NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MIGRATION OF A REAL-TIME OPTIMAL-CONTROL ALGORITHM: FROM MATLAB™ TO FIELD PROGRAMMABLE GATE ARRAY (FPGA)

by

Ron L. Moon II

December 2005

Thesis Advisor:	I. Michael Ross
Thesis Co-Advisor:	Herschel H. Loomis

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2005	**3. REPORT TYPE AND DATES COVERED** Astronautical Engineering Master's Thesis	
4. TITLE AND SUBTITLE: Migration of a Real-time Optimal-control Algorithm: from MATLAB[tm] to Field Programmable Gate Array (FPGA)			**5. FUNDING NUMBERS**
6. AUTHOR(S): Ron L. Moon II			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis presents an overarching plan to migrate a time-optimal spacecraft control algorithm from the MATLAB™ development environment into an FPGA-based embedded-platform development board. Research at the Naval Postgraduate School has produced a revolutionary time-optimal spacecraft control algorithm based upon the Legendre Pseudospectral method. Currently, the control algorithm is dependent on the MATLAB™ environment, a fourth generation language (4GL). 4GLs are powerful high-level abstraction and development tools, but are not efficiently instantiated into an embedded system. This study establishes three distinct development phases to migrate the algorithm from 4GL dependency to embedded operation. The first phase removes the algorithm's dependency on the 4GL environment by translating the algorithm into the C programming language. The second development phase compiles and embeds the algorithm into an FPGA-based development board. The final development phase introduces a custom computing machine (CCM) instantiated in an FPGA to reduce the control calculation time, thereby broadening the algorithm's potential application.

14. SUBJECT TERMS Satellite Control, Optimal Control, DIDO, Legendre Pseudospectral Method, FPGA, Custom Computing Machine, Embedded Computing			**15. NUMBER OF PAGES** 113
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

MIGRATION OF A REAL-TIME OPTIMAL-CONTROL ALGORITHM: FROM MATLAB™ TO FIELD PROGRAMMABLE GATE ARRAY (FPGA)

Ron L. Moon, II
Lieutenant Commander, United States Navy
B.S., Vanderbilt University, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ASTRONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2005

Author: Ron L. Moon, II

Approved by: I. Michael Ross
Thesis Advisor

Herschel H. Loomis
Co-Advisor

Anthony J. Healey
Chairman, Department of Mechanical and Astronautical Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis presents an overarching plan to migrate a time-optimal spacecraft control algorithm from the MATLAB™ development environment into an FPGA-based embedded-platform development board. Research at the Naval Postgraduate School has produced a revolutionary time-optimal spacecraft control algorithm based upon the Legendre Pseudospectral method. Currently, the control algorithm is dependent on the MATLAB™ environment, a fourth generation language (4GL). 4GLs are powerful high-level abstraction and development tools, but are not efficiently instantiated into an embedded system. This study establishes three distinct development phases to migrate the algorithm from 4GL dependency to embedded operation. The first phase removes the algorithm's dependency on the 4GL environment by translating the algorithm into the C programming language. The second development phase compiles and embeds the algorithm into an FPGA-based development board. The final development phase introduces a custom computing machine (CCM) instantiated in an FPGA to reduce the control calculation time, thereby broadening the algorithm's potential application.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Quatye, Jessica, and Wyatt, I appreciate the many sacrifices that you have made in support of my service to our country. Few fully understand the sacrifices you have made. I am blessed to have the three of you in my life.

To Dr.'s Mike Ross, Herschel Loomis, Alan Ross, and Walter Murray, thank you for allowing me to meander among giants.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

The purpose of this thesis is to provide an overarching plan to migrate a time-optimal, spacecraft attitude-control algorithm from the MATLAB™ development environment into an embedded development board. The pace of advances within the electrical and electronic industry is rapid. We recognize this fact and understand that in the course of this research, writing, and distribution, factors that influenced critical decisions in formulating the plan's path selection may change. To mitigate this effect, we will identify significant factors influencing the recommended development path. The identification and discussion of key influence factors will allow personnel implementing this plan the opportunity to alter the plan as technological advances make previously unattractive or unachievable development paths obtainable. The primary goal of this document is to develop a clear, concise, and methodical working-level plan, not an esoteric discourse.

At the onset of this research, the assignment was to investigate the plausibility of migrating an existing algorithm from MATLAB™ to an embedded system. After preliminary research, the migration task appeared feasible. The scope of research was then expanded to investigate potential migration paths, identify the most promising path in terms of cost and implementation time, and record key decision factors for these recommendations.

This document is not intended to be a standalone migration plan; it is not all-inclusive. The intention is to use this document as a starting point and serve as an overarching plan to guide the overall migration of the algorithm from desktop PC to embedded-development-system operation. This document will segment the migration process into distinct development efforts. This method of work breakdown facilitates distributing segments of the plan among multiple students or industry partners for implementation. Using the research information provided by this document, tasked individuals or organizations will formulate a more detailed plan for each respective work element.

B. CURRENT ALGORITHM STATE

Several variants of the time-optimal spacecraft attitude-control algorithm exist at the Naval Postgraduate School. The particular variant utilized in this work was originally developed by Andrew Fleming[1] and modified by Pooya Sekhavat[2]. The modifications removed fixed time-step calculations and improved problem scaling, reducing the time required to generate a solution. Henceforth, the variant of the time-optimal spacecraft slew maneuver control algorithm utilized in this document will be referred to as the control algorithm.

The control algorithm is based upon satisfying Pontryagin's Maximum Principle[3]. The algorithm achieves the time-optimal maneuver solution by maintaining a set of Maximum Principle conditions. The Legendre pseudospectral method[4] is employed to derive solutions that meet, and maintain, the Maximum Principle requirements throughout the solution space. The purpose of this thesis is to derive an achievable plan to migrate the algorithm from MATLAB™ into embedded hardware. This thesis does not delve deeply into the algorithm's behavior, unless that behavior significantly influences the migration process.

The control algorithm currently operates within MATLAB™, a proprietary fourth generation language (4GL) developed by The MathWorks, Inc. The control algorithm implementation is comprised of numerous programming script files, M-files, and functions. It is important to note that the implementation does not utilize SIMULINK™. SIMULINK™ is a MathWorks block-library modeling tool that is integrated with MATLAB™. The significance of this statement will be discussed further in the software section of this document. Function calls provide MATLAB™'s interpreter the cueing required to link the script files and create an executable program.

---

[1] Fleming, A. (2004). *Real-Time Optimal Slew Maneuver Design and Control*. Monterey, CA: Naval Postgraduate School.

[2] Sekhavat, P., Fleming A. and Ross, I. M. (2005, July). *Time-Optimal Nonlinear Feedback Control for NPSAT1 Sapcecraft*. Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Monterey, CA..

[3] Kopp, R.E. (1962). George Leitman (Ed.) *"Pontryagin Maximum Principle," in Optimization Techniques*. New York: Academic Press, Inc.

[4] Ross I. M. and Fahroo, F. (2003). *"Legendre Pseudospectral Approximations of Optimal Control Problems," Lecture Notes in Control and Information Sciences* (Vol. 295). New York: Springer-Verlag.

The control algorithm contains three proprietary elements: Sparse Non-linear OPTimizer (SNOPT), TOMLAB Optimization wrappers, and DIDO. SNOPT is a subservient algorithm, developed at Stanford University, which performs large-scale constrained optimization using sequential quadratic programming (SQP) methods[5]. TOMLAB Optimization, a Swedish company, developed a software adapter, wrapper, facilitating the use of SNOPT within MATLAB™. DIDO, not an acronym, is a MATLAB™ package capable of solving dynamic optimization problems[6]. DIDO was conceived and written by professors at the Naval Postgraduate School in Monterey, California. In its current configuration, DIDO is dependent upon the TOMLAB wrapper to access SNOPT in order to solve optimization problems.

The following provides a simplified description of the control algorithm's operation. All user interaction with the control algorithm occurs within MATLAB™'s Development Environment, a Graphical User Interface (GUI). The user specifies the initial and final spacecraft attitude and rotation rate in the main script file using programming constants. Programming constants used in this manner is commonly referred to as "hard wiring" and is useful for developing programs that will eventually receive a range of input values. The main script file includes programming constants that define the spacecraft's physical characteristics, such as moment of inertia and maneuvering capability. The main script file is synonymous with the spacecraft model. Once initial and final states are defined, the user runs the main script file within the MATLAB™ development environment. The main script file calls the DIDO function that, in turn, calls the SNOPT function. SNOPT operates in a recursive manner, executing major and minor iterations. DIDO collects SNOPT's iterative solutions and derives the overall optimal control solution. DIDO passes the control solutions back into the MATLAB™ environment as objects within a predefined programming structure. Each process level described above interacts with the PC's central processing unit (CPU) through the operating system, an important point when attempting to increase the

---

[5] Gill, Philip E., Murray, Walter, and Saunders, Michael A. (2005). *SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization*. Society for Industrial and Applied Mathematics Review (Vol. 47, No.1, pp. 99-131). Philadelphia: SIAM.

[6] Ross, I.M., and Fahroo, F. (2002). *User's Manual for DIDO 2003: A MATLABTM™ Application Package for Dynamic Optimization.* Monterey, CA: Naval Postgraduate School.

performance of the control algorithm. Figure 1 provides a representation of the hierarchical relationship between the spacecraft model, DIDO, SNOPT, and the operating system. The TOMLAB wrapper is viewed as an access portal to SNOPT. The wrapper does not perform any significant calculations.

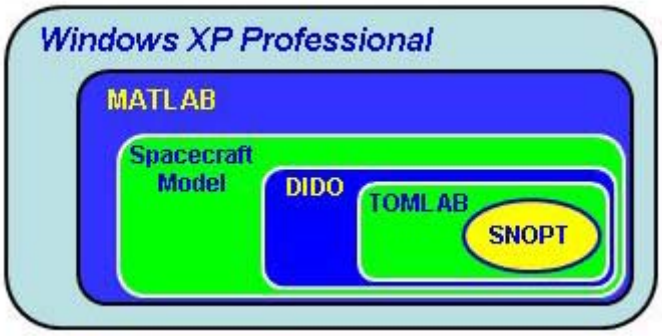

Figure 1. Real-time Optimal Control Algorithm

II. MIGRATION PATH

A. MIGRATION OVERVIEW

The migration overview outlines the control algorithm's transition from a MATLAB™ executable to an embedded application. The transition is performed using incremental development efforts. Dividing the migration plan into smaller development efforts is beneficial for two significant reasons. The work may be distributed among several students or contractors. This work may be performed concurrently, as a development team, or sequentially over time. The second benefit of dividing the development effort is that the algorithm can be tested, and validated, within each work element. If the migration effort were accomplished in a single effort, locating error sources would potentially require investigating the entire migration process. Conversely, if the migration process is broken into smaller work elements, developers can scrutinize the changes made within the bounds of the work element, assuming testing and validation was performed within each work element.

1. Phase 1: MATLAB™ Extraction

a. Why Extract?

Deploying a satellite attitude-control algorithm operating within MATLAB™ is not practical. Dependency on another application adds a layer of unnecessary hardware resources on the satellite. The unnecessary hardware resources burden the satellite with unnecessary mass, volume, and power requirements. Additional hardware resources increase the cost of the launch system and satellite. Furthermore, MATLAB™ script programs are interpreted during execution not compiled. Interpreted programs tend to run slower than their compiled counterparts, due to the run-time interpretation. Therefore, MATLAB™ programs are slower than their compiled counterparts.

b. MATLAB™ Extraction

The first development effort extracts the control algorithm from MATLAB™. The goal of control algorithm extraction is to operate as a stand-alone executable program within Microsoft® Windows XP. The MATLAB™ Compiler will be used to translate the control algorithm modules into the C programming language. A programming-development environment will be used to compile and link the control-algorithm modules and math libraries into a Microsoft® Windows XP executable program. The extraction removes the control algorithm's dependency on the MATLAB™ development environment and frees the algorithm from the associated resource overhead.

Modular testing will verify control signal generation and measure solution generation time. The MATLAB™-generated control solutions are used as the baseline throughout the migration process. The MATLAB™ based control algorithm has been verified. The developer will validate the stand-alone program by comparing its results with the MATLAB™ solution. Discrepancies will be investigated and corrected. The stand-alone program is expected to generate control solutions faster due to shedding MATLAB™'s resource overhead. However, there is possibility that the stand-alone control algorithm will perform slower. MATLAB™ contains an accelerator, JIT. The accelerator's enhancements may not translate or reside within the extraction libraries. The developer will execute a series of spacecraft maneuvers using both the MATLAB™ and stand-alone algorithm variants. The solution generation times will be recorded, compared, and analyzed during each phase of development.

The operator will interact with the stand-alone control algorithm using a disk operating system (DOS) window. Within the DOS window, the user will be prompted to enter the initial and final satellite orientation and rotation rates. Once entered, the executable control algorithm calculates the time-optimal control signals and writes the results to an ASCII file. The algorithm will also display the time required to generate the control solution in the DOS window. Figure 2 summarizes the algorithm's structure at the completion of phase one.

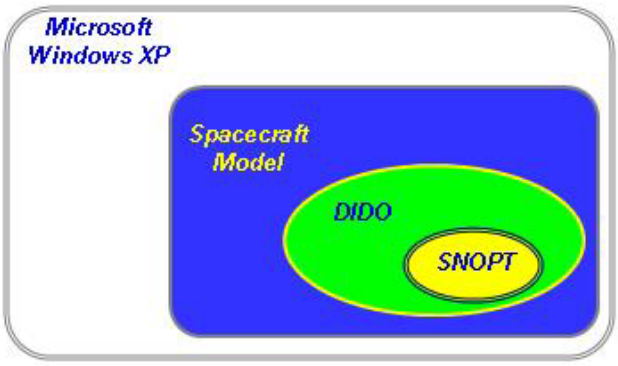

Figure 2. Phase 1 Algorithm Development

While developing an intricate graphical user interface (GUI) is tempting, it is not required. A GUI will not be required during embedded operations and would only serve to increase the hardware resources required to host the control algorithm. While a text based GUI is not impressive, the text-based interface within a DOS window is sufficient for testing and validating the extracted algorithm. It would be wise to apply personnel efforts and resources in other areas, areas that would provide a more direct benefit.

2. Phase 2: Embedded Platform Development

Phase two migrates the algorithm onto an embedded platform-development board, not directly to flight hardware. The control algorithm requires validation and verification in a stand-alone hardware configuration prior to expending resources for the transition to flight hardware. Anticipated challenges during this portion of the migration are similar, if not identical, to the challenges to be encountered during the transitioning to flight hardware. Resolving migration problems on an embedded development board that interfaces directly with a personal computer is more expedient and less expensive when compared to performing the same migration and troubleshooting on flight hardware.

a. Phase One Dependency

Phase two development utilizes source-code translated in phase one. Traditionally, source-code must be compiled for a specific target operating system and

processor. With some of the new programming languages, such as Java, this is not always true. However, this project is using programming languages and tools that compile to specific operating system and processor. In phase one, the source-code is compiled to operate on the host computer running Microsoft® Windows XP and an x86 CPU. In phase two, the source-code is compiled for operation on an embedded platform development board running a real-time operating system and a microcontroller.

b. Phase Two Hardware

Embedded platform development boards are small computer boards containing many of the following components: clock, processor, memory, input-output (I/O) ports, and field programmable gates array (FPGA). Embedded platform development boards are available through several manufactures. Figure 3 provides an example of an embedded platform development board, the Virtex-4 ML403 Embedded Platform. The ML403 is a very capable development board offered by Xilinx, Inc., a company headquartered in San Jose, California.

Figure 3. Virtex-4 ML403 Embedded Platform Development Board[7]

[7] Courtesy of Xilinx, Inc.

The phase two development board will operate as a microcomputer, hosting a Real-Time Operating System (RTOS). The control algorithm source-code developed in phase one will be compiled to operate with the development board's RTOS and processor configuration. The user will interact with the development board using a desktop PC. The PC communicates with the development board via an I/O cable, preferably a USB interface. The user will enter initial and final satellite state information. Once this information is forwarded to the development board, the control algorithm will generate the control signals and store the information in a predefined memory location.

The embedded development board will generate a control solution for the commanded maneuver and store the commands in the board's memory. The computed control commands will be extracted from the board using the I/O interface. The control commands will be recorded and compared with the solutions provided by the MATLAB™ algorithm. The time required for the development board to perform the calculations will be recorded and compared to the other variants. The control signal generation time will be used to determine potential aerospace applications.

3. Phase 3: Hardware Acceleration

After the control algorithm is successfully hosted within the development board, methods to reduce the computation time will be implemented in hardware. The control algorithm performs repetitive, large-vector inner-product calculations during each control-signal generation. Within a PC, these calculations are performed using the CPU's floating-point unit (FPU). These calculations can be performed faster using custom processing logic. Custom computing logic is commonly referred to as a custom computing machine (CCM). When used in conjunction with a generic processor, the CCM is also known as an auxiliary processing unit (APU).

Cost-effective implementation of a CCM requires an FPGA and algorithm-source-code modification. The CCM performs large-vector inner-product calculations rapidly. The design is instantiated within an FPGA residing on the development board. A data transfer bus connects the CPU and CCM. The algorithm's source-code requires

modification to utilize the CCM, vice an FPU, each time an inner-product calculation is required. The double-precision floating-point inner-product result is returned to the CPU via a data return bus. Figure 4 provides a block diagram of the phase three system.

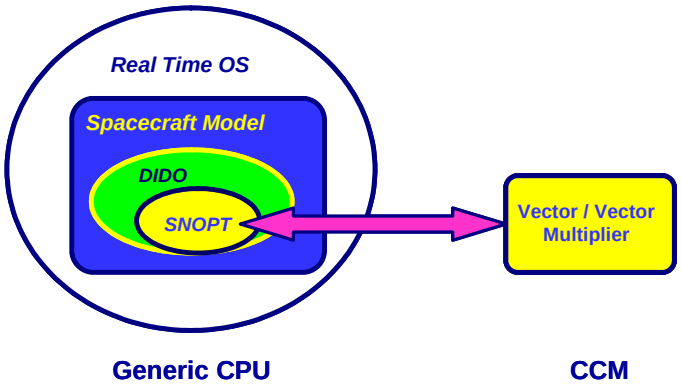

Figure 4. Phase 3 Block Diagram

Phase three operation is identical to phase two operations. The hardware acceleration does not affect the manner in which the operator interacts with the embedded development board. The embedded development board will generate a control solution for the commanded maneuver and store the commands in the board's memory. The computed control commands will be extracted from the board using the I/O cable. The control commands will be recorded and compared with the solutions provided by the MATLAB™ algorithm. The time required for the development board to perform the calculations will be recorded and compared to the other variants. Successful hardware acceleration will be apparent by experiencing a reduction in required calculation time. The control signal generation time will be used to determine potential aerospace applications.

III. PHASE ONE – EXTRACTION

A. EXTRACTION OPTIONS

MATLAB™ Release 13 (version 6.5) and higher are fourth generation languages (4GLs). The popularity of 4GLs is based on the ability to provide the user with high-level abstraction capabilities. MATLAB™ provides a very powerful abstraction capability; however, this abstraction capability is not provided without penalty. Programs written for the MATLAB™ development environment are dependent upon MATLAB™'s interpreter to execute. Because of this dependency, MATLAB™ programs are not directly executable within the PC's operating system. There are three primary options for severing the control algorithm's dependency on MATLAB™ and converting it into a stand-alone program: migrate to SIMULINK, rewrite, or translate.

1. Convert to SIMULINK

Conversion of the MATLAB™ script code into a SIMULINK™ model is one potential conversion path. MATLAB™ and SIMULINK™ are highly integrated software products offered by MathWorks. SIMULINK™ is a model and simulation software package. MathWorks offers a SIMULINK™ companion module, Real-Time Workshop, which translates a model into stand-alone C code. Furthermore, MathWorks sells additional SIMULINK™ companion products that provide rapid migration paths from modeling to select hardware devices.

While MATLAB™ and SIMULINK™ are closely integrated, migration of the control algorithm into SIMULINK™ may introduce two insurmountable problems. As discussed in section one, the control algorithm is dependent upon a module called SNOPT. This module is accessed using a third-party software wrapper. The wrapper was written to support MATLAB™, not SIMULINK™. If SNOPT does not operate properly after the control algorithm is migrated into SIMULINK™, it will be difficult to identify whether a problem resides in the wrapper or within SNOPT. Furthermore, the wrapper is proprietary. Isolating and correcting a problem will require negotiating a business agreement with TOMLAB. Secondly, at this time, several SIMULINK™

library blocks will not compile into ASNI C code. The control algorithm is constructed using a layering of MATLAB™ functions. While the SIMULINK™ library contains a user-defined MATLAB™ function block, the function block (fcn) is one of several library blocks that do not compile to production code[8]. The user can create custom SIMULINK™ blocks by writing S-Function[9] or Embedded MATLAB™ functions[10]. An S-Function is script code that defines the behavior of the SIMULINK™ block, and can be written in C, C++, Ada, or FORTRAN programming languages. It is conceivable that the control algorithm could be disassembled into basic function blocks, converted into S-Functions or Embedded MATLAB™ Functions, reassembled, debugged, and verified in SIMULINK™. After studying the control algorithm source-code, the time and effort required to perform this task is forecasted to rival the rewrite migration effort.

2. Rewrite

Rewriting the entire algorithm in a computer language that compiles to the desired embedded hardware is another migration option. A validated and verified control algorithm exists within MATLAB™. Figure 6 displays the modular structure of the MATLAB™ -hosted control algorithm. The existing structure parallels the structure developers would use to implement the control algorithm in a programming language. This existence and similarity of the MATLAB™ variant provides a useful tool to verify proper operation of each programmed function. Results from the corresponding MATLAB™ function can be used to verify the results provided by the written code.

Re-writing the control algorithm in a programming language requires a significant investment of time by proficient programmers. The programmers require training in advanced mathematical concepts, such as Pseudospectral methods[11], central to the

[8] A complete list of SIMULINK™ blocks suitable for production code generation can be retrieved from <www.mathworks.com/access/helpdesk/help/toolbox/rtw/ug/bqecl8b.html> or by typing "showblockdatatypetable" at the command line in MATLAB™ .

[9] The MathWorks. SIMULINK™ Product Page. Retrieved 10 Nov. 2005, from <www.mathworks.com/access/helpdesk/help/toolbox/simulink/sfg/f6-151.html>

[10] The MathWorks. MATLAB™ Product Page. Retrieved 10 Nov. 2005, from <www.mathworks.com/access/helpdesk/help/toolbox/simulink/slref/embeddedMATLABTMfunction.html>

[11] Ross I. M. and Fahroo, F. (2003). *"Legendre Pseudospectral Approximations of Optimal Control Problems."* Lecture Notes in Control and Information Sciences (Vol. 295). New York: Springer-Verlag.

control algorithm. The time and training requirement is the result of using MATLAB™, a 4GL, to create and validate the control algorithm. The 4GL provides a very high level of abstraction through complex, pre-packaged function calls. MATLAB™ functions are proprietary and native to its development environment. Equivalent functionality must be implemented in the selected programming language in order to extract the control algorithm from MATLAB™. In their current form, the MATLAB™ functions are not directly accessible by a stand-alone application running within a PC operating system. Secondly, if the functions are called from the operating system using a library scheme, propriety issues must be resolved with MathWorks prior to system deployment.

3. Translate and Compile

MATLAB™ contains a compiler capable of converting M-files into four different products: stand-alone applications, C or C++ shared libraries, Excel add-ins, or Component Object Models (COM). Throughout this thesis, a capital "C" will differentiate between a programming compiler and MATLAB™'s Compiler module. The control algorithm is comprised of a main M-file that calls a series of M-file functions. The goal of phase one is to develop a stand-alone control algorithm executable program. The Compiler option provides two paths in which to convert the control algorithm into a stand-alone application: translate and compile directly to a stand-alone program or C/C++ source-code modules.

4. Path Selection Influence: Embedded Programming Language

While there are several embedded programming languages, C has become the dominant language in embedded programming[12]. Several programming languages support embedded programming: assembly, Pascal, FORTRAN, C++, Ada, and Java. C is a dominant embedded programming language because it allows low-level control and provides high-level abstraction. These traits are often mutually exclusive, or significantly out of balance, in the other programming languages. Because of C's balance and

[12] Barr, Michael. (1999). *Programming Embedded Systems* (p. 9). Sebastopol, CA: O'Reilly & Associates, Inc.

flexibility, there is a plethora of C development and compiler tools. C is one of the programming languages that can be used within MATLAB™ and SIMULINK™ using wrappers.

C is the recommended programming language based on three factors. First, C is the dominant embedded programming language and enjoys widespread industry support. Most embedded hardware manufactures provide and support C compilers for their products. Secondly, The MathWorks offers a C compiler module that integrates with MATLAB™. The compiler translates M-file code into C or C++ object code. Lastly, Stanford University has provided the Naval Postgraduate School with a copy of SNOPT written in C code. This provision significantly reduces the development effort required to migrate the control algorithm should the C programming language be utilized.

5. Path Selection

The MATLAB™ Compiler option is recommended because it provides the best opportunity to achieve near-term migration advances. The SIMULINK migration path was not selected due to the limited library model set and potential proprietary delays. The risk of encountering time and cost delays during the limited time of this research versus the anticipated results made the SIMULINK option less attractive. The rewrite option was not pursued due to the time and programming proficiency required. The time required to pursue the rewrite option was beyond the limits of this research. Translating the control algorithm's M-files using MATLAB™'s established Compiler provided the most promising migration path. The MathWorks' Compiler literature indicated that the control algorithm's functions could be translated and compiled directly into a stand-alone application or the modules translated into C/C++ code.

B. MATLAB™ COMPILER

1. Single-step or Modules?

The MATLAB™ Compiler option offers two methods to translate and compile the control algorithm into a stand-alone application. The first method compiles the entire

algorithm into an application in a single step. This approach accomplishes the goal of phase one; however, the single-step compile approach is not the best method to support the following project phases. The single-step compile method compiles an application that operates only on the specified target operating system and CPU. The application cannot continue the migration process in phases two and three. Development effort must backtrack and repeat the translation process from within the MATLAB™ environment in order to proceed into phases two and three. The algorithm would be translated into C code, which, in turn, would then be linked and compiled for the target hardware platform.

The second compiler option translates the MATLAB™ control algorithm modules, not entire application, into the C programming language. The C modules are linked using a commercial programming application. Once linked, the code is compiled into executable code for a host platform. Potential host platforms range from common mainframe and desktop computers to embedded systems. Because C is a prevalent programming language, most manufacturers provide C compilers for their hardware.

Translating and compiling modules provide three benefits: reduced redundant effort, migration flexibility, and verification. The single-step compiled control algorithm cannot migrate into phases two and three. The single-step compiled control algorithm is fixed to the target operating system and processor for which it was compiled. Translators are available in industry. However, translators are often proprietary and are written to support a transition from a single application to a particular hardware platform, i.e. MATLAB™ to C. At the time of this report, a translator did not exist for the control algorithm. Therefore, prior to proceeding into phase two, the MATLAB™ control algorithm modules would require translation. A programming-development environment would be used to link and compile the translated code into an executable program capable of being hosted on a development board. Translating and compiling the entire control algorithm in one step requires redundant work.

Translating the individual control algorithm modules into C code provides migration flexibility. The control algorithm modules are translated into C using the MATLAB™ Compiler. A commercial programming application is used to link and compile the C modules. The liked code may then be compiled into an executable

program for a variety of host systems. In the translated state, the control algorithm may be compiled and hosted on a desktop PC, to support phase one, or on an FPGA-based embedded-platform development board, to support phase two and three. Borland C++ Builder, GCC, LCC-WIN32, Microsoft® Visual Studio are a few examples of common programming environments that accept C code. These programming packages provide their own integrated development environment, which compiles and links object code into executable programs for a variety of operating system and processor combinations. Additionally, some programming environments, like GCC, allow the user to add hardware manufacturer compilers. This addition allows the user the ability to cross-compile executable programs for the embedded hardware.

Translating the individual control algorithm modules provides a verification and comparison tool for phases two and three. The linking and compiling individual modules maintains a common control algorithm structure through each phase of development. If the phase one algorithm is translated and compiled directly into a stand-alone application, algorithm structure similarity cannot be guaranteed. The Compiler's manipulation of the control algorithm is unknown. Therefore, execution time comparisons between the development phases may not be directly comparable, nor future performance enhancements predictable. Additionally, if the generated control signals are in error, it will be difficult to determine the source of the error. A common development approach between phases one and two increases familiarity, maintains a common algorithm structure, and assists measuring algorithm performance in each development phase.

Modular translation provides a means to introduce algorithm improvements in a disciplined and verifiable manner. Modularity provides a natural means to make improvements. As improvements to the algorithm are introduced, the affected module can be updated, tested, and verified prior to integration into the algorithm. This modular approach provides discreet boundaries within which the changes have been made. If the algorithm fails verification after the introduction of a changed module, the boundary of change is known. This knowledge localizes the error source to one or more modules, vice the entire algorithm.

2. Compiling Modules

The control algorithm's modules are translated into C code using the MATLAB™ Compiler. The Compiler is an add-on companion to MATLAB™ and must be purchased separately from MathWorks. The control algorithm consists of a conglomeration M-files linked by function calls. The control algorithm structure is displayed in Figure 5. Note that the ".m" file extensions are omitted for brevity. Additionally, MATLAB™ function calls are not displayed due to their number and relationship complexity.

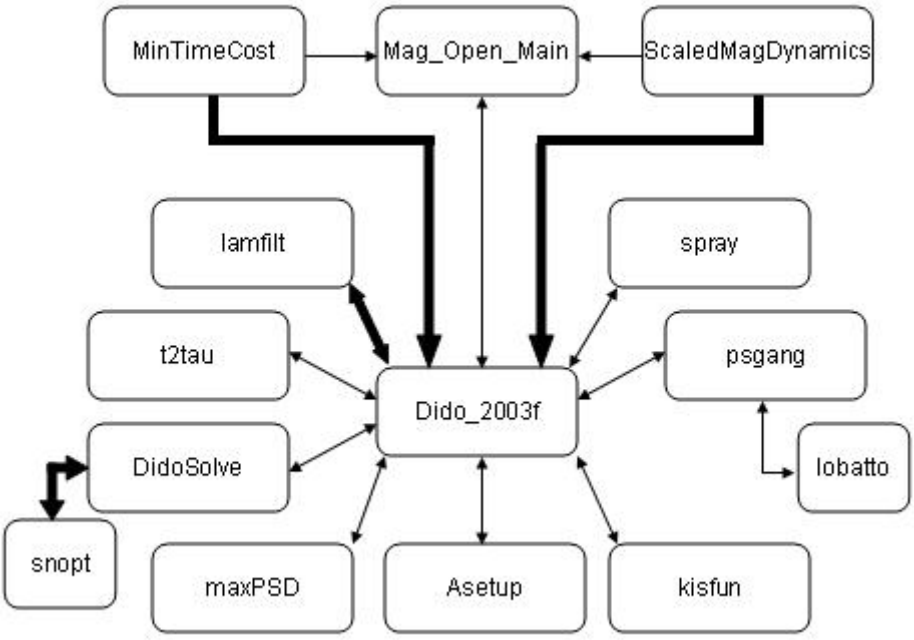

Figure 5. Control Algorithm Structure

The algorithm execution begins by opening and executing Mag_Open_Main from within the MATLAB™ IDE. Mag_Open_Main first collects information from MinTimeCost and ScaledMagDynamics and then calls Dido_2003f. Single-headed arrows represent information flow in one direction; dual-headed arrows represent information flow in both directs. Once called, Dido_2003f controls the algorithm process and now directly pulls information from MinTimeCost and ScaledMagDynamics. Dido_2003f calls the surrounding functions in a clockwise manner, beginning with spray and followed by psgang. Psgang calls its own function, lobatto, which is transparent to Dido_2003f. The

clockwise progression continues until DidoSolve. DidoSolve is the parent calling function to SNOPT, the Stanford University code. DidoSolve calls SNOPT repeatedly. The bolded arrows annotate multiple execution cycles. The function call execution continues the clockwise progression ending on lamfilt. Lamfilt is another function that is called on several occasions, but at different times earlier in the clockwise cycle. Two-dimensional representation of the algorithm execution makes it difficult to reproduce the calling sequence exactly, without generating further confusion.

MATLAB™ versions 7.0 and 6.5.1 utilize two different Compilers; each produces significantly different results. The companion compiler to MATLAB™ 7.0.1.24704 service pack 1 is Compiler version 4.1.1. The companion compiler to MATLAB™ 6.5.1.199709 service pack 1 is Compiler version 3.0.1. Phase one's migration effort translates the control algorithm's M-files into C source-code. Compiler 4, and later, does not generate C code for the entire M-function[13]. Beginning with Compiler version 4, the MATLAB™ Compiler generates wrappers, interface code, which allow the compiled M-files to be executed within the MATLAB™ Runtime Component (MCR). MCR is a set of proprietary MathWorks stand-alone runtime libraries. The MCR libraries are not C/C++ libraries and are not suitable for embedded deployment[14]. Compiler versions prior to Compiler version 4 translate the M-file into C code[15], minus library function calls. The desired migration path requires complete C code. The research and work presented in this document utilized Compiler version 3.0.1.

Each module in Figure 5, except SNOPT, has been translated into C code. The Compiler translates functions only. Therefore, the Mag_Open_Main module was converted into a function prior to translation; see Appendix A. The stand-alone algorithm will require the creation of a C main file to accept the user commands and initiate the stand-alone algorithm by calling the Mag_Open_Main function. Note: the Mag_Open_Main file executes the open-loop control algorithm. The closed-loop control algorithm is the more useful version. Therefore, when phase-one algorithm migration begins, the developers should use the Mag_Closed_Main file.

[13] The MathWorks. MATLAB™ Compiler Release Notes Page. Retrieved 15 Nov. 2005, from < http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/rn/compiler4_rn_fcs3.html>

[14] The MathWorks. MATLAB™ Compiler Release Notes Page. Retrieved 15 Nov. 2005, from < http://www.mathworks.com/support/solutions/data/1-H3RQL.html?solution=1-H3RQL>

The Compiler translation is performed by executing the **mcc** command from within the MATLAB™ development environment. The Compiler's output may be changed using option switches. Table 1 provides the Compiler command and options used to translate the control algorithm modules into C code.

Command:	mcc –t –A debugline:on –L c –d C:\filelocation\t2tau.m translated_t2tau
Options	**Meaning**
mcc	Calls the compiler
–t	Directs the compiler to translate the code to the target language specified
	If omitted, a C or C++ wrapper file is generated
–A debugline:on	Supports run-time error messages reporting source file name & line number
–L	The character following the switch specifies target language
c	Specifies C as the target language translation
	cpp is used, vice c, when C++ translation is desired
–d	All files are placed in the directory following the switch
C:\filelocation\t2tau.m	Identifies the file path and file name
translated_t2tau	User provided name for the translated files

Table 1. MATLAB™ Compiler Command and Options

The switch options identified in table one are a subset of the available Compiler options. An exhaustive listing can be reviewed by entering "help mcc" on the interpreter command line in the MATLAB™ development environment if the Compiler is installed. The Compiler options are also accessible through The MathWorks website[15].

The Compiler produces two or more files for each translated M-file. The translated files are placed in the same location as the source file, specified in the command line. The Compiler generates a translated *.c file and one or more *.h header files. The wild card"*" represents the compiled source file name. The header files are invaluable when compiling and linking the individual modules within a programming environment. This can be a tedious process without header files. A single translated file may generate more than one header file due to MATLAB™ function calls, not displayed

[15]The MathWorks. MATLAB™ Compiler Online Guide. Retrieved 15 Nov. 2005, from <http//: www.mathworks.com/access/helpdesk_r13/help/toolbox/compiler/compiler.html >

in Figure 5. The original, modified, and translated control algorithm files are maintained in CD-ROM media format and referenced as Appendix A. The files are available to persons involved with the migration project.

While translators are a powerful tool, translators can accept relatively simple source-code and produce nearly unreadable translated code. MATLAB™'s translator is a prime example. Appendix D displays the lobatto.m source file and Appendix E provides the translated C code. The files are provided as appendices due to their length. The Compiler's translator generates very lengthy and confusing variable names. Debugging translated code will be a challenge. One approach to simplifying the translated code is variable renaming. Most programming-development environments contain powerful editors. Most editors contain a search and replace function. Confusing or lengthy variable names can be changed using the editors search and replace function. If the programming editor does not have the search and replace function, the source-code can be copied and pasted into a modern word processor, such as Word Perfect. The word processor's search and replace function can be used to rename the variables. Once complete, the code can be copied and pasted back into the original code file and saved. Microsoft® Word is not a good editing environment for programming. Word has a tendency of adding hidden characters and formatting which cause untraceable compiler errors, even if the "save as type" is Rich Text Format or Plain Text. Some text editors, such as Microsoft®'s TextPad, are not powerful enough to handle large text files. TextPad is a low cost but powerful editor that can be easily configured to incorporate many different compilers. It can be downloaded from the internet and evaluated, free. A copy is included in Appendix A, along with the configuration instruction. Lastly, it is recommended that the programmer does not select search and replace all; rather, the programmer should step through and review each replacement prior to accepting the change. This prudent method will prevent the inadvertent partial renaming of long strings that contain the name being replaced.

The following are two practical comments concerning the Compiler options. The file name, and possibly location, will change for each module. Secondly, do not add an "o" or "c" extension to the translated filename. The Compiler automatically adds the extension. For example, do not use "translated_t2tau.c" as the target filename. The

Compiler will translate the source M-file into C code and name the file "translated_t2tau.c.c". The double ".c" extension may cause problems when using a programming-development environment later in the migration process. The Lcc programming-development environment had difficulty properly recognizing files containing double extensions.

Phase one migration work was terminated at his point in order to explore phase two and three migration options. Discussion concerning further work required to complete phase one is located in Section VI.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. PHASE TWO – HARDWARE MIGRATION

A. PHASE TWO OBJECTIVES

Phase two is comprised of three major objectives: embedding, verifying, and analyzing the control algorithm.

1. Embedding the Algorithm

The algorithm will be migrated into an embedded development board. Direct migration to flight hardware requires a program sponsor to provide flight hardware. Due to monetary limitations, sponsors cannot afford the cost of providing flight hardware every research and development program. Furthermore, sponsors are hesitant to assume the risk of integrating an immature algorithm into their project. The term "mature" algorithm refers to an algorithm that has been hosted on an embedded development board, verified, and performance metrics analyzed. Based on this definition, the control algorithm presented in this thesis is classified as immature. The process and problems experienced transitioning the algorithm to a development board is similar to the process and problems that will be encountered during the migration to flight hardware. Therefore, the lessons learned during the migration to a development board will be invaluable experience for the eventual migration to flight hardware.

2. Verifying the Algorithm

Control algorithm verification compares the embedded control algorithm's solution to the baseline algorithm, the MATLAB^{TM} control algorithm. A suite of standard test scenarios will be executed on each of the algorithm variants: MATLAB^{TM}, stand-alone, and embedded. Each control algorithm will execute the same spacecraft control maneuver. The three control signal results will be recorded, compared, and analyzed. One important factor when analyzing control signal solutions is precision. Control signal precision should be tempered relative to the host spacecraft's torque device. Control solution precision to sixteen decimal digits is not required if the spacecraft's torque devices are only sensitive to three significant digits.

3. Performance Measurement

The initial migration onto an embedded development board may result in a performance reduction. In comparison to personal computers (PC), most embedded programs and algorithms operate using slower processors with less memory resources. The faster embedded microcontrollers operate in the 200 to 500 MHz[16] frequency range. New desktop and laptop computers operate in the 1.5 GHz to 3.8 GHz frequency range. Embedded development boards host approximately 32-Megabyte of RAM and 16-Megabyte of ROM. Their desktop and laptop computer counterparts are capable of hosting RAM memory sizes in excess of 1 Gigabyte and hard drives in the two hundred Gigabyte range. The disparity between the embedded and personal computer system's clock frequency and resource capacity provide the personal computers with a significant performance edge.

Personal computers are generic computing devices and therefore must be capable of handling a variety of programs, applications, and algorithms. To provide this broad capability, Personal computers maintain robust software and hardware features. The conglomeration of the additional hardware and resident software processes hosted within a personal computer to handle the variety of tasks is often referred to as overhead. Overhead contributes directly to the system's power consumption, memory capacity, and computational requirement. While the PC features support broad capabilities, streamlined performance suffers. The control algorithm currently operates within MATLAB™ running on Microsoft® Windows XP, service pack 2. A Dell Dimension 4400 computer is the host platform for the research reported in this document: Pentium® IV 1.8GHz, 512 MB RAM, 400 MHz FSB, NVIDA GeForce2 MX/MX400 64 MB video card, and Maxtor® 6E040L0 hard drive. In contrast, a potential embedded development board hosts a 200 MHz processor and 64 MB of RAM[17].

The control algorithm's migration to embedded hardware introduces processing, memory, and power limitations could slow the calculation rate and may adversely affect precision. An embedded computer is normally constrained by strict power, size, and

[16] Xilinx MicroBlaze™ in a Virtex 2 Pro FPGA and an AMCC 440EP PowerPC.

[17] Axiom Manufacturing. CML-5485 Development Board with BDM. Retrieved 21 Nov. 2005, from < http://www.axman.com/cgi-bin/products.pl?ProdName=CML-5485W;.State=Show>

weight limitations. These three constraints form a trade space from which the final embedded computer design emerges. Because of these constraints, the processor clock rates, computational capability, and memory capacity is normally less than a PC. Furthermore, many current embedded processors, like the popular ARM® series, do not contain a hardware floating-point unit. The processor speed and memory size of the host system to potential development board mentioned above contrasts the significant computational difference between embedded computers and desktop PCs. However, embedded systems normally provide computational services for a limited scope of work; therefore, the overhead resources may be removed to improve computational performance. Estimating the control algorithm's embedded performance is difficult due to these competing effects. The control signal generation time and precision will be key metrics for comparison and analysis.

B. COMPONENT REQUIREMENTS

The following subsections provide a broad overview of the necessary development board hardware and system support capabilities. At the time of writing, the decision concerning whether academia or industry would perform the phase two migration had not been determined. Therefore, the assumption is made that a student will perform the work. While it is assumed that the student performing the work holds an undergraduate degree in Engineering, it is understood that the student performing the work may not hold an undergraduate degree within the field of the work being performed. The discussion serves to identify hardware and lab capabilities required to pursue phase two development; it is not intended to be an authoritative exposition. An impressive reference for further study concerning embedded systems and architectures is a book recently written by Tammy Noergaard[18]. A single book rarely bounds the spectrum of topics facing an embedded system development team.

[18] Noergaard, Tammy. (2005). *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Oxford: Elsevier, Inc.

1. Host System

The term "host" is used to identify the general-purpose computer used for code development while "target" refers to the embedded development board. The host system is a personal computer that performs the following functions: developing source-code, compiling, linking, locating, downloading, and running the remote debugger. The progression from source-code to executable program for both PC and embedded system is displayed in Figure 6.

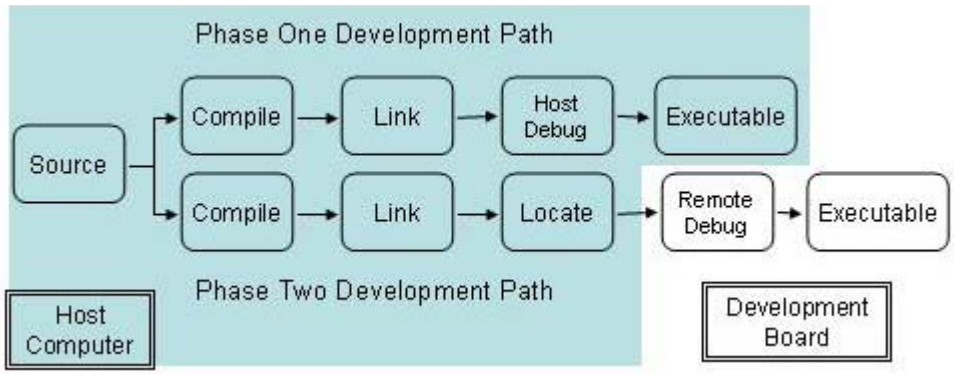

Figure 6. Software Development Flowchart

Most modern personal computers meet the system requirements for the programming-development environment, board support package utilities, and are capable of performing the host system duties. Phase one discussed writing source-code, compiling, and linking. The compiler and linker in phase two operate in the same manner as their phase one counterpart; however, the phase two compiler and linkers are specific to the embedded development board hardware. Since the compiler and linker behavior are the same, they will not be discussed further. Locating, downloading, and running the remote debugger are specific to embedded development and will be discussed briefly.

a. Locating

Locating converts compiled and linked code into an executable binary image. The locating process is performed by an application running on the host computer. Once located, the code is considered a "relocateable program." The binary

image may be downloaded into the target's memory and executed. Locating is often labor intensive and requires a working knowledge of the development board's memory configuration and operation. Most locator tools are not part of the programming-development environment. The locator is often offered as a software program, which is part of a manufacturer's board support package (BSP).

b. Loading

Loading transfers the executable binary file into the target board's memory. The transfer occurs over a communications link between the host and target. A common communication link between the host and target is a serial link. However, modern development boards are equipped with USB and Ethernet ports. These advanced ports may be available for application development if the development board is running a stand-alone operating system. The USB and Ethernet links will significantly reduce the loading time and expedite remote debugging efforts.

c. Remote Debugger

A remote debugger allows the developer to interact with the target hardware to study, locate, and correct errors in the code. The remote debugger is comprised of two parts. The first part is a graphical user interface (GUI) which runs on the host computer. The second part is the key element and runs on the target. This component provides the hardware control and reporting capability to the GUI. The remote debugger software is often purchased from the development board manufacturer, as part of the board support package bundle.

2. Target

The target is the embedded system, the development board. Many development boards are available for purchase on the commercial market. A basic embedded system and development board consists of, at a minimum, a processor, memory, input-output ports, and a clock. This section addresses necessary board components, capabilities, and

features required to host the control algorithm. The requirement for a system clock is commonly understood and will not be discussed.

a. The Processor

The processor interprets instructions, executes instructions, and passes data. It is a core component of all computational devices and has become a common term within today's society. Therefore, this thesis will not dwell on the specific capabilities; only identify important differences between a generic and an embedded processor. The term generic processor and central processing unit (CPU) are often used as synonyms. The term CPU frequently refers to generic microprocessors such as the Intel® Pentium® or AMD™ Athlon™ series used in personal computers. Embedded processors are commonly referred to as microcontrollers. Microcontroller designs are intended to be inexpensive yet possess greater self-sufficiency than their general-purpose counterpart. The self-sufficiency is introduced through hosting input-output (I/O) and memory features on the microcontroller die. Cost savings and self-sufficiency comes with a price. Microcontrollers generally cannot execute instructions or perform computations as quickly as their general-purpose counterparts perform.

Unlike the modern CPUs mentioned above, microcontrollers do not necessarily contain a floating-point unit (FPU). FPUs provide processors the ability to perform floating-point math via hardware, vice a software emulator. Hardware execution of floating-point math is faster but requires additional hardware circuitry. Floating-point emulation is slower and requires additional software code, which must be stored on the embedded system.

The development board selected for phase two development should contain a floating-point unit. A FPU will help maintain the control algorithm's current accuracy and speed of execution. DIDO and SNOPT are intrinsic software modules to the control algorithm. Both of these software modules are dependent upon floating-point calculations[19]. Additionally, variables created in the MATLAB™ script files, not declared as integer or single precision, are stored as double precision floating-point

[19] Murray, Walter and I. M. Ross. Personal interview. 22 Apr. 05.

numbers, by default[20]. A review of the control algorithm script files show numerous, numerically undeclared, variables meeting this condition. Performing floating-point emulation will slow the control algorithm's execution. Xilinx's MicroBlaze™ v4.00 FPU boasts a performance improvement factor of 6 times for Joint Photographic Experts Group (JEPG) operations, 50 for Fast Fourier Transforms (FFT) manipulations, and 120 for finite impulse response (FIR) filtering over software floating-point operations[21]. One goal of embedding the control algorithm is to improve the control algorithm's execution time, thus broadening the potential application. Utilizing an onboard hardware FPU will improve the control algorithm's embedded performance.

The development board should contain a double-precision floating-point unit to remain consistent with the existing control algorithm results and analysis. The previous paragraph details the control algorithm's dependency on double-precision floating-point math. Changing the development board to single-precision floating-point operations without analyzing the affects has the potential to allow coding or hardware error to propagate. Future research should analyze the effects of single-precision floating-point math operations on the control algorithm's accuracy and execution time. In the interim, microcontrollers containing double-precision FPUs are available. Table 2 provides an abbreviated list of FPU the more popular microcontrollers. The core column identifies whether the microcontroller is an ASIC chip, hard-core, or instantiated within a Field Programmable Gate Array (FPGA), soft core. Absent from the table is mention of ARM® processors. ARM® processing cores do not contain hardware FPU[22,23]. If future analysis indicates single precision calculations are sufficient, the microcontroller's FPU can be shifted into a single precision mode and the results analyzed.

---

[20] The MathWorks. MATLABTM Online Programming Documentation. Retrieved 17 Nov. 2005, <http//: http://www.mathworks.com/access/helpdesk/help/techdoc/MATLABTM_prog/ch11_st3.html>

[21] Xilinx, Inc. MicroBlaze™ Floating-Point Unit. Retrieved 17 Nov. 2005, from < http://www.xilinx.com/ipcenter/processor_central/microblaze/microblaze_fpu.htm#features>

[22] ARM®. ARM® Technical Support FAQ. Retrieved 18 Nov. 2005, from <http://www.arm.com/support/vfp_support_code.html>

[23] ARM®. ARM® VFP10 Coprocessor. Retrieved 18 Nov. 2005, from <http://www.arm.com/products/CPUs/VFP10.html>

Processor	Core	FPU	Precision	Company
Microblaze	soft	yes	single	Xilinx
TC1796 (AUDO-NG)	hard	yes	single	Infineon
PowerPC 440EP	hard	yes	both	AMCC
MPC5200	hard	yes	double	Freescale
TSC695F	hard	yes	double	ATMEL
PowerPC 405	soft	capable	double	Xilinx

Table 2. FPU Microcontrollers

b. Operating System

Using a standalone operating system on the development board will expedite the migration process. Embedded programs can integrate program functionality and board operating software into one executable program. However, this integration is normally done with a simple or very mature program. The control algorithm project meets neither of these criteria. An onboard operating system will allow the developer to focus on debugging and refining the algorithm. Combining an immature and untested algorithm with the development board's operating system will make differentiating algorithm or operating system errors very difficult.

A standalone-embedded operating system will allow the developers to focus on refining the algorithm. Without an operating system, the developers will need to write and test software routines to handle basic board operations. The developers would need to write code for operations such as input-output, interrupt handling, and multitasking. These are only a few of the numerous functions and utilities that reside within, are executed by an operating system, and are often transparent to the user. Developing an "in-house" embedded operating system is not insurmountable, especially for an experienced operating system programmer. However, this focus of this migration effort is the control algorithm, not embedded operating systems. This project's time and effort would be better utilized focusing on the control algorithm and purchasing a commercially available embedded operating system.

Several commercial-off-the-shelf (COTS) real-time operating systems (RTOS) are available for immediate use. There is a subtle, but notable, difference between operating systems and RTOS. As the name implies, the Real-time Operating System supports programs that must provide results in real time, like the control

algorithm. RTOS are compiled for operation on a specific target microcontroller. The recommended FPGA development board supports two soft-core microcontrollers: MicroBlaze™ and PowerPC 405. Several RTOS support these microcontrollers[24]. Two notable RTOS' among the list are Nucleus RTOS™ from Accelerated Technologies, Inc. and MontaVista™'s Professional Edition 4.0 Linux RTOS. Product data sheets for both RTOS' are provided in Appendix B. The Nucleus RTOS supports the MicroBlaze™ and PowerPC microcontrollers. The MontaVista™ Linux RTOS supports almost all popular hard microcontrollers and the soft PowerPC. The Linux RTOS does not currently support Xilinx's proprietary MicroBlaze™ microcontroller. Literature review and discussions with MontaVista™ indicate that the Linux RTOS would meet the needs of the project. However, the cost of the Linux RTOS is high, $9,200 per retail copy[25]. Reduced pricing options were not pursed at this stage of the project. The price includes the RTOS, software diagnostics, and user utilities to assist embedded program development. Over the course of a development project, professionally supported, mature development utilities are often worth the added expense.

c. Memory

Development board memory is provided in three basic forms: read-only memory (ROM), random access memory (RAM), and cache. Read-only memory (ROM) stores the embedded system's programs, including operating system if present. ROM is a non-volatile memory device. Non-volatile memory retains the state information if the system's power is interrupted. Most embedded systems do not contain rotating memory storage systems, such as traditional hard drives found in personal computers. Embedded systems tend to utilize solid-state memory technology such as programmable read-only memory (PROM), electrically programmable read-only memory (EPROM), or electrically erasable programmable read-only memory (EEPROM).

[24] Xilinx, Inc. Alliance Embedded Program Member List. Retrieved 29 Nov. 2005, < http://www.xilinx.com/ise/embedded/epartners/listing.htm>

[25] Quesenbury, Ann. MontaVista Software, Inc. Phone conversation. 16 Sep. 2005.

Random access memory (RAM) provides storage space for pending and temporary calculations and information required by the operating system and applications. RAM is a volatile memory storage device. Information stored in RAM is lost if the system's power is interrupted.

Level 1 cache is high-speed memory, normally located on the same silicon die as the microprocessor. Program information is loaded into the cache from ROM upon program initialization. Programmers accelerate the execution of programs by loading large, frequently used segments of a program into the level 1 cache. The amount of cache memory available to the development board is determined by the user's microcontroller or development board selection.

Calculating development board RAM and ROM requirements prior to the completion of phase one is difficult. After phase one completion, stand-alone control algorithm application, the algorithm's ROM requirements can be estimated from the size of the control algorithm's executable file. This value will be an estimate since the control algorithm's size will not be the same for the embedded variant. The embedded system has a different processor and operating system. This difference will require the use of different libraries and assembly code. However, this method will provide a reasonable estimate. The RAM requirements can be estimated using Windows XP's Task Manager. By selecting the Performance tab, the computer system's memory usage may be monitored. The algorithm's RAM requirements can be estimated by recording the peak memory usage during the algorithm's execution and subtracting the memory usage without the control algorithm running. A similar process can be performed using a Linux-based system.

Phase two development can proceed in parallel with phase one using a low-confidence estimation. Development programs are not always afforded the luxury of waiting until a definitive hardware accounting is available. Table 3 provides a summary of the estimated control algorithm size. The detailed spreadsheet supporting the summary is provided in Appendix A. The operating system requirements were obtained from vendor product sheets and modified by information provided during phone

conversations with the vendor[26,27]. Since this is a gross estimation, a factor of 100% will be added to the development board's ROM and RAM requirements. A conservative assumption is that the entire program must be resident in RAM during operation; therefore, potential development boards should contain a minimum of 23-Mbyte of RAM to be considered as a host for the control algorithm.

Component	Size (MBytes)
S/C Model	0.098
DIDO Files	0.539
MATLAB Libraries	4.447
SNOPT	5.16
RTOS	1.1
Control Algorithm:	**11.344**

Table 3. Control Algorithm Code Estimate

d. Input-Output

Input-output (I/O) ports allow the host computer and other electronic devices to communicate with the target. Traditionally, the I/O link has been a serial communications link. Serial communication links can lead to long delays when loading a large program to the target. Modern development boards contain Ethernet and USB ports. These advanced ports are accessible to the user if the development board is running an operating system, e.g. Linux, with Ethernet and USB support.

e. Board Indicator

The presence and type of board indicator should be factored into the development board selection. Board indicators range from light emitting diodes (LEDs) to liquid crystal displays (LCDs). These devices are invaluable tools when attempting to configure, operate, and debug a development board with a new operating system or program. The indicators can be programmed to flash or display simple communication sequences to verify basic board level operation.

---

[26] MontaVista Software, Inc. Linux Professional. Retrieved 16 Sep. 2005, from <http://www.mvista.com/products/pro/features.html>

[27] Murecky, John. MontaVista Software, Inc. Phone conversation. 16 Sep. 2005.

f. Development/Design Tools

The development and design tools included with a development board factor into the selection process. Development and design tools are a collection of software programs and utilities used to perform tasks supporting the embedded programming process. The term is being used in a very broad manner in this thesis. These tools range from self-written to professionally developed tools. Open source software provides another avenue by which these tools may be acquired. Development board and RTOS manufactures often provide a very useful collection of development tools to enhance their product's functionality. Too many development tools exist to coherently present in this thesis. Without adequate design and development tools, a development board is of little value. The cost of a development board is relatively inexpensive; $495 for an FPGA based board[28]. The price nearly doubles when the software development tools and intellectual property (IP) cores are added[29].

C. CANDIDATE DEVELOPMENT BOARDS

It is difficult to select an embedded development board without the completion of phase one. However, critical hardware requirements have been identified and discussed. These requirements can narrow the development board selection. First, the algorithm requires a fully functional microcontroller. Secondly, the current control algorithm requires floating-point math operations. In addition, the current control algorithm utilizes double-precision floating-point operations. Therefore, until analysis proves otherwise, the microcontroller will require access to a double-precision floating-point unit. Third, the estimated size of the algorithm's software is 23 Megabytes, rounded up. Lastly, the development board must be supported by a RTOS vendor. The RTOS is required to support real-time optimal-control calculations. The recommended development boards are detailed in Section VI, following the custom computing machine design discussion presented in the next section.

---

[28] Xilinx, Inc. Xilinx Virtex-4 ML-403 Embedded Platform. Retrieved 11 Nov. 2005, from <http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=HW-V4-ML403-USA&sGlobalNavPick=PRODUCTS&sSecondaryNavPick=BOARDS>

[29] Xilinx, Inc. Xilinx PowerPC & MicroBlaze™ Development Kit. Retrieved 30 Nov. 2005, from <http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?sGlobalNavPick=PRODUCTS&sSecondaryNavPick=Intellectual+Property&category=&iLanguageID=1&key=DO-ML403-EDK-ISE>

V. PHASE THREE – CUSTOM COMPUTING MACHINE (CCM)

A. HARDWARE ACCELERATION

Phase three development enlists an FPGA to decrease the time required for embedded control-signal generation without degrading precision. The FPGA will host a segment of the control algorithm, acting as an auxiliary math unit to the microcontroller and floating-point unit. ASIC algorithm implementations are often superior to their FPGA counterparts in the areas of power consumption, initialization, and clock rates[30]. However, the cost of designing and fabricating application-specific integrated circuits (ASIC) for rapid prototyping in a research environment is excessive[31]. Therefore, this thesis recommends the use of FPGAs to design and test algorithm modules. This approach does not concede that an FPGA variant will be the optimal platform for deployment, only development.

FPGA algorithm implementation does not avoid error introduction due to binary math operations. Precision degradation issues that have plagued ASIC math processors for decades directly are applicable to FPGA-based math computations. While dated, David Goldberg's paper, "What Every Computer Scientist Should Know About Floating-Point Arithmetic," explains the challenges facing binary floating-point computations[32]. More than a decade has past since the paper's publishing and numerous algorithms and libraries developed to mitigate error effects; yet, the basic concepts presented in his paper remain relevant.

VHDL[33] and Verilog[34] are the two dominant hardware descriptive languages (HDLs). These languages translate a hardware design into the digital format required for

30 Bartos, Frank J. (2005). Chip Wars: ASICs Versus FPGAs. *Control Engineering*. Retrieved 20 Nov. 2005, from < http://www.manufacturing.net/ctl/article/CA607224>

31 Bursky, Dave. (2005). We Must Hold The Line On Soaring ASIC Design Costs. *Electronic Design*. Retrieved 20 Nov. 2005, from <http://www.elecdesign.com/Articles/Index.cfm?AD=1&ArticleID=1955>.

32 Goldberg, David. (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. *Sun Microsystems*. Retrieved 30 Nov. 2005, from <http://docs.sun.com/source/806-3568/ncg_goldberg.html>

33 Hwang, Enoch O. (2006). *Digital Logic and Microprocessor Design with VHDL*. Canada: Thompson.

34 Brown, Stephen and Zvonko Vranesic. (2002) *Fundamentals of Digital Logic with Verilog Design*. New York: McGraw-Hill.

FPGA or ASIC design implementation. Both are used for low-level hardware development. Traditionally, VHDL and Verilog have not provided the high-level algorithm abstraction capability enjoyed by C. VHDL shares a pedigree with Ada while Verilog shares its pedigree with C. Despite VHDL's association with Ada, a programming language that is often shunned, VHDL has become a dominant language for FPGA development. Xilinx's Project Navigator 6.2i and ModelSim® SE 5.8a integrated development environments (IDE) were used to test FPGA modules during this research. The mentioned IDEs, and subsequently the Xilinx product line, were not selected based on superior performance with respect to their competitors; rather, their selection was simply based on cost, schedule, and availability.

B. MODULE IMPLEMENTATION

A single algorithm module is targeted for hardware implementation. One approach would be to migrate the entire control algorithm into hardware. However, performing the transition in one development step is challenging and cost prohibitive in an academic environment.

1. Ultimate Goal

The ultimate migration plan maintains the spacecraft model in executable software and incorporates the remaining portion of the algorithm into hardware. This configuration, displayed in Figure 7, provides control algorithm flexibility. Maintaining a software spacecraft model allows the incorporation of spacecraft design changes due to manufacturing problems or engineering changes. Furthermore, this hardware-software configuration provides the opportunity for the hardware to be coupled to practically any control system model, not limited to spacecraft. While this configuration is the program's ultimate goal, the work breakdown is still too large and cumbersome for the guidance and control lab's current staffing, expertise, and facility.

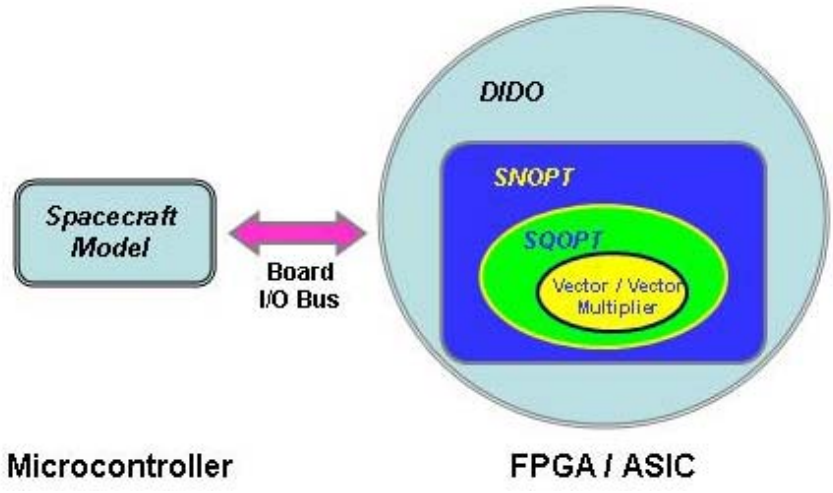

Figure 7. Ultimate Control Algorithm Goal

2. Proposed Goal

A single function will be migrated into hardware in phase three. This approach is based upon generating a plan that is measurable and achievable. As discussed above, migrating a large portion of the control algorithm in one development effort is a challenging task. Establishing and maintaining the personnel expertise and facility capability to migrate software algorithms into FPGA-based hardware is a development effort in itself. Given the current level of personnel experience and lab capability, attempting to implement the complete migration in one effort may prove too daunting. Migrating small control algorithm functions at the onset will serve to establish and mature the migration process and build the requisite expertise. Furthermore, migrating individual functions modularizes the development into achievable and executable tasks.

3. Targeted Function

The candidate function for initial hardware implementation is a vector inner-product multiplier. The ideal candidate function for hardware implementation is a repetitive, discrete math calculation. At the innermost core of the control algorithm resides the SNOPT function[35]. One of the fundamental calculations buried within the sub-functions of the SNOPT solver is vector multiplication. SQOPT is a sub-function to

[35] Due to the proprietary nature of the SNOPT algorithm, a detailed discussion concerning its internal structure is not offered. Additional SNOPT information may be obtained from Stanford University.

SNOPT and is dependent on large-vector inner-product calculations. Figure 8 provides a pictorial representation of the SNOPT to inner-product call sequence. The vector lengths range from ten thousand to fifty thousand real-numbered elements. The control algorithm calls SNOPT repeatedly during control signal generation.

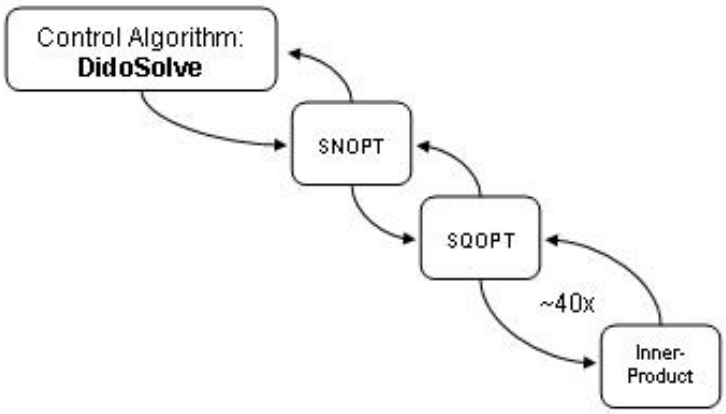

Figure 8. Inner-Product Call Sequence

In turn, SNOPT performs major and minor iterations that call the SQOPT sub-function several times. During each call, SQOPT performs numerous inner-product calculations prior to returning a solution to SNOPT. According to the algorithm's author, approximately forty inner-product calculations are performed each time SQOPT is called[36]. Figure 9 displays the control algorithm's structure after the creation of the inner-product multiplier.

[36]Murray, Walter. Personal interview. 22 Apr. 2005.

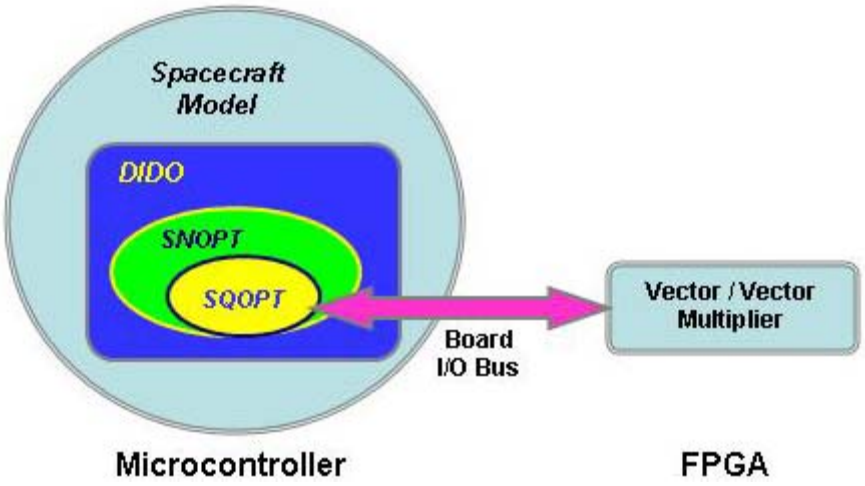

Figure 9. Control Algorithm with Inner-product Multiplier

4. FPGA Function Implementation

There are three development paths available to implement an FPGA inner-product core. The first two paths leverage the use of existing FPGA modules. The first method requires a notable monetary expenditure but shortens the development and implementation time. The second path utilizes open source FPGA modules and assumes greater compatibility and stability risk. The last path designs a new inner-product multiplier, avoiding the large monetary expenditure but extending the development time.

a. Modular Implementation – Commercial

The first development path involves leveraging existing commercially developed FPGA core components, commonly called intellectual property (IP) cores. For development purposes, the FPGA will be connected to the microcontroller via a high-speed data bus. This configuration is analogous to the old Intel® 386 CPU to 387 math co-processor design, circa 1987. Figure 10 is provides the conceptual layout of the development design.

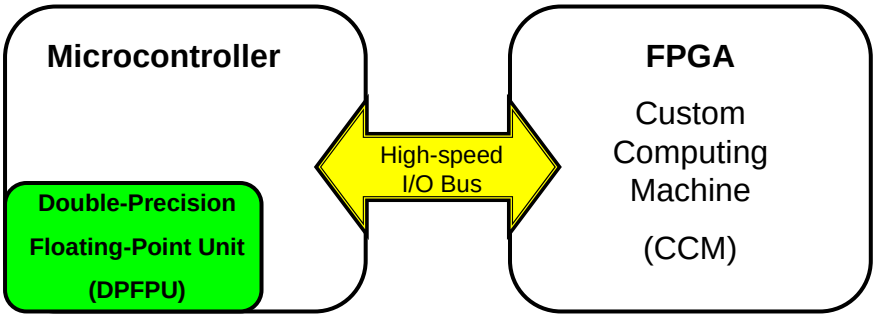

Figure 10. Conceptual FPGA Implementation

An inner-product multiplier may be constructed using existing double-precision floating-point IP cores developed by Nallatech[37]. Figure 11 is a block diagram of the proposed modular inner-product processor (IPP). Yellow blocks identify the pipelined Nallatech IP cores. Housekeeping commands such as clear, clock, and reset have been omitted for clarity. The required clock cycles for double-precision floating-point conversion, multiplication, and accumulation are displayed along the bottom. A copy of Nallatech's product sheet is included in Appendix C. Based on the datasheet, the adder module is the limiting IPP core component, 193 MHz clock frequency. Performance estimations are calculated with the IPP implemented as a co-processor to the microcontroller on the development board.

When calculating performance estimations in the co-processor configuration, a microcontroller to co-processor bus frequency is assumed. The control algorithm's intended host is a space-based platform. Therefore, the bus frequency limit was set conservatively at 50 MHz. This bus frequency is below the 66 MHz currently used by two commercial space processor vendors, SEAKR Engineering, Inc.[38] and EMS Technologies[39]. The 48.25 MHz clock rate is the assumed FPGA-to-microcontroller bus frequency. The input, output, and transfer clock counts are also assumed.

---

[37] Nallahtech. Double-Precision Floating-Point Core. Retrieved 18 Mar. 2005, from < http://www.nallatech.com/mediaLibrary/images/english/3269.pdf>

[38] Jungkind, Dave. SEAKR Space Processor Cards. E-mail to Ron Moon. 06 Dec. 2005.

[39] EMS Technologies. ESP603e PowerPC Space Processor Card Data Sheet. Retrieved 30 Nov. 2005, from < http://www.emsstg.com/pdf/esp603.pdf>

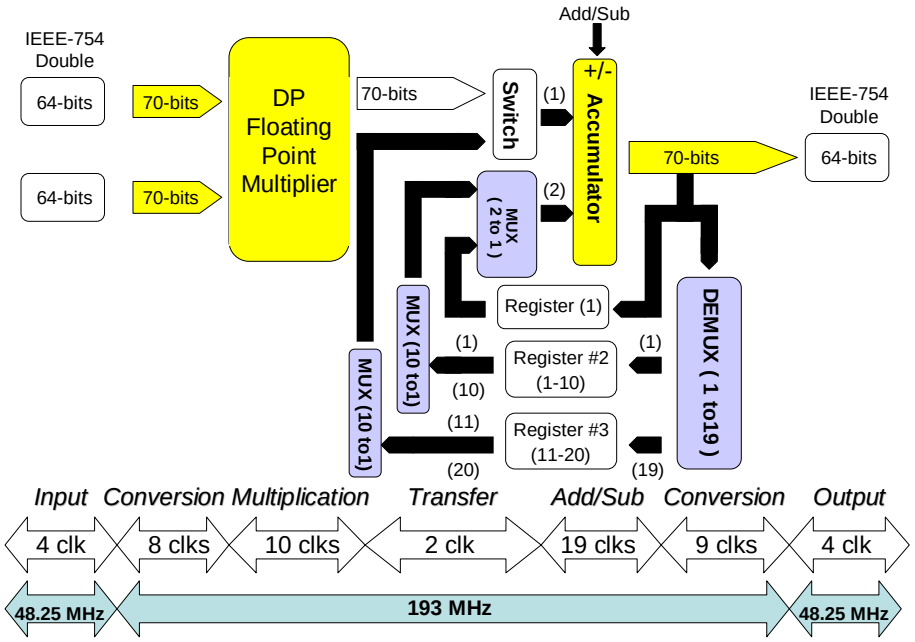

Figure 11. Modular Inner-Product Processor (IPP)

Multiple 64-bit input busses will be required feed the IPP's pipeline. The disparity between the input-output (I/O) bus frequency and the IPP core operating frequency would cause the IPP's pipeline to receive data once per four clock cycles. If this condition were allowed to exist, the IPP would operate in a data-starved condition. Therefore, eight 64-bit data busses will be required between the microcontroller and IPP, four data busses per 64-bit input. Current FPGAs provide I/O pins in excess of the required 512 pins. Each of the two input's four data busses will be multiplexed into a single data stream. Each clock cycle will cause the multiplexer to shift inputs, thus providing a continuous feed of data to the IPP. This data-feed design is an interim work-around and is not viewed as a desirable design, but necessary given the current state of FPGA technology.

The data-bus disparity identifies the significant problem of maintaining data-flow to a high-throughput custom computing machine. The computer industry experienced a similar problem when the desktop computer's CPU and FPU were mounted as separate devices on a motherboard. The significant I/O delay between the FPU and CPU led to the hosting of the FPU on the same die as the CPU. This dual

hosting allowed the construction of a high-speed data-bus between the two devices. Custom computing machines would benefit greatly from a similar design technology; hosting FPGA fabric on the same die as a hard-wired microcontroller. This arrangement would provide high-performance CCM capability. The I/O bottleneck can also be avoided if the IPP is connected to a soft-core microcontroller on the same FPGA; the IPP can potentially run at the clock rate as the microcontroller. These two configurations would not require the data-multiplexing scheme. The hard-wired microcontroller option is the desired solution. Hard-wired microcontrollers are capable of operating at a higher frequency; furthermore, they are not as susceptible to ionizing events in space applications.

The proposed inner-product processor utilizes a Multiply and ACcumulate (MAC) methodology. The IPP accepts two IEEE-754 double precision numbers. Both numbers are converted into a 70-bit Nallatech floating-point format and multiplied. The resulting product is transferred to accumulator input one. The accumulator is configured to operate as an adder. The accumulator's second input normally contains the accumulated inner-product value; provided by the multiplexer via register one in the inner, 70-bit, feedback loop. The outer-loop de-multiplexer feeds registers two and three. The outer-loop de-multiplexer continually steps through an address of one to nineteen; this stepping process populates registers two and three with the accumulator's output. Register three's last position, twenty in Figure 11, contains a fixed zero. The 50,000-element accumulation and register population process continues until the final vector product. An "end-of-vector" flag, not shown, accompanies the final vector product.

At the end of a calculation sequence, the accumulator's pipeline must be cleared to obtain the correct final value. The pipeline clear procedure is required because the accumulation process is carried out in a recursive manner. The accumulator's nineteen partial-sums must be added to obtain the total, final sum. These partial-sums were stored in registers two and three by the de-multiplexer during the MAC process. When the end-of-vector flag reaches the accumulator's output, registers two and three contain the last nineteen partial sums. An important note, the partial sum capture process is independent of the vector length. However, the registers' size is dependent upon the number of accumulator stages.

Once a pipeline clear is initiated, the accumulator's input switch and multiplexer change positions. With the new input paths, the accumulator now operates as a nineteen-stage pipeline adder. Register two and three begin feeding the nineteen partial sums through the pipeline adder. The summing sequence will require a five-layered process, determined by Equation 5.1 and rounded up.

$$Sum\min g\ Layers = \log 2(stages) = Log\,2(19) \qquad Eq.\,5.1$$
$$= 4.358 \approx 5\ Layers$$

Figure 12 provides a register state diagram for the following pipeline clear process explanation. The first layer begins with nine partial-sum pairs and one non-paired number. The non-paired number remains in register two, position ten, for the next layer calculation; it does not pass through the accumulator. The second layer contains five partial-sum pairs. The third layer contains two partial-sum pairs and one non-paired number. The non-paired number remains in register two, position three, for the next layer calculation; it does not pass through the accumulator.

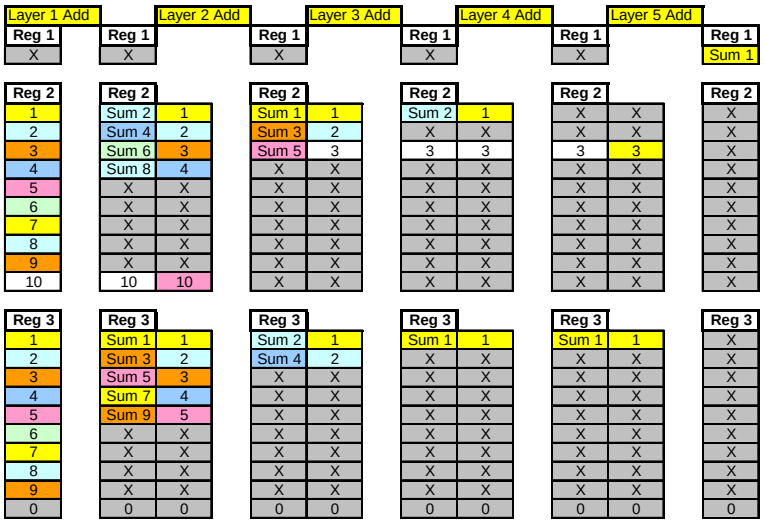

Figure 12. Pipeline Clear Process

The fourth layer contains one partial-sum pair and one non-paired number. The non-paired number remains in register two, position three, for the next layer calculation; it does not pass through the accumulator. The fifth layer contains one partial-sum pair; the sum is calculated and placed in register one. The "X's" represent don't care states. While the addressing is critical, the calculation and register storage process does not corrupt the original nineteen partial sums. The pipeline clear process requires one hundred and thirteen clock cycles. The clock cycle derivation is provided in Equation 5.2.

$$\begin{aligned} Pipeline\ Clear\ Clock\ Count &= (19+9)+(19+5)+(19+2)+(19+1)+(19+1) \\ &= 113\ Clock\ Cycles \end{aligned} \qquad Eq.\ 5.2$$

After the pipeline clear, the vector inner-product is complete and initiates the conversion process back to the IEEE-754 double precision format. The accumulator's pipeline is zeroed to prevent corrupting future calculations. The IEEE-754 solution is transferred to an I/O register, setting a "ready" bit, which may be used for polling or interrupt request communication with the microcontroller. The input conversion to accumulator output requires forty-three clock cycles per vector pair. The pipeline clear process requires one hundred and thirteen clock cycles. The conversion to IPP output requires an additional thirteen clock cycles; the output conversion cost is required once per vector inner-product calculation.

An algorithm exploitation technique is expected to reduce the inner-product calculation time, significantly. The accumulator's feedback loop contains two major paths: an inner and outer loop. The outer loop is the key factor that allows the IPP to exploit SNOPT's solution convergence behavior. During each SNOPT major-minor iteration sequence, SQOPT requests approximately forty inner-product vector calculations. After each individual vector inner-product calculation, SQOPT determines if the solution condition has been achieved. If the solution condition is not met, another vector inner-product calculation is requested. After the first vector inner-product calculation, the remaining thirty-nine vectors share kernel elements. The first inner-

product value may be stored and utilized to reduce the number of subsequent inner-product multiplications and accumulations. Subsequent vector inner-product calculations can avoid multiplying and accumulating 50,000 elements by subtracting off the unique portion of the original vector inner-product. The new vector's inner-product value is then obtained by adding the new unique vector inner-product elements. This process is called segmenting and is explained further in the following paragraph.

Segmenting the vector within the IPP hardware during the vector inner-product calculation will reduce SQOPT's execution time. The user identifies a single element within the 50,000-element vector prior to the initial SQOPT call. The number of vector elements preceding and including the user-selected element is called the segment, or segment size, see Figure 13. A segment size of ten will be used for the analysis in this thesis.

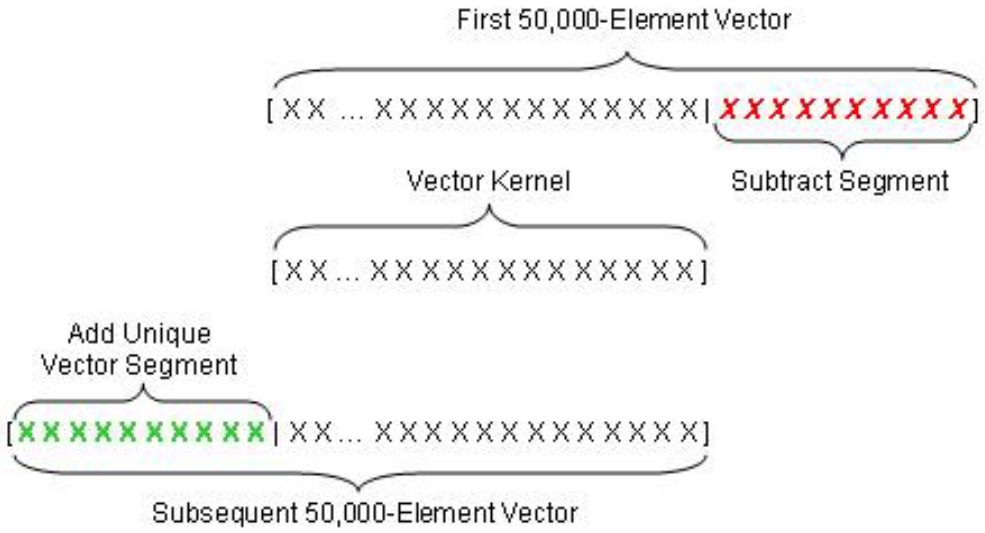

Figure 13. Segmentation Process

The IPP must calculate the first 50,000-element inner-product before the segmenting exploitation occurs. An end-of-vector flag accompanies the last vector element through the IPP to indicate the last vector element pair. When the end-of-vector flag reaches the output of the accumulator, a pipeline clear is performed, explained

previously. With the vector inner-product complete, the inner-product solution resides in register one of the accumulator's inner feedback loop. A copy of the solution is passed to the converter and sent back to the microcontroller. After the solution is stored in register one, the accumulator is reset, setting all pipeline stages to zero. The accumulator zeroing procedure must be performed to clear the residual numbers residing in the pipeline. The accumulator's input switch and multiplexer is ready to change state and apply the vector inner-product solution to accumulator input two as soon as the subtraction segment arrives. The subtraction segment elements are sequenced into the IPP, arriving at the accumulator's input one at the same time as the previous inner-product is sent to input two. Accumulator operation is shifted from addition to subtraction. The ten-element subtraction segment is provided to the accumulator operating in the subtraction mode. The effective result is the kernel vector-value required for the subsequent vector inner-product; see line two in Figure 13. This is not completely true since we have not performed a pipeline clear. We forego this pipeline clear until the segment addition is complete, saving one hundred and thirteen clock cycles. A nineteen clock cycle buffer is inserted to move the subtraction elements out of the accumulator pipeline prior to shifting to addition. During the buffer clock cycles, zeros are fed to the accumulator's input one. Accumulator operation is shifted from subtraction to addition. The subsequent inner-product is rapidly calculated by accumulating the unique vector segment products from the following vector; see line three in Figure 13. The final vector pair of the unique vector segment carries an associated end-of-vector flag. When the end-of-vector flag reaches the output of the accumulator, a pipeline clear is performed to complete the new vector inner-product accumulation. The new vector inner-product now resides on the accumulator's output. Segmentation reduces the cost for subsequent vector inner-products to approximately twenty multiply and accumulation operations, vice an entire 50,000. The vector segmenting process continues until the inner-product meets SQOPT's solution conditions, which are approximately thirty-nine segmented vector calculations.

The segmenting size is not fixed; however, the segment size must remain constant for a forty-run solution sequence. If the segment size does not remain the same, the vector kernel will not remain consistent, causing erroneous results. The SQOPT developers have not determined an optimal segment size exists, if one exists. Therefore,

the IPP design requires a control scheme allowing for a user defined segment size. Providing variable hardware segmenting requires additional FPGA resources and increases the IPP's control complexity. If an optimal, fixed segmenting size is determined, the IPP's segmenting control can be reduced to a simple counter slaved to the vector's "first element" flag. Side note: a close variant of the segmenting process can be exploited using current desktop computers. The SNOPT/SQOPT source code would need to be modified to execute the segmenting process using the CPU's FPU and registers. The IPP design in this thesis was pursued due to the desire to embed the control algorithm.

A SNOPT test case was defined to calculate the projected performance of the IPP versus the host system, Pentium® IV system detailed in section IV. The values provided in Table 4 were established to perform the comparison. The vector elements and inner-products per SQOPT call were established by Dr. Murray[40], SNOPT's author, and the remaining values by Dr. I. M. Ross[41], DIDO's author. These values represent the nominal occurrences for a single control algorithm solution.

Vector Elements	50000
SNOPT_Calls/Solution	100
SQOPT_Calls/Solution	1000
Inner-products/SQOPT_Call	40
Inner-products/SNOPT Solution	40000

Table 4. Nominal SNOPT Solution Vector Calculations

Using the values in Table 4, the IPP's performance is estimated using an Excel spreadsheet and compared to the host system. The host system's performance parameters were recorded using MATLAB™ 6.5.1. Two random 50,000-element vectors were created and their inner-product calculated. The inner-product calculation was repeated in four different loops: 40, 100, 1,000, and 40,000. The loop number corresponds to the number of times the inner-product was calculated in the particular

[40]Murray, Walter. Personal interview. 21 Jan. 2005.

[41] Ross, I. M. Personal interview. 21 Nov. 2005.

loop. Forty represents the average number of times an inner-product is calculated per SQOPT call. One hundred, one thousand, and ten thousand correspond to their respective Table 4 value. The looping tests were conducted to determine if the Pentium® IV's FPU utilized acceleration techniques, which were non-linear with respect to the number of inner-product calls. The calculation times were captured using MATLAB™'s "cputime" function. The total calculation time was divided by the number of inner-product operations performed to derive an average calculation time per inner-product. The M-file performing the baseline test is included in Appendix C. The host system's operating system, Windows XP Professional SP2, runs numerous background processes. Many of the running processes are not controlled by the user but could adversely affect the test time. Therefore, the inner-product test was repeated four times and the values averaged in an attempt to mitigate the background process effects, see Table 5.

Vectors	Run 1	Run 2	Run 3	Run 4	Each Avg.	
40	0.0043	0.0043	0.0035	0.0039	**0.0040**	seconds
100	0.0030	0.0036	0.0033	0.0031	**0.0033**	seconds
1000	0.0030	0.0038	0.0030	0.0030	**0.0032**	seconds
40000	0.0038	0.0043	0.0040	0.0038	**0.0040**	seconds
Run avg.	**0.0035**	**0.0040**	**0.0035**	**0.0035**	seconds	
Total avg.	**0.003606**	seconds				

Table 5. Pentium® IV Inner-product Calculation Time (50,000 elements)

An estimated IPP performance is calculated using the Figure 11 design and Nallatech's reported performance specifications[42]. Table 6 displays the clock cycles required for each operation. The FPGA addition/subtraction IP core module is the clock frequency-limiting component at 193 MHz. The IPP's implementation is a The CPU bus frequency is an assumed value and will be addressed later.

[42]Nallatech. Double-Precision Floating-Point Core. Retrieved 10 Mar. 2005, from < http://www.nallatech.com/mediaLibrary/images/english/3269.pdf>

Function	Cycles Double	Nallatech Freq (MHz)	Cycles Single
Multiplication	10	202	6
Add/Sub	19	**193**	14
IEEE to Nallatech	8	227	6
Nallatech to IEEE	9	244	8
Transfer Delay	2	n/a	2
CPU Bus (50 MHz max)	4	***48.25***	4
FPGA Clock Rate	n/a	**193**	n/a

Table 6. IP Core Clock Cycles and Frequencies

The formula in Equation 5.3 calculates the time required for the IPP to produce a full 50,000-element vector inner-product. Each of the three lines in equation 5.3 is in terms of time. The "transfer in" on line one of equation 5.3 accounts for the four clock cycles required to transfer the vector element pairs into the IPP. This transfer cost will accrue for each element pair, from the two input vectors. Therefore, the number of vector element pairs multiplies the "transfer in" cost. The frequency of the transfer in and out operates at the assumed CPU I/O frequency, 48.25 MHz, not the FPGA frequency.

$$50K\ Inner-product\ Calculation\ Time = \frac{(Transfer\_in * Vector\_pairs)}{CPU\ I/O\ Frequency} +$$

$$\frac{(Conversion + Multi + Transfer + Add + Vector\ Pairs) + Pipeline\ Clear + Conversion}{FPGA\ Frequency} + \qquad Eq.\ 5.3$$

$$\frac{Transfer\_out}{CPU\ I/O\ Frequency}$$

Line two of equation 5.3 accounts for the time required by the MAC process to calculate a 50,000-element inner-product. Each element pair is converted into the Nallatech format, multiplied, transferred to the accumulator, and added. The IPP is

pipelined; the clock count cost is the pipeline's stage length plus the length of the input vector. The clock cycles are assumed equal to the pipeline stages. The IP cores are proprietary and the internal structures are not available for examination. However, a senior Nallatech designer indicated that this assumption is reasonable[43]. After the vector elements are accumulated, the accumulator's feedback pipeline is cleared. Once the pipeline is cleared, the solution resides on the output of the accumulator in the 70-bit Nallatech format. The solution is converted back into the standard IEEE-754 format. The IEEE-formatted solution is returned to the microcontroller across the I/O bus. The Excel spreadsheet developed to tabulate the estimated IPP performance is included in Appendix C.

$$50K\ Inner-product\ Calculation\ Time\ (segmented) =$$

$$\left\{ \begin{array}{l} Single\_Vector\_IP \quad + \\ \{(2*Segment\_Size*Transfer\_in) \quad + \\ \left(\dfrac{Segment\_Subtraction + Segment\_Addition + Pipeline\ Clear + Conversion}{FPGA\_Frequency}\right) + Transfer\_out\}* \\ \left[\dfrac{Vector\_Calls}{SQOPT\_Calls} - 1\right] \end{array} \right\} * \frac{SQOPT\_Calls}{Solution} \qquad Eq.\ 5.4$$

Equation 5.4 calculates the time required to perform a segmented SNOPT solution. Note, Equation 5.4 lines two through five are mathematically in series, inline. The segmentation process requires one full 50,000-element inner-product. Line two of Equation 5.4 accounts for this cost. The segmentation exploitation begins by subtracting the segment value from the previous vector's inner-product solution; see line four of

[43] Dunn, Paul. Nallatech Double-Precision FP Cores. E-mail to Ron Moon. 23 Nov. 2005.

Equation 5.4. A segment subtraction contains the clock cycles identified in Equation 5.5. Since the IPP is pipelined, the required clock cycles will be the sum of the MAC pipeline, and segment size, see Table 6. Technically, the segment addition operation requires the same number of clock cycles since the accumulator's add and subtract operations is equal. Equation 5.6 is provided for completeness.

$$Segment\_Subtraction = converstion + multiply + transfer + subtract + segment\ size \qquad Eq.\ 5.5$$

$$Segment\_Addition = converstion + multiply + transfer + addition + segment\ size \qquad Eq.\ 5.6$$

Once the segment subtraction and addition are complete, the new vector inner-product has been calculated. The new inner-product is converted back to the IEEE-754 format and returned to the microcontroller. The segmenting cost is multiplied by one less the number of times that SQOPT calls for an inner-product. The minus one accounts for the first, full, inner-product that must be calculated. In our example, the segmenting process is repeated thirty-nine times; see Table 4. Lastly, line four multiplies the segmenting calls by the number of times that SQOPT is called by SNOPT during the course of a control algorithm solution; see Table 4.

		SNOPT Solution (Seconds)	Improvement Factor	Scale Factor
	Pentium IV 1.8 GHz	144.25	Baseline	Baseline
1	IPP Core w/out I/O	0.30	473.95	0.0021
2	wI/O w/out Segmenting	176.20	0.82	1.2215
3	wI/O w/Segment (DP/DP)	4.52	31.93	0.0313
4	wI/O w/Segment (SP/DP)	4.52	31.95	0.0313
5	wI/O OpenCores Seg. (DP/DP)	44.75	3.22	0.3102

Table 7. Estimated Inner-Product Processor Performance

Table 7 summarizes the IPP's estimated performance relative to the Pentium® IV's FPU. The table is based on calculating a complete SNOPT solution,

400,000 inner-products. Without exploiting the segmenting technique, the IPP's performance is significantly inferior to the Pentium® IV's FPU. This performance difference is primarily due to the difference in clock frequency between the two cores. The Pentium® IV core operates at 1.8 GHz while the IPP's core operates at 193 MHz. Introducing segmenting reverses the results; the IPP provides a significant performance improvement. Row four in Table 7 displays the projected performance improvement using a single precision multiplier. Theory and practice indicates that the precision of a 50,000-element multiply and accumulate solution is dominated by the accumulation process, not the multiplication process[44]. A single precision floating-point multiplier requires six clock cycles vice ten for double precision. Equations 5.3 and 5.4 were modified in the Excel spreadsheet to reflect the multiply savings realized in a single precision multiplier. The estimated performance improvement achieved with a single precision multiplier does not appear to provide a significant timesaving, roughly three ten thousandths of a second for an entire SNOPT solution. Therefore, it is recommended that the IPP design avoid the potential schedule risk and maintain the double precision multiplier, exploring the single precision multiplier design should time and resources permit.

b. Modular Implementation – Public

The second potential IPP development path involves constructing the IPP described in the preceding paragraph using publicly developed modules. An internet-based organization called OpenCores.org[45] hosts the development and distribution of open source IP cores. The development projects are primarily developed through a consortium of individuals. The OpenCores organization uses CVS[46] to maintain and distribute the latest version of an IP core along with providing the core's development pedigree. Usage of the OpenCores modules is governed by a document modeled after the Lesser General Public License[47].

---

[44] Loomis, Herschel. Personal Interview. 23 Nov. 2005.

[45] OpenCores Organization. Website. Retrieved 05 Mar. 2005, from < http://www.opencores.org>

[46] Opencores Organization. CVS Howto. Retrieved 26 Nov 2005, from < http://www.opencores.org/projects.cgi/web/opencores/cvs_howto>

[47] The GNU Operating System. GNU Lesser General Public License. Retrieved 26 Nov. 2005, from < http://www.gnu.org/copyleft/lesser.html>

The inner-product processor (IPP) may be constructed in the same format as Figure 11 using modules obtained from the OpenCores organization. The OpenCores site currently contains a CF Floating-point Multiplier and HCSA Adder[48]. The CF in the Floating-point Multiplier's title is notable and deserves a brief explanation. CF is an abbreviation for Confluence, a programming language that compiles into VHDL, Verilog, or C. The Confluence developers claim that CF provides high order functional programming, understandable source-code, and a two to ten time reduction in code size[49].

The IEEE-754 compliant multiplier may be configured to compute in single, double precision and combinatorial, or pipeline. The pipeline latency is four plus the mantissa accuracy[50]; for this analysis, fifty-six is utilized. The multiplier's clock frequency is assumed greater than 150 MHz. The Hierarchical Carry Save Algorithm (HCSA) adder accepts 128-bit operands and operates at 6.64 nanoseconds, 150 MHz[51]. The HCSA adder is assumed to require nineteen clock cycles to complete a pipeline addition. Using the specifications and assumptions presented, the OpenCores IPP performance is modeled using the same Equations, 5.3 through 5.6, and Excel spreadsheet. Row five in Table 7 shows the predicted the performance of the OpenCores IPP to rival the Nallatech-based design.

While the IPP could be constructed using the OpenCores multiplier and adder, this development path is not recommended unless the Nallatech IP cores cannot be purchased. As noted in the previous paragraph, many assumptions were made to estimate the OpenCores IPP performance. The performance estimate is a likely a best-case scenario. Furthermore, the OpenCores IP modules will require modification. The OpenCores HCSA IP module is not currently pipelined; therefore, the design must be modified prior to implementation. While the existing modules provide a starting foundation, they do not come with professional documentation. Attempting to

[48] Opencores Organization. Projects by category. Retrieved 01 Nov. 2005, from < http://www.opencores.org/browse.cgi/by_category>

[49]Confluence. Confleunce Overview. Retrieved 26 Nov. 2005, from < http://www.confluent.org/wiki/doku.php>

[50] Opencores Organization. CF Floating Point Multiplier. Retrieved 01 Aug. 2005, from < http://www.opencores.org/projects.cgi/web/cf_fp_mul/overview>

[51] Opencores Organization. HCSA Adder. Retrieved 01 Aug 2005, from < http://www.opencores.org/projects.cgi/web/hsca_adder/overview>

understand another engineer's design, without credible documentation, has the potential to consume more time than creating a new design. A professionally supported product has the benefit of technical support. A company will often allow the design engineer to be contacted to answer questions. While the OpenCores designers can be contacted, their response is not required, nor the response time known.

c. Custom Inner-product Processor

The final IPP design option is to forego the pre-fabricated IP core modules and design a completely custom FPGA core. This design could implement the scheme outlined in Figure 11 or exploit other implementation methods. Other methodologies exist by which vector multiplication and accumulation are exploited using parallel operations[52]. The investigation and pursuit of these designs are left to the student, or developer, employed to execute this migration phase.

Should the migration effort pursue the design of a completely custom FPGA core design, it is recommended that the design use a hardware descriptive language (HDL) such as VHDL or Verilog, previously discussed. An HDL provides three main benefits over schematic designs. HDL designed components and modules can be simulated immediately using the Xilinx or third party simulator, such as ModelSim®. Secondly, an HDL design is product or device independent; a common buzzword is technology independent. This provides design portability across different vendors, or among a vendor's own product line, to locate the most cost effective hosting device. Lastly, Xilinx claims that large designs are better managed using HDL design tools, vice schematic design tools[53].

[52]Loomis, Herschel. Personal Interview. 23 Nov. 2005.

[53]Xilinx, Inc. Design Entry and Synthesis. Retrieved 04 Dec. 2005, from < http://toolbox.xilinx.com/docsan/xilinx7/books/data/docs/dev/dev0014_5.html>

VI. FUTURE WORK ROADMAP

The three-phase migration process further subdivides into task elements. Figure 14 displays the relationship between the task elements and each phase. Each element is formulated to require the talents of a primary engineering discipline and focus on a single task. Each task element contains a recommendation regarding whether academia or industry should perform the task. It is recognized that the academia or industry recommendation may not be followed. Therefore, a recommended academic discipline for each task element will be included. Because each task element has the potential to be accomplished by a thesis student, each task element is designed to fit within a thesis student's schedule.

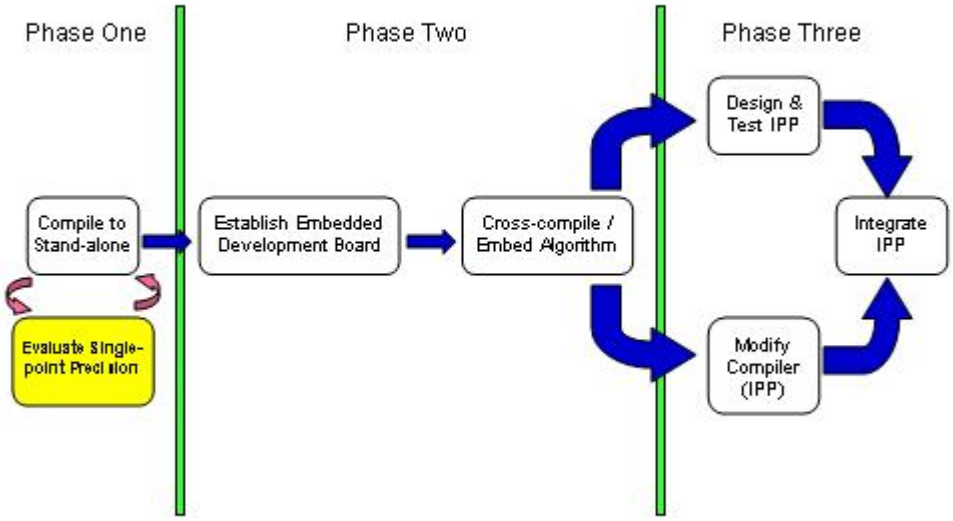

Figure 14. Migration Task Breakdown

A. PHASE ONE: SOFTWARE

Phase one contains two work elements; generating the standalone executable algorithm and evaluating single-point precision.

1. Stand-alone Algorithm

The hardware and work required to migrate the MATLAB™ control algorithm into a standalone program is outlined in the work, development, and task sub-sections. Of the two phase one tasks, the stand-alone algorithm is the most time consuming and challenging.

a. Scope of Work

Generating the standalone algorithm will require modifying, translating, compiling, and linking the existing spacecraft and DIDO functions. Once this work is complete and proper operation verified, the spacecraft and DIDO portion of the control algorithm must be linked with SNOPT. This two-step approach assists in the localization of error sources by "half-splitting" the overall control algorithm. SNOPT is provided by Stanford University as a "C" program. SNOPT does require installation and setup effort, which is outlined in the provided "help" files; see Appendix A. Once installed and configured, SNOPT may be provided test cases to verify proper operation. The spacecraft and DIDO modules have yet to be executed outside the MATLAB™ environment. Therefore, these modules will need to be modified to provide intermediate solutions prior to translation.

The MATLAB™ 6.5.1 compiler will be used to translate the MATLAB™ function files. The command and associated options were detailed earlier. Two C files are generated during the translation process: source and header. The Compiler can also compile the code. By default, the source code is compiled for the host platform's CPU and operating system. This setting can be changed by installing the desired compiler and configuring MATLAB™ to call the new compiler. The additional installation and configuration provides the ability to cross-compile for another platform. The steps to change MATLAB™'s default compiler is located in the installed MATLAB™'s help files and can be found within the support section of the MathWorks website. The translated code will be linked and compiled using a third-party integrated development environment; the reason for this recommendation was discussed earlier in this thesis and will not be readdressed.

Modifying each spacecraft and DIDO function module to display, or write to file, input and output results prior to translation will assist in isolating migration errors. Each spacecraft and DIDO mode must be converted into a function before the Compiler will translate the module. Modifying the MATLAB™ functions to display the results prior to translation will assist in testing. The translated function's operation may be verified by passing in known test cases. The translation process makes the source-code very difficult to read, even by an experienced programmer. Therefore, it is recommended that the modifications be made to the functions in the MATLAB™ script files. A simple main file can be written to pass a known test case to the function. The main file and function is then compiled, linked into an executable program. The program is executed and the results compared to the corresponding MATLAB™ function. Each of the MATLAB™ functions' operation has been previously verified. Therefore, if the corresponding compiled and MATLAB™ function results differ, a faulty compiled function is identified and can be corrected prior to linking it with other translated functions.

After the functions have been translated and tested individually, a programming-development environment is used to link the translated and compiled code into an executable program. The source, header, and MATLAB™ library files must be imported into a programming-development environment. The selection of the integrated development environment will be influenced by MATLAB™'s compiler library support. MATLAB™ 6.5.1 contains math library support for Borland®, Digital, lcc, Microsoft®, and WATCOM. The library support for each compiler is version dependent. The MATLAB™ library support for the different compilers and versions is the following: Borland® versions 5.0, 5.3, and 5.4; Digital 5.0 and 6.0; lcc versions not specified; Microsoft® Visual Studio® 5.0, 6.0, and 7.0; and Watcom 10.6 and 11.0. One could select and integrated development environment that does not support the identified compilers; however, this is strongly discouraged. The MATLAB™ libraries provide the necessary C code to perform calculations previously performed by MATLAB™ in the translated code. Selecting a non-supported compiler will require the programmers to write their own math library to perform the calculations. Writing a math library to

integrate into the control algorithm is a project, itself. Therefore, utilizing an integrated development environment using one of the identified compilers is strongly recommended.

An integrated development environment provides the user with powerful automated tools, depending on the selected environment. The translated functions and library brought into the development environment must be compiled and linked. Several professionally supported development environments now perform the linking automatically, in the background. The compiler examines the main, header, and function files and determines the dependency between the files. The resulting work of this examination is the creation of a "make" file. The make file describes the relationships and dependencies that exist in order to compile the code into an executable program. In the past, and with less capable development environments, the make file is manually written. Programmer competency in writing a make files is slowly dwindling due to the automated process performed by modern development environments. The Microsoft® Visual Studio® 6.0 integrated development environment was tested on a simple main and function file. The environment produced the make file in the background and generated the executable program with little effort. The Microsoft® Visual Studio® 6.0 integrated development environment was utilized simply because it was available and MATLAB™ provided library support; no other selection criteria were utilized.

The source-code is compiled and the individual modules linked into a standalone program. The linking process includes linking the main, function, and MATLAB™ math libraries into an executable program. The development environment is useful for debugging problems encountered during the compile and linking process. As mentioned earlier, each function should be individually translated, compiled, and proper operation verified prior to attempting to link and compile the entire algorithm. After the individual functions are verified, a short main file will need to be written to request the ordered attitude, which then calls the "Mag_Open_Main" function to initiate the control algorithm. Once the main file is written, it is recommended that function calls be added, individually, to the program. After each addition, the program should be tested and the results verified against the MATLAB™ variant. The process is repeated until all

of the functions are incorporated and the program is complete. Performing the outlined process is tedious; however, the process will quickly identify faulty functions.

b. Development Hardware and Software

Much of the development hardware and software required for this phase has already been discussed and is currently available within the guidance and control lab. The host system, Dell Dimension 4400, has MATLAB™ 6.5.1 and Microsoft® Visual Studio® 6.0 installed. Furthermore, should the developer desire to use the GCC compiler on a Linux operating system, the host system contains a rack-mount hard-drive system. A second, identical, hard-drive contains the Red Hat Work Station 4.0 operating system. The host system is capable of performing the phase one through three developments. Furthermore, maintaining the same host system provides each phase the ability to perform equitable control algorithm performance comparisons.

c. Task Assignment

The migration to stand-alone control algorithm should be accomplished by industry. A computer science professional is a better choice to perform the stand-alone application development. Generally, professional programmers are more proficient at using translators and integrated development environments. While graduate students are intelligent and work diligently, on average, they do not posses the same level of programming technical expertise. Furthermore, hiring a professional provides the flexibility to locate and hire a programmer intimately familiar with the translation and compiler tools for this project. The disparity in proficiency would, most likely, lead to a longer phase one development time, should a student be used to perform the work.

If the task is not assigned to industry, the migration to a stand-alone control algorithm should be performed by a Computer Science thesis student. The ideal candidate would be a Space Systems Engineering student following the computer science track, vice a student within the normal computer science curriculum. The Space Systems Engineering students understand the algorithm's application due to their controls, optimization, and dynamics courses. These courses are an integral part of the Space Systems Engineering curriculum and provide control algorithm familiarization opportunities, opportunities not afforded in the regular Computer Science curriculum.

2. Evaluate Single-Point Precision Performance

The control algorithm's execution time may be further reduced if the IPP's multiplier does not require double precision. In general, embedded single-precision floating-point calculations are faster than their double-precision counterparts[54,55]. Control algorithm calculations that can be shifted into single-point precision, without adversely affecting precision, will result in faster algorithm execution.

a. Scope of Work

The control algorithm's single-point precision can be evaluated using MATLAB™. MATLAB™ contains a "single" function that forces a number or computational solution to single-point precision. Using the single function, the control algorithm or its individual functions may be evaluated for precision affects. If performed on the host system, the computation time is not expected to change. MATLAB™, and the Pentium® IV's FPU, performs all calculations in double precision and reports the results in the requested format.

b. Development Hardware and Software

A personal computer, MATLAB™, and the control algorithm are required to perform this analysis. The control algorithm is comprised of the spacecraft model and DIDO. This evaluation scrutinizes precision, not execution speed. Therefore, the host computer is not required to perform this study. It is recognized that other mathematics-based software program exist that could perform the single-double precision evaluation. However, the control algorithm's functions are written in, and executable within, the MATLAB™ development environment. The development time and effort required to translate, or reproduce, the control algorithm in different mathematical software is projected to be excessive.

---

54 Nallahtech. Double-Precision Floating-Point Core. Retrieved 18 Mar. 2005, from < http://www.nallatech.com/mediaLibrary/images/english/3269.pdf>

55 Nallahtech. Single-Precision Floating-Point Core. Retrieved 18 Mar. 2005, from < http://www.nallatech.com/?node_id=1.2.2&id=20&searchTerm=single%20point%20floating>

c. Task Assignment

The numerical analysis is best performed by academia. A student with a math or numerical analysis background is desired. Specifically, a student with a computational mathematics background is highly desirable. The candidate should also posses programming and MATLAB™ experience.

B. PHASE TWO: HARDWARE

Phase 2 is comprised of two work elements: establishing the embedded development board and cross compiling the algorithm for embedded operation.

1. Establishing the Embedded Development Board

Establishing the embedded development board's operation requires a defined scope of work, embedded development hardware, and a student or contractor.

a. Scope of Work

This work element establishes the embedded development board to an operable state. The real-time operating system is loaded into the board ROM, the onboard peripherals made operable, and the board support package utilities tested. These tasks may appear to be a trivial work element. However, this capability is being established in a lab that does not currently posses embedded systems development experience and capability. Establishing the development board in a fully operable state and building experience with its operation will help minimize erroneous troubleshooting once the algorithm is migrated onto the development board.

After the development board is operating, simple programs will be cross complied and executed to verify proper board operation. The experience gained cross compiling and debugging will be recorded. Often, the manufacturer's support documentation is written poorly, missing key steps that are assumed or simply omitted out of error. Cross-compiling, debugging, and running simple programs provides an opportunity to develop comprehensive "in-house" procedures for migrating programs onto the development board. In this section, cross-compiling includes the development processes required to download an executable program onto the development board.

After the cross-compiling and downloading of simple programs is perfected, the developers will cross-compile control algorithm modules into executable code. This effort will identify, early on, control algorithm code that does not compile to the embedded hardware. Algorithms written in high-level programming languages, such as C, can utilize functions that may not have a cross-compiled equivalent for the embedded hardware. In such a case, the offending function must be rewritten using hardware-supported functions. Modules, which fail to cross-compile, will be studied and the code modified to support the embedded hardware.

b. Development Hardware

The double-precision floating-point unit (DPFPU) requirement narrowed the list of potential development boards significantly. The research and recommendations are based upon locating current commercially available products. The DPFPU requirement presented a significant challenge when trying to locate an FPGA-based board. An important note, phase two does not require an FPGA; only phase three. A development board hosting an FPGA would reduce program cost and redundant development effort since the same development board would be reused. Phase two migration can be satisfied using a hard-core microcontroller containing a DPFPU. If the hard-core microcontroller approach were selected for phase two, an FPGA-based development board must be purchased for phase three. This approach will increase the program cost, but would decouple the development effort. IPP designers could design and develop in parallel with phase two. With this information in mind, a hard-core and a soft-core (FPGA-based) board are recommended and their benefits discussed.

c. Hardcore CPU and FPU Board

The AMCC PowerPC™ 440EP Evaluation Kit is very capable and meets phase two hardware requirements. The PowerPC™ 440EP is a hard-core microcontroller. The PowerPC™ 440EP microcontroller operates in excess of 500 MHz, over two times better than the FPGA-based development board. While the development board is not space-rated, other PowerPC™ microcontroller variants are on orbit. The common PowerPC™ pedigree and architecture will help identify non-compliable functionality, early. This identification will help transitioning the control algorithm to a

space-rated processing board containing a PowerPC™. The advantages and disadvantages of the AMCC development board are summarized below.

Figure 15. AMCC PowerPC 440EP Evaluation Board[56]

Pros:
Double precision floating-point unit
256-Mbyte RAM
32-Mbyte Flash ROM
Ethernet I/O connector
USB I/O connector
Linux Kernel OS and File system included on CD-ROM
Firmware Bootstrap – in flash memory
Kit cost: approximately $2,590.80[57]

Cons:
Lacks onboard FPGA
Included Linux OS is not a real-time OS
Not space rated

[56] Provided courtesy of AMCC.

[57] Rodreguiez, Thomas. "EV-440EP-WIN-01 Price Quote." E-mail to Ron Moon. 02 Dec. 2005.

d. FPGA-based Development Board

The Xilinx PowerPC™ & MicroBlaze™ Development Kit Virtex-4 FX-12 Edition is impressive. This FPGA-based development board is very capable and has the potential to meet phase two hardware requirements after significant integration effort. The Xilinx ML403 board supports soft-core microcontrollers: MicroBlaze™ and PowerPC405. The Xilinx PowerPC & MicroBlaze™ Development Kit Virtex-4 FX-12 Edition product sheet is provided in Appendix B.

Figure 16. Xilinx Virtex-4 ML403 Development Board[58]

Pros:
FPGA-based development board
64-Mbyte DDR SDRAM
1-Mbyte ZBT SRAM
512-Mbyte Compact Flash EPROM
Ethernet I/O connector
USB I/O connector
16x2 LCD Display
GNU Tools and Debugger
MicroBlaze™ IP Core
Kit cost: approximately $895

[58] Provided courtesy of Xilinx, Inc.

Cons:
Lacks Double-precision FPU
Lacks RTOS

e. Recommendation

Unfortunately, neither of the discussed development boards fulfills the needs for both phase two and three. The AMCC evaluation board meets the requirements for phase two but lacks an FPGA for phase three. Therefore, the AMCC board will not support phase three's hardware accelerator. Xilinx's Virtex-4 ML403 development board has the potential to meet phase two and three's requirements but will require significant design and integration efforts. Commercial developers are not currently selling soft-core IP microcontrollers with an integrated DPFPU.

The AMCC PowerPC 440EP development board would provide an excellent migration platform. The development board's 256-Mbyte of RAM, 32-Mbyte of ROM, 440EP PowerPC microcontroller and DPFPU provide the necessary hardware processing capability to host the compiled control algorithm. However, the development board does not contain an FPGA. A second, FPGA-based development board would be required to support phase three's hardware accelerator design. The product data sheet for the AMCC 440EP PowerPC Evaluation Board is provided in Appendix B.

The Xilinx ML403 supports two soft-core microcontrollers: Xilinx's MicroBlaze™ and PowerPC™ 405. The MicroBlaze™ microcontroller recently received the addition of a single-precision FPU. The PowerPC™ 405 microcontroller does not contain a floating-point unit. Both microcontrollers would require a DPFPU. The DPFPU would need to be purchased through a third-party company or designed. Once obtained, the DPFPU will require integration with the microcontroller via a high-speed data and control bus. An existing compiler will need to be modified, or one created, to support the new microcontroller and DPFPU combination. Lastly, the combination will require integration with a RTOS prior to supporting the development board and compiled control algorithm. The DPFPU design and integration process outlined is not a trivial task.

The embedded processing and FPGA sectors are progressing rapidly. The MicroBlaze™ microcontroller's FPU debuted May 16, 2005. Borrowing loosely from

Moore's law, one could predict the release of a DPFPU IP core for the MicroBlaze™ around November 2006. However, Xilinx is not working on, nor planning to create, a double-precision floating-point unit for the MicroBlaze™ IP core[59]. Since phase one migration is not completed, phase two migration efforts could wait and anticipate the release of a soft-core microcontroller with an integrated DPFPU from a third-party vendor. In the meantime, phase two development can progress using the AMCC Evaluation Board. A final option would be to purchase the AMCC board for phase two and the Xilinx board for phase three development. Purchasing both development boards enables immediate and concurrent phase two and three development.

f. Task Assignment

This task assignment carries two recommendations based on the board selection. If the FPGA-based board is selected, the design and integration of a DPFPU with either soft-core microcontroller should be performed by industry. While students could perform this effort, a professional company is better equipped and staffed to provide an established, functioning, and well-documented development board in a timely manner. If academia retains the development, a student from the Engineering curriculum, computing track, would be best suited to perform this work.

If the AMCC hard-core microcontroller development board is selected, a thesis student or long-term research assistant, vice a dedicated hired contractor, is the best person to establish the embedded development board's operation. The development board would be operated over a period of twelve to twenty-four months. The financial cost to maintain a dedicated contractor over this period would be excessive. A student in the Space Systems Engineering – Electrical Engineering track would be best suited to perform this work.

[59] Gazdik, Nate. "MicroBlaze™ Floating Point Unit." E-mail to Ron Moon. 05 Dec. 2005.

2. Cross-compile Program

a. Scope of Work

This work element compiles the control algorithm for operation on the embedded development board. The steps required create an embedded algorithm were displayed in Figure 6 and discussed in section IV. The translated C-code function files are to be cross-compiled, linked, relocated, and downloaded for operation on the embedded microcontroller. For the remaining discussion, the term "cross-compile" will be used to depict the entire compile to download process. The modular build process discussed in phase one's scope of work, migration to stand-alone program, should be mimicked. The main file should be compiled, first, and verified. Then, individual functions should be cross-compiled and merged with the main file. The cross-compiled functions should contain input and output capability to verify the functions' proper operation against the corresponding MATLAB™ control algorithm function. Individual functions are added until the entire control algorithm is operating on the embedded development board. The process outlined in this paragraph is tedious, but quickly identifies problematic functions.

b. Development Hardware and Software

Often the compiler, linker, debugger, and locator are purchased with the development board, usually part of the Board Support Package (BSP). However, if the microcontroller and RTOS are known, work may be able to progress prior to actually purchasing the development board. GCC[60] is an open source development tool that contains a cross-compiler, linker, debugger, and locator. GCC is often found as part of the utility programs for Linux and UNIX based computers. The new MAC operating system "OS X" is Linux-based and capable of hosting GCC. Furthermore, the most promising microcontroller for hosting the control algorithm is a PowerPC™. The Apple iMac's CPU is a PowerPC™. An iMac is available for use in the Guidance and Control Lab. However, the iMac is not currently running OS X. The OS X operating system would need to be purchased and installed. Compiling to a host system's processor that closely matches the microcontroller is advantageous. The PowerPC core series share the

[60] GCC. GCC Homepage. Retrieved 27 Nov. 2005, from <http://gcc.gnu.org>

same instruction sets. While not the exact embedded PowerPC microcontroller, compiling the control algorithm code on a PowerPC™ CPU will advance the understanding of how the control algorithm will behave, vice compiling to an x86 CPU. This is applicable for phase one and two development.

The x86 based host computer in the Guidance and Control Lab has two rack mount hard drives, each with a different operating system. One hard drive contains Windows XP Professional, SP2. The second hard drive contains Red Hat Linux Work Station 4. Red Hat contains GCC as a native compiler. However, the compiler is configured to support an x86 processor. The Red Hat GCC compiler may be able to be configured to perform as a cross-compiler. This avenue was not researched further due to time constraints.

c. Task Assignment

The cross-compile work element would, most likely, progress at a faster rate if a professional embedded programmer performed the work. This recommendation is not based on student inability; rather, the recommendation is based on professional proficiency and providing the final product in an expeditious manner. If the work is to be performed in the academic environment, a student experienced in embedded programming should perform the task. Embedded programming falls within the boundaries of Electrical Engineering and Computer Science. However, most Computer Science curriculums focus on object oriented programming and forego the topics critical to embedded programming. Therefore, the potential student's proficiency with embedded programming should drive the selection process, not the engineering discipline.

C. PHASE THREE: ACCELERATOR

Phase 3 contains three work elements: Design the IPP, modify the compiler, and integrate the IPP into the development board design.

1. Design and Test IPP

The developer will design and test one of the three potential IPP development paths presented in section four.

a. Scope of Work

IPP development will be performed within an integrated development environment. IPP design exploration, performed in conjunction with this research, utilized Xilinx's ISE 7.1i programmable logic design environment. ISE 7.1i was used in this research due to availability and the development environment's ability to meet the research goals. Other programmable logic development environments exist, but were not explored. The recommended FPGA-based development board is a Xilinx FPGA and board. Therefore, the Xilinx development environment, ISE, was used to in an effort to reduce potential compatibility issues.

ISE 7.1i is relatively easy to use due to the Project Navigator graphical user interface. The IPP FPGA design may be constructed using a HDL programming language such as VHDL or Verilog, or constructed using a schematic method. The schematic method involves selecting basic building blocks from a library. The blocks are then wired together by the designer to create the functional design. If the migration effort follows the Nallatech design option, the schematic design method will be followed. The Nallatech modules are provided as NGC files on a CD-ROM and cost $9,200[61]. The IP core data sheet is provided in Appendix C. If the IPP is designed using the OpenCores modules, IPP development will also use the schematic design method. Should the migration effort decide to follow the original design path, the design should be written in a HDL such as VHDL, not the schematic design method. However, individual modules can be written in one of the HDL formats, then converted into a schematic library symbol. Once converted into a symbol, the module can be wired within the schematic design environment. It is strongly recommended that the design effort utilize a HDL whenever possible. The Xilinx compiler optimizes the HDL files for optimal FPGA resource utilization. The schematic design method is not as effective at performing this optimization. Additionally, it is recommended that a single HDL is used, VHDL or

[61] Houlihan, Paul. Phone conversation. 18 Mar. 2005.

Verilog. An overall design can use modules written in different languages. However, Xilinx's website discussion board contains numerous posts by people trying to solve a development problem due to modules using different HDL languages.

As the design progresses within ISE, the IPP is tested within the ISE or externally using third-party modeling tools. The internal ISE testing module provides limited testing. However, ISE can identify improper results, estimate propagation delays, and identify the amount of FPGA resources required by the design. If the internal testing capabilities are not sufficient, third-party testing software may be used. The school currently uses ModelSim®[62], a powerful HDL simulator. The simulation testing capabilities provided by ISE and ModelSim® helps reduce the frustration of troubleshooting design errors on the development board. Many design errors are located prior to downloading the design to the FPGA. Once the IPP design is complete and tested using simulation tools, the design is downloaded into the FPGA for hardware testing. Test vectors can be stored in the development board's memory. The IPP test will include calling the test vectors from the development board's memory and performing the inner-product calculation. The testing metrics are calculation time and solution precision. A benefit of the design and simulation software, ISE and ModelSim®, is that this software can reside on the host computer. The purchase of additional computer hardware is not required.

b. Development Hardware and Software

The FPGA-based board selection was discussed earlier; therefore, the discussion will not be duplicated other than reiterate that the cost of ASIC fabrication for research designs is beyond current research funding levels. Once a complete deign scheme is formulated, microcontroller, DPFPU, and IPP, obtaining an ASIC fabrication cost estimate would be prudent for cost comparison purposes.

The recommended development board for phase three is an FPGA-based development board. Currently, industry does not offer a single development board hosting a microcontroller, DPFPU, and FPGA. As development phases one and two

[62] ModelSim®. Products List. Retrieved 11 Dec. 2005, from < http://www.model.com/products/60/default.asp>

progress, industry may release a development board containing these desired capabilities on a single board. Currently, the most promising board is the Xilinx Virtex-4 ML403 Development Board.

The developer will require an integrated development environment, such as ISE 7.1i. A copy of Xilinx ISE will need to be purchased. A limited-use version of ISE 7.1i is available for download. The developer will need to verify if the limited version will meet the project's design needs. A limited-use version of ModelSim®, ModelSim® XE-III Starter, may be downloaded from Xilinx's website. The ModelSim® company provides Xilinx users a one-year license after registering via e-mail. If the developer determines that the limited version does not meet the project's design needs, a site license copy of ModelSim® PE for VHDL costs $5538.00[63]. The ModelSim® product is much more capable than the simulation software included within Xilinx's ISE and will help the individual assigned to the task to identify design problems prior to embedded operations.

c. Task Assignment

The IPP design and implementation work would, most likely, progress at a faster rate if a professional FPGA core designer performed the work. Similar to the programming work, this recommendation is not based on student inability; rather, the recommendation is based on professional proficiency and providing the final product in an expeditious manner. However, building the proposed Nallatech-based IPP as thesis work is within the capability of a Naval Postgraduate School Electrical Engineering. If the work is to be performed in the academic environment, an Electrical Engineering student proficient with HDL programming should perform the task. A student with experience using a FPGA integrated development environment is desired. However, this experience can be provided through online and on-site vendor classes.

2. Modify Microcontroller/IPP Compiler

The addition of the IPP changes the control algorithm's use of the microcontroller, which includes the DPFPU. In phase two, the microcontroller's DPFPU

[63] Reynolds, Dennis. Phone conversation. 13 Dec. 2005.

performed the inner-product calculations. With the introduction of the IPP, the control algorithm can realize a performance increase by using the IPP. In order to achieve this performance improvement, the microcontroller's compiler must be updated to reflect the existence, and potential use, of the IPP.

a. Scope of Work

A developer will modify an existing compiler to include IPP functionality. The project's selected microcontroller will be supported by one, possibly more, software compilers. A compiler supporting the selected microcontroller and DPFPU will be modified to reflect the presence and capability of the IPP. The new compiler will direct the use of the IPP each time the control algorithm requests an inner-product, vice the DPFPU. After compiler modification, the compiler will be introduced into the programming-integrated development environment. This integration will allow the compiling of the control algorithm modules using the new, modified, compiler. The deliverable from this work element is a modified compiler and a compiled control algorithm. The brief scope of work explanation may make this task appear trivial. However, compiler generation, or modification, is not a simple task. The work is tedious and requires detailed knowledge of the hardware components involved. The tools used to perform compiler work are common to the tools used in phase one.

b. Development Hardware and Software

This should not require the purchase of new hardware or software. The developer should be able to utilize the same host system and programming-integrated development environment used in phase one to modify the compiler. The developer will require access to the development board and working IPP.

c. Task Assignment

The migration to stand-alone control algorithm should be accomplished by industry. A professional programmer is the best choice to modify the existing compiler. Specifically, the company that wrote the existing compiler for the microcontroller should be contracted to perform this work. Generally, professional programmers are more proficient at writing compilers. A detailed understanding of the microcontroller is required to modify the existing compiler. The project's schedule would benefit from

using a programmer that is currently familiar with the microcontroller. While students are intelligent and work diligently, on average, they do not posses the same level of programming technical expertise. The disparity in proficiency would, most likely, lead to a longer phase one development time, should a student be used to perform the work.

If the task is not assigned to industry, the compiler modification should be performed by a Computer Science thesis student. The ideal candidate would be a Space Systems Engineering student following the computer science track, vice a student within the normal computer science curriculum. The Space Systems Engineering students understand the algorithm's application due to their controls, optimization, and dynamics courses. These courses are an integral part of the Space Systems Engineering curriculum and provide control algorithm familiarization opportunities, opportunities not afforded in the regular Computer Science curriculum. The student should form a partnership, or trusted working relationship, with the company that wrote the original compiler. If it can be arranged, it may be beneficial for the student to work at the company's facility for an extended period of time. This on-site work will foster a working relationship with the compiler developers.

3. Integrate IPP

After the IPP and the compiler modification are complete, the design must be integrated into the development board.

a. Scope of Work

The IPP module design will be instantiated into the development board's FPGA. The new control algorithm, compiled with the modified compiler, will be downloaded into the development board ROM. The board will be tested for proper operation. The control signal generation time and precision will be compared to the MATLAB™ baseline. It is difficult to provide additional details outlining this work task due to the number of development path variables. These variables include the IPP development path, the integrated development environment software, and the modeling software selected.

Integrating the IPP into the development board will be challenging and may require more than a single person. It is highly recommended that the person performing the integration work overlap with the IPP design and Compiler modification work. The overlap should be, at a minimum, three months. The integrator will require the assistance of the IPP designer and compiler writer. It would be ideal if the two, or more, personnel performing the IPP design and compiler modification also perform the integration work. However, as explained at the beginning of this section, the task elements were broken down into work elements capable of fitting within a thesis student's schedule. Since integration work often uncovers previously undiscovered errors, the integration work may require more than one person.

b. Development Hardware and Software

This task will utilize hardware and software obtained in the earlier phases of work. New hardware and software should not be required unless a new development board is obtained. Should a new development board be introduced, the similar utilities, tools, and support software identified in the earlier sections will need to be obtained.

c. Task Assignment

Selecting an individual to perform the IPP integration will be difficult. Systems integration is a challenging interdisciplinary field. The integration phase would most likely progress at a faster rate if industry performed the work. Industry will likely avoid accepting an integration project that has been piecemealed between academia and industry, unless they have been a significant contributor in the development. The recommendation to employ industry is not based on student inability; rather, the recommendation is based on professional proficiency and the ability of industry to draw from various engineering disciplines to provide the final product in an expeditious manner.

While industry would be the best candidate, the most likely result will be that the systems integration will be preformed by a thesis student. The IPP integration and compiled code migration fall within embedded systems development. The student system integration recommendation closely follows the embedded programming recommendation provided in phase two. The IPP integration work falls within the

boundaries of Electrical Engineering and Computer Science. Therefore, it is recommended that the IPP integration be carried out by two students, an electrical engineering and computer science student. Additionally, the integration work should overlap with the IPP design and Compiler modification work. The overlap will allow the system integrators to familiarize themselves with the existing work, prior to the IPP designer and Compiler programmer's departure.

D. FURTHER RESEARCH

1. State Update Rate - Sensor Saturation

Classical control theories are constrained in their ability to provide control signal updates by the rate at which their sensors can provide state determination updates. The work performed in association with this migration plan suggests that the real-time optimal-control algorithm may be able to decouple the sensor update rate for some control applications. If the system's control model is accurate and error-generating disturbances are small in relation to the control authority, the control algorithm may be able to achieve acceptable performance even though the sensor update rate is slower than the control command rate.

Each real-time optimal-control algorithm solution provides the complete control sequence required to go from the existing state to the ordered final state. In spacecraft attitude-control applications, each control algorithm solution provides the entire control signal stream required by the torque devices to maneuver the spacecraft from the current attitude orientation to the commanded orientation. The spacecraft can execute the control sequence and achieve a final orientation without further updates. The final orientation will contain sensor and disturbance errors. However, the important distinction is that the classical control system produces one control signal, and holds that control signal. Classical control systems cannot provide a control update without first obtaining a spacecraft state condition, one control command per state determination.

The control algorithm may be able to provide acceptable performance in applications where classical control systems fail. The proposed control algorithm

implementation would utilize two control storage register sets and a switch. The first register set stores the first control solution. The spacecraft begins the maneuver, executing the control signal stream in the first register set. As soon as the sensors can provide a state update, the control algorithm generates a new control solution. In the meantime, the spacecraft continues to execute the maneuver using the control signals in the first register set. Once the new control solution is complete, the solution is stored into the second register set. The switch changes and places the second register set into service. The spacecraft torque devices then receive the control commands stored in the second register set. The process continues until the spacecraft reaches the commanded orientation.

2. C to VHDL Compilers/Function Generators

Both academia and the private sector are pursuing the development of C to VHDL compilers. In the academia world, the University of California campuses of Irvine and San Diego have collaborated and developed a C to VHDL compiler, SPARK[64], with private sector research support. Nallatech will soon release their C to VHDL function generator, Dime-C. Dime-C will be integrated into their DIMEtalk-3 development suite[65]. Both products are in an infancy stage. When developed further, these products will provide users the high-level abstraction capability of C and allow rapid migration of behavioral algorithms into hardware. Currently, the conversion of behavioral C algorithms into hardware requires a design team to replicate the algorithm's behavior. Some industry professionals predict that efficient C to VHDL tools will remain beyond reach. While not mature at this time, this technology is worth watching.

---

64 UC San Diego. Center for Embedded Computer Systems. Retrieved 01 Nov. 2005, from < http://mesl.ucsd.edu/spark/>

65 Nallatech. FPGA Computing Application Development Environment–DIMEtalk3. Retrieved 14 Dec. 2005, from <http://www.nallatech.com/?node_id=1.2.2&id=19>

APPENDIX A: PHASE ONE MATERIALS

1. Master Files
    a. NPSAT Model
    b. DIDO_2003f
    c. SNOPT
2. Modified Files
    a. NPSAT Model
    b. DIDO_2003f
    c. SNOPT
3. Translated (Modified) Files
    a. NPSAT Model
    b. DIDO_2003f
    c. SNOPT
4. MATLAB™ Compiler 3 User's Guide
5. LCC Programming Development Environment
    a. Programming Manual
    b. Install Files
6. GCC Compiler Manual
7. Control Algorithm Code Estimation
8. TextPad 4
9. Spark (XP Pro Version)

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: PHASE TWO MATERIALS

1. MontaVista RTOS Product Sheet
2. Nucleus RTOS Product Sheet
3. PowerPC Microcontroller Product Selector
4. AMCC PowerPC 440EP Evaluation Kit
5. Xilinx Virtex-4 Product Family
6. Xilinx Virtex-4 FX Embedded Development Kit

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: PHASE THREE MATERIALS

1. Xilinx Virtex4 FX FPGA Device Combination Table
2. Nallatech Double-Precision Floating-point Core Product Sheet
3. Pentium® 4 Inner-product Test M-file
4. IPP Performance Estimator Excel File
5. IEEE VHDL Reference Manual 2002
6. VHDL Cookbook
7. IEEE RTL Synthesis Manual 2004

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D: LOBATTO.M (MATLAB™)

Note: due to the file's length, a partial reproduction is provided. Appendix A's CD-ROM contains the entire translated file.

```
function [x,w] = lobatto(n,a,b)
%   [X, W] = LOBATTO(N) or [X, W] = LOBATTO(N,ALPHA,BETA):
%
%     Computes abscissa and weights for the n-point Gauss-Jacobi-
Lobatto
%     quadrature formula using the method of Gene H. Golub, Some
modified
%     matrix eigenvalue problems, SIAM Rev. 15 (1973) 318-334.  Another
early
%     algorithm for this is by David Galant, An implementation of
Christoffel's
%     formula in the theory of orthogonal polynomials, Math. Comp. 25
(1971)
%     111-113.  All such algorithms should be "reviewed", in light of
recent
%     improvements in tqr and Cholesky LR algorithms.  But, this
algorithm
%     "ain't bad".

%     Copyright (c) 23 August 1997 by Bill Gragg.  All rights reserved.
%       Edited by I. Michael Ross, 17 April 2001

%     lobatto calls subfunctions:  mxtj, mxt and tqr.


%     begin lobatto

         if nargin < 2
        a = 0;   b = 0;
         end

         m = 2^(a + b + 1)*beta(a+1,b+1);   us = a == b;

         n = n - 1;           [a b] = mxtj(n,a,b);   T = mxt(a,b);
     I = eye(n);          e = zeros(n,1);        e(n) = 1;
     c = (T + I)\e;       c = c(n);              d = (T - I)\e;
     d = d(n);            e = c - d;             c = (c + d)/e;
     d = sqrt(2/e);       a(n+1) = c;            b(n) = d;
     [x u] = tqr(a,b);    u = u';                w = m*u.^2;

%        "Purify" formulas in the ultraspherical case.

         if us
            x = (x - flipud(x))/2;   w = (w + flipud(w))/2;
         end
```

```
%       Consider sorting x for future reference
%     end lobatto
%
%       BEGIN SUBFUNCTION MXTJ

function [a,b] = mxtj(n,alpha,beta)

% [a b] = mxtj(n,alpha,beta), [a b] = mxtj(n,alpha), [a b] = mxtj(n),
%     T = mxtj(n,alpha,beta),     T = mxtj(n,alpha) or   T = mxtj(n):
%
% mxtj(n,alpha,beta):  T = mxt(a,b) is the Jacobi matrix whose
characteristic
% polynomial p is (a nonzero scalar multiple of) the nth JACOBI
polynomial.
% The eigenvalues of T are the abscissas of the nth order Gauss-
Christoffel
% quadrature formula for the weight function ((1 - t)^alpha)((1 +
t)^beta) on
% the interval - 1 < t < 1.  The Gauss-Christoffel weights are m(0)
times the
% squares of the first elements of the normalized eigenvectors of T,
where
% m(0) = b(0)^2 = B(alpha + 1,beta + 1)2^(alpha + beta - 1) is the
total mass.
% B is the beta function.  The weight function is positive and
integrable if
% alpha + 1 > 0 and beta + 1 > 0.
%
% mxtj(n,alpha) takes beta = alpha.  p is the nth ULTRASPHERICAL
polynomial,
% with weight function (1 - t^2)^alpha on the interval - 1 < t < 1.
Special
% cases are the CHEBYSHEV polynomial of the FIRST KIND, with alpha = -
1/2,
% and of the SECOND KIND, with alpha = 1/2.
%
% mxtj(n) takes alpha = beta = 0.  p is the nth LEGENDRE polynomial,
with
% weight function w(t) = 1 on the interval - 1 < t < 1.  The quadrature
% formula here is originally due to Gauss.  Christoffel generalized
Gauss'
% formula to a wide class of weight functions.  Because of this the
Gauss-
% Christoffel weights are usually called Christoffel numbers.

% Copyright (c) 2 February 1991 by Bill Gragg.  All rights reserved.

% mxtj calls mxt.

% begin mxtj
     if nargin < 2  alpha = 0;  end;   if nargin < 3  beta = alpha;
end
     a = alpha;   b = beta;   c = a + b;   d = b - a;
     s(1) = d/(c + 2);   t(1) = (a + 1)*(b + 1)/(c + 2)^2/(c + 3);
     if n > 2
        d = c*d;
```

```
        n = (2:n)';   m = 2*n;   mm = m - 1;   mp = m + 1;
        s(n) = d./(c + m)./(c + (m - 2));
        t(n) = n.*(a + n).*(b + n).*(c + n)./(c + mm)./((c + m).^2)./(c
+ mp);
     end
     a = s(:);   b = 2*sqrt(t(:));
     if nargout < 2  a = mxt(a,b);  end
% end mxtj
%
%       BEGIN SUBFUNCTION MXT

      function T = mxt(a,b,c)

%     T = mxt(a,b,c) or T = mxt(a,b):
%
%     T = mxt(a,b,c) is the TRIDIAGONAL MATRIX with diagonal elements
a(1:n),
%     subdiagonal elements b(1:n-1) and superdiagonal elements c(1:n-
1).
%
%     T = mxt(a,b) is the HERMITIAN tridiagonal matrix with diagonal
elements
%     real(a(1:n)) and subdiagonal elements b(1:n-1).

%     Copyright (c) 1 December 1990 by Bill Gragg.  All rights
reserved.
%     Revised 21 November 1992.


%     mxt calls no extrinsic functions.


%     begin mxt

         if nargin < 3
            a = real(a);   c = b';
         end

         n = length(a);   b = b(1:n-1);   c = c(1:n-1);   z = zeros(n-
1,1);

         if n < 500

            B = diag(b);   B = [z' 0; B z];   C = diag(c);   C = [z C;
0 z'];
            T = diag(a);   T = T + B + C;

         else

            T = zeros(n);

            for k = 1:n-1
               T(k,k) = a(k);   T(k+1,k) = b(k);   T(k,k+1) = c(k);
            end
```

```
            T(n,n) = a(n);

         end
%     end mxt
%
%       BEGIN SUBFUNCTION TQR (note: TQR calls SGN)

      function [lam,U] = tqr(a,b,U)

%     [lam u] = tqr(a,b) or [lam U] = tqr(a,b,U):
%
%     [lam u] = tqr(a,b):
%
%     The column lam contains the eigenvalues of the Hermitian
tridiagonal
%     matrix T = mxt(a,b) computed by one version of the (real
symmetric) tqr
%     algorithm with Wilkinson's shift.  The column u contains the
first
%     elements of the eigenvectors of T normalized to be nonnegative
and such
%     that the eigenvectors are unit vectors.  In practice this is an
O(n^2)
%     process.  If u is omitted only the eigenvalues are computed.  The
%     computed eigenvalues are real and are sorted to be nonincreasing.
%
%     [lam U] = tqr(a,b,U):
%
%     This replaces the input U by UV with V a matrix of orthonormal
eigen-
%     vectors of T.  If the input U is I the output U is V.  If the
input U is
%     unitary with AU = UT then the output U is unitary with AU = UD
and D =
%     diag(lam).
%
%     If the input U is e(1)' the output U is u'.  If the input U is
%     [e(1)'; e(n)'] the output U is [u'; v'] with v the column of last
%     elements of the normalized eigenvectors.  If the subdiagonal
elements of
%     T are all nonzero then the elements of v alternate in sign, at
least
%     mathematically.

%     Copyright (c) 2 February 1991 by Bill Gragg.  All rights
reserved.
%     Revised 15 July 1994.
%     begin tqr
```

File truncated here due to length!

APPENDIX E: LOBATTO.C (TRANSLATED)

Note: due to the file's length, a partial reproduction is provided. Appendix A's CD-ROM contains the entire translated file.

```
/*
 * MATLAB Compiler: 3.0
 * Date: Thu Jul 21 18:35:55 2005
 * Arguments: "-B" "macro_default" "-O" "all" "-O" "fold_scalar_mxarrays:on"
 * "-O" "fold_non_scalar_mxarrays:on" "-O" "optimize_integer_for_loops:on" "-
O"
 * "array_indexing:on" "-O" "optimize_conditionals:on" "-t" "-A" "debugline:on"
 * "-L" "c" "-d"
 *
"C:\Ron_Moons\DIDO_Convert_to_C\DIDO_Working_Folder\DIDOmodules\Compiled
_Fold
 * ers\lobatto_compiled" "lobatto"
 */
#include "lobatto.h"
#include "beta.h"
#include "flipud.h"
#include "libmatlbm.h"
static mxArray * _mxarray0_;
static mxArray * _mxarray1_;
static mxArray * _mxarray2_;
static mxArray * _mxarray3_;
```

```
static mxArray * _mxarray4_;
static mxArray * _mxarray5_;
static mxArray * _mxarray6_;

static mxChar _array8_[7] = { 'c', 'o', 'm', 'p', 'a', 'c', 't' };
static mxArray * _mxarray7_;
static mxArray * _mxarray9_;
static mxArray * _mxarray10_;

static mxChar _array12_[45] = { 't', 'q', 'r', ' ', 'i', 't', 'e', 'r', 'a',
                                't', 'i', 'o', 'n', ' ', 'd', 'i', 'd', ' ',
                                'n', 'o', 't', ' ', 't', 'e', 'r', 'm', 'i',
                                'n', 'a', 't', 'e', ' ', 'i', 'n', ' ', '1',
                                '0', 'n', ' ', 's', 't', 'e', 'p', 's', '!' };
static mxArray * _mxarray11_;

void InitializeModule_lobatto(void) {
    _mxarray0_ = mclInitializeDouble(0.0);
    _mxarray1_ = mclInitializeDouble(2.0);
    _mxarray2_ = mclInitializeDouble(1.0);
    _mxarray3_ = mclInitializeDouble(3.0);
    _mxarray4_ = mclInitializeDouble(500.0);
    _mxarray5_ = mclInitializeDoubleVector(0, 0, (double *)NULL);
    _mxarray6_ = mclInitializeDouble(1024.0);
```

```
    _mxarray7_ = mclInitializeString(7, _array8_);
    _mxarray9_ = mclInitializeDouble(-1.0);
    _mxarray10_ = mclInitializeDouble(10.0);
    _mxarray11_ = mclInitializeString(45, _array12_);
}

void TerminateModule_lobatto(void) {
    mxDestroyArray(_mxarray11_);
    mxDestroyArray(_mxarray10_);
    mxDestroyArray(_mxarray9_);
    mxDestroyArray(_mxarray7_);
    mxDestroyArray(_mxarray6_);
    mxDestroyArray(_mxarray5_);
    mxDestroyArray(_mxarray4_);
    mxDestroyArray(_mxarray3_);
    mxDestroyArray(_mxarray2_);
    mxDestroyArray(_mxarray1_);
    mxDestroyArray(_mxarray0_);
}
static mxArray * mlfNLobatto_mxtj(int nargout,
                                  mxArray * * b,
                                  mxArray * n,
                                  mxArray * alpha,
                                  mxArray * beta);
```

```
static void mlxLobatto_mxtj(int nlhs,
                            mxArray * plhs[],
                            int nrhs,
                            mxArray * prhs[]);
static mxArray * mlfLobatto_mxt(mxArray * a, mxArray * b, mxArray * c);
static void mlxLobatto_mxt(int nlhs,
                           mxArray * plhs[],
                           int nrhs,
                           mxArray * prhs[]);
static mxArray * mlfNLobatto_tqr(int nargout,
                                 mxArray * * U,
                                 mxArray * a,
                                 mxArray * b,
                                 mxArray * U_in);
static void mlxLobatto_tqr(int nlhs,
                           mxArray * plhs[],
                           int nrhs,
                           mxArray * prhs[]);
static mxArray * mlfLobatto_sgn(mxArray * Z1, mxArray * Z2);
static void mlxLobatto_sgn(int nlhs,
                           mxArray * plhs[],
                           int nrhs,
                           mxArray * prhs[]);
```

File truncated here

LIST OF REFERENCES

ARM®. ARM® Technical Support FAQ. Retrieved 18 Nov. 2005, from <http://www.arm.com/support/vfp_support_code.html>

ARM®. ARM® VFP10 Coprocessor. Retrieved 18 Nov. 2005, from <http://www.arm.com/products/CPUs/VFP10.html>

Axiom Manufacturing. CML-5485 Development Board with BDM. Retrieved 21 Nov. 2005, from < http://www.axman.com/cgi-bin/products.pl?ProdName=CML-5485W;.State=Show>

Barr, Michael. (1999). *Programming Embedded Systems* (p. 9). Sebastopol, CA: O'Reilly & Associates, Inc.

Bartos, Frank J. (2005). Chip Wars: ASICs Versus FPGAs. *Control Engineering*. Retrieved 20 Nov. 2005, from < http://www.manufacturing.net/ctl/article/CA607224>

Brown, Stephen and Zvonko Vranesic. (2002) *Fundamentals of Digital Logic with Verilog Design*. New York: McGraw-Hill.

Bursky, Dave. (2005). We Must Hold The Line On Soaring ASIC Design Costs. *Electronic Design*. Retrieved 20 Nov. 2005, from <http://www.elecdesign.com/Articles/Index.cfm?AD=1&ArticleID=1955>.

Confluence. Confleunce Overview. Retrieved 26 Nov. 2005, from < http://www.confluent.org/wiki/doku.php>

Dunn, Paul. Nallatech Double-Precision FP Cores. E-mail to Ron Moon. 23 Nov. 2005.

EMS Technologies. ESP603e PowerPC Space Processor Card Data Sheet. Retrieved 30 Nov. 2005, from < http://www.emsstg.com/pdf/esp603.pdf>

Murray, Walter. Personal interview. 21 Jan. 2005.

Fleming, A. (2004). *Real-Time Optimal Slew Maneuver Design and Control*. Monterey, CA: Naval Postgraduate School.

Gazdik, Nate. "MicroBlaze™ Floating Point Unit." E-mail to Ron Moon. 05 Dec. 2005.

GCC. GCC Homepage. Retrieved 27 Nov. 2005, from <http://gcc.gnu.org>

Gill, Philip E., Murray, Walter, and Saunders, Michael A. (2005). *SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization*. Society for Industrial and Applied Mathematics Review (Vol. 47, No.1, pp. 99-131). Philadelphia: SIAM.

GNU Operating System. GNU Lesser General Public License. Retrieved 26 Nov. 2005, from < http://www.gnu.org/copyleft/lesser.html>

Goldberg, David. (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. *Sun Microsystems*. Retrieved 30 Nov. 2005, from <http://docs.sun.com/source/806-3568/ncg_goldberg.html>

Houlihan, Paul. Phone conversation. 18 Mar. 2005.

Hwang, Enoch O. (2006). *Digital Logic and Microprocessor Design with VHDL*. Canada: Thompson.

Jungkind, Dave. SEAKR Space Processor Cards. E-mail to Ron Moon. 06 Dec. 2005.

Kopp, R.E. (1962). George Leitman (Ed.) *"Pontryagin Maximum Principle", in Optimization Techniques*. New York: Academic Press, Inc.

Loomis, Herschel. Personal Interview. 23 Nov. 2005.

ModelSim®. Products List. Retrieved 11 Dec. 2005, from < http://www.model.com/products/60/default.asp>

Murecky, John. MontaVista Software, Inc. Phone conversation. 16 Sep. 2005.

Murray, Walter. Personal interview. 22 Apr. 2005.

Murray, Walter and I. M. Ross. Personal interview. 22 Apr. 05.

Nallatech. FPGA Computing Application Development Environment–DIMEtalk3. Retrieved 14 Dec. 2005, from <http://www.nallatech.com/?node_id=1.2.2&id=19>

Nallahtech. Double-Precision Floating-Point Core. Retrieved 18 Mar. 2005, from < http://www.nallatech.com/mediaLibrary/images/english/3269.pdf>

Nallahtech. Single-Precision Floating-Point Core. Retrieved 18 Mar. 2005, from <http://www.nallatech.com/?node_id=1.2.2&id=20&searchTerm=single%20point%20floating>

Noergaard, Tammy. (2005). *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Oxford: Elsevier, Inc.

Opencores Organization. CF Floating Point Multiplier. Retrieved 01 Aug. 2005, from< http://www.opencores.org/projects.cgi/web/cf_fp_mul/overview>

Opencores Organization. CVS Howto. Retrieved 26 Nov 2005, from < http://www.opencores.org/projects.cgi/web/opencores/cvs_howto>

Opencores Organization. HCSA Adder. Retrieved 01 Aug 2005, from
< http://www.opencores.org/projects.cgi/web/hsca_adder/overview>

Opencores Organization. Projects by category. Retrieved 01 Nov. 2005, from
< http://www.opencores.org/browse.cgi/by_category>

OpenCores Organization. Website. Retrieved 05 Mar. 2005, from <
http://www.opencores.org>

Quesenbury, Ann. MontaVista Software, Inc. Phone conversation. 16 Sep. 2005.
MontaVista Software, Inc. Linux Professional. Retrieved 16 Sep. 2005, from
<http://www.mvista.com/products/pro/features.html>

Reynolds, Dennis. Phone conversation. 13 Dec. 2005.
UC San Diego. Center for Embedded Computer Systems. Retrieved 01 Nov. 2005, from
< http://mesl.ucsd.edu/spark/>

Rodreguiez, Thomas. "EV-440EP-WIN-01 Price Quote." E-mail to Ron Moon. 02 Dec. 2005.

Ross, I. M. Personal interview. 21 Nov. 2005.

Ross, I.M., and Fahroo, F. (2002). *User's Manual for DIDO 2003: A MATLABTM™ Application Package for Dynamic Optimization*. Monterey, CA: Naval Postgraduate School.

Ross I. M. and Fahroo, F. (2003). *"Legendre Pseudospectral Approximations of Optimal Control Problems," Lecture Notes in Control and Information Sciences* (Vol. 295). New York: Springer-Verlag.

Sekhavat, P., Fleming A. and Ross, I. M. (2005, July). *Time-Optimal Nonlinear Feedback Control for NPSAT1 Sapcecraft*. Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Monterey, CA.

The MathWorks. MATLAB™ Compiler Release Notes Page. Retrieved 15 Nov. 2005, from <http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/rn/compiler4_rn_fcs3.html>

The MathWorks. MATLAB™ Compiler Online Guide. Retrieved 15 Nov. 2005, from <http//:www.mathworks.com/access/helpdesk_r13/help/toolbox/compiler/compiler.html>

The MathWorks. MATLAB™ Product Page. Retrieved 10 Nov. 2005, from <www.mathworks.com/access/helpdesk/help/toolbox/simulink/slref/embeddedMATLABTMfunction.html>

The MathWorks. MATLABTM Online Programming Documentation. Retrieved 17 Nov. 2005, from <http://www.mathworks.com/access/helpdesk/help/techdoc/

MATLABTM_prog/ch11_st3.html>

The MathWorks. SIMULINK™ Product Page. Retrieved 10 Nov. 2005, from <www.mathworks.com/access/helpdesk/help/toolbox/simulink/sfg/f6-151.html>

Xilinx, Inc. Alliance Embedded Program Member List. Retrieved 29 Nov. 2005, < http://www.xilinx.com/ise/embedded/epartners/listing.htm>

Xilinx, Inc. Design Entry and Synthesis. Retrieved 04 Dec. 2005, from < http://toolbox.xilinx.com/docsan/xilinx7/books/data/docs/dev/dev0014_5.html>

Xilinx, Inc. MicroBlaze™ Floating-Point Unit. Retrieved 17 Nov. 2005, from <http://www.xilinx.com/ipcenter/processor_central/microblaze/microblaze_fpu.htm#features>

Xilinx, Inc. Xilinx PowerPC & MicroBlaze™ Development Kit. Retrieved 30 Nov. 2005, from <http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?sGlobalNavPick=PRODUCTS&sSecondaryNavPick=Intellectual+Property&category=&iLanguageID=1&key=DO-ML403-EDK-ISE>

Xilinx, Inc. Xilinx Virtex-4 ML-403 Embedded Platform. Retrieved 11 Nov. 2005, from <http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=HW-V4-ML403-USA&sGlobalNavPick=PRODUCTS&sSecondaryNavPick=BOARDS>

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. I. Michael Ross
   Naval Postgraduate School
   Monterey, California

4. Herschel H. Loomis
   Naval Postgraduate School
   Monterey, California

5. LCDR Ron Moon
   SPAWAR Space Field Activity
   Falls Church, Virginia