasking) benchmark speed in KWIPS: 4476.28

Full task set:

Total Tasks	Deadlines Per Second	Task Set Utilization	Total KWIPS
5	78.00	7.86 %	352.00

Highest-frequency task:

Period (msec)	Deadlines Per Second	Task Utilization	Task KWIPS
20.833	48.00	2.14 %	96.00

Experiment step size: 0.71 %

==========

END OF HARTSTONE BENCHMARK SUMMARY RESULTS

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.47 %
2	4.00	16	64.00	1.47 %
3	8.00	8	64.00	1.47 %
4	16.00	4	64.00	1.47 %
5	32.00	2	64.00	1.47 %
			320.00	7.33 %

Experiment step size: 0.73 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.20	32	70.40	1.61 %
2	4.40	16	70.40	1.61 %
3	8.80	8	70.40	1.61 %
4	17.60	4	70.40	1.61 %
5	35.20	2	70.40	1.61 %
			352.00	8.06 %

Experiment step size: 0.73 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	454.545	23	0	0	0.000
2	227.273	45	0	0	0.000
3	113.636	88	0	0	0.000
4	56.818	177	0	0	0.000
5	28.409	350	1	1	4.000

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 3 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.40	32	76.80	1.76 %
2	4.80	16	76.80	1.76 %
3	9.60	8	76.80	1.76 %
4	19.20	4	76.80	1.76 %
5	38.40	2	76.80	1.76 %
			384.00	8.80 %

Experiment step size: 0.73 %

Test 3 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	416.667	25	0	0	0.000
2	208.333	48	0	0	0.000
3	104.167	97	0	0	0.000
4	52.083	192	0	0	0.000
5	26.042	385	0	0	0.000

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 4 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.60	32	83.20	1.91 %
2	5.20	16	83.20	1.91 %
3	10.40	8	83.20	1.91 %
4	20.80	4	83.20	1.91 %
5	41.60	2	83.20	1.91 %
			416.00	9.53 %

Experiment step size: 0.73 %

Test 4 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	384.615	26	0	0	0.000
2	192.308	53	0	0	0.000
3	96.154	104	0	0	0.000
4	48.077	208	0	0	0.000
5	24.038	414	1	1	1.000

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 5 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.80	32	89.60	2.05 %
2	5.60	16	89.60	2.05 %
3	11.20	8	89.60	2.05 %
4	22.40	4	89.60	2.05 %
5	44.80	2	89.60	2.05 %
			448.00	10.26 %

Experiment step size: 0.73 %

Test 5 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	357.143	28	0	0	0.000
2	178.571	56	0	0	0.000
3	89.286	113	0	0	0.000
4	44.643	224	0	0	0.000
5	22.321	448	0	0	0.000

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 6 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	3.00	32	96.00	2.20 %
2	6.00	16	96.00	2.20 %
3	12.00	8	96.00	2.20 %
4	24.00	4	96.00	2.20 %
5	48.00	2	96.00	2.20 %
			480.00	11.00 %

Experiment step size: 0.73 %

Test 6 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	333.333	30	0	0	0.000
2	166.667	61	0	0	0.000
3	83.333	120	0	0	0.000
4	41.667	239	1	1	6.000
5	20.833	477	1	2	25.000

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 7 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	3.20	32	102.40	2.35 %
2	6.40	16	102.40	2.35 %
3	12.80	8	102.40	2.35 %
4	25.60	4	102.40	2.35 %
5	51.20	2	102.40	2.35 %
			512.00	11.73 %

Experiment step size: 0.73 %

Test 7 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	312.500	32	0	0	0.000
2	156.250	64	0	0	0.000
3	78.125	128	0	0	0.000
4	39.063	256	0	0	0.000
5	19.531	512	0	0	0.000

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 8 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	3.40	32	108.80	2.49 %
2	6.80	16	108.80	2.49 %
3	13.60	8	108.80	2.49 %
4	27.20	4	108.80	2.49 %
5	54.40	2	108.80	2.49 %
			544.00	12.46 %

Experiment step size: 0.73 %

Test 8 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	294.118	35	0	0	0.000
2	147.059	68	0	0	0.000
3	73.529	137	0	0	0.000
4	36.765	271	1	1	11.000
5	18.382	447	48	49	1.688

HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.47 %
2	4.00	16	64.00	1.47 %
3	8.00	8	64.00	1.47 %
4	16.00	4	64.00	1.47 %
5	32.00	2	64.00	1.47 %
			320.00	7.33 %

Experiment step size: 0.73 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Last test with no missed/skipped deadlines:

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 7 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	3.20	32	102.40	2.35 %
2	6.40	16	102.40	2.35 %
3	12.80	8	102.40	2.35 %
4	25.60	4	102.40	2.35 %
5	51.20	2	102.40	2.35 %
			512.00	11.73 %

Experiment step size: 0.73 %

Test 7 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	312.500	32	0	0	0.000
2	156.250	64	0	0	0.000
3	78.125	128	0	0	0.000
4	39.063	256	0	0	0.000
5	19.531	512	0	0	0.000

Test when deadlines first missed/skipped:

==========

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.20	32	70.40	1.61 %
2	4.40	16	70.40	1.61 %
3	8.80	8	70.40	1.61 %
4	17.60	4	70.40	1.61 %
5	35.20	2	70.40	1.61 %
			352.00	8.06 %

Experiment step size: 0.73 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	454.545	23	0	0	0.000
2	227.273	45	0	0	0.000
3	113.636	88	0	0	0.000
4	56.818	177	0	0	0.000
5	28.409	350	1	1	4.000

==========

Final test performed:

==========

Experiment: EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4364.91

Test 8 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	3.40	32	108.80	2.49 %
2	6.80	16	108.80	2.49 %
3	13.60	8	108.80	2.49 %
4	27.20	4	108.80	2.49 %
5	54.40	2	108.80	2.49 %
			544.00	12.46 %

Experiment step size: 0.73 %

Test 8 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	294.118	35	0	0	0.000
2	147.059	68	0	0	0.000
3	73.529	137	0	0	0.000
4	36.765	271	1	1	11.000
5	18.382	447	48	49	1.688

==========

Benchmark : Hartstone Benchmark, version 1.0
Compiler : Verdix 6.0 -> Sun SPARC
Target : Sun SPARC Station 1+ (25 MHz) - multiuser mode

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

Test 7 of Experiment 2

Raw (non-tasking) benchmark speed in KWIPS: 4364.91

Full task set:

Total Tasks	Deadlines Per Second	Task Set Utilization	Total KWIPS
5	99.20	11.73 %	512.00

Highest-frequency task:

Period (msec)	Deadlines Per Second	Task Utilization	Task KWIPS
19.531	51.20	2.35 %	102.40

Experiment step size: 0.73 %

END OF HARTSTONE BENCHMARK SUMMARY RESULTS

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.48 %
2	4.00	16	64.00	1.48 %
3	8.00	8	64.00	1.48 %
4	16.00	4	64.00	1.48 %
5	32.00	2	64.00	1.48 %
			320.00	7.40 %

Experiment step size: 1.43 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	33	66.00	1.53 %
2	4.00	17	68.00	1.57 %
3	8.00	9	72.00	1.67 %
4	16.00	5	80.00	1.85 %
5	32.00	3	96.00	2.22 %
			382.00	8.84 %

Experiment step size: 1.43 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	13.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 3 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	34	68.00	1.57 %
2	4.00	18	72.00	1.67 %
3	8.00	10	80.00	1.85 %
4	16.00	6	96.00	2.22 %
5	32.00	4	128.00	2.96 %
			444.00	10.27 %

Experiment step size: 1.43 %

Test 3 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 4 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	35	70.00	1.62 %
2	4.00	19	76.00	1.76 %
3	8.00	11	88.00	2.04 %
4	16.00	7	112.00	2.59 %
5	32.00	5	160.00	3.70 %
			506.00	11.70 %

Experiment step size: 1.43 %

Test 4 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 5 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	36	72.00	1.67 %
2	4.00	20	80.00	1.85 %
3	8.00	12	96.00	2.22 %
4	16.00	8	128.00	2.96 %
5	32.00	6	192.00	4.44 %
			568.00	13.14 %

Experiment step size: 1.43 %

Test 5 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 6 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	37	74.00	1.71 %
2	4.00	21	84.00	1.94 %
3	8.00	13	104.00	2.41 %
4	16.00	9	144.00	3.33 %
5	32.00	7	224.00	5.18 %
			630.00	14.57 %

Experiment step size: 1.43 %

Test 6 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	7.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 7 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	38	76.00	1.76 %
2	4.00	22	88.00	2.04 %
3	8.00	14	112.00	2.59 %
4	16.00	10	160.00	3.70 %
5	32.00	8	256.00	5.92 %
			692.00	16.01 %

Experiment step size: 1.43 %

Test 7 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 8 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	39	78.00	1.80 %
2	4.00	23	92.00	2.13 %
3	8.00	15	120.00	2.78 %
4	16.00	11	176.00	4.07 %
5	32.00	9	288.00	6.66 %
			754.00	17.44 %

Experiment step size: 1.43 %

Test 8 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 9 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	40	80.00	1.85 %
2	4.00	24	96.00	2.22 %
3	8.00	16	128.00	2.96 %
4	16.00	12	192.00	4.44 %
5	32.00	10	320.00	7.40 %
			816.00	18.87 %

Experiment step size: 1.43 %

Test 9 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 10 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	41	82.00	1.90 %
2	4.00	25	100.00	2.31 %
3	8.00	17	136.00	3.15 %
4	16.00	13	208.00	4.81 %
5	32.00	11	352.00	8.14 %
			878.00	20.31 %

Experiment step size: 1.43 %

Test 10 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 11 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	42	84.00	1.94 %
2	4.00	26	104.00	2.41 %
3	8.00	18	144.00	3.33 %
4	16.00	14	224.00	5.18 %
5	32.00	12	384.00	8.88 %
			940.00	21.74 %

Experiment step size: 1.43 %

Test 11 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 12 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	43	86.00	1.99 %
2	4.00	27	108.00	2.50 %
3	8.00	19	152.00	3.52 %
4	16.00	15	240.00	5.55 %
5	32.00	13	416.00	9.62 %
			1002.00	23.18 %

Experiment step size: 1.43 %

Test 12 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 13 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	44	88.00	2.04 %
2	4.00	28	112.00	2.59 %
3	8.00	20	160.00	3.70 %
4	16.00	16	256.00	5.92 %
5	32.00	14	448.00	10.36 %
			1064.00	24.61 %

Experiment step size: 1.43 %

Test 13 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	8.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 14 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	45	90.00	2.08 %
2	4.00	29	116.00	2.68 %
3	8.00	21	168.00	3.89 %
4	16.00	17	272.00	6.29 %
5	32.00	15	480.00	11.10 %
			1126.00	26.04 %

Experiment step size: 1.43 %

Test 14 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 15 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	46	92.00	2.13 %
2	4.00	30	120.00	2.78 %
3	8.00	22	176.00	4.07 %
4	16.00	18	288.00	6.66 %
5	32.00	16	512.00	11.84 %
			1188.00	27.48 %

Experiment step size: 1.43 %

Test 15 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	20.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 16 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	47	94.00	2.17 %
2	4.00	31	124.00	2.87 %
3	8.00	23	184.00	4.26 %
4	16.00	19	304.00	7.03 %
5	32.00	17	544.00	12.58 %
			1250.00	28.91 %

Experiment step size: 1.43 %

Test 16 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 17 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	48	96.00	2.22 %
2	4.00	32	128.00	2.96 %
3	8.00	24	192.00	4.44 %
4	16.00	20	320.00	7.40 %
5	32.00	18	576.00	13.32 %
			1312.00	30.35 %

Experiment step size: 1.43 %

Test 17 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	20.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 18 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	49	98.00	2.27 %
2	4.00	33	132.00	3.05 %
3	8.00	25	200.00	4.63 %
4	16.00	21	336.00	7.77 %
5	32.00	19	608.00	14.06 %
			1374.00	31.78 %

Experiment step size: 1.43 %

Test 18 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 19 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	50	100.00	2.31 %
2	4.00	34	136.00	3.15 %
3	8.00	26	208.00	4.81 %
4	16.00	22	352.00	8.14 %
5	32.00	20	640.00	14.80 %
			1436.00	33.21 %

Experiment step size: 1.43 %

Test 19 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	3.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 20 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	51	102.00	2.36 %
2	4.00	35	140.00	3.24 %
3	8.00	27	216.00	5.00 %
4	16.00	23	368.00	8.51 %
5	32.00	21	672.00	15.54 %
			1498.00	34.65 %

Experiment step size: 1.43 %

Test 20 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 21 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	52	104.00	2.41 %
2	4.00	36	144.00	3.33 %
3	8.00	28	224.00	5.18 %
4	16.00	24	384.00	8.88 %
5	32.00	22	704.00	16.28 %
			1560.00	36.08 %

Experiment step size: 1.43 %

Test 21 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	1.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 22 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	53	106.00	2.45 %
2	4.00	37	148.00	3.42 %
3	8.00	29	232.00	5.37 %
4	16.00	25	400.00	9.25 %
5	32.00	23	736.00	17.02 %
			1622.00	37.52 %

Experiment step size: 1.43 %

Test 22 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 23 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	54	108.00	2.50 %
2	4.00	38	152.00	3.52 %
3	8.00	30	240.00	5.55 %
4	16.00	26	416.00	9.62 %
5	32.00	24	768.00	17.76 %
			1684.00	38.95 %

Experiment step size: 1.43 %

Test 23 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 24 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	55	110.00	2.54 %
2	4.00	39	156.00	3.61 %
3	8.00	31	248.00	5.74 %
4	16.00	27	432.00	9.99 %
5	32.00	25	800.00	18.50 %
			1746.00	40.38 %

Experiment step size: 1.43 %

Test 24 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 25 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	56	112.00	2.59 %
2	4.00	40	160.00	3.70 %
3	8.00	32	256.00	5.92 %
4	16.00	28	448.00	10.36 %
5	32.00	26	832.00	19.24 %
			1808.00	41.82 %

Experiment step size: 1.43 %

Test 25 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

C-3

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 26 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	57	114.00	2.64 %
2	4.00	41	164.00	3.79 %
3	8.00	33	264.00	6.11 %
4	16.00	29	464.00	10.73 %
5	32.00	27	864.00	19.98 %
			1870.00	43.25 %

Experiment step size: 1.43 %

Test 26 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 27 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	58	116.00	2.68 %
2	4.00	42	168.00	3.89 %
3	8.00	34	272.00	6.29 %
4	16.00	30	480.00	11.10 %
5	32.00	28	896.00	20.72 %
			1932.00	44.69 %

Experiment step size: 1.43 %

Test 27 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 28 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	59	118.00	2.73 %
2	4.00	43	172.00	3.98 %
3	8.00	35	280.00	6.48 %
4	16.00	31	496.00	11.47 %
5	32.00	29	928.00	21.46 %
			1994.00	46.12 %

Experiment step size: 1.43 %

Test 28 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	2.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 29 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	60	120.00	2.78 %
2	4.00	44	176.00	4.07 %
3	8.00	36	288.00	6.66 %
4	16.00	32	512.00	11.84 %
5	32.00	30	960.00	22.20 %
			2056.00	47.56 %

Experiment step size: 1.43 %

Test 29 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 30 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	61	122.00	2.82 %
2	4.00	45	180.00	4.16 %
3	8.00	37	296.00	6.85 %
4	16.00	33	528.00	12.21 %
5	32.00	31	992.00	22.94 %
			2118.00	48.99 %

Experiment step size: 1.43 %

Test 30 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	158	1	1	2.000
5	31.250	318	1	1	22.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 31 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	62	124.00	2.87 %
2	4.00	46	184.00	4.26 %
3	8.00	38	304.00	7.03 %
4	16.00	34	544.00	12.58 %
5	32.00	32	1024.00	23.69 %
			2180.00	50.42 %

Experiment step size: 1.43 %

Test 31 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 32 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	63	126.00	2.91 %
2	4.00	47	188.00	4.35 %
3	8.00	39	312.00	7.22 %
4	16.00	35	560.00	12.95 %
5	32.00	33	1056.00	24.43 %
			2242.00	51.86 %

Experiment step size: 1.43 %

Test 32 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	21.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 33 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	64	128.00	2.96 %
2	4.00	48	192.00	4.44 %
3	8.00	40	320.00	7.40 %
4	16.00	36	576.00	13.32 %
5	32.00	34	1088.00	25.17 %
			2304.00	53.29 %

Experiment step size: 1.43 %

Test 33 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 34 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	65	130.00	3.01 %
2	4.00	49	196.00	4.53 %
3	8.00	41	328.00	7.59 %
4	16.00	37	592.00	13.69 %
5	32.00	35	1120.00	25.91 %
			2366.00	54.73 %

Experiment step size: 1.43 %

Test 34 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	6.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 35 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	66	132.00	3.05 %
2	4.00	50	200.00	4.63 %
3	8.00	42	336.00	7.77 %
4	16.00	38	608.00	14.06 %
5	32.00	36	1152.00	26.65 %
			2428.00	56.16 %

Experiment step size: 1.43 %

Test 35 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 36 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	67	134.00	3.10 %
2	4.00	51	204.00	4.72 %
3	8.00	43	344.00	7.96 %
4	16.00	39	624.00	14.43 %
5	32.00	37	1184.00	27.39 %
			2490.00	57.59 %

Experiment step size: 1.43 %

Test 36 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	26.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 37 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	68	136.00	3.15 %
2	4.00	52	208.00	4.81 %
3	8.00	44	352.00	8.14 %
4	16.00	40	640.00	14.80 %
5	32.00	38	1216.00	28.13 %
			2552.00	59.03 %

Experiment step size: 1.43 %

Test 37 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 38 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	69	138.00	3.19 %
2	4.00	53	212.00	4.90 %
3	8.00	45	360.00	8.33 %
4	16.00	41	656.00	15.17 %
5	32.00	39	1248.00	28.87 %
			2614.00	60.46 %

Experiment step size: 1.43 %

Test 38 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 39 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	70	140.00	3.24 %
2	4.00	54	216.00	5.00 %
3	8.00	46	368.00	8.51 %
4	16.00	42	672.00	15.54 %
5	32.00	40	1280.00	29.61 %
			2676.00	61.90 %

Experiment step size: 1.43 %

Test 39 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 40 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	71	142.00	3.28 %
2	4.00	55	220.00	5.09 %
3	8.00	47	376.00	8.70 %
4	16.00	43	688.00	15.91 %
5	32.00	41	1312.00	30.35 %
			2738.00	63.33 %

Experiment step size: 1.43 %

Test 40 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 41 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	72	144.00	3.33 %
2	4.00	56	224.00	5.18 %
3	8.00	48	384.00	8.88 %
4	16.00	44	704.00	16.28 %
5	32.00	42	1344.00	31.09 %
			2800.00	64.76 %

Experiment step size: 1.43 %

Test 41 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 42 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	73	146.00	3.38 %
2	4.00	57	228.00	5.27 %
3	8.00	49	392.00	9.07 %
4	16.00	45	720.00	16.65 %
5	32.00	43	1376.00	31.83 %
			2862.00	66.20 %

Experiment step size: 1.43 %

Test 42 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 43 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	74	148.00	3.42 %
2	4.00	58	232.00	5.37 %
3	8.00	50	400.00	9.25 %
4	16.00	46	736.00	17.02 %
5	32.00	44	1408.00	32.57 %
			2924.00	67.63 %

Experiment step size: 1.43 %

Test 43 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	78	1	1	2.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	26.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 44 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	75	150.00	3.47 %
2	4.00	59	236.00	5.46 %
3	8.00	51	408.00	9.44 %
4	16.00	47	752.00	17.39 %
5	32.00	45	1440.00	33.31 %
			2986.00	69.07 %

Experiment step size: 1.43 %

Test 44 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 45 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	76	152.00	3.52 %
2	4.00	60	240.00	5.55 %
3	8.00	52	416.00	9.62 %
4	16.00	48	768.00	17.76 %
5	32.00	46	1472.00	34.05 %
			3048.00	70.50 %

Experiment step size: 1.43 %

Test 45 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	158	1	1	25.000
5	31.250	318	1	1	28.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 46 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	77	154.00	3.56 %
2	4.00	61	244.00	5.64 %
3	8.00	53	424.00	9.81 %
4	16.00	49	784.00	18.13 %
5	32.00	47	1504.00	34.79 %
			3110.00	71.93 %

Experiment step size: 1.43 %

Test 46 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 47 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	78	156.00	3.61 %
2	4.00	62	248.00	5.74 %
3	8.00	54	432.00	9.99 %
4	16.00	50	800.00	18.50 %
5	32.00	48	1536.00	35.53 %
			3172.00	73.37 %

Experiment step size: 1.43 %

Test 47 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	158	1	1	27.000
5	31.250	318	1	1	20.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 48 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	79	158.00	3.65 %
2	4.00	63	252.00	5.83 %
3	8.00	55	440.00	10.18 %
4	16.00	51	816.00	18.87 %
5	32.00	49	1568.00	36.27 %
			3234.00	74.80 %

Experiment step size: 1.43 %

Test 48 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 49 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	80	160.00	3.70 %
2	4.00	64	256.00	5.92 %
3	8.00	56	448.00	10.36 %
4	16.00	52	832.00	19.24 %
5	32.00	50	1600.00	37.01 %
			3296.00	76.24 %

Experiment step size: 1.43 %

Test 49 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	9.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 50 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	81	162.00	3.75 %
2	4.00	65	260.00	6.01 %
3	8.00	57	456.00	10.55 %
4	16.00	53	848.00	19.61 %
5	32.00	51	1632.00	37.75 %
			3358.00	77.67 %

Experiment step size: 1.43 %

Test 50 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 51 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	82	164.00	3.79 %
2	4.00	66	264.00	6.11 %
3	8.00	58	464.00	10.73 %
4	16.00	54	864.00	19.98 %
5	32.00	52	1664.00	38.49 %
			3420.00	79.10 %

Experiment step size: 1.43 %

Test 51 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	158	1	1	2.000
5	31.250	318	1	1	17.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 52 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	83	166.00	3.84 %
2	4.00	67	268.00	6.20 %
3	8.00	59	472.00	10.92 %
4	16.00	55	880.00	20.35 %
5	32.00	53	1696.00	39.23 %
			3482.00	80.54 %

Experiment step size: 1.43 %

Test 52 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 53 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	84	168.00	3.89 %
2	4.00	68	272.00	6.29 %
3	8.00	60	480.00	11.10 %
4	16.00	56	896.00	20.72 %
5	32.00	54	1728.00	39.97 %
			3544.00	81.97 %

Experiment step size: 1.43 %

Test 53 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 54 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	85	170.00	3.93 %
2	4.00	69	276.00	6.38 %
3	8.00	61	488.00	11.29 %
4	16.00	57	912.00	21.09 %
5	32.00	55	1760.00	40.71 %
			3606.00	83.41 %

Experiment step size: 1.43 %

Test 54 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 55 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	86	172.00	3.98 %
2	4.00	70	280.00	6.48 %
3	8.00	62	496.00	11.47 %
4	16.00	58	928.00	21.46 %
5	32.00	56	1792.00	41.45 %
			3668.00	84.84 %

Experiment step size: 1.43 %

Test 55 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 56 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	87	174.00	4.02 %
2	4.00	71	284.00	6.57 %
3	8.00	63	504.00	11.66 %
4	16.00	59	944.00	21.83 %
5	32.00	57	1824.00	42.19 %
			3730.00	86.27 %

Experiment step size: 1.43 %

Test 56 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	302	9	9	0.222

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 57 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	88	176.00	4.07 %
2	4.00	72	288.00	6.66 %
3	8.00	64	512.00	11.84 %
4	16.00	60	960.00	22.20 %
5	32.00	58	1856.00	42.93 %
			3792.00	87.71 %

Experiment step size: 1.43 %

Test 57 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	18	1	1	1.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 58 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	89	178.00	4.12 %
2	4.00	73	292.00	6.75 %
3	8.00	65	520.00	12.03 %
4	16.00	61	976.00	22.57 %
5	32.00	59	1888.00	43.67 %
			3854.00	89.14 %

Experiment step size: 1.43 %

Test 58 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	38	1	1	70.000
3	125.000	80	0	0	0.000
4	62.500	158	1	1	34.000
5	31.250	318	1	1	31.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 59 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	90	180.00	4.16 %
2	4.00	74	296.00	6.85 %
3	8.00	66	528.00	12.21 %
4	16.00	62	992.00	22.94 %
5	32.00	60	1920.00	44.41 %
			3916.00	90.58 %

Experiment step size: 1.43 %

Test 59 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	14	3	3	40.333
2	250.000	32	4	4	49.250
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 60 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	91	182.00	4.21 %
2	4.00	75	300.00	6.94 %
3	8.00	67	536.00	12.40 %
4	16.00	63	1008.00	23.32 %
5	32.00	61	1952.00	45.15 %
			3978.00	92.01 %

Experiment step size: 1.43 %

Test 60 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	10	5	5	369.200
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	140	10	10	4.300
5	31.250	311	4	5	8.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 61 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	92	184.00	4.26 %
2	4.00	76	304.00	7.03 %
3	8.00	68	544.00	12.58 %
4	16.00	64	1024.00	23.69 %
5	32.00	62	1984.00	45.89 %
			4040.00	93.45 %

Experiment step size: 1.43 %

Test 61 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	0	4	16	1812.750
2	250.000	36	2	2	69.500
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 62 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	93	186.00	4.30 %
2	4.00	77	308.00	7.12 %
3	8.00	69	552.00	12.77 %
4	16.00	65	1040.00	24.06 %
5	32.00	63	2016.00	46.63 %
			4102.00	94.88 %

Experiment step size: 1.43 %

Test 62 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	3	9	8	176.222
2	250.000	10	15	15	56.000
3	125.000	56	12	12	1.750
4	62.500	158	1	1	12.000
5	31.250	318	1	1	29.000

HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.48 %
2	4.00	16	64.00	1.48 %
3	8.00	8	64.00	1.48 %
4	16.00	4	64.00	1.48 %
5	32.00	2	64.00	1.48 %
			320.00	7.40 %

Experiment step size: 1.43 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Last test with no missed/skipped deadlines:

==========

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 55 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	86	172.00	3.98 %
2	4.00	70	280.00	6.48 %
3	8.00	62	496.00	11.47 %
4	16.00	58	928.00	21.46 %
5	32.00	56	1792.00	41.45 %
			3668.00	84.84 %

Experiment step size: 1.43 %

Test 55 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

==========

Test when deadlines first missed/skipped:

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	33	66.00	1.53 %
2	4.00	17	68.00	1.57 %
3	8.00	9	72.00	1.67 %
4	16.00	5	80.00	1.85 %
5	32.00	3	96.00	2.22 %
			382.00	8.84 %

Experiment step size: 1.43 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	13.000

Final test performed:

==========

Experiment: EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4323.39

Test 62 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	93	186.00	4.30 %
2	4.00	77	308.00	7.12 %
3	8.00	69	552.00	12.77 %
4	16.00	65	1040.00	24.06 %
5	32.00	63	2016.00	46.63 %
			4102.00	94.88 %

Experiment step size: 1.43 %

Test 62 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	3	9	8	176.222
2	250.000	10	15	15	56.000
3	125.000	56	12	12	1.750
4	62.500	158	1	1	12.000
5	31.250	318	1	1	29.000

==========

Benchmark : Hartstone Benchmark, version 1.0
Compiler : Verdix 6.0 -> Sun SPARC
Target : Sun SPARC Station 1+ (25 MHz) - multiuser mode

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

Test 55 of Experiment 3

Raw (non-tasking) benchmark speed in KWIPS: 4323.39

Full task set:

Total Tasks	Deadlines Per Second	Task Set Utilization	Total KWIPS
5	62.00	84.84 %	3668.00

Highest-frequency task:

Period (msec)	Deadlines Per Second	Task Utilization	Task KWIPS
31.250	32.00	41.45 %	1792.00

Experiment step size: 1.43 %

END OF HARTSTONE BENCHMARK SUMMARY RESULTS

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
			320.00	7.15 %

Experiment step size: 1.43 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 2 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
			384.00	8.58 %

Experiment step size: 1.43 %

Test 2 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 3 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
			448.00	10.01 %

Experiment step size: 1.43 %

Test 3 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 4 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
			512.00	11.44 %

Experiment step size: 1.43 %

Test 4 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	2.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 5 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
			576.00	12.87 %

Experiment step size: 1.43 %

Test 5 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 6 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
			640.00	14.30 %

Experiment step size: 1.43 %

Test 6 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 7 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
			704.00	15.73 %

Experiment step size: 1.43 %

Test 7 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 8 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
			768.00	17.16 %

Experiment step size: 1.43 %

Test 8 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 9 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
			832.00	18.59 %

Experiment step size: 1.43 %

Test 9 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 10 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
			896.00	20.02 %

Experiment step size: 1.43 %

Test 10 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 11 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
			960.00	21.45 %

Experiment step size: 1.43 %

Test 11 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 12 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
			1024.00	22.88 %

Experiment step size: 1.43 %

Test 12 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 13 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
			1088.00	24.31 %

Experiment step size: 1.43 %

Test 13 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 14 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
			1152.00	25.74 %

Experiment step size: 1.43 %

Test 14 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 15 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
			1216.00	27.17 %

Experiment step size: 1.43 %

Test 15 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	10.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 16 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
			1280.00	28.60 %

Experiment step size: 1.43 %

Test 16 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 17 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
			1344.00	30.02 %

Experiment step size: 1.43 %

Test 17 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	4.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 18 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
			1408.00	31.45 %

Experiment step size: 1.43 %

Test 18 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 19 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
			1472.00	32.88 %

Experiment step size: 1.43 %

Test 19 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	7.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 20 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
			1536.00	34.31 %

Experiment step size: 1.43 %

Test 20 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 21 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
			1600.00	35.74 %

Experiment step size: 1.43 %

Test 21 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	18.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 22 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
			1664.00	37.17 %

Experiment step size: 1.43 %

Test 22 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 23 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
			1728.00	38.60 %

Experiment step size: 1.43 %

Test 23 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 24 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
			1792.00	40.03 %

Experiment step size: 1.43 %

Test 24 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000
28	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 25 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
			1856.00	41.46 %

Experiment step size: 1.43 %

Test 25 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000
28	125.000	80	0	0	0.000
29	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 26 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
			1920.00	42.89 %

Experiment step size: 1.43 %

Test 26 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000
28	125.000	80	0	0	0.000
29	125.000	80	0	0	0.000
30	125.000	80	0	0	0.000

==========

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 27 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
			1984.00	44.32 %

Experiment step size: 1.43 %

Test 27 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000
28	125.000	80	0	0	0.000
29	125.000	80	0	0	0.000
30	125.000	80	0	0	0.000
31	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 28 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
			2048.00	45.75 %

Experiment step size: 1.43 %

Test 28 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	78	1	1	22.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	16.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	78	1	1	25.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	78	1	1	27.000
16	125.000	80	0	0	0.000
17	125.000	78	1	1	30.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	78	1	1	11.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000
28	125.000	80	0	0	0.000
29	125.000	80	0	0	0.000
30	125.000	80	0	0	0.000
31	125.000	80	0	0	0.000
32	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 29 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
33	8.00	8	64.00	1.43 %
			2112.00	47.18 %

Experiment step size: 1.43 %

Test 29 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000
28	125.000	80	0	0	0.000
29	125.000	80	0	0	0.000
30	125.000	80	0	0	0.000
31	125.000	80	0	0	0.000
32	125.000	80	0	0	0.000
33	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 30 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
33	8.00	8	64.00	1.43 %
34	8.00	8	64.00	1.43 %
			2176.00	48.61 %

Experiment step size: 1.43 %

Test 30 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	5.000
6	125.000	78	1	1	11.000
7	125.000	78	1	1	96.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	78	1	1	30.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	78	1	1	27.000
20	125.000	78	1	1	15.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	78	1	1	24.000
26	125.000	78	1	1	18.000
27	125.000	78	1	1	21.000
28	125.000	80	0	0	0.000
29	125.000	80	0	0	0.000
30	125.000	80	0	0	0.000
31	125.000	80	0	0	0.000
32	125.000	80	0	0	0.000
33	125.000	80	0	0	0.000
34	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 31 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
33	8.00	8	64.00	1.43 %
34	8.00	8	64.00	1.43 %
35	8.00	8	64.00	1.43 %
			2240.00	50.04 %

Experiment step size: 1.43 %

Test 31 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000
28	125.000	80	0	0	0.000
29	125.000	80	0	0	0.000
30	125.000	80	0	0	0.000
31	125.000	80	0	0	0.000
32	125.000	80	0	0	0.000
33	125.000	80	0	0	0.000
34	125.000	80	0	0	0.000
35	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 32 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
33	8.00	8	64.00	1.43 %
34	8.00	8	64.00	1.43 %
35	8.00	8	64.00	1.43 %
36	8.00	8	64.00	1.43 %
			2304.00	51.47 %

Experiment step size: 1.43 %

Test 32 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	15.000
6	125.000	78	1	1	33.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	78	1	1	120.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	78	1	1	35.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	78	1	1	23.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	78	1	1	39.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	78	1	1	29.000
28	125.000	78	1	1	20.000
29	125.000	80	0	0	0.000
30	125.000	80	0	0	0.000
31	125.000	80	0	0	0.000
32	125.000	78	1	1	26.000
33	125.000	80	0	0	0.000
34	125.000	78	1	1	2.000
35	125.000	80	0	0	0.000
36	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 33 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
33	8.00	8	64.00	1.43 %
34	8.00	8	64.00	1.43 %
35	8.00	8	64.00	1.43 %
36	8.00	8	64.00	1.43 %
37	8.00	8	64.00	1.43 %
			2368.00	52.90 %

Experiment step size: 1.43 %

Test 33 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000
28	125.000	80	0	0	0.000
29	125.000	80	0	0	0.000
30	125.000	80	0	0	0.000
31	125.000	80	0	0	0.000
32	125.000	80	0	0	0.000
33	125.000	80	0	0	0.000
34	125.000	80	0	0	0.000
35	125.000	80	0	0	0.000
36	125.000	80	0	0	0.000
37	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 34 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
33	8.00	8	64.00	1.43 %
34	8.00	8	64.00	1.43 %
35	8.00	8	64.00	1.43 %
36	8.00	8	64.00	1.43 %
37	8.00	8	64.00	1.43 %
38	8.00	8	64.00	1.43 %
			2432.00	54.33 %

Experiment step size: 1.43 %

Test 34 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	38	1	1	127.000
3	125.000	78	1	1	1.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	7.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	78	1	1	31.000
9	125.000	78	1	1	8.000
10	125.000	78	1	1	38.000
11	125.000	80	0	0	0.000
12	125.000	78	1	1	21.000
13	125.000	78	1	1	2.000
14	125.000	78	1	1	35.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	78	1	1	46.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000
28	125.000	80	0	0	0.000
29	125.000	80	0	0	0.000
30	125.000	78	1	1	18.000
31	125.000	78	1	1	43.000
32	125.000	78	1	1	27.000
33	125.000	78	1	1	5.000
34	125.000	78	1	1	24.000
35	125.000	80	0	0	0.000
36	125.000	80	0	0	0.000
37	125.000	80	0	0	0.000
38	125.000	80	0	0	0.000

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 35 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
33	8.00	8	64.00	1.43 %
34	8.00	8	64.00	1.43 %
35	8.00	8	64.00	1.43 %
36	8.00	8	64.00	1.43 %
37	8.00	8	64.00	1.43 %
38	8.00	8	64.00	1.43 %
39	8.00	8	64.00	1.43 %
			2496.00	55.76 %

Experiment step size: 1.43 %

C-4

Test 35 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	78	1	1	2.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	78	1	1	1.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	78	1	1	2.000
28	125.000	78	1	1	12.000
29	125.000	80	0	0	0.000
30	125.000	78	1	1	2.000
31	125.000	80	0	0	0.000
32	125.000	80	0	0	0.000
33	125.000	80	0	0	0.000
34	125.000	80	0	0	0.000
35	125.000	80	0	0	0.000
36	125.000	80	0	0	0.000
37	125.000	80	0	0	0.000
38	125.000	80	0	0	0.000
39	125.000	80	0	0	0.000

==========

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 36 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
33	8.00	8	64.00	1.43 %
34	8.00	8	64.00	1.43 %
35	8.00	8	64.00	1.43 %
36	8.00	8	64.00	1.43 %
37	8.00	8	64.00	1.43 %
38	8.00	8	64.00	1.43 %
39	8.00	8	64.00	1.43 %
40	8.00	8	64.00	1.43 %
			2560.00	57.19 %

Experiment step size: 1.43 %

Test 36 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	16	2	2	250.000
2	250.000	32	4	4	1.500
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	11.000
6	125.000	78	1	1	4.000
7	125.000	80	0	0	0.000
8	125.000	78	1	1	31.000
9	125.000	78	1	1	37.000
10	125.000	78	1	1	12.000
11	125.000	80	0	0	0.000
12	125.000	76	2	2	60.500
13	125.000	76	2	2	22.000
14	125.000	76	2	2	115.000
15	125.000	76	2	2	1.000
16	125.000	78	1	1	1.000
17	125.000	78	1	1	1.000
18	125.000	78	1	1	112.000
19	125.000	78	1	1	123.000
20	125.000	78	1	1	1.000
21	125.000	80	0	0	0.000
22	125.000	78	1	1	15.000
23	125.000	78	1	1	111.000
24	125.000	78	1	1	53.000
25	125.000	80	0	0	0.000
26	125.000	76	2	2	1.500
27	125.000	80	0	0	0.000
28	125.000	78	1	1	44.000
29	125.000	78	1	1	56.000
30	125.000	78	1	1	47.000
31	125.000	78	1	1	50.000
32	125.000	80	0	0	0.000
33	125.000	76	2	2	12.500
34	125.000	78	1	1	5.000
35	125.000	78	1	1	1.000
36	125.000	78	1	1	27.000
37	125.000	74	3	3	7.000
38	125.000	78	1	1	21.000
39	125.000	80	0	0	0.000
40	125.000	78	1	1	1.000

HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 1 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
			320.00	7.15 %

Experiment step size: 1.43 %

Test 1 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000

Last test with no missed/skipped deadlines:

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 33 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
33	8.00	8	64.00	1.43 %
34	8.00	8	64.00	1.43 %
35	8.00	8	64.00	1.43 %
36	8.00	8	64.00	1.43 %
37	8.00	8	64.00	1.43 %
			2368.00	52.90 %

Experiment step size: 1.43 %

Test 33 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	320	0	0	0.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000
9	125.000	80	0	0	0.000
10	125.000	80	0	0	0.000
11	125.000	80	0	0	0.000
12	125.000	80	0	0	0.000
13	125.000	80	0	0	0.000
14	125.000	80	0	0	0.000
15	125.000	80	0	0	0.000
16	125.000	80	0	0	0.000
17	125.000	80	0	0	0.000
18	125.000	80	0	0	0.000
19	125.000	80	0	0	0.000
20	125.000	80	0	0	0.000
21	125.000	80	0	0	0.000
22	125.000	80	0	0	0.000
23	125.000	80	0	0	0.000
24	125.000	80	0	0	0.000
25	125.000	80	0	0	0.000
26	125.000	80	0	0	0.000
27	125.000	80	0	0	0.000
28	125.000	80	0	0	0.000
29	125.000	80	0	0	0.000
30	125.000	80	0	0	0.000
31	125.000	80	0	0	0.000
32	125.000	80	0	0	0.000
33	125.000	80	0	0	0.000
34	125.000	80	0	0	0.000
35	125.000	80	0	0	0.000
36	125.000	80	0	0	0.000
37	125.000	80	0	0	0.000

Test when deadlines first missed/skipped:

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 4 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
			512.00	11.44 %

Experiment step size: 1.43 %

Test 4 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	20	0	0	0.000
2	250.000	40	0	0	0.000
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	2.000
6	125.000	80	0	0	0.000
7	125.000	80	0	0	0.000
8	125.000	80	0	0	0.000

Final test performed:

Experiment: EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 4476.28

Test 36 characteristics:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second	Requested Workload Utilization
1	2.00	32	64.00	1.43 %
2	4.00	16	64.00	1.43 %
3	8.00	8	64.00	1.43 %
4	16.00	4	64.00	1.43 %
5	32.00	2	64.00	1.43 %
6	8.00	8	64.00	1.43 %
7	8.00	8	64.00	1.43 %
8	8.00	8	64.00	1.43 %
9	8.00	8	64.00	1.43 %
10	8.00	8	64.00	1.43 %
11	8.00	8	64.00	1.43 %
12	8.00	8	64.00	1.43 %
13	8.00	8	64.00	1.43 %
14	8.00	8	64.00	1.43 %
15	8.00	8	64.00	1.43 %
16	8.00	8	64.00	1.43 %
17	8.00	8	64.00	1.43 %
18	8.00	8	64.00	1.43 %
19	8.00	8	64.00	1.43 %
20	8.00	8	64.00	1.43 %
21	8.00	8	64.00	1.43 %
22	8.00	8	64.00	1.43 %
23	8.00	8	64.00	1.43 %
24	8.00	8	64.00	1.43 %
25	8.00	8	64.00	1.43 %
26	8.00	8	64.00	1.43 %
27	8.00	8	64.00	1.43 %
28	8.00	8	64.00	1.43 %
29	8.00	8	64.00	1.43 %
30	8.00	8	64.00	1.43 %
31	8.00	8	64.00	1.43 %
32	8.00	8	64.00	1.43 %
33	8.00	8	64.00	1.43 %
34	8.00	8	64.00	1.43 %
35	8.00	8	64.00	1.43 %
36	8.00	8	64.00	1.43 %
37	8.00	8	64.00	1.43 %
38	8.00	8	64.00	1.43 %
39	8.00	8	64.00	1.43 %
40	8.00	8	64.00	1.43 %
			2560.00	57.19 %

Experiment step size: 1.43 %

Test 36 results:

Test duration (seconds): 10.0

Task No.	Period in msecs	Met Deadlines	Missed Deadlines	Skipped Deadlines	Average Late (msec)
1	500.000	16	2	2	250.000
2	250.000	32	4	4	1.500
3	125.000	80	0	0	0.000
4	62.500	160	0	0	0.000
5	31.250	318	1	1	11.000
6	125.000	78	1	1	4.000
7	125.000	80	0	0	0.000
8	125.000	78	1	1	31.000
9	125.000	78	1	1	37.000
10	125.000	78	1	1	12.000
11	125.000	80	0	0	0.000
12	125.000	76	2	2	60.500
13	125.000	76	2	2	22.000
14	125.000	76	2	2	115.000
15	125.000	76	2	2	1.000
16	125.000	78	1	1	1.000
17	125.000	78	1	1	1.000
18	125.000	78	1	1	112.000
19	125.000	78	1	1	123.000
20	125.000	78	1	1	1.000
21	125.000	80	0	0	0.000
22	125.000	78	1	1	15.000
23	125.000	78	1	1	111.000
24	125.000	78	1	1	53.000
25	125.000	80	0	0	0.000
26	125.000	76	2	2	1.500
27	125.000	80	0	0	0.000
28	125.000	78	1	1	44.000
29	125.000	78	1	1	56.000
30	125.000	78	1	1	47.000
31	125.000	78	1	1	50.000
32	125.000	80	0	0	0.000
33	125.000	76	2	2	12.500
34	125.000	78	1	1	5.000
35	125.000	78	1	1	1.000
36	125.000	78	1	1	27.000
37	125.000	74	3	3	7.000
38	125.000	78	1	1	21.000
39	125.000	80	0	0	0.000
40	125.000	78	1	1	1.000

```
================================================================================

Benchmark : Hartstone Benchmark, version 1.0
Compiler  : Verdix 6.0 -> Sun SPARC
Target    : Sun SPARC Station 1+ (25 MHz) - multiuser mode

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

   Test 33 of Experiment 4

   Raw (non-tasking) benchmark speed in KWIPS: 4476.28

   Full task set:

      Total       Deadlines        Task Set        Total
      Tasks       Per Second      Utilization      KWIPS
        37          318.00          52.90 %      2368.00

   Highest-frequency task:

      Period      Deadlines          Task          Task
      (msec)      Per Second      Utilization      KWIPS
      31.250        32.00           1.43 %         64.00

   Experiment step size:   1.43 %

================================================================================

                END OF HARTSTONE BENCHMARK SUMMARY RESULTS
```