2004

DSP IMPLEMENTATION OF A DIGITAL NON-LINEAR INTERVAL CONTROL ALGORITHM FOR A QUASI-KEYHOLE PLASMA ARC WELDING PROCESS

Matthew Wayne Everett
University of Kentucky, rdrash371@aol.com

ABSTRACT OF THESIS

DSP IMPLEMENTATION OF A DIGITAL NON-LINEAR INTERVAL CONTROL ALGORITHM FOR A QUASI-KEYHOLE PLASMA ARC WELDING PROCESS

The Quasi-Keyhole plasma arc welding (PAW) process is a relatively simple concept, which provides a basis for controlling the weld quality of a subject work piece by cycling the arc current between a static base and variable peak level. Since the weld quality is directly related to the degree of penetration and amount of heat that is generated and maintained in the system, the Non-Linear Interval Control Algorithm provides a methodology for maintaining these parameters within acceptable limits by controlling the arc current based upon measured peak current times. The Texas Instrument's TMS320VC5416 DSK working in conjunction with Signalware's AED-109 Data Converter provides a hardware solution to implement this control algorithm. This study outlines this configuration process and demonstrates its validity.

KEYWORDS: TMS320VC5416 DSK, AED-109, Interval Control, Quasi-Keyhole, Plasma Arc Welding

Author: Matthew Wayne Everett

Date: 29 May 2004

DSP IMPLEMENTATION OF A DIGITAL NON-LINEAR INTERVAL CONTROL ALGORITHM FOR A QUASI-KEYHOLE PLASMA ARC WELDING PROCESS

By

Matthew Wayne Everett

Dr. YuMing Zhang
Director of Thesis

Dr. William T. Smith
Director of Graduate Studies

Date: 29 May 2004

RULES FOR THE USE OF THE THESES

Unpublished theses submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the thesis in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

THESIS

Matthew Wayne Everett

The Graduate School

University of Kentucky

2004

DSP IMPLEMENTATION OF A DIGITAL NON-LINEAR INTERVAL CONTROL ALGORITHM FOR A QUASI-KEYHOLE PLASMA ARC WELDING PROCESS

THESIS

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in Electrical Engineering in the
College of Engineering
at the University of Kentucky

By

Matthew Everett, PE, LSIT, MCP

Lexington, Kentucky

Director: Dr. YuMing Zhang, Department of Electrical and Computer Engineering

Lexington, Kentucky

2004

At this time, I wish to dedicate this thesis to my mother, Linda Glass. She has always been an example of principle, perseverance, and dedication. I can only hope that she is proud of me as I am of her. I love you Mom.

Acknowledgements

This work has been supported and partially funded by the National Science Foundation under Grant DMI-0114982 and the Center for Manufacturing Systems at the University of Kentucky.

At this time, I would like to thank Dr. Yuming Zhang for the personal, academic, and material support he has provided throughout my undergraduate and graduate work at the University of Kentucky in the Department of Electrical and Computer Engineering. As the Director of the Welding Research Laboratory and the Applied Sensing and Control Laboratory, Dr. Zhang has graciously allowed me to participate in this program, utilize his laboratory facilities, and provide technical guidance. Since first meeting Dr. Zhang in the summer of 2001, I have found him to not only be extremely intellectually gifted, but also a refreshing individual. I cannot remember a time when Dr. Zhang was unable to smile and provide an upbeat tone to our discussions, or in my observations, his discussions with others. The University of Kentucky is quite fortunate to have a professor such as Dr. Zhang on staff.

I would also like to thank Wei Lu for his technical guidance and patience. Wei is currently pursuing his PHD here at the University of Kentucky, which I am sure he will complete quite soon. Wei is a fine individual who enriches the academic community. If in the future our paths do not cross, I wish Wei and his family the best life has to offer. I will not forget his gratitude.

Table of Contents

List of Tables

List of Figures

List of Files

Matt614.pdf ……..… 11.4 Mb

Chapter One

Introduction

1.1 Objective

The objective of this project was to digitally implement a non-linear interval control algorithm for the Quasi-Keyhole plasma arc welding (PAW) process. Several means are available to accomplish this end such as: Personal Computer (PC) based general purpose processor (hereafter referred to as Micro-Processor); Digital Control Processors (DSP); or Micro-Controllers. All three of these approaches provide Analog-to-Digital Converters (ADC), which are sometimes simply called Analog-Inputs (AI), and Digital-to-Analog Converters (DAC), which are sometimes referred to Analog-Outputs (AO), capabilities in order to provide an interface between the digital and analog arenas. It is noteworthy to mention that technically a Digital-Input (DI) could have been utilized instead of an AI since the subject methodology simply references a voltage threshold level, which is akin to the digital realm.

1.2 Micro-Processor [1]

The Micro-Processor is traditionally defined as a general purpose computer or processor built on a single Integrated Chip (IC), which contains no RAM, ROM, or data converters. Its purpose is solely to receive digital data, process, and output digital information. The platform where the Micro-Processor resides determines what is processed, how data is stored, and I/O functions. The Micro-Processor based solution, such as the Intel Pentium 4 for example built on a platform such as a PC or MAC, is a more traditional approach for control implementations where general purpose, relatively high speed, and expandability are of concern. This type of implementation is relatively simple to implement since the associated Operating System handles most background configuration, control, and timing issues, while an abundance of on-board and peripheral hardware choices allow for easy expansion. The downsides of a Micro-Processor approach are excessive monetary cost, large size, high power demand, and large overhead. However, since the PC based solution has been previously demonstrated, the choice was immediately reduced to a DSP or Micro-Controller technology.

1.3 Micro-Controller [1] [2] [3]

A Micro-Controller attempts to combine the abilities of a PC with features such as Micro-Processor, memory, Input/Output (I/O), timers, and other peripherals into a single Integrated Chip (IC). In general, Micro-Controllers utilize a Von Nueman Architecture, which utilize a shared memory space and bus for both data and instructions. As a result, two fetches are required at least to execute an instruction. One fetch would be used to fetch the instruction, while the other would be used to execute.

A good example of a Micro-Controller would be a Motorola 68HC912DT128A. This Micro-Controller contains the following: 8192 bytes RAM; 2048 bytes EEPROM; 131072 bytes Flash; 8 Timer Channels; 8 MHz Bus; 5 Volt Supply; 8 ADC Channels, 10 bits each; 2 PWM Channels, 8 bits; 4 PWM Channels, 16 bits; and 67 I/O pins.

The Micro-Controller is the most popular and widely used low speed control device. It has a heavy use in the appliance and automotive industry. An interesting example of how much more the Micro-Controller is used compared to the Micro-Processor would be to examine the average household in the United States. With Micro-Controllers being in everything from the microwave to the can opener, the average household may have several dozen Micro-Controllers, while on average only every other household has a Micro-Processor, which is usually associated with the family personal computer. The advantages of the Micro-Controller are small size, low cost, and small power demand, while the disadvantages are associated with limited expandability, slow speed, and application mismatching.

1.4 Digital Signal Processor [2] [4]

The Digital Signal Processor (DSP) is similar to the Micro-Controller in that both are field programmable and combine multiple functions such as processing, memory, I/O, timing, and other peripherals into a single Integrated Chip (IC). However, the DSP and Micro-Controller are very different in the applications that they are used due primarily to differences in their architecture and peripherals. In general, the DSP utilizes a Modified Harvard Architecture, which allows data and instructions to each have their own independent separate memory space and bussing structure. As a result, pre-fetching the following instruction while the latter is being executed speeds up processing by a factor of two, except in the case of branching. Other additional features such as the inclusion of a single cycle hardware multiplier, tend to greatly increase the processing speed. With the lack of overhead associated with a general purpose Micro-Processor, the DSP tends to be the fastest of the three options when trying to implement control processing. Therefore, DSP's tend to be used in applications where real-time processing is necessary such as imaging or speech applications. The only real disadvantages of a DSP is cost, complexity, and power consumption when compared to a Micro-Controller.

Realistically, the Quasi-Keyhole plasma arc welding process could be controlled with a Micro-Controller, but other research avenues such as control methodologies involving weld pool imaging are currently being pursued at the University of Kentucky; therefore, it was decided that this control process would be a good test bench for implementing a DSP solution. As a result, the Spectrum Digital's TMS320VC5416 DSK stand-alone and evaluation module was chosen as a platform to implement the non-linear interval control algorithm for the Quasi-Keyhole plasma arc welding (PAW) process. The key features of this multi-layered printed circuit board DSK are as follows: TMS320VC5416 DSP operating at 16 to 160 MHz; On-board USB JTAG controller with plug and play drivers; 64K words of external on-board RAM; 256K words of external on-board Flash ROM; Three Expansion Connectors (Memory Interface, Peripheral Interface, and Host Port Interface); On-board IEEE 1149.1 JTAG Connection for Optional Emulation Debug; Burr Brown PCM 3002 Stereo Codec; +5 volt operation; and Texas Instrument's Code Composer Studio configuration software.

1.5 Daughter Card [5] [6]

The DSK contains on it board an Audio CODEC PCM3002 data converter whose analog-to-digital (AD) sampling rate may be user selected up to a maximum speed of 48 KHz. For this reason, the PCM3002 is referred to as an Audio data converter since the maximum range of

human hearing perception is usually on the order of 15 to 20 KHz. For this control implementation, this rate would be sufficient; however, it was desired to pursue faster sampling rates for future research applications as previously discussed. As a result, Signalware's AED-109 Multi-Channel 8 MHz maximum sampling rate Analog Expansion Daughter Card was chosen as a substitute data converter.

Chapter Two

Quasi-Keyhole Plasma Arc Welding Process

2.1 Arc Welding [7] [8] [9]

Arc welding utilizes an electrical arc as a heat source in order to heat, melt, and join metals. As a result, a very basic understanding of how heat is transferred to the work piece is necessary before developing a reasonable control algorithm. An arc is defined as an electric current flowing between two electrodes through an ionized column of gas. A negatively charged cathode and a positively charged anode create the intense heat of the welding arc. Since there must be an ionized path to conduct electricity across a gap, the mere switching on of a power supply over a cold electrode will not start the arc. The arc must be ignited or struck. The arc may be ignited be either applying an initial voltage high enough to cause a discharge or more commonly by touching the electrode to the anode and then withdrawing once the contact area becomes heated, which is called striking. The intense heat at the welding tip, which approaches 6500 °F, causes the Argon gas, which surrounds the Tungsten cathode tip, to become ionized, which effectively means that some of the outer electrons have been stripped away from the central nucleus forming an ionized gas. A plasma is defined as an ionized gas, which consists of a sea of ions and electrons that are a very good conductor of electricity. As a result, once the Argon gas surrounding the Tungsten tip has been heated sufficiently it ionizes forming a plasma, thus providing a conductive path between the cathode and anode to support the arc current.

The power dissipated in the arc does not occur entirely in the anode. Instead, the arc can be thought of as impedance to the flow of current, which can be subdivided into the anode, column, and cathode as shown below:

$$P_t = P_{cathode} + P_{column} + P_{anode} = I_{arc\,current}\left(V_{cathode} + V_{column} + V_{anode}\right) \qquad 2-1$$

While considering equation 2-1, it should be realized that a majority of the power loss occurs in the interfaces. In other words, the voltage drop in the column is typically small compared to that in the anode and cathode. Furthermore, the energy input into an arc welding process is proportional to the input power and inversely proportional to the travel speed of the torch. In addition, the heat transfer efficiency, f_1, to the work piece is less then one due to losses associated with heating the welding rod, conductive dissipation, and radiation, which results in the following equation:

$$H_{net} = f_1\left(P_{anode}/Speed\right) = f_1\left(V_{anode}I/Speed\right) \qquad 2-2$$

The choice of which electrode is the anode or cathode is another interesting topic. In this application, the welding tip is chosen as the cathode, since a non-consumable tungsten electrode was used. The reason for this choice is that the cathode emits electrons, which will accelerate in the Electric Field between the torch and work piece gaining additional kinetic energy. When the electron strikes the anode, additional energy will be released into the work piece. Thus helping to melt the work piece.

Finally, it should be realized that a large current is more important then a large voltage when welding. A terminal voltage level between 20 to 80 Volts is typical, but the amperage should be in the range of 30 to 300 Amperes. In certain instances, the current level may even approach several thousand amperes. Typically, power drops are available at 120, 240, 480, and 600 V AC. These voltages are stepped down through the use of a transformer and then rectified if a DC supply is desired. As a result, many control algorithms including the Quasi-Keyhole Non-Linear Interval Control Algorithm utilize the power supply's current as a control input. The power supply may be either alternating or direct depending on the welding process being utilized.

2.2 Plasma Arc Welding Process and Laboratory Experimental System [10] [11] [12] [13]

There are several types of arc welding processes such as: Shielded Metal Arc Welding (SMAW); Submerged Arc Welding (SAW); Gas Metal Arc Welding (GMAW); Flux Cored Arc Welding (FCAW); Gas Tungsten Arc Welding (GTAW); Double-Sided Arc Welding (DSAW); and Plasma Arc Welding (PAW) to just name a few. For the Quasi-KeyHole process, it was decided to utilize the PAW process due to its high energy density and very stable arc. A graphical description of a Quasi-Keyhole PAW system used in this study is shown below:

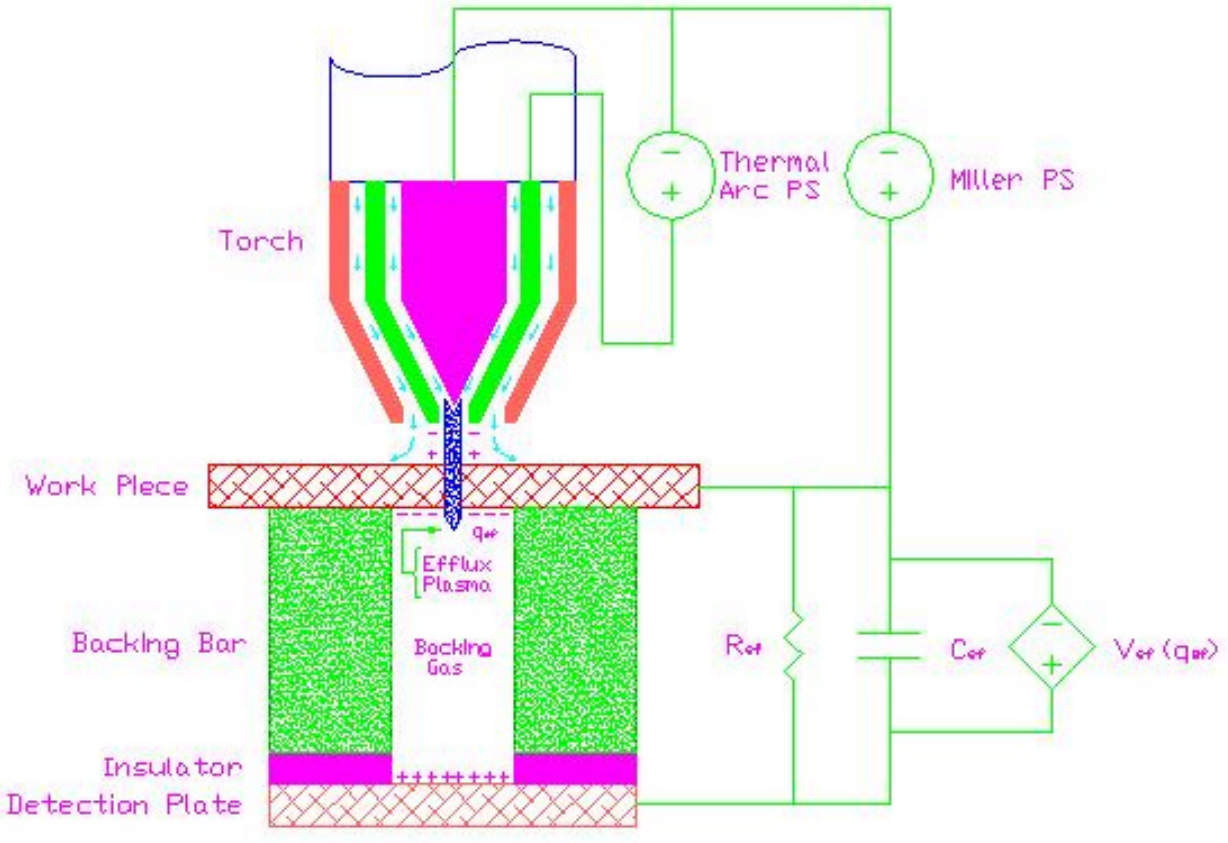


Figure 2-1 Laboratory Experimental System for Quasi-Keyhole PAW Process

The Plasma Arc welding process as implemented in Figure 2-1utilizes two different power supplies and a B & B Precision Machine custom made torch. The Thermal Arc Ultra Flex 350 Pulse power supply manufactured by Prestolite Power Corporation is utilized to initially strike the arc at an initial constant current level of 15 Amperes, which does not require that the torch be located over the work piece. Since the plasma Argon gas is being blown in the direction of the work piece, a conductive path or plasma is formed between the torch and the work piece once they are in the vicinity of each other. At this time, the Miller Electric Maxtron 450 CC/CV power supply can be turned on. The Miller power supply operates in a constant current mode with the current level being remotely controlled by the DSP. This control is realized through Port 17 of the power supply by varying a DC voltage level on pin G between zero and ten volts, while realizing a conversion ratio of one volt equals fifty-five amperes. At this point, the Miller power supply operates independently from the Thermal power supply; therefore, the Thermal power supply may be left on or turned off.

The work piece, detection plate, resistor, and capacitor together form the efflux plasma charge sensor (EPCS). The purpose of the EPCS is to determine when the keyhole has been established through the work piece. This is accomplished by realizing that a dependent voltage source is formed between the work piece and the detection plate. Before keyhole has been established, there is no charge on the backside of the work piece and the efflux or keyhole potential is zero, but when a charge begins to accumulate on the backside of the work piece the dependent voltage source begins to grow. As a result, a threshold level of 0.5 volts was chosen for this process to dictate when a keyhole had actually been established. A level below this was not recommended to avoid false indications due to random noise associated with the process. A resistor of 1 kΩ is utilized between these two plates to insure that the work piece and the detection plate are not electrically common, while limiting the current magnitude. The capacitor is utilized to eliminate random noise; however, in this study, the capacitor was eliminated from the circuit.

This process utilizes three Argon gas sources. The Argon source surrounding the tungsten tip is referred to as the plasma gas, which is used to establish the plasma jet. The Argon source, which surrounds the plasma jet, is referred to as the shielding gas, which exists to shield the weld pool from harmful oxidation effects. The Argon source, which feeds the platform, is referred to as the backing gas whose purpose is to shield the backside of the work piece from oxidation.

Finally, the last item, which is not shown in Figure 2-1, is the Dynetic Systems Servo Motor model 22134B. This servo is used to move the torch along the work piece at a fixed speed. Supplying a reference voltage to the actuator controls the speed of the servo. For this process, a reference voltage of .362 Volts was chosen which corresponds to a travel speed of about 2.5 mm/second.

Chapter Three

Digital Signal Processor

3.1 TMS320VC5416 DSP Developmental Starter Kit (DSK) [4] [14]

The TMS320C5416 DSP developer's starter kit (DSK) is a low-cost tabletop mounted printed circuit board (PCB) designed to allow the user to evaluate characteristics of the TMS320VC5416 DSP (C5416 DSP) to determine if the processor meets the requirements of the application as shown below in Figure 3-1:

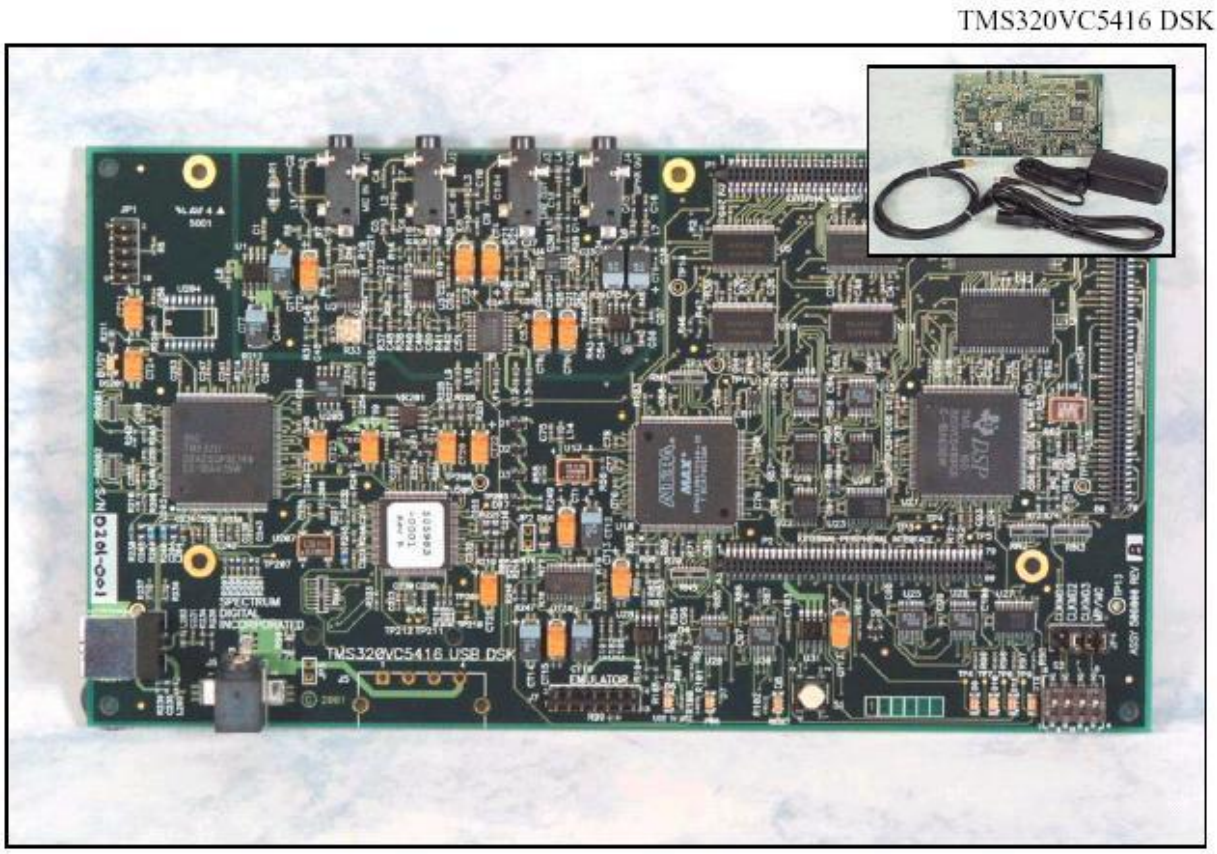


Figure 3-1 TMS320VC5416 DSK

The C5416 DSP, which operates between 16 to 160 MHz, is just one aspect of the development board. Key additional features of this module, as previously stated in the introduction include: On board USB JTAG controller with plug and play drivers; 64K words of external on-board RAM; 256K words of external on-board Flash ROM; Three Expansion Connectors (Memory Interface, Peripheral Interface, and Host Port Interface); On board IEEE 1149.1 JTAG Connection for Optional Emulation Debug; and Burr Brown PCM 3002 Stereo Codec. In addition to the development board, the kit includes: C5416 DSK Code Composer Studio™ v2.1 Integrated Development Environment (IDE); Quick Start Guide; Technical Reference; Customer Support Guide; USB Cable; Universal Power Supply; and a AC Power Cord. Although this

module is targeted towards audio signal processing applications, it can be easily modified for applications that require faster sampling rates through the use of the on board expansion connectors for the interfacing of Daughter Cards, such as the AED-109 data converter.

3.1.1 DSK Architecture [4] [15]

The DSK is divided into five major blocks of logic as follows: C5416 External on-board Memory (Flash ROM and SRAM); Codec (Data Converter); CPLD Registers (Interface Control); Expansion (Daughter Card and Host Communication); and JTAG (Emulation). The control of these logic blocks is divided between a complex program logic device (CPLD) and a field programmable gate array (FPGA). The Altera CPLD handles the interfacing concerned with the Flash ROM, SRAM, Codec Control, and Daughter Card expansion by maintaining registers, decoding memory addresses, and generating the appropriate control, while the FPGA performs a similar function for the emulation. Please refer to Appendix A for a complete listing of the DSK's central processing unit (CPU), status, CPLD, and peripheral memory-mapped registers.

3.1.1.1 Emulation [4]

Emulation is defined by Texas Instruments as providing a bridge between the Debugger and hardware. This implies a method of communication between the DSK and Code Composer Studio (CCS) located on the PC by utilizing the on board universal serial bus (USB) port, J201, or the joint test action group (JTAG) 14-pin header, J7. This block of logic, located in the bottom left corner, occupies roughly a quarter of the board area. This logic block is independent of the C5416 DSP operation, and is strictly intended as a communication path between the DSK and CCS for configuration, observation, testing, and troubleshooting. However, it is interesting to note that the DA250 DSP performs processing associated with this logic block instead of the C5416 DSP.

3.1.1.2 Hardware Expansion [16]

The expansion capability is divided between three ports, which are the external memory (P1), external peripheral (P2), and host port interface (P3). Ports P1 and P2 primary purpose is to allow hardware expansion of the DSK through the use of connecting boards referred to a Daughter Cards. The layout of a Daughter Card is specified such that a card may reside within a PC chassis when attached to a PCI motherboard, though the interface is not restricted to a PCI platform. The TMS320 Cross-Platform Daughter Card Specifications requires that Daughter Cards intended to attach to a DSP platform must conform to the following component side layout:

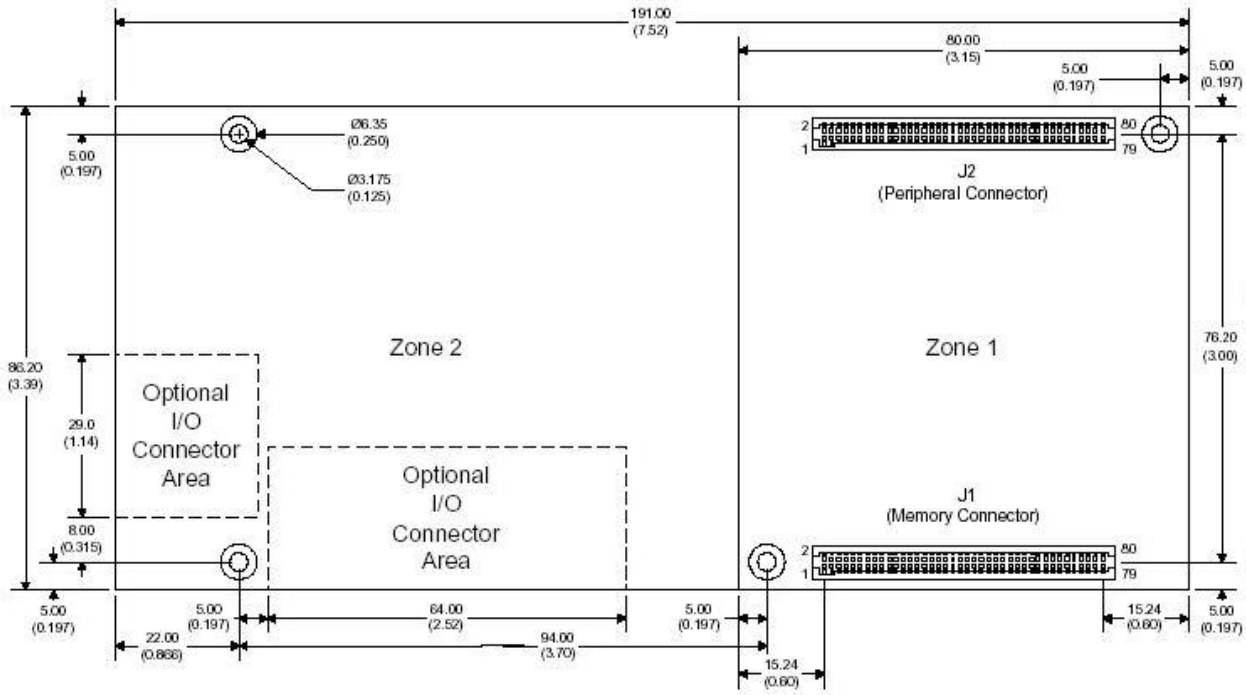


Figure 3-2 Daughter Card Layout

Communication between the DSK and Daughter Card occurs through the two 80-pin headers referenced in Figure 3-2, with one being primarily dedicated for external memory interfacing (P1) and the other for peripheral signals (P2). It should be noted that the designators on the Daughter Card utilize "J" designators while the DSK utilize "P" designators for the expansion ports. The same aforementioned TI standard requires that the component and connector heights be restricted to the following criteria:

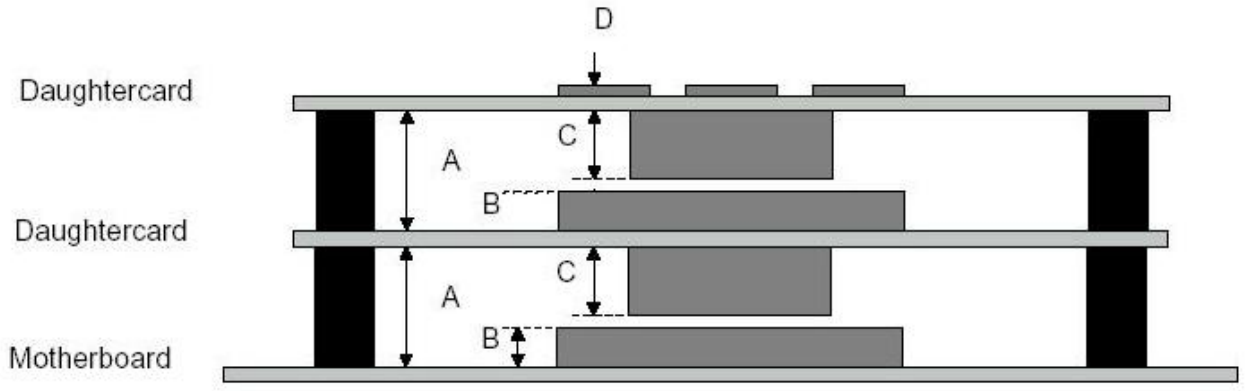


Figure 3-3 Stacked Daughter Card Illustration

Table 3-1Motherboard and Daughter Card Component Height

Label	Description	Dimension Zone 1	Dimension Zone 2
A	Mated Height	11.81 (.465)	11.81 (.465)
B	Maximum Motherboard Component Height	3.81 (.150)	4.78 (.180)
C	Maximum Daughter Board Component Height	6.73 (.265)	5.97 (.235)
D	Maximum "Bottom Side" Daughter Card Component Height	1.00 (.039)	1.00 (.039)

Daughter Cards are designed to be stackable as outlined in Figure 3-3 in case they do not require all of the available resources of the DSP motherboard. Applications of these Daughter Cards can be wide ranging with examples such as extended memory, RS232/RS422 serial ports, Ethernet, or data converters being very prevalent. The host port interface (HPI) provides a parallel port for which a host processor can access memory internal to the DSK for monitoring, configuration, and a variety of other reasons.

3.1.1.3 Memory [2] [4]

It should be remembered from the introduction that DSP's in general utilize a Modified Harvard Architecture, which allows data and instructions to each have their own independent separate memory space and bussing structure. As a result, the DSK actually takes this concept a step further by defining a shared memory space for Program, Data, and I/O addressing. The user defines how the memory space is to be allocated separately between Program, Data, and I/O by defining separate independent page layouts for each memory type, while realizing it is recommended to reserve Page 0 as Program space and to have different memory types on different pages. Furthermore, the DSK can be configured to operate in a Micro-Processor or Micro-Controller mode. This nomenclature is somewhat of a misnomer since the basic difference between the two modes is that the 16K word on-chip ROM is addressable in the Micro-Controller mode, but not in the Micro-Processor mode. Strictly speaking both modes are still in the DSP realm with the real intention of the Micro-Controller mode being to help facilitate boot loading upon initial power on.

3.1.1.3.1 Processor Mode Status Register [4] [17] [18] [19]

The aforementioned configuration and other features are established in the Processor Mode Status Register (PMST). The PMST register is shown below in Figure 3-4:

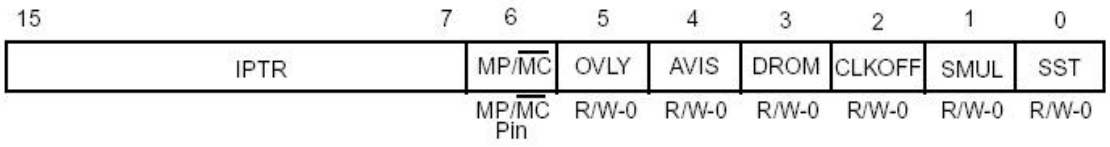


Figure 3-4 Processor Mode Status Register

The bit fields described in Figure 3-4 are outlined below in Table 3-2:

Table 3-2 PMST Bit Field Definition

Bit #	Bit Name	Reset Value	Function
15-7	IPTR	1FFh	Interrupt vector pointer – The 9-bit IPTR field points to the 128-word program page where the interrupt vectors reside. The interrupt vectors can be remapped to RAM for boot-loaded operations. At reset, these bits are all set to 1. The reset vector always resides at address FF80h in the program memory space. The RESET instruction does not affect this field.
6	MP/MC’	MP/MC’ Pin	DSP/Micro-Controller Mode – MP/MC’ enables/disables the on-chip ROM to be addressable in program memory space. - MP/MC’ = 0: The on-chip ROM is enabled and addressable. - MP/MC’ = 1: The on-chip ROM is not addressable. MP/MC’ is set to the value corresponding to the logic level on the MP/MC’ pin when samples are reset. This pin is not sampled again until the next reset. The RESET instruction does not affect this bit. This can also be set or cleared by software.
5	OVLY	0	RAM overlay. OVLY enables the on-chip dual address dual-access data RAM blocks to be mapped into program space. The values for the OVLY bit are: - OVLY = 0: The on-chip RAM is addressable in data space but not in program space. - OVLY = 1: The on-chip RAM is mapped into program space and data space. Data page 0 (address 0h to 7Fh), however, is not mapped into program space.
4	AVIS	0	Address visibility mode. AVIS enables/disables the internal program address to be visible at the address pins . - AVIS = 0: The external address line does not change with the internal program address. Control and data lines are not affected, and the address bus is driven with the last address on the bus. - AVIS = 1: The mode allows the internal program address to appear at the pins of the 5416 so that the internal program address can be traced. Also, it allows the interrupt vector to be decoded in

Table 3-2 PMST Bit Field Definition (Continued)

Bit #	Bit Name	Reset Value	Function
			conjunction with IACK when interrupt vectors reside in chip memory.
3	DROM	0	DROM – Enables on-chip DARAM4-7 to be mapped into data space. The DROM values are: - DROM = 0: The on-chip DARAM4-7 is not mapped into data space. - DROM = 1: The on-chip DARAM4-7 is mapped into data space.
2	CLKOFF	0	CLKOUT off. When the CLKOFF bit is a 1, the output of the CLKOUT is disabled and remains at a high level.
0	SST	N/A	Saturation on store. When SST = 1, saturation of the data from the accumulator is enabled before storing in memory. The saturation is performed after the shift operation.

3.1.1.3.2 Program Memory Map for Page 0 and Data Memory Map [4] [17] [18] [19]

Referencing the bit definitions in Table 3-2, the program memory map for Page 0 and data memory map are developed for the C5416 DSK as shown below in Figure 3-5:

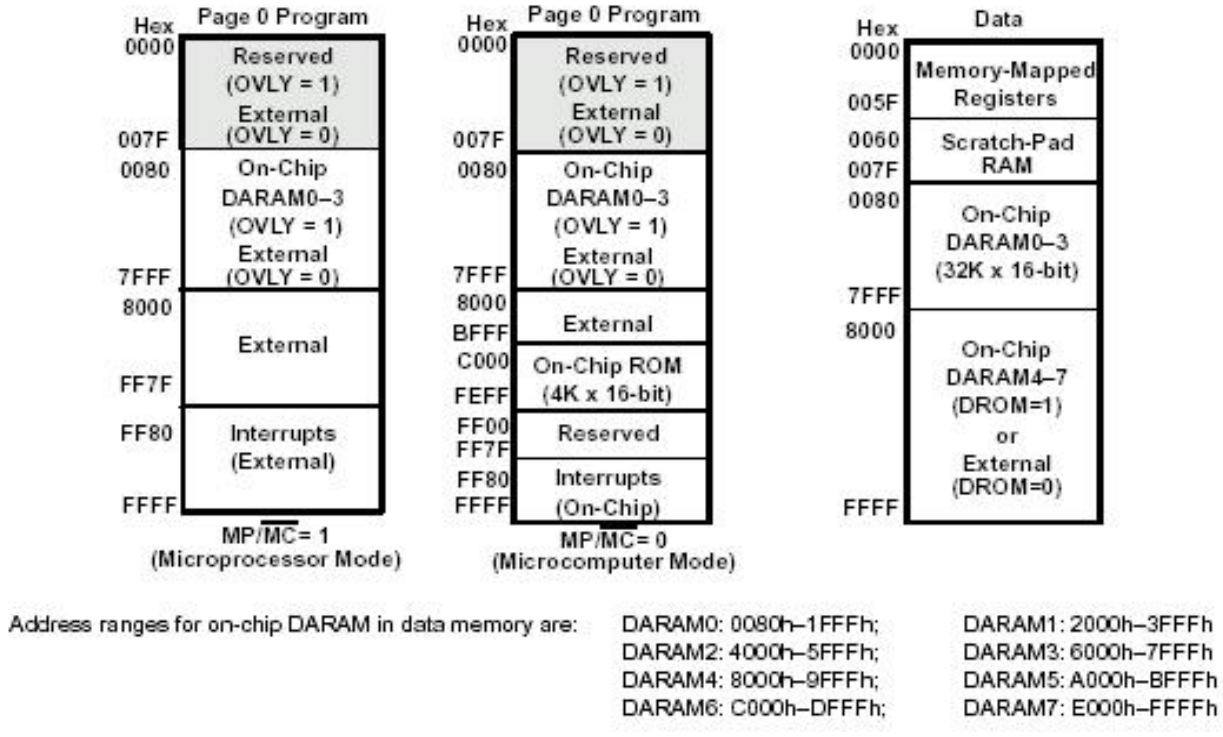


Figure 3-5 C5416 DSK Program Memory Map for Page 0 and Data Memory Map

Program memory is configured based upon the OVLY bit, MP/MC' bit, and Page number. Addresses 0000_{16} through $7FFF_{16}$ are internally mapped to the DSP chip provided the OVLY bit is set to 1, which is the preferred mode to be used by the DSK. These memory addresses are dual addressable, implying that two operations can occur on an address in a single clock cycle. Addresses 8000_{16} through $FF7F_{16}$ are always external if operating in the Micro-Processor mode, while addresses $FF80_{16}$ through $FFFF_{16}$ are always on-chip and reserved for external interrupts. If Program memory is set to operate in a Micro-Controller mode, addresses 8000_{16} through $FFFF_{16}$ are mapped externally except on Page 0. On Page 0, Program space is subdivided to allow for on-chip accessing of 16K word ROM, which results in 8000_{16} through $BFFF_{16}$ being external, $C000_{16}$ through $FEFF_{16}$ being on-chip ROM, $FF00_{16}$ through $FF7F_{16}$ being reserved, and $FF80_{16}$ through $FFFF_{16}$ being on-chip interrupts. Regardless of mode, addresses 0000_{16} through $007F_{16}$ are reserved and cannot be accessed by the programmer on Page 0.

Data memory is configured based upon the DROM bit, while the basic memory map is independent of page number except on Page 0 where it is recommended that the Data memory map not be applied due to reserved ranges and mixed memory types. Addresses 0000_{16} through $7FFF_{16}$ are always on-chip. This address range is subdivided into three divisions as follows: 0000_{16} through $005F_{16}$ are reserved for Memory-Mapped Registers; 0060_{16} through $007F_{16}$ are utilized as a Scratch-Pad; and 0080_{16} through $7FFF_{16}$ are available as general data space. Addresses 8000_{16} through $FFFF_{16}$ are located on-chip if DROM bit is asserted and off-chip otherwise. All on-chip Data memory is dual addressable, which implies once again that two operations can occur on an address in a single clock cycle.

3.1.1.3.3 Extended Program Memory Map [4] [17] [18] [19]

After the initial page, the same configuration bits are utilized, but the Program memory map is slightly different. If the OVLY bit is set high, memory ranges $0x0000_{16}$ through $0x7FFF_{16}$ will always utilize dual access on-chip memory. Otherwise, this memory range will be mapped off-chip. If the chip is operating in a Micro-Processor mode, memory addresses $0x8000_{16}$ through $FFFF_{16}$, will use dual access on Page 1, single access on Pages 2 and 3, and off-chip for Pages 4 through 127. Single access memory implies that only a single access per clock cycle can occur on an individual address. The MP/MC' bit only affects the configuration on Pages 1, 2, and 3 in the program extended memory map. If the MP/MC' bit is held low, memory addresses $0x8000_{16}$ through $0xFFFF_{16}$ will be mapped on-chip for Pages 1, 2, and 3. In all other cases, this memory range will map program space off-chip. Figure 3-6 shown below outlines how the extended program memory map is defined:

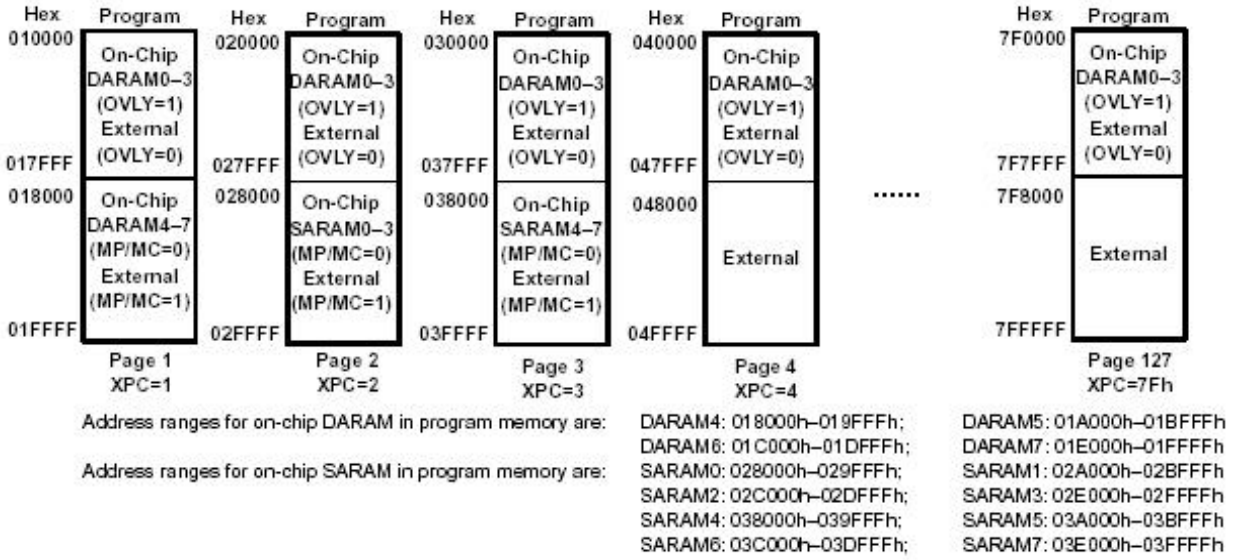


Figure 3-6 TMS320VC5416 DSK Extended Program Memory Map

3.1.1.3.4 I/O Memory Map [4] [17] [18] [19]

The C5416 DSP processor has no on-chip I/O accesses. The C5416 DSP uses a CPLD to interface to the external on-board Flash and SRAM, on-chip Data memory space, Codec control, and Daughter Card interface. As a result, the DSK uses I/O space to map the eight CPLD control registers starting at addresses 0000_{16} through 0007_{16}. This I/O map is shown below in Figure 3-7:

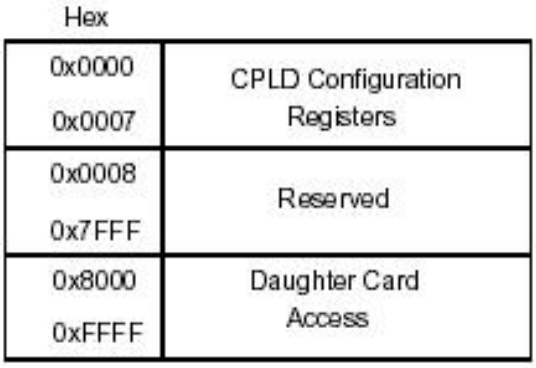


Figure 3-7 TMS320VC5416 DSK I/O Memory Map

This memory map does not vary based upon Page location except for Page 0 where it is not used.

3.1.1.3.5 Data Memory Page Map [4] [17] [18] [19]

Figure 3-5, Figure 3-6, and Figure 3-7 demonstrate how the DSK memory space is divided into separate pages. Each external data page has 32K words, where each word is 16 bits in length. The DSK utilizes a 20-bit word address for Data space, where the 15 least significant

bits are used to address the appropriate word on the subject page. The five most significant bits are defined within the DM_CNTL register, which is one of the eight control registers associated with the C5416 DSP CPLD. Based upon how the five most significant bits and DM_SEL bit are defined within the DM_CNTL register, the Data memory map is capable of addressing up to 1.024M words located on 32 different off-chip pages. However, of the 16 available C5416 DSP memory address lines, only A_0 A_{14} are used. A_{16} is set high only if an off-chip access is required. Finally, it should be realized that the 16 least significant bit is always asserted if external Data memory mapping is desired. The DM_CNTL register bit definitions are defined below in Table 3-3:

Table 3-3 Data Memory (DM_CNTL) Bit Definitions

Bit #	Bit Name	R/W	Description
7	DM_SEL	R/W	Data Memory Selection: - 0 = on-board memory - 1 = off-board Daughter Card
6	MEMTYPE_DS	R/W	- 0 = Flash Enabled - 1 = SRAM for Data Space Access
5	MEMTYPE_PS	R/W	- 0 = Flash Enabled - 1 = SRAM for Program Space Access
4	DM_PG4	R/W	Flash/SRAM/Daughter Card Memory Page, Bit 4, MSB
3	DM_PG3	R/W	Flash/SRAM/Daughter Card Memory Page, Bit 3
2	DM_PG2	R/W	Flash/SRAM/Daughter Card Memory Page, Bit 2
1	DM_PG1	R/W	Flash/SRAM/Daughter Card Memory Page, Bit 1
0	DM_PG0	R/W	Flash/SRAM/Daughter Card Memory Page, Bit 0, LSB

3.1.1.3.6 Program Memory Page Map [4] [17] [18] [19]

The DSK utilizes a 23-bit word address for external Program space, where the lower 15 bits are used to address the appropriate word on the subject page in a similar manner to Data space. The upper seven bits are defined in the Program Counter Extension (XPC) Register. The XPC register defines the page selection for Program Space. This register is memory-mapped into data space at address $001E_{16}$. At hardware reset, the XPC is initialized to 0. Since the XPC register is seven bits in length, Program space is capable of addressing 4096M words on 128 different off-chip pages. The DSK does not have on-board resources to fully utilize this space; however, the memory space does allow for memory expansion through external sources such as Daughter Cards or other Host processors, which can be easily interfaced through the Host Parallel Interface (HPI) Port . It also allows for utilization on other DSP platforms that have more robust resources.

3.1.1.3.7 Memory Resources [4] [17] [18] [19]

It should be noted that the DSK and the AED-109 Daughter Card do not possess the resources to take full advantage of this mapping. The C5416 DSP contains 64K words of DARAM, 64K words of SARAM, and 16K of Program ROM on-chip, while external on-board resources of 64K words SRAM and 256K words Flash ROM are available as well. The AED-109 Daughter Card does have the option to be delivered with a two 256K word Flash memory chips, but this option was not chosen for the card being utilized. Furthermore, the on-chip memory is divided into 8K word blocks. The memory division is on a per page basis and is divided between single and dual access. Figure 3-5 and Figure 3-6 define these blocks as being either DARAM0 through DARAM7 or SARAM0 through SARAM7 on each page. It should be noted that ROM is not volatile when power is removed from the DSK, which implies the programmer may read or write to these locations. As a result, a variety of on-chip, on-board, and Daughter Card RAM and ROM memory options are available to the programmer.

3.1.1.3.8 Wait State Generator [4] [17] [18] [19]

Memory access time tends to increase in the following order: on-chip Ram; on-chip ROM; external on-board RAM; external on-board Flash; external on-board CPLD; and external off-board Daughter Card Flash. In addition, when fetching or loading to a memory location the access time is not only dependent upon the hardware being utilized, but the speed at which the clock is running. As a result, the C5416 DSP has an on-chip wait state generator, for off-chip accesses, controlled by two registers referred to as the software wait state generator register (SWWSR) and the software wait state control register (SWCR). These two registers work in conjunction with each other to determine the appropriate number of wait states per external memory operation. This wait state generator can be extended up to 14 machine cycles. Devices that require more then 14 wait states can be interfaced using the hardware READY line. For further details, please refer to TI's TMS320VC5416 Fixed Point Digital Signal Processor Data Manual.

3.1.1.4 CPLD Registers [4]

Figure 3-8 outlines the bit definitions for the 8 registers contained within the CPLD as shown below, while realizing that the DM_CNTL register has been previously detailed:

I/O Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	USER_REG	USR_SW3 R	USR_SW2 R	USR_SW1 R	USR_SW0 R	USR_LED3 R/W	USR_LED2 R/W	USR_LED1 R/W	USR_LED0 R/W
1	DC_REG	DC_DET R	DC_IO_CTL R/W 0	DC_STAT1 R	DC_STAT0 R	DC_RST R 0(on reset)	0	DC_CNTL1 R/W 0(low)	DC_CNTL0 R/W 0(low)
2	CODEC_L	CODEC_L_CMD[7..0] R/W 0							
3	CODEC_H	CODEC_H_CMD[15..8] R/W 0							
4	VERSION	CPLD_VER[3.0] R				0	BOARD VERSION[2.0] R		
5	DM_CNTL	DM_SEL R/W 0(int)	MEMTYPE_DS R/W 0(flash)	MEMTYPE_PS R/W 0(flash)	DM_PG4 R/W 0(page 0)	DM_PG3 R/W 0(page 0)	DM_PG2 R/W 0(page 0)	DM_PG1 R/W 0(page 0)	DM_PG0 R/W 0(page 0)
6	MISC	CODEC_RDY R 0(Ready)	0	0	0	0	DC_WIDE R/W 0(16 bits)	DC32-ODD R/W 0(even)	BSP2SEL R/W 0(CODEC)
7	CODEC_CLK	0	0	0	0	DIV_SEL R/W	CLK_STOP R/W	CLK_DIV1 R/W	CLK_DIV0 R/W

Figure 3-8 CPLD Registers

3.1.1.4.1 USER_REG Register [4]

The USER_REG register controls the status of the 4 user LED's with the four least significant bits. The four most significant bits are used to read the state of the User Dip Switch. Table 3-4 shown below outlines this register:

Table 3-4 USER_REG Bit Definition

Bit #	Bit Name	R/W	Description
7	USER_SW3	R	User DIP Switch S2 – 4 (1 = Off, 0 = On)
6	USER_SW2	R	User DIP Switch S2 – 3 (1 = Off, 0 = On)
5	USER_SW1	R	User DIP Switch S2 – 2 (1 = Off, 0 = On)
4	USER_SW0	R	User DIP Switch S2 – 1 (1 = Off, 0 = On)
3	USER_LED3	R/W	User Defined LED D12 (0 = Off, 1 = On)
2	USER_LED2	R/W	User Defined LED D11 (0 = Off, 1 = On)
1	USER_LED1	R/W	User Defined LED D10 (0 = Off, 1 = On)
0	USER_LED0	R/W	User Defined LED D9 (0 = Off, 1 = On)

3.1.1.4.2 DC_REG Register [4]

The DC_REG register provides user control of the two Daughter Card control outputs and the Daughter Card reset Signal. This register also monitors the two Daughter Card status signals and the Daughter Card Detect Signal. These bit definitions are defined below in Table 3-5:

Table 3-5 DC_REG Bit Definition

Bit #	Bit Name	R/W	Description
7	DC_DET	R	Daughter Card Detection (0 = No Board, 1 = Daughter Card Detected)
6	DC_IO_CTL	R/W	0 = None, 1 = DC_RE - DC_RE is active on I/O Cycles
5	DC_STAT1	R	Daughter Card Status 1 (0 = low, 1 = high)
4	DC_STAT0	R	Daughter Card Status 2 (0 = low, 1 = high)
3	DC_RST	R/W	Daughter Card Reset (Reset Active Low)
2	0	R	Always zero
1	DC_CNTL1	R/W	Daughter Card Control 1 (0 = low, 1 = high)
0	DC_CNTL0	R/W	Daughter Card Control 0 (0 = low, 1 = high)

3.1.1.4.3 CODEC_L_CMD and CODEC_H_CMD Registers [4]

The CODEC_L_CMD and CODEC_H_CMD registers are used to send command codes to the on-board Burr-Brown PCM3002 Codec. These two individual 8 bit registers combined form the sixteen bit command word.. For more specific register information, please refer to the datasheet and TI's technical support for the appropriate cross-reference of embedded control signals. However, Ti's Data Converter Plug-In (DCP) is available within Code Composer Studio, which allows fast and easy software development for data converters attached to the DSP. This Plug-In will generate source code files and add them to your DSP development project. These files contain the lowest level of interface software required for the subject data converter. As a result, a general familiarity provided in the data sheets should allow the user to modify these Plug-In generated source files as required.

3.1.1.4.4 VERSION Register [4]

The VERSION register is used to signify the DSK board version and the VHDL firmware installed on the CPLD. The DSK board version occupies the lowest three significant bits, referred to as bits 2, 1, and 0, which are set during board assembly. The four most significant bits signify the VHDL firmware version, which are set during implementation of the CPLD. Bit 3 is not currently used and is always zero.

3.1.1.4.5 MISC Register [4]

The MISC register controls data memory access width, the DSP multi-channel buffered serial port 2 (MCBSP2) selection, and the Codec control shift register ready status. The most significant control bit is the DC_Wide, which signifies if the Daughter Card memory length is to be 16 or 32 bits. When the selection is 32 bits, two 16-bit words are combined with one residing in the upper 16 bits and the other in the lower 16 bits, which makes for a much faster transfer of data between the DSK and Daughter Card. Since the DSK memory addressing is based upon 16 bits, 32-bit addressing is accomplished by writing the 16 most significant bits first to the destination address plus one, and the writing the lower 16 bits to the destination address. When

reading from memory the 16 least significant address bits should be read first. The DC32_ODD bit is 1 if the destination or source address is odd and 0 if the destination or source address is even. The 16 least significant address bits always use the supplied address, while 16 most significant bits always adds 1 to the supplied address. Furthermore, for 32-bit addressing a specific sequence of events must be followed. When writing, the page address in the DM_CNTL register must be set first. Secondly, the DC_WIDE and DC32_ODD bits in the MISC register must be configured next. Third, write the 16 most significant bits first followed by the 16 least significant bits. The final step involves turning off the DC_WIDE bit in the MISC register. The procedure for reading is the same except that 16 least significant bits are read first followed by the 16 higher bits. The bit definitions are outlined below in Table 3-6:

Table 3-6 MISC Register Bit Definition

Bit #	Bit Name	R/W	Description
7	CODEC Ready	R	Codec Command Transfer Ready (0 = Ready 1 = Not Ready)
6	0	R	Always Zero
5	0	R	Always Zero
4	0	R	Always Zero
3	0	R	Always Zero
2	DC-WIDE	R/W	Daughter Card Memory Width Selection (0 = 16 bits, 1 = 32 bits)
1	DC32_ODD	R/W	32 Bit Daughter Card Address Access Mode (0 = even, 1 = odd)
0	BSP2SEL	R/W	MCBSP2 Select (0 = PCM3002 Data Channel, 1 = Daughter Card

3.1.1.4.6 CODEC_CLK Register [4]

The CODEC_CLK register effectively controls the sampling rate of the ADC aboard the PCM3002 as outlined below in Table 3-7:

Table 3-7 CODEC_CLK Register Bit Definition

Bit #	Bit Name	R/W	Description
7	0	R	Always Zero
6	0	R	Always Zero
5	0	R	Always Zero
4	0	R	Always Zero
3	DIV_SEL	R/W	- 1 = Codec Clock Divide Selected Rate is set by CLK_DIV Bits - 0 = Codec In Clock same as Codec Out Clock
2	CLK_STOP	R/W	CLK_STOP

Table 3-7 CODEC_CLK Register Bit Definition (Continued)

Bit #	Bit Name	R/W	Description
1	CLK_DIV1	R/W	- 00 = 24 KHz - 01 = 12 KHz - 10 = 8 KHz - 11 = 6 KHz
0	CLK_DIV2	R/W	

The default sampling rate is 48 KHz, but can be user selected to other values by pulling bit number 3 high and configuring the clock divisor bits appropriately. In order to set the sampling rate properly, a specific order must be followed. The CLK_STOP bit must be set first. The clock divisor bits are set next. Reset the CLK_STOP bit next. Set the DIV_SEL to one last.

3.2 TMS320VC5416 DSP (C5416 DSP) Functional Overview [2] [17] [19]

The TMS320VC5416 fixed-point, digital signal processor (DSP) is based on an advanced modified Harvard architecture that has one program bus and three data memory busses. This processor is the CPU for the C5416 DSK. A block diagram of the C5416 DSP is shown below in Figure 3-9:

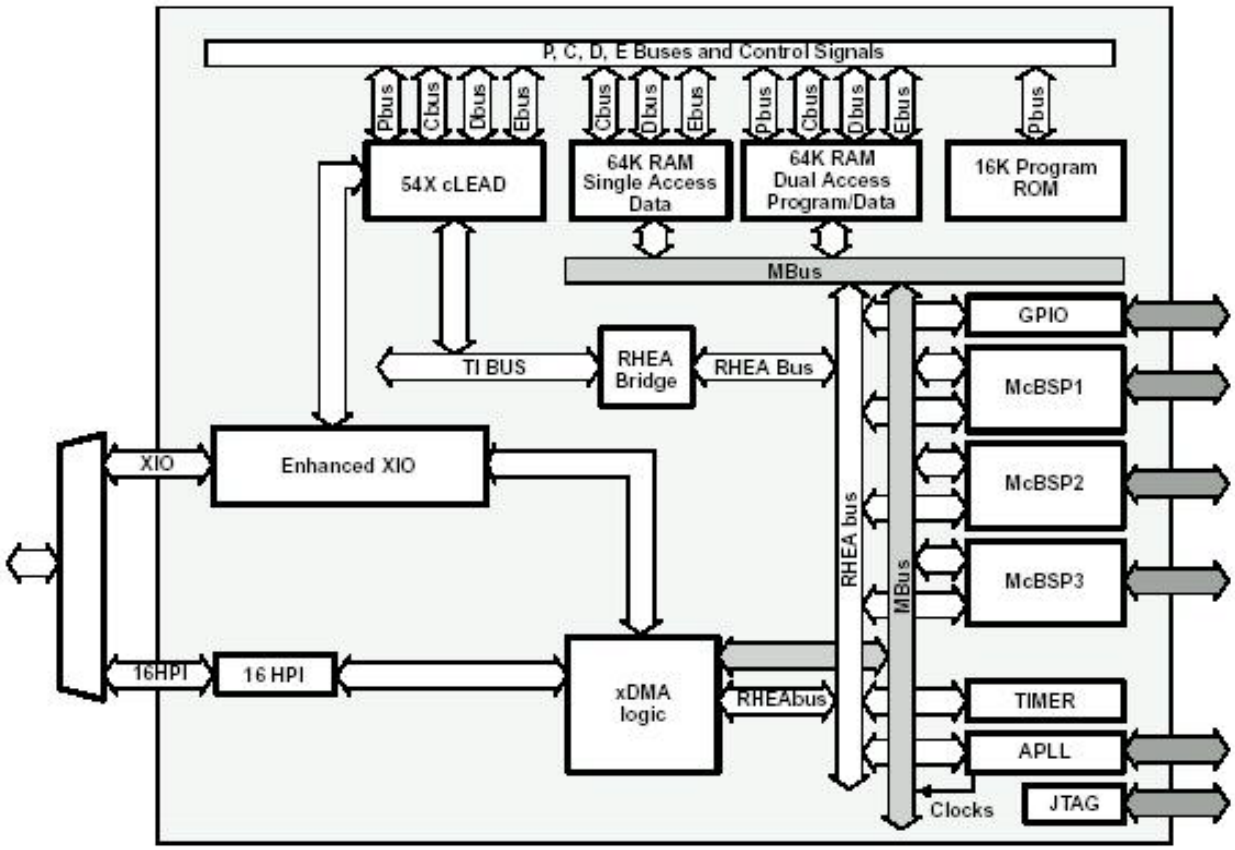


Figure 3-9 C5416 DSP Block Diagram

The C5416 DSP provides on-chip 16K words of Program ROM, 64K words of DARAM, and 64K words of SARAM as outlined previously in Section 3.1.1.3.7. In addition, external memory sources are available on the DSK, but faster processing speeds can be achieved if addressing remains on-chip to the greatest extent possible. Furthermore, the available memory space has been outlined in a detailed extended memory map, which is shared amongst the Data, I/O, and Program mappings as outlined in Section 3.1.1.3.

The C5416 DSP has several peripherals built on-chip listed as follows: software-programmable wait-state generator; programmable bank switching; host-port interface (HPI8/16); three multi-channel buffered serial ports (McBSPs); hardware time; clock generator with a multiple phase-locked loop (PLL); enhanced external parallel interface (XIO2); and a direct memory access (DMA) controller. The wait-state generator was previously introduced in Section 3.1.1.3.8.

3.2.1 5416 Processor [18]

This processor provides an arithmetic logic unit (ALU) with a high degree of parallelism, application-specific hardware logic, on-chip memory, and additional on-chip peripherals. The basis of the operational flexibility and speed of this DSP is a highly specialized instruction set. Please refer to Ti's TM320C54x™ DSP Functional Overview for a complete list of the 54x Instruction Set Opcodes.

Separate Program and data spaces allow simultaneous access to program instructions and data, which provides the high degree of parallelism referenced previously. Two reads and one write can be performed in single cycle. In addition, data can be transferred between program and data spaces. The 5416 processor accomplishes this configuration by integrating a 40-bit arithmetic logic unit (ALU), two 40-bit accumulators, a barrel shifter, a 17 x 17 bit multiplier/adder, and a compare/select/store (CSSU) unit into a single CPU. A functional block diagram is shown below in Figure 3-10:

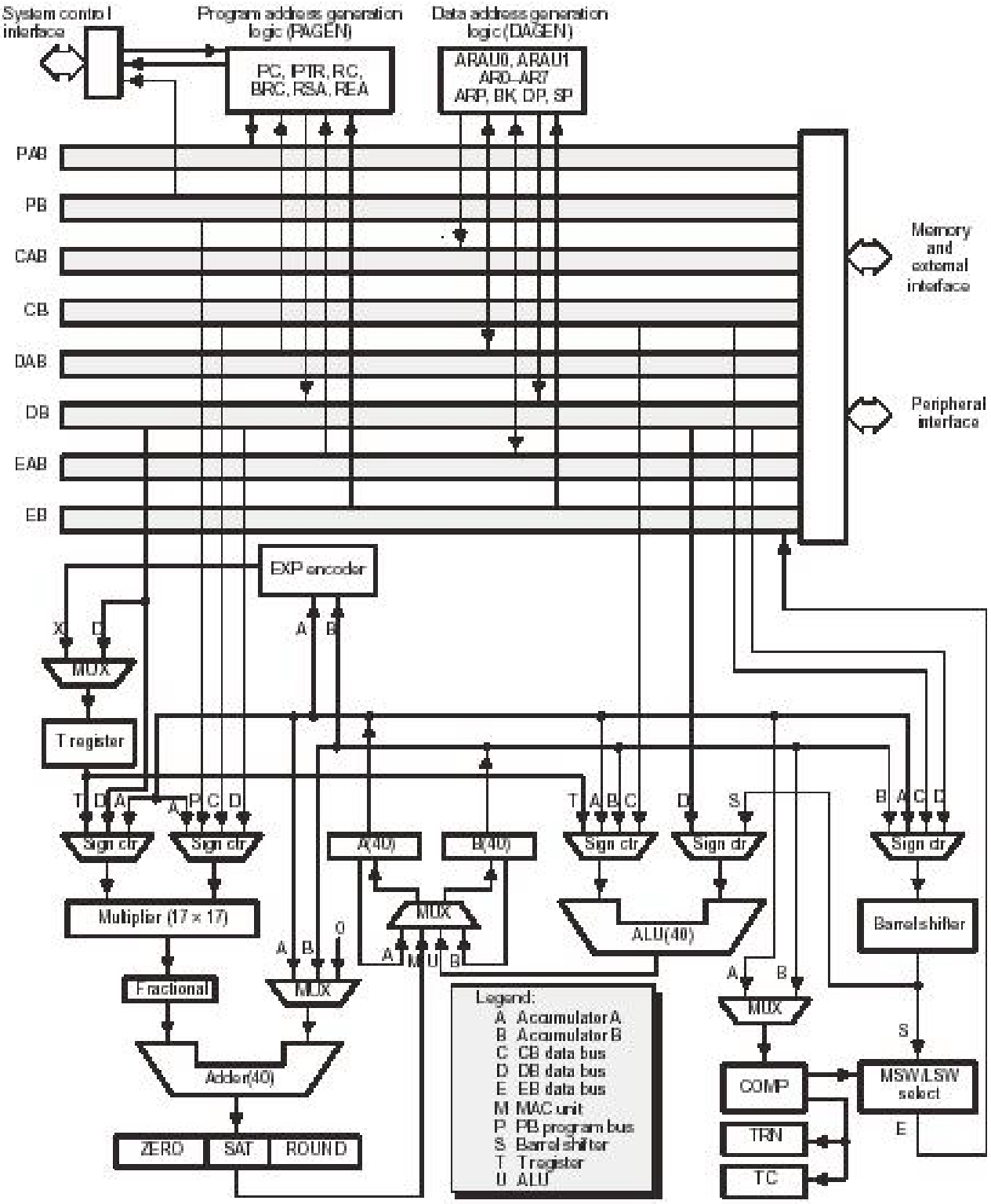


Figure 3-10 5416 Processor Block Diagram

Please refer to Ti's TM320C54x^{TM} DSP Functional Overview for a detailed hardware description list of the hardware elements contained within the 5416 processor.

3.2.2 Pin Assignments for the PGE Package [19]

The complete part number for the C5416 is TMS320VC5416PGE-160. All on-chip elements of the C5416 DSP are built on a 144-pin low-profile quad flatpack (LQFP). The pin assignments are shown below:

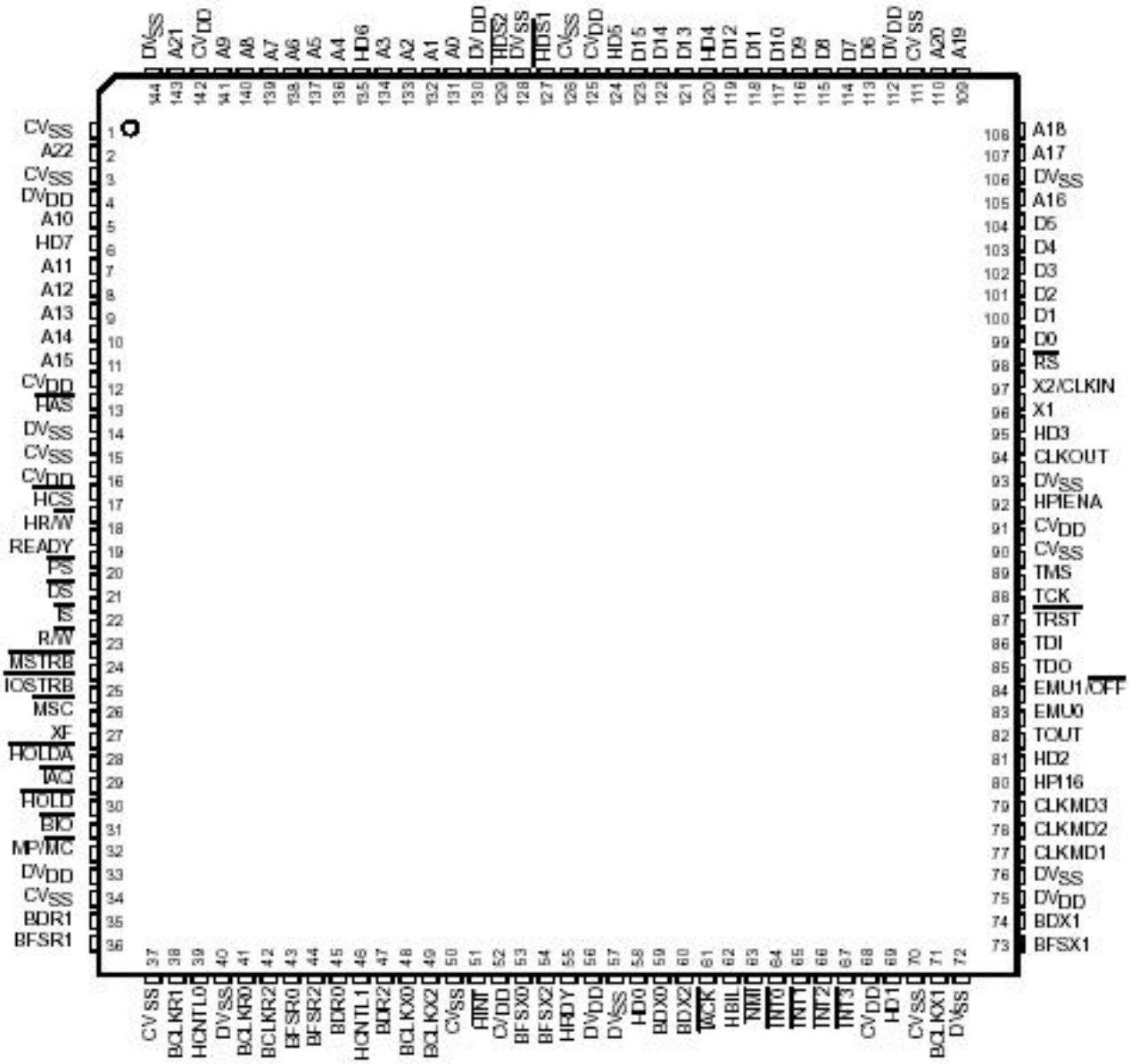


Figure 3-11 144-Pin PGE LQFP

3.2.3 Device and Development Support Tool Nomenclature [18]

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all TMS320 devices and supports tools such as TMS, TMP, or TMS. A full part number description is outlined below to help define the TMS320CV5416PGE:

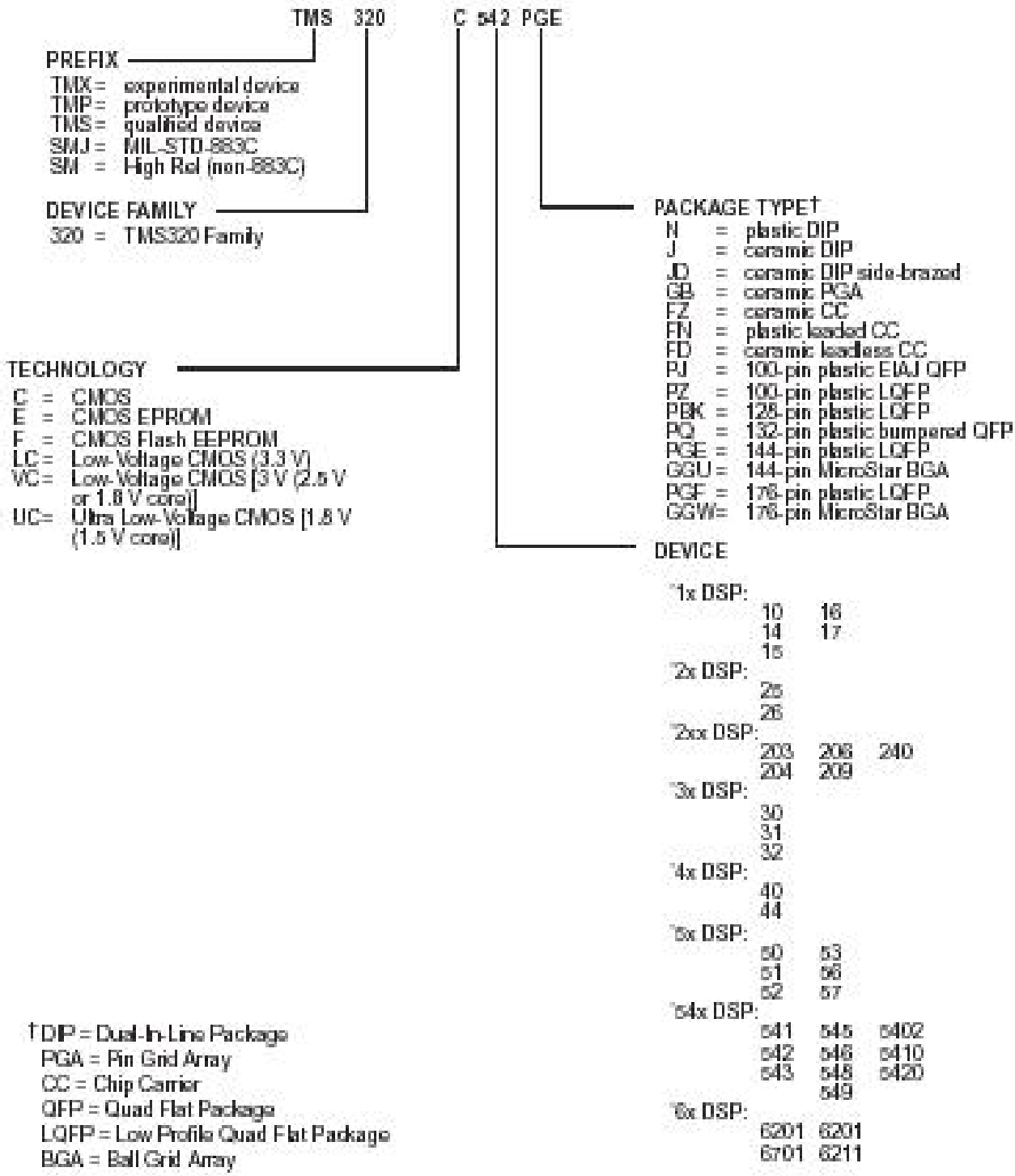


Figure 3-12 TMS320 Part Number Specification

The TMS320 DSP family consists of three supported DSP platforms: TMS320C2000 ™, TMS320C5000 ™, and the TMS320C6000™. Within the C5000 ™ DSP platform, there are three generations: TMS320C5x ™; TMS320C54x ™; and TMS320C55x ™. A general overview of these platforms is shown below:

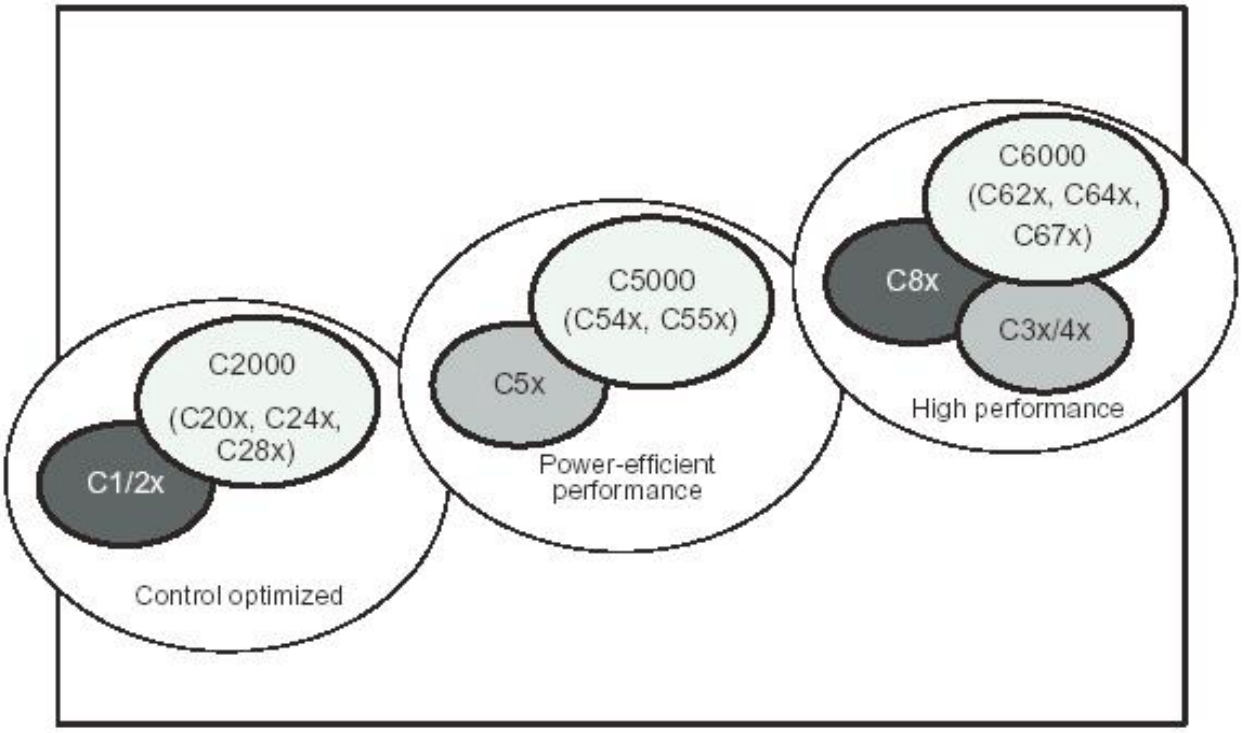


Figure 3-13 TMS320 Platforms

Referencing Figure 3-13, the C5416 can be seen to be a midrange TMS320 platform. The C6000 family has clock speeds up to 1 GHz, while the TMS320C67x model actually incorporates a math-coprocessor for floating-point hardware manipulation.

3.2.4 Programmable Bank-Switching [17] [18] [19]

Programmable bank-switching logic allows the C5416 to switch between external memory banks without requiring external wait states (Reference 3.2.10). This process inserts one cycle automatically when crossing memory-bank boundaries inside or between Program memory and Data memory space. This extra cycle allows memory devices to release the bus before other devices start driving the bus. Bank-switching is defined by the bank-switching control register (BSCR), which is memory-mapped at address 0029_{16}.

3.2.5 Enhanced 8-/16-Bit Host-Port Interface (HPI8/16) [17] [19]

The HPI8/16 is a parallel I/O port using an 8 or 16-bit bi-directional bus, which can be used to interface to an external host processor through the DSK Host Port Interface Expansion Connector, P3. This port is controlled by the HPI address register (HPIA), HPI data register (HPID), and HPI control register (HPIC). The host, using the DMA bus, and DSP have access to the entire on-chip RAM at all times based on the DSP clock with the host taking priority. As a result, the HPI could be used for remote boot loading, monitoring, control, or hardware expansion to an off-board host. Finally, the HPI memory is mapped into Data memory space as shown below:

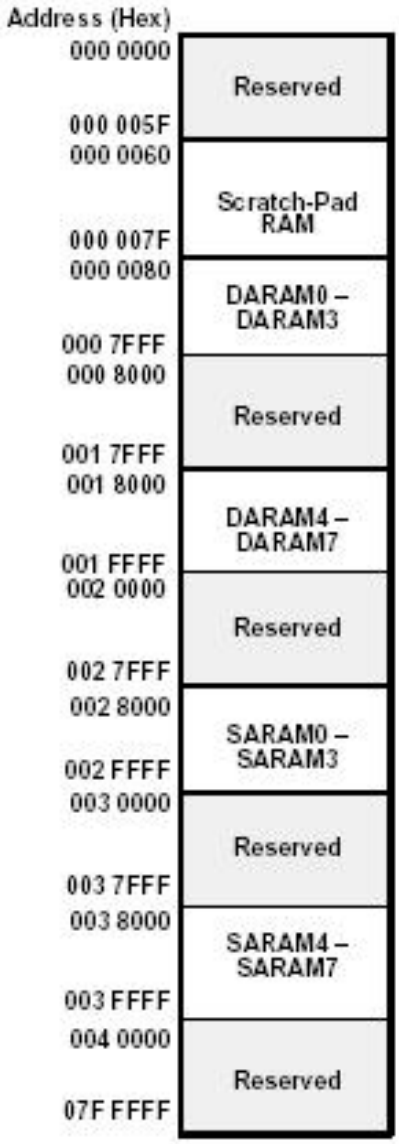


Figure 3-14 HPI Memory Map

3.2.6 Multichannel Buffered Serial Ports [17] [18] [19]

The C5416 provides three high-speed, full duplex multichannel buffered serial ports. These ports allow direct interface to external devices such as the on-board Codec and off-board devices that may be associated with Daughter Cards. Two memory-mapped registers are used for data transfer, which are called the data-transmit register (DXR) and the data-receive register (DRR). The serial data can be transferred in 8, 12, 16, 20, 24, or 32-bits. Serial port receive and transmit operations can generate their own maskable transmit and receive interrupts (XINT and RINT), which allows software control of serial-port transfers. In addition, the C5416 is capable of transmitting and receiving up to 128 channels. Each McBSP external interface consists of seven pins, which are outlined below:

Table 3-8 McBSP External Interface Pins

Pin Name	Description
BCLKX	Transmit reference clock
BDX	Transmit data
BFSX	Transmit frame sync
BCLKR	Receive reference clock
BDR	Receive data
BFSR	Receive frame sync
BCLKS	External clock reference for the programmable clock generator

The CPU or DMA can initiate transmission of data by writing to the DXR. Data that has been written to the DXR is shifted out utilizing a transmit shift register (XSR) through the BDX pin. This structure allows the DXR to loaded with the next word while the current data is being transmitted.

The CPU or DMA can read received data through the DRR. Data that has been received in on the BDR pin is shifted into a receive shift register (RSR) and then buffered into the receive buffer register (RBR). If the DRR is empty, the RBR information is copied into the DRR. If the DRR is in use, the RBR holds the data until the DRR is available. This configuration allows for storage of two previous words, while reception of the current word is in progress.

Since not all C54x devices with McBSPs implement the BLCKS pin, the C5416 is configurable to allow either the BCLKR or BCLKX pin to be used as the input clock to the sample rate generator. This configuration is set by using a combination of bit 7 (enhanced sample clock mode, SCLKME) of the pin control register (PCR) and bit 13 (McBSP sample rate generator clock mode, CLKSM) of the sample rate generator register 2 (SRGR2).

The McBSP allows the clock and frame sync generation to be programmable as well. The programmable functions are as follows: frame sync pulse width; frame period; frame sync delay; clock reference (internal versus external); clock division; and clock and frame sync polarity.

The McBSP allows for compression of data also. Two formats are available which are either μ-law or A-law. When compression is being utilized, data is encoded based on either the μ-law or A-law format, while the received data is decoded in a 2's complement format.

The C5416 allows for up to 128 independent channels. When one of the available 128 channels is selected, each frame represents a 128-bit time-division multiplexed (TDM) data stream. In other words, each channel consists of a frame of 128 bits, where the frames corresponding to each channel have been multiplexed based upon a clock.

3.2.7 General-Purpose I/O (GPIO) Pins [17] [18] [19]

Every C54x device provides two general purpose I/O pins called BIO and XF. BIO is a general input pin upon which conditional instructions can be based. The XF pin is an external

flag output that can be driven low or high under software control. Quite often the BIO and XF functions are used for handshaking. In addition, the C5416 has 26 additional pins that are multiplexed between the GPIO, HPI, and McBSPs functions that are controlled by software.

3.2.8 Hardware Timer [17] [18] [19]

All C54x devices feature a 16-bit timing circuit with a four-bit pre-scalar. The timer counter is decremented by one every time a CLKOUT cycle occurs. Each time the counter decrements to zero, a timer interrupt is generated. The timer can be stopped, restarted, reset, or disabled by specific status bits.

3.2.9 Clock Generator [17] [18] [19]

The clock generator provides clocking to the C5416 device. This device is based upon a phase-locked loop (PLL) circuit. This hardware requires an external clock reference, which is connected to the X2/CLKIN, while the X1 pin is not connected. By multiplying this external clock by a scale factor an internal faster or slower frequency may be obtained depending upon mode of operation.

The PLL is software controllable and can be configured in one of two modes being the PLL mode or DIV (divider) mode. In the PLL mode the clock on the X2/CLKIN pin is multiplied by 1 of 31 possible ratios, which allows for an external source with a much slower clock then compared to the C5416. In the DIV mode, the input clock on same pin is divided by 2 or 4 at which time the PLL circuitry can be disabled to conserve power.

The software-programmable PLL is controlled through the 16-bit memory-mapped clock mode register (CLKMD) address 0058_{16}. However, upon reset, the CLKMD register is initialized with a predetermined value dependent only upon the state of CLKMD1, CLKMD2, and CLKMD3 pins. The CLKMD pin configurations are shown below:

Table 3-9 Clock Mode Settings at Reset

CLKMD1	CLKMD2	CLKMD3	CLKMD RESET VALUE	CLOCK MODE
0	0	0	0000_{16}	½ (PLL disabled)
0	0	1	9007_{16}	PLL x 10
0	1	0	4007_{16}	PLL x 5
1	0	0	1007_{16}	PLL x 2
1	1	0	$F007_{16}$	PLL x 1
1	1	1	0000_{16}	½ (PLL disabled)
1	0	1	$F000_{16}$	¼ (PLL disabled)
0	1	1	-	Reserved (Bypass Mode)

3.2.10 Enhanced External Parallel Interface (XIO2) [17] [19]

The XIO2 has several features built into its logic such as the ability for DMA transfers to extend to external memory, insertion of bank switching cycles when crossing 32K memory boundaries (See Section 3.2.4), programming up to 14 wait states through software (See Section 3.1.1.3.8), and the ability to divide down the CLKOUT signal by a factor of 1, 2, 3, or 4. Dividing down the CLKOUT signal is an alternative to wait states when interfacing with slower external hardware. The CLKOUT divide-down factor is controlled through the DIVFCT field in the bank-switching control register (BSCR).

3.2.11 DMA Controller [17] [18] [19]

The C5416 direct memory-access (DMA) controller transfers data between points in the memory map without intervention by the CPU. This bi-directional transfer realm includes internal RAM, internal peripherals such as the McBSPs, and external off-chip memory devices, which can all be happening in the background while the CPU is performing other operations. The DMA has six independent programmable channels. In addition, the DMA even has higher priority for memory access then the CPU. Internal on-chip transfers can occur in either 32-bit double word or single word 16-bit formats, while external transfers are limited to single word 16 bit transfers. Furthermore, the C5416 DMA controller does not support transfers between peripheral devices to external memory and vice versa (i.e. No DMA support for transfers from Daughter cards to or from external on-board memory.), transfers between external to external devices, and synchronized external transfers.

Since the DMA controller on the C5416 supports transfers of data between Program, Data, and I/O space, two memory maps are defined for the DMA with both being subsets of the overall Program, Data, and I/O memory maps shown on the following figures:

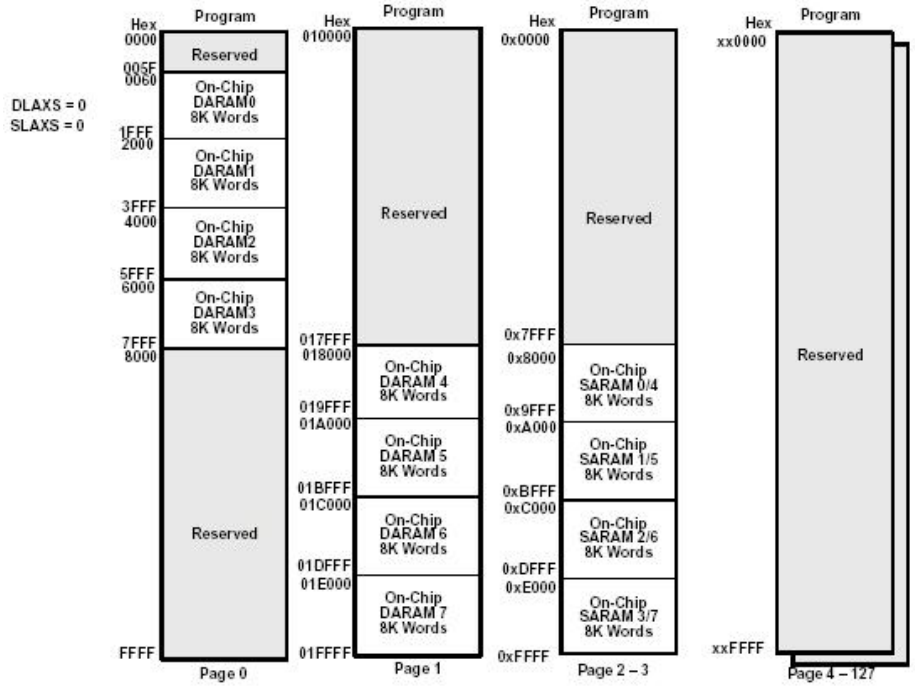


Figure 3-15 DMA Memory Map for Program Space

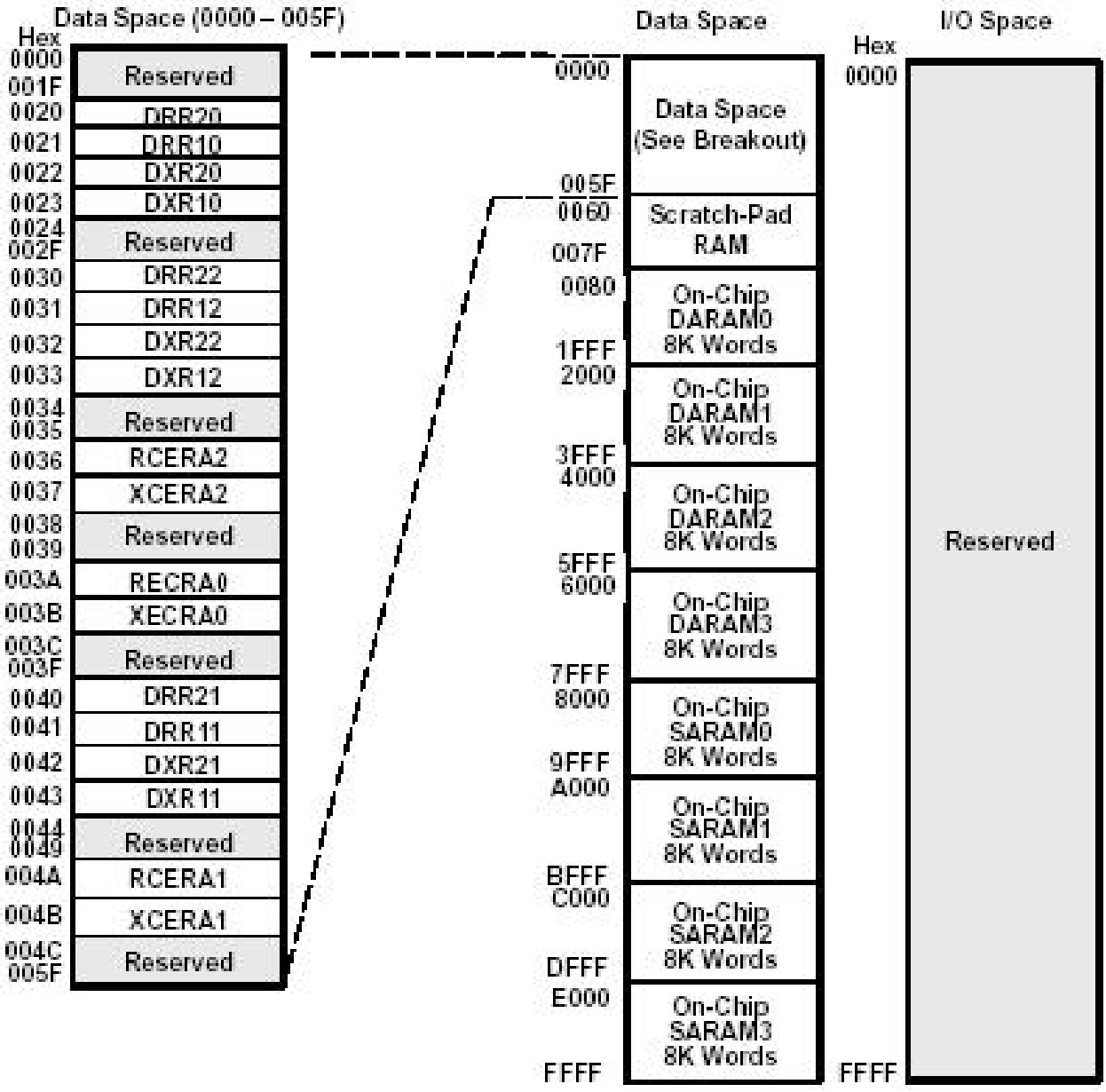


Figure 3-16 DMA Memory Map for Data and I/O Space

Based upon the way these DMA memory maps are defined, the DMA memory space is independent of the memory control bits MP/MC', DROM, and OVLY contained within the PMST register.

External memory accesses are possible with the C5416 DMA controller as has been previously stated, but only two of the six channels may be used in this manner. One channel is for reading, while the other is for writing. For more detailed information regarding the C5416 DMA controller, please refer to TI's TMS320VC5416 Fixed-Point Digital Signal Processor Data Manual.

3.3 Power Requirements [4]

The DSK can be powered by two different methods. The default power connection is through a standard double pole 2.5 mm jack, J6. This connection provides a +5 Volt and reference ground. J6 is located on the bottom side of the lower left corner of the DSK. The DSK comes with a +5 Volt 3 Amp power supply, which can be connected to a standard 120 V AC 60 Hz source. However, the DSK can also be powered through, J5, which is referred to as the optional power connector. This connection is a 4-pole system with the following pinout:

Table 3-10 Pin-Out for Optional Power Connector, J5

Pin #	DC Voltage Level
1	+12 Volts
2	-12 Volts
3	Ground
4	+5 Volts

The ±12 Volts DC sources are required for certain Daughter Cards such as the AED-109. The DSK is not delivered with a connector soldered onto the J5 leads since J5 and J6 should not be connected simultaneously or damage can result to the board. If the J5 connection is required, this leads can be populated using a Molex part number 15-24-4041 connector.

3.4 Switches [4]

The C5416 DSK has two switches, which are the Reset switch, and a 4-position user DIP switch. There are three methods of resetting the DSK. The first one is an automated power on reset. This circuit waits until the power supply is within acceptable an acceptable range before asserting pin RS' on the PGE. The second is software driven through the on-board USB JTAG emulator. The third is a on-board pushbutton, S1, which is located on the bottom side immediately below the external peripheral interface, P2.

3.5 C5416 DSK Reference Designator Layout [4]

A graphical illustration of various reference designators located throughout the DSK is shown below:

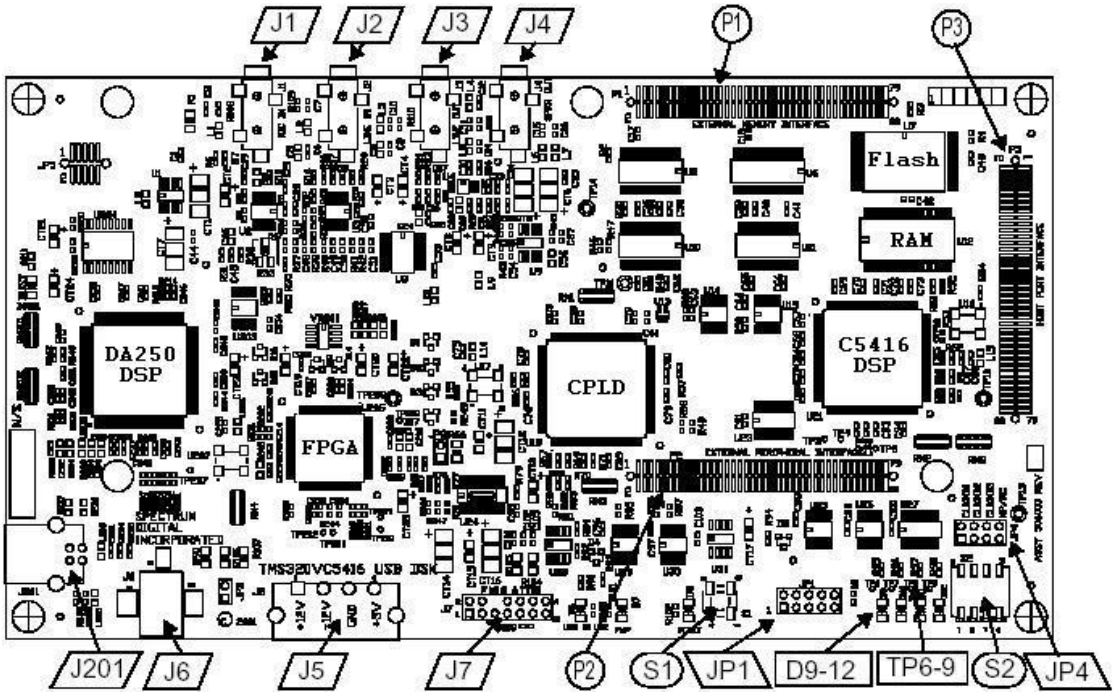


Figure 3-17 DSK Reference Designator Board Layout

3.6 External JTAG Connector, J7 [4]

When using Code Composer Studio, the standard method of emulation is through the J201 USB JTAG port. However, the emulation can occur through the on-board 14-header JTAG connector whose reference designator is J7. This is the standard method for interfacing with TI's DSP when not using a developer's starter kit. As a result, the ability to utilize this method has been provided for user convenience. The pinout for this connection is shown below:

Table 3-11 Pin-Out for JTAG 14-Pin Header, J7

Pin #	DC Voltage Level
1	TMS
2	TRST-
3	TDI
4	GND
5	PD
6	no pin
7	TDO
8	GND
9	TCK-RET
10	GND
11	TCK
12	GND
13	EMU0
14	EMU1

3.7 USB Embedded JTAG Emulation Connector, J210 [4]

Connector J201 provides a universal serial bus (USB) interface between the Code Composer Studio's (CCS) debugger and the emulation logic located on the DSK. This is the standard method of connection for the DSK with CCS in order to allow configuration, monitoring, and testing. Officially, CCS cannot start without this connection being made; however, the CCS C5416 version 2.10.05 developmental package can be started unofficially by closing the CCS splash screen and terminating the error screen by clicking the upper right x. This approach will not work on other CCS versions, but it is a useful tool. The pinout for the USB connector is shown below:

Table 3-12 Pin-Out for USB JTAG Connector, J201

Pin #	DC Voltage Level
1	USBVdd
2	D+
3	D-I
4	USB Vss
5	Shield
6	Shield

3.8 LEDs [4]

The C5416 DSK has eight light emitting diodes (LEDs) on-board. Half of this can be user configurable, while the system LEDs cannot be programmed by the user. The four user definable LEDs are used by the system for the power on self test (POST), but are available to the user at all other times. The user LEDs are accessed via I/O address 0000_{16} as shown below:

Table 3-13 User LEDs

Ref Des	LED #	Color	Controlling Signal	On Signal State
D9	1	Green	CPLD Register 0, Data Bit 0	1
D10	2	Green	CPLD Register 0, Data Bit 1	1
D11	3	Green	CPLD Register 0, Data Bit 2	1
D12	4	Green	CPLD Register 0, Data Bit 3	1

The system LED's are reserved for defining the DSK status. They are defined below:

Table 3-14 System LEDs

Ref Des	Color	Function	On Signal State
D6	Green	USB Emulation in use. When the External JTAG emulator is used this LED is off.	1
D7	Green	+5 Volts DC Present	1
D8	Green	Reset Active	1
D201	Green	USB Active, Blinks during data transfer.	1

Chapter Four

Code Composer Studio

4.1 Code Composer Studio Overview [20] [21] [22] [23]

The C5416 DSK is delivered with a specific version of Texas Instrument's flagship development tool Code Composer Studio (CCS) referred to as C5416 DSK CCS™ V2.1 IDE. CCS consists of a C/C++ compiler, assembler, linker, integrated development environment (IDE), and a variety of other support utilities. The IDE is the graphical user interface (GUI) of CCS. It consists of an editor for creating source code, a debugger for real-time troubleshooting, and a project manager for file organization and configuration before calling the appropriate components to compile, assemble, and link. The main difference between the DSK and complete versions of CCS is that the DSK version can only be ran with the TMS32VC5416 DSK and there are no upgrade rights.

The source code utilized within CCS can be written in C, C++, assembly, or a combination thereof. If the source code is written in C or C++, the extended address C runtime library rts_ext must be included for C54x processors. The C language that the TMS32054x supports is based upon the ANSI C standard as described in the 2nd edition of Kernighan and Ritchie's "The C Programming Language", while the C54x compiler also supports C++ as defined in Ellis and Sroustrup's "The Annotated C++ Reference Manual" and many features contained within the ISO/IEC 14882-1998. However, complete C++ standard library support is not included, with libraries such as iostream being omitted. Alternatively, the programmer may choose to program at the assembly level utilizing the TMS320C54x instruction set, which is divided into four basic types: arithmetic; logical; program-control; and load store operations. For a complete list of the available instruction, the reader is directed to TI's technical manual spru172c.

The most common software development path involves utilizing the ANSI C or C++ route instead of writing at the assembly level as shown below:

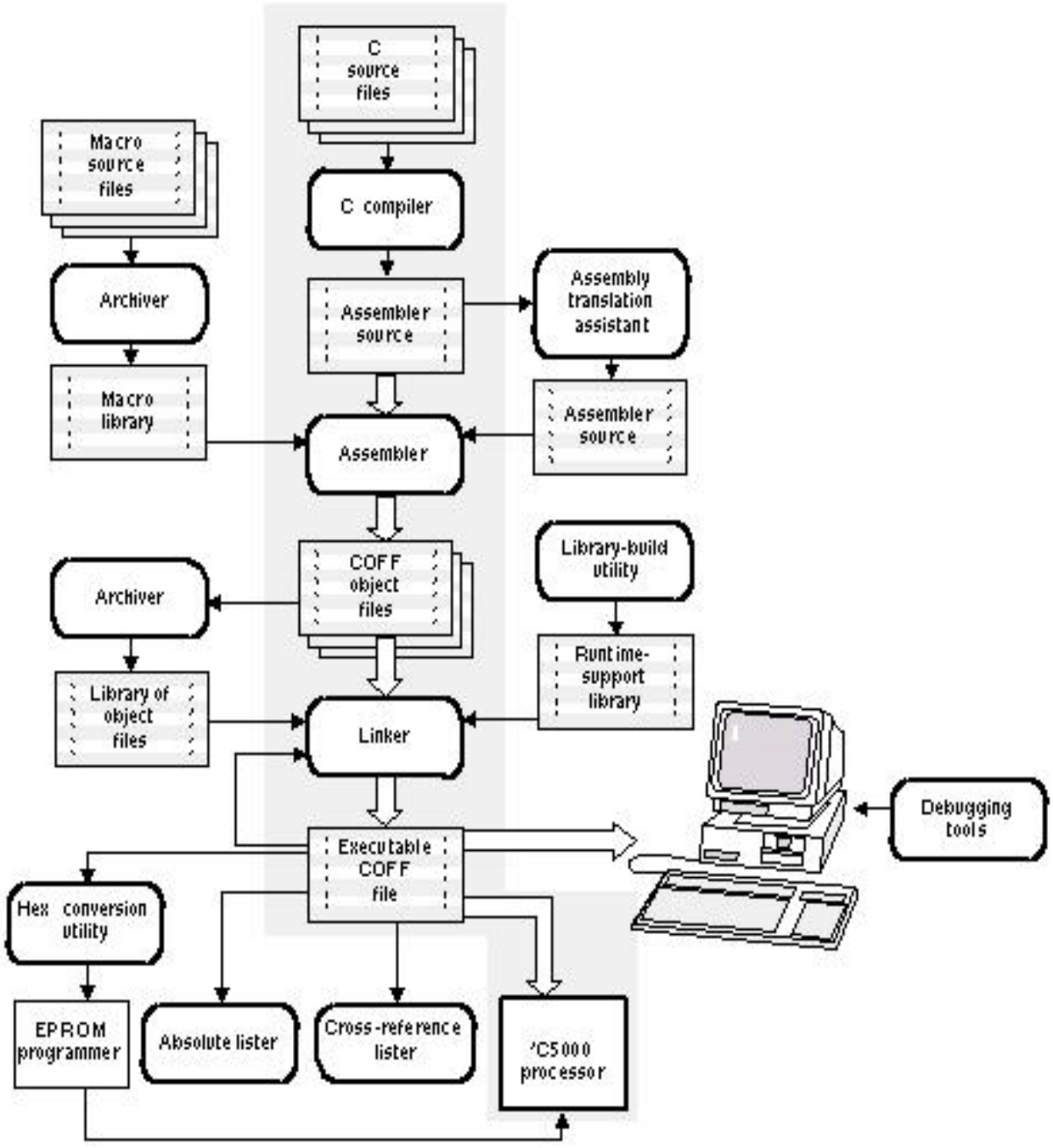


Figure 4-1 CCS Software Development Flow

Based upon Figure 4-1, the C or C++ code is first compiled into assembly instructions, then assembled into machine language object files, and finally linked in order to combine the several object files into one single executable file. Other development paths are available, but are less common and were not used in this project; therefore, for further details on this subject please consult TI's technical literature.

4.2 System Requirements [24]

CCS runs on a Microsoft Windows platform with the following operating systems being supported: Windows 98SE; ME; 2000 SP1 or higher; and XP. The minimum hardware requirements are as follows: 233 MHz Pentium™; 600 Mb disk free space; 64 Mb of RAM; SVGA (800 x 600) display; Internet Explorer™ (4.0 or later) or Netscape Navigator™ (4.0 or later); and a local CD-ROM drive. However, for better performance, TI prefers at least 128 Mb Ram, 16-bit color, and a 500 MHz Pentium™ or higher processor.

4.3 Installation [24]

In order to install CCS on the host PC, the CCS installation CD should first be loaded into the CD-ROM drive from which point an install screen should appear on the host monitor. If no install window appears, go to the Start menu and select Run from which the browse button should be utilized to select the setup executable file on the CD-ROM. The CCS install option should then be selected, after which the dialog boxes should be responded to appropriately as they appear on the screen. Once the install program has finished, two icons should have been placed on the user's desktop titled C5416 DSK Startup and the C5416 DSK Diagnostic Utility. After the DSK has been connected to the PC through the USB port, the DSK startup icon should then be selected. The Launch Setup option from the File pull-down menu should then be selected in order to configure CCS for the appropriate target board. Drivers from the CD-ROM will be needed to complete this process. The Diagnostic Utility program is used to perform detailed tests on the various board subsystems such as the DSP core, codec, memory, and on-board emulation. It can run in standalone mode or from within Code Composer. The PC should be restarted before running this utility.

4.4 Project Management [21] [25]

The best way to introduce the project management capabilities it to first review the CCS IDE, which is shown below:

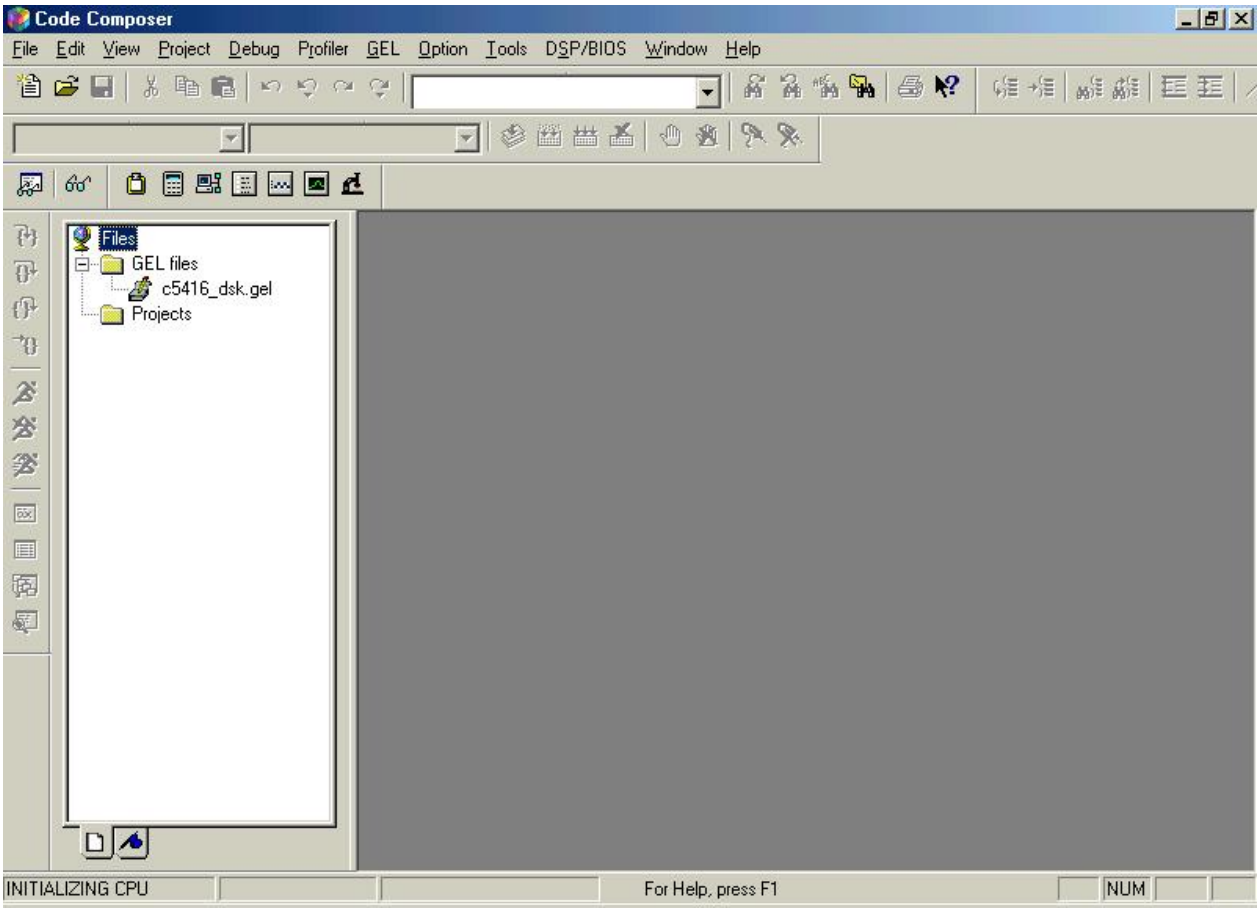


Figure 4-2 CCS IDE

The project view window is on the left, while the line editor is on the right. Within the project window, the general extension language (GEL) file is already present. This file is created during the target board configuration, and will not vary between projects as long as the same C5416 DSK target is being utilized.

In order to open a project, the Open option from the Project pull-down menu should be selected. Underneath the Project folder, a project file will now appear, which can be opened by highlighting the file and double-clicking on the left button of the mouse. At this point, the linker command file and the following folders will now be visible: DSP/BIOS Config; Generated Files; Include; Libraries; and Source. The linker command file or any of the listed folders can be opened by highlighting and double-clicking the left button on the mouse. With the exception of library files, the linker command file and all other files may be opened in the line editor by highlighting and double-clicking the left mouse button with the following results:

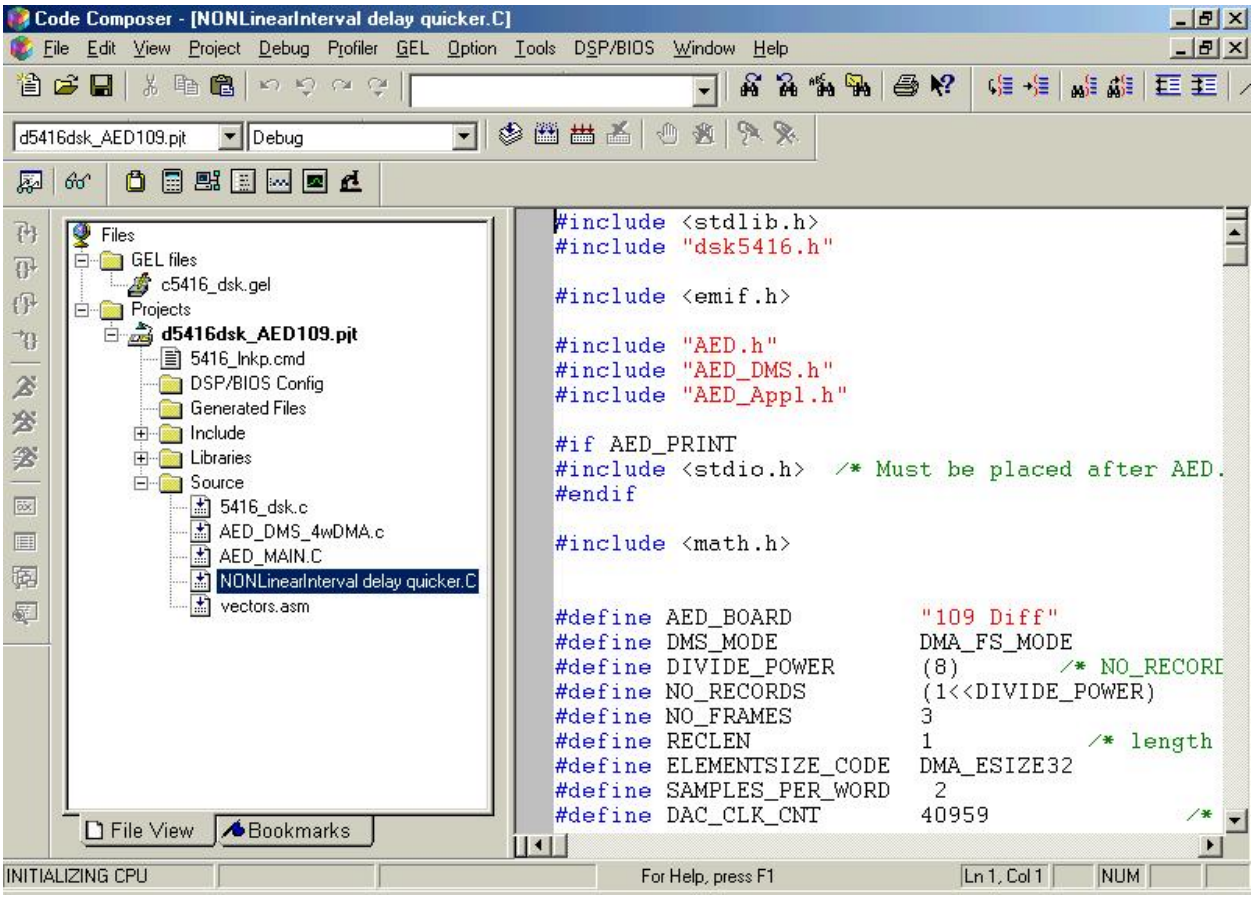


Figure 4-3 Project Tree and Line Editor Display

Alternatively, a new project could have been created by selecting the New option under the Project pull-down menu and selecting the appropriate options.

Files may be added or removed from the project at any time. To remove a file, the Remove from Project option must be selected from the pull-down menu after highlighting the subject file and single clicking the right button on your mouse. In order to add a file, the Add Files to Project option under the Project pull-down menu must be selected.

The Compile, Assemble, and Link functions can be accomplished for the entire project by selecting the Rebuild All option under the Project pull-down menu. If errors are detected during compilation, the associated file, line, and error message is displayed in the Debug Window, which is automatically opened upon start of the compilation process. If compilation, assembly, and linking were successful a zero error message will be displayed instead in the same Debug window. At this point, the executable file may be downloaded to the target DSK by selecting the Load option from the File pull-down menu. The execution of the program can then commence by selecting the Run option under the Debug pull-down menu, and halted by manually by selecting the Halt option under the Debug pull-down menu as well. After termination, the executable code is not automatically reset. The program counter is residing at the last code line executed. If a reset is wished, the code can be simply reloaded into the DSK,

which resets the program counter implicitly, or the Reset CPU Option under the Debug pull-down menu can be selected to explicitly perform the same function.

The CCS line editor incorporates a CodeMaestro coding assistant. This feature assists the programmer in creating syntactically correct code by suggesting words that are incompletely typed, listing members of a structure or object, displaying parameter information, correcting case, color coding, and creating a line limit for CodeMaestro deactivation to avoid excessive line editor response times. Alternatively, an external third party line editor can be configured within CCS, but is only available for editing files and not the various debug functions offered by CCS.

4.5 CCS Debug Tools [21] [25]

The CCS IDE Debug tools are supplied to provide a means of allowing the user to troubleshoot software and automate the configuration process for on-board data converters through the use of the Data Converter Plug-In. Perhaps the easiest and most useful debug function is the ability to determine the value of a variable after the execution has been halted by simply placing the cursor over the variable of interest in the line editor. CCS also provides a Selection Margin on the left side of the line editor, which is used to the display the program counter after execution termination with a yellow arrow and the manual setting of Bookmarks, Breakpoints, or Probe Points.

4.5.1 Bookmarks [21] [25]

Bookmarks are represented by a blue flag. Highlighting a particular line in the code, clicking the right mouse button once, choosing the Bookmark pull-down menu, and selecting the Set a Bookmark option sets Bookmarks. The advantage of Bookmarks is that allows the programmer to quickly move to particular lines in the code. This is accomplished by highlighting a particular line in the code, clicking the right mouse button once, choosing the Bookmark pull-down menu, and selecting the Bookmarks, highlighting a Bookmark of interest, and selecting Go To. Each bookmark is required to have its own unique description. Adding, deleting, editing, and moving to a Bookmark location can also be accomplished by selecting Bookmarks option from the Edit pull-down menu.

4.5.2 Breakpoints [21] [25]

A red circle in the Selection Margin represents Breakpoints. The Breakpoint tool is software driven, which is implemented by actually modifying the target code at the location of choice in order to halt the execution of the program once the associated line has been reached. They can be set by selecting a line of code and double clicking the left mouse button. There is no limit to the number of Breakpoints that may be configured. Alternatively, they can be set by choosing the Debug pull-down menu and selecting the Breakpoint option. While controlling the welding process, the Breakpoint should never be utilized or damage to torch could result.

4.5.3 Probe Points [21] [25]

Probe Points are represented by a cyan diamond in the Selection Margin. They are set by selecting the Debug pull-down menu and choosing the Probe Points Option. Probe Points temporarily halt a program to allow for file I/O at which point the execution is resumed. As a result, Probe Points can be useful for troubleshooting, but cannot be used in a real-time application due to the latency they induce.

4.5.4 Watch Windows [21] [25]

The Watch Window is another very useful feature within Debug toolset. The Watch Window allows for the values contained within a variable or structure element to be monitored in real-time during execution. A Watch Window can be established by selecting the Watch Window Option under the View pull-down and manually typing a local or global variable. Alternatively, highlighting a variable within the Line Editor, clicking the right button on the mouse, and selecting Watch Window will establish the same goal. During code execution, the value represented by a variable or loaded into a memory location being monitored is constantly updated in the Watch Window in a numerical form.

The disadvantage of the Breakpoint or Watch Window methodology is that it difficult to visualize how the data set for a variable is varying during execution. This is overcome by utilizing the Graph Option within the View pull-down menu. As a result, a visual display of the complete data set can be built in real-time as the data is collected. By using this tool, errors, trends, and relationships can be evaluated much more efficiently.

4.5.5 Symbol Browser [21] [25]

The Symbol Browser is used to disseminate information contained within the executable output file. This tool is invoked by selecting the Symbol Browser option under the Tools pull-down menu. At this point, the Symbol Browser window is opened, which contains five separate tabs which are labeled: Files; Functions; Globals; Types; and Labels. This tool was not utilized in this project. For further details, please consult TI's technical literature.

4.5.6 General Extension Language [21] [25]

The General Extension Language (GEL) is an interpretive language similar to C. This syntax is used to allow the programmer to create user specific functions to extend the CCS IDE capability. GEL files can be created in any text editor external to CCS as long as the appropriate syntax is followed and the GEL file extension is utilized. In order to load GEL files into CCS, the Load GEL option under the File pull-down menu should be selected. At this time, the loaded GEL file will appear under the GEL folder within the project view window. The GEL files are not specific to a project. As a result, GEL functions can be accessed in any location within the source code as well as within the Watch Window. Upon initial target setup, the c5416_dsk gel file is configured by the CCS IDE. This file establishes initial register configurations in order to configure functions associated with starting, resetting, and initializing the C5416 DSP, peripherals, DMA, Multi-Channel Buffered Serial Ports, Timer, and the General Purpose I/O ports.

4.5.7 Command Window [21] [25]

The command window allows the programmer a method to manually type in commands that can be accessed by selecting the appropriate option under the correct pull-down menu. Many other software applications use the same thought process with CAD systems being a good examples. Typically, this feature is usually just a user's preference option; however, there are some commands that are not available in the pull-down menus that can only be accessed by manually typing in the correct syntax within the Command Window. The Command Window is opened by selecting the Command Window option under the Tools pull-down menu.

4.5.8 Data Converter Plug-In [21] [25]

The Data Converter Plug-In automates the configuration process for on-board data converters. This tool is activated by selecting the Data Converter Support Option under the Tools pull-down menu. Once activated, the correct data converter and DSP must be selected. In order to select the correct data converter, the appropriate data converter must be highlighted, the right mouse button must be clicked, and the add option must be chosen. A pull-down menu is then created for the selected data converter to allow for user option selection. From the Files pull-down menu, the write tab should then be selected. This tool will then write several ANSI C files to the active project. From these files, the appropriate functions can be called to utilize the on-board data converter. Furthermore, the C5416 DSK CCS™ V2.1 IDE is not delivered with this feature. However, the Data Converter Plug-In is considered shareware and can be downloaded from TI's website at no monetary cost.

4.6 CCS DSP/BIOS [21] [26] [27]

The DSP/BIOS is a scalable real-time kernel. This toolset is provided within the CCS IDE. When utilized, it formulates a very basic operating system for the C5416 DSK. The DSP/BIOS kernel is packaged as a set of modules that can be linked into the project application. Since there are over 150 DSP/BIOS application programming interface (API) functions that can be called by the application source code as an interface to the kernel, a scaleable DSP/BIOS kernel support library can be linked into the project application as necessary based upon how the DSP/BIOS API functions are referenced directly or indirectly by the application. Since the kernel is scaleable, features that are not being utilized may be disabled in order to optimize performance.

By definition, the kernel includes an interrupt handler, scheduler, supervisor, and memory manager. The interrupt handler receives and stores all requests for kernel services. The scheduler establishes the precedence, and the supervisor grants access. The memory manager organizes the memory map as required, which eliminates the need for creating a linker command file manually.

The kernel is configured using the DSP/BIOS Configuration tool. Selecting the DSP/BIOS Configuration option under the File/New pull-down menu accesses this tool. The DSP/BIOS Configuration tool provides the ability to statically declare and configure DSP/BIOS kernel objects during development rather then during code execution. These static objects exist

for the duration of the program. The DSP/BIOS does allow for dynamic creation and deletion of kernel objects during execution, but the statically defined objects will utilize the minimal memory footprint. In addition, the static objects allow for accurate predictions of memory requirements during execution.

C, C++, or assembly source code can utilize the DSP/BIOS kernel by calling the appropriate DSP/BIOS application programming interface (API) function. In order to allow the application to call these functions, the configuration file, which utilizes a cdb file extension, should be added to the project once appropriate template has been configured and saved. The cdb file is extracted into three files. The file called program.cdb is added to the DSP/BIOS folder, while the files programcfg.s62 and programcfg_c.c are both added to the Generated File folder. The linker command file generated by the DSP/BIOS configuration should then be added, while being sure to remove any other previously added linker command files. The vector.asm and rts_ext.lib should then be removed from the project since these are automatically defined within the DSP/BIOS configuration file.

4.6.1 CCS Chip Support Library [21] [26] [27]

The chip support library (CSL) is configured within the DSP/BIOS Configuration tool. The CSL provides a set of macros and functions that simplify the configuration and management of on-chip peripherals such as DMA blocks, memory or parallel interfaces, serial ports, and timers. By utilizing this feature, the programmer can realize significant timesavings by avoiding time-consuming manual configurations.

4.6.2 CCS Real-Time Analysis [21] [26] [27]

The DSP/BIOS Real-Time Analysis (RTA) tool utilizes the Real-Time Data Exchange (RTDX) feature to obtain, transfer, and display target data in a low-speed albeit real-time communication link between the target and Host computer. For example, DSP/BIOS provides a fully reentrant printf capability that can be executed in approximately 60 instruction cycles. As a result, a visibility into program execution is provided while implying a minimal intrusion into the real-time application processing.

The RTDX consists of both target and host components. A small RTDX software library runs on the target application emulator in order to pass data to or from the DSK through the JTAG interface. On the host platform, an RTDX Host library operates in conjunction with the Host CCS IDE to send or receive the same data. The Host library supports two modes of receiving data, which are Continuous and Non-Continuous. The Continuous mode is used when the RTDX Host library is required to act as a buffer to obtain and display data from the target application without storing. The Non-Continuous mode writes the received data into a log file for permanent storage.

The RTDX is configured under the Tools/RTDX pull-down menu. Under this selection, three choices are provides as follows: Diagnostic Control; Configuration Control; and Channel Viewer Control. The diagnostic feature provides a means of verifying that the RTDX is working properly, while the configuration feature allows the ability to disable/enable, view current

configuration settings, and modify the current configuration by accessing the RTDX Configuration Control Properties page. The Channel Viewer feature allows the user to add/remove and enable/disable RTDX channels as necessary.

4.7 Training Recommendation

This chapter has only provided a very basic overview of the Code Composer Studio software package. In addition, the DSP/BIOS and Chip Support Library were not utilized in this embedded application. However, if the full capacity of the DSK were to be achieved, it would be advantageous to utilize these tools since they provide a very powerful method for configuring on-board resources. In addition, there are other features that are offered within CCS IDE, but are not functional in the C5416 DSK implementation. As a result, it is recommended by the author that a Texas Instrument's CCS workshop be attended in order to allow the user to become familiar with this methodology in an efficient manner.

Chapter Five

Data Converter Daughter Card

5.1 AED-109 [6]

Signalware's 12-bit AED-109 Daughter Card capable of sampling rates up to 8 MHz is a relatively efficient and economical means of building a fast feedback control system when used in conjunction with a TI, DNA Enterprise, or Blue Wave Systems DSP platform. In particular, the AED-109 is ideally suited for evaluation modules (EVM) and DSP developer's starter kits (DSK) for TI's TMS320C6xxx and TMS320C5xxx series processors. Other applications of the AED-109 involve sensor processing and communications. A picture of the top surface of the AED-109 is shown below:

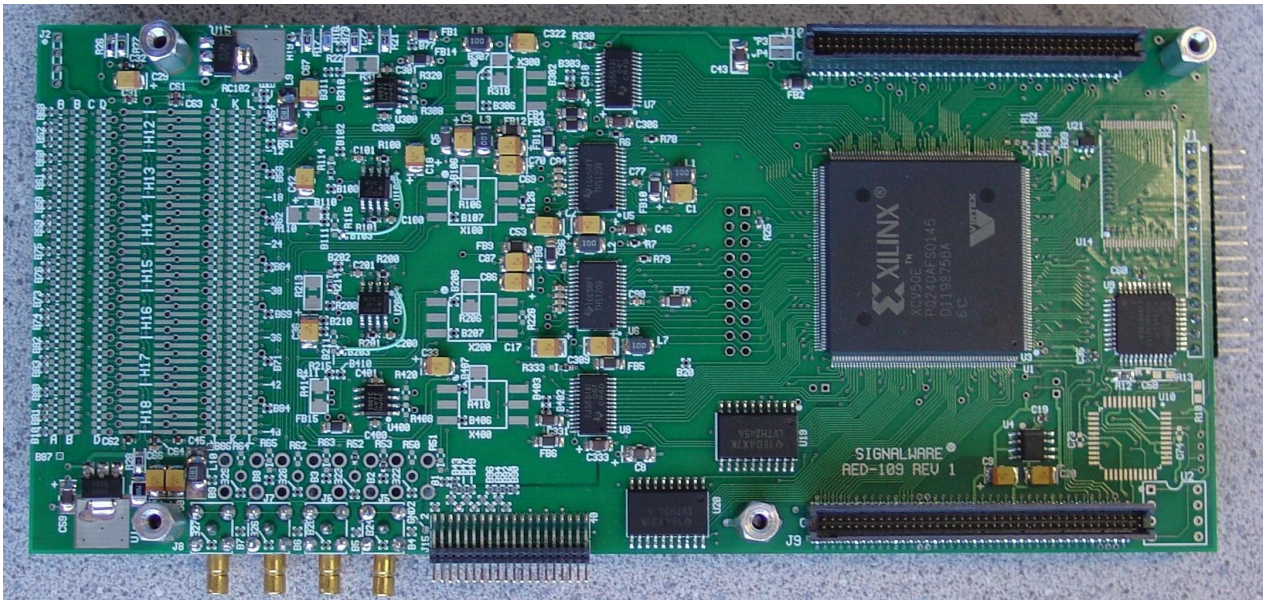


Figure 5-1 Signalware AED-109 Top Surface

The AED-109 is a versatile full size data converter whose main configuration consists: two dual channel A-to-D converters; two dual channel D-to-A converters; three adjustable voltage references; 16 digital I/O configurable ports; and a programmable logic interface for the on-board FPGA, and several connector interfaces which include: EVM Expansion; JTAG; Digital I/O; and Analog SMB. In addition, the AED-109 can be delivered in a number of different configurations based upon user requirements. Additional options include analog buffers, transformers, additional SMB connectors, flash memory for DSP boot loading, and a variety of AI/AO input/output configurations. A basic block diagram of the AED-109 operation is shown below:

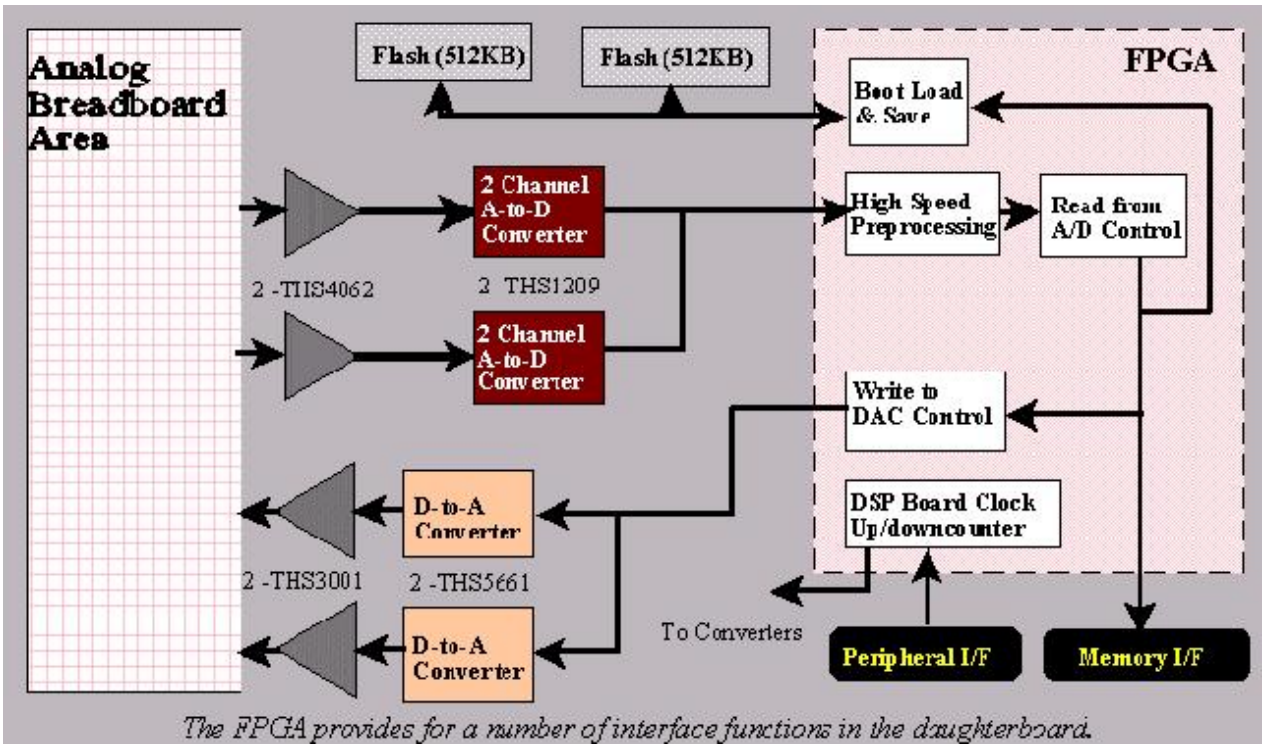


Figure 5-2 AED-109 Basic Block Diagram

5.2 EVM Expansion Interfaces [6]

There are actually two EVM expansion connectors called the expansion memory interface, J9, and the expansion peripheral interface, J10. Both connectors are 80-pin and are used to interface with the DSP platform, which in this case is the TMS320VC5416 DSK. Referencing Figure 5-1, it should be noted that the top layer of AED-109 is actually facing down when the EVM expansion connectors are engaged into the C5416. Furthermore, a stacking feature was not chosen for this board, although available; therefore, no expansion connectors are present on the bottom side of the board. The pin-out of the expansion memory interface is shown below, where the letter "O" refers to output, "I" refers to input, and "Z" refers to high impedance:

Table 5-1 Expansion Memory Interface, J9

Pin #	Signal Name	Type	FPGA Pin #
1	5 V	PWR	-
2	5 V	PWR	-
3	XA21	O	220
4	XA20	O	221
5	XA19	O	222
6	XA18	O	223
7	XA17	O	224
8	XA16	O	228

Table 5-1 Expansion Memory Interface, J9 (Continued)

Pin #	Signal Name	Type	FPGA Pin #
9	XA15	O	229
10	XA14	O	230
11	GND	-	-
12	GND	-	-
13	XA13	O	-
14	XA12	O	-
15	XA11	O	-
16	XA10	O	-
17	XA9	O	-
18	XA8	O	-
19	XA7	O	-
20	XA6	O	-
21	5 V	PWR	-
22	5 V	PWR	-
23	XA5	O	231
24	XA4	O	236
25	XA3	O	237
26	XA2	O	238
27	\XABE3	O	-
28	\XABE2	O	-
29	\XABE1	O	-
30	\XABE0	O	-
31	GND	-	-
32	GND	-	-
33	XD31	I/O/Z	53
34	XD30	I/O/Z	52
35	XD29	I/O/Z	50
36	XD28	I/O/Z	49
37	XD27	I/O/Z	48
38	XD26	I/O/Z	47
39	XD25	I/O/Z	46
40	XD24	I/O/Z	41
41	3.3 V	PWR	-
42	3.3 V	PWR	-
43	XD23	I/O/Z	40
44	XD22	I/O/Z	39
45	XD21	I/O/Z	38
46	XD20	I/O/Z	36
47	XD19	I/O/Z	35
48	XD18	I/O/Z	34
49	XD17	I/O/Z	33
50	XD16	I/O/Z	31
51	GND	-	-

Table 5-1 Expansion Memory Interface, J9 (Continued)

Pin #	Signal Name	Type	FPGA Pin #
52	GND	-	-
53	XD15	I/O/Z	5
54	XD14	I/O/Z	6
55	XD13	I/O/Z	7
55	XD12	I/O/Z	9
57	XD11	I/O/Z	11
58	XD10	I/O/Z	12
59	XD9	I/O/Z	13
60	XD8	I/O/Z	17
61	GND	-	-
62	GND	-	-
63	XD7	I/O/Z	18
64	XD6	I/O/Z	19
65	XD5	I/O/Z	21
66	XD4	I/O/Z	23
67	XD3	I/O/Z	24
68	XD2	I/O/Z	26
69	XD1	I/O/Z	27
70	XD0	I/O/Z	28
71	GND	-	-
72	GND	-	-
73	\XRE	O	235
74	\XWE	O	234
75	\XOE	O	20
76	XRDY	I	42
77	SPARE (N/C)	-	-
78	\XCE1	O	10
79	GND	-	-
80	GND	-	-

The expansion peripheral connector pin-out is shown below:

Table 5-2 Expansion Peripheral Interface, J10

Pin #	Signal Name	Type	FPGA Pin #
1	12 V	PWR	-
2	-12 V	PWR	-
3	GND	-	-
4	GND	-	-
5	5 V	PWR	-
6	5 V	PWR	-
7	GND	-	-
8	GND	-	-

Table 5-2 Expansion Peripheral Interface, J10 (Continued)

Pin #	Signal Name	Type	FPGA Pin #
9	5 V	PWR	-
10	5 V	PWR	-
11	SPARE (N/C)	-	-
12	SPARE (N/C)	-	-
13	RSVD (N/C)	-	-
14	RSVD (N/C)	-	-
15	RSVD (N/C)	-	-
16	RSVD (N/C)	-	-
17	SPARE (N/C)	-	-
18	SPARE (N/C)	-	-
19	3.3 V	PWR	-
20	3.3 V	PWR	-
21	XCLKX0	I/O/Z	213
22	XCLKS0	I	102
23	XFSX0	I/O/Z	101
24	XDX0	O	100
25	GND	-	-
26	GND	-	-
27	XCLKR0	I/O/Z	99
28	SPARE (N/C)	-	-
29	XFSR0	I/O/Z	97
30	XDR0	I	96
31	GND	-	-
32	GND	-	-
33	XCLKX1	I/O/Z	95
34	XCLKS1	I	94
35	XFSX1	I/O/Z	93
36	XDX1	O	87
37	GND	-	-
38	GND	-	-
39	XCLKR1	I/O/Z	86
40	SPARE (N/C)	-	-
41	XFSR1	I/O/Z	84
42	XDR1	I	82
43	GND	-	-
44	GND	-	-
45	TOUT0	O	92
46	TINP0	I	81
47	SPARE (N/C)	-	-
48	SPARE (N/C)	-	-
49	TOUT1	O	80
50	TINP1	I	79
51	GND	-	-

Table 5-2 Expansion Peripheral Interface, J10 (Continued)

Pin #	Signal Name	Type	FPGA Pin #
52	GND	-	-
53	XENT INT7	I	78
54	IACK	O	-
55	INUM3	O	-
55	INUM2	O	-
57	INUM1	O	-
58	INUM0	O	-
59	\XRESET	O	74
60	DSP_PD	O	-
61	GND	-	-
62	GND	-	-
63	XCNTL1	O	73
64	XCNTL0	O	72
65	XSTAT1	I	71
66	XSTAT0	I	70
67	SPARE (N/C)	-	-
68	SPARE (N/C)	-	-
69	\XCE2	O	-
70	\XCE3	O	-
71	DMAC3	O	68
72	DMAC2	O	67
73	DMAC1	O	66
74	DMAC0	O	65
75	GND	-	-
76	GND	-	-
77	GND	-	-
78	XCLKOUT2	O	89
79	GND	-	-
80	GND	-	-

5.3 JTAG Header [6]

In addition, this Signalware board is delivered standard with a 14-pin single row header JTAG port, J1, which is used in the programming configuration of the on-board FPGA and associated flash. This port is located on the right side of the board referencing Figure 5-1, where pin 1 is located at the top with a dot adjacent. The pin-out of J1 is shown below:

Table 5-3 JTAG Pin-Out, J1

Pin #	Signal Name
1	GND
2	INIT
3	D_+3P3V
4	GND
5	CCLK
6	DONE
7	DIN
8	PROG
9	TCK
10	TDO
11	TDI
12	TMS
13	D_+3P3V
14	GND

5.4 Digital I/O Connector [6]

The Digital I/O connector is a 40-pin, 50 mil pitch, double-row connector designated, J15. This connector has 16 digital I/O ports, 6 grounds, and 18 pins that are connected to pads adjacent to the breadboard area. The pin-out for this connector is shown below:

Table 5-4 Digital I/O Pin-Out, J15, and FPGA Digital I/O Control Lines

Pin #	Signal Name	FPGA Pin #
1	Digital Ground	-
2	BB Pad	-
3	BB Pad	-
4	Digital Ground	-
5	BB Pad	-
6	BB Pad	-
7	BB Pad	-
8	BB Pad	-
9	BB Pad	-
10	BB Pad	-
11	BB Pad	-
12	BB Pad	-
13	Digital Ground	-
14	I/O 24	-
15	I/O 23	-
16	I/O 22	-
17	I/O 21	-
18	I/O 20	-
19	I/O 19	-

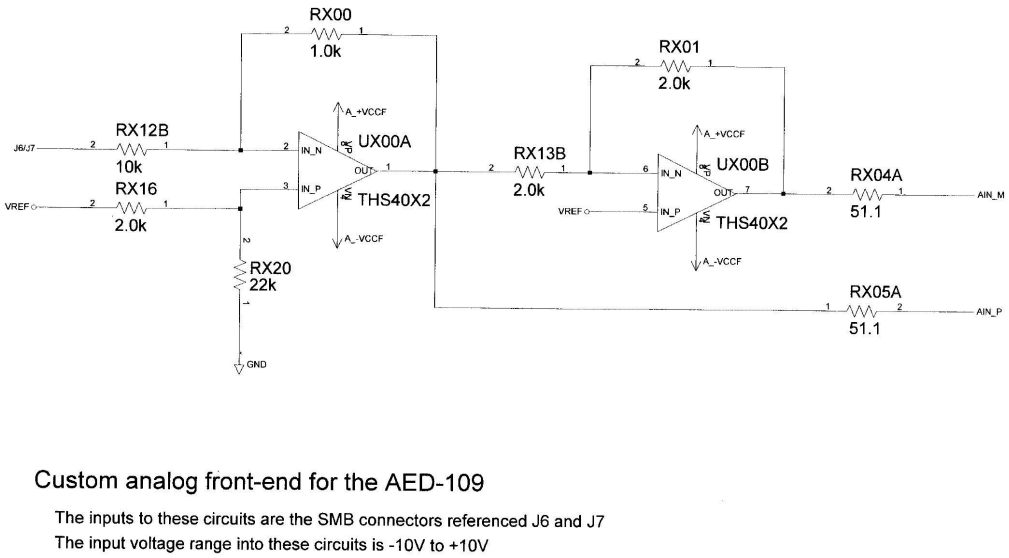
Table 5-4 Digital I/O Pin-Out, J15, and FPGA Digital I/O Control Lines (Continued)

Pin #	Signal Name	FPGA Pin #
20	I/O 18	-
21	I/O 17	-
22	Digital Ground	-
23	I/O 16	200
24	I/O 15	201
25	I/O 14	195
26	I/O 13	199
27	I/O 12	193
28	I/O 11	194
29	I/O 10	191
30	I/O 9	192
31	Digital Ground	-
32	O8	218
33	I7	217
34	O6	216
35	I5	215
36	O4	209
37	I3	208
38	O2	206
39	I1	187 or 210 for Clock Input
40	Digital Ground	-
-	I/O Enable	185
-	I/O 9–16 Direction	178
-	I1, I3, I5, I7 Enable	177

5.5 Analog I/O Connectors [6]

The Analog I/O can be connected through 8 SMB connectors. There are eight ports with plated-through hole connection points available on the AED-109 board for these AI or AO connectors. Not all eight ports are populated at delivery. As a result, it should be realized that the Analog I/O could be configured in a single-ended or differential manner. When in the differential mode, the SMB connectors can be configured in two different manners with the intention of being used in conjunction with either a floating (single) coax or dual coax cable. For a single coax, the negative signal is wired to the backshell of the SMB resulting in only one SMB connector being used for a single differential input; however, for a dual coax, the positive and negative signals are actually brought in on two different SMB connectors and the backshell of each connector is grounded resulting in two SMB connectors being used for a single differential input. In the single ended mode, the backshell is grounded and only one SMB connector is utilized per single ended input. However, it should be realized that this is merely the manner in which the signal is delivered to the board and not necessarily the method in which the signal is introduced to the THS1209 ADC. In other words a differential signal could be delivered to the board, but the THS1209 ADC could still be configure to work in a single ended manner.

The delivered AED-109 actually had its front end Analog I/O inputs custom designed. The AI are designed to be ±10 Volts DC single ended with a differential output. The AO are designed to be ±1 Volts DC single ended. The AI's are passed through a two-stage amplifier circuit. The first stage consists of a inverter with a gain of approximately one-tenth, while the second stage is a inverter with a unity gain. The differential output of the AI is accomplished by having the output of the first stage split such that in one direction the unity inverter is bypassed and in the other it is not. The result is a differential output of the AI circuitry as shown below:

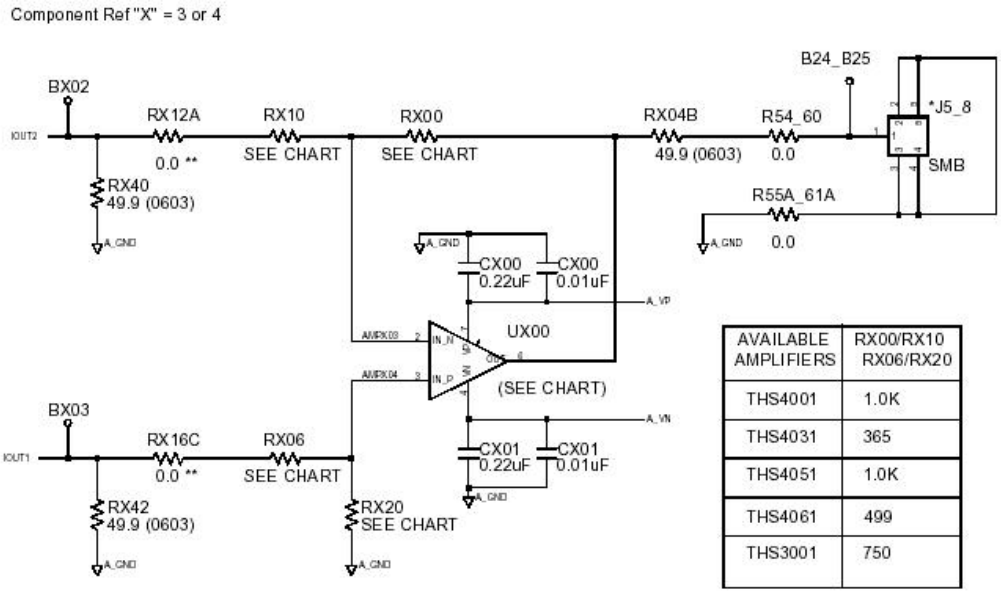


Custom analog front-end for the AED-109

The inputs to these circuits are the SMB connectors referenced J6 and J7
The input voltage range into these circuits is -10V to +10V

Figure 5-3 AED-109 Custom AI Front End

The AO circuit is shown below:



AVAILABLE AMPLIFIERS	RX00/RX10 RX06/RX20
THS4001	1.0K
THS4031	365
THS4051	1.0K
THS4061	499
THS3001	750

Figure 5-4 AED-109 Custom AO

5.6 FPGA [6]

The FGPA utilized can be a PQ or HQ 240 package, reference designator U1, or for smaller footprint applications, a PQ or HQ 208 package. For this implementation, an XCV50E-PQ240AFS0145 FPGA was chosen. The pin-out for this device is shown below:

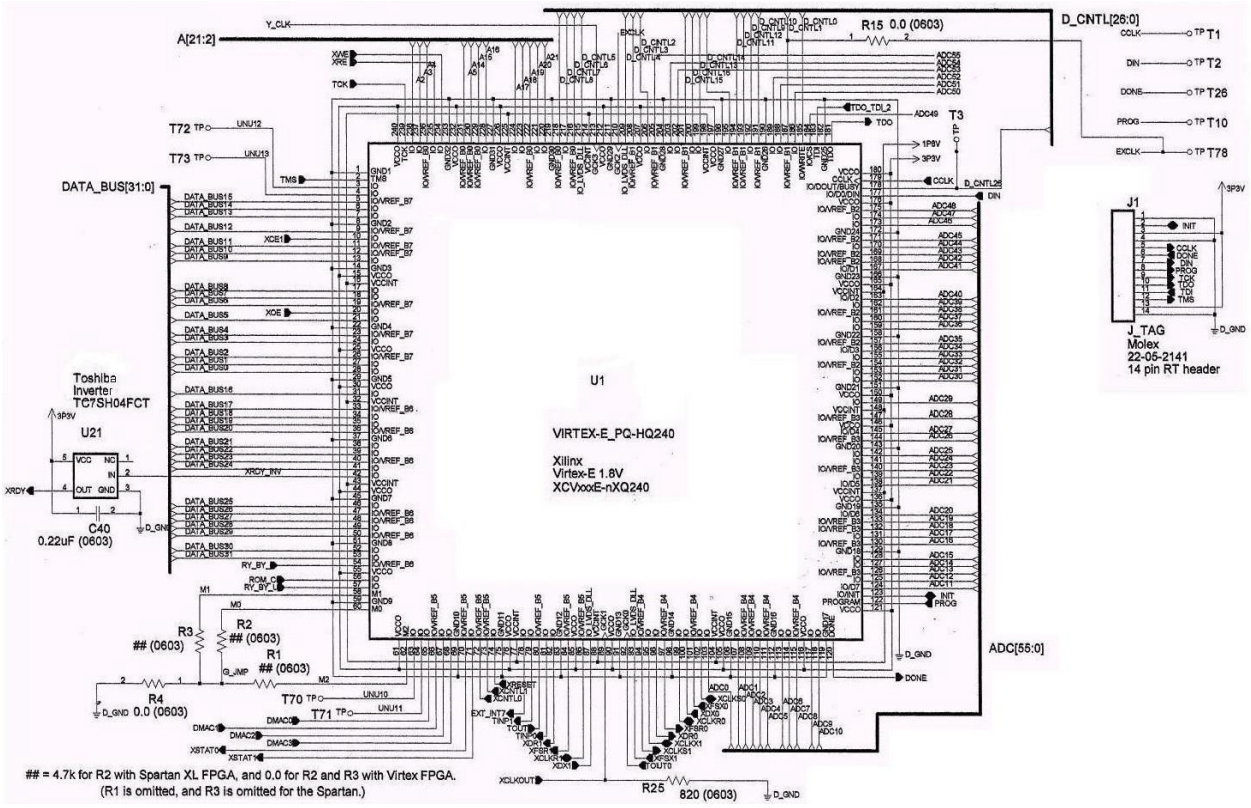


Figure 5-5 XCV50E-PQ240AFS0145 FPGA Pin-Out

The FPGA controls the interface between the DSK and Daughter card. As a result, all signals originating from the expansion connectors are terminated at the FPGA chip with the exception of the address and data lines driving the flash utilized as external memory for the DSK, U13 and or U14. Only the control signals originate from the FPGA to the flash, U13 and U14. In addition, the 16 digital I/O ports are also configured, read, and written to through the FPGA. Furthermore, the FPGA is also responsible for configuration and control of the 12-bit input to the DAC and the 12-bit output from the ADC.

5.6.1 FPGA Configuration [6]

The FPGA configuration is volatile (i.e. it is lost on power down); therefore, the configuration for the FPGA is loaded on power-up from the serial PROM, U2, or the flash, U9 and or U10. In this configuration, only U9 is populated. Flash by its design is non-volatile and is referred to as ROM. Alternatively, the FPGA can be configured through the Xilinx tool iMPACT, in boundary scan mode. The latter method is volatile since the flash or serial PROM is bypassed.

Tools for developing and downloading FPGA configurations are available from Xilinx. The Series Software tools offer a freeware introductory version, WebPack, and non-shareware more advanced software suites. WebPack can be used for basic configurations without FPGA options. The exact tools required for synthesis and implementations depend upon the configuration mode utilized. Since a PROM device was not populated, the procedure for programming will not be covered in this document. As a result, the Xilinx tools iMPACT and Prom File Formatter are sufficient to accomplish the task. Both of these tools are contained within the WebPack software package.

IMPACT is utilized to create the bit file, and the Prom File Formatter is used to create the mcs file from the bit file. The bit file is used to program the FPGA chip while the mcs file is used to program the flash. In order to implement these files, the JTAG Cable Model IJC-2 from Memec Design should be utilized to interface between the J1 port on the AED-109 and the parallel port on the PC containing the Xilinx tools. If a volatile implementation is desired, then only the FPGA should be programmed by using iMPACT in a boundary scan mode; however, if non-volatile solution is desired, the Prom File Formatter should be used to program the flash memory. If the boundary scan mode is chosen, a jumper must be placed across pins 1 and 2 of the J1 connector. This jumper must then be removed before the next restart. If this jumper is not removed, the FPGA will not be loaded from the flash upon power-on and will be un-configured.

5.6.2 FPGA Control Registers [4] [6]

The FPGA makes use of several memory-mapped registers in order to configure, control, and read/write to the various hardware elements on the AED-109. These registers are mapped into Data space on the DSK as defined within the Test program. The following table outlines the memory-mapped registers for the AED-109 used in conjunction with the C5416 DSK:

Table 5-5 AED-109 FPGA Memory-Mapped Registers

Register Name	Byte Addressing	16-Bit Word Addressing	Data Space Page #	Read/Write Capability
Digital I/O	01900000_{16}	640000_{16}	8_{10}	R/W
Digital I/O Control	01900004_{16}	640001_{16}	8_{10}	R/W
A/D and D/A Status	01920000_{16}	648000_{16}	9_{10}	R
Interrupt Start	01920004_{16}	648001_{16}	9_{10}	R/W
Interrupt Period	01920008_{16}	648002_{16}	9_{10}	R/W
Interrupt Down Counter	$0192000C_{16}$	648003_{16}	9_{10}	R
A/D Clock Rate	01920010_{16}	648004_{16}	9_{10}	R/W
A/D Clock Down Counter	01920014_{16}	648005_{16}	9_{10}	R
D/A Clock Rate	01920018_{16}	648006_{16}	9_{10}	R/W
D/A Clock Down Counter	$0192001C_{16}$	648007_{16}	9_{10}	R
A/D Clock Pulse Width	01920020_{16}	648008_{16}	9_{10}	R/W
A/D CR0	01920024_{16}	648009_{16}	9_{10}	R/W
A/D CR1	01920028_{16}	$64800A_{16}$	9_{10}	R/W
A/D Data	$01A00000_{16}$	680000_{16}	16_{10}	R
D/A Data	$01A00000_{16}$	680000_{16}	16_{10}	W

All of the FPGA memory mapped registers are 16-bits wide, but in the above table, two addressing schemes are shown. The C5416 DSK is capable of supporting 16-bit addressing for all memory spaces, but Data and I/O can also support 32-bit addressing as well. When using a 32-bit scheme, byte addressing is utilized which effectively makes two least significant bits always zero. For example, bits 24, 23, and 20 are high for the Digital I/O byte addressing. For 16-bit addresses, the 32-bit address is shifted 2 bits to the right. As a result, the same Digital I/O address is now expressed by having bits 22, 21, and 18 high.

The FPGA does not need to address more then 16 memory-mapped registers; therefore, bits 15 through 6 are shown as zeros for byte addresses, but are actually don't cares since they are not connected to the FPGA. Bits 14 through 4 are don't cares for 16-bit addressing for the same reason.

Remembering Section 3.1.1.4.5 MISC Register, it was outlined that Daughter Card memory accesses were permitted if the C5416 address line A_{15} is a logic 1, the DROM bit in the PMST register is zero, and the DM_SEL bit in the DM_CNTL register contained with the CPLD is set to 1. In addition, the AED-109 uses a 32-bit addressing scheme, which requires that the DC_WIDE bit within the CPLD MISC register be set to a logic 1. Furthermore, the DC_32ODD bit contained within the same MISC register must be set low if the supplied address is even and high if odd. Finally, the transfer of data will be successful if the proper logic sequence is followed as outlined in Section 3.1.1.4.5.

It is also worth revisiting the C5416 DSP address lines. This processor utilizes 16 external address lines, but A_{15} is only being set to signify to the processor that an off-chip memory access will take place. The DM_SEL bit contained within the CPLD register DM_CNTL signifies that a Daughter Card will being making this access provided this bit is a logic 1. As a result, the A_{15} bit is not utilized in the data memory map address. Therefore, concatenating the four DM_CNTL bits with the 15 remaining DSP address bits forms the complete memory-mapped address. The DM_CNTL bits contained within the CPLD DM_CNTL register are used to determine the correct page. An outline of this Data Space bit-addressing scheme is shown below:

				A15=1	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	DSP Address
DM_PG4	DM_PG3	DM_PG2	DM_PG1	DM_PG0																Page Address
M19	M18	M17	M16	M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0	Memory Address

A[15:0] is DSP Address
DM_PG[4:0] are located in DM_CTNL Register of CPLD at I/O Location 0x0005 bits 4-0. See section 2.2.4.1.5 that discusses the DM_CTNL register.
M[19:0] is memory Address

Figure 5-6 Data Space Bit Addressing

The linker command file used by Code Composer Studio is responsible for defining how the compiled executable code should be placed in Program space on page 0 and Data space on page 1; however, once this code is executed variable information such as the FPGA memory-mapped registers can be mapped outside the confines of the linker command file. See Chapter 6 for more details.

5.6.2.1 Digital I/O Register [6]

Each Digital I/O register bit corresponds to one digital I/O pin. The relationship is bit 0 corresponds to digital I/O 1, bit 1 corresponds to digital I/O 2, and so on. When the transceivers are set to output, the Digital I/O register is R/W, but when the transceiver is set to input, the Digital I/O register is read only. The layout of this register is shown below:

Digital I/O Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	I/O 15	I/O 14	I/O 13	I/O 12	I/O 11	I/O 10	I/O 9	I/O 8	O 7	I 6	O 5	I 4	O 3	I 2	O 1	I 0
Access	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
I/0 15-8	Digital I/O Bit Values: 0-low, 1-high
I0, I2, I4, I6	Digital I/O Bit Values: 0-low, 1-high
O1, O3, O5, O7	Digital I/O Bit Values: 0-low, 1-high

Figure 5-7 Digital I/O Register

5.6.2.2 Digital I/O Control Register [6]

The Digital I/O Control register is used to enable and control the buffers and transceivers. The OE bit enables or disables these devices. The D_n bits allow the Digital I/O on reference designator J15 to be configured as 12 DO's and 4 DI's or 12 DI's and 4 DO's as shown below:

Digital I/O Control Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	OE		D2	D1	SPARE											
Access																
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
O/E	Output Enable I/O 1-24: 0-enabled, 1-disabled
D2	Direction I/O 9-16: 0-input, 1-output
D1	Output Enable Outputs 1, 3, 5, 7: 0-input, 1-output

Figure 5-8 Digital I/O Control Register

The buffer that is utilized is the 74LVTH244 reference designator, U20, and the transceiver is a 74LVTH245 reference designator, U19.

5.6.2.3 D/A Data Register [6]

The D/A Data register is 32-bits wide in order to support 32-bit double word DMA transfers between the C5416 DSP and the AED-109 FPGA D/A FIFO. Once the Interrupt Down Counter register reaches zero, an interrupt is generated and the DMA controller commences to transfer data between the Data registers in the FPGA FIFOs and the C5416 independent of the CPU. Since this 32-bit double word mode is actually comprised of two separate 16-bit words where one occupies the upper 16-bits and the other the lower 16-bits, two consecutive 16-bit transfers are performed per single 32-bit double word. After each 16-bit transfer, the source and destination address are automatically incremented. The initial source address is contained within this D/A Data register. Once a block of data has been transferred to the FPGA D/A FIFO, the initial source address is reinitialized independent of the CPU.

The DMA controller uses the same memory address for the A/D Data register and the D/A Data register. Although these memory addresses are the same, they do not represent the same hardware. These transfers utilize the same bi-directional EMIF bus. As a result, only one function is enabled at a time to avoid conflicts. Furthermore, the DMA controller has six channels, but only 2 are available for off-board to on-chip transfers, which is sufficient for reading and writing in this application. When the block transfer is completed, the DMA channels autoinitialize in order to start fresh at the same physical memory address still contained within the D/A Data register. This register is shown below:

D/A Data Register

	MSB															LSB
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field	D 31	D 30	D 29	D 28	D 27	D 26	D 25	D 24	D 23	D 22	D 21	D 20	D 19	D 18	D 17	D 16
Acess	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	D 15	D 14	D 13	D 12	D 11	D 10	D 9	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Acess	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
D 0-11	Data Sample (12 bits) for device A
D 16-27	Data Sample (12 bits) for device B

Figure 5-9 D/A Data Register

5.6.2.4 D/A Clock Rate Register [6] [28] [29]

This register is used to establish the DAC sampling rate. This register contains the DAC EMIF clock rate divisor minus one. This register is used to load the D/A Clock Down Register once it has decremented to zero. Since the D/A Clock Down register is counting down to zero, the value held in the D/A Clock Rate Register is incremented by one to obtain the correct divisor and avoid an interrupt that is generated one EMIF clock cycle early. When establishing this rate, care should be taken to establish the appropriate ADC/DAC ratio. This ratio should always be an interval number of binary bits such that the ratio could be 1, 2, 4, 8, and so on up to a maximum interval number of 512. The reason for this maximum is that the FPGA FIFOs cannot hold more then 512 data samples. In this case, an DAC sampling rate of 1.953125 KHz was chosen. Realizing that the EMIF clock rate is 80 MHz and picking a divisor of 40960 in the Test Program allowed the DAC sampling rate to be calculated for the THS5661. The DAC rate cannot be set lower then the EMIF clock rate divided by 65535, which results in a minimum sampling rate of 1.22 KHz at EMIF equal to 80 MHz. The maximum DAC rate should be determined by experimentation by watching the U2 bit in the Status register. When a maximum DAC rate has been exceeded, an underflow of the FPGA D/A FIFO will occur an bit U2 will be a logic 1. However, the DAC sampling rate should never exceed the maximum ADC sampling

rate of 8 MHz established by the THS1209 data sheet. The D/A Clock Rate Register is shown below:

D/A Clock Rate Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	D 15	D 14	D 13	D 12	D 11	D 10	D 9	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Acess																
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
D 0-15	Value loaded into the DAC clock down counter DAC clock rate = CONV_CLK rate/(DAC Clock Rate Register + 1)

Figure 5-10 D/A Clock Rate Register

5.6.2.5 D/A Clock Down Counter Register [6] [28]

The D/A Clock Rate register initially loads the D/A Clock Down Counter register with the divisor it contains plus one when the XCNTL0 pin is low. This counter decrements every EMIF clock cycle. Since the decrementing of this register does not start until the XCNTL0 pin is raised high, the D/A FPGA FIFO should have its contents initialized in the Test Program in order to avoid an initial underflow of the D/A FIFO. Once the number of EMIF clock cycles equals the divisor number used to calculate the DAC clock rate, an interrupt is generated and the processed 16-bit control output is downloaded to THS5661 DAC from the FPGA D/A FIFO. At the same time, the D/A Clock Rate register reloads the D/A Clock Down Counter register and the process is repeated. The D/A Clock Down Counter register is shown below:

D/A Clock Down Counter Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	D 15	D 14	D 13	D 12	D 11	D 10	D 9	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Acess	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
D 0-15	Value read from the D/A down counter. A clock pulse is sent to the D/A converters when this count reaches zero. It is reloaded from the D/A Clock Rate Register.

Figure 5-11 D/A Clock Down Counter Register

5.6.2.6 A/D Data Register [6]

The A/D Data register utilizes the same process as outlined in Section 5.6.2.3.

A/D Data Register

	MSB															LSB
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field	D 31	D 30	D 29	D 28	D 27	D 26	D 25	D 24	D 23	D 22	D 21	D 20	D 19	D 18	D 17	D 16
Acess	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	D 15	D 14	D 13	D 12	D 11	D 10	D 9	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Acess	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
D 0-11	Data Sample (12 bits) for device A
D 16-27	Data Sample (12 bits) for device B

Figure 5-12 A/D Data Register

5.6.2.7 A/D Clock Rate Register [6] [29]

This register is used to establish the ADC sampling rate. This register contains the ADC EMIF clock rate divisor minus one. This register is used to load the A/D Clock Down Register once it has decremented to zero. Since the A/D Clock Down register is counting down to zero, the value held in the A/D Clock Rate Register is incremented by one to obtain the correct divisor and avoid an interrupt that is generated one EMIF clock cycle early. When establishing this rate, care should be taken to establish the appropriate ADC/DAC ratio. This ratio should always be an interval number of binary bits such that the ratio could be 1, 2, 4, 8, and so on up to a maximum interval number of 512. The reason for this maximum is that the FPGA FIFOs cannot hold more then 512 data samples. In this case, an ADC sampling rate of 500 KHz was chosen. Realizing that the EMIF clock rate is 80 MHz and picking a divisor of 160 in the Test Program allowed the ADC sampling rate to be calculated for the THS1209. The ADC rate cannot be set lower then 100 KHz based upon the TH1209 data sheet. The maximum ADC rate cannot be set higher then 8 MHz based upon the same requirement. The A/D Clock Rate Register is shown below:

A/D Clock Rate Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	D 15	D 14	D 13	D 12	D 11	D 10	D 9	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Acess																
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
D 0-15	Value loaded into the ADC clock down counter ADC clock rate = CONV_CLK rate/(ADC Clock Rate Register + 1)

Figure 5-13 A/D Clock Rate Register

5.6.2.8 A/D Clock Down Counter Register [6] [29]

The A/D Clock Rate register initially loads the A/D Clock Down Counter register with the divisor it contains plus one when the XCNTL0 pin is low. This counter decrements every EMIF clock cycle. Since DMA will not download the contents of the A/D FPGA FIFO until the Interrupt Down Counter register decrements to zero, the contents of this FIFO do not have to be initialized in the Test Program in order to avoid an initial underflow of the A/D FIFO. Once the number of EMIF clock cycles equals the divisor number used to calculate the ADC clock rate, an interrupt is generated and the contents of one word of the A/D FIFO is uploaded from the THS1209 ADC. At the same time, the A/D Clock Rate register reloads the A/D Clock Down

Counter register and the process is repeated. The A/D Clock Down Counter register is shown below:

A/D Clock Down Counter Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	D 15	D 14	D 13	D 12	D 11	D 10	D 9	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Acess	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	**Definition**
D 0-15	Value read from the A/D down counter. A clock pulse is sent to the A/D converters when this count reaches zero. It is reloaded from the A/D Clock Rate Register.

Figure 5-14 A/D Clock Down Counter Register

5.6.2.9 A/D Clock Pulse Width Register [6] [29]

When the A/D Clock Down Counter register has decremented to zero, the clock pulse that is sent to the THS1209 ADC has a duration equal to a multiple of the EMIF or Daughter Card clock period, which effectively produces a variable duty cycle for the ADC. The A/D Clock Pulse Width register configures this value. In the appl_parms function within the Test Program, the pulse width is configured to be 5 times the EMIF clock period or 62.5 ns. The A/D Clock Pulse Width register is shown below:

A/D Clock Pulse Width Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	D 15	D 14	D 13	D 12	D 11	D 10	D 9	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Acess																
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
D 0-15	Conversion clock pulse width in base clock cycles. Minimum - specified minimum clock pulse width in data sheet Maximum - half of the Clock Rate Register

Figure 5-15 A/D Clock Pulse Width Register

5.6.2.10 A/D Control CR_0 Register [6] [29]

The TH1209 contains two analog input channels, which can be configured as two single ended inputs or one differential input. Since there are two THS1209 ADC's on the AED-109, this board can be configured for two differential or two single ended 12-bit inputs to the FPGA since the THS1209 can only quantitize a single 12-bit output regardless if it is single or differential in nature. The external interface for these ports are the J6 and J7 SMB connectors or the Breadboard area. In order to utilize the THS1209 in a single differential channel mode, the Control Register 0 must be configured on the THS1209. This configuration is accomplished through the A/D Control CR_0 register as shown below in the following table and figure:

Table 5-6 THS1209 Control Register 0 Bit Functions

Bit	Reset Value	Name	Function
0	0	VREF	V_{ref} select: 0 => The internal reference is selected. 1 => The external reference voltage is selected.
1	0	RES	Reserved
2	0	PD	Power Down: 0 => The ADC is active. 1 => The ADC is powered down.
3 4 5	0 0 1	CHSEL0 CHSEL1 DIFF0	Channel Select for bits 3-5: 000 => Analog input AINP (single ended) 001 => Analog input AINM (single ended) 010 => Reserved 011 => Reserved

Table 5-6 THS1209 Control Register 0 Bit Functions (Continued)

Bit	Reset Value	Name	Function
			100 => Differential channel (AINP-AINM) 101 => Reserved 110 => Not Used 111 => Not Used
6	0	DIFF1	Reserved
7	0	SCAN	Autoscan enable
8 9	0 0	TEST0 TEST1	Test Mode: 00 => Normal Mode 01 => V_{refp} 10 => $(V_{refm} + V_{refp})/2$ 11 => V_{refm}
10	0	N/A	Bit is always zero.
11	0	N/A	Bit is always zero.

A/D Control CR_0 Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	WC	Spare			D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Acess																
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
D 0-11	Device Control Register for THS1209
WC	Write Control: 1 = write new value, 0 = reset register (Register must be reset by CPU before a new write is performed.)

Figure 5-16 A/D Control CR_0 Register

Based upon the THS1209 data sheet, bit 5 should be asserted, while all other bits 11 through 0 should be a logic low in order to achieve a single channel differential mode. In addition, the THS1209 only has one 12-bit configuration port, which results in the bits 11 and 12 being used to identify if the Control Register 1 or 0 is being configured. The bit pattern 00 refers to Control Register 0, while 10 refers to Control Register 1. In addition, bit 15 should be asserted in order to allow the register to be initially configured. Only bits 0 through 11 are actually written to the TH1209 Control Register 0. The appropriate hexadecimal value of 8020_{16} is established in the appl_parms function within the Test Program. For further details, consult the THS1209 data sheet.

5.6.2.11 A/D Control CR_1 Register [6] [29]

The THS1209 ADC utilizes one other register called the Control Register 1 to properly configure this device. This register is responsible for resetting the device, establishing the appropriate read/write logic levels, selecting 2's complement or binary bit format, differential offset error reduction, and debugging capabilities. Bit 2, 3, 4, 5, and 11 are always zero, while bit 10 is always 1. In addition, the THS1209 only has one 12-bit configuration port, which results in the bits 11 and 12 being used to identify if the Control Register 1 or 0 is being configured. The bit pattern 00 refers to Control Register 0, while 10 refers to Control Register 1. In this configuration, a R/W', binary, no offset correction, and no debugging configuration was selected by choosing a hexadecimal value of $84C0_{16}$ in the appl_parms function within the Test Program. The following table outlines the appropriate configuration for the A/D Control CR_1 register:

Table 5-7 THS1209 Control Register 1 Bit Functions

Bit	Reset Value	Name	Function
0	0	RESET	Reset: Writing a 1 into this bit resets the device and sets the CR0 and CR1 to the reset Values. To bring out of a reset, write 0 to this bit.
1	0	SRST	Writing a 1 into this bit resets the sync generator. When running in multichannel mode, this must be set during the configuration cycle.
2	0	RES	Always write 0.
3	0	RES	Always write 0.
4	1	RES	Always write 0.
5	1	RES	Always write 0.
6	0	R/W'	R/W', RD/(WR)' selection: Bit 6 of CR1 controls the function of the inputs (RD)' and (WR)'. When bit 6 in CR1 is set to 1, (WR)' becomes a R/W' input and (RD)' is disabled. From now on a read is signaled with R/W high and a write with R/W as a low signal. If bit 6 in CR 1 is set to 0, the input (RD)' becomes a read input and the input (WR)' becomes a write input.
7	0	BIN/2s	Complement Select: If bit 7 of CR 1 is set to 0, the output value of the ADC is in twos complement. If bit 7 of CR 1 is set to 1, the output value of ADC is in binary format.
8	0		Offset Cancellation Mode: 0 => normal conversion mode 1 => enable calibration mode If a 1 is written into bit 8 of CR1, the device internally sets the inputs to zero and does a conversion. The conversion result is stored in an offset register and

Table 5-7 THS1209 Control Register 1 Bit Functions (Continued)

Bit	Reset Value	Name	Function
			subtracted from all conversions in order to reduce the offset error.
9	0		Debug Mode 0 => normal conversion mode 1 => enable debug mode When bit 9 of CR1 is set to 1, debug mode is enabled. In this mode, the contents of the CR0 and CR1 can be read back. The first read after bit 9 is set to 1 contains the value of the CR0. The second read after bit 9 is set to 1 contains the value of CR1. To bring the device back into normal conversion mode, this bit has to be set back to 0 by writing again to the CR1.
10	1	N/A	Bit always 1.
11	0	N/A	Bit always 0.

The A/D Control CR_1 Register is shown below:

A/D Control CR_1 Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	WC	Spare			D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Acess																
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
D 0-11	Device Control Register for THS1209
WC	Write Control: 1 = write new value, 0 = reset register (Register must be reset by CPU before a new write is performed.)

Figure 5-17 A/D Control CR_1 Register

5.6.2.12 Interrupt Down Counter Register [6]

The Interrupt Start Register initially loads the Interrupt Down Counter register when the XCNTL0 pin is low. When the XCNTL1 pin goes high, the counter begins to decrement every ADC clock cycle. The Test Program has been configured to generate an interrupt every 256 A/D clock cycles. This interrupt is routed through the pin 53 on the Expansion Peripheral Interface, J10, to the DSK. Since the Test Program has configured the ADC at 500 KHz and the DMA

transfer rate is 1.953125 KHz, which is the same as the DAC, an underflow condition will not occur on the A/D FPGA FIFO. If the DMA transfer rate was set slower then the DAC rate, an underflow would occur on the D/A FIFO. As a result, a DMA data transfer of the A/D FPGA FIFO and D/A FPGA FIFO proceeds on the rising edge of the interrupt line generated by the Interrupt Down Counter register. The DMA controller determines the order of the A/D and D/A data transfer. The appl_process function within the Test Program then sums the A/D data and averages based upon the ADC/DAC ratio. At this same rising edge, the Interrupt Period register reloads the D/A Clock Down Counter Register now that the XCNTL0 pin is high and the process is repeated. The Interrupt Down Counter register is shown below:

Interrupt Down Counter Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	D 15	D 14	D 13	D 12	D 11	D 10	D 9	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Acess	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
D 0-15	Value of the interrupt down counter. This register is loaded with the contents of the Interrupt Start Register while the XCNTL0 pin is low. The counter counts down while the XCNTL0 pin is high. An interrupt is sent when this counter reaches zero. Then it is reloaded with the contents of the Period Register. It continues to count down while the XCNTL0 pin is high.

Figure 5-18 Interrupt Down Counter Register

5.6.2.13 Interrupt Start Register [6]

The Interrupt Start register is used to load the Interrupt Down Counter and Interrupt Period registers when the XCNTL0 line is low. The Interrupt Start register is shown below:

A/D and D/A Status Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	SPARE								SPARE						U1	O1
Acess	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
O1	ADC FIFO Overflow
U1	ADC FIFO Underflow
O2	DAC FIFO Overflow
U2	DAC FIFO Underflow

All status register bits are latched and held while XCNTL0 pin is high. They are all cleared to 0 when the XCNTL0 pin goes low.

Figure 5-19 Interrupt Start Register

The Interrupt Start Register value is user defined in the Test Program at 256. When the XCNTL0 line is high, this register is irrelevant since the Interrupt Period register loads the Interrupt Down Counter.

5.6.2.14 Interrupt Period Register [6]

The Interrupt Period register is used to reload the Interrupt Down Counter register every time it decrements to zero when the XCNTL1 line is high. This register is loaded based upon the contents of the Interrupt Start register when the XCNTL0 line is low. The Interrupt Period Register is shown below:

Interrupt Period Register

	MSB															LSB
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	D 15	D 14	D 13	D 12	D 11	D 10	D 9	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Acess																
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
D 0-15	Value loaded into interrupt down counter after each interrupt is sent. (Length of time in EMIF clock cycles before the next interrupt is sent.)

Figure 5-20 Interrupt Period Register

5.6.2.15 A/D and D/A Status Register [6]

The A/D and D/A status register allows the user to identify if an overflow or underflow condition has occurred on the either the A/D or D/A FPGA FIFO's by asserting the appropriate bit. An A/D FPGA FIFO underflow occurs when the DMA attempts to read data from this FIFO and it is empty, while a D/A FPGA FIFO underflow occurs when the DAC sampling rate is set to fast such that the DMA cannot load the subject FIFO fast enough. An A/D FPGA FIFO overflow occurs when the ADC clock rate is set faster then the DMA can download the data out of the its FIFO. Conversely, a D/A FPGA FIFO overflow occurs when the DMA attempts to write into the D/A FIFO and data is still present. The Test Program avoids the overflow or underflow problems on the D/A FPGA FIFO by filling this FIFO half full (i.e. 256) with duplicate 16-bit data words. When this set of data is sent, a pulse is sent to the DSK through J10 on the TINP1 line, which in turn drives a counter. When the counter reaches half the FIFO size another group of words are sent to the D/A FIFO and the process is repeated. The A/D and D/A Status register is shown below:

A/D and D/A Status Register

	MSB														LSB	
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	SPARE								SPARE						U1	O1
Acess	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Definition
O1	ADC FIFO Overflow
U1	ADC FIFO Underflow
O2	DAC FIFO Overflow
U2	DAC FIFO Underflow

All status register bits are latched and held while XCNTL0 pin is high. They are all cleared to 0 when the XCNTL0 pin goes low.

Figure 5-21 Status Register

5.7 Amplifiers [6] [30] [31]

The AED-109 uses the THS4062, U100 and U200, operational amplifiers to drive the inputs of the A/D converters, and the THS3001, U30 and U31, operational amplifiers to buffer the outputs of the D/A converters. The gain of these devices is set to one with their sole purpose being to sink or sufficient current as well providing a degree of electrical isolation from the outside world. The input range to the THS4062 is ±1 Volt, while the output range of the THS3001 is again ±1 Volt. A loading resistor is placed across the input ADC amplifier to reference the input to ground; however, because the inverting amplifiers source some current, a bias voltage is developed across the input resistor, which is evident in the A/D readings when there is no input source. This was verified by observing that the efflux plasma charge sensor measured 0.654386 Volts before the keyhole in the work piece was established. Subtracting this bias voltage from the measured readings alleviated this problem. For sources that do not have an open-circuit connection, this problem does not exist.

5.8 Breadboard Area [6]

The AED-109 is furnished with a Breadboard to allow for the building of custom analog circuitry. This area is on the far left side when looking at the top of the AED-109. Adjoining the Breadboard area are input and output amplifiers for the converters. The Breadboard area was designed for using both through-hole and 50 mil surface-mount components. In addition, regulated ±9 Volts is available throughout the Breadboard area. It is the author's opinion that utilizing this area is a bad practice due to the ability to damage the board, which removes

Signalware's liability. If custom analog components are desired, the author recommends outlining the requirements and having Signalware place these components.

5.9 Boot Flash [6]

The AED-109 has the ability to be upgraded with two additional flash memory chips part number, AM29F400B, reference designators U13 and U14. U14 is located on the topside and U13 is located directly beneath on the backside of the AED-109 adjacent to the JTAG J1 connector. The intention of this additional flash is to be able to boot load the C5416 DSP on start up. Each chip contains 16-bit 256K words. This option was not purchased.

5.10 Reference Voltage Supplies [6]

Three reference supplies are provided: 4.096 Volts; 2.048 Volts; and 1.024 Volts. The 2.048 V and 1.024 V unregulated supplies rated at a static 0.1 mA are adjusted with potentiometers R20 and R24 located on the top-right backside of the board. The 4.096 V supply rated at 20 mA is regulated with a reference diode. If the loads are dynamically varying, a buffer amplifier to source or sink current should be utilized. These supplies can be easily overloaded and thus damaged; therefore, the author does not recommend utilizing due to liability concerns.

5.11 DAC Reference Currents [6]

The reference currents for the D/A converters are generated internally by the THS5611. These currents may be scaled by varying the potentiometers R327 and R332 for the two D/A converters, U7 an U8. The effective gain of the A/D converters are determined by the current references and the output resistors R340/R342 and R440/R442. It is the author's recommendations that these values are factory set and should not be adjusted by anyone other then Signalware due to liability concerns.

5.12 Digital Buffers [6]

The AED-109 has been supplied with two octal transceivers reference designators, U19 and U20. U19 buffers I/O numbers 9-16, and U20 buffers I/O numbers 1-8. The part number for U19 is SN74LVTH245ADW, and the part number for U20 is SN74LVTH245ADW. These buffers can support update rates up to 100 MHz; however, the EMIF clock is only 80 MHZ, the DMA can't support this rate, and the ADC maximum sampling rate is 8 MHz, which places these transceivers well within their operational limits.

Chapter Six

Embedded Programming

6.1 Test Program [6]

Each Daughter Card supplied by Signalware is delivered with a Test Program for a particular targeted development board. In this case, the targeted board was the TMS320VC5416 DSK. Because this program is intended primarily as a hardware test, it does not use the DSP/BIOS or chip support library established in Code Composer Studio (CCS). Signalware has chosen this route because these CCS tools tend to cloak the hardware operation and make hardware troubleshooting difficult. Support for CCS tools is not included with the AED-109 warranty technical support provided by Signalware.

The Test Program is a data acquisition and signal generation program written in ANSI C, which can be easily modified for such applications as data collection or feedback control as in this case. The expansion memory interface, J9, using the DMA controller is utilized to transfer A/D and D/A data from the AED-109 to the C5416 DSP. The advantage of this method is that the DMA controller handles this transfer independent of the CPU except when a block of data is ready and an interrupt is generated.

A majority of the code associated with the Test Program concerns initializing the hardware and printing the results. This is very useful when developing an application since the initialization and output framework have already been provided. Minor modifications to this methodology can be sufficient for a significant number of applications.

The basic operation of the Test Program is divided into three functions: A/D; Digital I/O; and D/A. In the A/D step, the DMA registers are configured, starts the DMA controller, starts the FPGA, and waits for interrupts indicating that a frame of data is ready. This process is continued for several frames before stopping at which time test results are printed to the screen. The Digital I/O is then turned on one at a time for a fraction of a second starting with pin 1 on J15. An OK is then printed to the screen. The final step involves printing a triangular waveform of ±0.5 Volts that can be observed on either J5 or J8 with an oscilloscope or as a varying voltage level with a digital multimeter.

The Test Code architecture divides the functions into four module types which are: AED_MAIN; AED_DMS_xxx; AED_xxx; and xxx_BRD. The AED_MAIN module is a general-purpose main program used to transfer between the C5416 and the AED-109. The AED_DMS_xxx modules are used to set up the DMA registers and starting of the DMA controller for the purpose of transferring data between the A/D and D/A converters to the C5416 DSP. The AED_xxx is application dependent whose purpose is to configure, initialize, process, and terminate the Test Program. The xxx_BRD are development board dependent functions, which support the previously listed three modules.

Since the AED-109 is being used in a differential mode in conjunction with a TMS320VC5416 DSK, the appropriate application dependent module is the AED_109_32d.

This module is composed of five basic function modules which are: appl_parm, appl_init, appl_process, appl_idle, and appl_end. The appl_parm function configures the FPGA memory mapped registers. The appl_init function initializes variables and pre-loads the FPGA D/A FIFO to avoid an initial underflow. The appl_idle function is simply a loop that is performed while the main program waits for an interrupt to be generated signaling that a new block of A/D data is available in the A/D FPGA FIFO. In the mean time, the main program simply calls the appl_idle function instead of the appl_process function. This is done to avoid an underflow condition associated with the A/D FPGA FIFO. The appl_process function processes the data that has been downloaded from the A/D FPGA FIFO by the DMA controller, controls the D/A output, and terminates the Test Program by returning a value other then 0. The last function is the appl_end, which is responsible for printing to screen or file the results of the execution. A complete list of files required to allow the test program to function is shown in Appendix B.

6.2 Test Program Modification [6]

In order to implement the Non-Linear Interval Control Algorithm, the following files were modified: AED.h; AED_109_32d.c; AED_Appl.h; AED_Cfg.h; and AED_MAIN.c. The AED.h file simply changed the statement, #define AED_PRINT 1, to the following, #define AED_PRINT 2. In this header file, 1 is the default value, which causes the appl_end function to print to the screen rather then to a file as altered.

The AED_Appl.h file was modified by changing the data types in the appl_end function call for the parameters bufs_proc, buf_count, and prev_buf_count from int to unsigned int. This was done to increase maximum count from 32767 to 65535 for the subject parameters, which are responsible for counting the number of blocks of data that are downloaded from the A/D FPGA FIFO and the actual number of blocks that have been processed by the appl_process function. A deviation can occur if the appl_process function takes longer then the DMA download rate in which case an overflow has occurred. These comparisons are printed to the screen from the appl_end at program termination.

The AED_Cfg.h file was modified by simply removing the comment out statements surrounding the preprocessor token replacement define statement for the variable CHECK_FPGA_OVFL. This variable is now established as 0, which prevents the AED.h file from establishing the same variable as 1, which in turn will establish the FPGA_OVFL_CHECK_ENABLE as 0, which effectively disables overflow checking. This is beneficial since the Test Program will terminate if an overflow condition occurs which is likely to happen if the appl_process function takes longer then the DMA download rate.

In the AED_MAIN program the blocks variable was modified to always be 1. Previously, this value was being set to the frames value of 3. This result in turn was inducing an error by preventing the member byte array of the structure dual_data_buffer from assigning its 0 element to the member byte of the structure address in the AED_MAIN program. If this error is not corrected, the initial memory address of A/D DMA downloaded data will have an offset error when presented to the appl_process function. The frames variable represents the number of frames per block and is defined within the AED_109_32d module. This variable has multiple functions and should not be modified from its present logic state other then to prevent the

equating to the number of blocks in the main program. In addition, the data types for the variables bufs_proc and prev_buf_count were changed from int to unsigned int in the AED_MAIN program in order to increase the number of blocks of data that can be tracked when comparing the number of blocks of data that have been transferred from the A/D FPGA FIFO to the C5416 versus the number of blocks that have actually been processed by the appl_process function.

6.2.1 Printing [6]

As previously stated, the AED_109_32d is the appropriate module to utilize when configuring the THS1209 ADAC to accept a differential input. This module is divided into five functions: appl_parm; appl_init; appl_process; appl_idle; and appl_end. Modifications were required in all functions except the appl_idle. Furthermore, it is important to note troubleshooting code is commonplace particularly when writing initial versions or performing modifications of existing unfamiliar code. As a result, it is often a common practice to track problems by performing printouts of information at various levels of implementation. This is practice is acceptable provided one rule is followed. No printing is allowed from the appl_process or appl_idle functions. The reason being that these are real time applications, which can be adversely affected by operations that take excessively long periods of time such as the case when printing. If this advice is ignored, a printout will be observed, but observation will be more a function of the induced timing error associated with printing then any underlying problem.

6.2.2 Clock Rates [6] [28] [29]

The first requirement is the appropriate adjustment of the A/D and D/A Clock Rate registers. This is accomplished by considering the ADC/DAC ratio, DMA transfer rate, FPGA FIFO sizes, and appl_process function processing time simultaneously. The ADC THS1209 has a configurable sampling rate between 100 KHz and 8 MHz, while the DAC THS5661 DAC sampling frequency has a minimum value of the EMIF clock rate divided by 2^{16}. In this case, the EMIF clock rate is 80 MHz resulting in a minimum DAC clock rate of 1.22 KHz. The maximum DAC update rate is more complicated to establish. By itself, the THS5661 has a maximum clock rate 100 MHz, but realistically should never exceed the ADC sampling frequency divided by two. In addition, the DAC clock rate should not exceed the DMA transfer rate or an underflow will occur. The sizes of the D/A and A/D FPGA FIFO's are the equal and fixed at 512 16-bit words, which limits the maximum ADC/DAC ratio to 512. The speed of the appl_process function is dependent upon its complexity, mathematical requirements, and algorithm architecture. In the case of the appl_process, the best advice is simplicity and avoidance of division whenever possible.

With the aforementioned information in mind, the best starting point is with the DMA transfer rate. This rate is established by choosing a divisor of the ADC sampling rate and storing this information in the Interrupt Start register within the appl_parm function. The best rate is one that will avoid overflow and underflow problems associated with the A/D and D/A FPGA FIFO's, allow variance in the appl_process processing time, and maximize the data transfer rate. As a result, the best recommendation would be to choose a divisor of 256 that is half the size of

the FIFO's. This divisor will allow the transfer of A/D and D/A data between the AED-109 and the C5416 to commence when the FIFO's are halfway utilized, but at the same time allow for some variance in the appl_process processing time to help avoid overflow and underflow situations. Based on this divisor, the DMA transfer rate is established at 1.953125 KHz.

In order to keep the sampling rates relatively slow relative to the AED-109, but above the minimum specified ranges, an ADC clock rate was chosen at 500 KHz. This rate is established by choosing the appropriate divisor of the EMIF clock, which is stored in the A/D Clock Rate register in the appl_parm function. The next decision was to choose an interval base two ADC/DAC ratio of 256, by assigning the appropriate EMIF clock divisor for the D/A Clock register with a global preprocessor token replacement define statement. This choice allows for some variance in the appl_process function processing time in order to avoid overflow or underflow problems as well. The net result was a DAC sampling frequency, which equals the DMA transfer rate.

6.2.3 Test and Platform Code Removal [6]

The next step involved stripping the AED_109_32d module of code associated with testing, different TI platforms, and unnecessary commenting. Since this Test Program was originally written with the intention of applicability for various TI platforms, numerous conditional compilation if statements are used throughout the algorithm. The net effect is to mask the compiled end result. Therefore, it is easier to remove the conditional compilation statements and only retain the portion of code that is applicable to the TMS320VC5416 DSK for clarity and ease in troubleshooting. Code associated with generating a triangular waveform for the DAC, ADC record skipping, test data initialization/generation/storage/printing, Digital I/O, and iteration tracking was removed due to irrelevancy to embedded control algorithm as well.

6.2.4 Algorithm Reconfiguration [6]

The original intention of the AED_109_32d code was to set register values in the appl_parm function, initialize variables in the appl_init function, and observe the ADC operation in the appl_process function for a fixed number of iterations before returning a termination value. Based upon this termination response, the appl_end function would be called and proceed with an initial check for error conditions followed by a printout of the ADC results, manipulation of the Digital I/O, and then generation of a triangle wave to the DAC. However, this procedure is not acceptable for a feedback control algorithm.

For a feedback control algorithm, the original intent of each function would be kept, but procedures that would be performed in each function would be changed. Although not used in this application, the Digital I/O could be moved to the appl_init function. The intent of the Digital input would be to program the control algorithm to some preset value such as different base times or reference peak times in relation to the Non-Linear Interval Control Algorithm, while the Digital outputs could be used to drive relays for a variety of purposes. However, of greatest importance, the ability to read the system output from the ADC and output a control signal through the DAC should be in the same real-time function, which in this case is referred to as the appl_process. The appl_end function would be used solely for testing for error conditions

and outputting results. Alternatively, the outputting of information could even be possible in the real-time process. This could be achieved by allowing a host to observe memory locations through the HPI Expansion Connector, P3, on the C5416 DSK that would contain system output and control signal data. Although very efficient, this degree of complexity was not pursued in this project. Instead, it was chosen to simply store the control signal, next peak iteration current level, peak current time measurement, keyhole potential, peak delay, and k^{th} time measurement in an array format and save the results to an ASCII file on the host computer using the USB port in the appl_end function. The appl_init function main change would be that it would be used to initialize the control output to zero and avoid an underflow situation with the D/A FPGA FIFO upon startup. Various other variables associated with the control algorithm and saving data would be added to the appl_init file as well. The appl_idle would remain unchanged.

6.2.5 Globals for Diagnostic Termination [6]

As has been previously stated, the globals for diagnostic termination are used to monitor the number of blocks of data that have been transferred from the A/D FPGA FIFO to the C5416 versus the number of blocks that have actually been processed by the appl_process function. Within the original AED_109_32d module, these variables were only declared as an int variable, which in CCS is only 8-bits in length. As a result, only 256 different iterations could be tracked. Since the control algorithm would iterate to a much greater degree, an unsigned int variable type was chosen instead, which is 16-bits in length. This C5416 overflow tracking is a useful tool when trying to measure the relative speed of the control algorithm-processing rate in relation to the ADC sampling rate.

6.2.6 Base Ten Conversion [6] [11] [28] [29]

The next step was determining how to properly decode the A/D quantization and encode the quantization of the D/A. Both the A/D THS1209 and D/A THS5661 are 12-bit devices; however, the THS5661 can be delivered in 14, 12, 10, and 8-bit configurations. As a result, Signalware chooses to utilize both the 14-bit and 12-bit D/A converters in its various Daughter card designs. In order to provide the maximum cross compatibility in its FPGA code, Signalware has chosen to utilize a 14-bit software resolution in its Test Program regardless of the actual hardware being utilized, which has been configured into the requirements of its write32b_reg function. Therefore, from the appl_process perspective, the D/A quantization is 14-bits. Within the FPGA firmware, the D/A quantitized value is reconfigured to the appropriate 12-bit value by simply shifting the value 2-bits to the right. Since the THS1209 is only delivered in a 12-bit version, the quantitized value delivered by the A/D FPGA FIFO is also 12-bits. However, in order to make the code less confusing, it was chosen to multiply this A/D value by 4 to allow the control algorithm to work in just one fictitious 14-bit quantization bit pattern.

It was now necessary to determine how to convert the integer base 2 quantization to a base ten floating-point number and vice versa. The AED-109 custom analog input ports accept ±10 Volts, which are immediately passed through a one-tenth gain inverting op-amp. The signal is then split with one path passing through a unity gain inverting op-amp and other bypassing. As a result, the THS1209 is presented with an effective ±1 Volt differential signal. Since we are not using a 2's complement encoding the, the all bits zero pattern will represent –1 Volts, and the

all bits 1 will represent +1 Volts. In order to achieve the true analog input value, the quantitized value must be first multiplied by the quantization interval, then shifted down by 1, multiplied by ten, and finally modified by having the offset bias removed by subtraction due to an open circuit connection. The D/A quantization equation is shown below:

$$AI_v = \left(Quantization_{ADC} \cdot \frac{2}{2^{14}} - 1 \right) \cdot 10 - Bias \qquad 6-1$$

The AED-109 analog output ports work in the reverse manner with the following exceptions: the AO's are ±1 Volts instead of ±10 Volts; no Bias offset is present; and adjustments for the Miller Electric Maxtron 450 CC/CV power supply must be made. The first two exceptions are self-evident, but power supply should be outlined in detail. Supplying a 0 to 10 Volt analog signal to pin G on port 17 controls the output of the Maxtron power supply. This control signal corresponds to a 1 Volt equals 55 Amperes conversion ratio. Since the DSK is only capable of supplying a ±1 Volts analog output, the Maxtron current output would not exceed 55 Amperes if this problem was not addressed. As a result, an external single stage-inverting amplifier was required; therefore, the floating-point base 10 control signal is multiplied by the conversion ratio and the effective inverse gain of the amplifier. The gain was configured at about –9.24. The A/D quantization equation is shown below:

$$Quantization_{DAC} = \left(AO_A \cdot \frac{1V}{55\,A} \cdot \frac{-1}{9.24} + 1 \right) \cdot \frac{2^{14}}{2} \qquad 6-2$$

It should be noted that since equations 6-1, 6-2, and the control algorithm utilize mathematical operations the header file math has been added to the AED_109_32d module.

6.2.7 NONLinearInterval_delay_quicker [6]

The NONLinearInterval_delay_quicker module is the modified version of the AED_109_32d module provided by Signalware, which contains the actual Non-Linear Interval Control algorithm. The module begins by utilizing the include directive for several necessary header files as follows: stdlib; dsk5416; emif; AED; AED_DMS; AED_Appl; stdio; and math. The include directives are followed by a list of preprocessor token replacement define statements, which define the subject variables as global to the subject module. These directives are divided into two groups where one group defines the data transfer methodology and the other defines the FPGA memory registers.

6.2.7.1 Data Transfer Variables [6]

The data transfer methodology is divided into several global variables listed as follows: AED_BOARD; DMS_MODE; DIVIDE_POWER; NO_RECORDS; NO_FRAMES; RECLEN; ELEMENTSIZE_CODE; SAMPLES_PER_WORD; and DAC_CLK_CNT. The AED_BOARD module is simply a character string used to define the subject Signalware Daughter Card being used. The DMS_Mode defines the data memory service to work in a frame synchronous continuous mode. The DIVIDE_POWER defines the base two exponent value for the

ADC/DAC clock ratio. The NO_RECORDS is the actual ADC/DAC clock ratio, or in other words, the number of A/D data samples that will be averaged to constitute one data value. The NO_FRAMES defines the number of frames per block. The RECLEN variable defines the number of words per record. The ELEMENTSIZE_CODE describes the number of bits per word as 32. The SAMPLES_PER_WORD defines the number of data samples per 32-bit word as two. The DAC_CLK_CNT defines the D/A EMIF clock divisor minus one.

6.2.7.2 FPGA Memory Register Declarations [6]

The AED-109 FPGA memory registers are defined next. These 32-bit addresses are actually only used to define the lower 15 bits, or in other words, the appropriate 16-bit word on an undefined page. The upper five bits used to define that actual Data space page address are defined in the appl_parms function by performing a bitwise OR command.

6.2.7.3 Global Declarations [6] [28]

The next step involved assigning global data types. These definitions are divided into two groups. The first group is used as termination diagnostics, which are passed as parameters in the functions within this module back to the AED_MAIN program. Since this diagnostic variables are being passed back to the AED_MAIN program, these variables do not need to retain their values once they are returned. The AED_MAIN will maintain their values in between function calls. The other variables need to be global, but also retain their values within this module upon function return. As a result, the static variable definition is utilized.

The unsigned long variable A_value is used as the average value for the long variable A/D sum variable sumA. The sumA variable is the sum of all A/D samples. Since the AED-109 A/D converter has been configured to sample 256 times before downloading, the sumA variable represents the sum of all of these measurements. This value is then shifted left based upon the DIVIDE_POWER value equaling to eight in this application. At this point, the A_value is a base ten representation of a base 2 12-bit quantitized value of the analog input.

The unsigned long variable output is the used as the quantization variable for the D/A function write_32b_reg function. This value a base ten representation of a base 2 fictitious 14-bit quantization for the THS5661. It is described as fictitious because the THS5661 being utilized is actually 12-bits. The data is being shifted two bits to the right in the FPGA firmware.

The int variables input_cout and loop_count are not being utilized. The loop_count variable is tracking the number of iterations that they appl_idle loop iterates; however, nothing is done with this information.

Two pointers variables are defined globally for this module. They are cntl_base_addr and data_base_addr. These pointers are pointing to the memory addresses associated with these variables. The hexadecimal values used to assign Data memory Page addressing are assigned to these memory locations in the appl_parms functions. The cntl_base_addr is used to define the Data memory page for the FPGA memory-mapped registers, while the data_base_addr is used to define the page address associated with DMA writes to the THS5661.

The unsigned long variable fpga_io_reg is actually a method of assigning the bit values to the Digital I/O LSB_DIO_REG and the MSB_DIO_REG registers in one line. This is done with the preprocessor token replacement define statement where a single 32-bit value is listed. In the appl_parms function, the LSB_DIO_REG, which is really the Digital I/O register, is assigned to the 16 least significant bits, while the MSB_DIO_REG, which is really the Digital I/O Control register, is assigned to the 16 most significant bits. The purpose of this line is to be able to load the Digital I/O register with all 0's in every bit except the least significant bit, which has a 1, while the Digital I/O Control register is loaded with all bits 0 except bit 15, bit 13, and bit 12. Referring to sections 5.6.2.1 and 5.6.2.2, the Digital I/O is then initially configured as all output bits disabled and bits 8 through 15 configured as outputs. The reason for this configuration is that this is the initial setup that Signalware uses when the want to test the Digital I/O. By manipulating the Digital I/O Control register properly and shifting the bits 1 bit the left at a time in the Digital I/O register, the test methodology of having all the Digital I/O initially off and then turning one bit on at a time can be achieved. This setup is not used in this control, but was left in for clarity for potential future use.

6.2.7.4 Static Declarations [6]

The float variables outputsave, u3save, y3save, KeyHolesave, Msec, and delaysave are defined as arrays. These variables are used to store system information. Once the appl_process function terminates the application, the appl_end function writes these results to an ASCII file through the serial USB port on the host computer in order to analyze the results. Outputsave represents the control signal output at all times. As a result, the Outputsave variable stores the base current value in amperes while being output and then switches to the k^{th} peak current level once the base time frame has expired. U3save represents the calculated next k^{th} peak current level in amperes. Y3save represents the k^{th} measured peak current time frame in milliseconds. The KeyHolesave variable represents the efflux or keyhole potential between the work piece and the detection plate in volts. The Msec variable represents the time in milliseconds since the control process began. The final variable delaysave represented the time period in milliseconds from when the keyhole potential threshold of 0.5 volts was exceeded to when the keyhole potential falls below this threshold.

The array float variables A1, A2, and A3 represent system model parameters. These variables are in an array format to allow for the storage of minimum and maximum values to be stored. These extremes are established based upon the parameter estimation program detailed in following section. Based upon these extremes, an iterative combinational comparison is performed to determine the combination, which results in the maximum system response. The parameter values, which result in the maximum response, are then stored in the float variables Aone, Atwo, and Athree. Based upon how the non-linear interval system model was defined, the A0 system parameter is subtracted out of the result when calculated the k+1 peak time duration; therefore, its inclusion in this module is unnecessary. For detail concerning the non-linear interval system model please refer to chapter 7.

The array float y was used to track the measured peak current level time durations. The array size was seven in order to track the current k^{th} measurement as well as three iterations in the past and three iterations in the future. For the Quasi-Keyhole process this number of

iterations has been found to produce sufficient results. For other processes where the Non-Linear Interval control methodology is applied, additional iterations may be necessary to produce stable results.

The array float ymeasured measures the actual peak time duration every time the keyhole potential has exceeded the threshold voltage. The measurement does not actually occur until the keyhole potential falling back below the peak threshold value. This value is then immediately assigned to the variable y[3] in order that this measurement can be used in the control process and saved backwards in time for the next k^{th} iteration.

The float array u is used to track the peak current control signal. This array is of size five because of the system model is structured in such a manner that the peak current duration is only needed one iteration into the future when calculating the peak current time duration three iterations into the future. These variables are initially assigned to the maximum allowable peak current level of 135 amperes in the appl_init function. It is important to note that u[4] is always equal to u[3] throughout the control process because when predicting the system response into the future the results are based upon a static peak current input at time frame k.

The float array du is used to track the peak current control signal change between iterations. An array size of six could have been used instead of seven since u[6] is never used in the control process; however, no error is induced due to this oversight. These variables are initially assigned to zero in the appl_init function. It is important to note that du[4] and du[5] are always equal to zero and never changes throughout the control process because when predicting the system response into the future the results are based upon a static peak current input at time frame k.

The float variable y0 is the reference peak current time duration in milliseconds. In this process, a time period of 325 ms was found to work efficiently. This variable is initialized in the appl_init function. It does not change throughout the module.

The float variable KeyHolePotential tracks the measured keyhole or efflux potential. This variable is initialized as zero in the appl_init function. Once a measurement is made it is immediately assigned to the k^{th} KeyHolesave variable for storage purposes.

The float variable BaseTime is used to assign the base current time duration in milliseconds. In this process, a time period of 400 ms was found to work efficiently. This variable is initialized in the appl_init function. It does not change throughout the module. The BaseTime is initiated once the keyhole potential drops below the threshold level. Once the BastTime period has expired, the next k^{th} peak current level is initiated. It is also important to note that the base time duration can vary significantly between different hardware applications of this process. The reason being that the amount of heat that can be dissipated from the system is dependent upon how quickly the hardware can set the base current level. In other words, the time constant of the analog output is very important.

The float variables x0, x1, and x2 are used to track the peak time duration and the delay associated with dropping the keyhole potential below the threshold once the base current level

has been initiated. All of these measurements are made in milliseconds referenced from when the control process was initiated. The variable x0 represents the time when the peak current level was initiated. The variable x1 represents the time that the current is dropped to the base level. The variable x2 represents the time when the keyhole potential actually drops below the threshold level after the base current level has been initiated. As a result, the measured peak time duration, ymeasured, is equal to the difference between the x2 and x0, and the delay is equal to the difference between x2 and x1. This time measurements are established by monitoring the float variable counter. The counter is based upon the DMA download rate, which in this application is equal to the D/A clock rate. Every time a new block of data is available from the DMA the appl_process function is called. Since the DMA and D/A rates are set at 1.953125 KHz, counting the number of times that the appl_process function is called and multiplying this number by 0.512 to obtain a time measurement in milliseconds can establish the time. It is important to point out that if the appl_process is still processing data when next block transfer is ready an overflow condition will exist. This will result in a small timing error; however, if the number of A/D samples that are missed is small, the error will be insignificant particularly when considering the A/D sampling rate of 500 KHz. This error is not cumulative since the peak time measurements are only relative to the welding cycle that they occur.

The float variable Time is used to capture the present time measurement. It is assigned to the variable Msec for storage purposes, but more importantly it is used to compare against x0 to determine when the base time period has expired. Since the Time assignment is being made after the comparison to the x0 variable, there is a 0.512 ms error; however, this error is insignificant when considering the magnitude of time frames being considered. If one wished to remove this error, the time assignment would be made immediately after the counter increment in the appl_process function and all other time assignments would be removed.

The float variable BaseCurrent is used to assign the base current level in amperes. In this case, a base current level of 30 amperes was found to work efficiently. This variable is initialized in the appl_init function. It does not change throughout the module.

The float variable ylargest is used to store the predicted system response for the various system parameter combinations three times steps into the future. This response is based upon no change in the existing peak current level. If y[7] exceeds this variable, ylargest is reassigned to y[7] and the associated system parameters are stored to Aone, Atwo, and Athree. This variable is assigned to zero in the appl_process function before the combinational parameter comparison takes place.

The float variables meltdown and TimeMeltdown exist to prevent the welding nozzle from becoming damaged. These variables are initialized to zero in the appl_init function. Within the appl_process function, if the peak current level saturates at the maximum rating, the meltdown variable acts as a counter, and the TimeMeltdown variable acts as the physical time measurement. If this saturation current exists for 1.024 seconds, the control output will output a zero value to prevent the nozzle from melting. If the predicted peak current level is below the maximum current level, both variables are reinitialized to zero.

The float variables StartCurrent and MaxCurrent are used to establish the initial peak current starting level and the saturation level. These variables are assigned in the appl_init function. The initial and maximum current ratings are established to be equal in this application at 135 amperes.

The int variable count is used to track the array index for the storage variables used to output system information in the appl_end function. It is initialized to zero in the appl_init function. Within the appl_process function, this variable is incremented every time storage variables have information saved to them. The count variable is also used to determine when to terminate the process. When the count value equals 309, the write_32b_reg function outputs to the analog output a quantitized variable that will result in a zero output. The program is allowed to iterate one more time to insure that this output is processed to the analog output port, at which time the process function will terminate the process. It is important to perform this procedure properly, or the analog outputs will remain at their last programmed value and the torch will not shut off.

The float variable skip exists to allow the program to be able to terminate if the peak current level stabilizes at a value below a level that can establish a keyhole through the work piece. This variable is initialized to zero in the appl_init function. This variable also serves the purpose of saving random data that is not tied to the peak current rising or falling edges. This variable causes information to be stored every 2000 iterations or 1.024 ms as configured in the appl_process function. After every store, this variable is reinitialized to zero and the count starts over.

The int variable start acts as a masking variable to allow other conditional statements to be bypassed upon startup. At which point, the peak current level is set for next k^{th} iteration and the masking provided by this variable is removed. This variable is initialized to zero in the appl_init function and reset to one after initial start up.

The int variable mask acts as a masking variable to prevent conditional statements from being processed. Once the delay time frame has passed this variable is set to 1. This variable is reset to 0 once the base time frame has expired. This variable is initialized to zero in the appl_init function.

The float variable openloopthree exists to allow the program to operate in an open-loop condition upon startup for a user-configured period of iterations. This value was set to zero and not utilized, but the logic is still in the code and may be useful depending on the application.

The int variable maskdelay is a masking variable used to prevent conditional statements from being processed. Once the keyhole potential has exceeded the threshold voltage, this variable is set to 1. This variable is reset to 0 once the base time frame has expired. This variable is initialized to zero in the appl_init function.

6.2.7.5 Appl_Parms Outline [4] [6]

The main purpose of the appl_parms function is to configure the FPGA memory mapped registers properly and to return the data transfer variables back to the AED_MAIN program. In addition, the appl_parms function provides the page mapping defined within the five most significant bits of the 20-bit Data Space memory addresses by defining the hexadecimal value for the DSK5416_DM_CNTL register. The Data Space page mappings are defined by this process as follows: control page is defined as page 8; register page is defined as page 9; and the page for DMA transfer data is page 16.

The hexadecimal return values for the functions get_cntl_addr and get_data_addr have been previously defined in the 5416_dsk module as 8000_{16} and 0000_{16}. A bitwise OR is performed with the get_data_addr with the hexadecimal address 8000_{16} to allow both functions to have the same hexadecimal address. Since the hexadecimal value of 8000_{16} sets all bits to zero except bit 15, an external Daughter Card access can be performed once this value is added to the previously defined 16-bit FPGA memory-mapped register addresses.

6.2.7.6 Appl_Init Outline [6]

The appl_init function is rather self-explanatory. It is used to initialize variables. It does do two other important functions. It loads the DMA data_block with all zero's to avoid an underflow condition, and it instructs the write_32b_reg function to output a quantization value that will cause the analog output to be zero as well.

6.2.7.7 Appl_Process Outline [6]

The appl_process is where the Non-Linear Control Algorithm resides. The function is called by the AED_MAIN every time a new block of data has been downloaded from the THS1209 through the DMA controller read channel. The function starts by performing a summation of the A/D data samples. If the actual 12-bit quantization had been desired, the number of iterations would be equal to the actual number of A/D samples that were taken; however, in order to match the quantization requirements of the write_32b_reg function, the entire data block was sampled a four times instead of just one. This was done to effectively multiply the sum by 4, which is equivalent to shifting 2 bits to the left as well. As a result, the quantization has been spread from $0 \rightarrow 4096$ to $0 \rightarrow 16384$. The result of this fictitious 14-bit quantization is that it now takes a change of four intervals to effect a change in the output through the write_32b_function. A_value represents the number of quantitized intervals. It has base ten representation, but is calculated by shifting the base 2 representation by the base 2 exponent equivalent to the number of A/D samples being averaged. The actual keyhole potential is calculated using equation 6-1.

Once a base ten floating-point representation of the analog input has been obtained, the counter variable begins incrementing in order to provide a means of calculating a time measurement. The concept for calculating time is based upon knowing the time interval that the appl_process function is called, which is 0.512 ms.

A conditional else-if statement is then entered, which has four potential possibilities. Only one of this conditions will be true for every iteration. The first condition detects when the

keyhole potential threshold has first been exceeded. At this point, a time measurement is made, the base current is set, and data is saved. The next condition detects when the keyhole potential falls below the threshold level after the base current has been initiated. At this point, a time measurement is made from which the delay variable and the base current ending time can be calculated. In addition, the control algorithm commences.

The control algorithm proceeds by first performing a combinational parameter comparison in order to determine the combination which produces the largest system response based upon no change in the peak current level. The system parameters that produce the maximum response are then used to calculate the next iteration peak current output. By comparing the largest system response based upon no change in the peak current to the reference peak time frame, the peak current level can either be decremented or incremented one ampere per iteration until the system response three times steps into the future changes state in relation to the reference peak time duration. The measured peak time duration and peak currents are then saved backwards one iteration while maintaining the present k^{th} value. If the k^{th} value is above or below the maximum current level or the base current level, the peak current duration is truncated to these limiting values. Data is then saved.

The third conditional statement is only used at initial start up. Its purpose is to make a time measurement, output the starting peak current level, and save data.

The fourth conditional statement is only executed after the base current time period has been exceeded. At this point, a time measurement is made, the peak current is set, and data is saved.

Once the conditional else-if statement has been processed, a conditional if statement is encountered whose purpose is to insure that the peak current level does not saturate for a period greater then 1.024 seconds. If the peak current is saturated for 1.024 seconds the analog output is instructed to fall back to zero in order to keep the welding nozzle from melting.

Another conditional if statement is then encountered whose purpose is to allow the program to terminate if the peak current level is set to a point that cannot establish a keyhole through the work piece. In addition, this conditional if statement also provides a method of saving data in a fashion that is not dependent upon the rising and falling edges of the peak current time frame.

The last two conditional if statements provide a method of setting the analog output to zero and then terminating the application. This ability is based upon tracking the number of data saves that occur. Once a threshold has been exceeded, this logic will take effect.

6.2.7.8 Appl_Idle Outline [6]

The appl_idle function has a relatively simple, but important purpose. In order to prevent an underflow situation associated with the DMA data block, the appl_idle function is entered if the appl_process function completes it's processing before the next set of data is available from

the A/D data converter. The AED_MAIN program will continually call the appl_idle function until the next data set is ready. No processing is performed in the appl_idle function.

6.2.7.9 Appl_End Outline [6]

Once the appl_process terminates the application, the AED_MAIN program calls the appl_end function before actually ending the program. In the appl_end function, the results of the diagnostic terminators are printed to the host's monitor, but more importantly the system data is saved to an ASCII file on the host computer through the on-board USB port.

6.2.8 Parameter Estimation Layout [6]

The parameter estimation program is another modified version of the AED_109_32d module. The purpose of this module is to provide random data input to the system and measure the results. NONLinearIntervalParameterEstimate is the name of this module. This module is similar to the NONLinearInterval_delay_quicker with several exceptions. The first exception is that there is no control algorithm within this file, which reduces the number of required variables and shortens the code extensively. The second exception is that an ASCII random input file has been created by MatLab and is resident on the Host PC. This data set peak current range was configured bounded between 85 to 115 amperes. This range was chosen to insure establishment of a keyhole and to simulate the operating range of the process. As a result, this file is opened, read, and closed within the appl_init function. The third exception is that current limiting is moved from the appl_process to the appl_init function immediately following the data input and is not as complex. The current limiting function simply insures that the maximum current level is not exceeded. The fourth exception is that data is only saved when keyhole potential threshold is exceeded. The fifth exception is that the initial start up current is set to produce a zero output on the AO. The sixth exception is that the conditional statement that provides a method for termination if the peak current level is set in a manner that establishment of a keyhole is unachievable is commented out under normal operation. This conditional statement is only used to initialize the AO to zero with the Miller power supply turned off. Once the AO is set to zero, the comments surrounding this conditional statement are reinserted and the parameter testing can commence without damaging the equipment.

6.3 Code Files [6]

All of the required source code is attached in Appendixes C, D, E, and F, with the exception of the project file and libraries. The project and library files could not be attached because they were already compiled by Code Composer Studio, resulting in the ASCII text mode being unavailable. Appendix C contains the GEL file generated by CCS when a target board is detected. Appendix D illustrates the linker command file. Appendix E contains the headers, and Appendix F contains the actual function modules. In Appendix F, the original AED_109_32d file is specified along with the modified versions for the actual control and parameter estimation: NONLinearInterval_delay_quicker and NONLinearIntervalParameterEstimate.

Chapter Seven

Non-Linear Control Algorithm

7.1 Overview [10]

The non-linear interval control algorithm for the Quasi-Keyhole plasma arc welding process provides a means of robust control for systems with bounded parameter variations. This is accomplished by systematically varying the peak and base current levels as well as their time duration in order to control weld quality, which is directly related to the level of heat present in the work piece. In order to simplify the model for this application, the base current level and time duration were pre-defined to static values; however, for a more complex model, this may not be allowable.

The establishment of a keyhole potential threshold provides a means of determining when a keyhole has been established through the work piece. At this time, the control signal can be reduced to the base level in order to close the keyhole and allow heat to dissipate from the system. A peak current reference time period is then used by the control algorithm to calculate the next iteration peak current level before repeating the cycle again.

7.2 Model Description [10]

As previously mentioned, the time duration and current level provided to the system is directly related to the amount of heat present. As a result, the non-linear interval model is presented in equation 7-1, where y_k represents the k^{th} weld cycle predicted peak current time duration, y_{k-1} through y_{k-n+1} represents actual measured peak current time durations for all previous weld cycles, u_{k-1} through u_{k-n} represents the peak current levels for all previous weld cycles, and coefficients a_0 through a_n represent the system parameters:

$$y_k = a_0 + a_1 u_{k-1} + a_2 u_{k-2} y_{k-1} + a_3 u_{k-3} y_{k-2} + \ldots + a_n u_{k-n} y_{k-n+1} \qquad 7-1$$

It is evident that the product of the peak current level and time duration creates a non-linear based system. In order to reduce processing complexity, the interval model presented in 7-1 was only considered for three previous iterations realizing that for other applications this assumption may not be sufficient to insure stability. The simplified model is shown below:

$$y_k = a_0 + a_1 u_{k-1} + a_2 u_{k-2} y_{k-1} + a_3 u_{k-3} y_{k-2} \qquad 7-2$$

Since the system is subject to varying operating parameters due to the manufacturing environment and the stochastic nature of the physics describing the weld pool and welding arc, several random input test runs are required to characterize the parameter bounds of the system. For the most robust control, these test runs would be performed for a variety of work piece thickness and travel speeds; however, for simplicity, this project was restrained to working with a single work piece thickness of 3.0 mm and travel speed of 2.534 mm/sec. As a result, four test runs were performed from which the system parameters could be determined utilizing a least

squares approximation. For specific details on the least squares methodology or parameter test runs please refer to chapters 8 and 9.

The units of the system parameters are as follows: a_o is expressed in milliseconds; a_1 is expressed in milliseconds per ampere; and a_2 through a_n are expressed in inverse amperes. The parameter a_1 through a_n are in general usually negative. The reason for this basis is that the previous peak current durations contribute to the total amount of heat in the system at the present iteration. Since the a_o parameter is positive, the previous heat input cycles effectively reduce the amount of heat that needs to be input to the system on the next iteration, which in turn reduces the magnitude of the peak current time duration. However, when identifying a model using the least squares approximation, the higher order term parameters may be slightly positive, which effectively accounts for system characteristics that have not been described in the subject model. Realistically, the higher order terms contribute less as they fade farther into the past. As a result, the parameter a_3 may actually be positive.

7.3 Feedback Algorithm [10]

In order to provide an effective feedback mechanism, equation 7-2 should first be examined. By subtracting the difference between successive peak current time durations, the following equation can be derived:

$$y_{k+1} = y_k + a_1 \Delta u_k + a_2 \left(u_{k-1} y_k - u_{k-2} y_{k-1} \right) + a_3 \left(u_{k-2} y_{k-1} - u_{k-3} y_{k-2} \right) \qquad 7-3$$

In some circumstances, it is desired to correlate positive changes in system outputs with positive changes in control signal inputs despite the fact that this thought process is inversely related to the subject PAW process. As a result, the authors of this algorithm envisioned describing positive values; therefore, the following substitutions were made resulting in 7-6:

$$\tilde{a}_j = -a_j, \text{ where } j = 1, 2, 3 \qquad 7-4$$

$$\tilde{u} = -u \qquad 7-5$$

$$y_{k+1} = y_k + \tilde{a}_1 \Delta \tilde{u}_k + \tilde{a}_2 \left(\tilde{u}_{k-1} y_k - \tilde{u}_{k-2} y_{k-1} \right) + \tilde{a}_3 \left(\tilde{u}_{k-2} y_{k-1} - \tilde{u}_{k-3} y_{k-2} \right) \qquad 7-6$$

If the reader examines the module NONLinearInterval_delay_quicker carefully, it would be noticed that 7-6 was being used, but the non-tilde expressions for 7-4 and 7-5 were being substituted. It is the author's opinion, that the evaluation of all test data in a positive form masks the actual physics of the process and should not be used. As a result, 7-3 will be used in the discussion thereafter.

All next iteration peak current time durations are predicted based upon values that occurred three iterations into the past. In order to balance the prediction in relation to past references, the system response is then predicted three iterations into the future. Based upon the parameters outlined in Table 9-1, the minimum and maximum values for each parameters were chosen to characterize the bounds of the system. As a result, during every iteration, the

parameter combination that produces the maximum system response based upon no change in the present peak current level is determined first. By using the combination of parameters that produce the maximum predicted response, the smallest change in the control signal can be utilized.

Once the combination of parameters that produce the maximum system response based upon no change in the present peak current level is known, the associated system response can then be compared to the reference value. If this maximum system response is larger, the peak current level will need to be increased. Otherwise, if the maximum system response is smaller, the peak current level needs to be decreased.

The magnitude of peak current level change is the next logic to be addressed. In order to calculate the necessary change in peak current levels, the peak current is changed by 1 ampere per iterative comparison. The resulting change in predicted peak current time duration three cycles into the future is then compared to the reference peak current duration again. If the predicted response still has the same relationship to the reference as the original maximum system response based upon no peak current change, an additional 1 ampere change will be processed and compared again. This iterative comparison is continued until the predicted response changes relationship with the reference. Once the next iteration peak current level has been calculated, the control iteration is complete and the welding cycle is ready to be repeated.

Chapter Eight

Parameter Estimation

8.1 Construct [10]

Previously, a non-linear model has been proposed to mathematically describe the Quasi-Keyhole
process as follows:

$$y_k = a_0 + a_1 u_{k-1} + a_2 u_{k-2} y_{k-1} + a_3 u_{k-3} y_{k-2} \qquad 8-1$$

The coefficients of this model are often referred to as system parameters. System parameters provide a means to calibrate the system to the local environment. In addition by having a basic understanding of the reasoning behind a model, certain generalities may be established in regards to these parameters. This model is attempting to predict the present peak current duration based upon previous current levels and peak current times. As the peak current and associated durations are increased, the amount of heat in the system increases, which results in a smaller time frame before the keyhole is established. Therefore, in general, the a_1, a_2, and a_3 coefficients should be negative and relatively small in magnitude in order to reduce the estimated k^{th} peak current time duration, while a_0 should be positive and relatively large in order to prevent the peak duration from becoming negative, which is a physical impossibility.

8.2 Matrix Expansion

Often when describing systems, it is convenient to group the iterative variance together into one mathematical construct called a matrix equation. The matrix expansion for the aforementioned model is made exact by including an error vector as shown on the following page:

$$\begin{bmatrix} y_k \\ y_{k+1} \\ . \\ . \\ y_{k+n} \end{bmatrix} = \begin{bmatrix} 1 & u_{k-1} & u_{k-2} y_{k-1} & u_{k-3} y_{k-2} \\ 1 & u_k & u_{k-1} y_k & u_{k-2} y_{k-1} \\ . & . & . & . \\ . & . & . & . \\ 1 & + u_{k+n-1} & u_{k+n-2} y_{k+n-1} & u_{k+n-3} y_{k+n-2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} e_k \\ e_{k+1} \\ . \\ . \\ e_{k+n} \end{bmatrix} \qquad 8-2$$

Realizing that a vector is defined as a matrix whose dimensions are (n + 1) x 1, this matrix expansion can be simplified as follows:

$$\bar{Y} = \tilde{\Phi}\bar{\hat{\theta}} + \bar{E} \qquad 8-3$$

These matrixes are commonly referred to as the output vector, observation matrix, estimated parameter vector, and error vector where the dimensions are (n + 1) x 1, (n + 1) x 4, 4 x 1, and (n

+ 1) x 1 respectively. It is also important to note that the observation matrix is singular or noninvertible in system identification problems, which prevents a trivial solution for the parameter vector. Furthermore, it is also of interest to note that estimated output is formed by non-linear system since the input and output form a cross product within the observation matrix, but it will be shown later that a linear relationship can be developed for the least squares vector, which is solely a function of the real output vector and the observation matrix.

8.3 Cost Function

In order to calibrate the system, unbiased numerical information must be obtained in order to quantify the differences between the real physical system and the estimated. As a result, unbiased system data is obtained by supplying random control signals, within a specified range, and measuring the corresponding system outputs. The numerical comparison is achieved by defining a cost function as follows:

$$J\left(\bar{\hat{\theta}}\right) = \sum_{k=L+1}^{n}\left(y_k - \hat{y}_k\right)^2 = \bar{E}^T\bar{E} \qquad 8-4$$

Since the product of the observation matrix and estimated parameter vector forms the measured output vector, the error vector is defined as the difference between measured output and the estimated as shown below:

$$\bar{E} = \tilde{\Phi}\bar{\hat{\theta}} - \bar{\hat{Y}} = \bar{Y} - \bar{\hat{Y}} \qquad 8-5$$

8.4 Least Squares Parameter Solution

Now that a mathematical comparison has been defined, it should be realized that this cost function effectively forms a parabola with only a single minimum value as the system parameters are varied; therefore, from basic calculus, the least squares parameter values may be obtained by taking the partial derivative of the cost function with respect to the parameter vector, setting the equation equal to zero, and solving for the least squares parameters, since the slope of the cost function will be zero at this minimum value. The equation below summarizes:

$$\frac{\partial J\left(\bar{\hat{\theta}}\right)}{\partial J\bar{\hat{\theta}}}\Bigg|_{\bar{\hat{\theta}}=\bar{\hat{\theta}}_{ls}} = \left[\frac{\partial J\left(\hat{\theta}\right)}{\partial\hat{\theta}_0} \quad \frac{\partial J\left(\hat{\theta}\right)}{\partial\hat{\theta}_1} \quad \frac{\partial J\left(\hat{\theta}\right)}{\partial\hat{\theta}_2} \quad \frac{\partial J\left(\hat{\theta}\right)}{\partial\hat{\theta}_3}\right]^T\Bigg|_{\bar{\hat{\theta}}=\bar{\hat{\theta}}_{ls}} = 0 \qquad 8-6$$

It is important to note that although the cost function itself is a scalar, equation 8-6 is actually a vector since the parameter vector is actually a function of four independent parameters. Based upon equation 8-6, the least squares parameter solution will simplify to the following form:

$$\underline{\hat{\theta}}_{ls} = \left[\tilde{\Phi}^T \tilde{\Phi}\right]^{-1} \tilde{\Phi}^T \tilde{Y}, \quad \text{provided} \quad \left[\tilde{\Phi}^T \tilde{\Phi}\right]^{-1} \quad \text{exists} \qquad 8-7$$

As a result, the least squares parameter estimation becomes very simple, with the use of MatLab, once a method of providing a random input and sampling the corresponding output is achieved.

8.5 Proof of Least Squares Parameter Solution

The proof of the least squares approximation is based in linear algebra. As a result, the proof starts with the cost function defined in equation 8-4. By substituting the error vector, equation 8-5, into the cost function the following result is obtained:

$$J\left(\underline{\hat{\theta}}\right) = \left[\underline{Y} - \underline{\hat{Y}}\right]^T \left[\underline{Y} - \underline{\hat{Y}}\right] = \left[\underline{Y}^T - \underline{\hat{Y}}^T\right] \left[\underline{Y} - \underline{\hat{Y}}\right] \qquad 8-8$$

The foil method can then be applied as follows:

$$J\left(\underline{\hat{\theta}}\right) = \underline{Y}^T \underline{Y} - \underline{\hat{Y}}^T \underline{Y} - \underline{Y}^T \underline{\hat{Y}} + \underline{\hat{Y}}^T \underline{\hat{Y}} \qquad 8-9$$

By substituting the estimated output, equation 8-3, into equation 8-9 transforms as shown below:

$$J\left(\underline{\hat{\theta}}\right) = \underline{Y}^T \underline{Y} - \left[\tilde{\Phi}\underline{\hat{\theta}}\right]^T \underline{Y} - \underline{Y}^T \left[\tilde{\Phi}\underline{\hat{\theta}}\right] + \left[\tilde{\Phi}\underline{\hat{\theta}}\right]^T \left[\tilde{\Phi}\underline{\hat{\theta}}\right] \qquad 8-10$$

This equation can then be further algebraically manipulated as described below:

$$J\left(\underline{\hat{\theta}}\right) = \underline{Y}^T \underline{Y} - \underline{\hat{\theta}}^T \tilde{\Phi}^T \underline{Y} - \underline{Y}^T \tilde{\Phi}\underline{\hat{\theta}} + \underline{\hat{\theta}}^T \tilde{\Phi}^T \tilde{\Phi}\underline{\hat{\theta}} \qquad 8-11$$

Now the third term needs to be examined in further detail as follows:

$$\underline{Y}^T \tilde{\Phi} \underline{\hat{\theta}} = \left[\underline{\hat{\theta}}^T \left[\underline{Y}^T \tilde{\Phi} \right]^T \right]^T = \left[\underline{\hat{\theta}}^T \tilde{\Phi}^T \underline{Y} \right]^T = \underline{\hat{\theta}}^T \tilde{\Phi}^T \underline{Y} \qquad 8-12$$

Equation 8-12 demonstrates that the second and third terms in equation 8-11 are equal, which allows the cost function to be further manipulated as follows:

$$J\left(\underline{\hat{\theta}}\right) = \underline{Y}^T \underline{Y} - 2 \underline{\hat{\theta}}^T \tilde{\Phi}^T \underline{Y} + \underline{\hat{\theta}}^T \tilde{\Phi}^T \tilde{\Phi} \underline{\hat{\theta}} \qquad 8-13$$

The following temporary variables are now defined in order to simplify the algebraic manipulation as shown below:

$$\tilde{A} = \tilde{\Phi}^T \underline{Y} \qquad 8-14$$

$$\tilde{B} = \tilde{\Phi}^T \tilde{\Phi} = \left[\tilde{\Phi}^T \left[\tilde{\Phi}^T \right]^T \right]^T = \left[\tilde{\Phi}^T \tilde{\Phi} \right]^T = \tilde{B}^T \qquad 8-15$$

It is important to note that equation 8-15 demonstrates that this matrix is symmetric. Since equation 8-15 is symmetric, the elements within this matrix have the following property:

$$b_{ij} = b_{ji} \qquad 8-16$$

Substituting equations 8-14 and 8-15 into equation 8-13 the following transformation is shown:

$$J\left(\underline{\hat{\theta}}\right) = \underline{Y}^T \underline{Y} - 2 \underline{\hat{\theta}}^T \tilde{A} + \underline{\hat{\theta}}^T \tilde{B} \underline{\hat{\theta}} \qquad 8-17$$

Equation 8-6 is then applied to equation 8-17, which results in the following:

$$\frac{\partial J\left(\underline{\hat{\theta}}\right)}{\partial \underline{\hat{\theta}}} = 0 - 2 \frac{\partial J\left(\underline{\hat{\theta}}^T \tilde{A}\right)}{\partial \underline{\hat{\theta}}} + \frac{\partial J\left(\underline{\hat{\theta}}^T \tilde{B} \underline{\hat{\theta}}\right)}{\partial \underline{\hat{\theta}}} \qquad 8-18$$

At this point, it is convenient to look at the second and third terms independently. The second term is examined first as shown below:

$$2\frac{\partial J\left(\hat{\underline{\theta}}^{T}\tilde{A}\right)}{\partial\hat{\underline{\theta}}} = 2\frac{\partial J\left(\left[\hat{\theta}_0 \quad \hat{\theta}_1 \quad \hat{\theta}_2 \quad \hat{\theta}_3\right]\left[a_{11} \quad a_{21} \quad a_{31} \quad a_{41}\right]^{T}\right)}{\partial\hat{\underline{\theta}}} \qquad 8-19$$

Matrix multiplication is then performed as outlined below:

$$2\frac{\partial J\left(\hat{\underline{\theta}}^{T}\tilde{A}\right)}{\partial\hat{\underline{\theta}}} = 2\frac{\partial J\left(\left[\hat{\theta}_0 a_{11} + \hat{\theta}_1 a_{21} + \hat{\theta}_2 a_{31} + \hat{\theta}_3 a_{41}\right]\right)}{\partial\hat{\underline{\theta}}} \qquad 8-20$$

Performing the partial derivative in accordance with equation 8-6, the following solution for the second term of equation 8-18 is developed:

$$2\frac{\partial J\left(\hat{\underline{\theta}}^{T}\tilde{A}\right)}{\partial\hat{\underline{\theta}}} = 2\left[a_{11} \quad a_{21} \quad a_{31} \quad a_{41}\right]^{T} = 2\tilde{A} = 2\tilde{\Phi}^{T}\tilde{Y} \qquad 8-21$$

The third term in equation 8-18 is now examined as the following shows:

$$\frac{\partial J\left(\hat{\underline{\theta}}^{T}\tilde{B}\hat{\underline{\theta}}\right)}{\partial\hat{\underline{\theta}}} = \frac{\partial J\left(\left[\hat{\theta}_0 \quad \hat{\theta}_1 \quad \hat{\theta}_2 \quad \hat{\theta}_3\right]\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}\begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \hat{\theta}_2 \\ \hat{\theta}_3 \end{bmatrix}\right)}{\partial\hat{\underline{\theta}}} \qquad 8-22$$

Matrix multiplication is then performed in two steps as outlined below:

$$\frac{\partial J\left(\begin{bmatrix}\hat{\theta}^T & \tilde{B}\hat{\theta}\end{bmatrix}\right)}{\partial \bar{\hat{\theta}}} = \frac{\partial J\left(\begin{bmatrix}\hat{\theta}_0 & \hat{\theta}_1 & \hat{\theta}_2 & \hat{\theta}_3\end{bmatrix}\begin{bmatrix} b_{11}\hat{\theta}_0 + b_{12}\hat{\theta}_1 + b_{13}\hat{\theta}_2 + b_{14}\hat{\theta}_3 \\ b_{21}\hat{\theta}_0 + b_{22}\hat{\theta}_1 + b_{23}\hat{\theta}_2 + b_{24}\hat{\theta}_3 \\ b_{31}\hat{\theta}_0 + b_{32}\hat{\theta}_1 + b_{33}\hat{\theta}_2 + b_{34}\hat{\theta}_3 \\ b_{41}\hat{\theta}_0 + b_{42}\hat{\theta}_1 + b_{43}\hat{\theta}_2 + b_{44}\hat{\theta}_3 \end{bmatrix}\right)}{\partial \bar{\hat{\theta}}} \quad 8-23$$

$$\frac{\partial J\left(\begin{bmatrix}\hat{\theta}^T & \tilde{B}\hat{\theta}\end{bmatrix}\right)}{\partial \bar{\hat{\theta}}} = \frac{\partial J\left(\begin{matrix} b_{11}\hat{\theta}_0^2 + b_{12}\hat{\theta}_0\hat{\theta}_1 + b_{13}\hat{\theta}_0\hat{\theta}_2 + b_{14}\hat{\theta}_0\hat{\theta}_3 + \text{next row} \\ b_{21}\hat{\theta}_0\hat{\theta}_1 + b_{22}\hat{\theta}_1^2 + b_{23}\hat{\theta}_1\hat{\theta}_2 + b_{24}\hat{\theta}_1\hat{\theta}_3 + \text{next row} \\ b_{31}\hat{\theta}_0\hat{\theta}_2 + b_{32}\hat{\theta}_1\hat{\theta}_2 + b_{33}\hat{\theta}_2^2 + b_{34}\hat{\theta}_2\hat{\theta}_3 + \text{next row} \\ b_{41}\hat{\theta}_0\hat{\theta}_3 + b_{42}\hat{\theta}_1\hat{\theta}_3 + b_{43}\hat{\theta}_2\hat{\theta}_3 + b_{44}\hat{\theta}_3^2 \end{matrix}\right)}{\partial \bar{\hat{\theta}}} \quad 8-24$$

At this point, it is interesting to note that the expression that the partial is in relationship with is a scalar as may have not been immediately apparent in equation 8-17. Performing the partial derivative in accordance with equation 8-6, the following vector is developed:

$$\frac{\partial J\left(\begin{bmatrix}\hat{\theta}^T & \tilde{B}\hat{\theta}\end{bmatrix}\right)}{\partial \bar{\hat{\theta}}} = \begin{bmatrix} 2b_{11}\hat{\theta}_0 + b_{12}\hat{\theta}_1 + b_{13}\hat{\theta}_2 + b_{14}\hat{\theta}_3 + b_{21}\hat{\theta}_1 + b_{31}\hat{\theta}_2 + b_{41}\hat{\theta}_3 \\ b_{12}\hat{\theta}_0 + b_{21}\hat{\theta}_0 + 2b_{22}\hat{\theta}_1 + b_{23}\hat{\theta}_2 + b_{24}\hat{\theta}_3 + b_{32}\hat{\theta}_2 + b_{42}\hat{\theta}_3 \\ b_{13}\hat{\theta}_0 + b_{23}\hat{\theta}_1 + b_{31}\hat{\theta}_0 + b_{32}\hat{\theta}_1 + 2b_{33}\hat{\theta}_2 + b_{34}\hat{\theta}_3 + b_{43}\hat{\theta}_3 \\ b_{14}\hat{\theta}_0 + b_{24}\hat{\theta}_1 + b_{34}\hat{\theta}_2 + b_{41}\hat{\theta}_0 + b_{42}\hat{\theta}_1 + b_{43}\hat{\theta}_2 + 2b_{44}\hat{\theta}_3 \end{bmatrix} \quad 8-25$$

Now remembering from equation 8-16 that elements within the matrix defined by equation 8-15 are symmetric the following simplification can be derived:

$$\frac{\partial J\left(\hat{\bar{\theta}}^{T}\tilde{B}\hat{\bar{\theta}}\right)}{\partial\hat{\bar{\theta}}} = \begin{bmatrix} 2b_{11}\hat{\theta}_0+2b_{12}\hat{\theta}_1+2b_{13}\hat{\theta}_2+2b_{14}\hat{\theta}_3 \\ 2b_{12}\hat{\theta}_0+2b_{22}\hat{\theta}_1+2b_{23}\hat{\theta}_2+2b_{24}\hat{\theta}_3 \\ 2b_{13}\hat{\theta}_0+2b_{23}\hat{\theta}_1+2b_{33}\hat{\theta}_2+2b_{34}\hat{\theta}_3 \\ 2b_{14}\hat{\theta}_0+2b_{24}\hat{\theta}_1+2b_{34}\hat{\theta}_2+2b_{44}\hat{\theta}_3 \end{bmatrix} \qquad 8-26$$

This expression can then be manipulated into a compact matrix equation, which is the solution to the third term of equation 8-17, as shown below:

$$\frac{\partial J\left(\hat{\bar{\theta}}^{T}\tilde{B}\hat{\bar{\theta}}\right)}{\partial\hat{\bar{\theta}}} = 2\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{12} & b_{22} & b_{23} & b_{24} \\ b_{13} & b_{23} & b_{33} & b_{34} \\ b_{14} & b_{24} & b_{34} & b_{44} \end{bmatrix}\begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \hat{\theta}_2 \\ \hat{\theta}_3 \end{bmatrix} = 2\left[\tilde{\Phi}\tilde{\Phi}^{T}\right]\hat{\bar{\theta}} \qquad 8-27$$

Substituting this result into equation 8-6 the following is obtained:

$$\frac{\partial J\left(\hat{\bar{\theta}}_{ls}\right)}{\partial\hat{\bar{\theta}}_{ls}} = -2\tilde{\Phi}^{T}\tilde{Y} + 2\tilde{\Phi}^{T}\tilde{\Phi}\hat{\bar{\theta}}_{ls} = 0 \qquad 8-28$$

Solving this expression provides a solution to the least squares approximation parameter vector a shown below:

$$\hat{\bar{\theta}}_{ls} = \left[\tilde{\Phi}^{T}\tilde{\Phi}\right]^{-1}\tilde{\Phi}^{T}\tilde{Y}, \quad \text{provided} \quad \left[\tilde{\Phi}^{T}\tilde{\Phi}\right]^{-1} \quad \text{exists} \qquad 8-29$$

This expression verifies that the individual elements of the parameter vector have a very simple solution provided equation 8-15 is nonsingular, a method of providing random input within specified parameters while measuring the corresponding output has been developed, and the data results can be mathematically analyzed with an automated mathematical software package such as Matlab.

Chapter Nine

Parameter Estimation Test Runs

9.1 Parameter Test Setup

In order to determine the system parameters for the Non-Linear Interval system model, the NONLinearIntervalParameterEstimate program should be substituted for the AED_109_32d module. In order for the system parameters to be truly representative of the process, external variables independent of the control process must be organized in a known, stable, and repetitive condition. These variables are established in the following configuration: 2.4 mm nozzle diameter; 2.534 mm/sec travel speed equating to a 0.362 volts servo reference voltage; argon shielding gas pressure, 35 CFH middle of ball; argon backing gas pressure, 35 CFH middle of ball; argon plasma jet pressure, 4 CFH bottom of ball; 5 mm nozzle height above work piece; 304 stainless steel work piece material; 3 mm work piece thickness; and a peak current operating range between 85 and 115 amperes. If these external variables are changed for any reason, a new set of system parameters will be required for the control algorithm to work effectively.

9.2 Calibration

The astute reader has probably realized that equations 6-1 and 6-2, which describes the analog input voltage and D/A quantization, have been slightly modified. These equations describe how the A/D and D/A should work ideally; however, due to board specific biasing concerns, these equations require calibration. With the Miller power supply off, the D/A quantization should be calibrated first. This is accomplished by examining the D/A output equation in the else-if statement that is executed when the time exceeds the x0 variable. By forcing the peak current duration to zero manually in this equation, the quantization interval, which corresponds to a zero analog output, $2/2^{14}$, can be adjusted until a zero AO is obtained. By experimentation, this quantization was determined to be 7991, which is slightly different from the idea value of 8192. At this point, the ratio described by -1/(55*9.24) can then be calibrated by setting the peak current duration to a fixed level in the midpoint of the operating range, executing, and measuring the analog output. In this case, 120 amperes was chosen. By following this method, the adjusted value may be obtained –0.00196679597881. As a result, equation 6-2 has been modified as follows:

$$\text{Quantization}_{\text{DAC}} = \left(\text{AO}_{\text{A}} \cdot \left(-0.00196679597881\right) + 1\right) \cdot 7991 \qquad 9-1$$

The A/D analog input voltage described by equation 6-1 is calibrated next. This is accomplished by wiring the analog input to a power supply. Since the analog input is not open circuited, there will be not an offset bias induced by the loading resistor placed across the input ADC amplifier used to reference the input to ground. The ratio $2/2^{14}$ can then be adjusted in order to match the analog input configured by the power supply to a set value equivalent to a normal operating voltage. By following this procedure, this ratio was determined to be 0.000124177. Equation 6-1 is thus modified as follows:

$$\text{AI}_{\text{V}} = \left(\text{Quantization}_{\text{ADC}} \cdot \left(0.000124177\right) - 1\right) \cdot 10 - \text{Bias} \qquad 9-2$$

9.3 Random Input Generator

This random input generator provides a method of providing a random input set from which the corresponding output can be measured and recorded. This input set is created in Matlab, which has limited the peak current range between 85 and 115 amperes. Once saved to an ASCII file, the input file can be accessed through the appl_init function using the USB port. The code that was used to generate this input file is shown below:

```
random=rand(125,1);
random=random*30;
random=random-15;
random=random+100;
plot(random)
fid=fopen('infile 1 12504.txt','w');
fprintf(fid,'%3.16f\n',random);
fclose(fid);
```

9.4 Analog Output Initialization

When the AED-109 starts up, the initial quantization provided to the THS5661 from the FPGA is all zeros. Unfortunately, this corresponds to a –1.0 volts output on the AO. Since the gain of the external amplifier is set at –9.24, the effective control voltage to the Miller Electric Maxtron 450 CC/CV power supply is 9.24 volts. If the Miller power supply powered up, this would equate to a peak current level of 508.2 Amperes, which would immediately damage the equipment. As a result, the iteration that provides for a termination method if a keyhole is not achievable is commented out under normal operation; however, at startup, these comments are removed and the interval used to increment the counter is set very short. Therefore, the Miller supply is left powered down and this program is executed for a very short duration. Once this program starts executing, the initial and final outputs are configured in such a manner as to produce a zero output on the AO. At the application termination, the AO will be outputting zero. At this point, the comments are reinserted around the conditional loop containing the counter called skip. The test runs can now commence without damaging the equipment.

9.5 Test Runs

Four test runs were performed to account for the inherent variance in the system. From each test run, a least-squares approximation was performed resulting in four sets of system parameters. The code for performing a least-squares approximation in MatLab is shown below:

```
load data.txt
L=4;
N=114;
for K=L+1:N;
   UpperPhi(K-L,☺=[1 data(K,1) (data(K-1,1)*data(K-1,2)) (data(K-2,1)*data(K-2,2))];
   UpperY(K-L)=data(K,2);
end;
```

```
UpperY=UpperY';
ThetaHatLS=(UpperPhi'*UpperPhi)^(-1)*UpperPhi'*UpperY;
ThetaHatLS
```

The system responses for each test run are shown on the following pages:

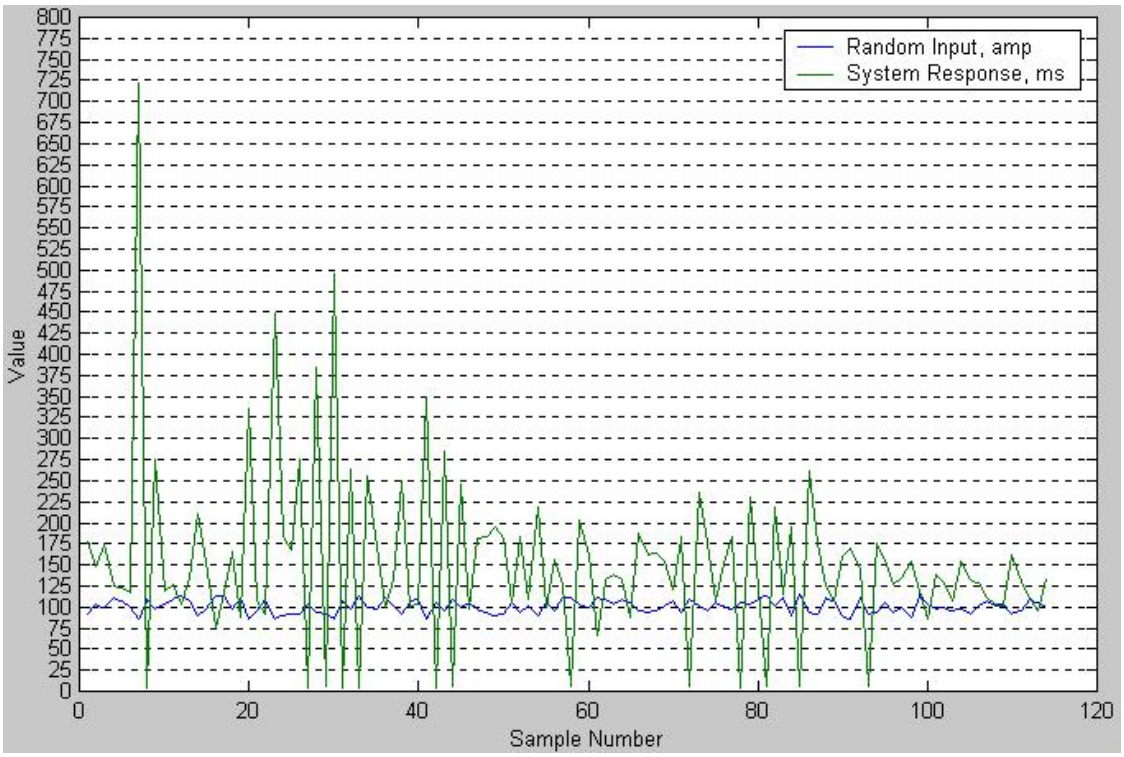


Figure 9-1 Test Run 1

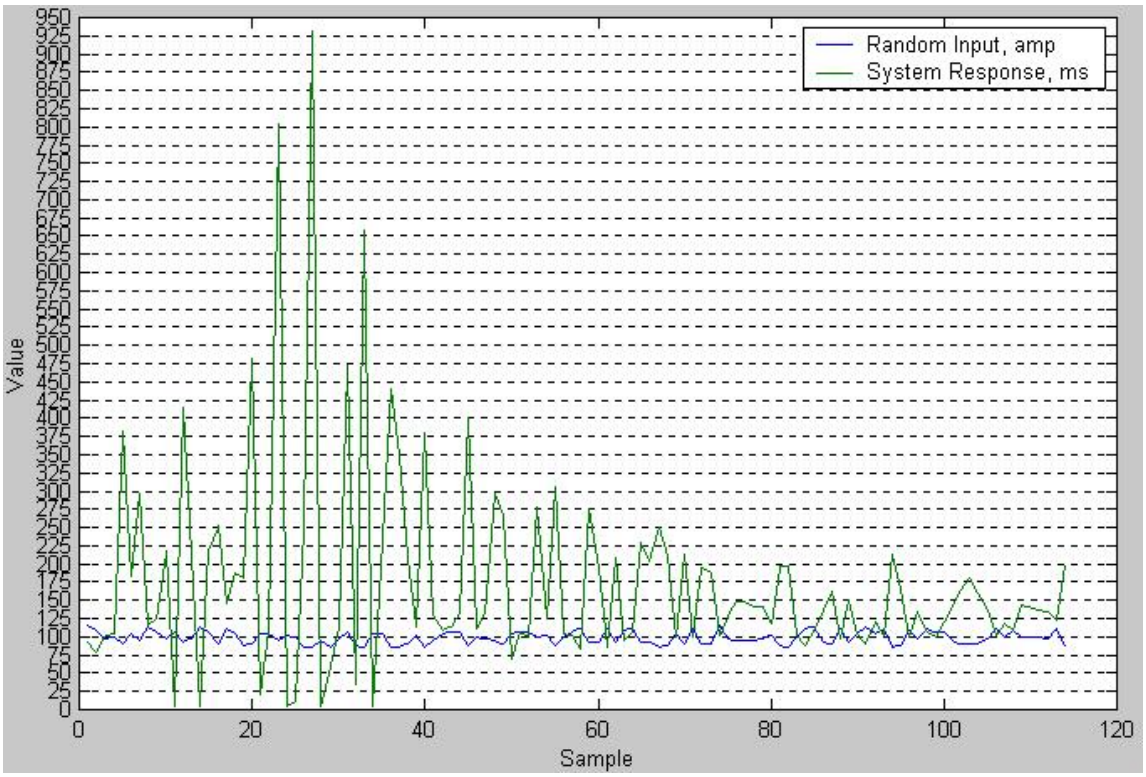


Figure 9-2 Test Run 2

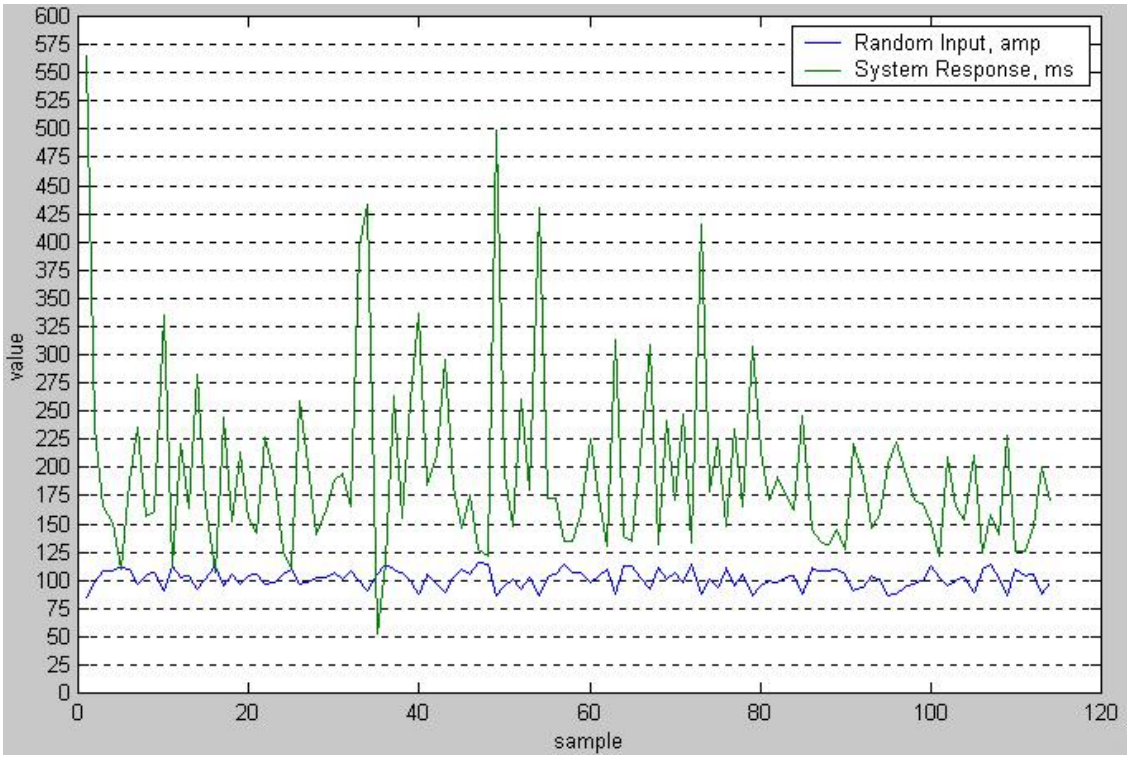


Figure 9-3 Test Run 3

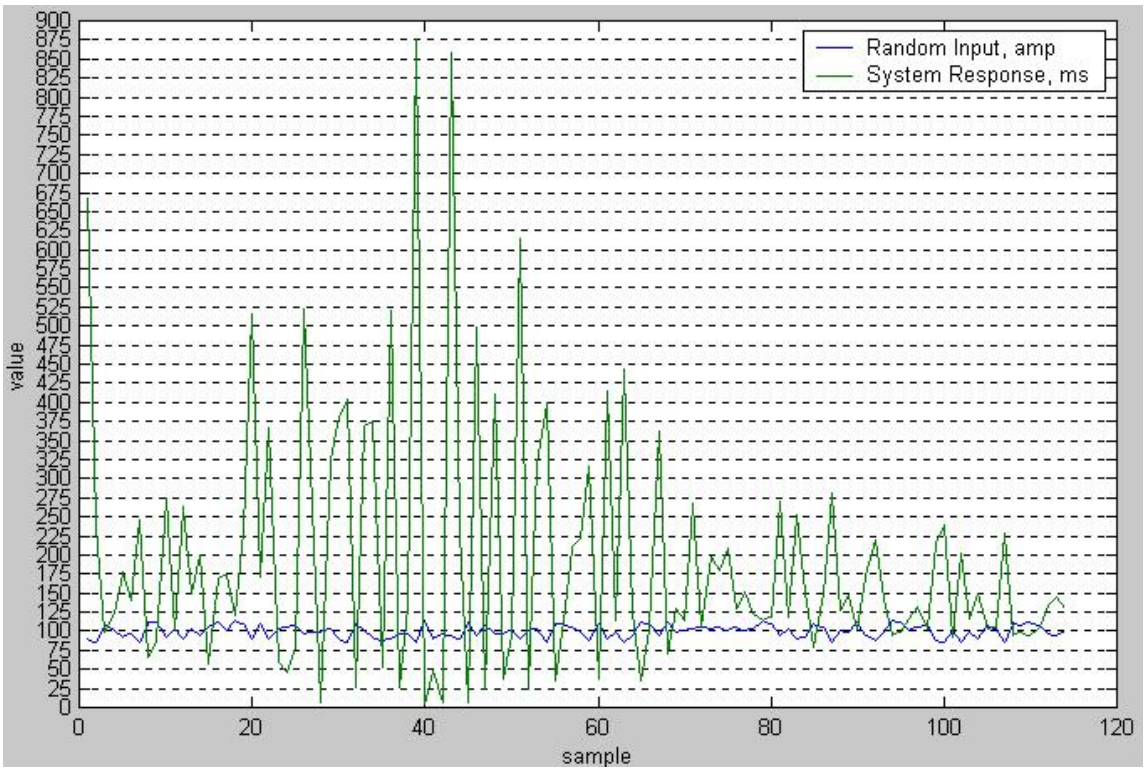


Figure 9-4 Test Run 4

From these sets of parameters, the minimum and maximum value for each parameter were defined, which in turn are then used in the Non-Linear Interval Control algorithm. Within the control algorithm, the set of system parameters that produces the maximum system response after each A/D data sample is selected in order to determine the minimum change in the control signal for the next iteration. The following minimum and maximum system parameters were determined:

Table 9-1 System Parameter Bounds

System Parameter	Minimum Value	Maximum Value
A_0	912.594537112136	1292.04854490696
A_1	-10.1609654622258	-6.94168976627553
A_2	-0.00424252156547636	-0.000418976974092593
A_3	-0.00211824837968971	0.000588460632832352

Photos of the actual test runs are shown below:

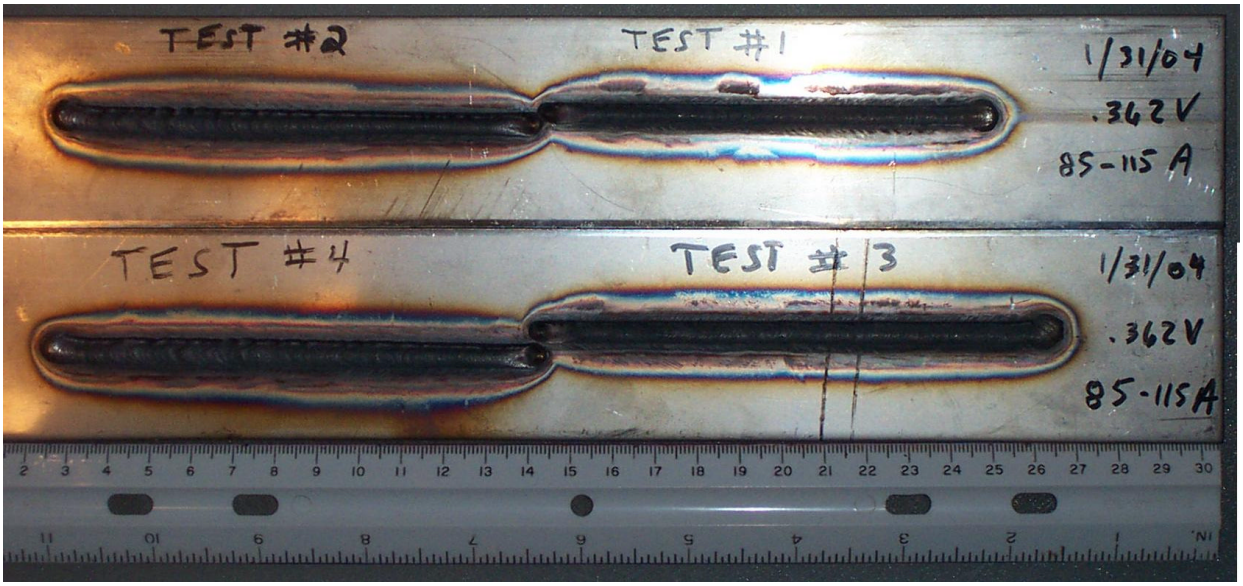


Figure 9-5 Topside All Four Test Runs

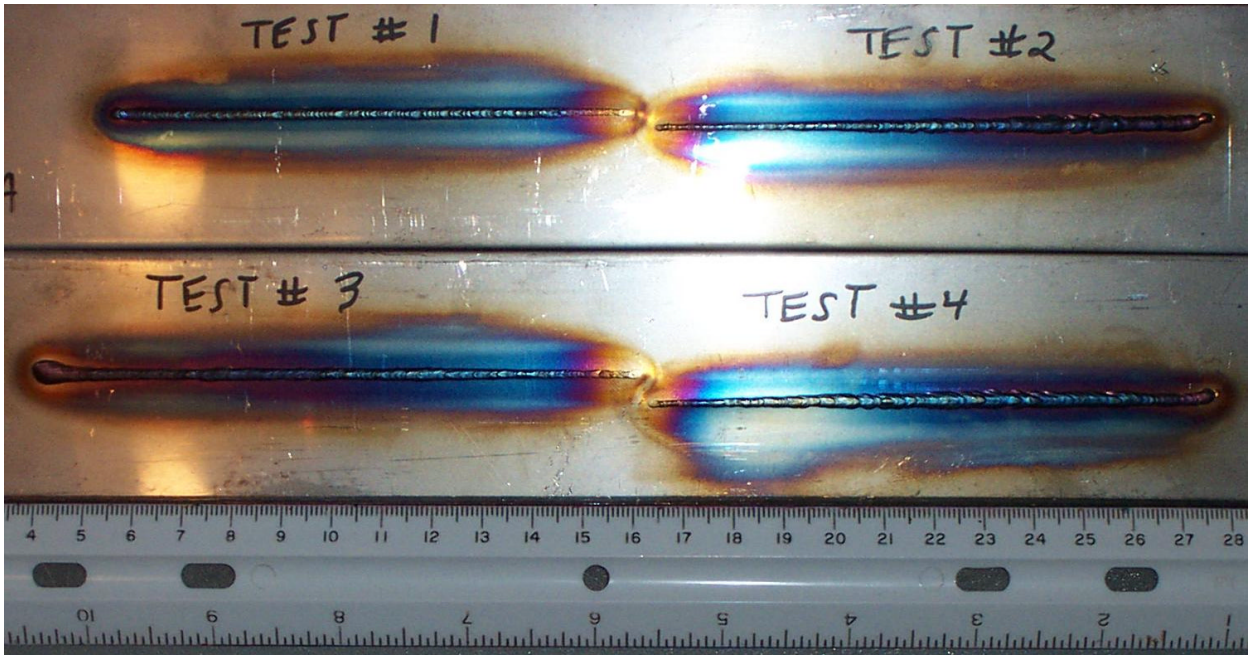


Figure 9-6 Bottom Side All Four Test Runs

Chapter Ten

Implementation

10.1 Implementation Setup

In order to embed the Non-Linear Interval Control Algorithm, the AED_109_32d module should be replaced with the NONLinearInterval_delay_quicker program. In order for the control to work properly, external variables independent of the control process must be organized in a known, stable, and repetitive condition. These variables are established in the following configuration: 2.4 mm nozzle diameter; 2.534 mm/sec travel speed equating to a 0.362 volts servo reference voltage; argon shielding gas pressure, 35 CFH middle of ball; argon backing gas pressure, 35 CFH middle of ball; argon plasma jet pressure, 4 CFH bottom of ball; 5 mm nozzle height above work piece; 304 stainless steel work piece material; 3 mm work piece thickness; and a peak current operating range between 85 and 115 amperes. If these external variables are changed for any reason, a new set of system parameters will be required for the control algorithm to work effectively.

10.2 Analog Output Initialization

When the AED-109 starts up, the initial quantization provided to the THS5661 from the FPGA is all zeros. Unfortunately, this corresponds to a –1.0 volts output on the AO. Since the gain of the external amplifier is set at –9.24, the effective control voltage to the Miller Electric Maxtron 450 CC/CV power supply is 9.24 volts. If the Miller power supply powered up, this would equate to a peak current level of 508.2 Amperes, which would immediately damage the equipment. As a result, the iteration that provides for a termination method if a keyhole is not achievable is normally set to increment the counter every 2000 iterations; however, at startup, the iteration interval should be set to a very small number. Therefore, the Miller supply is left powered down and this program is executed for a very short duration. Once this program starts executing, the final output is configured in such a manner as to produce a zero output on the AO. At the application termination, the AO will be outputting zero. At this point, the number of iterations before incrementing should be reset to 2000 in the conditional loop containing the counter called skip. The control implementation can now commence without damaging the equipment.

10.3 Control Results

Once the system parameters determined in the parameter estimation program have been encoded, testing of the actual Non-Linear Interval Control Algorithm can begin. Before describing the results, it is important to realize that different hardware characteristics can have significant effects on the results. The time constant of the analog output is of particular importance. The most effective AO is one with a very short time constant; however, if the time constant is relatively long, the base time will have to be substantially longer to allow the AO to settle and the heat to dissipate from the system. The best weld quality was determined for this hardware application to occur at a reference peak time of 325 ms and a base time of 400 ms. For shorter base times, the reference time will decrease substantially in order to dissipate the heat

that has been absorbed by the work piece. However, the peak current will tend to grow to establish a keyhole in a shorter time duration, with the net effect of reducing the stability of system and the overall weld quality. The results at the ideal reference peak and base time are shown initially followed by a shorter less ideal reference peak and base time as follows:

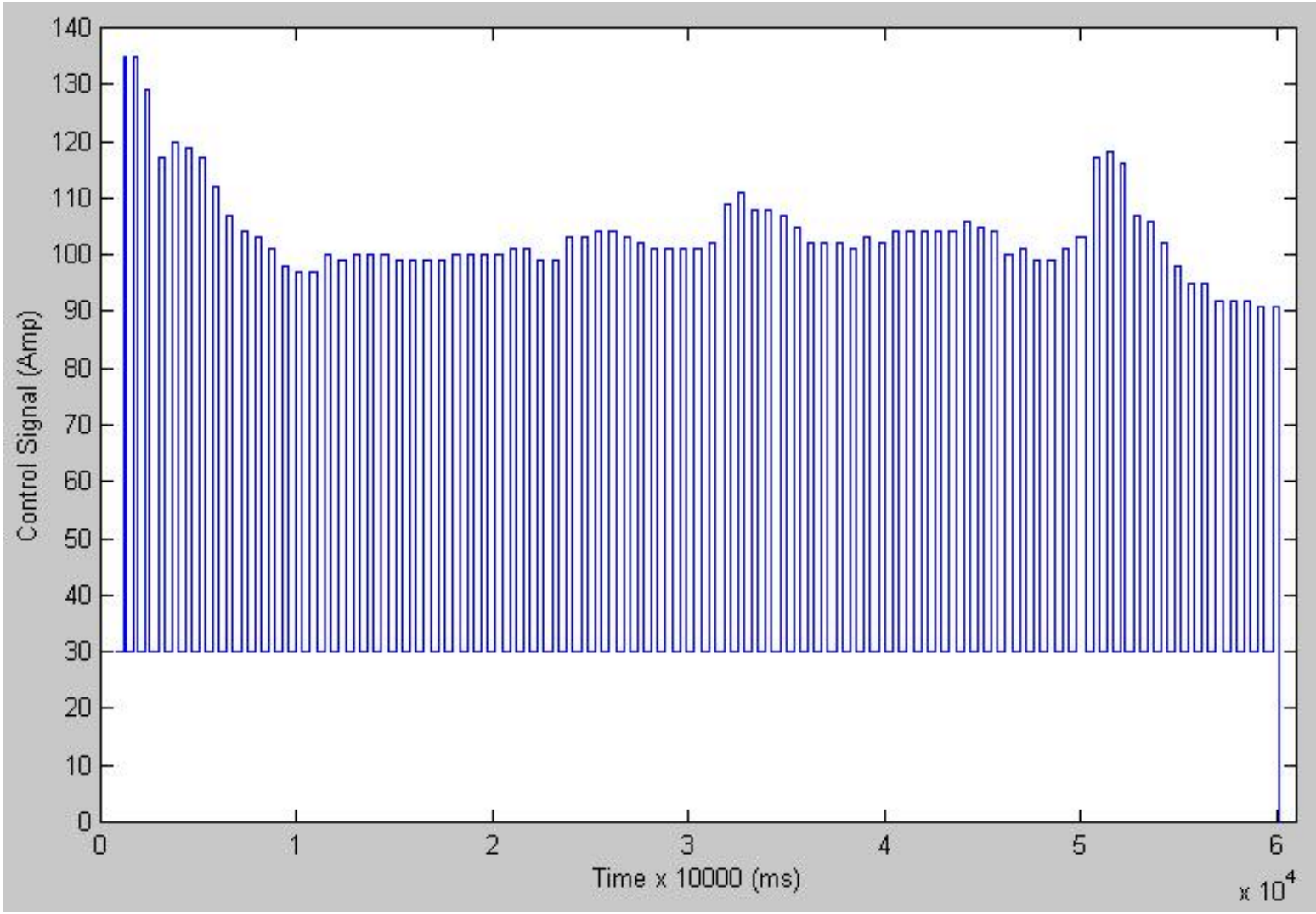


Figure 10-1 Control Signal, Peak Time $_{ref}$ = 325 ms, Base Time = 400 ms, Start Peak Current = 135 A

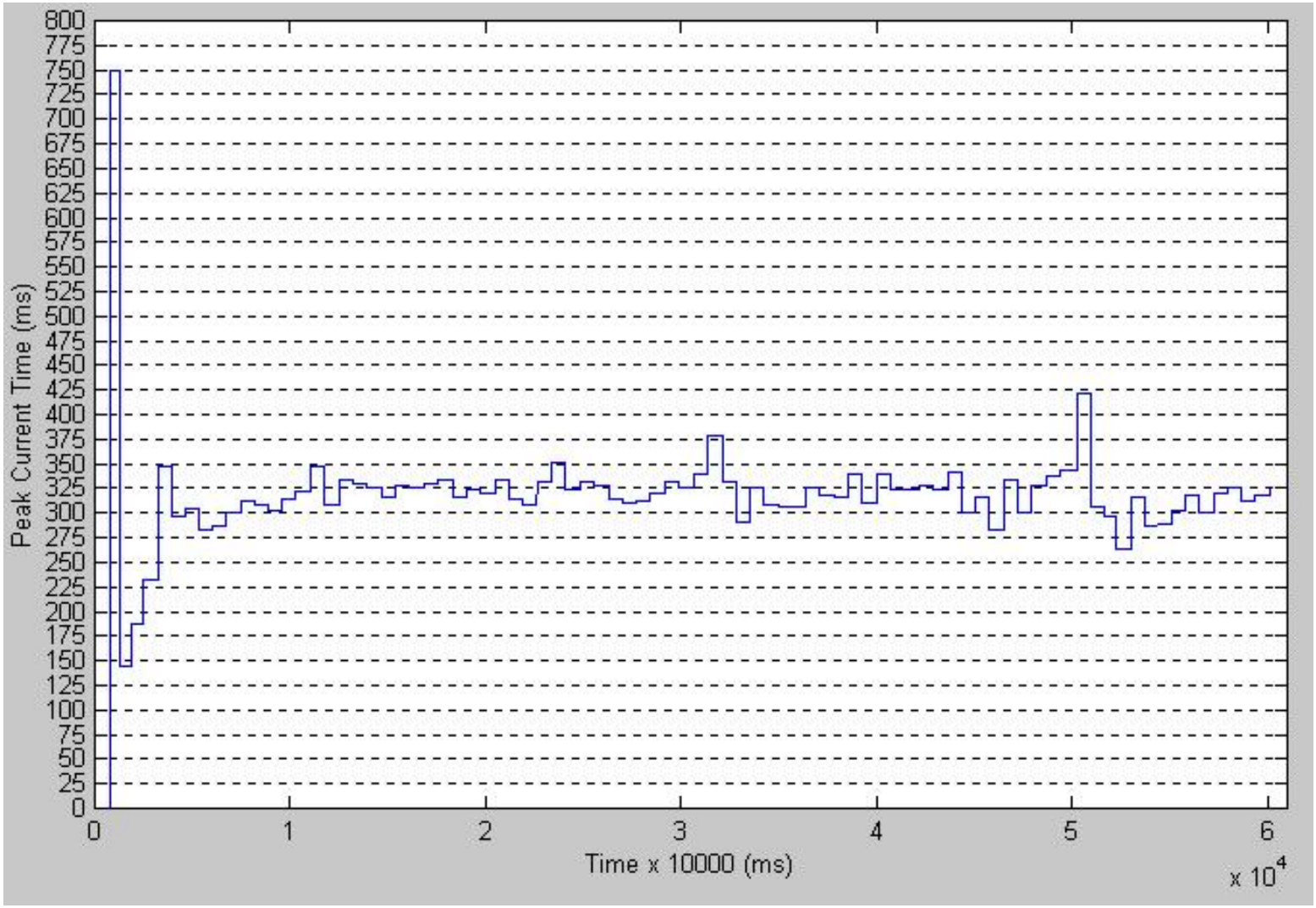


Figure 10-2 Peak Current Time, Peak Time $_{ref}$ = 325 ms, Base Time = 400 ms, Start Peak Current = 135 A

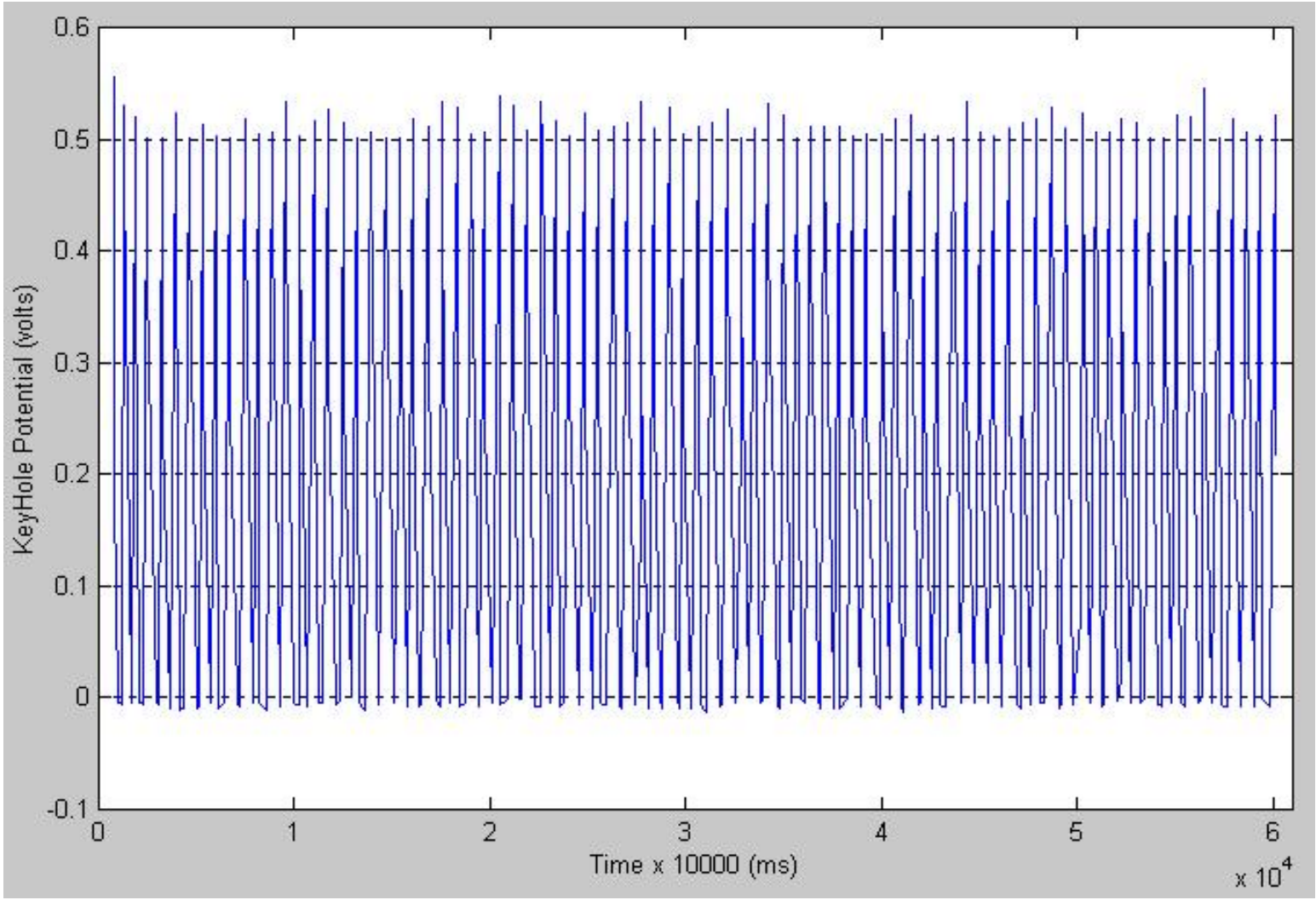


Figure 10-3 Keyhole Potential, Peak Time $_{ref}$ = 325 ms, Base Time = 400 ms, Start Peak Current = 135 A

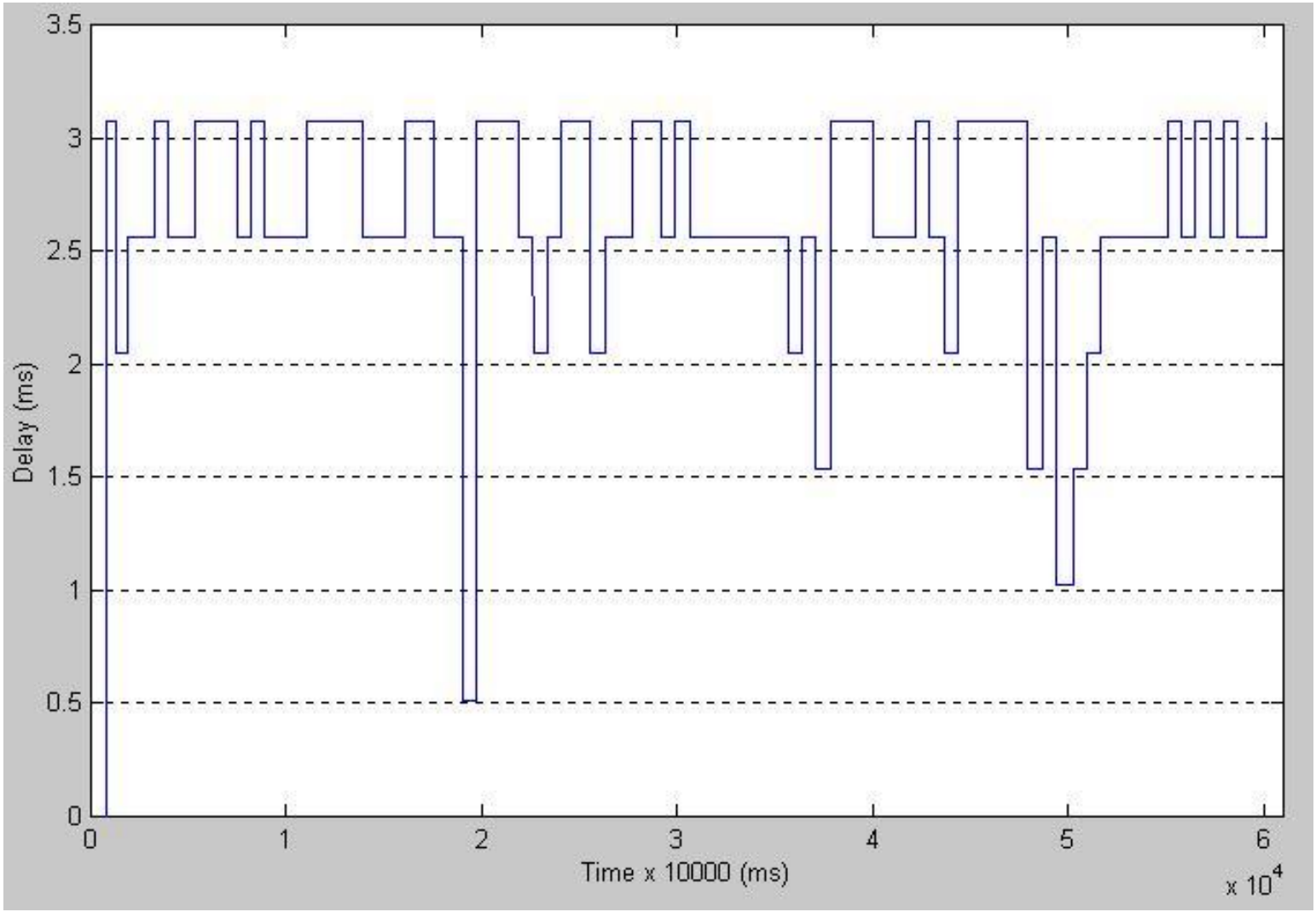


Figure 10-4 Delay, Peak Time $_{ref}$ = 325 ms, Base Time = 400 ms, Start Peak Current = 135 A

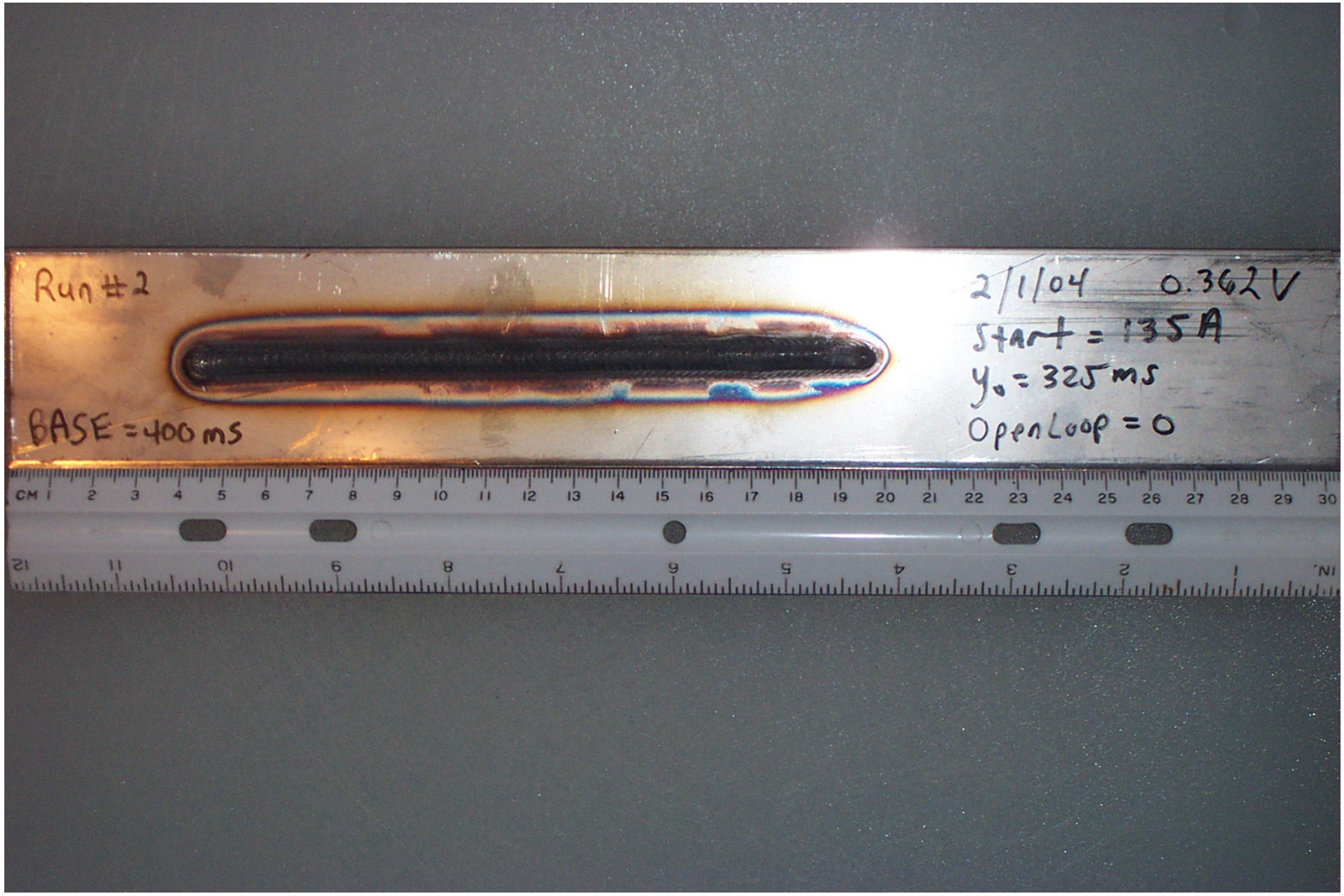


Figure 10-5 Topside Work Piece, Peak Time $_{ref}$ = 325 ms, Base Time = 400 ms, Start Peak Current = 135 A

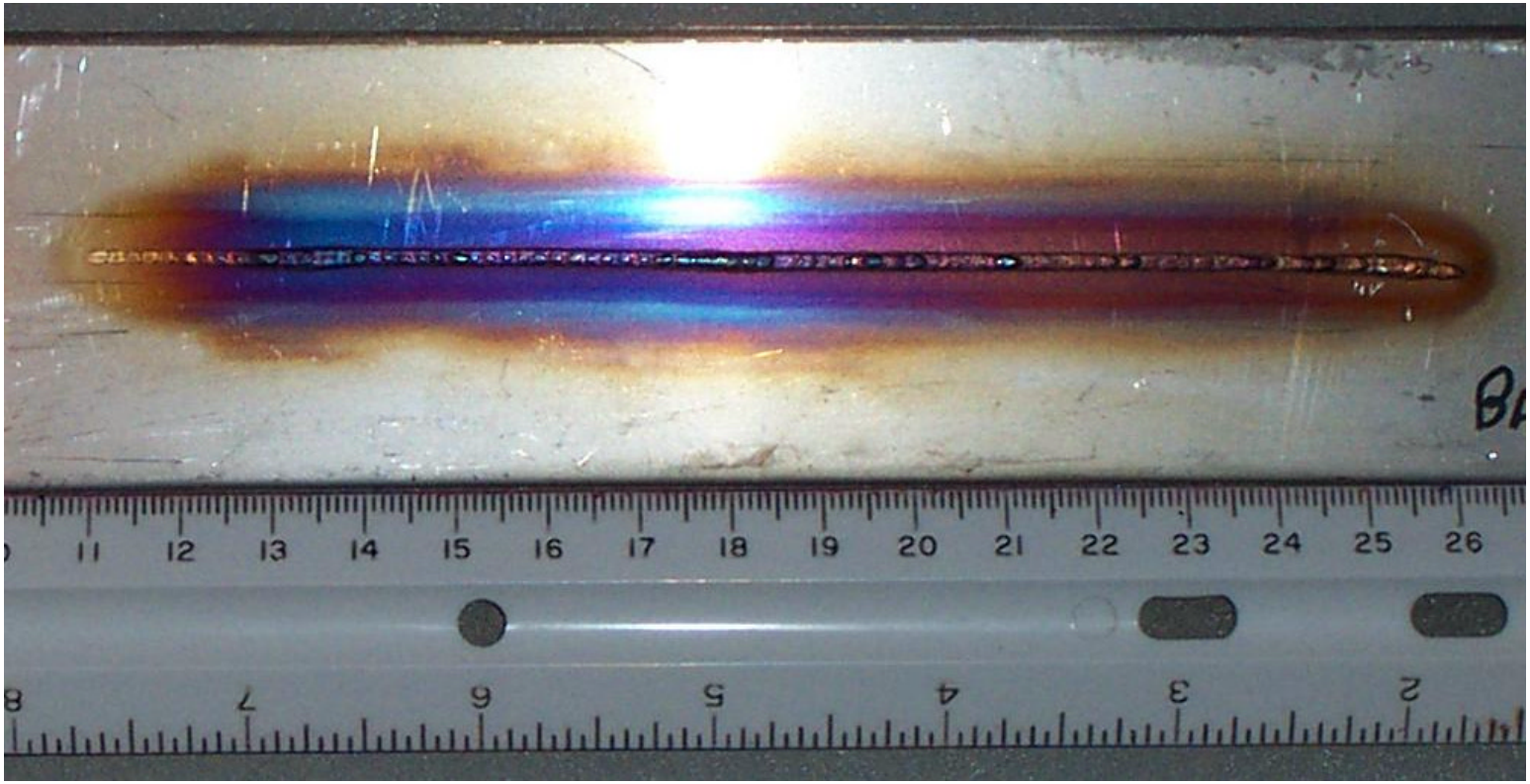


Figure 10-6 Bottom Side Work Piece, Peak Time $_{ref}$ = 325 ms, Base Time = 400 ms, Start Peak Current = 135 A

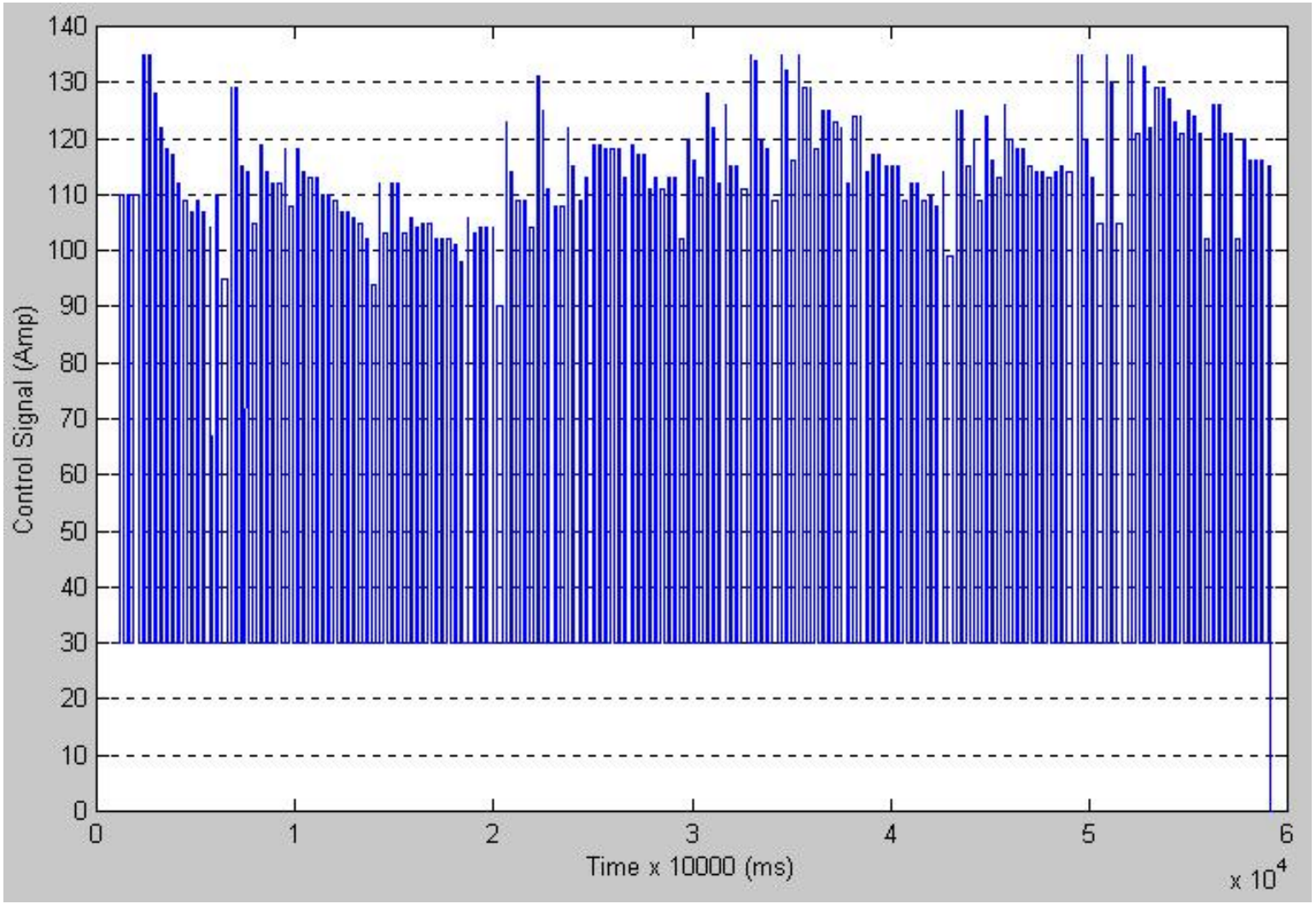


Figure 10-7 Control Signal, Peak Time $_{ref}$ = 125 ms, Base Time = 200 ms, Start Peak Current = 110 A

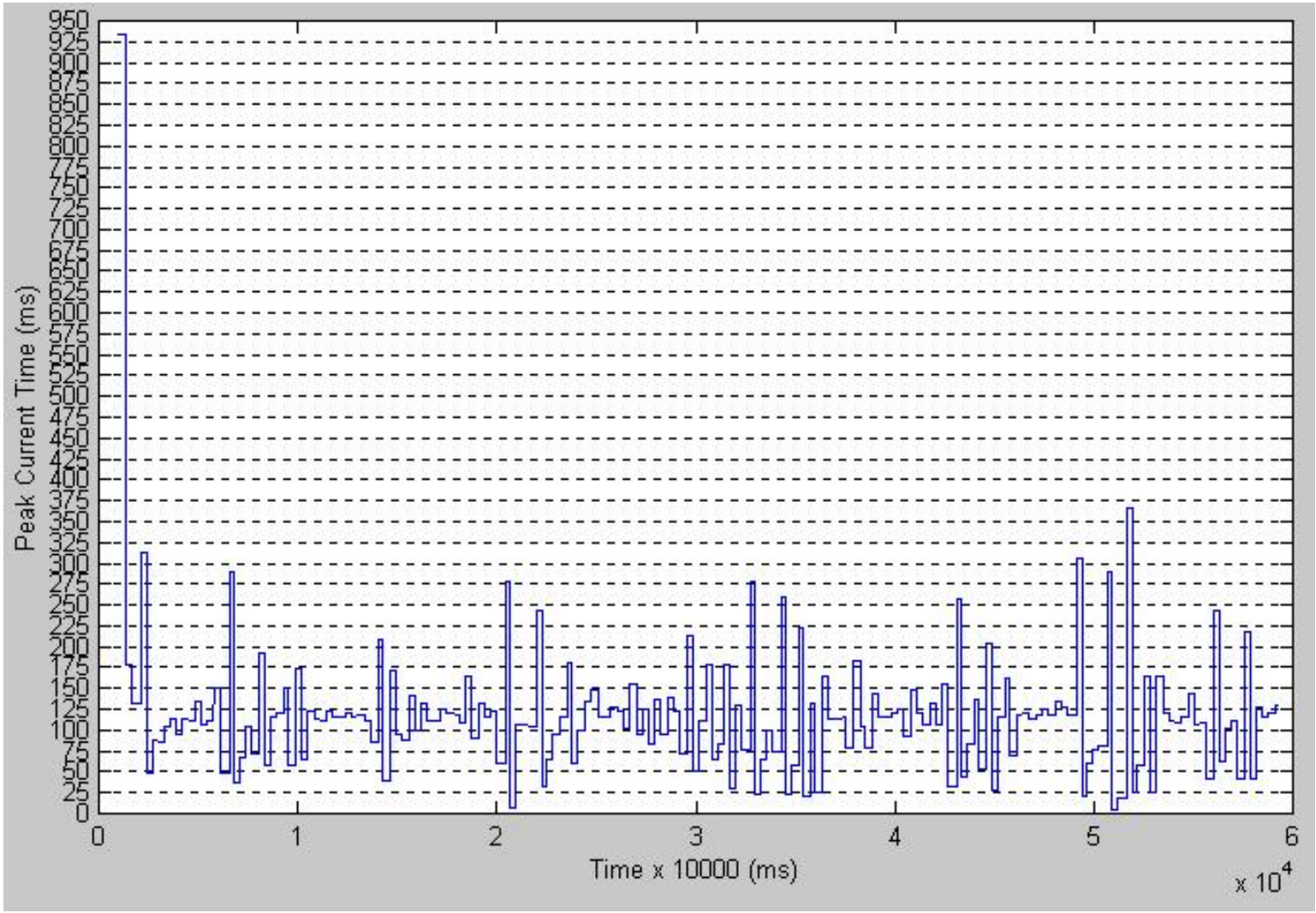


Figure 10-8 Peak Current Time, Peak Time $_{ref}$ = 125 ms, Base Time = 200 ms, Start Peak Current = 110 A

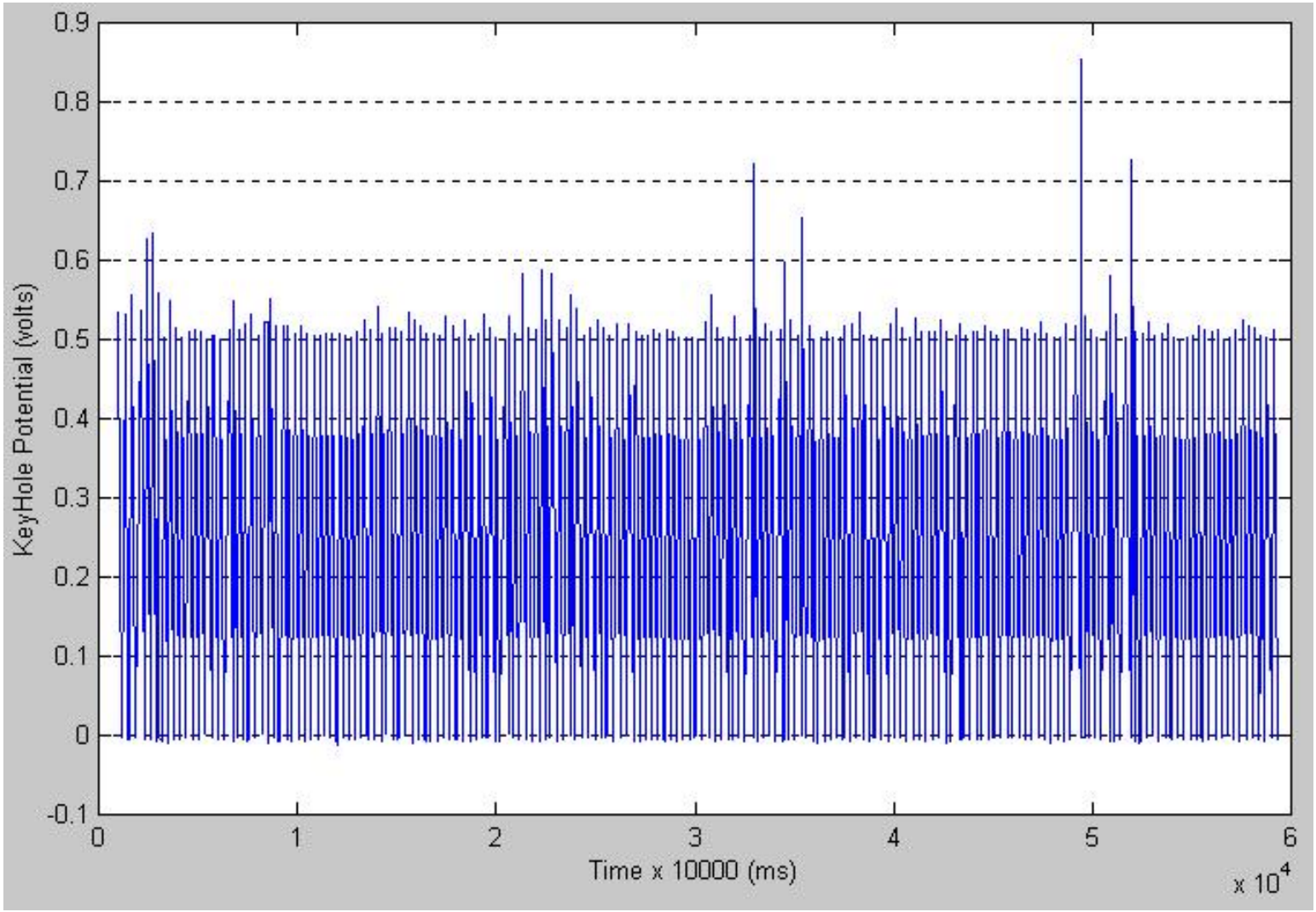


Figure 10-9 Keyhole Potential, Peak Time $_{ref}$ = 125 ms, Base Time = 200 ms, Start Peak Current = 110 A

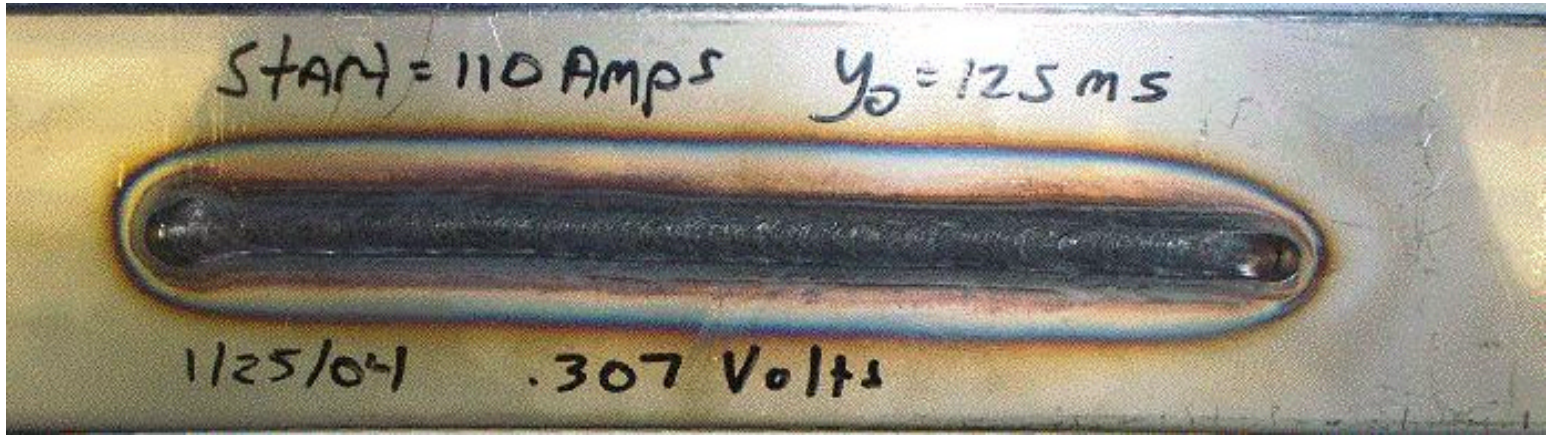


Figure 10-10 Topside Work Piece, Peak Time $_{ref}$ = 125 ms, Base Time = 200 ms, Start Peak Current = 110 A

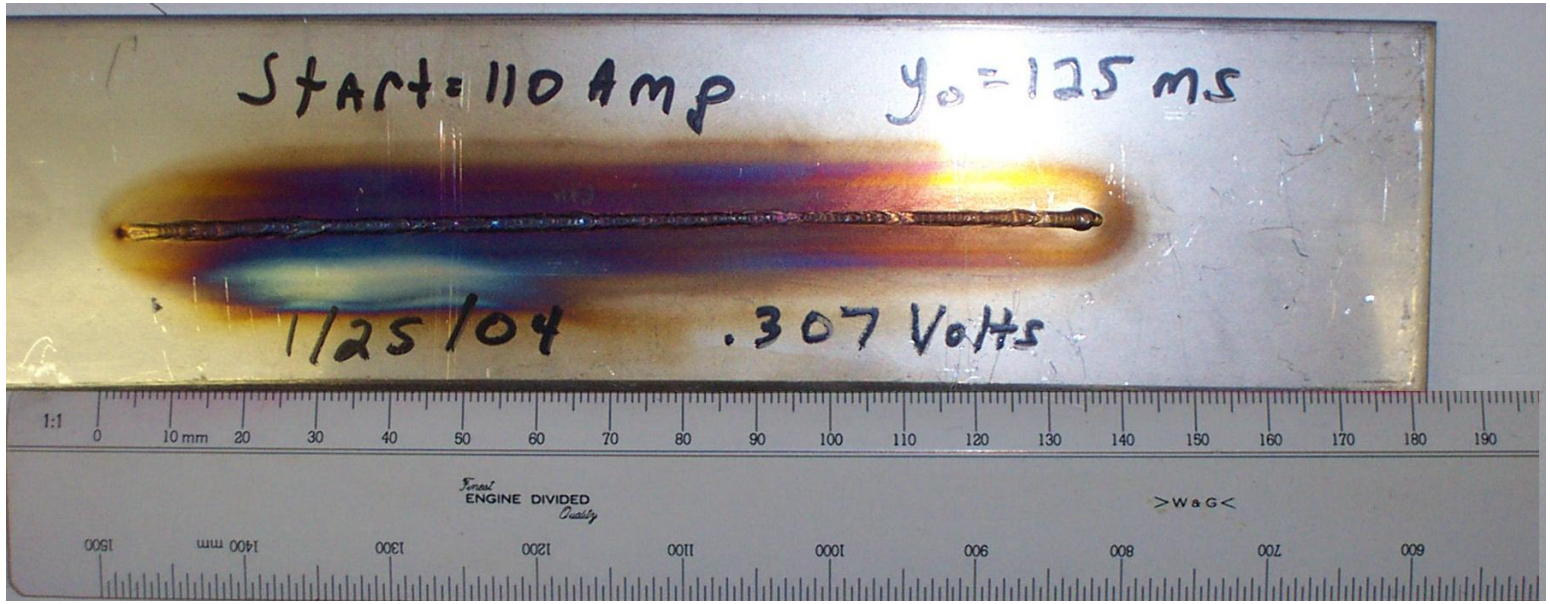


Figure 10-11 Bottom Side Work Piece, Peak Time $_{ref}$ = 125 ms, Base Time = 200 ms, Start Peak Current = 110 A

Chapter Eleven

Conclusion

11.1 Accomplishment

The stated goal of this project was to implement the non-linear interval control algorithm in regards to the Quasi-KeyHole plasma arc welding (PAW) process utilizing a TMS320VC5416 DSK in conjunction with an AED-109 data converter. By configuring the control process to operate with a 325 ms reference peak current time and a static 400 ms base current time, very stable repetitive results with an excellent weld quality were achieved. It was also demonstrated that each implementation is not only dependent upon the process and control algorithm, but the digital hardware choices as well. With smaller analog output response times, the base current period can be significantly reduced; however, the weld quality is highly dependent upon the ability of embedded control process to dissipate heat effectively and in such a manner that the peak current control signal is very stable. If the system response is stable, but the control signal is erratic, a low weld quality will result.

Although somewhat complex to configure, the C5416 does provide a powerful tool with a wealth of options to implement control processes. In comparison to other hardware choices such as the Micro-Processor or Micro-Controller, the apparent main drawback would the complexity of the configuration process with other factors such as cost or power consumption possibly being of concern depending on the application. However, for welding control applications such as the PAW process, the definitive results are usually of primary emphasis. Once the configuration process has been conquered, the C5416 is an excellent choice due to its expandability and speed of processing.

11.2 Additional Features

Realistically, the PAW process only requires one analog input and a single analog output capable of sampling in the KHz range to be successful. This implementation does take advantage of the stability offered by the differential analog input port and processing speed, but a wealth of options supplied by the combination of the C5416 DSK and AED-109 Daughter Card are not being utilized. Options such as configurable Digital I/O, HPI ports for external data storage, on-board audio I/O, and programmable flash for boot loading are not being utilized in this implementation.

The 16-bit Digital I/O configurable as 12 outputs and 4 inputs or 12 inputs and 4outputs could be used in a number of ways. For example, the digital inputs could be used to program pre-set reference peak and base times, or simply to recognize threshold inputs for signaling purposes. If a 12-bit digital output pattern was configured, the 12-bit quantization pattern for either the A/D or D/A could be output for external observation or storage. Other digital output possibilities include the control of relays, lights, or buzzers for a multitude of purposes. As a result, the existing control module NONLinearInterval_delay_quicker has the required memory-mapped registers provided in its code to allow for future Digital I/O implementations if desired.

The HPI port provided on the C5416 DSK is definitely an underused asset. The present control methodology is configured to store system information in arrays saved in on-board SRAM. This practice is functional for the desired observations; however, this process increases the code size, which effectively slows down processing times and limits the amount of data that may be stored. If the HPI port had been utilized as an avenue for external storage, the amount of data stored could be greatly increased by simply observing on-board or on-chip memory locations.

The on-board audio ports, although not capable of higher sampling rates such as AED-109, are still an effective means of interfacing between the analog and digital worlds. In the current configuration, the control signal is assumed to be in compliance with the quantization that is generated by the AED-109. As a result, a good use of an additional lower frequency analog input would be to measure the actual control signal current that is being generated by the Miller power supply to verify compliance. Another good example would be to use additional analog input ports for multiple system output systems, while realizing that the same logic could be applied for analog outputs as well. Regardless of the application, the combination of AED-109 and C5416 DSK analog I/O would allow for a variety of other configurations that are not being utilized in this control.

Finally, the programmable Flash is an interesting feature for stand-alone environments such is common in applications such as appliances, cell phones, or factory environments. The programmable flash provides a method of loading not volatile boot-loadable programs upon start up, independent of outside influences. This configuration is very useful when in a production or non-testing mode; however, in the laboratory environment, such is the case in this application where observation, modification, speed, and troubleshooting are at premium, flash configurations are not recommended. Flash has the added disadvantage of memory accessing times, which are significantly slower then RAM. As a result, real-time applications tend to avoid flash for any other application other then initial boot loading into RAM at start up.

11.3 Final Thoughts

If the DSP venue is chosen as the tool of choice for future embedded control processes, there are many advanced features that could be pursued for future applications. The driving motivation for most modifications is the desire for greater processing speed. For example in the case of the C5416, no math co-processor is provided. This tends to slow processing speeds in mathematical intensive operations. As a result, other DSP's such as the TMS320C67x actually incorporate math co-processors, which are very useful in floating-point hardware manipulation. In addition, clock speeds in the range of 1 GHz are now available for the C6000 series DSP's, which is significantly faster then the 160 MHz provided by the C5416. Other process specific aspects could also be pursued. A good example would be the analog output response times, which are of critical importance to Quasi-KeyHole PAW process. In this case, providing the manufacturer with the required operational time constants could greatly increase the overall system stability and weld quality.

In conclusion, the TMS320VC5416 DSK used in conjunction with the AED-109 data converter has been shown to be an effective tool in implementing feedback control algorithms.

With the robust configuration options provided by this system, the existing implementation could be easily modified for other feedback or data processing applications. With the modern demands of faster, smaller, cheaper, the DSP solution is and will continue to be a fixture concerning signal-processing applications for the foreseeable future.

Appendix A

TMS320VC5416 DSK Registers

A.1 CPU Registers [19]

The C5416 DSK has 27 memory-mapped CPU registers that are mapped into Data memory space addresses $0h_{16}$ to $1Fh_{16}$. The following is a list of available memory-mapped registers (MMRs):

Table A-1 CPU Memory-Mapped Registers

Name	Dec	Hex	Description
IMR	1	0	Interrupt Mask Register
IFR	2	1	Interrupt Mask Register
-	3-5	2-5	Reserved for Testing
STO	6	6	Status Register 0
ST1	7	7	Status Register 1
AL	8	8	Accumulator A Low Word (15-0)
AH	9	9	Accumulator A High Word (31-16)
AG	10	A	Accumulator A Guard Bits (39-32)
BL	11	B	Accumulator B Low Word (15-0)
BH	12	C	Accumulator B High Word (31-16)
BG	13	D	Accumulator B Guard Bits (39-32)
TREG	14	E	Temporary Register
TRN	15	F	Transition Register
AR0	16	10	Auxiliary Register 0
AR1	17	11	Auxiliary Register 1
AR2	18	12	Auxiliary Register 2
AR3	19	13	Auxiliary Register 3
AR4	20	14	Auxiliary Register 4
AR5	21	15	Auxiliary Register 5
AR6	22	16	Auxiliary Register 6
AR7	23	17	Auxiliary Register 7
SP	24	18	Stack Pointer Register
BK	25	19	Circular Buffer Size Register
BRC	26	1A	Block Repeat Counter
RSA	27	1B	Block Repeat Start Address
REA	28	1C	Block Repeat End Address
PMST	29	1D	Processor Mode Status (PMST) Register
XPC	30	1E	Extended Program Page Register
-	31	1F	Reserved

A.1.1 Status Registers [19]

The C5416 contains three status registers: ST0; ST1; and PMST. Each status register is further divided into several distinct fields. Although each field is often thought of as a separate register, it is not possible to access these fields individually; therefore, in order to set one field, it is necessary to set all of the fields within the same status register. The ST0 fields are listed as follows:

- Auxiliary Register Pointer (ARP)
- Carry Bit (C)
- Data Page Pointer (DP)
- Overflow Flag for Accumulator A (OVA)
- Overflow Flag for Accumulator B (OVB)
- Test/Control Flag (TC)

The ST1 fields are listed below:

- Accumulator Shift Mode (ASM)
- Block Repeat Active Bit (BRAF)
- Dual 16-bit Math Bit (C16)
- Compatibility Mode Bit (CMPT)
- Compiler Mode Bit (CPL)
- Fractional Mode Bit (FRCT)
- Hold Mode Bit (HM)
- Interrupt Mask (INTM)
- Overflow Mode Bit (OVM)
- Fractional Mode Bit (SXM)
- External Flag (XF)

The PMST is divided as shown below:

- Address Visibility Bit (AVIS)
- CLKOUT Disable Bit (CLKOFF)
- Map ROM into Data Space (DROM)
- Interrupt Vector Table Pointer (IPTR)
- Micro-Processor/Micro-Controller Mode Bit (MP/MC)
- RAM Overlay Bit (OVLY)
- Saturation on Multiply Bit (SMUL)
- Saturation on Store (SST)

A.2 Peripheral Memory-Mapped Registers [19]

The C5416 uses Peripheral Memory-Mapped Register to control and configure on-chip peripherals. These registers are listed as follows:

Table A-2 Peripheral Memory-Mapped Registers

Name	Dec	Hex	Description
DRR20	32	20	McBSP 0 Data Receive Register 2
DRR10	33	21	McBSP 0 Data Receive Register 1
DXR20	34	22	McBSP 0 Data Transmit Register 2
DXR10	35	23	McBSP 0 Data Transmit Register 1
TIM	36	24	Timer Register
PRD	37	25	Timer Period Register

Table A-2 Peripheral Memory-Mapped Registers (Continued)

Name	Dec	Hex	Description
TCR	38	26	Timer Control Register
-	39	27	Reserved
SWWSR	40	28	Software Wait-State Register
BSCR	41	29	Bank-Switching Control Register
-	42	2A	Reserved
SWCR	43	2B	Software Wait-State Control Register
HPIC	44	2C	HPI Control Register (HMODE = 0 only)
-	45-47	2D-2F	Reserved
DRR22	48	30	McBSP 2 Data Receive Register 2
DRR12	49	31	McBSP 2 Data Receive Register 1
DXR22	50	32	McBSP 2 Data Transmit Register 2
DXR12	51	33	McBSP 2 Data Transmit Register 1
SPSA2	52	34	McBSP 2 Sub-bank Address Register
SPSD2	53	35	McBSP 2 Sub-bank Data Register
-	54-55	36-37	Reserved
SPSA0	56	38	McBSP 0 Sub-bank Address Register
SPSD0	57	39	McBSP 0 Sub-bank Data Register
-	58-59	3A-3B	Reserved
GPIOCR	60	3C	General Purpose I/O Control Register
GPIOSR	61	3D	General Purpose I/O Status Register
CSIDR	62	3E	Device ID Register
-	63	3F	Reserved
DRR21	64	40	McBSP 1 Data Receive Register 2
DRR11	65	41	McBSP 1 Data Receive Register 1
DXR21	66	42	McBSP 1 Data Transmit Register 2
DXR11	67	43	McBSP 1 Data Transmit Register 1
-	68-71	44-47	Reserved
SPSA1	72	48	McBSP 1 Sub-bank Address Register
SPSD1	73	49	McBSP 1 Sub-bank Data Register
-	74-83	4A-53	Reserved
DMPREC	84	54	DMA Priority and Enable Control Register
DMSA	85	55	DMA Sub-bank Address Register
DMSDI	86	56	DMA Sub-bank Data Register with Autoincrement
DMSDN	87	57	DMA Sub-bank Data Register
CLKMD	88	58	Clock Mode Register (CLKMD)
-	89-95	59-5F	Reserved

A.3 CPLD Registers [19]

The main Board Setup Library (BSL) header file dsk5416.h includes definitions for the I/O mapped CPLD registers. It is acceptable practice to make assignments with ioport variables just like any other variable. The eight CPLD registers are listed below:

- DSK5416_USER_REG, 0
- DSK5416_DC_REG, 1
- DSK5416_PCM3002_L, 2
- DSK5416_PCM3002_H, 3
- DSK5416_VERSION, 4
- DSK5416_DM_CNTL, 5

- DSK5416_MISC, 6
- DSK5416_CODEC_CLK, 7

A.4 McBSP Control Registers and Sub-Addresses [19]

The control registers for the multichannel buffered serial port (McBSP) are accessed using the sub-bank addressing scheme. This allows a set or sub-bank of registers to be accessed through a single memory location. The McBSP sub-bank address register (SPSA) is used as a pointer to select a particular register within the sub-bank. The McBSP data register (SPSDx) is used to access (read or write) to the selected register.

Table A-3 McBSP Control Registers and Sub-Addresses

Name	Hex Add	Name	Hex Add	Name	Hex Add	Hex Sadd	Description
SPCR10	39	SPCR11	49	SPCR12	35	0	Serial Port Control Register 1
SPCR20	39	SPCR21	49	SPCR22	35	1	Serial Port Control Register 2
RCR10	39	RCR11	49	RCR12	35	2	Receive Control Register 1
RCR20	39	RCR21	49	RCR22	35	3	Receive Control Register 2
XCR10	39	XCR11	49	XCR12	35	4	Transmit Control Register 1
XCR20	39	SCR21	49	SCR22	35	5	Transmit Control Register 2
SRGR10	39	SRGR11	49	SRGR12	35	6	Sample Rate Generator Reg 1
SRGR20	39	SRGR21	49	SRGR22	35	7	Sample Rate Generator Reg 2
MCR10	39	MCR11	49	MCR12	35	8	Multichannel Control Register 1
MCR20	39	MCR21	49	MCR22	35	9	Multichannel Control Register 2
RCERA0	39	RCERA1	49	RCERA2	35	A	Receive Channel Enable Register Partition A
RCERB0	39	RCERB1	49	RCERB2	35	B	Receive Channel Enable Register Partition B
XCERA0	39	XCERA1	49	XCERA2	35	C	Transmit Channel Enable Register Partition A
XCERB0	39	XCERB1	49	XCERB2	35	D	Transmit Channel Enable Register Partition B
PCR0	39	PCR1	49	PCR2	35	E	Pin Control Register
RCERC0	39	RCERC1	49	RCERC2	35	10	Additional Channel Enable Register for 128-Channel Select
RCERD0	39	RCERD1	49	RCERD2	35	11	Additional Channel Enable Register for 128-Channel Select
XCERC0	39	XCERC1	49	XCERC2	35	12	Additional Channel Enable Register for 128-Channel Select
XCERD0	39	XCERD1	49	XCERD2	35	13	Additional Channel Enable Register for 128-Channel Select
RCERE0	39	RCERE1	49	RCERE2	35	14	Additional Channel Enable Register for 128-Channel Select
RCERF0	39	RCERF1	49	RCERF2	35	15	Additional Channel Enable Register for 128-Channel Select
XCERE0	39	SCERE1	49	SCERE2	35	16	Additional Channel Enable Register for 128-Channel Select
XCERF0	39	SCERF1	49	SCERF2	35	17	Additional Channel Enable Register for 128-Channel Select
RCERG0	39	RCERG1	49	RCERG2	35	18	Additional Channel Enable Register for 128-Channel Select
RCERH0	39	RCERH1	49	RCERH2	35	19	Additional Channel Enable

Table A-3 McBSP Control Registers and Sub-Addresses (Continued)

Name	Hex Add	Name	Hex Add	Name	Hex Add	Hex Sadd	Description
							Register for 128-Channel Select
XCERG0	39	XCERG1	49	XCERG2	35	1A	Additional Channel Enable Register for 128-Channel Select
XCERH0	39	XCERH1	49	XCERH2	35	1B	Additional Channel Enable Register for 128-Channel Select

A.5 DMA Sub-Bank Addressed Registers [19]

Table A-4 DMA Sub-Bank Addressed Registers

Name	Hex Add	Hex Sub-Add	Description
DMSRC0	56	0	DMA Channel 0 Source Address Register
DMDST0	56	1	DMA Channel 0 Destination Address Register
DMCTR0	56	2	DMA Channel 0 Element Count Register
DMSFC0	56	3	DMA Channel 0 Sync Select and Frame Count Register
DMMCR0	56	4	DMA Channel 0 Transfer Mode Control Register
DMSRC1	56	5	DMA Channel 1 Source Address Register
DMDST1	56	6	DMA Channel 1 Destination Address Register
DMCTR1	56	7	DMA Channel 1 Element Count Register
DMSFC1	56	8	DMA Channel 1 Sync Select and Frame Count Register
DMMCR1	56	9	DMA Channel 1 Transfer Mode Control Register
DMSRC2	56	A	DMA Channel 2 Source Address Register
DMDST2	56	B	DMA Channel 2 Destination Address Register
DMCTR2	56	C	DMA Channel 2 Element Count Register
DMSFC2	56	D	DMA Channel 2 Sync Select and Frame Count Register
DMMCR2	56	E	DMA Channel 2 Transfer Mode Control Register
DMSRC3	56	F	DMA Channel 3 Source Address Register
DMDST3	56	10	DMA Channel 3 Destination Address Register
DMCTR3	56	11	DMA Channel 3 Element Count Register
DMSFC3	56	12	DMA Channel 3 Sync Select and Frame Count Register
DMMCR3	56	13	DMA Channel 3 Transfer Mode Control Register
DMSRC4	56	14	DMA Channel 4 Source Address Register
DMDST4	56	15	DMA Channel 4 Destination Address Register
DMCTR4	56	16	DMA Channel 4 Element Count Register
DMSFC4	56	17	DMA Channel 4 Sync Select and Frame Count Register
DMMCR4	56	18	DMA Channel 4 Transfer Mode Control Register
DMSRC5	56	19	DMA Channel 5 Source Address Register
DMDST5	56	1A	DMA Channel 5 Destination Address Register
DMCTR5	56	1B	DMA Channel 5 Element Count Register
DMSFC5	56	1C	DMA Channel 5 Sync Select and Frame Count Register
DMMCR5	56	1D	DMA Channel 5 Transfer Mode Control Register
DMSRCP	56	1E	DMA Source Program Page Address (common channel)
DMDSTP	56	1F	DMA Dest Program Page Address (common channel)
DMIDX0	56	20	DMA Element Index Address Register 0
DMIDX1	56	21	DMA Element Index Address Register 1
DMFRI0	56	22	DMA Frame Index Register 0
DMFRI1	56	23	DMA Frame Index Register 1
DMGSA0	56	24	DMA Global Source Address Reload Register, Chan 0
DMGDA0	56	25	DMA Global Dest Address Reload Register, Chan 0

Table A-4 DMA Sub-Bank Addressed Registers (Continued)

Name	Hex Add	Hex Sub-Add	Description
DMGCR0	56	26	DMA Global Count Reload Register, Channel 0
DMGFR0	56	27	DMA Global Frame Count Reload Register, Channel 0
XSRCDP	56	28	DMA Ext Source Data Page (Currently Not Supported)
XDSTDP	56	29	DMA Ext Dest Data Page (Currently Not Supported)
DMGSA1	56	2A	DMA Global Source Address Reload Register, Chan 1
DMGDA1	56	2B	DMA Global Dest Address Reload Register, Chan 1
DMGCR1	56	2C	DMA Global Count Reload Register, Channel 1
DMGFR1	56	2D	DMA Global Frame Count Reload Register, Channel 1
DMGSA2	56	2E	DMA Global Source Address Reload Register, Chan 2
DMGDA2	56	2F	DMA Global Dest Address Reload Register, Chan 2
DMGCR2	56	30	DMA Global Count Reload Register, Channel 2
DMGFR2	56	31	DMA Global Frame Count Reload Register, Channel 2
DMGSA3	56	32	DMA Global Source Address Reload Register, Chan 3
DMGDA3	56	33	DMA Global Dest Address Reload Register, Chan 3
DMGCR3	56	34	DMA Global Count Reload Register, Channel 3
DMGFR3	56	35	DMA Global Frame Count Reload Register, Channel 3
DMGSA4	56	36	DMA Global Source Address Reload Register, Chan 4
DMGDA4	56	37	DMA Global Dest Address Reload Register, Chan 4
DMGCR4	56	38	DMA Global Count Reload Register, Channel 4
DMGFR4	56	39	DMA Global Frame Count Reload Register, Channel 4
DMGSA5	56	3A	DMA Global Source Address Reload Register, Chan 5
DMGDA5	56	3B	DMA Global Dest Address Reload Register, Chan 5
DMGCR5	56	3C	DMA Global Count Reload Register, Channel 5
DMGFR5	56	3D	DMA Global Frame Count Reload Register, Channel 5

A.6 Interrupts [19]

Table A-5 Interrupts

Name	Trap	Dec	Hex	Pr	Description
RS', SINTR	1	0	0	1	Reset (Hardware and Software Reset)
NMI', SINT16	2	4	4	2	Non-Maskable Interrupt
SINT17	3	8	8	-	Software Interrupt #17
SINT18	4	12	C	-	Software Interrupt #18
SINT19	5	16	10	-	Software Interrupt #19
SINT20	6	20	14	-	Software Interrupt #20
SINT21	7	24	18	-	Software Interrupt #21
SINT22	8	28	1C	-	Software Interrupt #22
SINT23	9	32	20	-	Software Interrupt #23
SINT24	10	36	24	-	Software Interrupt #24
SINT25	11	40	28	-	Software Interrupt #25
SINT26	12	44	2C	-	Software Interrupt #26
SINT27	13	48	30	-	Software Interrupt #27
SINT28	14	52	34	-	Software Interrupt #28
SINT29	15	56	38	-	Software Interrupt #29
SINT30	16	60	3C	-	Software Interrupt #30
INT0', SINT0	17	64	40	3	External User Interrupt #0
INT1', SINT1	18	68	44	4	External User Interrupt #1
INT2', SINT2	19	72	48	5	External User Interrupt #2
TINT, SINT3	20	76	4C	6	Timer Interrupt
RINT0, SINT4	21	80	50	7	McBSP #0 Receive Interrupt (default)

Table A-5 Interrupts (Continued)

Name	Trap	Dec	Hex	Pr	Description
XINT0, SINT5	22	84	54	8	McBSP #0 Transmit Interrupt (default)
RINT2, SINT6	23	88	58	9	McBSP #2 Receive Interrupt (default)
XINT2, SINT7	24	92	5C	10	McBSP #2 Transmit Interrupt (default)
ITE', SINT8	25	96	60	11	External User Interrupt #3
HINT', SINT9	26	100	64	12	HPI Interrupt
DMAC4, SINT10	27	104	68	13	McBSP #1 Receive Interrupt (default)
DMAC4, SINT11	28	108	6C	14	McBSP #1 Transmit Interrupt (default)
DMAC4, SINT12	29	112	70	15	DMA Channel 4 (default)
DMAC5, SINT13	30	116	74	16	DMA Channel 5 (default)
Reserved	30-31	120-127	78-7F	-	Reserved

Appendix B

Code Composer Studio Test Program Required Files

B.1 Required Files [6]

Table B-1 Code Composer Studio Required Files for Test Program Execution

GEL	Project	Linker Command	DSP/ BIOS	Generated	Include	Libraries	Source
C5416_dsk.gel	d5416dsk_AED.pjt	5416_lnkp.cmd			AED.h	drv5402f.lib	5416_dsk.c
					AED_Appl.h	dsk5416f.lib	AED_DMS_4wDMA.c
					AED_Brd.h	rts_ext.lib	AED_MAIN.c
					AED_Cfg.h		AED_109_32d.c
					AED_DMS.h		Vectors.asm
					dma5416.h		
					dsk5416.h		
					emif.h		
					intr5416.h		
					regs.h		
					regs5416.h		
					timr5416.h		

Appendix C

General Extension File (GEL)

C.1 C5416_dsk.gel [32]

```
/* set PMST to: OVLY on; DROM on, CLKOUT off */
#define PMST_VAL        0x7facu
/* set wait-state control reg for: 2 wait states, 4 for I/O */
#define SWWSR_VAL       0x4492u
/* set external-banks switch control for: set CONSEC and BH, CLKOUT/=2 */
#define BSCR_VAL        0xa002u
/* Set Default Reset Initialization Value */
#define ZEROS           0x0000u
/* Set CLKMD register to PLL multiplier of 10 */
#define CLKMD_VAL       0x9107u
/* Set Peripheral Control Register Addresses for DEV_RESET */
#define DMPREC          0x0054u
#define DMSA            0x0055u
#define DMSDI           0x0056u
#define DMA_CH0_DMFSC_SUB_ADDR  0x0003u
#define DMA_CH1_DMFSC_SUB_ADDR  0x0008u
#define DMA_CH2_DMFSC_SUB_ADDR  0x000Du
#define DMA_CH3_DMFSC_SUB_ADDR  0x0012u
#define DMA_CH4_DMFSC_SUB_ADDR  0x0017u
#define DMA_CH5_DMFSC_SUB_ADDR  0x001cu
#define MCBSP0_SPSA     0x0038u
#define MCBSP0_SPSD     0x0039u
#define MCBSP1_SPSA     0x0048u
#define MCBSP1_SPSD     0x0049u
#define MCBSP2_SPSA     0x0034u
#define MCBSP2_SPSD     0x0035u
#define MCBSP_SPCR1_SUB_ADDR   0x0000u
#define MCBSP_SPCR2_SUB_ADDR   0x0001u
#define MCBSP_SRGR1_SUB_ADDR   0x0006u
#define MCBSP_SRGR2_SUB_ADDR   0x0007u
#define MCBSP_MCR1_SUB_ADDR    0x0008u
#define MCBSP_MCR2_SUB_ADDR    0x0009u
#define SRGR1_INIT            0x0001u
#define PRD0            0x0025u
#define TCR0            0x0026u
#define PRD1            0x0031u
#define TCR1            0x0032u
#define TIMER_STOP            0x0010u
#define TIMER_RESET           0x0020u
#define PRD_DEFAULT           0xFFFFu
#define GPIOCR                0x0010u
/* The Startup() function is executed when the GEL file is loaded. */
StartUp()
{
C5416_DSK_Init();
GEL_TextOut("Gel StartUp complete.\n");
}
menuitem "C5416_DSK_Configuration";
hotmenu CPU_Reset()
```

```
{
GEL_Reset();
PMST = PMST_VAL;
/* don't change the wait states, let the application code handle it */
/* note: at power up all wait states will be the maximum (7)       */
/*                 SWWSR = SWWSR_VAL;                              */
BSCR = BSCR_VAL;
DSK5416_DisableFlash();
GEL_TextOut("CPU Reset Complete.\n");
}
/* All memory maps are based on the PMST value of 0xFFE0 */
hotmenu C5416_DSK_Init()
{
GEL_Reset();
PMST = PMST_VAL;
/* don't change the wait states, let the application code handle it */
/* note: at power up all wait states will be the maximum (7)       */
/*                 SWWSR = SWWSR_VAL;                              */
BSCR = BSCR_VAL;
DSK5416_DisableFlash();
C5416_Periph_Reset();
GEL_XMDef(0,0x1eu,1,0x8000u,0x7f);
GEL_XMOn();
GEL_MapOn();
GEL_MapReset();
GEL_MapAdd(0x80u,0,0x7F80u,1,1);          /* DARAM */
GEL_MapAdd(0x08000u,0,0x8000u,1,1);       /* External */
GEL_MapAdd(0x18000u,0,0x8000u,1,1);       /* DARAM */
GEL_MapAdd(0x18000u,0,0x8000u,1,1);       /* SARAM */
GEL_MapAdd(0x28000u,0,0x8000u,1,1);       /* SARAM */
GEL_MapAdd(0x0u,1,0x60u,1,1);             /* MMRs */
GEL_MapAdd(0x60u,1,0x7FA0u,1,1);          /* DARAM */
GEL_MapAdd(0x08000u,1,0x8000u,1,1);       /* DARAM */
GEL_MapAdd(0x00000u,2,0x10000u,1,1);      /* IO Space */
GEL_TextOut("C5416_Init Complete.\n"); }
/* ***************************** */
C5416_Periph_Reset()
{
IFR = 0xFFFFu;
IFR = 0x0000u;
DMA_Reset();
MCBSP0_Reset();
MCBSP1_Reset();
MCBSP2_Reset();
TIMER0_Reset();
GPIO_Reset();
}
DMA_Reset()
{
*(int *)DMPREC = ZEROS;
*(int *)DMSA = DMA_CH0_DMFSC_SUB_ADDR;
*(int *)DMSDI = ZEROS;
*(int *)DMSDI = ZEROS;
*(int *)DMSA  = DMA_CH1_DMFSC_SUB_ADDR;
*(int *)DMSDI = ZEROS;
*(int *)DMSDI = ZEROS;
```

```
*(int *)DMSA  = DMA_CH2_DMFSC_SUB_ADDR;
*(int *)DMSDI = ZEROS;
*(int *)DMSDI = ZEROS;
*(int *)DMSA  = DMA_CH3_DMFSC_SUB_ADDR;
*(int *)DMSDI = ZEROS;
*(int *)DMSDI = ZEROS;
*(int *)DMSA  = DMA_CH4_DMFSC_SUB_ADDR;
*(int *)DMSDI = ZEROS;
*(int *)DMSDI = ZEROS;
*(int *)DMSA  = DMA_CH2_DMFSC_SUB_ADDR;
*(int *)DMSDI = ZEROS;
*(int *)DMSDI = ZEROS;
}
MCBSP0_Reset()
{
*(int *)MCBSP0_SPSA = MCBSP_SPCR1_SUB_ADDR;
*(int *)MCBSP0_SPSD = ZEROS;
*(int *)MCBSP0_SPSA = MCBSP_SPCR2_SUB_ADDR;
*(int *)MCBSP0_SPSD = ZEROS;
*(int *)MCBSP0_SPSA = MCBSP_SRGR1_SUB_ADDR;
*(int *)MCBSP0_SPSD = SRGR1_INIT;
*(int *)MCBSP0_SPSA = MCBSP_SRGR2_SUB_ADDR;
*(int *)MCBSP0_SPSD = ZEROS;
*(int *)MCBSP0_SPSA = MCBSP_MCR1_SUB_ADDR;
*(int *)MCBSP0_SPSD = ZEROS;
*(int *)MCBSP0_SPSA = MCBSP_MCR2_SUB_ADDR;
*(int *)MCBSP0_SPSD = ZEROS;
}
MCBSP1_Reset()
{
*(int *)MCBSP1_SPSA = MCBSP_SPCR1_SUB_ADDR;
*(int *)MCBSP1_SPSD = ZEROS;
*(int *)MCBSP1_SPSA = MCBSP_SPCR2_SUB_ADDR;
*(int *)MCBSP1_SPSD = ZEROS;
*(int *)MCBSP1_SPSA = MCBSP_SRGR1_SUB_ADDR;
*(int *)MCBSP1_SPSD = SRGR1_INIT;
*(int *)MCBSP1_SPSA = MCBSP_SRGR2_SUB_ADDR;
*(int *)MCBSP1_SPSD = ZEROS;
*(int *)MCBSP1_SPSA = MCBSP_MCR1_SUB_ADDR;
*(int *)MCBSP1_SPSD = ZEROS;
*(int *)MCBSP1_SPSA = MCBSP_MCR2_SUB_ADDR;
*(int *)MCBSP1_SPSD = ZEROS;
}
MCBSP2_Reset()
{
*(int *)MCBSP2_SPSA = MCBSP_SPCR1_SUB_ADDR;
*(int *)MCBSP2_SPSD = ZEROS;
*(int *)MCBSP2_SPSA = MCBSP_SPCR2_SUB_ADDR;
*(int *)MCBSP2_SPSD = ZEROS;
*(int *)MCBSP2_SPSA = MCBSP_SRGR1_SUB_ADDR;
*(int *)MCBSP2_SPSD = SRGR1_INIT;
*(int *)MCBSP2_SPSA = MCBSP_SRGR2_SUB_ADDR;
*(int *)MCBSP2_SPSD = ZEROS;
*(int *)MCBSP2_SPSA = MCBSP_MCR1_SUB_ADDR;
*(int *)MCBSP2_SPSD = ZEROS;
*(int *)MCBSP2_SPSA = MCBSP_MCR2_SUB_ADDR;
```

```
*(int *)MCBSP2_SPSD = ZEROS;
}
TIMER0_Reset()
{
*(int *)TCR0 = TIMER_STOP;
*(int *)PRD0 = PRD_DEFAULT;
*(int *)TCR0 = TIMER_RESET;
}
GPIO_Reset()
{
*(int *)GPIOCR = ZEROS;
}
DSK5416_DisableFlash()
{
/* Disable Flash so SRAM is visible */
GEL_MemoryFill(0x0005, 2, 1, 0x40);
}
```

Appendix D

Linker Command File

D.1 5416_lnkp.cmd [33]

```
MEMORY
{
PAGE 0: P_DARAM45: origin = 0x18000, len = 0x4000
VECT: origin = 0x7f80, len = 0x80
PAGE 1: USERREGS: origin = 0x60, len = 0x1c
BIOSREGS: origin = 0x7c, len = 0x4
D_DARAM0: origin = 0x80, len = 0x1f80
D_DARAM13: origin = 0x2000, len = 0x6000
D_DARAM67: origin = 0xc000, len = 0x4000
}
SECTIONS
{
.vectors: {} > VECT PAGE 0
.sysregs: {} > BIOSREGS PAGE 1
.trcinit: {} > P_DARAM45 PAGE 0
.gblinit: {} > P_DARAM45 PAGE 0
frt: {} > P_DARAM45 PAGE 0
.text: {} > P_DARAM45 PAGE 0
.cinit: {} > P_DARAM45 PAGE 0
.pinit: {} > P_DARAM45 PAGE 0
.sysinit: {} > P_DARAM45 PAGE 0
.bss: {} > D_DARAM0 PAGE 1
.far: {} > D_DARAM67 PAGE 1
.const: {} > D_DARAM0 PAGE 1
.data: {} > D_DARAM0 PAGE 1
.switch: {} > D_DARAM0 PAGE 1
.sysmem: {} > D_DARAM13 PAGE 1
.cio: {} > D_DARAM0 PAGE 1
.MEM$obj: {} > D_DARAM0 PAGE 1
.sysheap: {} > D_DARAM0 PAGE 1
.stack: {} > D_DARAM0 PAGE 1
}
```

Appendix E

Include Files

E.1 AED.h [6]

```
/*********************************************************************
AED System Descriptions
TYPES and CONSTANTS:
AED_xxxx - Standard error return values
AED_FLASH_xxxx - Standard flashing constants
AED_PRINT - Indicates printing preference
CHIP_6xxx - Determines processor model
CHECK_FPGA_OVFL - Determines if overflow check is desired
FPGA_ADDRESS - Determines the address of the FPGA data
FPGA_OVFL_CHECK_ENABLE - Determines if overflow check is possible
OLD_FPGA_REV - Determines if old FPGA revision is used
TALK_TO_FPGA - Determines if the FPGA is present
xxxx_LED - Standard LED definitions
FUNCTIONS:
error_flashing - Flashes a numeric code to ERR_LED forever
error_flashing_while - Flashes a numeric code while condition
*********************************************************************/
#ifndef _AED_H_
#define _AED_H_
#include "AED_Cfg.h"
#include "AED_Brd.h"
#ifndef AED_PRINT
/* Should be set in Project|Options|Symbols|"Define Symbols" or AED_Cfg.h
      2 = File
      1 = Print  (Default)
      0 = No print */
#define AED_PRINT  2
#endif
#ifndef CHECK_FPGA_OVFL
/* Should be set in Project|Options|Symbols|"Define Symbols" or AED_Cfg.h
0 = No Check Made
1 = Check FPGA overflow pin(default)
*/
#define CHECK_FPGA_OVFL 1
#endif
#if (!(defined(CHIP_2810)  || \
defined(CHIP_5416)  || \
defined(CHIP_5510)  || \
defined(CHIP_6201)  || \
defined(CHIP_6701)  || \
defined(CHIP_6211)  || \
defined(CHIP_6211X) || \
defined(CHIP_6416)  || \
defined(CHIP_6711)))
/* Should be set in Project|Options|Symbols|"Define Symbols" or AED_Cfg.h  */
#error "Must #define processor type as CHIP_xxxx"
#endif
#ifndef FPGA_ADDRESS
/* Should be set in Project|Options|Symbols|"Define Symbols" or AED_Cfg.h
```

```
Not defined = Use standard FPGA address
Defined = Use the FPGA address specified
*/
#define FPGA_ADDRESS  get_data_addr()
#endif
/* Indicates if FPGA supports overflow checking and it is enabled */
#define FPGA_OVFL_CHECK_ENABLE ((OLD_FPGA_REV<2) && \
          TALK_TO_FPGA && CHECK_FPGA_OVFL)
#ifndef OLD_FPGA_REV
/* Should be set in Project|Options|Symbols|"Define Symbols" or AED_Cfg.h
0 = Rev 1 board (default)
Rev 0 board; FPGA Version 102J, 100x (default)
1 = Rev 0 board; FPGA Version 102I
2 = Rev 0 board; FPGA Version 102H or older
*/
#define OLD_FPGA_REV 0
#endif
#ifndef TALK_TO_FPGA
/* Should be set in Project|Options|Symbols|"Define Symbols" or AED_Cfg.h
0 = No FPGA present
1 = Communication with FPGA (default)
*/
#define TALK_TO_FPGA 1
#endif
/* Standard LED definitions */
#define ERR_LED  0
#define APPL_LED 1
/* Standard error returns */
#define AED_OK   1
#define AED_ERR  0
/* Standard flashing constants, DON'T use 10s */
#define AED_FLASH_MAIN_DMS_ERROR 11
#define AED_FLASH_MAIN_FPGA_OVERFLOW 12
#define AED_FLASH_NORMAL_COMPLETION 13
#define AED_FLASH_DAC_DATA_BUFFER_MALLOC_ERROR 14
#define AED_FLASH_DAC_ERROR_SAMPLE_RATE 15
#define AED_FLASH_EDMA_ERROR_LINK_ALLOCATION 16
#define AED_FLASH_MAIN_DATA_BUFFER_MALLOC_ERROR 17
#define AED_FLASH_APPL_SAVE_BUFFER_MALLOC_ERROR 18
#define AED_FLASH_MAIN_TEST_BUFFER_MALLOC_ERROR 19
#define AED_FLASH_FEATURE_NOT_IMPLEMENTED 21
/*-------------------------------------------------------------*/
/*        FUNCTIONS                                            */
/*-------------------------------------------------------------*/
/*****************************************************************
   error_flashing - Flashes a numeric code to ERR_LED forever
   Parameters:  IN flashes - numeric code ranging 1 - 99
*****************************************************************/
void error_flashing(int flashes);
/*****************************************************************
error_flashing_while - Flashes a numeric code while condition
Parameters:  IN flashes - numeric code ranging 1 - 99
IN cond    - pointer to a boolean condition
*****************************************************************/
void error_flashing_while(int flashes, int *cond);
#endif
```

E.2 AED_Appl.h [6]

```
/*********************************************************************
AED System Descriptions
TYPES and CONSTANTS:
ApplBlockType - Standard buffer type definition
EXTERNAL VARIABLES:
appl_test_data - pointer to test data area for
TALK_TO_FPGA = 0
FUNCTIONS:
appl_parms - Defines size of block for DMA transfers
appl_init - Performs block init before data transfer
appl_test - Fills block with test data from FPGA
appl_process - Processes 1 buffer of data
appl_idle - Performs background processing
appl_end - Processing after termination of main loop
*********************************************************************/
#ifndef _APPL_HDR_
#define _APPL_HDR_
#if (defined(CHIP_5416) || defined(CHIP_5510))
typedef union {
long * word;
unsigned long * uword;
short * hword;
unsigned short * uhword;
char * byte;
unsigned char * ubyte;
} ApplBlockType;
typedef long WordType;
typedef unsigned long UWordType;
#else
/* standard buffer type definition - 32 bit DSPs */
typedef union {
int * word;
unsigned int * uword;
short * hword;
unsigned short * uhword;
char * byte;
unsigned char * ubyte;
} ApplBlockType;
typedef int WordType;
typedef unsigned long UWwordType;
#endif
extern ApplBlockType appl_test_data;
extern ApplBlockType dual_data_buffer;
/*********************************************************************/
/* standard application interface functions - called by AED_main    */
/*********************************************************************/
/*********************************************************************
appl_parms - Defines mode and size of block for DMS transfers
Parameters:  OUT frames  - number of frames in the block
OUT records - number of records in the frame
OUT reclen  - number of transfer elements in record
OUT esize   - transfer element size code (brd_hdr.h)
OUT mode    - transfer mode code (DMS_hdr.h)
```

```
*********************************************************************/
void appl_parms(unsigned int *frames, unsigned int *records, unsigned int *reclen, unsigned int
*esize, unsigned int
*mode);
/* masks for mode parameter */
#define DMA_SYNC_MASK  0x0fff
#define DMA_PORT_MASK  0xf000
/*********************************************************************
appl_init - Performs buffer init before data transfer
Parameters:  OUT data_block - pointer to beginning of entire input
block (all frames)
IN block_bytes - number of bytes in block
IN dma_chan - DMA channel allocated by main for input
Note: This routine is autmatically called from main before the data transfer is intiated, to initialize data buffers.
*********************************************************************/
void appl_init(ApplBlockType data_block, unsigned int  block_bytes, Dma_channel   dma_chan);
/*********************************************************************
appl_test - Fills separate test block to simulate data from FPGA
Parameters:  OUT fill - pointer to beginning of test block
IN frame_bytes - number of bytes in each frame
IN frames - number of frames in block
Note: This routine is autmatically called from main to initialize the test buffer with data.
*********************************************************************/
unsigned long  appl_test(ApplBlockType fill, int frame_bytes, int frame);
/*********************************************************************
appl_process - Processes 1 frame of buffer data
Returns: user defined termination code, 0 is no termination
Parameters:  IN data_buffer - pointer to beginning of frame of data just received from FPGA
IN buf_number - number of buffer just received
Note: This routine is autmatically called from main when a full buffer of data has been transferred from the
daughterboard.
*********************************************************************/
int  appl_process(ApplBlockType data_buffer, int buf_number);
/*********************************************************************
appl_idle - Performs background processing
Returns:   user defined termination code, 0 is no termination
Note: This routine is automatically called from main when no other processing is required, but may not be called
regularly.
*********************************************************************/
int  appl_idle(void);
/*********************************************************************
appl_end - Final processing before termination
Parameters:  IN times - main program loop cycles executed
IN bufs_proc - number of buffers processed by appl_process
            IN buf_count - number of buffers received
            IN prev_buf_count - last buffer given to appl_process
            IN DMS_err - error code from DMS
            IN appl_term_code - user termination code from either appl_process or appl_idle
            IN FIFO_ovfl - indication that the FIFOin the FPGA has overflowed
            IN DMS_count      - number of frames remaining to be received in the block
   Note: This routine is automatically called from main at program termination.
*********************************************************************/
void appl_end(unsigned int times, unsigned int bufs_proc, unsigned int buf_count, unsigned int
                prev_buf_count, int DMS_err, int appl_term_code, unsigned int FIFO_ovfl,
                unsigned int DMS_count);
#endif
```

E.3 AED_Brd.h [6]

```
/************************************************************************
TYPES and CONSTANTS:
INTR_xxxx - Type for interrupt selection number (ISN)
DMA_esize - Defines for DMA transfer element size code
FUNCTIONS:
alloc_timer_intr - Provide an interrupt at specified period
board_init - Initialize EVM or DSK board for use
brd_led_enable - Illuminate user LED on board
brd_led_disable - Extinguish user LED on board
byte_size - Number of address increments in element sizes
cpu_freq - Frequency of internal CPU clock in MHz
delay_usec - Delay specified number of microseconds
delay_msec - Delay specified number of milliseconds
FPGA_start - Start the FPGA collecting data
FPGA_stat_addr - Pointer to FPGA status address
FPGA_stat_mask - Mask FPGA FIFO overflow bit
FPGA_stop - Stop the FPGA collecting data and reset FIFO
get_data_addr - Daughterboard data read/write address
get_cntl_addr - Daughterboard control read/write address
interrupt_init - Bind interrupt service routine to an ISN
intr_pause - Disable ISN
intr_start - Enable ISN with cleared flag
mcbsp_freq - Frequency of internal MCBSP clock in MHz
read_32b_reg - Read 32 bit data from daughterboard
write_32b_reg - Write 32 bit data to daughterboard
************************************************************************/
#ifndef _BRD_H_
#define _BRD_H_
/*----------------------------------------------------------------*/
/* TYPES and CONSTANTS                                            */
/*----------------------------------------------------------------*/
/* Interrupt Selection Numbers */
typedef enum {
INTR_TIMER0 = 0x1,
INTR_TIMER1 = 0x2,
INTR_SDRAM = 0x3,
INTR_PCI_IRQ = 0x4,
INTR_FIFO_READ = 0x5,
INTR_FIFO_WRITE = 0x6,
INTR_DBOARD = 0x7,
INTR_EDMA = 0x8,
INTR_DMA_0 = 0x8,
INTR_DMA_1 = 0x9,
INTR_DMA_2 = 0xA,
INTR_DMA_3  = 0xB,
INTR_MCBSP_TRANS_0 = 0xC,
INTR_MCBSP_REC_0  = 0xD,
INTR_MCBSP_TRANS_1 = 0xE,
INTR_MCBSP_REC_1  = 0xF
} IntrSelNumType;
/* Element size codes for daughterboard transfers */
#define DMA_ESIZE32 0x00
#define DMA_ESIZE16 0x01
```

```
#define DMA_ESIZE8 0x02
/*-------------------------------------------------------------*/
/*                    FUNCTIONS                                */
/*-------------------------------------------------------------*/
/*************************************************************************
alloc_timer_intr - Provide an interrupt at specified period
Returns: ISN for interrupt at period timing
Parameters: IN period_in - desired period in 500 ns increments
*************************************************************************/
IntrSelNumType alloc_timer_intr (unsigned long period_in);
/*************************************************************************
board_init - Initialize EVM or DSK board for use
*************************************************************************/
void board_init (void);
/*************************************************************************
brd_led_enable - Illuminate user LED on board
Returns: error code indicating incorrect LED for this board
Parameters: IN LED_number - number of LED beginning at zero
Note: The number available will vary with board.
*************************************************************************/
int brd_led_enable(int LED_number);
/*************************************************************************
brd_led_disable -  Extinguish LED on EVM board
Returns: error code indicating incorrect LED for this board
Parameters:  IN LED_number - number of LED beginning at zero
Note: The number available will vary with board.
*************************************************************************/
int brd_led_disable(int LED_Number);
/*************************************************************************
byte_size - Number of address increments in element sizes
RETURNS: address increments (same as sizeof() returns)
PARAMETERS: IN esize_code - element size codes for daughterboard transfers
*************************************************************************/
int byte_size (unsigned int esize_code);
/*************************************************************************
cpu_freq - Frequency of internal CPU clock in MHz
RETURNS: CPU frequence in MHz
*************************************************************************/
int cpu_freq (void);
/*************************************************************************
delay_usec -  Delay specified number of microseconds
RETURN: error code
PARAMETERS:  IN numUsec - number of microseconds delay
*************************************************************************/
int delay_usec (unsigned short numUsec);
/*************************************************************************
delay_msec -  Delay specified number of milliseconds
RETURN: error code
PARAMETERS:  IN numMsec - number of microseconds delay
*************************************************************************/
int delay_msec (unsigned short numMsec);
/*************************************************************************
 FPGA_enable - Enable the FPGA to receive data
*************************************************************************/
void FPGA_enable (void);
/*************************************************************************
```

```
 FPGA_start - Start the FPGA collecting data
*************************************************************************/
void FPGA_start (void);
/*************************************************************************
FPGA_stat_addr - Pointer to FPGA status address
RETURN: pointer to status address word
*************************************************************************/
unsigned int * FPGA_stat_addr (void);
/*************************************************************************
FPGA_stat_mask - Mask FPGA FIFO overflow bit
RETURN: mask for overflow bit in status address word
*************************************************************************/
unsigned int FPGA_stat_mask (void);
/*************************************************************************
FPGA_stop - Stop the FPGA collecting data and reset FIFO
*************************************************************************/
void FPGA_stop (void);
/*************************************************************************
get_data_addr - Daughterboard data read/write address
Returns:  void pointer for read/write of data
Notes: This board may set bits in registers if necessary to
render this address active (like paging bits)
*************************************************************************/
void * get_data_addr (void);
 /*************************************************************************
 get_cntl_addr - Daughterboard control read/write address
 Returns:  void pointer for read/write of control
 Notes: This board may set bits in registers if necessary to render this address active (like paging bits)
*************************************************************************/
void * get_cntl_addr (void);
/*************************************************************************
interrupt_init - Bind interrupt service routine to an interrupt
PARAMETERS:  IN ptr_isr - pointer to an interrupt service routine
IN isn - interrupt selection number
*************************************************************************/
void interrupt_init (void(*ptr_isr)(void), IntrSelNumType isn);
/*************************************************************************
intr_pause - Disable ISN
PARAMETERS:  IN isn - interrupt selection number
*************************************************************************/
void intr_pause (IntrSelNumType isn);
/*************************************************************************
intr_start - Enable ISN with cleared flag
PARAMETERS:  IN isn - interrupt selection number
*************************************************************************/
void intr_start (IntrSelNumType isn);
/*************************************************************************
mcbsp_freq - Frequency of internal MCBSP clock in MHz
RETURNS: MCBSP frequency in MHz
*************************************************************************/
int mcbsp_freq (void);
/*************************************************************************
read_32b_reg - Read 32 bit data from daughterboard
RETURNS: value read from the specified address
PARAMETERS:  IN addr - pointer to the read address
NOTE: This routine transmits 32 bits to the daughterboard for
```

```
both 16 and 32 bit buses.
*****************************************************************/
unsigned long read_32b_reg (unsigned long *addr);
/*****************************************************************
write_32b_reg - Write 32 bit data to daughterboard
PARAMETERS:  IN addr - pointer to the write address
IN data - value to be written
NOTE: This routine transmits 32 bits to the daughterboard for both 16 and 32 bit buses.
*****************************************************************/
void write_32b_reg (unsigned long *addr, unsigned long data);
/*****************************************************************/
#endif
```

E.4 AED_Cfg.h [6]

```
/*****************************************************************
TYPES and CONSTANTS:
FUNCTIONS:
*****************************************************************/
#ifndef _CONFIG_H_
#define _CONFIG_H_
/*****************************************************************
APPLICATION DEFINES
*****************************************************************/
/*#define AED_PRINT                       0*/
/*#define TALK_TO_FPGA                    0*/
#define CHECK_FPGA_OVFL                   0
#endif
```

E.5 AED_DMS.h [6]

```
/*****************************************************************
TYPES and CONSTANTS:
DMAC_EX_xxxx - controls for DMA status lines
FROM/TO_FPGA_xxxx - direction values for DMS setup
DMA_CH_NUMBER - maximum number of channels
DMA_xxx_MODE - mode values for DMS setup
Dma_channel - channel ID type
FUNCTIONS:
alloc_DMA_channel - allocate a channel and return id
count_DMA_channel - return frames remaining in block
operate_DMA_channel - transfer one block with the channel
pause_DMA_channel - pause the transfer in the channel
program_DMA_channel - setup channel registers
start_DMA_channel - begin continuous transfer in channel
test_DMA_channel - setup channel for test mode
VARIABLES:
read_err - error codes returned by channels
buf_count - count of frames or blocks transferred (depends on channel mode)
*****************************************************************/
#ifndef DMS_HDR_
#define DMS_HDR_
/* operation of the DMA status pin */
#define DMAC_EN_LOW   0 /* DMAC pin is held low */
#define DMAC_EN_HIGH  1 /* DMAC pin is held high */
#define DMAC_EN_RSYNC_STAT 2 /* DMAC reflects RSYNC STAT */
#define DMAC_EN_WSYNC_STAT 3 /* DMAC reflects WSYNC STAT */
#define DMAC_EN_FRAME_COND 4 /* DMAC reflects FRAME COND */
#define DMAC_EN_BLOCK_COND 5 /* DMAC reflects BLOCK COND */
/* direction parameter values for channel programming */
#define FROM_FPGA_MEM                   0
#define FROM_FPGA_SER1                  1
#define FROM_FPGA_SER0                  2
#define TO_FPGA_MEM                     3
#define TO_FPGA_SER1                    4
#define TO_FPGA_SER0                    5
/* maximum number of channels available */
#define DMA_CH_NUMBER  4
/* DMS mode values */
#define DMA_RS_MODE  0   /* read sync mode */
#define DMA_WS_MODE  1   /* write sync mode */
#define DMA_FS_MODE  2   /* frame sync cont mode */
#define DMA_FSB_MODE 3   /* frame sync block mode */
#define DMA_NONE_MODE 4
#define DMA_XFER_METHOD(mode) (mode & 0xf) /* remove start/stop */
#define DMA_START_ON  0x000   /* start DMA imediately - default */
#define DMA_START_OFF 0x100   /* do not start the DMA */
/* channel ID type */
typedef enum
{
DMA_CH_NONE= DMA_CH_NUMBER,
DMA_CH_0= 0,
DMA_CH_1,
DMA_CH_2,
```

```
DMA_CH_3
}Dma_channel;
/*********************************************************************/
extern volatile int read_err[DMA_CH_NUMBER+1], buf_count[DMA_CH_NUMBER+1];
/*********************************************************************
Standard DMS interface functions - called by main and applications
*********************************************************************/
/*********************************************************************
alloc_DMA_channel - allocate a channel and return id
Return:  channel ID code
*********************************************************************/
Dma_channel alloc_DMA_channel (void);
/*********************************************************************
count_DMA_channel - return frames remaining in block
Returns:  frame count remaining
Parameters:  IN chan - channel ID code
*********************************************************************/
int  count_DMA_channel(Dma_channel chan);
/*********************************************************************
operate_DMA_channel - transfer one block with the channel
Parameters:  IN chan - channel ID code
*********************************************************************/
void operate_DMA_channel(Dma_channel chan);
/*********************************************************************
pause_DMA_channel - pause the transfer in the channel
Parameters:  IN chan - channel ID code
*********************************************************************/
void pause_DMA_channel(Dma_channel chan);
/*********************************************************************
program_DMA_channel - setup channel registers
Parameters:  IN chan - channel ID code
IN dir - direction values for DMS setup
IN dest - pointer to destination address
IN src - pointer to source address
IN count - number of transfer elements in frame
IN frames - number of frames in block
IN esize - transfer element size code (brd_hdr.h)
IN mode - mode values for DMS setup
*********************************************************************/
void program_DMA_channel(Dma_channel chan, int dir, void *dest, void *src, unsigned int count,
                         unsigned int frames, unsigned int esize, unsigned int mode);
/*********************************************************************
start_DMA_channel - begin continuous transfer in channel
Parameters:  IN chan - channel ID code
*********************************************************************/
void start_DMA_channel(Dma_channel chan);
/*********************************************************************
test_DMA_channel - setup channel for test mode
 Parameters:  IN chan - channel ID code
IN src - pointer to test source block
IN period - period between desired between interrupts (0.5 millisecond units )
*********************************************************************/
void test_DMA_channel(Dma_channel chan, void *src, unsigned long period);
#endif
```

E.6 dma5416.h [34]

```
#ifndef DSK5416_
#define DSK5416_
#ifdef __cplusplus
extern "C" {
#endif
/*
 *  Note:  Bit definitions for each register field needs to be supplied here for the CPLD and other board peripherals.
 */
/* Board specific I/O registers */
ioport unsigned                                 port0;
ioport unsigned                                 port1;
ioport unsigned                                 port2;
ioport unsigned                                 port3;
ioport unsigned                                 port4;
ioport unsigned                                 port5;
ioport unsigned                                 port6;
ioport unsigned                                 port7;

/* Define easier to read names for I/O registers */
#define DSK5416_USER_REG                        port0
#define DSK5416_DC_REG                          port1
#define DSK5416_PCM3002_L                       port2
#define DSK5416_PCM3002_H                       port3
#define DSK5416_VERSION                         port4
#define DSK5416_DM_CNTL                         port5
#define DSK5416_MISC                            port6
#define DSK5416_CODEC_CLK                       port7
/* Initialize all board APIs */
void DSK5416_init();
#ifdef __cplusplus
}
#endif
#endif
```

E.7 dsk5416.h [35]

```
*  ======== dsk5416.h ========
*
*  This files contains DSK5416 board specific I/O registers
*  define for the CPLD.
*/
#ifndef DSK5416_
#define DSK5416_
#ifdef __cplusplus
extern "C" {
#endif
/*  Note:  Bit definitions for each register field needs to be supplied here for the CPLD and other board peripherals.
 */
/* Board specific I/O registers */
ioport unsigned         port0;
ioport unsigned         port1;
ioport unsigned         port2;
ioport unsigned         port3;
ioport unsigned         port4;
ioport unsigned         port5;
ioport unsigned         port6;
ioport unsigned         port7;
/* Define easier to read names for I/O registers */
#define DSK5416_USER_REG                    port0
#define DSK5416_DC_REG                      port1
#define DSK5416_PCM3002_L                   port2
#define DSK5416_PCM3002_H                   port3
#define DSK5416_VERSION                     port4
#define DSK5416_DM_CNTL                     port5
#define DSK5416_MISC                        port6
#define DSK5416_CODEC_CLK                   port7
/* Initialize all board APIs */
void DSK5416_init();
#ifdef __cplusplus
}
#endif
#endif
```

E.8 emif.h [36]

```
/*****************************************************************/
/* This header files defines the data structures and macros to access the Software Wait State and Bank Switch
Control Regs and their bits/fields. */
/*****************************************************************/
#include "regs.h"
/*****************************************************************/
/* BNKCMP_MASK(val) - creates mask to set bank size of ext mem */
/* val   - bank size of external memory (4,8,32,64) */
/*****************************************************************/
#define BNKCMP_MASK(val) (0x0010u - (val/0x4u))
/*****************************************************************/
/* BSCR_MASK(bnkcmp, psds, bh, exio) - set bank switch ctrl reg */
/* bnkcmp - size of external memory banks (4 - 64) */
/* psds - flag, equal 1 if extra cycle for back-to-back program-data or data-program memory reads */
/* bh - flag, equal 1 if data bus holder is active , holds data bus , D(15-0) at previous logic level */
/* exio - flag, equal 1 if external-bus-off function is enabled. (normally set to 0) */
/*****************************************************************/
#define BSCR_MASK(bnkcmp, psds, bh, exio)\
                ((MASK_FIELD(BNKCMP,BNKCMP_MASK(bnkcmp), BNKCMP_SZ)) &\
                (psds ? (MASK_BIT(PSDS) | MASK_TARGET_WORD) : ~MASK_BIT(PSDS)) &\
                (bh   ? (MASK_BIT(BH)   | MASK_TARGET_WORD) : ~MASK_BIT(BH))   &  (exio))
/*****************************************************************/
/* CLEAR_WAIT_STATES - clears all software wait states */
/* addr - address of software wait state register */
/*****************************************************************/
#define CLEAR_WAIT_STATESSWWSR &= ~MASK_TARGET_WORD
/*****************************************************************/
/* SET_WAIT_STATES(ctrl) - set software wait states */
/* addr  - address of software wait state register */
/* ctrl  - mask to use in setting wait state register */
/*****************************************************************/
#define SET_WAIT_STATES(ctrl) SWWSR |= ctrl
/*****************************************************************/
/* SET_BUS_CTRL(ctrl) - sets bus control register */
/* ctrl - mask to use in setting register value */
/*****************************************************************/
#define SET_BUS_CTRL(ctrl)BSCR |= ctrl
```

E.9 intr5416.h [37]

```
#include "regs5416.h"
typedef void (*ISRFUNC)(void);
void software_trap(int trap);     /* Initiates trap to given interrupt */
/*****************************************************************/
/* Define all macros needed to enable/disable interrupts, set */
/* interrupt vectors, allocate space for interrupt vectors and  */
/* set interrupt vector pointer. */
/*****************************************************************/
// extern unsigned int _vectors; /* Start label of vector table */
// extern ISRFUNC isr_jump_table[]; /* Array of ISR pointers       */
/*****************************************************************/
/* Define interrupt trap numbers                                 */
/*****************************************************************/
#define RS_TRAP                                  0
#define NMI_TRAP                                 1
#define INT0_TRAP                               16
#define INT1_TRAP                               17
#define INT2_TRAP                               18
#define TINT_TRAP                               19
#define RINT0_TRAP                              20
#define XINT0_TRAP                              21
#define RINT2_TRAP                              22
#define DMAC0_TRAP                              22
#define XINT2_TRAP                              23
#define DMAC1_TRAP                              23
#define INT3_TRAP                               24
#define HPINT_TRAP                              25
#define RINT1_TRAP                              26
#define DMAC2_TRAP                              26
#define XINT1_TRAP                              27
#define DMAC3_TRAP                              27
#define DMAC4_TRAP                              28
#define DMAC5_TRAP                              29
#define SINTR                                    0
#define SINT16                                   1
#define SINT17                                   2
#define SINT18                                   3
#define SINT19                                   4
#define SINT20                                   5
#define SINT21                                   6
#define SINT22                                   7
#define SINT23                                   8
#define SINT24                                   9
#define SINT25                                  10
#define SINT26                                  11
#define SINT27                                  12
#define SINT28                                  13
#define SINT29                                  14
#define SINT30                                  15
#define SINT0                                   16
#define SINT1                                   17
#define SINT2                                   18
#define SINT3                                   19
```

```
#define SINT4                           20
#define SINT5                           21
#define SINT6                           22
#define SINT7                           23
#define SINT8                           24
#define SINT9                           25
#define SINT10                          26
#define SINT11                          27
#define SINT12                          28
#define SINT13                          29
/*****************************************************************/
/* INTR_ENABLE - enables all masked interrupts by resetting INTM bit in Status Register 1 */
/*****************************************************************/
#define INTR_GLOBAL_ENABLEasm("\tRSBXINTM")
/*****************************************************************/
/* INTR_DISABLE - disables all masked interrupts by setting INTM bit in Status Register 1 */
/*****************************************************************/
#define INTR_GLOBAL_DISABLEasm("\tSSBXINTM")
/*****************************************************************/
/* INTR_CHECK_FLAG(flag) - check the corresponding flag in the IFR register */
/*****************************************************************/
#define INTR_CHECK_FLAG(flag) (IFR & (0x1u << flag))
/*****************************************************************/
/* INTR_CLR_FLAG(flag) - clears the corresponding flag in the IFR register */
/*****************************************************************/
#define INTR_CLR_FLAG(flag){IFR &= (0x1u << flag);}
/*****************************************************************/
/* IDLE(mode) - sets CPU in idle mode based on level selected */
/*****************************************************************/
#define IDLE(mode){ asm("\tidle " #mode); }
/*****************************************************************/
/* INTR_ENABLE (flag) - set interrupt vector flags to enable/reset specific device interrupts */
/* flag - bit to set in interrupt mask register */
/*****************************************************************/
#define INTR_ENABLE(flag) IMR |= MASK_BIT(flag)
/*****************************************************************/
/* INTR_DISABLE (flag) - set interrupt vector flags to reset specific device interrupts */
/* flag - bit to set in interrupt mask register */
/*****************************************************************/
#define INTR_DISABLE(flag)IMR &= ~MASK_BIT(flag)
/*****************************************************************/
/* INTR_INIT - sets interrupt vector pointer */
/*****************************************************************/
#define INTR_INIT{PMST &= 0x7f; PMST |= ( ( (unsigned int)&_vectors ) & 0xff80u); }
/*****************************************************************/
/* INTR_HOOK(isr, isrfunc) - sets interrupt service routine vec */
/* isr - interrupt trap number (see TRAP instruction) */
/* isrfunc - address of interrupt service routine */
/*****************************************************************/
// #define INTR_HOOK(trap_no, isrfunc)
```

E.10 regs.h [38]

```
/*********************************************************************/
/* DEFINE ALL PERIPHERAL MEMORY MAPPED REGISTER ADRESSES */
/*********************************************************************/
/*********************************************************************/
/* Check to see if mmregs.h has been previously included by */
/* another header, if so, skip this and go on */
/*********************************************************************/
#if !defined(__MMREGS)
#include <limits.h>
/*********************************************************************/
/* Target specific data and macros */
/* MASK_TARGET_WORD - bit pattern to mask all bits in a target word */
/* WORD_SIZE - size in bits of target word */
/* BASE_ADDR - base address of memory-mapped peripheral control registers */
/*********************************************************************/
#define MASK_TARGET_WORD 0xffff
#define TARGET_WRD_SZ CHAR_BIT
#define BYTES_PER_WORD TARGET_WRD_SZ/8
#define WORD_SIZE (CHAR_BIT * sizeof(unsigned int))
#define SP_ADDR(port) (0x22 + (0x10 * port))
#define DRR_ADDR(port) (0x20 + (0x10 * port))
#define DXR_ADDR(port) (0x21 + (0x10 * port))
#define BSP_ADDR(port) (0x22 + (0x20 * port))
#define BDRR_ADDR(port) (0x20 + (0x20 * port))
#define BDXR_ADDR(port) (0x21 + (0x20 * port))
#define BSPCE_ADDR(port) (0x23 + (0x20 * port))
#define AXR_ADDR(port) (0x38 + (0x04 * port))
#define ARR_ADDR(port) (0x3a + (0x04 * port))
#define TDM_ADDR TSPC
#define TIMER_ADDR TCR_ADDR
/*-------------------------------------------------------------------*/
/* MACRO FUNCTIONS */
/*-------------------------------------------------------------------*/
/*********************************************************************/
/* Define data structures for all memory mapped registers */
/*********************************************************************/
/*-----------------------------------------------------------*/
/* Define bit fields for Serial Port Control Registers */
/*-----------------------------------------------------------*/
#define RSRFULL 13
#define RSRFULL_SZ 1
#define XSREMPTY 12
#define XSREMPTY_SZ 1
#define IN1 9
#define IN1_SZ 1
#define IN0 8
#define IN0_SZ 1
#define TXM 5
#define TXM_SZ 1
#define MCM 4
#define MCM_SZ 1
#define FSM 3
#define FSM_SZ 1
```

```
#define FO                                  2
#define FO_SZ                               1
#define TDM                                 0
#define TDM_SZ                              1
#define CLKDV                               0
#define CLKDV_SZ                            5
#define FSP                                 5
#define FSP_SZ                              1
#define CLKP                                6
#define CLKP_SZ                             1
#define FE                                  7
#define FE_SZ                               1
#define FIG                                 8
#define FIG_SZ                              1
#define PCM                                 9
#define PCM_SZ                              1
#define BXE                                10
#define BXE_SZ                              1
#define XH                                 11
#define XH_SZ                               1
#define HALTX                              12
#define HALTX_SZ                            1
#define BRE                                13
#define BRE_SZ                              1
#define RH                                 14
#define RH_SZ                               1
#define HALTR                              15
#define HALTR_SZ                            1
/*****************************************************************/
/* Define Timer Period, and Control Registers with all related data structures, macros, and functions */
/*****************************************************************/
#define PSC                                 6
#define PSC_SZ                              4
#define TRB                                 5
#define TRB_SZ                              1
#define TSS                                 4
#define TSS_SZ                              1
#define TDDR                                0
#define TDDR_SZ                             4
/*---------------------------------------------------------------*/
/* Data structures, macros for Clock Mode Register     */
/*---------------------------------------------------------------*/
#define PLLMUL                             12
#define PLLMUL_SZ                           4
#define PLLDIV                             11
#define PLLDIV_SZ                           1
#define PLLCOUNT                            3
#define PLLCOUNT_SZ                         8
#define PLLON_OFF                           2
#define PLLON_OFF_SZ                        1
#define PLLNDIV                             1
#define PLLNDIV_SZ                          1
#define PLLSTATUS                           0
#define PLLSTATUS_SZ                        1
/*****************************************************************/
/* Define bit fields for Software Wait State Register                                 */
```

```
/*****************************************************************/
#define IO                                      12
#define IO_SZ                                    3
#define DATA_HI                                  9
#define DATA_HI_SZ                               3
#define DATA_LO                                  6
#define DATA_LO_SZ                               3
#define PROGRAM_HI                               3
#define PROGRAM_HI_SZ                            3
#define PROGRAM_LO                               0
#define PROGRAM_LO_SZ                            3
/*---------------------------------------------------------------*/
/* Define structure for Bank Switch Control Register             */
/*---------------------------------------------------------------*/
#define BNKCMP                                  12
#define BNKCMP_SZ                                4
#define PSDS                                    11
#define PSDS_SZ                                  1
#define BH                                       1
#define BH_SZ                                    1
#define EXIO                                     0
#define EXIO_SZ                                  1
#define INT0                                     0
#define INT1                                     1
#define INT2                                     2
#define TINT                                     3
#define RINT0                                    4
#define XINT0                                    5
#define RINT2                                    6
#define XINT2                                    7
#define INT3                                     8
#define HPINT                                    9
#define RINT1                                   10
#define XINT1                                   11
#define DMAC0                                    6
#define DMAC1                                    7
#define DMAC2                                   10
#define DMAC3                                   11
#define DMAC4                                   12
#define DMAC5                                   13
/*****************************************************************/
/* DEFINE DATA STRUCTURE FOR HOST PORT INTERFACE CONTROL REG */
/*****************************************************************/
#define BOB                                      0
#define SMOD                                     1
#define DSPINT                                   2
#define HINT                                     3
/*****************************************************************/
/* Serial Port 0 defined for C541 only                           */
/*****************************************************************/
#define DRR0                                *(volatile unsigned int *)0x20
#define DRR0_ADDR                           0x20
#define DXR0                                *(volatile unsigned int *)0x21
#define DXR0_ADDR                           0x21
#define SPC0                                *(volatile unsigned int *)0x22
#define SPC0_ADDR                           0x22
```

```
/*************************************************************/
/* Buffered Serial Port 0 defined for all devices except C541 */
/*************************************************************/
#define BSPC0                   *(volatile unsigned int *)0x22
#define BSPC0_ADDR              0x22
#define BSPCE0                  *(volatile unsigned int *)0x23
#define BSPCE0_ADDR             0x23
#define BDRR0                   *(volatile unsigned int *)0x20
#define BDRR0_ADDR              0x20
#define BDXR0                   *(volatile unsigned int *)0x21
#define BDXR0_ADDR              0x21
/*************************************************************/
/* Defined flags for use in setting control for HPI host interface control pins */
/* The value of these constants is their relative bit position in */
/* the control structure for the host side of the HPI interface */
/*************************************************************/
#define HAS_PIN                 0
#define HBIL_PIN                1
#define HCNTL0_PIN              2
#define HCNTL1_PIN              3
#define HCS_PIN                 4
#define HD0_PIN                 5
#define HDS1_PIN                6
#define HDS2_PIN                7
#define HINT_PIN                8
#define HRDY_PIN                9
#define HRW_PIN                 10
/*************************************************************/
/* AUTOBUFFERING UNIT (Buffered Serial Port 0)               */
/* Defined for all except C541                               */
/*************************************************************/
#define AXR0                    *(volatile unsigned int *)0x38
#define AXR0_ADDR               0x38
#define BKX0                    *(volatile unsigned int *)0x39
#define BKX0_ADDR               0x39
#define ARR0                    *(volatile unsigned int *)0x3a
#define ARR0_ADDR               0x3a
#define BKR0                    *(volatile unsigned int *)0x3b
#define BKR0_ADDR               0x3b
/*************************************************************/
/* AUTOBUFFERING UNIT (Buffered Serial Port 1)               */
/* Defined for C548 Only!!!!!!                               */
/*************************************************************/
#define AXR1                    *(volatile unsigned int *)0x3c
#define AXR1_ADDR               0x3c
#define BKX1                    *(volatile unsigned int *)0x3d
#define BKX1_ADDR               0x3d
#define ARR1                    *(volatile unsigned int *)0x3e
#define ARR1_ADDR               0x3e
#define BKR1                    *(volatile unsigned int *)0x3f
#define BKR1_ADDR               0x3f
/*************************************************************/
/* Buffered Serial Port 1 defined only for C548              */
/*************************************************************/
#define BSPC1                   *(volatile unsigned int *)0x42
#define BSPC1_ADDR              0x42
```

```
#define BSPCE1                  *(volatile unsigned int *)0x43
#define BSPCE1_ADDR             0x43
#define BDRR1                   *(volatile unsigned int *)0x40
#define BDDR1_ADDR              0x40
#define BDXR1                   *(volatile unsigned int *)0x41
#define BDXR1_ADDR              0x41
#define __MMREGS
#endif
```

E.11 regs5416.h

```
#if !defined(__54XXREGS)
#include <limits.h>
#include "regs.h"
/*----------------------------------------------------------------------------*/
/* MACRO FUNCTIONS                                                            */
/*----------------------------------------------------------------------------*/
#define CONTENTS_OF(addr)(*((volatile unsigned int*)(addr)))
#define LENGTH_TO_BITS(length)(~(0xffffffff << (length)))
/* MACROS to SET, CLEAR and RETURN bits and bitfields in Memory Mapped locations using the address of the
specified register. */
#define REG_READ(addr)(CONTENTS_OF(addr))
#define REG_WRITE(addr,val) (CONTENTS_OF(addr) = (val))
#define MASK_BIT(bit) (1 << (bit))
#define RESET_BIT(addr,bit)(CONTENTS_OF(addr) &= (~MASK_BIT(bit)))
#define GET_BIT(addr,bit) (CONTENTS_OF(addr) & (MASK_BIT(bit)) ? 1 : 0)
#define SET_BIT(addr,bit)(CONTENTS_OF(addr) = (CONTENTS_OF(addr)) | (MASK_BIT(bit)))
#define ASSIGN_BIT_VAL(addr,bit,val)( (val) ? SET_BIT(addr,bit) : RESET_BIT(addr,bit) )
#define CREATE_FIELD(bit,length) (LENGTH_TO_BITS(length) << (bit))
#define RESET_FIELD(addr,bit,length)( CONTENTS_OF(addr) &= (~CREATE_FIELD(bit,length)))
#define TRUNCATE(val,bit,length) (((unsigned int)(val) << (bit)) & (CREATE_FIELD(bit, length)))
#define MASK_FIELD(bit,val,length)TRUNCATE(val, bit, length)
#define GET_FIELD(addr,bit,length)((CONTENTS_OF(addr) & CREATE_FIELD(bit,length)) >> bit)
#define LOAD_FIELD(addr,val,bit,length)(CONTENTS_OF(addr) &= (~CREATE_FIELD(bit,length))\
                        | TRUNCATE(val, bit, length))
/*****************************************************************************/
/* Memory-mapped Byte Manipulation Macros                                    */
/*****************************************************************************/
#define CSET_BIT(reg,bit)((*((volatile unsigned char *)(reg))) |= (MASK_BIT(bit)))
#define CGET_BIT(reg,bit)((*((volatile unsigned char *)(reg))) & (MASK_BIT(bit)) ? 1 : 0)
#define CCLR_BIT(reg,bit)((*((volatile unsigned char *)(reg))) &= (~MASK_BIT(bit)))
#define CGET_FIELD(reg,bit,length)\
                        ((*((volatile unsigned char *)(reg)) & (MASK_FIELD(bit,length))) >> bit)
#define CLOAD_FIELD(reg,bit,length,val)((*((volatile unsigned char *)(reg))) = \
                          ((*((volatile unsigned char *)(reg)) & (~MASK_FIELD(bit,length)))) | (val<<bit))
#define CREG_READ(addr)(*((unsigned char *)(addr)))
#define CREG_WRITE(addr,val)(*((unsigned char *)(addr)) = (val))
/* MACROS to SET, CLEAR and RETURN bits and bitfields in Memory Mapped and Non-Memory Mapped using
register names. */
#define GET_REG(reg)(reg)
#define SET_REG(reg,val) ((reg)=(val))
#define GET_REG_BIT(reg,bit)((reg) & MASK_BIT(bit) ? 1 : 0)
#define SET_REG_BIT(reg,bit)((reg) |= MASK_BIT(bit))
#define RESET_REG_BIT(reg,bit)((reg) &= (~MASK_BIT(bit)))
#define GET_REG_FIELD(reg,bit,length)(reg & CREATE_FIELD(bit,length)) >> bit)
#define LOAD_REG_FIELD(reg,val,bit,length)(reg &= (~CREATE_FIELD(bit,length)) | (val<<bit))
/*****************MCBSP Registers, Bits, Bitfields*****************/
/*----------------------------------------------------------------------*/
/* Define bit fields for Serial Port Control Registers 1 and 2   */
/*----------------------------------------------------------------------*/
#define DLB                                             15
#define DLB_SZ                                           1
#define RJUST                                           13
#define RJUST_SZ                                         2
```

```
#define CLKSTP                  11
#define CLKSTP_SZ                2
#define DXENA                    7
#define DXENA_SZ                 1
#define ABIS                     6
#define ABIS_SZ                  1
#define RINTM                    4
#define RINTM_SZ                 2
#define RSYNCERR                 3
#define RSYNCERR_SZ              1
#define RFULL                    2
#define RFULL_SZ                 1
#define RRDY                     1
#define RRDY_SZ                  1
#define RRST                     0
#define RRST_SZ                  1
#define FREE                     9
#define FREE_SZ                  1
#define SOFT                     8
#define SOFT_SZ                  1
#define FRST                     7
#define FRST_SZ                  1
#define GRST                     6
#define GRST_SZ                  1
#define XINTM                    4
#define XINTM_SZ                 2
#define XSYNCERR                 3
#define XSYNCERR_SZ              1
#define XEMPTY                   2
#define XEMPTY_SZ                1
#define XRDY                     1
#define XRDY_SZ                  1
#define XRST                     0
#define XRST_SZ                  1
/*----------------------------------------------------------------------*/
/* Define bit fields for Receive Control Registers 1 and 2      */
/*----------------------------------------------------------------------*/
#define RFRLEN1                  8
#define RFRLEN1_SZ               7
#define RWDLEN1                  5
#define RWDLEN1_SZ               3
#define RPHASE                  15
#define RPHASE_SZ                1
#define RFRLEN2                  8
#define RFRLEN2_SZ               7
#define RWDLEN2                  5
#define RWDLEN2_SZ               3
#define RCOMPAND                 3
#define RCOMPAND_SZ              2
#define RFIG                     2
#define RFIG_SZ                  1
#define RDATDLY                  0
#define RDATDLY_SZ               2
/*----------------------------------------------------------------------*/
/* Define bit fields for Transmit Control Registers 1 and 2     */
/*----------------------------------------------------------------------*/
```

```
#define XFRLEN1                 8
#define XFRLEN1_SZ              7
#define XWDLEN1                 5
#define XWDLEN1_SZ              2
#define XPHASE                 15
#define XPHASE_SZ               1
#define XFRLEN2                 8
#define XFRLEN2_SZ              7
#define XWDLEN2                 5
#define XWDLEN2_SZ              3
#define XCOMPAND                3
#define XCOMPAND_SZ             2
#define XFIG                    2
#define XFIG_SZ                 1
#define XDATDLY                 0
#define XDATDLY_SZ              2
/*---------------------------------------------------------------------------*/
/* Define bit fields for Sample Rate Generator Registers 1 and 2   */
/*---------------------------------------------------------------------------*/
#define FWID                    8
#define FWID_SZ                 8
#define CLKGDV                  0
#define CLKGDV_SZ               8
#define GSYNC                  15
#define GSYNC_SZ                1
#define CLKSP                  14
#define CLKSP_SZ                1
#define CLKSM                  13
#define CLKSM_SZ                1
#define FSGM                   12
#define FSGM_SZ                 1
#define FPER                    0
#define FPER_SZ                12
/*---------------------------------------------------------------------------*/
/* Define bit fields for Multi-Channel Control Registers 1 and 2   */
/*---------------------------------------------------------------------------*/
#define RPBBLK                  7
#define RPBBLK_SZ               2
#define RPABLK                  5
#define RPABLK_SZ               2
#define RCBLK                   2
#define RCBLK_SZ                3
#define RMCM                    0
#define RMCM_SZ                 1
#define XPBBLK                  7
#define XPBBLK_SZ               2
#define XPABLK                  5
#define XPABLK_SZ               2
#define XCBLK                   2
#define XCBLK_SZ                3
#define XMCM                    0
#define XMCM_SZ                 2
/*---------------------------------------------------------------------------*/
/* Define bit fields for Receive Channel Enable Register Partition A   */
/*---------------------------------------------------------------------------*/
#define RCEA15                 15
```

```
#define RCEA15_SZ                                1
#define RCEA14                                  14
#define RCEA14_SZ                                1
#define RCEA13                                  13
#define RCEA13_SZ                                1
#define RCEA12                                  12
#define RCEA12_SZ                                1
#define RCEA11                                  11
#define RCEA11_SZ                                1
#define RCEA10                                  10
#define RCEA10_SZ                                1
#define RCEA9                                    9
#define RCEA9_SZ                                 1
#define RCEA8                                    8
#define RCEA8_SZ                                 1
#define RCEA7                                    7
#define RCEA7_SZ                                 1
#define RCEA6                                    6
#define RCEA6_SZ                                 1
#define RCEA5                                    5
#define RCEA5_SZ                                 1
#define RCEA4                                    4
#define RCEA4_SZ                                 1
#define RCEA3                                    3
#define RCEA3_SZ                                 1
#define RCEA2                                    2
#define RCEA2_SZ                                 1
#define RCEA1                                    1
#define RCEA1_SZ                                 1
#define RCEA0                                    0
#define RCEA0_SZ                                 1
/*-------------------------------------------------------------------------------*/
/* Define bit fields for Receive Channel Enable Register Partition B   */
/*-------------------------------------------------------------------------------*/
#define RCEB15                                  15
#define RCEB15_SZ                                1
#define RCEB14                                  14
#define RCEB14_SZ                                1
#define RCEB13                                  13
#define RCEB13_SZ                                1
#define RCEB12                                  12
#define RCEB12_SZ                                1
#define RCEB11                                  11
#define RCEB11_SZ                                1
#define RCEB10                                  10
#define RCEB10_SZ                                1
#define RCEB9                                    9
#define RCEB9_SZ                                 1
#define RCEB8                                    8
#define RCEB8_SZ                                 1
#define RCEB7                                    7
#define RCEB7_SZ                                 1
#define RCEB6                                    6
#define RCEB6_SZ                                 1
#define RCEB5                                    5
#define RCEB5_SZ                                 1
```

```
#define RCEB4                                4
#define RCEB4_SZ                             1
#define RCEB3                                3
#define RCEB3_SZ                             1
#define RCEB2                                2
#define RCEB2_SZ                             1
#define RCEB1                                1
#define RCEB1_SZ                             1
#define RCEB0                                0
#define RCEB0_SZ                             1
/*---------------------------------------------------------------------------*/
/* Define bit fields for Transmit Channel Enable Register Partition A */
/*---------------------------------------------------------------------------*/
#define XCEA15                              15
#define XCEA15_SZ                            1
#define XCEA14                              14
#define XCEA14_SZ                            1
#define XCEA13                              13
#define XCEA13_SZ                            1
#define XCEA12                              12
#define XCEA12_SZ                            1
#define XCEA11                              11
#define XCEA11_SZ                            1
#define XCEA10                              10
#define XCEA10_SZ                            1
#define XCEA9                                9
#define XCEA9_SZ                             1
#define XCEA8                                8
#define XCEA8_SZ                             1
#define XCEA7                                7
#define XCEA7_SZ                             1
#define XCEA6                                6
#define XCEA6_SZ                             1
#define XCEA5                                5
#define XCEA5_SZ                             1
#define XCEA4                                4
#define XCEA4_SZ                             1
#define XCEA3                                3
#define XCEA3_SZ                             1
#define XCEA2                                2
#define XCEA2_SZ                             1
#define XCEA1                                1
#define XCEA1_SZ                             1
#define XCEA0                                0
#define XCEA0_SZ                             1
/*---------------------------------------------------------------------------*/
/* Define bit fields for Transmit Channel Enable Register Partition B */
/*---------------------------------------------------------------------------*/
#define XCEB15                              15
#define XCEB15_SZ                            1
#define XCEB14                              14
#define XCEB14_SZ                            1
#define XCEB13                              13
#define XCEB13_SZ                            1
#define XCEB12                              12
#define XCEB12_SZ                            1
```

```
#define XCEB11                          11
#define XCEB11_SZ                        1
#define XCEB10                          10
#define XCEB10_SZ                        1
#define XCEB9                            9
#define XCEB9_SZ                         1
#define XCEB8                            8
#define XCEB8_SZ                         1
#define XCEB7                            7
#define XCEB7_SZ                         1
#define XCEB6                            6
#define XCEB6_SZ                         1
#define XCEB5                            5
#define XCEB5_SZ                         1
#define XCEB4                            4
#define XCEB4_SZ                         1
#define XCEB3                            3
#define XCEB3_SZ                         1
#define XCEB2                            2
#define XCEB2_SZ                         1
#define XCEB1                            1
#define XCEB1_SZ                         1
#define XCEB0                            0
#define XCEB0_SZ                         1
/*----------------------------------------------------------------------*/
/* Define bit fields for Pin Control Register                           */
/*----------------------------------------------------------------------*/
#define XIOEN                           13
#define XIOEN_SZ                         1
#define RIOEN                           12
#define RIOEN_SZ                         1
#define FSXM                            11
#define FSXM_SZ                          1
#define FSRM                            10
#define FSRM_SZ                          1
#define CLKXM                            9
#define CLKXM_SZ                         1
#define CLKRM                            8
#define CLKRM_SZ                         1
#define CLKS_STAT                        6
#define CLKS_STAT_SZ                     1
#define DX_STAT                          5
#define DX_STAT_SZ                       1
#define DR_STAT                          4
#define DR_STAT_SZ                       1
#define FSXP                             3
#define FSXP_SZ                          1
#define FSRP                             2
#define FSRP_SZ                          1
#define CLKXP                            1
#define CLKXP_SZ                         1
#define CLKRP                            0
#define CLKRP_SZ                         1
/******************************/
/* Register Definition  MCBSP */
/******************************/
```

```
/*----------------PORT----------------------------------|-2--|--1--|-0--|*/
#define SPCR1_ADDR(port)     (port ? 0x49 : 0x39)
#define SPCR2_ADDR(port)                    (port ? 0x49 : 0x39)
#define SPSA_ADDR(port)                     (port ? 0x48 : 0x38)
#define SPSD_ADDR(port)                     (port ? 0x49 : 0x39)
#define DRR2_ADDR(port)                     (port ? 0x40 : 0x20)
#define DRR1_ADDR(port)                     (port ? 0x41 : 0x21)
#define DXR2_ADDR(port)                     (port ? 0x42 : 0x22)
#define DXR1_ADDR(port)                     (port ? 0x43 : 0x23)
#define MCBSP_ACCSUB_ADDR(port)     (port ? 0x49 : 0x39)
#define SPCR1_SUBADDR                       0x00
#define SPCR2_SUBADDR                       0x01
#define RCR1_SUBADDR                        0x02
#define RCR2_SUBADDR                        0x03
#define XCR1_SUBADDR                       0x04
#define XCR2_SUBADDR                        0x05
#define SRGR1_SUBADDR                       0x06
#define SRGR2_SUBADDR                       0x07
#define MCR1_SUBADDR                        0x08
#define MCR2_SUBADDR                        0x09
#define RCERA_SUBADDR                       0x0A
#define RCERB_SUBADDR                       0x0B
#define XCERA_SUBADDR                       0x0C
#define XCERB_SUBADDR                       0x0D
#define PCR_SUBADDR                         0x0E
/*****************DMA   Registers, Bits, Bitfields*****************/
/*-------------------------------------------------------------------------*/
/*  Define bit fields for DMPRE Register                                   */
/*-------------------------------------------------------------------------*/
#define DMA_FREE                            15
#define DMA_FREE_SZ                          1
#define DPRC                                 8
#define DPRC_SZ                              6
#define DPRC5                               13
#define DPRC5_SZ                             1
#define DPRC4                               12
#define DPRC4_SZ                             1
#define DPRC3                               11
#define DPRC3_SZ                             1
#define DPRC2                               10
#define DPRC2_SZ                             1
#define DPRC1                                9
#define DPRC1_SZ                             1
#define DPRC0                                8
#define DPRC0_SZ                             1
#define INTSEL                               6
#define INTSEL_SZ                            2
#define DE                                   0
#define DE_SZ                                6
#define DE5                                  5
#define DE5_SZ                               1
#define DE4                                  4
#define DE4_SZ                               1
#define DE3                                  3
#define DE3_SZ                               1
#define DE2                                  2
```

```
#define DE2_SZ                                  1
#define DE1                                     1
#define DE1_SZ                                  1
#define DE0                                     0
#define DE0_SZ                                  1
/*---------------------------------------------------------------------------*/
/*  Define bit fields for DMSEFCn Register                                   */
/*---------------------------------------------------------------------------*/
#define FRAMECOUNT                              0
#define FRAMECOUNT_SZ                           8
#define DSYN                                   12
#define DSYN_SZ                                 4
#define DLBW                                   11
#define DLBW_SZ                                 1
/*---------------------------------------------------------------------------*/
/*  Define bit fields for DMMRCn Register                                    */
/*---------------------------------------------------------------------------*/
#define AUTOINIT                               15
#define AUTOINIT_SZ                             1
#define DINM                                   14
#define DINM_SZ                                 1
#define IMOD                                   13
#define IMOD_SZ                                 1
#define CTMOD                                  12
#define CTMOD_SZ                                1
#define SLAXS                                  11
#define SLAXS_SZ                                1
#define SIND                                    8
#define SIND_SZ                                 3
#define DMS                                     6
#define DMS_SZ                                  2
#define DLAXS                                   5
#define DLAXS_SZ                                1
#define DIND                                    2
#define DIND_SZ                                 3
#define DMD                                     0
#define DMD_SZ                                  2
/****************************/
/* Register Definition  DMA */
/****************************/
#define DMPREC      *(volatile unsigned int*)0x54
#define DMPRE_ADDR  0x54
#define DMSBA_ADDR                          0x55
#define DMSAI_ADDR                          0x56
#define DMA_ACCSUB_ADDR                     0x57
#define DMSRC_SUBADDR                       0x00
#define DMDST_SUBADDR                       0x01
#define DMCTR_SUBADDR                       0x02
#define DMSEFC_SUBADDR                      0x03
#define DMMCR_SUBADDR                       0x04
#define DMSRCP_SUBADDR                      0x1E
#define DMDSTP_SUBADDR                      0x1F
#define DMIDX0_SUBADDR                      0x20
#define DMIDX1_SUBADDR                      0x21
#define DMFRI0_SUBADDR                      0x22
#define DMFRI1_SUBADDR                      0x23
```

```
#define DMGSA_SUBADDR                   0x24
#define DMGDA_SUBADDR                   0x25
#define DMGCR_SUBADDR                   0x26
#define DMGFR_SUBADDR                   0x27
/*********************************************************************/
/* Subregister Read / Write

*/
/*********************************************************************/
#define MCBSP_SUBREG_WRITE(port, subaddr, value) \
 ((REG_WRITE(SPSA_ADDR(port), subaddr)), (REG_WRITE(MCBSP_ACCSUB_ADDR(port), value)))
#define MCBSP_SUBREG_READ(port, subaddr) \
      ((REG_WRITE(SPSA_ADDR(port), subaddr)), (REG_READ(MCBSP_ACCSUB_ADDR(port))))
#define DMA_SUBREG_WRITE(chan, subaddr, value)((subaddr>=0x1E) ?\
      (REG_WRITE(DMSBA_ADDR, (subaddr)), REG_WRITE(DMSAI_ADDR, value))\
      :(REG_WRITE(DMSBA_ADDR, (chan*5+subaddr)), REG_WRITE(DMSAI_ADDR, value)) )
#define DMA_SUBREG_READ(chan, subaddr)((subaddr>=0x1E) ?\
      (REG_WRITE(DMSBA_ADDR, (subaddr)), REG_READ(DMSAI_ADDR))\
      :(REG_WRITE(DMSBA_ADDR, (chan*5+subaddr)), REG_READ(DMSAI_ADDR)) )
/*********************************************************************/
/* Subregister Bit Field Read / Write
                                                    */
/*********************************************************************/
#define MCBSP_SUBREG_BITWRITE(port, subaddr, bit, size, value) \
      REG_WRITE(MCBSP_ACCSUB_ADDR(port), (((REG_WRITE(SPSA_ADDR(port), subaddr),
REG_READ(MCBSP_ACCSUB_ADDR(port))) & ~CREATE_FIELD(bit, size)) | ((value) << (bit)) ) )
#define MCBSP_SUBREG_BITREAD(port, subaddr, bit, size) \
                                    (unsigned int) (REG_WRITE(SPSA_ADDR(port), subaddr),
(REG_READ(MCBSP_ACCSUB_ADDR(port)) & CREATE_FIELD(bit, size)) >>(bit) )
#define DMA_SUBREG_BITWRITE(chan, subaddr, bit, size, value)((subaddr>=0x1E) ?\
                (REG_WRITE(DMSBA_ADDR, (((REG_WRITE(DMSBA_ADDR, subaddr),
REG_READ(DMA_ACCSUB_ADDR)) & ~CREATE_FIELD(bit, size)) | ((value) << (bit)) ) ) )\
                 :(REG_WRITE(DMSBA_ADDR, (((REG_WRITE(DMSBA_ADDR, (chan)),
(REG_READ(DMA_ACCSUB_ADDR) & CREATE_FIELD(bit, size))>>(bit)))))))
#define DMA_SUBREG_BITREAD(chan, subaddr, bit, size)((subaddr>=0x1E) ?\
     ((unsigned int) (REG_WRITE(DMSBA_ADDR, subaddr), (REG_READ(DMA_ACCSUB_ADDR) &
CREATE_FIELD(bit, size)) >>(bit) )):((unsigned int) (REG_WRITE(DMSBA_ADDR, (chan*5+subaddr)),
(REG_READ(DMA_ACCSUB_ADDR) & CREATE_FIELD(bit, size)) >>(bit) )))
 /*---------------------------------------------------------------------------*/
/*  |    The following part of 54XXregs.h was not needed for my purposes. */
/*  |    It has already been included in the regs.h I have received from  */
/*  \/   Karen Baldwin (?) some time ago. */
/*********************/
/* Interrupt Vectors */
/*********************/
#define BASE_VEC_ADR                    0x80
#define RESET_VEC                       0x0
#define NMI_VEC                         4
#define SINT17_VEC                      8
#define SINT18_VEC                      12
#define SINT19_VEC                      16
#define SINT20_VEC                      20
#define SINT21_VEC                      24
#define SINT22_VEC                      28
#define SINT23_VEC                      32
#define SINT24_VEC                      36
```

```
#define SINT25_VEC                      40
#define SINT26_VEC                      44
#define SINT27_VEC                      48
#define SINT28_VEC                      52
#define SINT29_VEC                      56
#define SINT30_VEC                      60
#define INT0_VEC                        64
#define INT1_VEC                        68
#define INT2_VEC                        72
#define TINT0_VEC                       76
#define RINT0_VEC                       80
#define XINT0_VEC                       84
#define DMAC0_VEC                       88
#define TINT1_VEC                       92
#define INT3_VEC                        96
#define HPI_VEC                        100
#define RINT1_VEC                      104
#define XINT1_VEC                      108
#define DMAC2_VEC                      104
#define DMAC3_VEC                      108
#define DMAC4_VEC                      112
#define DMAC5_VEC                      116
/*************************************************************************/
/* Define data structures for all memory mapped registers                */
/*************************************************************************/
/*-----------------------------------------------------------------*/
/* Data bitfields Period for Timer                                 */
/*-----------------------------------------------------------------*/
#define PSC                             6
#define PSC_SZ                          4
#define TRB                             5
#define TRB_SZ                          1
#define TSS                             4
#define TSS_SZ                          1
#define TDDR                            0
#define TDDR_SZ                         4
/*-----------------------------------------------------------------*/
/* Data bitfields for Clock Mode Register                          */
/*-----------------------------------------------------------------*/
#define PLLMUL                         12
#define PLLMUL_SZ                       4
#define PLLDIV                         11
#define PLLDIV_SZ                       1
#define PLLCOUNT                        3
#define PLLCOUNT_SZ                     8
#define PLLON_OFF                       2
#define PLLON_OFF_SZ                    1
#define PLLNDIV                         1
#define PLLNDIV_SZ                      1
#define PLLSTATUS                       0
#define PLLSTATUS_SZ                    1
/*-----------------------------------------------------------------*/
/* Define bit fields for Software Wait State Register              */
/*-----------------------------------------------------------------*/
#define IO                             12
#define IO_SZ                           3
```

```
#define DATA_HI                                 9
#define DATA_HI_SZ                              3
#define DATA_LO                                 6
#define DATA_LO_SZ                              3
#define PROGRAM_HI                              3
#define PROGRAM_HI_SZ                           3
#define PROGRAM_LO                              0
#define PROGRAM_LO_SZ                           3
/*-------------------------------------------------------------------*/
/* Define bitfields for Bank Switch Control Register       */
/*-------------------------------------------------------------------*/
#define BNKCMP                                 12
#define BNKCMP_SZ                               4
#define PS_DS                                  11
#define PS_DS_SZ                                1
#define HBH                                     2
#define HBH_SZ                                  1
#define BH                                      1
#define BH_SZ                                   1
#define EXIO                                    0
#define EXIO_SZ                                 1
/*-------------------------------------------------------------------*/
/* Define bitfields for Interruput Mask Register             */
/*-------------------------------------------------------------------*/
#define INT0                                    0
#define INT1                                    1
#define INT2                                    2
#define TINT0                                   3
#define RINT0                                   4
#define XINT                                    5
#define RINT2                                   6
#define DMAC0                                   6
#define XINT2                                   7
#define DMAC1                                   7
#define INT3                                    8
#define HPINT                                   9
#define RINT1                                  10
#define DMAC2                                  10
#define XINT1                                  11
#define DMAC3                                  11
#define DMAC4                                  12
#define DMAC5                                  13
/*-------------------------------------------------------------------------------------------*/
/* DEFINE DATA STRUCTURE FOR HOST PORT INTERFACE CONTROL REG   */
/*-------------------------------------------------------------------------------------------*/
#define BOB                                     0
#define SMOD                                    1
#define DSPINT                                  2
#define HINT                                    3
#define XHPIA                                   4
/*****************************************************************/
/* Define Interrupt Flag and Interrupt Mask Registers                    */
/*****************************************************************/
#define IMR                         *(volatile unsigned int*)0x00
#define IMR_ADDR                                0x0
#define IFR                         *(volatile unsigned int*)0x01
```

```
#define IFR_ADDR                                0x1
/*********************************************************************/
/* NOTE:  YOU CAN ACCESS THESE REGISTERS IN THIS MANNER ONLY  */
/* IF THE SUBADDRESS REGISTER HAS BEEN DEFINED ALREADY          */
/*********************************************************************/
/*********************************************************************/
/* MultiChannel Buffer Serial 0 defined for 54XX                     */
/*********************************************************************/
#define SPCR10                          *(volatile unsigned int*)0x39
#define SPCR10_ADDR                             0x39
#define SPCR20                          *(volatile unsigned int*)0x39
#define SPCR20_ADDR                             0x39
#define DRR20                           *(volatile unsigned int*)0x20
#define DRR20_ADDR                              0x20
#define DRR10                           *(volatile unsigned int*)0x21
#define DRR10_ADDR                              0x21
#define DXR20                           *(volatile unsigned int*)0x22
#define DXR20_ADDR                              0x22
#define DXR10                           *(volatile unsigned int*)0x23
#define DXR10_ADDR                              0x23
/*********************************************************************/
/* MultiChannel Buffer Serial 1 defined for 54XX                     */
/*********************************************************************/
#define SPCR11                          *(volatile unsigned int*)0x49
#define SPCR11_ADDR                             0x49
#define SPCR21                          *(volatile unsigned int*)0x49
#define SPCR21_ADDR                             0x49
#define DRR21                           *(volatile unsigned int*)0x40
#define DRR21_ADDR                              0x40
#define DRR11                           *(volatile unsigned int*)0x41
#define DRR11_ADDR                              0x41
#define DXR21                           *(volatile unsigned int*)0x42
#define DXR21_ADDR                              0x42
#define DXR11                           *(volatile unsigned int*)0x43
#define DXR11_ADDR                              0x43
#define SPCR12                          *(volatile unsigned int*)0x35
#define SPCR12_ADDR                             0x35
/*********************************************************************/
/* MultiChannel Buffer Serial 2 defined for 54XX                     */
/*********************************************************************/
/*
#define SPCR22                          *(volatile unsigned int*)0x35
#define SPCR22_ADDR                             0x35
#define DRR22                           *(volatile unsigned int*)0x30
#define DRR22_ADDR                              0x30
#define DRR12                           *(volatile unsigned int*)0x31
#define DRR12_ADDR                              0x31
#define DXR22                           *(volatile unsigned int*)0x32
#define DXR22_ADDR                              0x32
#define DXR12                           *(volatile unsigned int*)0x33
#define DXR12_ADDR                              0x33
*/
/*********************************************************************/
/* Direct Memory Access defined for 54XX
                                                                     */
/*********************************************************************/
```

```
#define DMPRE                                   (0x54)
#define DMSBA                                   (0x55)
#define DMSAI                                   (0x56)
#define DMSRCP                                  (0x57)
#define DMDSTP                                  (0x57)
#define DMGSA                                   (0x57)
#define DMGDA                                   (0x57)
#define DMGCR                                   (0x57)
#define DMGFR                                   (0x57)
#define DMFRI(reg) ((reg) ? 0x57:0x57)
#define DMIDX(reg) ((reg) ? 0x57:0x57)
#define DMSRC(channel) ((channel) ? 0x57:0x57)
#define DMDST(channel) ((channel) ? 0x57:0x57)
#define DMCTR(channel) ((channel) ? 0x57:0x57)
#define DMSEFC(channel) ((channel) ? 0x57:0x57)
#define DMMCR(channel) ((channel) ? 0x57:0x57
#define DMA_REG_READ(dma_subaddress, channel) (DMSAI(channel)=dma_subaddress), *(volatile unsigned
int*) DMFRI(channel))
/*---------------------------------------------------------------------*/
/* Data bitfields Period for DMPRE                                     */
/*---------------------------------------------------------------------*/
#define DPRC5                                  13
#define DPRC5_SZ                                1
#define DPRC4                                  12
#define DPRC4_SZ                                1
#define DPRC3                                  11
#define DPRC3_SZ                                1
#define DPRC2                                  10
#define DPRC2_SZ                                1
#define DPRC1                                   9
#define DPRC1_SZ                                1
#define DPRC0                                   8
#define DPRC0_SZ                                1
#define INTSEL                                  6
#define INTSEL_SZ                               2
#define DE5                                     5
#define DE5_SZ                                  1
#define DE4                                     4
#define DE4_SZ                                  1
#define DE3                                     3
#define DE3_SZ                                  1
#define DE2                                     2
#define DE2_SZ                                  1
#define DE1                                     1
#define DE1_SZ                                  1
#define DE0                                     0
#define DE0_SZ                                  1
/*---------------------------------------------------------------------*/
/* Data bitfields Period for DMSEFCn                                   */
/*---------------------------------------------------------------------*/
#define DSYN                                   12
#define DSYN_SZ                                 4
#define FRAME_CNT                               0
#define FRAME_CNT_SZ                            8
/*---------------------------------------------------------------------*/
/* Data bitfields Period for Mode Control Register                     */
```

```
/*----------------------------------------------------------*/
#define AUTOINIT                                15
#define AUTOINIT_SZ                              1
#define DINM                                    14
#define DINM_SZ                                  1
#define IMOD                                    13
#define IMOD_SZ                                  1
#define CTMOD                                   12
#define CTMOD_SZ                                 1
#define SIND                                     8
#define SIND_SZ                                  3
#define DMS                                      6
#define DMS_SZ                                   2
#define DIND                                     2
#define DIND_SZ                                  3
#define DMD                                      0
#define DMD_SZ                                   2
/************************************************************/
/* TIMER REGISTER ADDRESSES   (TIM0 = Timer 0, TIM1 = Timer 1 */
/* Defined for all devices                                  */
/************************************************************/
#define TIM_ADDR(port) (port ? 0x30 : 0x24)
#define TIM(port)               *(volatile unsigned int*)TIM_ADDR(port)
#define PRD_ADDR(port)                  (port ? 0x31 : 0x25)
#define PRD(port)               *(volatile unsigned int*)PRD_ADDR(port)
#define TCR_ADDR(port)                  (port ? 0x32 : 0x26)
#define TCR(port)               *(volatile unsigned int*)TCR_ADDR(port)
/************************************************************/
/* EXTERNAL BUS CONTROL REGISTERS                           */
/************************************************************/
#define BSCR                        *(volatile unsigned int*)0x29
#define BSCR_ADDR                           0x29
#define SWCR                        *(volatile unsigned int*)0x2B
#define SWCR_ADDR                           0x2B
#define SWWSR                       *(volatile unsigned int*)0x28
#define SWWSR_ADDR                          0x28
/************************************************************/
/* HOST PORT INTERFACE REGISTER ADDRESS                     */
/* Defined for C54XX                                        */
/************************************************************/
#define HPIC                        *(volatile unsigned int*)0x2C
#define HPIC_ADDR                           0x2C
#define HPI_ADDR                           0x1000
/************************************************************/
/* Defined flags for use in setting control for HPI host interface control pins */
/* The value of these constants is their relative bit position in */
/* the control structure for the host side of the HPI interface */
/************************************************************/
#define HAS_PIN                                  0
#define HBIL_PIN                                 1
#define HCNTL0_PIN                               2
#define HCNTL1_PIN                               3
#define HCS_PIN                                  4
#define HD0_PIN                                  5
#define HDS1_PIN                                 6
#define HDS2_PIN                                 7
```

```
#define HINT_PIN                    8
#define HRDY_PIN                    9
#define HRW_PIN                     10
/*********************************************************************/
/* CLOCK MODE REGISTER ADDRESS                                       */
/* Defined for C54XX                                                 */
/*********************************************************************/
#define CLKMD                   *(volatile unsigned int*)0x58
#define CLKMD_ADDR                  0x58
/*********************************************************************/
/* Extended Program Counter -XPC register                            */
/*********************************************************************/
extern volatile unsigned int XPC;
#define XPC                     *(volatile unsigned int*)0x1e
#define XPC_ADDR                    0x1e
/*********************************************************************/
/* Program Control and Status Registers (PMST, ST0, ST1)             */
/*********************************************************************/
#define PMST                    *(volatile unsigned int*)0x1d
#define PMST_ADDR                   0x1d
#define ST0                     *(volatile unsigned int*)0x06
#define ST0_ADDR                    0x06
#define ST1                     *(volatile unsigned int*)0x07
#define ST1_ADDR                    0x07
/*********************************************************************/
/* General-purpose I/O pins control registers (GPIOCR, GPIOSR)       */
/*********************************************************************/
#define GPIOCR                  *(volatile unsigned int*)0x3C
#define GPIOCR_ADDR                 0x3C
#define GPIOSR                  *(volatile unsigned int*)0x3D
#define GPIOSR_ADDR                 0x3D
#define __54XXREGS
#endif
```

E.12 timr5416.h [40]

```
/*****************************************************************************/
/* Define Timer Period, and Control Registers with all related data structures, macros, and functions */
/*****************************************************************************/
#include "regs5416.h"
/*************************************************************/
/* TIMER_START - starts timer operation                      */
/*************************************************************/
#define TIMER_START(port)TCR(port) &= ~MASK_BIT(TSS)
/*************************************************************/
/* TIMER_HALT - halts timer operation                        */
/*************************************************************/
#define TIMER_HALT(port)TCR(port) |= MASK_BIT(TSS)
/*************************************************************/
/* TCR_MASK - creates mask to set relevant fields in Timer Cntrl register */
/*   trb    - val to set timer reload bit */
/*   tss    - val to set timer stop/start bit */
/*   tddr   - val to set timer divide-down ratio */
/*************************************************************/
#define TCR_MASK(trb, tss, tddr)\
       ((trb ? MASK_BIT(TRB) | MASK_TARGET_WORD: ~MASK_BIT(TRB))&\
        (tss ? MASK_BIT(TSS) | MASK_TARGET_WORD: ~MASK_BIT(TSS))&\
        (tddr ? (MASK_FIELD(TDDR,tddr, TDDR_SZ) | MASK_TARGET_WORD) :\
             ~MASK_FIELD(TDDR,tddr, TDDR_SZ)))
/*************************************************************/
/* TIMER_INIT (ctrl, prd) - init and start timer             */
/*  ctrl - mask used to set timer control register           */
/*  prd  - value to set timer period register                */
/*************************************************************/
#define TIMER_INIT(port, ctrl, prd){ SET_BIT(TCR_ADDR(port),TSS);TCR(port) = ctrl;\
                          PRD(port) = prd;TIMER_RELOAD(port); }
/*************************************************************/
/* timer_reset - reset timer to conditions defined by device reset */
/*  i.e.  period = 0xffff, tddr = 0x000 */
/*************************************************************/
#define TIMER_RESET(port){ TIMER_HALT(port);\
          LOAD_FIELD(TCR_ADDR(port), 0x000, TDDR, TDDR_SZ); \
          PRD(port) = 0xffff;TIMER_START(port); }
/*************************************************************/
/* TIMER_RELOAD() - reloads timer with previsouly set period value, etc..  */
/*************************************************************/
#define TIMER_RELOAD(port){ TIMER_HALT(port);TCR(port) |= MASK_BIT(TRB);\
                                    TIMER_START(port); }
/*************************************************************/
/* TIMER_READ - reads current value of timer                 */
/*************************************************************/
#define TIMER_READ(port)TIM(port)
/*************************************************************/
/* MASK_CLKMD(PLLMUL, PLLDIV, PLLCOUNT, PLLONOFF, PLLNDIV) */
/* creates mas kto set PLL clock mode register */
/* PLLMUL - defines frequency multiplier */
/* PLLDIV - PLL divider is used with PLLMUL & PLLNDIV to define multiplier frequency */
/* PLLCOUNT - # of cycles for PLL to count before processor is clocked */
/* PLLONOFF - enables/disables the analog part of the PLL */
```

```
/* PLLNDIV  - in conjunction with PLLMIL & PLLDIV determines value of multiplier. */
/*****************************************************************/
#define MASK_CLKMD(pllmul, plldiv, pllcount, pllonoff, pllndiv)\
     ((pllmul ? (MASK_FIELD(PLLMUL, pllmul, PLLMUL_SZ) | MASK_TARGET_WORD):\
       ~MASK_FIELD(PLLMUL, 0xf, PLLMUL_SZ))     &\
      (plldiv ? (MASK_BIT(PLLDIV) | MASK_TARGET_WORD) :\
       ~MASK_BIT(PLLDIV))           &\
      (pllcount ? (MASK_FIELD(PLLCOUNT,pllcount, PLLCOUNT_SZ) | MASK_TARGET_WORD):\
       ~MASK_FIELD(PLLCOUNT,0x7f, PLLCOUNT_SZ)) &\
      (pllonoff ? (MASK_BIT(PLLONOFF) | MASK_TARGET_WORD): \
       ~MASK_BIT(PLLONOFF))       &\
      (pllndiv ? MASK_BIT(PLLNDIV) | MASK_TARGET_WORD :\
       ~MASK_BIT(PLLNDIV)))
/*****************************************************************/
/* CLOCK_RESET(ctrl) - resets clock mode register                */
/* ctrl - mask to set control register                           */
/*****************************************************************/
#define CLOCK_RESET(ctrl)CLKMD = ctrl
```

Appendix F

Source Files

F.1 5416_dsk.c [6]

```
typedef unsigned long       Uint32;
#include "AED.h"
#include "dsk5416.h"
#include "timr5416.h"
#include "intr5416.h"
#include "regs5416.h"
#if AED_PRINT /* Must be placed after AED.h */
#include <stdio.h>
#endif
#define OK                          0
#undef  ERROR
#define ERROR                       -1
/* timer to be used to for delays    */
#define DELAY_TIMER             INTR_TIMER0
/* timer to be used FPGA simulation  */
#define PERIOD_TIMER            INTR_TIMER0
#define PORT(INTR_TIMER)        (INTR_TIMER-1)
#define DEFAULT_PRESCALE    9
#define DC_WIDE                 4
#define XCNTL0_MASK             0x01
#define XCNTL1_MASK             0x02
#define XSTAT0_MASK             0x10
#define XSTAT1_MASK             0x20
/* Interrupt ISN translation table, account for processor dependant */
/* connections to the expansion bus and daughterboard               */
const int  isn_trap[]={
  RS_TRAP,          /* INTR_DSPINT                  0x0 */  /*UNUSED*/
  TINT_TRAP,        /* INTR_TIMER0                  0x1 */  /*TIMERS*/
  RS_TRAP,          /* INTR_TIMER1                  0x2 */
  HPINT_TRAP,   /* INTR_HPI                     0x3 */
  INT1_TRAP,        /* INTR_Alt Intr                0x4 */
  INT2_TRAP,        /* INTR_Alt Intr                0x5 */
  INT3_TRAP,        /* INTR_Alt Intr                0x6 */
  INT0_TRAP,        /* INTR_DBOARD                  0x7 */  /*DAUGHTER BOARD INTR*/
  DMAC2_TRAP,   /* INTR_DMA0                    0x8 */  /*DMA*/
  DMAC3_TRAP,   /* INTR_DMA1                    0x9 */
  DMAC4_TRAP,   /* INTR_DMA2                    0xA */
  DMAC5_TRAP,   /* INTR_DMA3                    0xB */
  XINT0_TRAP,    /* INTR_MCBSP_TRANS_0      0xC */
  RINT0_TRAP,    /* INTR_MCBSP_REC_0        0xD */
  XINT1_TRAP,    /* INTR_MCBSP_TRANS_1      0xE */
  RINT1_TRAP,    /* INTR_MCBSP_REC_1        0xF */
};
const int  isn_flag[]={
  0,        /* INTR_DSPINT                      0x0 */  /*UNUSED*/
  TINT,     /* INTR_TIMER0                      0x1 */  /*TIMERS*/
  0,        /* INTR_TIMER1                      0x2 */
  HINT,     /* INTR_HPI                         0x3 */
  INT1,     /* INTR_Alt Intr                    0x4 */
```

```
  INT2,    /* INTR_Alt Intr                    0x5 */
  INT3,    /* INTR_Alt Intr                    0x6 */
  INT0,    /* INTR_DBOARD                      0x7 */  /*DAUGHTER BOARD INTR*/
  DMAC2,   /* INTR_DMA0                        0x8 */  /*DMA*/
  DMAC3,   /* INTR_DMA1                        0x9 */
  DMAC4,   /* INTR_DMA2                        0xA */
  DMAC5,   /* INTR_DMA3                        0xB */
  XINT0,   /* INTR_MCBSP_TRANS_0               0xC */
  RINT0,   /* INTR_MCBSP_REC_0                 0xD */
  XINT1,   /* INTR_MCBSP_TRANS_1               0xE */
  RINT1,   /* INTR_MCBSP_REC_1                 0xF */
};
unsigned int cpld_misc_reg_default;
unsigned int cpld_cntl_reg_default;
static unsigned int cpuFreqInMhz;
/************************************************************************
   alloc_timer_intr - Provide an interrupt at specified period
   Returns:    ISN for interrupt at period timing
   Parameters:  IN period_in - desired period in 500 ns increments
************************************************************************/
IntrSelNumType alloc_timer_intr (unsigned long period_in) {
 unsigned int period_reg, ctrl_reg;
 /* period_in is in 0.5 usec units */
  period_reg = ((cpuFreqInMhz/(DEFAULT_PRESCALE+1))*period_in)>>1;

  ctrl_reg =  MASK_BIT(TSS)|DEFAULT_PRESCALE;  /* stop timer */

  TIMER_INIT(PORT(PERIOD_TIMER), ctrl_reg, period_reg);
  return PERIOD_TIMER; /* selected period timer */
}
/************************************************************************
  board_init - Initialize EVM or DSK board for use
************************************************************************/
void board_init (void) {
  DSK5416_DC_REG = 0x08;  /* DB reset */
  /* change the clock mode to 160MHz with CLKOUT at 80MHz */
  CLKMD = 0x9007;  /* reset value for x10 mode */
  cpuFreqInMhz =160;
  /* daughter board access */
  cpld_misc_reg_default = 0; /* 16 bit default */
#if (TALK_TO_FPGA)
  cpld_cntl_reg_default = 0xd0; /* data address default */
#else
  cpld_cntl_reg_default = 0x40; /* SRAM for test data */
#endif
  DSK5416_USER_REG = 0; /* clear leds */
  DSK5416_MISC = cpld_misc_reg_default;
  DSK5416_CODEC_CLK = 0;
  DSK5416_DM_CNTL = cpld_cntl_reg_default;
  IMR = 0;
  BSCR = 0x2002; /* bus hold and CLKOUT = CLK/2 */
  SWWSR = 0x745B; /* data wait = 2 */
  PMST = 0x7FA0; /* MP/MC= 0, ovly=1, DROM=0, IPT=0x0ff */
  DSK5416_DC_REG = 0x0;  /* take DB out of reset */
  INTR_GLOBAL_ENABLE;
}
```

```
/************************************************************************
  brd_led_enable - Illuminate user LED on board
  Returns: error code indicating incorrect LED for this board
  Parameters:  IN LED_number - number of LED beginning at zero
  Note: The number available will vary with board.
************************************************************************/
int brd_led_enable(int LED_number) {
  DSK5416_USER_REG |= 1<<LED_number; /* set led bit */
  return OK;
} /* brd_led_enable end */
/************************************************************************
  brd_led_disable -  Extinguish LED on EVM board
  Returns: error code indicating incorrect LED for this board
  Parameters:  IN LED_number - number of LED beginning at zero
  Note: The number available will vary with board.
************************************************************************/
int brd_led_disable(int LED_number) {
  DSK5416_USER_REG &= ~(1<<LED_number); /* clear led bit */
  return OK;
} /* brd_led_disable end */
/************************************************************************
  byte_size - Number of address increments in element sizes
  RETURNS: address increments (same as sizeof() returns)
  PARAMETERS: IN esize_code - element size codes for daughterboard transfers
************************************************************************/
int byte_size (unsigned int esize_code) {
const int size [] = {2, 1, 1};
return size[esize_code];
}
/************************************************************************
  cpu_freq - Frequency of internal CPU clock in MHz
  RETURNS: CPU frequence in MHz
************************************************************************/
int cpu_freq (void)
{
  return  cpuFreqInMhz;
;
}
/************************************************************************
  delay_usec -  Delay specified number of microseconds
  RETURN: error code
  PARAMETERS:  IN numUsec - number of microseconds delay
************************************************************************/
int delay_usec (unsigned short numUsec)
{
  unsigned int timer_limit
    = ((unsigned long)cpuFreqInMhz*(unsigned long)numUsec)/(DEFAULT_PRESCALE+1);
  unsigned int start, end;
//  printf ("Limit = %u usec= %u\n",
//     timer_limit, numUsec);
  TIMER_INIT(PORT(DELAY_TIMER), MASK_BIT(TSS)|DEFAULT_PRESCALE, 0xffff);
  TIMER_START(PORT(DELAY_TIMER));
  start = TIMER_READ(PORT(DELAY_TIMER));
  end = start - timer_limit;
  while ((TIMER_READ(PORT(DELAY_TIMER))) > end);
  TIMER_HALT(PORT(DELAY_TIMER));
```

```
//  printf ("Start= %x End=%x Next= %x\n", start, end, next);
  return AED_OK;
}  /* delay_usec */
/*************************************************************************
  delay_msec -  Delay specified number of milliseconds
  RETURN: error code
  PARAMETERS:  IN numMsec - number of microseconds delay
*************************************************************************/
int delay_msec (unsigned short numMsec)
{
  unsigned short j;
/*  printf ("Msec=%u\n", numMsec); */
  for (j=0; j<numMsec; j++)
    delay_usec(999);
  return AED_OK;
} /* end delay_msec */
/*************************************************************************
 FPGA_enable - Enable the FPGA to receive data
*************************************************************************/
void FPGA_enable (void)
{
  DSK5416_DC_REG |= XCNTL1_MASK;
} /* FPGA_enable */
/*************************************************************************
 FPGA_start - Start the FPGA collecting data
*************************************************************************/
void FPGA_start (void) {
#if (TALK_TO_FPGA)
  unsigned int cpu_intr = isn_flag[INTR_DBOARD];
  unsigned int mask = 1<<(cpu_intr);
  IFR = mask; /* clear interrupt in IFR */
  DSK5416_MISC = cpld_misc_reg_default;
  DSK5416_DM_CNTL = cpld_cntl_reg_default;
  DSK5416_DC_REG |= XCNTL0_MASK;
#else
  TIMER_START(PERIOD_TIMER);
#endif
}
/*************************************************************************
  FPGA_stat_addr - Pointer to FPGA status address
  RETURN: pointer to status address word
*************************************************************************/
unsigned int * FPGA_stat_addr (void) {
 return (unsigned int *)DSK5416_DC_REG;
}
/*************************************************************************
  FPGA_stat_mask - Mask FPGA FIFO overflow bit
  RETURN: mask for overflow bit in status address word
*************************************************************************/
unsigned int FPGA_stat_mask (void) {
        return XSTAT0_MASK;
}
/*************************************************************************
  FPGA_stop - Stop the FPGA collecting data and reset FIFO
*************************************************************************/
void FPGA_stop (void) {
```

```
#if (TALK_TO_FPGA)
  DSK5416_DC_REG &= ~XCNTL0_MASK;
#else
  TIMER_HALT(PERIOD_TIMER);
#endif
}
/************************************************************************
  get_data_addr - Daughterboard data read/write address
  Returns:  void pointer for read/write of data
  Notes:    This board may set bits in registers if necessary to render this address active (like paging bits)
************************************************************************/
void * get_data_addr (void) {
  void *ptr = (void*) 0x8000; /* word address - CE2 */
   return ptr;
}
/************************************************************************
  get_cntl_addr - Daughterboard control read/write address
  Returns:  void pointer for read/write of control
  Notes:  This board may set bits in registers if necessary to render this address active (like paging bits)
************************************************************************/
void * get_cntl_addr (void) {
  void *ptr = (void*) 0x0000; /* word address - CE2 */
  return ptr;
}
/************************************************************************
  interrupt_init - Bind interrupt service routine to an interrupt
    PARAMETERS:  IN ptr_isr - pointer to an interrupt service routine
                 IN isn - interrupt selection number
************************************************************************/
extern unsigned int IntVM[4];
extern unsigned int _vectors[128];
void interrupt_init (void(*ptr_isr)(void), IntrSelNumType isn) {
  unsigned int location = (isn_trap[isn])<<2;
  _vectors[location+0] = IntVM[0];
  _vectors[location+1] = (unsigned long)ptr_isr;
  _vectors[location+2] = IntVM[2];
  _vectors[location+3] = IntVM[3];
} /* end interrupt_init */
/************************************************************************
  intr_pause - Disable ISN
  PARAMETERS:  IN isn - interrupt selection number
************************************************************************/
void intr_pause (IntrSelNumType isn){
   int cpu_intr = isn_flag[isn];
   unsigned int mask = 1<<(cpu_intr);
   IMR &= ~mask; /* disable interrupt in IER1*/
}
/************************************************************************
  intr_start - Enable ISN with cleared flag
  PARAMETERS:  IN isn - interrupt selection number
************************************************************************/
void intr_start (IntrSelNumType isn) {
   unsigned int cpu_intr = isn_flag[isn];
   unsigned int mask = 1<<(cpu_intr);
   printf("Mask=%x isn=%x cpu_intr=%x\n",
       mask, isn, cpu_intr);
```

```
   IFR = mask; /* disable interrupt in IFR1*/
   IMR |= mask; /* enable interrupt in IER1*/
}
/************************************************************************
  mcbsp_freq - Frequency of internal MCBSP clock in MHz
  RETURNS: MCBSP frequency in MHz
************************************************************************/
int mcbsp_freq (void)
{
  return cpuFreqInMhz;
;
}
/************************************************************************
  read_32b_reg - Read 32 bit data from daughterboard
  RETURNS: value read from the specified address
  PARAMETERS:  IN addr - pointer to the read address
  NOTE: This routine transmits 32 bits to the daughterboard for both 16 and 32 bit buses.
************************************************************************/
unsigned long read_32b_reg (unsigned long *addr) {
  unsigned long ret_msb, ret_lsb;
  unsigned int addr0_14 =
     ((unsigned int)addr & 0x7FFF);
  unsigned long *uba = (unsigned long *)0x8000;
  if (!((unsigned long)addr & 0x8000)) {
    DSK5416_DM_CNTL = 0xc8; /* control page */
  }
  uba += addr0_14;
  DSK5416_MISC = DC_WIDE;
  ret_msb = ((unsigned long)(*(unsigned int *)uba));
  ret_lsb = ((unsigned long)(*((unsigned int *)uba+1)));
  DSK5416_MISC = cpld_misc_reg_default;
  DSK5416_DM_CNTL = cpld_cntl_reg_default;
  return (ret_msb<<16) | ret_lsb;
}
/************************************************************************
  write_32b_reg - Write 32 bit data to daughterboard
    PARAMETERS:  IN addr - pointer to the write address
                 IN data - value to be written
    NOTE: This routine transmits 32 bits to the daughterboard for both 16 and 32 bit buses.
************************************************************************/
void write_32b_reg (unsigned long *addr, unsigned long data){
#if(TALK_TO_FPGA)
  unsigned int addr0_14 =
     ((unsigned int)addr & 0x7FFF);
  unsigned long *uba = (unsigned long *)0x8000;
  if (!((unsigned long)addr & 0x8000)) {
    DSK5416_DM_CNTL = 0xc8; /* control page */
  }
  uba += addr0_14;
  DSK5416_MISC = DC_WIDE;
  *((unsigned int *)uba+1) = (unsigned int)(data>>16);
  *(unsigned int *)uba= (unsigned int)data;
  DSK5416_MISC = cpld_misc_reg_default;
  DSK5416_DM_CNTL = cpld_cntl_reg_default;
#endif
}
```

F.2 AED_DMS_4wDMA.c [6]

```
#include <string.h>
#include <limits.h>
#include "intr5416.h"
#include "dma5416.h"
#include "timr5416.h"
#include "AED.h"
#include "AED_DMS.h"
#include "AED_Brd.h"
#if AED_PRINT /* Must be placed after AED.h */
  #include <stdio.h>
#endif
#define SET_FIELD(addr,val,bit,length) \
      (CONTENTS_OF(addr) |= ((unsigned int)(val) << (bit)))
#define CLR_FIELD(addr,bit,length) \
      (CONTENTS_OF(addr) &= ~CREATE_FIELD(bit,length))
typedef struct {
  unsigned int ch;
  unsigned int *src;
  unsigned int srcp;
  unsigned int *dst;
  unsigned int dstp;
  unsigned int cnt;
  unsigned int sfc;
  unsigned int mcr;
  unsigned int gsa;
  unsigned int gda;
  unsigned int gcr;
  unsigned int gfr;
  unsigned int frm;
} Dma5416Config;
extern void interrupt dma_intr0_fs(void);
extern void interrupt dma_intr0_fs_start(void);
extern void interrupt dma_intr0_ws(void);
extern void interrupt dma_intr1_fs(void);
extern void interrupt dma_intr1_fs_start(void);
extern void interrupt dma_intr1_ws(void);
extern void interrupt dma_intr2_fs(void);
extern void interrupt dma_intr2_fs_start(void);
extern void interrupt dma_intr2_ws(void);
extern void interrupt dma_intr3_fs(void);
extern void interrupt dma_intr3_fs_start(void);
extern void interrupt dma_intr3_ws(void);
/*=========================================================================*/
extern unsigned int cpld_misc_reg_default;
volatile int read_err[DMA_CH_NUMBER+1], buf_count[DMA_CH_NUMBER+1];
/*=========================================================================*/
static Dma5416Config dma[DMA_CH_NUMBER];
volatile unsigned int trans_cnt[DMA_CH_NUMBER];
static unsigned int frame_sync[DMA_CH_NUMBER];
static unsigned int frame_cnt[DMA_CH_NUMBER];
static IntrSelNumType sync_isn[DMA_CH_NUMBER];
static unsigned int alloc_chan = 0;
static unsigned int *src_addr[DMA_CH_NUMBER];
```

```
static unsigned int *dst_addr[DMA_CH_NUMBER];
static const unsigned int dma_fs_mode[] = {
    0,        /* read sync mode */
    0,        /* write sync mode */
    1,        /* frame sync cont mode */
    1};       /* frame sync burst mode */
static const unsigned int dma_ie_mode[] = {
    IMOD_FRAME,        /* read sync mode */
    IMOD_FRAME,        /* write sync mode */
    IMOD_FRAME,        /* frame sync cont mode */
    IMOD_BLOCK};       /* frame sync burst mode */
static const IntrSelNumType dma_isn_trans[] = {
  INTR_DMA_0,
  INTR_DMA_1,
  INTR_DMA_2,
  INTR_DMA_3};
static const unsigned int dma_size_trans[] = {
  DBLW_ENABLE,
  DBLW_DISABLE,
  DBLW_DISABLE,
  DBLW_DISABLE};
/*=========================================================================*/
void start_DMA_channel(Dma_channel chan){
//  DMA_AUTO_ENABLE(dma[chan].ch);
} /*end start_DMA*/
/*=========================================================================*/
void operate_DMA_channel(Dma_channel chan){
  DMA_ENABLE(dma[chan].ch);
} /*end start_DMA*/
/*=========================================================================*/
void pause_DMA_channel(Dma_channel chan){
  DMA_DISABLE(dma[chan].ch);
  intr_pause (sync_isn[chan]);
} /*end pause_DMA_channel */
/*=========================================================================*/
int count_DMA_channel(Dma_channel chan){
   return DMA_SUBREG_READ(dma[chan].ch, DMCTR_SUBADDR);
} /* end count_DMA_channel */
/*************************************************************************/
  Dma_channel alloc_DMA_channel (void) {
  Dma_channel dma_chan;
  dma_chan = (Dma_channel) alloc_chan++;
  dma[dma_chan].ch = DMA_CH2 +dma_chan;
  return dma_chan;
} /* end alloc_DMA_channel */
/*=========================================================================*/
void program_DMA_channel(Dma_channel chan, int dir,
          void *dest, void *src, unsigned int count,
          unsigned int nbuf,
          unsigned int size, unsigned int dms_mode) {
  IntrSelNumType dma_isn;
  /* pause the DMA channel before programming */
  pause_DMA_channel(chan);
  dma[chan].sfc = 0;
  dma[chan].mcr = 0;
  dma[chan].gfr = 0;
```

```
/* configure source address */
dma[chan].src = src;
dma[chan].srcp = 0;
src_addr[chan] = src;
/* configure dest address */
dma[chan].dst = dest;
dma[chan].dstp = 0;
dst_addr[chan] = dest;
/* configure transfer counter */
dma[chan].cnt = (dma_size_trans[size]) ? (count*2)-1 : count-1;
dma[chan].frm = nbuf;
frame_cnt[chan] = nbuf;
/*get cpu intr and bind intr routine */
dma_isn = dma_isn_trans[chan];
frame_sync[chan] = dma_fs_mode[DMA_XFER_METHOD(dms_mode)];
  switch (dir) {
  case FROM_FPGA_MEM:
    sync_isn[chan] = INTR_DBOARD;
    break;
  case FROM_FPGA_SER0:
    sync_isn[chan] = INTR_MCBSP_REC_0;
    break;
  case FROM_FPGA_SER1:
    sync_isn[chan] = INTR_MCBSP_REC_1;
    break;
  case TO_FPGA_MEM:
    sync_isn[chan] = INTR_DMA_0;
    break;
  case TO_FPGA_SER0:
    sync_isn[chan] = INTR_MCBSP_TRANS_0;
    break;
  case TO_FPGA_SER1:
    sync_isn[chan] = INTR_MCBSP_TRANS_1;
    break;
  default:
    error_flashing(AED_FLASH_FEATURE_NOT_IMPLEMENTED);
  } /* end switch */
if (frame_sync[chan]) {
  /* For the frame sync mode, set up a
  dma to transfer one frame, use an isr
  to process the sync interrupt */
  switch (chan) {
  case 0:
    interrupt_init(dma_intr0_fs, dma_isn);
    interrupt_init(dma_intr0_fs_start, sync_isn[0]);
    break;
  case 1:
    interrupt_init(dma_intr1_fs, dma_isn);
    interrupt_init(dma_intr1_fs_start, sync_isn[1]);
    break;
  case 2:
    interrupt_init(dma_intr2_fs, dma_isn);
    interrupt_init(dma_intr2_fs_start, sync_isn[2]);
    break;
  case 3:
    interrupt_init(dma_intr3_fs, dma_isn);
```

```
  interrupt_init(dma_intr3_fs_start, sync_isn[3]);
  break;
 } /* end switch */
 intr_start(dma_isn);
 switch (dir) {
 case FROM_FPGA_MEM:
  SET_FIELD(&dma[chan].mcr, LAXS_EXTERNAL, SLAXS, SLAXS_SZ);
  SET_FIELD(&dma[chan].mcr, LAXS_INTERNAL, DLAXS, DLAXS_SZ);
  SET_FIELD(&dma[chan].mcr, SPACE_DATA, DMS, DMS_SZ);
  SET_FIELD(&dma[chan].mcr, SPACE_DATA, DMD, DMD_SZ);
  SET_FIELD(&dma[chan].mcr,
    (dma_size_trans[size]) ? INDEXMODE_INC : INDEXMODE_NOMOD,
    SIND, SIND_SZ);
  SET_FIELD(&dma[chan].mcr, INDEXMODE_INC, DIND, DIND_SZ);
  break;
 case FROM_FPGA_SER1:  case FROM_FPGA_SER0:
  SET_FIELD(&dma[chan].mcr, LAXS_INTERNAL, SLAXS, SLAXS_SZ);
  SET_FIELD(&dma[chan].mcr, LAXS_INTERNAL, DLAXS, DLAXS_SZ);
  SET_FIELD(&dma[chan].mcr, SPACE_IO, DMS, DMS_SZ);
  SET_FIELD(&dma[chan].mcr, SPACE_DATA, DMD, DMD_SZ);
  SET_FIELD(&dma[chan].mcr, INDEXMODE_NOMOD, SIND, SIND_SZ);
  SET_FIELD(&dma[chan].mcr, INDEXMODE_INC, DIND, DIND_SZ);
 break;
 case TO_FPGA_MEM:
  SET_FIELD(&dma[chan].mcr, LAXS_INTERNAL, SLAXS, SLAXS_SZ);
  SET_FIELD(&dma[chan].mcr, LAXS_EXTERNAL, DLAXS, DLAXS_SZ);
  SET_FIELD(&dma[chan].mcr, SPACE_DATA, DMS, DMS_SZ);
  SET_FIELD(&dma[chan].mcr, SPACE_DATA, DMD, DMD_SZ);
  SET_FIELD(&dma[chan].mcr, INDEXMODE_INC, SIND, SIND_SZ);
  SET_FIELD(&dma[chan].mcr,
     (dma_size_trans[size]) ? INDEXMODE_INC : INDEXMODE_NOMOD,
     DIND, DIND_SZ);
  break;
 case TO_FPGA_SER1:  case TO_FPGA_SER0:
  SET_FIELD(&dma[chan].mcr, LAXS_INTERNAL, SLAXS, SLAXS_SZ);
  SET_FIELD(&dma[chan].mcr, LAXS_INTERNAL, DLAXS, DLAXS_SZ);
  SET_FIELD(&dma[chan].mcr, SPACE_DATA, DMS, DMS_SZ);
  SET_FIELD(&dma[chan].mcr, SPACE_IO, DMD, DMD_SZ);
  SET_FIELD(&dma[chan].mcr, INDEXMODE_INC, SIND, SIND_SZ);
  SET_FIELD(&dma[chan].mcr, INDEXMODE_NOMOD, DIND, DIND_SZ);
  break;
 default:
  error_flashing(AED_FLASH_FEATURE_NOT_IMPLEMENTED);
 } /* end switch */
 /* set mode control */
 SET_BIT(&dma[chan].mcr, DINM);
 SET_FIELD(&dma[chan].mcr, dma_ie_mode[dms_mode], IMOD, IMOD_SZ);
 /* set the priority high */
 DMPREC |= MASK_BIT(DPRC0+dma[chan].ch);
 SET_FIELD(&DMPREC, INTSEL_01, INTSEL, INTSEL_SZ);
} else {
 /* For word sync mode process with an isr only */
 switch (chan) {
 case 0:
  interrupt_init(dma_intr0_ws, sync_isn[0]);
  break;
```

```
    case 1:
      interrupt_init(dma_intr1_ws, sync_isn[1]);
      break;
    case 2:
      interrupt_init(dma_intr2_ws, sync_isn[2]);
      break;
    case 3:
      interrupt_init(dma_intr3_ws, sync_isn[3]);
      break;
    } /* end switch */
  } /* end if */
  /* set word size */
  cpld_misc_reg_default = (dma_size_trans[size]) ? 0x04 : 0x00;
  /* initialize globals */
  read_err[chan] = 0;
  buf_count[chan] = 0;
  trans_cnt[chan] = dma[chan].cnt;
  intr_start(sync_isn[chan]);
} /* end init_DMA_channel */
/*=========================================================================*/
void test_DMA_channel (Dma_channel chan, void *src,
                       unsigned long period) {
  IntrSelNumType timer_isn;
  /* this function only covers FROM_FPGA_MEM */
  timer_isn = alloc_timer_intr(period);
  /* reconfigure source address -
       Change source address
       convert to byte address */
  dma[chan].src = src;
  src_addr[chan] = src;
  if (frame_sync[chan]) {
    /* set increment code over no adjustment (0) */
    SET_FIELD(&dma[chan].mcr, INDEXMODE_INC, SIND, SIND_SZ);
    /* clear the CSDP SRC field to indicate SARAM source */
    CLR_FIELD(&dma[chan].mcr, SLAXS, SLAXS_SZ);
    SET_FIELD(&dma[chan].mcr, LAXS_INTERNAL, SLAXS, SLAXS_SZ);
    /* substitute timer for the FPGA synchronization */
    intr_pause(sync_isn[chan]);
    switch (chan) {
    case 0:
      interrupt_init(dma_intr0_fs_start, timer_isn);
      break;
    case 1:
      interrupt_init(dma_intr1_fs_start, timer_isn);
      break;
    case 2:
      interrupt_init(dma_intr2_fs_start, timer_isn);
      break;
    case 3:
      interrupt_init(dma_intr3_fs_start, timer_isn);
      break;
    } /* end switch */
    intr_start(timer_isn);
  } else {
    /* word sync */
    intr_pause(sync_isn[chan]);
```

```
    switch (chan) {
    case 0:
      interrupt_init(dma_intr0_ws, timer_isn);
      break;
    case 1:
      interrupt_init(dma_intr1_ws, timer_isn);
      break;
    case 2:
      interrupt_init(dma_intr2_ws, timer_isn);
      break;
    case 3:
      interrupt_init(dma_intr3_ws, timer_isn);
      break;
    } /* end switch */
    intr_start(timer_isn);
  } /* end if */
} /* end test_DMA_channel */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr0_fs(void) {
  read_err[0] = DMPREC & MASK_BIT(DE0+dma[0].ch);
  if (read_err[0]) return;
  buf_count[0]++;
} /* end dma_intr0_fs */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr1_fs(void) {
  read_err[1] = DMPREC & MASK_BIT(DE0+dma[1].ch);
  if (read_err[1]) return;
  buf_count[1]++;
} /* end dma_intr1_fs */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr2_fs(void) {
  read_err[2] = DMPREC & MASK_BIT(DE0+dma[2].ch);
  if (read_err[2]) return;
  buf_count[2]++;
} /* end dma_intr2_fs */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr3_fs(void) {
  read_err[3] = DMPREC & MASK_BIT(DE0+dma[3].ch);
  if (read_err[3]) return;
  buf_count[3]++;
} /* end dma_intr3_fs */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr0_fs_start(void) {
    dma_init(dma[0].ch,
            dma[0].sfc,
            dma[0].mcr,
            dma[0].cnt,
            dma[0].srcp,
            (unsigned int) src_addr[0],
            dma[0].dstp,
            (unsigned int) dst_addr[0]);
```

```
   DMA_ENABLE(dma[0].ch);
   if (--frame_cnt[0] == 0) {
     frame_cnt[0] = dma[0].frm;
     src_addr[0] = dma[0].src;
     dst_addr[0] = dma[0].dst;
   } else {
     src_addr[0] += trans_cnt[0];
     dst_addr[0] += trans_cnt[0];
   } /* end if end of block */
} /* end dma_intr0_fs_start */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr1_fs_start(void) {
   dma_init(dma[1].ch,
            dma[1].sfc,
            dma[1].mcr,
            dma[1].cnt,
            dma[1].srcp,
            (unsigned int) dma[1].src,
            dma[1].dstp,
            (unsigned int) dma[1].dst );
   DMA_ENABLE(dma[1].ch);
   if (--frame_cnt[1] == 0) {
     frame_cnt[1] = dma[1].frm;
     src_addr[1] = dma[1].src;
     dst_addr[1] = dma[1].dst;
   } else {
     src_addr[1] += trans_cnt[1];
     dst_addr[1] += trans_cnt[1];
   } /* end if end of block */
} /* end dma_intr1_fs_start */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr2_fs_start(void) {
   dma_init(dma[2].ch,
            dma[2].sfc,
            dma[2].mcr,
            dma[2].cnt,
            dma[2].srcp,
            (unsigned int) dma[2].src,
            dma[2].dstp,
            (unsigned int) dma[2].dst );
   DMA_ENABLE(dma[2].ch);
   if (--frame_cnt[2] == 0) {
     frame_cnt[2] = dma[2].frm;
     src_addr[2] = dma[2].src;
     dst_addr[2] = dma[2].dst;
   } else {
     src_addr[2] += trans_cnt[2];
     dst_addr[2] += trans_cnt[2];
   } /* end if end of block */
} /* end dma_intr2_fs_start */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr3_fs_start(void) {
   dma_init(dma[3].ch,
```

```
            dma[3].sfc,
            dma[3].mcr,
            dma[3].cnt,
            dma[3].srcp,
            (unsigned int) dma[3].src,
            dma[3].dstp,
            (unsigned int) dma[3].dst );
    DMA_ENABLE(dma[3].ch);
    if (--frame_cnt[3] == 0) {
      frame_cnt[3] = dma[3].frm;
      src_addr[3] = dma[3].src;
      dst_addr[3] = dma[3].dst;
    } else {
      src_addr[3] += trans_cnt[3];
      dst_addr[3] += trans_cnt[3];
    } /* end if end of block */
} /* end dma_intr3_fs_start */
/*=========================================================================*/
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr0_ws(void) {
  ST1 &= ~(0x4000); /* set off CLP bit */
  *dst_addr[0]++ = *src_addr[0]++;
  if (--trans_cnt[0] == 0) {
    buf_count[0]++;
    trans_cnt[0] = dma[0].cnt;
    if (--frame_cnt[0] == 0) {
      frame_cnt[0] = dma[0].frm;
      dst_addr[0] = dma[0].dst;
      src_addr[0] = dma[0].src;
    } /* end if end block */
  } /* end if end frame */
} /* end dma_intr0_ws */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr1_ws(void) {
  ST1 &= ~(0x4000); /* set off CLP bit */
  *dst_addr[1]++ = *src_addr[1]++;
  if (--trans_cnt[1] == 0) {
    buf_count[1]++;
    trans_cnt[1] = dma[1].cnt;
    if (--frame_cnt[1] == 0) {
      frame_cnt[1] = dma[1].frm;
      dst_addr[1] = dma[1].dst;
      src_addr[1] = dma[1].src;
    } /* end if end block */
  } /* end if end frame */
} /* end dma_intr1_ws */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr2_ws(void) {
  ST1 &= ~(0x4000); /* set off CLP bit */
  *dst_addr[2]++ = *src_addr[2]++;
  if (--trans_cnt[2] == 0) {
    buf_count[2]++;
    trans_cnt[2] = dma[2].cnt;
```

```
    if (--frame_cnt[2] == 0) {
      frame_cnt[2] = dma[2].frm;
      dst_addr[2] = dma[2].dst;
      src_addr[2] = dma[2].src;
    } /* end if end block */
  } /* end if end frame */
} /* end dma_intr2_ws */
/*=========================================================================*/
/* This code is the interrupt routine; it should not be modified. */
extern void interrupt dma_intr3_ws(void) {
  ST1 &= ~(0x4000); /* set off CLP bit */
  *dst_addr[3]++ = *src_addr[3]++;
  if (--trans_cnt[3] == 0) {
    buf_count[3]++;
    trans_cnt[3] = dma[3].cnt;
    if (--frame_cnt[3] == 0) {
      frame_cnt[3] = dma[3].frm;
      dst_addr[3] = dma[3].dst;
      src_addr[3] = dma[3].src;
    } /* end if end block */
  } /* end if end frame */
} /* end dma_intr3_ws */
/*=========================================================================*/
```

F.3 AED_MAIN.c [6]

```
#include <stdlib.h>
#include "AED.h"
#include "AED_DMS.h"
#include "AED_Appl.h"
#ifdef CHIP_5416
  #include "dsk5416.h"
#endif
#if AED_PRINT /* Must be placed after AED.h */
  #include <stdio.h>
#endif
ApplBlockType dual_data_buffer;
static unsigned int  bufs_proc;        //static int  bufs_proc;           //matt
static unsigned int prev_buf_count; //static int prev_buf_count;   //matt
static unsigned int frames, records, reclen, esize, mode;
static unsigned int frame_bytes, blocks;
static Dma_channel ex_dma_chan;
#if (!TALK_TO_FPGA)
  ApplBlockType appl_test_data;
  static unsigned long period;
#endif
#if FPGA_OVFL_CHECK_ENABLE
  #ifndef CHIP_5416
    static unsigned int *FPGA_status_addr;
  #endif
  static unsigned int FPGA_status_mask;
#endif
main()
{
/************************************************************************/
/*                                    Initialize                        */
/************************************************************************/
  board_init();
/************************************************************************/
 /*              Clear static variables, allocate buffers, init status variables          */
 /************************************************************************/
#if FPGA_OVFL_CHECK_ENABLE
  #ifndef CHIP_5416
    FPGA_status_addr = FPGA_stat_addr();
  #endif
  FPGA_status_mask = FPGA_stat_mask();
#endif
#if (!TALK_TO_FPGA)
  appl_test_data.uword = NULL;
#endif
  prev_buf_count = 0;
  bufs_proc = 0;
  appl_parms(&frames, &records, &reclen, &esize, &mode); /*appl_hdr*/
  frame_bytes = records*reclen*(byte_size(esize));
  if (mode&DMA_SYNC_MASK == DMA_FSB_MODE) {
    blocks = 1;
  } else {
    blocks = 1;       //frames;  //matt
  }
```

```
#ifdef CHIP_5510
  dual_data_buffer.byte = (char *)malloc(frames*frame_bytes+1)+1;
#else
  dual_data_buffer.byte = malloc(frames*frame_bytes);
#endif
  if (!dual_data_buffer.byte) {
    error_flashing(AED_FLASH_MAIN_DATA_BUFFER_MALLOC_ERROR);
  } /* end if */
#if (!TALK_TO_FPGA)
  if (!appl_test_data.byte) {
    error_flashing(AED_FLASH_MAIN_TEST_BUFFER_MALLOC_ERROR);
  } /* end if */
#endif
  /*****************************************************************/
  /* Clear the buffers, start DMA, and wait for interrupt          */
  /*****************************************************************/
  if (mode & DMA_NONE_MODE) {
    ex_dma_chan = DMA_CH_NONE;
  } else {
    ex_dma_chan= alloc_DMA_channel();
    program_DMA_channel(ex_dma_chan, mode&DMA_PORT_MASK, /*dma_ex*/
                    dual_data_buffer.byte, FPGA_ADDRESS,
                    records*reclen, frames, esize,
                    mode&DMA_SYNC_MASK);
  } /* end if */
  read_err[ex_dma_chan] = 0;
  buf_count[ex_dma_chan] = 0;
  appl_init(dual_data_buffer, (frames*frame_bytes), ex_dma_chan);
  /* set up dma channel to read from fill in lieu of FPGA */
#if (!TALK_TO_FPGA)
  period = appl_test(appl_test_data, records*reclen, frames);   /*appl_hdr*/
  #if AED_PRINT
    printf ("Period = %lu (0.5 usec)\n", period);
  #endif
  test_DMA_channel (ex_dma_chan, appl_test_data.byte, period); /*dma_ex*/
#endif
  /* check for DMA OFF not set by the user, default is ON */
  if (!(DMA_START_OFF&mode)) {
    start_DMA_channel(ex_dma_chan);                  /*dma_ex*/
#ifdef CHIP_5510
    delay_usec(40);  /* allow DMA to do start up frame before event */
#endif
    /* tell FPGA to start collecting data */
    FPGA_start ();
  }
  {    /*begin Block*/
    unsigned int times = 0;//unsigned int times = 0; //matt
    unsigned int fpga_err=0;
    ApplBlockType address;
    int appl_err = 0;
    while (!(read_err[ex_dma_chan]) &&  !appl_err) {
      /* interrupt always occurs in ISE configurations */
      times++;
     if ((prev_buf_count) != (buf_count[ex_dma_chan])) {
#if FPGA_OVFL_CHECK_ENABLE
       #ifdef CHIP_5416
```

```
        if (((unsigned int)DSK5416_DC_REG) & FPGA_status_mask)
       #else
        /* C6xxx and C55xx */
        if ((* FPGA_status_addr)& FPGA_status_mask)
       #endif
       break;
#endif
       address.byte = &dual_data_buffer.byte
        [(prev_buf_count%blocks)*frame_bytes];
       appl_err = appl_process(address, prev_buf_count);
       prev_buf_count= buf_count[ex_dma_chan];
       bufs_proc++;
    } else {
#ifdef TIMEOUT
       if ((times>>TIMEOUT) > (bufs_proc+1)) {
        FPGA_stop ();
        #if AED_PRINT
          printf ("Timeout\n");
        #endif
        break;
       } /* end if */
#endif
       appl_err = appl_idle();
     } /* end if */
    } /* end while */
   /* get the FIFO overflow code before stopping */
#if FPGA_OVFL_CHECK_ENABLE
   fpga_err =
   #ifdef CHIP_5416
    ((unsigned int)DSK5416_DC_REG) & FPGA_status_mask;
   #else
    /* C6xxx and C55xx */
   (* FPGA_status_addr)& FPGA_status_mask;
   #endif
#else
   fpga_err = 0;
#endif
   /* tell FPGA to stop collecting data */
   FPGA_stop ();
#if AED_PRINT
   printf("Main loop ended.\n");
#else
   delay_msec(5);
#endif
   pause_DMA_channel(ex_dma_chan);          /*dma_ex*/
  /*****************************************************************/
  /*                    Check for processing errors                */
  /*****************************************************************/
   appl_end(times, bufs_proc, buf_count[ex_dma_chan],prev_buf_count,
          read_err[ex_dma_chan], appl_err,
          fpga_err, (count_DMA_channel(ex_dma_chan)));
  } /* end block */
  return(0);
} /* end main */
/*********************************************************************
  error_flashing - Flashes a numeric code to ERR_LED forever
```

```
  Parameters:  IN flashes - numeric code ranging 1 - 99
*************************************************************************/
void error_flashing(int flashes)
{
  int on = 1;
  error_flashing_while(flashes, &on);
} /* end error flashing */
/*************************************************************************
  error_flashing_while - Flashes a numeric code while condition
  Parameters:  IN flashes - numeric code ranging 1 - 99
               IN cond    - pointer to a boolean condition
*************************************************************************/
void error_flashing_while(int flashes, int *cond)
{
  int j;
  #if AED_PRINT
    if (flashes != AED_FLASH_NORMAL_COMPLETION) {
      printf("ERROR %d, lookup in AED.h\r\n", flashes);
    }
  #endif
  /* two digit flash */
  brd_led_disable(ERR_LED);
  do {
    delay_msec(2100);
    for(j=0; j<flashes/10; j++){
      brd_led_enable(ERR_LED);
      delay_msec(100);
      brd_led_disable(ERR_LED);
      delay_msec(300);
    }
    delay_msec(300);
    for(j=0; j<flashes%10; j++){
      brd_led_enable(ERR_LED);
      delay_msec(100);
      brd_led_disable(ERR_LED);
      delay_msec(300);
    }
  } while (*cond);
} /* end error_flashing */
```

F.4 Vectors.asm [6]

```
;
;  ======== vectors.asm ========
;  Plug in the entry point at RESET in the interrupt vector table
;
;
;  ======== unused ========
;  plug inifinite loop -- with nested branches to
;  disable interrupts -- for all undefined vectors
;
      .sect ".vectors"
      .ref _c_int00         ; C entry point
      .align  0x80          ; must be aligned on page boundary
__vectors:
      .def __vectors
RESET:                      ; reset vector
      BD _c_int00           ; branch to C entry point
      STM #200,SP           ; stack size of 200
nmi:   RETE                 ; enable interrupts and return from one
            NOP
            NOP
            NOP             ;NMI~
                            ; software interrupts
sint17 .space 4*16
sint18 .space 4*16
sint19 .space 4*16
sint20 .space 4*16
sint21 .space 4*16
sint22 .space 4*16
sint23 .space 4*16
sint24 .space 4*16
sint25 .space 4*16
sint26 .space 4*16
sint27 .space 4*16
sint28 .space 4*16
sint29 .space 4*16
sint30 .space 4*16
int0:  RETE
            NOP
            NOP
            NOP
int1:  RETE
            NOP
            NOP
            NOP
int2:  RETE
            NOP
            NOP
            NOP
tint:  RETE
            NOP
            NOP
            NOP
rint0: RETE
```

```
            NOP
            NOP
            NOP
xint0:  RETE
            NOP
            NOP
            NOP
xint2:  RETE
            NOP
            NOP
            NOP
rint2:  RETE
            NOP
            NOP
            NOP
int3:   RETE
            NOP
            NOP
            NOP
hint:   RETE
            NOP
            NOP
            NOP
rint1:  RETE
            NOP
            NOP
            NOP
xint1:  RETE
            NOP
            NOP
            NOP
dmac4:   RETE
            NOP
            NOP
            NOP
dmac5:   RETE
            NOP
            NOP
            NOP
; Model Interrupt Vector
      .data
_IntVM: BD _c_int00    ; branch to C entry point
                  PSHM 1eh   ; push the XPC register
                  NOP
                  .def _IntVM
            .end
```

F.5 AED_109_32d.c [6]

```
#include <stdlib.h>
#ifdef CHIP_5416
#include <timr5416.h>
#include "dsk5416.h"
#else
#include <timer.h>
#endif
#if defined(CHIP_6711) || defined(CHIP_6211) || defined(CHIP_6211X)
#include "6x11DSK.h"
#include "regs.h"
#else
#include <emif.h>
#endif
#include "AED.h"
#include "AED_DMS.h"
#include "AED_Appl.h"
#if AED_PRINT /* Must be placed after AED.h */
  #include <stdio.h>
#endif
/*---------------------------------------------------------------------
  In documentation in this module, the term "buffer" is used to refer
  to either a "block" or a "frame".  "Block" is the whole memory area
  allocated to transferring data from the AED's (daughterboard) FPGA
  to the DSP's memory via DMS through the EMIF bus.  The "block" is
  divided into one or more "frames".
  The DMS, Data Movement Service, is implemented in several forms for
  use in various DSPs.  Access to the DMS is through a common header
  module (AED_DMS.h) which serves all implementations.  The most
  elementary implementation is the use of the CPU to tranfer the data
  (AED_DMS_intr.c).  This implementation supports DSP with not data
  movement hardware, but it is slow and not recommended if another
  implementation can be used.  Other implementations are available
  for DMAs and EDMAs on various DSPs.
  In most modes of operation, one frame of data is processed at a
  time.  The DMS is setup to give an interrupt at the completion of
  transfer of each frame.  Upon the receipt of the interrupt, but
  not in the interrupt service routine, the frame last transfered
  is presented to appl_process as the "buffer" for processing.
  However, in block processing modes, interrupts occur only once for
  the entire block, and the entire block is presented as the "buffer"
  for processing in the appl_process function.
  The mode, number of frames in the block, number of records in the
  frame, and the size of the records are selected in appl_parms.
  Records are a application dependent subdivision of the frame.
  Frequently, multiple records, identical in format, are transferred
  in a frame in order to reduce the number of interrupts.
---------------------------------------------------------------------*/
/* application may change these defines */
#define AED_BOARD                       "109 Diff"
#define DMS_MODE                        DMA_FS_MODE
#define RECORD_SKIP_POWER               (6)     /* NO_RECORDS to skip for speed*/
#define DIVIDE_POWER                    (8)     /* NO_RECORDS = 2^DIVIDE_POWER */
#define NO_RECORDS                      (1<<DIVIDE_POWER)       /* records/frame */
```

```
#define NO_FRAMES                       3                   /* frames/block  */
#define RECLEN                          1          /* length of record in words */
#define ELEMENTSIZE_CODE                DMA_ESIZE32
#define SAMPLES_PER_WORD                2
#define SAVE_RECORDS                    32                  /* size of printout */
#define ITERATIONS                      16                  /* size of averages */
#define DAC_CLK_CNT                     79             /* divide 80MHz by 80 */
/* FPGA register address definitions  */
#ifdef CHIP_5416
#define LSB_DIO_REG (0x0000/sizeof(unsigned short))
#define MSB_DIO_REG (0x0001/sizeof(unsigned short))
#define STATUS_REG  (0x0000/sizeof(unsigned short)) /* read only */
#define START_REG   (0x0001/sizeof(unsigned short))
#define PERIOD_REG  (0x0002/sizeof(unsigned short))
#define INTR_CD_REG (0x0003/sizeof(unsigned short)) /* read only */
#define ADC_CLK_REG (0x0004/sizeof(unsigned short))
#define ADC_CPW_REG (0x0008/sizeof(unsigned short))
#define ADC_CR0_REG (0x0009/sizeof(unsigned short))
#define ADC_CR1_REG (0x000a/sizeof(unsigned short))
#define ADC_CD_REG  (0x0005/sizeof(unsigned short)) /* read only */
#define DAC_CLK_REG (0x0006/sizeof(unsigned short))
#define DAC_CD_REG  (0x0007/sizeof(unsigned short)) /* read only */
#else
#define LSB_DIO_REG (0x00000/sizeof(unsigned short))
#define MSB_DIO_REG (0x00004/sizeof(unsigned short))
#define STATUS_REG  (0x20000/sizeof(unsigned short)) /* read only */
#define START_REG   (0x20004/sizeof(unsigned short))
#define PERIOD_REG  (0x20008/sizeof(unsigned short))
#define INTR_CD_REG (0x2000C/sizeof(unsigned short)) /* read only */
#define ADC_CLK_REG (0x20010/sizeof(unsigned short))
#define ADC_CPW_REG (0x20020/sizeof(unsigned short))
#define ADC_CR0_REG (0x20024/sizeof(unsigned short))
#define ADC_CR1_REG (0x20028/sizeof(unsigned short))
#define ADC_CD_REG  (0x20014/sizeof(unsigned short)) /* read only */
#define DAC_CLK_REG (0x20018/sizeof(unsigned short))
#define DAC_CD_REG  (0x2001C/sizeof(unsigned short)) /* read only */
#endif
/* Globals for diagnostics of termination */
unsigned int debug_times;
int debug_bufs_proc;
int debug_buf_count;
int debug_prev_buf_count;
int debug_DMS_err;
int debug_appl_term_code;
unsigned int debug_FIFO_ovfl;
unsigned int debug_DMS_count;
ApplBlockType save_data;
#if(!TALK_TO_FPGA)
  #if (defined(CHIP_5416) || defined(CHIP_5510))
    unsigned long test_data[NO_RECORDS*NO_FRAMES*RECLEN];
  #else
    far unsigned int test_data[NO_RECORDS*NO_FRAMES*RECLEN];
  #endif
#endif
unsigned long fpga_io_reg = 0x70000001;
unsigned short * cntl_base_addr;
```

```
unsigned long * data_base_addr;
Dma_channel fpga_chan;
int input_count;
/*-------------------------------------------*/
/* the following code is application dependent */
static int loop_count;
static int iteration;
unsigned int A_value[ITERATIONS];
unsigned int B_value[ITERATIONS];
ApplBlockType addr[ITERATIONS] =  /* for testing */
   {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0 };
int bufs[ITERATIONS] =  /* for testing */
   {99, 99, 99, 99, 99, 99, 99, 99, 99, 99,
    99, 99, 99, 99, 99, 99 };
/* end of application dependent code */
/*-------------------------------------------*/
/*************************************************************************
  appl_parms - Defines mode and size of block for DMS transfers
    Parameters:  OUT frames  - number of frames in the block
                 OUT records - number of records in the frame
                 OUT reclen  - number of transfer elements in record
                 OUT esize   - transfer element size code (AED_Brd.h)
                 OUT mode    - transfer mode code (AED_DMS.h)
    Notes:
    1) This routine is always called from main prior to allocation
    of the block for data transfer.  This function returns the
    specifics for the allocation and the DMS setup.
    2) The DMS can transfer the data in different widths up to the
    EMIF bus size.  The code selects the size independent of DSP.

    3) The DMS mode codes include both word and frame synchronization.
    Synchronization is the pulses sent on a wire connecting the AED
    with the DSP used by the DMS to determine when to read or write
    to the AED.
    4) The DMS can be implemented with any of the facilities available
    on any particular DSP.  Interrupts are available on all DSPs so
    the AED_DSM_intr support can always be used, although it is slow.
    The other implementations depend on what the DSP has for data
    transfer hardware.
*************************************************************************/
void appl_parms(unsigned int *frames,
                unsigned int *records,
                unsigned int *reclen,
                unsigned int *esize,
                unsigned int *mode)
{
  unsigned long * io_addr = get_cntl_addr();
  /*-------------------------------------------*/
  /* the following code may be application dependent or
     additional code may be required */
  cntl_base_addr = (unsigned short *)
#ifdef CHIP_5416
    ((unsigned short)get_cntl_addr() | 0x8000);
#else
    get_cntl_addr();
```

```
#endif
  data_base_addr = get_data_addr();
/* AED109 Standard EMIF settings
  Parm                  6x01    6x11            5510      5416
  Write setup           4       3
  Write strobe          3       2
  Write hold            3       2
  Read setup            2       2
  Read strobe           13      7
  Read hold             0       0
*/
#if defined(CHIP_6711) || defined(CHIP_6211) || defined(CHIP_6211X)
  REG_WRITE (EMIF_CE2, 0x30A20720); /* CE2 control, 32bit async*/
  #if (TALK_TO_FPGA)
    REG_WRITE (MAR0,   0x1);     /* Enable cache for SRAM      */
    REG_WRITE (L2CFG,  0x80000002);  /* 2-Way cache & EDMA Pri */
  #endif
  fpga_io_reg = 0x71000001;
  #define ADC_MIN_CPW 7
#else
  #ifdef CHIP_5510
    EMIF_CE2_CTRL1 = 0x2105;
    EMIF_CE2_CTRL2 = 0x5105;
    EMIF_CE2_CTRL3 = 0;
    #define ADC_MIN_CPW 7 /* 7 times the EMIF CLK period */
  #else
    #ifdef CHIP_5416
      #define ADC_MIN_CPW 5
    #else
      /* C6x01 EVMs */
      EMIF_CE1_CTRL = 0x40F20D20;
      #define ADC_MIN_CPW 5
    #endif
  #endif
#endif
  puts("Begin Processing\n");
  brd_led_enable(ERR_LED);
#if(!TALK_TO_FPGA)
  #ifdef CHIP_5416
    appl_test_data.uword = (UWordType *)test_data;
  #else
    appl_test_data.uword = test_data;
  #endif
#endif
#ifdef CHIP_5416
  DSK5416_DM_CNTL = 0xc8; /* control page */
  delay_usec(1);
#endif
  *frames = NO_FRAMES;
  *records = NO_RECORDS;
  *reclen = RECLEN;
  *esize = ELEMENTSIZE_CODE;
  *mode = DMS_MODE;
  /* end of application dependent code */
  /*-----------------------------------------------*/
  *(cntl_base_addr+LSB_DIO_REG) = fpga_io_reg; /* LSB */
```

```
  *(cntl_base_addr+MSB_DIO_REG) = fpga_io_reg>>16; /* MSB */
  delay_usec(1);
#ifdef CHIP_5416
  DSK5416_DM_CNTL = 0xc9; /* register page */
  delay_usec(1);
#endif
  *(cntl_base_addr+START_REG) = 256;
  delay_usec(1);
  *(cntl_base_addr+PERIOD_REG) = NO_RECORDS*RECLEN;
            /* number of words per frame */
  delay_usec(1);
  /* ADC and DAC clock rate = EMIF CLOCK / (CLK_REG + 1)*/
  *(cntl_base_addr+ADC_CLK_REG) = 19; /* EMIF_CLK / 20 */
  delay_usec(1);
  *(cntl_base_addr+ADC_CPW_REG) = ADC_MIN_CPW;
  delay_usec(1);
  *(cntl_base_addr+ADC_CR0_REG) = 0x8020; /* 2 differential channels */
  delay_usec(1);
  *(cntl_base_addr+ADC_CR1_REG) = 0x84C0;
  delay_usec(1);
  *(cntl_base_addr+DAC_CLK_REG) = DAC_CLK_CNT;
#ifdef CHIP_5416
  delay_usec(1);
  DSK5416_DM_CNTL = 0xd0; /* data address default */
#endif
#if AED_PRINT
  printf("\n*** AED " AED_BOARD " TEST PROGRAM STARTED ***\n");
#endif
} /* end appl_parms */
/*************************************************************************
  appl_init - Performs buffer init before data transfer
    Parameters:  OUT data_block - pointer to beginning of entire input block (all frames)
                 IN block_bytes - number of bytes in block
                 IN dma_chan - DMA channel allocated by main for input
    Note: This routine is autmatically called from main before the
          data transfer is intiated, to initialize data buffers.
*************************************************************************/
void appl_init(ApplBlockType data_block, unsigned int  block_bytes, Dma_channel   dma_chan)
{
  /* the following code may be application dependent or additional code may be required */
  int i, j, k;
  unsigned short *ptr;
    save_data.byte = malloc(SAVE_RECORDS*RECLEN*byte_size(ELEMENTSIZE_CODE));
  if (!save_data.byte) {
    error_flashing(AED_FLASH_APPL_SAVE_BUFFER_MALLOC_ERROR);
  } /* end if */
  fpga_chan = dma_chan;
  ptr = data_block.uhword;
  for (i=0; i<NO_FRAMES; i++) {
    for (j=0; j<NO_RECORDS; j++) {
      for (k=0; k<RECLEN*SAMPLES_PER_WORD; k++) {
        *(ptr++) = 0;
      }
    }
  }
  ptr = save_data.uhword;
```

```
  for (j=0; j<SAVE_RECORDS; j++) {
    for (k=0; k<RECLEN*SAMPLES_PER_WORD; k++) {
      *(ptr++) = 0;
    }
  }
  for (j=0; j<ITERATIONS; j++) {
    A_value[j] = 0;
    B_value[j] = 0;
  } /* end for */
  loop_count = 0;
  iteration = 0;
  input_count = 0;
#if AED_PRINT
  printf("Begin application processing (Block size = %d bytes)\n",
          block_bytes);
#endif
  brd_led_disable(ERR_LED);
  /* end of application dependent code */
} /* end appl_init */
/*********************************************************************
  appl_process - Processes 1 frame of buffer data
  Returns:     user defined termination code, 0 is no termination
  Parameters:  IN data_buffer - pointer to beginning of frame of data just received from FPGA
               IN buf_number - number of buffer just received
    Note: This routine is autmatically called from main when a full
          buffer of data has been transferred from the daughterboard.
*********************************************************************/
int  appl_process(ApplBlockType data_buffer, int buf_number)
{
  int ret;
  /* the following code is application dependent */
  int j;
  unsigned int sumA = 0, sumB = 0;
  ApplBlockType ptr;
  input_count++;
  brd_led_enable(APPL_LED);
  if (iteration==(ITERATIONS-1)) {
    for (j=0; j<SAVE_RECORDS*RECLEN; j++)  {
        save_data.uword[j] = data_buffer.uword[j];
    } /* end for */
  } /* end if */
  addr[iteration&(ITERATIONS-1)] = data_buffer;
  bufs[iteration&(ITERATIONS-1)] = buf_number;
  /* compute the buffer mean */
  ptr = data_buffer;
  for (j=0; j<(NO_RECORDS>>RECORD_SKIP_POWER); j++) {
    sumA += (unsigned int)*(ptr.uhword++);
    sumB += (unsigned int)*(ptr.uhword++);
  } /* end for */
  A_value[iteration&(ITERATIONS-1)] = sumA>>(DIVIDE_POWER-RECORD_SKIP_POWER);
  B_value[iteration&(ITERATIONS-1)] = sumB>>(DIVIDE_POWER-RECORD_SKIP_POWER);
  ret = ++iteration > (ITERATIONS-1)? 99:0;
  /* end of application dependent code */
  return (ret);
} /* end appl_process */
/*********************************************************************
```

```
  appl_idle - Performs background processing
  Returns:   user defined termination code, 0 is no termination
  Note: This routine is automatically called from main when no other
          processing is required, but may not be called regularly.
*****************************************************************/
int  appl_idle(void)
{
  /* the following code is application dependent */
  loop_count++;
  /* end of application dependent code */
  return 0;
} /* end appl_idle */
/*****************************************************************
  appl_end - Final processing before termination
  Parameters:  IN times          - main program loop cycles executed
               IN bufs_proc      - number of buffers processed by appl_process
               IN buf_count      - number of buffers received
               IN prev_buf_count - last buffer given to appl_process
               IN DMS_err        - error code from DMS
               IN appl_term_code - user termination code from either appl_process or appl_idle
               IN FIFO_ovfl      - indication that the FIFOin the FPGA has overflowed
               IN DMS_count      - number of frames remaining to be received in the block
    Note: This routine is automatically called from main at program termination.
*****************************************************************/
void appl_end(unsigned int times,
          int         bufs_proc,
          int         buf_count,
          int         prev_buf_count,
          int         DMS_err,
          int         appl_term_code,
          unsigned int FIFO_ovfl,
          unsigned int DMS_count)
{
  /* the following code is application dependent */
  unsigned long pattern, save;
  int j;
#if AED_PRINT
  FILE *outfile;
#endif
  debug_times         = times;
  debug_bufs_proc     = bufs_proc;
  debug_buf_count     = buf_count;
  debug_prev_buf_count = prev_buf_count;
  debug_DMS_err       = DMS_err;
  debug_appl_term_code = appl_term_code;
  debug_FIFO_ovfl     = FIFO_ovfl;
  debug_DMS_count     = DMS_count;
#if AED_PRINT
  #if !TALK_TO_FPGA
    printf("FPGA channel simulated\n");
  #endif
  printf("Interrupts received = %d\n", input_count);
  /*********************************************************/
  /*                          Check for processing errors                          */
  /*********************************************************/
  if (DMS_err) {
```

```
   printf("Read drop error %d (bufct=%d, trnct=%d)\n",
        DMS_err, prev_buf_count, (DMS_count));
  }
  if (appl_term_code) {
   printf("Application Termination %d\n", appl_term_code);
  }
  if (FIFO_ovfl) {
   printf("FPGA FIFO Overflowed\n");
  }
  printf("Test Loops = %u Bufs Processed = %d Bufs Received = %d\n",
          times, bufs_proc, buf_count);
  #if (AED_PRINT == 2) /* file I/O */
   printf("Writing outfile.txt\n");
   outfile = fopen("outfile.txt", "w");
  #else
   outfile = stdout;
  #endif
  if (outfile) {
   #if (AED_PRINT == 2) /* file I/O */
    fprintf(outfile,"\n*** AED " AED_BOARD " TEST PROGRAM DATA FILE ***\n");
   #endif
   fprintf(outfile, "Block Addresses:");
   for (j=0; j<ITERATIONS; j++) {
    char *aptr = addr[j].byte;
    if (j%5 == 0) fprintf(outfile, "\n    ");
#if (defined(CHIP_5416))
    fprintf (outfile, "%06x", aptr);
#elif (defined(CHIP_5510))
    fprintf (outfile, "%08lx", aptr);
#else
    /* C6x processors */
    fprintf (outfile, "%08x", aptr);
#endif
    fprintf (outfile, "(%2d)", bufs[j]);
   } /* end for */
   fprintf (outfile, "\n");
   fprintf (outfile,
    "A= %04x %04x %04x %04x %04x %04x %04x %04x %04x %04x\n",
    A_value[0], A_value[1], A_value[2], A_value[3], A_value[4],
    A_value[5], A_value[6], A_value[7], A_value[8], A_value[9]);
   fprintf (outfile,
    "B= %04x %04x %04x %04x %04x %04x %04x %04x %04x %04x\n",
    B_value[0], B_value[1], B_value[2], B_value[3], B_value[4],
    B_value[5], B_value[6], B_value[7], B_value[8], B_value[9]);
   for (j=0; j<SAVE_RECORDS*SAMPLES_PER_WORD*RECLEN; j+=16) {
    fprintf (outfile, "[%3d] %03x %03x %03x %03x %03x %03x %03x %03x %03x %03x %03x %03x %03x
%03x %03x %03x\n",
            j,
            save_data.uhword[j],
            save_data.uhword[j+1],
            save_data.uhword[j+2],
            save_data.uhword[j+3],
            save_data.uhword[j+4],
            save_data.uhword[j+5],
            save_data.uhword[j+6],
            save_data.uhword[j+7],
```

```
            save_data.uhword[j+8],
            save_data.uhword[j+9],
            save_data.uhword[j+10],
            save_data.uhword[j+11],
            save_data.uhword[j+12],
            save_data.uhword[j+13],
            save_data.uhword[j+14],
            save_data.uhword[j+15]);
    }/* end for */
    #if (AED_PRINT == 2) /* file I/O */
      fprintf(outfile,"*** Data file complete ***\n");
      fclose(outfile);
      printf("Data file write complete\n");
    #endif
  } else {
    printf("Failure to open outfile.txt\n");
  } /* end if */
#endif
  #if AED_PRINT
    printf("Checking Digital Outputs ....\n");
  #endif
#ifdef CHIP_5416
  DSK5416_DM_CNTL = 0xc8; /* control page */
  DSK5416_MISC = 0x00; /* 16 bit data */
  delay_usec(1);
#endif
#define DIG_O_MASK  0xaa
  pattern = 1;
  for (j=0; j<15; j++) {
    fpga_io_reg &= ~pattern;
    pattern <<= 1;
    fpga_io_reg |= pattern;
    *(cntl_base_addr+LSB_DIO_REG) = fpga_io_reg; /* LSB */
    delay_msec(1);
    save = (*(cntl_base_addr+LSB_DIO_REG))<<1;
    if ((save&DIG_O_MASK) != (pattern&DIG_O_MASK)) {
      if (j)
        printf("Digital I/O Mismatch In= %01lx Out= %01lx\n",
          save, pattern);
      else {
        printf("Digital inputs not all connected. In= %01lx\n",
          save);
 /*       break; */
      } /* end else */
    } /* end if */
    delay_msec(250);
  } /* end for */
#ifdef CHIP_5416
  DSK5416_DM_CNTL = 0xd0; /* data address default */
  delay_usec(1);
#endif
  #if AED_PRINT
     printf("OK\n");
  #endif
  if (DMS_err) {
    error_flashing(AED_FLASH_MAIN_DMS_ERROR);
```

```
  } else if (FIFO_ovfl) {
    error_flashing(AED_FLASH_MAIN_FPGA_OVERFLOW);
  } else {
    unsigned int DAC_value = 0;
    printf("Running test wave on DAC\n");
    for (j=0; j<511; j++) {
      /* fill whole FIFO */
      write_32b_reg(data_base_addr,
        ((DAC_value)&0x3FFF) | (((unsigned long)~(DAC_value++))<<16));
    }
#ifdef CHIP_5416
    TIMER_INIT(0,0,0xffff);
    TIMER_START(0);
    FPGA_start();
    for (;;) {
      while (~TIMER_READ(0) <
            (DAC_value-256)*(DAC_CLK_CNT+1)*2);
      /* DSP timer counts at CPU, FPGA counts at CPU/2
      because the EMIF runs at CPU/2 */
#else
    TIMER_INIT(1,0,0xffffffff,0);
    TIMER_START(1);
    FPGA_start();
    for (;;) {
      while (TIMER_READ(1) < DAC_value-256);
#endif
      j = 256;
      while (j--) {
        /* fill half of the FIFO */
        write_32b_reg(data_base_addr,
          ((DAC_value)&0x3FFF) | (((unsigned long)~(DAC_value++))<<16));
      } /* end for */
    } /* forever */
  }
  /* end of application dependent code */
} /* end appl_end */
/*************************************************************************
  appl_test - Fills separate test block to simulate data from FPGA
  Parameters:  OUT fill - pointer to beginning of test block
               IN frame_bytes - number of bytes in each frame
               IN frames - number of frames in block
    Note: This routine is autmatically called from main to initialize the test buffer with data.
*************************************************************************/
#if(!TALK_TO_FPGA)
#define SAMPLE_RATE_KHZ  10000L   /* Sample rate for timer setup */
unsigned long  appl_test(ApplBlockType fill, int frame_elements, int frames)
{
  /* the following code is application dependent */
  int i, j;
  int samples_per_element = (byte_size(ELEMENTSIZE_CODE)/sizeof(short));
  int frame_samples = frame_elements*samples_per_element;
  /* period is in 0.5 microseconds units */
  /* for a time between frames (frame sync)
     period = (frame_elements/SAMPLE_RATE)/(0.5e-6 us) */
  unsigned long period = (2000L*(long)frame_elements)/SAMPLE_RATE_KHZ;
  /* fill can be changed to accomodate testing */
```

```
  for (i=0; i<frames; i++) {
    for (j=0; j<frame_samples; j++)  {
       fill.uhword[i*frame_samples+j]
         = ((i+1)*0x1000)+(j+1);
    } /* end for */
  } /* end for */
  /* end of application dependent code */
  return period;
} /* end appl_test */
#endif
```

F.6 NONLinearInterval_delay_quicker.c

```
#include <stdlib.h>
#include "dsk5416.h"
#include <emif.h>
#include "AED.h"
#include "AED_DMS.h"
#include "AED_Appl.h"
#if AED_PRINT
#include <stdio.h>  /* Must be placed after AED.h */
#endif
#include <math.h>
#define AED_BOARD        "109 Diff"
#define DMS_MODE         DMA_FS_MODE
#define DIVIDE_POWER     (8)      /* NO_RECORDS = 2^DIVIDE_POWER */
#define NO_RECORDS       (1<<DIVIDE_POWER)       /* records/frame */
#define NO_FRAMES        3                   /* frames/block  */
#define RECLEN           1          /* length of record in words */
#define ELEMENTSIZE_CODE  DMA_ESIZE32
#define SAMPLES_PER_WORD   2
#define DAC_CLK_CNT      40959             /* divide 80MHz by 40960 */
/* FPGA register address definitions  */
#define LSB_DIO_REG (0x0000/sizeof(unsigned short))
#define MSB_DIO_REG (0x0001/sizeof(unsigned short))
#define STATUS_REG  (0x0000/sizeof(unsigned short)) /* read only */
#define START_REG   (0x0001/sizeof(unsigned short))
#define PERIOD_REG  (0x0002/sizeof(unsigned short))
#define INTR_CD_REG (0x0003/sizeof(unsigned short)) /* read only */
#define ADC_CLK_REG (0x0004/sizeof(unsigned short))
#define ADC_CPW_REG (0x0008/sizeof(unsigned short))
#define ADC_CR0_REG (0x0009/sizeof(unsigned short))
#define ADC_CR1_REG (0x000a/sizeof(unsigned short))
#define ADC_CD_REG  (0x0005/sizeof(unsigned short)) /* read only */
#define DAC_CLK_REG (0x0006/sizeof(unsigned short))
#define DAC_CD_REG  (0x0007/sizeof(unsigned short)) /* read only */
/* Globals for diagnostics of termination */
unsigned int debug_times;//unsigned int debug_times; //matt
unsigned int debug_bufs_proc;//int debug_bufs_proc;  //matt
unsigned int debug_buf_count;//int debug_buf_count; //matt
unsigned int debug_prev_buf_count;//int debug_prev_buf_count;  //matt
int debug_DMS_err;
int debug_appl_term_code;
unsigned int debug_FIFO_ovfl;
unsigned int debug_DMS_count;
#if(!TALK_TO_FPGA)
   unsigned long test_data[NO_RECORDS*NO_FRAMES*RECLEN];
#endif
unsigned long fpga_io_reg = 0x70000001;
unsigned short * cntl_base_addr;
unsigned long * data_base_addr;
Dma_channel fpga_chan;
int input_count;
static int loop_count;
static unsigned long A_value;
static unsigned long output;
```

```
static float outputsave[310];
static float u3save[310];
static float y3save[310];
static float KeyHolesave[310];
static float Msec[310];
static float delaysave[310];
static float A1[2];
static float A2[2];
static float A3[2];
static float Aone;
static float Atwo;
static float Athree;
static float y[7];
static float ymeasured;
static float u[5];
static float du[7];
static float y0;
static float KeyHolePotential;
static float BaseTime;
static float x1;
static float x0;
static float x2;
static float Time;
static float BaseCurrent;
static float ylargest;
static float counter;
static float meltdown;
static float TimeMeltdown;
static float StartCurrent;
static float MaxCurrent;
static float delay;
static int count;
static float skip;
static int start;
static int mask;
static float openloopthree;
static int maskdelay;
void appl_parms(unsigned int *frames,
                unsigned int *records,
                unsigned int *reclen,
                unsigned int *esize,
                unsigned int *mode)
 {
 unsigned long * io_addr = get_cntl_addr();
 cntl_base_addr = (unsigned short *)
 ((unsigned short)get_cntl_addr() | 0x8000);
 data_base_addr = get_data_addr();
 #define ADC_MIN_CPW 5 /* 5 times the EMIF CLK period */
 puts("Begin Processing\n");
 brd_led_enable(ERR_LED);
 #if(!TALK_TO_FPGA)
   appl_test_data.uword = (UWordType *)test_data;
 #endif
 DSK5416_DM_CNTL = 0xc8; /* control page */
 delay_usec(1);
 *frames = NO_FRAMES;
```

```
 *records = NO_RECORDS;
 *reclen = RECLEN;
 *esize = ELEMENTSIZE_CODE;
 *mode = DMS_MODE;
 *(cntl_base_addr+LSB_DIO_REG) = fpga_io_reg; /* LSB */
 *(cntl_base_addr+MSB_DIO_REG) = fpga_io_reg>>16; /* MSB */
 delay_usec(1);
 DSK5416_DM_CNTL = 0xc9; /* register page */
 delay_usec(1);
 *(cntl_base_addr+START_REG) = 256;
 delay_usec(1);
 *(cntl_base_addr+PERIOD_REG) = NO_RECORDS*RECLEN;
          /* number of words per frame */
 delay_usec(1);
 /* ADC and DAC clock rate = EMIF CLOCK / (CLK_REG + 1)*/
 *(cntl_base_addr+ADC_CLK_REG) = 159; /* EMIF_CLK / 160 */
 delay_usec(1);
 *(cntl_base_addr+ADC_CPW_REG) = ADC_MIN_CPW;
 delay_usec(1);
 *(cntl_base_addr+ADC_CR0_REG) = 0x8020; /* 2 differential channels */
 delay_usec(1);
 *(cntl_base_addr+ADC_CR1_REG) = 0x84C0;
 delay_usec(1);
 *(cntl_base_addr+DAC_CLK_REG) = DAC_CLK_CNT;
 delay_usec(1);
 DSK5416_DM_CNTL = 0xd0; /* data address default */
 #if AED_PRINT
   printf("\n*** AED " AED_BOARD " TEST PROGRAM STARTED ***\n");
 #endif
 }
void appl_init(ApplBlockType data_block, unsigned int  block_bytes, Dma_channel   dma_chan)
 {
 int e,g,w,x,y,z;
 unsigned short *ptr;
 //2.4 mm nozzle, .362 Volts for Servo equates to travel speed of 2.534 mm/sec,
 //max 135 Amp, min 30 Amp, Argon Plasma Jet pressure of 4 CFH bottom of ball,
 //start 135 Amp, 5 mm nozzle height, 3 mm Work Piece,
 //304 stainless steel, Argon Shield Jet pressure of 35 CFH middle of ball,
 //Argon Backing Jet pressure of 35 CFH middle of ball
 for (g=0;g<7;g++)
   {
   du[g]=0;
   }
 for (e=0;e<5;e++)
   {
   u[e]=135;//135 Amps initial output
   }
 ymeasured=0;
 A1[0]=-6.94168976627553;
 A1[1]=-10.1609654622258;
 A2[0]=-0.000418976974092593;
 A2[1]=-0.00424252156547636;
 A3[0]=0.000588460632832352;
 A3[1]=-0.00211824837968971;
 y0=325;//325 ms
 BaseTime=400;//400 ms
```

```
 BaseCurrent=30;//30 Amp
 StartCurrent=135;//135 Amp
 MaxCurrent=135;//135 Amp
 mask=0;
 maskdelay=0;
 counter=0;
 count=0;
 skip=0;
 start=0;
 openloopthree=0;
 meltdown=0;
 TimeMeltdown=0;
 A_value = 0;
 KeyHolePotential = 0;
 for (z=0;z<310;z++)
   {
   outputsave[z]=0;
   u3save[z]=0;
   y3save[z]=0;
   delaysave[z]=0;
   KeyHolesave[z]=0;
   }
 fpga_chan = dma_chan;
 ptr = data_block.uhword;
 for (w=0; w<NO_FRAMES; w++)
   {
   for (x=0; x<NO_RECORDS; x++)
     {
     for (y=0; y<RECLEN*SAMPLES_PER_WORD; y++)
       {
       *(ptr++) = 0;
       }
     }
   }
 write_32b_reg(data_base_addr,7991);
 loop_count = 0;
 input_count = 0;
 #if AED_PRINT
   printf("Begin application processing (Block size = %d bytes)\n",block_bytes);
 #endif
 brd_led_disable(ERR_LED);
 }/* end appl_init */
int  appl_process(ApplBlockType data_buffer,int buf_number)
 {
 int ret;
 int i,j,g,n,w,m,t,r;
 unsigned long sumA = 0;
 ApplBlockType ptr;
 brd_led_enable(APPL_LED);
 /* compute the buffer mean */
 ptr = data_buffer;
 //NO_RECORDS=ADsamplingRate/DAsamplingRate=500k/1.953125k=256
 //Requires that the same signal be provided to both input
 //ports.  For bias correction, remove 0.654386 from KeyHolePotential
 //Equation.  By multiplying NO_RECORDS by four, we have assumed
 //a fictitious 14 bit Resolution for the ADC.  The actual
```

```
//quantization for the ADC is 12 bit.  This fictitious 14 bit
//assumption does not produce an error.  The KeyHole quantization
//(i.e. DAC) is actually 12 bit also, but the write32b function
//requires that it be handled in 14 bit fictitious manner.
//The Control Signal 14 Bit quantization will have to change
//by four intervals to produce an interval change at 12 Bit.
//No errors are induced by this data handling method.  True
//12 Bit realization can be utilized for the ADC, but not for
//the DAC.  If 12 bit quantization is desired for the ADC,
//you should not multiply NO_RECORDS by four and adjust the
//input ratio appropriately; This data handling methodogy is
//not ideal, but is dictated by Signalware's firmware which
//is loaded on the FPGA chip.
for (j=0; j<(NO_RECORDS*4); j++)
  {
  sumA += (unsigned long)*(ptr.uhword++);
  }/* end for */
A_value = sumA >> DIVIDE_POWER;
KeyHolePotential = (((A_value*.000124177)-1)*10)-.654386;
counter=counter+1;
if (((( mask == 0 ) && (KeyHolePotential >= .5)) && (start == 1)) && (maskdelay == 0))
  {
  x1=counter*.512;//Note: The DA rate is .512 ms/cycle or 1.953125 kHz
  output=7520;//Corresponds to BaseCurrent of 30 Amps
  write_32b_reg(data_base_addr,output);
  Time=x1;
  maskdelay=1;
  //Don't need a skip because of mask
  outputsave[count]=BaseCurrent;
  y3save[count]=ymeasured;
  u3save[count]=u3save[count-1];
  KeyHolesave[count]=KeyHolePotential;
  delaysave[count]=delaysave[count-1];
  Msec[count]=Time;
  count=count+1;
  }
else if ((((mask == 0) && (KeyHolePotential < .5 )) && (start == 1)) && (maskdelay == 1))
  {
  x2=counter*.512;
  write_32b_reg(data_base_addr,output);
  Time=x2;
  mask=1;
  maskdelay=0;
  ylargest=0;
  openloopthree=openloopthree+1;
  ymeasured=x2-x0;
  delay=x2-x1;
  y[3]=ymeasured;
  x0=x2+BaseTime;
  if (openloopthree > 0)
    {
    du[3]=0;
    for (w=0;w<2;w++)//A1
      {
      for (n=0;n<2;n++)//A3
        {
```

```
        for (g=0;g<2;g++)//A2
          {
          for (i=3;i<6;i++)//y
            {
           y[i+1]=y[i]+A1[w]*du[i]+((-A2[g])*(((-u[i-1])*y[i])-((-u[i-2])*y[i-1])))+((-A3[n])*(((-u[i-2])*y[i-1])-((-u[i-
3])*y[i-2])));
            }//i
            if (y[6] > ylargest)
              {
              ylargest=y[6];
              Aone=A1[w];
              Atwo=A2[g];
              Athree=A3[n];
                    }
          }//g
        }//n
      }//w
    if (ylargest>y0) //y[3]
      {
      do{
        du[3]=du[3]+1;
        u[3]=u[3]+1;
        u[4]=u[3];
        for (t=3;t<6;t++)
          {
          y[t+1]=y[t]+Aone*du[t]+((-Atwo)*(((-u[t-1])*y[t])-((-u[t-2])*y[t-1])))+((-Athree)*(((-u[t-2])*y[t-1])-((-u[t-
3])*y[t-2])));
          }
        }while(y[6]>y0);
        du[3]=du[3]-1;
        u[3]=u[3]-1;
        u[4]=u[3];
      }//>y0
    else if (ylargest<y0)
      {
      do{
        du[3]=du[3]-1;
        u[3]=u[3]-1;
        u[4]=u[3];
        for (r=3;r<6;r++)
          {
          y[r+1]=y[r]+Aone*du[r]+((-Atwo)*(((-u[r-1])*y[r])-((-u[r-2])*y[r-1])))+((-Athree)*(((-u[r-2])*y[r-1])-((-u[r-
3])*y[r-2])));
          }
        }while(y[6]<y0);
      du[3]=du[3]+1;
      u[3]=u[3]+1;
      u[4]=u[3];
      }//<y0
    else
      {
      du[3]=du[3];
      u[3]=u[3];
      u[4]=u[3];
      }//=y0
    }//end openloop
```

```
  for (m=0;m<3;m++)//save y measurements and u/du predictions backwards
    {
    y[m]=y[m+1];
    u[m]=u[m+1];
    }
  if (u[3] > MaxCurrent)
    {
    u[2]=MaxCurrent;
    u[3]=MaxCurrent;
    u[4]=MaxCurrent;
    }
  if (u[3] < BaseCurrent)
    {
    u[2]=BaseCurrent;
    u[3]=BaseCurrent;
    u[4]=BaseCurrent;
    }
  //Don't need a skip because of mask
  outputsave[count]=BaseCurrent;
  y3save[count]=ymeasured;
  if (openloopthree > 0)
    {
    u3save[count]=u[3];
    }
  KeyHolesave[count]=KeyHolePotential;
  delaysave[count]=delay;
  Msec[count]=Time;
  count=count+1;
  }//end < keyhole loop
 else if (start == 0) //Only saves at beggining of program.
  {
  x0=counter*.512;//Note: The DA rate is .512 ms/cycle or 1.953125 kHz
  Time=x0;//Note: The DA rate is .512 ms/cycle or 1.953125 kHz
  output=5869;//135//6419;//100//6104;//120//6262;//110 Amps Start Current
  write_32b_reg(data_base_addr,output);
  start=1;
  outputsave[count]=StartCurrent;
  y3save[count]=0;
  u3save[count]=StartCurrent;
  KeyHolesave[count]=KeyHolePotential;
  delaysave[count]=0;
  Msec[count]=Time;
  count=count+1;
  }
 else if (((mask == 1) && (Time > x0)) && (start ==1))
  {
  output=(((u[3]*(-.00196679597881))+1)*7991);//calibrated at 120 Amp
                                  //within .5 Amps between 30 - 135 Amps
                                  //slightly non-linear due to Amplifier
                                  //7991 is calculated by setting u[3] to
                                  //zero and measuring output voltage, then
                                  //by knowing the interval at 120 Amp by
                                  //experimentation the ratio may be calculated
  write_32b_reg(data_base_addr,output);
  mask=0;
  outputsave[count]=u[3];//next k output
```

```
  y3save[count]=ymeasured;//last measured output
  if (openloopthree > 0)
    {
    u3save[count]=u[3];//next k output
    }
  else
    {
    u3save[count]=StartCurrent;//Initial Output
    }
  KeyHolesave[count]=KeyHolePotential;
  delaysave[count]=delaysave[count-1];
  Msec[count]=Time;
  count=count+1;
  }
Time=counter*.512;//Note: The DA rate is .512 ms/cycle or 1.953125 kHz
if (((u[3] == MaxCurrent) && (openloopthree > 0)) && (maskdelay == 1))
  {
  meltdown=meltdown+1;
  TimeMeltdown=meltdown*.512;//Note: The DA rate is .512 ms/cycle or 1.953125 kHz
  if (TimeMeltdown == 2000)  //Shutsdown system if output at maximum
                             //level for 2 seconds.
    {
    write_32b_reg(data_base_addr,7991);
    outputsave[count]=0;
    y3save[count]=ymeasured;
    u3save[count]=u[3];
    KeyHolesave[count]=KeyHolePotential;
    delaysave[count]=delaysave[count-1];
    Msec[count]=Time;
    count=count+1;
    }
  }
else
  {
  meltdown=0;
  TimeMeltdown=0;
  }
ret = 0;
skip=skip+1;
if ((skip == 2000) && (maskdelay == 0)) //This iteration only useful if
                                        //KeyHolePotential observation every 1024 ms
                                        //is wanted. If not wanted, can remove
                                        //skip iteration algorythm.
  {
  skip=0;
  if (count > 0)
    {
    outputsave[count]=outputsave[count-1];
    y3save[count]=ymeasured;
    if (openloopthree > 0)
      {
      u3save[count]=u[3];//next k output
      }
    else
      {
      u3save[count]=StartCurrent;
```

```
      }
     KeyHolesave[count]=KeyHolePotential;
     delaysave[count]=delaysave[count-1];
     Msec[count]=Time;
     count=count+1;
     }
    }
  if (count > 308)  //Sets Output to Zero for Next Program Run
    {
    write_32b_reg(data_base_addr,7991);
     outputsave[count-1]=0;
    }
  if (count > 309)  //Returns 1 to terminate process function
    {
    ret=1;
     outputsave[count-1]=0;
    }
  return (ret);
} /* end appl_process */
int  appl_idle(void)
  {
  loop_count++;
  return 0;
  } /* end appl_idle */
void appl_end(unsigned int times,
              unsigned int bufs_proc,
              unsigned int buf_count,
              unsigned int prev_buf_count,
              int          DMS_err,
              int          appl_term_code,
              unsigned int FIFO_ovfl,
              unsigned int DMS_count)
  {
  int p;
  #if AED_PRINT
    FILE *outfile;
  #endif
  debug_times          = times;
  debug_bufs_proc      = bufs_proc;
  debug_buf_count      = buf_count;
  debug_prev_buf_count = prev_buf_count;
  debug_DMS_err        = DMS_err;
  debug_appl_term_code = appl_term_code;
  debug_FIFO_ovfl      = FIFO_ovfl;
  debug_DMS_count      = DMS_count;
  #if AED_PRINT
    #if !TALK_TO_FPGA
      printf("FPGA channel simulated\n");
    #endif
    printf("Interrupts received = %d\n", input_count);
    if (DMS_err)
      {
      printf("Read drop error %d (bufct=%d, trnct=%d)\n",
           DMS_err, prev_buf_count, (DMS_count));
      }
    if (appl_term_code)
```

```
    {
    printf("Application Termination %d\n", appl_term_code);
    }
  if (FIFO_ovfl)
    {
    printf("FPGA FIFO Overflowed\n");
    }
  printf("Test Loops = %u Bufs Processed = %u Bufs Received = %u\n",
         times, bufs_proc, buf_count);
  // Should be set in AED.h
  //     2 = File
  //     1 = Print  (Default)
  //     0 = No print
  #if (AED_PRINT == 2) /* file I/O */
    printf("Writing outfile.txt\n");
    outfile = fopen("outfile.txt", "w");
  #else
    outfile = stdout;
  #endif
  if (outfile)
    {
    #if (AED_PRINT == 2) /* file I/O */
      fprintf(outfile,"\n*** AED " AED_BOARD " Non-Linear Control Algorythm System Data ***\n");
    #endif
    fprintf (outfile,"  control      u[k]      y[k]      KeyHole      Delay      Time\n");
    y3save[0]=0;
    KeyHolesave[0]=0;
    Msec[0]=0;
    for (p=1;p<310;p++)
      {
      fprintf      (outfile,"%016f      %016f      %016f      %016f      %016f
%016f\n",outputsave[p],u3save[p],y3save[p],KeyHolesave[p],delaysave[p],Msec[p]);
      }
    #if (AED_PRINT == 2) /* file I/O */
      fprintf(outfile,"*** Data file complete ***\n");
      fclose(outfile);
      printf("Data file write complete\n");
    #endif
    }
  else
    {
    printf("Failure to open outfile.txt\n");
    }/* end if */
 #endif
 if (DMS_err)
   {
   error_flashing(AED_FLASH_MAIN_DMS_ERROR);
   }
 else if (FIFO_ovfl)
   {
   error_flashing(AED_FLASH_MAIN_FPGA_OVERFLOW);
   }
} /* end appl_end */
```

F.7 NONLinearIntervalParameterEstimate.c

```
#include <stdlib.h>
#include "dsk5416.h"
#include <emif.h>
#include "AED.h"
#include "AED_DMS.h"
#include "AED_Appl.h"
#if AED_PRINT
#include <stdio.h>  /* Must be placed after AED.h */
#endif
#include <math.h>
#define AED_BOARD               "109 Diff"
#define DMS_MODE                DMA_FS_MODE
#define DIVIDE_POWER            (8)      /* NO_RECORDS = 2^DIVIDE_POWER */
#define NO_RECORDS              (1<<DIVIDE_POWER)      /* records/frame */
#define NO_FRAMES               3         /* frames/block  */
#define RECLEN                  1         /* length of record in words */
#define ELEMENTSIZE_CODE    DMA_ESIZE32
#define SAMPLES_PER_WORD    2
#define DAC_CLK_CNT             40959           /* divide 80MHz by 40960 */
/* FPGA register address definitions  */
#define LSB_DIO_REG (0x0000/sizeof(unsigned short))
#define MSB_DIO_REG (0x0001/sizeof(unsigned short))
#define STATUS_REG  (0x0000/sizeof(unsigned short)) /* read only */
#define START_REG   (0x0001/sizeof(unsigned short))
#define PERIOD_REG  (0x0002/sizeof(unsigned short))
#define INTR_CD_REG (0x0003/sizeof(unsigned short)) /* read only */
#define ADC_CLK_REG (0x0004/sizeof(unsigned short))
#define ADC_CPW_REG (0x0008/sizeof(unsigned short))
#define ADC_CR0_REG (0x0009/sizeof(unsigned short))
#define ADC_CR1_REG (0x000a/sizeof(unsigned short))
#define ADC_CD_REG  (0x0005/sizeof(unsigned short)) /* read only */
#define DAC_CLK_REG (0x0006/sizeof(unsigned short))
#define DAC_CD_REG  (0x0007/sizeof(unsigned short)) /* read only */
/* Globals for diagnostics of termination */
unsigned int debug_times;
unsigned int debug_bufs_proc;
unsigned int debug_buf_count;
unsigned int debug_prev_buf_count;
int debug_DMS_err;
int debug_appl_term_code;
unsigned int debug_FIFO_ovfl;
unsigned int debug_DMS_count;
#if(!TALK_TO_FPGA)
   unsigned long test_data[NO_RECORDS*NO_FRAMES*RECLEN];
#endif
unsigned long fpga_io_reg = 0x70000001;
unsigned short * cntl_base_addr;
unsigned long * data_base_addr;
Dma_channel fpga_chan;
int input_count;
static int loop_count;
static unsigned long A_value;
static unsigned long output;
```

```
static float outputsave[125];
static float y3save[125];
static float KeyHolesave[125];
static float Msec[125];
static float ymeasured;
static float KeyHolePotential;
static float x1;
static float x0;
static float Time;
static float counter;
static float random_currentinput[125];
static float random_input[125];
static int count;
static int skip;
static int start;
static int mask;
void appl_parms(unsigned int *frames,
                unsigned int *records,
                unsigned int *reclen,
                unsigned int *esize,
                unsigned int *mode)
 {
 unsigned long * io_addr = get_cntl_addr();
 cntl_base_addr = (unsigned short *)
 ((unsigned short)get_cntl_addr() | 0x8000);
 data_base_addr = get_data_addr();
 #define ADC_MIN_CPW 5 /* 5 times the EMIF CLK period */
 puts("Begin Processing\n");
 brd_led_enable(ERR_LED);
 #if(!TALK_TO_FPGA)
   appl_test_data.uword = (UWordType *)test_data;
 #endif
 DSK5416_DM_CNTL = 0xc8; /* control page */
 delay_usec(1);
 *frames = NO_FRAMES;
 *records = NO_RECORDS;
 *reclen = RECLEN;
 *esize = ELEMENTSIZE_CODE;
 *mode = DMS_MODE;
 *(cntl_base_addr+LSB_DIO_REG) = fpga_io_reg; /* LSB */
 *(cntl_base_addr+MSB_DIO_REG) = fpga_io_reg>>16; /* MSB */
 delay_usec(1);
 DSK5416_DM_CNTL = 0xc9; /* register page */
 delay_usec(1);
 *(cntl_base_addr+START_REG) = 256;
 delay_usec(1);
 *(cntl_base_addr+PERIOD_REG) = NO_RECORDS*RECLEN;
            /* number of words per frame */
 delay_usec(1);
 /* ADC and DAC clock rate = EMIF CLOCK / (CLK_REG + 1)*/
 *(cntl_base_addr+ADC_CLK_REG) = 159; /* EMIF_CLK / 160 */
 delay_usec(1);
 *(cntl_base_addr+ADC_CPW_REG) = ADC_MIN_CPW;
 delay_usec(1);
 *(cntl_base_addr+ADC_CR0_REG) = 0x8020; /* 2 differential channels */
 delay_usec(1);
```

```
  *(cntl_base_addr+ADC_CR1_REG) = 0x84C0;
  delay_usec(1);
  *(cntl_base_addr+DAC_CLK_REG) = DAC_CLK_CNT;
  delay_usec(1);
  DSK5416_DM_CNTL = 0xd0; /* data address default */
  #if AED_PRINT
    printf("\n*** AED " AED_BOARD " TEST PROGRAM STARTED ***\n");
  #endif
  }
void appl_init(ApplBlockType data_block,
               unsigned int  block_bytes,
               Dma_channel   dma_chan)
  {
  int w,x,y,z,s,t,p;
  FILE *infile;
  unsigned short *ptr;
  //2.4 mm nozzle, .362 Volts for Servo equates to travel speed of 2.534 mm/sec,
  //max 115 Amp, min 85 Amp, Argon Plasma Jet pressure of of 4 CFH bottom of ball,
  //5 mm nozzle height, 3 mm Work Piece,
  //304 stainless steel, Argon Shield Jet pressure of 35 CFH middle of ball,
  //Argon Backing Jet pressure of 35 CFH middle of ball
  infile=fopen("infile.txt","r");
  for (s=0;s<125;s++)
    {
    fscanf(infile,"%f\n",&random_currentinput[s]);
    }
  fclose(infile);
  for (t=0;t<125;t++) //Limit input to maximum value
    {
    if (random_currentinput[t] > 135)
               {
               random_currentinput[t]=135;
               }
    }
//  printf("%f\n",random_currentinput[0]);
  fclose(infile);
  for (p=0;p<125;p++)
    {
    random_input[p]=random_currentinput[p];
    }
//  printf("%f\n",random_input[0]);
  ymeasured=0;
  mask=0;
  counter=0;
  count=0;
  skip=0;
  start=0;
  A_value = 0;
  KeyHolePotential = 0;
  for (z=0;z<200;z++)
    {
    outputsave[z]=0;
    y3save[z]=0;
    KeyHolesave[z]=0;
    Msec[z]=0;
    }
```

```
 fpga_chan = dma_chan;
 ptr = data_block.uhword;
 for (w=0; w<NO_FRAMES; w++)
   {
   for (x=0; x<NO_RECORDS; x++)
     {
     for (y=0; y<RECLEN*SAMPLES_PER_WORD; y++)
       {
       *(ptr++) = 0;
       }
     }
   }
 write_32b_reg(data_base_addr,7991);
 loop_count = 0;
 input_count = 0;
 #if AED_PRINT
   printf("Begin application processing (Block size = %d bytes)\n",block_bytes);
 #endif
 brd_led_disable(ERR_LED);
 }/* end appl_init */
int  appl_process(ApplBlockType data_buffer,int buf_number)
 {
 int ret;
 int j;
 unsigned long sumA = 0;
 ApplBlockType ptr;
 brd_led_enable(APPL_LED);
 /* compute the buffer mean */
 ptr = data_buffer;
 for (j=0; j<1024; j++)
   {
   sumA += (unsigned long)*(ptr.uhword++);
   }/* end for */
 A_value = sumA >> DIVIDE_POWER;
 KeyHolePotential = (((A_value*.000124177)-1)*10)-.654386;
 counter=counter+1;
 if (count >= 116)
   {
    count=count+1;
   }
 if ( mask == 0 )
   {
   if ((KeyHolePotential >= .5) && (start == 1))
     {
     x0=counter*.512;//Note: The DA rate is .512 ms/cycle or 1.953125 kHz
     output=7520;//Corresponds to BaseCurrent of 30 Amps
     write_32b_reg(data_base_addr,output);
     Time=x0;
     mask=1;
     ymeasured=x0-x1;
     x1=x0+200;//BaseTime;
       //Don't need a skip because of mask
         outputsave[count]=random_input[count];
         y3save[count]=ymeasured;
         KeyHolesave[count]=KeyHolePotential;
         Msec[count]=Time;
```

```
        count=count+1;
     }//end keyhole loop
    }
  if (start == 0)  //Only saves at beggining of program.
    {
    x1=counter*.512;//Note: The DA rate is .512 ms/cycle or 1.953125 kHz
    Time=x1;//Note: The DA rate is .512 ms/cycle or 1.953125 kHz
    output=(((random_input[count]*(-.00196679597881))+1)*7991);//calibrated at 120 Amp
                                          //within .5 Amps between 30 - 135 Amps
                                          //slightly non-linear due to Amplifier
                                          //7991 is calculated by setting u[3] to
                                          //zero and measuring output voltage, then
                                          //by knowing the interval at 120 Amp by
                                          //experimentation the ratio may be calculated
    write_32b_reg(data_base_addr,output);
    start=1;
    }
  Time=counter*.512;//Note: The DA rate is .512 ms/cycle or 1.953125 kHz
  if (mask == 1)
    {
    if (Time > x1)  //Only saves once until mask reset.
      {
      output=(((random_input[count]*(-.00196679597881))+1)*7991);//calibrated at 120.065 Amp
                                            //within .2 Amps between 30 - 135 Amps
                                            //slightly non-linear due to Amplifier
      write_32b_reg(data_base_addr,output);
      mask=0;
      }
    }
  ret = 0;
  skip=skip+1;
///*
  if (skip == 19)//This iteration only useful to
                 //initialize output to zero.
                 //Once initialized, comment out.
    {
    skip=0;
    count=count+1;
    }
//*/
  if (count > 115)  //Returns 1 to terminate process function
    {
      write_32b_reg(data_base_addr,7991);
      outputsave[count-1]=0;
    }
  if (count >= 125) //Returns 1 to terminate process function
    {
    ret=1;
      write_32b_reg(data_base_addr,7991);
      outputsave[count-1]=0;
    }
  return (ret);
} /* end appl_process */
int  appl_idle(void)
  {
  loop_count++;
```

```
  return 0;
  } /* end appl_idle */
void appl_end(unsigned int times,
              unsigned int bufs_proc,
              unsigned int buf_count,
              unsigned int prev_buf_count,
              int          DMS_err,
              int          appl_term_code,
              unsigned int FIFO_ovfl,
              unsigned int DMS_count)
  {
  int p;
  #if AED_PRINT
    FILE *outfile;
  #endif
  debug_times          = times;
  debug_bufs_proc      = bufs_proc;
  debug_buf_count      = buf_count;
  debug_prev_buf_count = prev_buf_count;
  debug_DMS_err        = DMS_err;
  debug_appl_term_code = appl_term_code;
  debug_FIFO_ovfl      = FIFO_ovfl;
  debug_DMS_count      = DMS_count;
  #if AED_PRINT
    #if !TALK_TO_FPGA
      printf("FPGA channel simulated\n");
    #endif
    printf("Interrupts received = %d\n", input_count);
    if (DMS_err)
      {
      printf("Read drop error %d (bufct=%d, trnct=%d)\n",
           DMS_err, prev_buf_count, (DMS_count));
      }
    if (appl_term_code)
      {
      printf("Application Termination %d\n", appl_term_code);
      }
    if (FIFO_ovfl)
      {
      printf("FPGA FIFO Overflowed\n");
      }
    printf("Test Loops = %u Bufs Processed = %u Bufs Received = %u\n",
           times, bufs_proc, buf_count);
    // Should be set in AED.h
    //     2 = File
    //     1 = Print  (Default)
    //     0 = No print
    #if (AED_PRINT == 2) /* file I/O */
      printf("Writing outfile.txt\n");
      outfile = fopen("outfile.txt", "w");
    #else
      outfile = stdout;
    #endif
    if (outfile)
      {
      #if (AED_PRINT == 2) /* file I/O */
```

```
      fprintf(outfile,"\n*** AED " AED_BOARD " Non-Linear Control Algorythm System Data ***\n");
    #endif
    fprintf (outfile,"      u[k]            y[k]          KeyHole          Time\n");
    y3save[0]=0;
    KeyHolesave[0]=0;
    Msec[0]=0;
    for (p=1;p<125;p++)
      {
      fprintf (outfile,"%016f %016f  %016f  %016f\n",outputsave[p],y3save[p],KeyHolesave[p],Msec[p]);
      }
    #if (AED_PRINT == 2) /* file I/O */
      fprintf(outfile,"*** Data file complete ***\n");
      fclose(outfile);
      printf("Data file write complete\n");
    #endif
    }
   else
    {
    printf("Failure to open outfile.txt\n");
    }/* end if */
 #endif
 if (DMS_err)
   {
   error_flashing(AED_FLASH_MAIN_DMS_ERROR);
   }
 else if (FIFO_ovfl)
   {
   error_flashing(AED_FLASH_MAIN_FPGA_OVERFLOW);
   }
} /* end appl_end */
```

Bibliography

[1] James E. Lump Jr., Electrical and Computer Engineering, University of Kentucky, EE583 Introduction, What is a Micro-Controller, and why would I take a course about one? http://www.engr.uky.edu/~jel/course/583/583intro/index.htm, October 2002.

[2] Differences between a DSP and Micro Controller (Micro-controller). Texas Instrument's Inc., DSP KnowledgeBase, http://focus.ti.com/general/docs/techsupport.jsp.

[3] Lee Rosenburg, Electrical and Computer System's Engineering, Rensselaer Polytechnic Institute, ECSE-4790 Microprocessor Systems Motorola 68HC12 User's Manual, Revision 1.1, August 2000.

[4] TMS320VC5416 DSK Technical Reference. 5416_dsk_techref, 506005-0001 Rev. a, Spectrum Digital Inc., March 2002.

[5] PCM3002/PCM3003 16-/20-Bit Single-Ended Analog Input/Output STEREO AUDIO CODECS. PDS-1414C, Burr-Brown Inc., January 2000.

[6] Documentation Package for AED-109 Multi-Channel Analog Expansion Daughterboard. Signalware Corporation, Version 1.1, March 2003.

[7] Arc-Welding Fundamentals. The Lincoln Electric Company, Articles, 1994, http://www.lincolnelectric.com/knowledge/articles/content/arcweld.asp.

[8] General Atomics Fusion Education Slideshow. Slide 27, Plasma Defined, General Atomics Fusion Education, http://fusioned.gat.com/images/pdf/slides01-67.pdf.

[9] Yuming Zhang, Electrical and Computer Engineering, University of Kentucky, EE/MFS 699, Joining Processes, Class Handout Notes, Spring 2003.

[10] W. Lu, W.-Y. Lin, and Y. M. Zhang. Non-Linear Interval Model Control of Quasi-Keyhole Arc Welding Process: Automatica, 40(5): 805-813, 2004.

[11] Miller Owner's Manual MAXTRON 450, Form: OM-2206A, Serial No. KC301174. Miller Electric Mfg. Co., November 2002.

[12] Thermal Arc, Arc Welding Power Supplies & Accessories Catalog, Third Edition, Form No. B4-2027. Thermal Arc Inc., July 2003.

[13] Standard Product Catalog, Dynetic Systems Highly Energized DC Servo Motors: Form No. 75dpi.

[14] DSP Starter Kit (DSK) for the TMS320VC5416, Quick Start Installation Guide. 5416_dsk_quickstartguide, 506006-4001B, Spectrum Digital Inc., April 2002.

[15] CPLDs vs FPGAs, Comparing High-Capacity Programmable Logic. PIB18, Version 1, Product Information Bulletin. ALTERA Inc., February 1995.

[16] TMS320 Cross-Platform Daughtercard Specification, Revision 1: Literature No. spra711. Texas Instruments Inc., November 2000.

[17] TMS320VC5416 Fixed-Point Digital Signal Processor Data Manual: Literature No. sprs095k. Texas Instruments Inc., September 2003.

[18] TMS320C54x DSP Reference Set Volume 1: CPU and Peripherals, Literature No. spru131g. Texas Instruments Inc., March 2001.

[19] TMS320VC5416 Fixed-Point Digital Signal Processor Data Manual: Literature No. sprs095h. Texas Instruments Inc., December 2001.

[20] TMS320C54x DSP Reference Set Volume 2: Mnemonic Instruction Set, Literature No. spru172c. Texas Instruments Inc., March 2001.

[21] Code Composer Studio Getting Started Guide: Literature No. spru509c. Texas Instruments Inc., November 2001.

[22] TMS320C54x C/C++ Language Implementation. TMS320C54x Code Generation Tools Help, Texas Instruments Inc., May 2001.

[23] Software Development Tools Overview. TMS320C54x Code Generation Tools Help, Texas Instruments Inc., May 2001.

[24] Code Composer Studio IDE Quick Start: Literature No. spru405a. Texas Instruments Inc., February 2001.

[25] Code Composer Studio User's Guide: Literature No. spru328b. Texas Instruments Inc., February 2000.

[26] TMS320 DSP/BIOS User's Guide: Literature No. spru423a. Texas Instruments Inc., November 2001.

[27] DSP/BIOS Quick Start Reference Guide: Literature No. spru426. Texas Instruments Inc., February 2001.

[28] THS5661A, 12-Bit, 125 MSPS, CommsDAC Digital-to-Analog Converter: Literature No. slas247b. Texas Instruments Inc., September 2002.

[29] THS1209 12-Bit, 2 Analog Input, 8 MSPS, Simultaneous Sampling Analog-to-Digital Converters: Literature No. sla5288b. Texas Instruments Inc., December 2002.

[30] THS4061, THS4062 180-MHz High-Speed Amplifiers: Literature No. slos234d. Texas Instruments Inc., December 1998.

[31] THS3001 420-MHz High-Speed Current-Feedback Amplifier: Literature No. slos217c. Texas Instruments Inc., July 1998.

[32] C5416_dsk. General Extension Language File for TMS320VC5416 DSK. Texas Instruments Inc..

[33] 5416_lnk. Linker Command File for TMS320VC5416 DSK. Texas Instruments Inc., June 2003.

[34] dma5416. DMA5416 Routines Header File. Texas Instruments Inc., February 2002.

[35] dsk5416. Header File for DSK5416 Board Specific I/O Register Definitions for the CPLD. Texas Instruments Inc., December 2001.

[36] emif. Header File which Defines Data Structures and Macros to Access Software Wait State and Bank Switch Control Registers and Associated Bits/Fields, V0.00. Texas Instruments Inc. 2000.

[37] intr5416. Header File which Defines Macros Needed to Enable/Disable Interrupts, Set Interrupt Vectors, Allocate Space for Interrupt Vectors, and Set Interrupt Vector Pointer, V0.00. Texas Instruments Inc., 2000.

[38] regs. Header File which Defines All Peripheral Memory Mapped Register Addresses, V0.00. Texas Instruments Inc., 2000.

[39] regs5416. Header File Extension for regs Header File. Texas Instruments Inc., 2000.

[40] timr5416. Header File which Defines Timer Period and Control Registers with All Related Data Structures, Macros, and Functions, V0.00. Texas Instruments Inc., 2000.

Vita

1. Date and Place of Birth:

   30 March 1968, Louisville, Kentucky

2. Educational Institutions Attended and Degrees Awarded:

   Georgetown College: BS Physics, BA Engineering Arts, and Math Minor 1990.
   University of Kentucky: BS Civil Engineering 1992
   BS Electrical Engineering 2001

3. Professional Positions Held:

   USDA, Soil Conservation Service: Eng. Aide, September 1991 – September 1992
   Central Associated Engineers: Civil Engineer, October 1992 – December 1992
   U.S. Army Corps of Engineers: Civil Engineer, December 1992 – January 2000
   Intertek Testing Services: Electrical Engineer, June 2000 – January 2001
   Lexmark International: Electrical Engineer, January 2001 – April 2001
   Raytheon/L3 Communications: Electrical Engineer, May 2001 – Present

4. Scholastic and Professional Honors:

   Professional Engineer: Kentucky Registration #19849
   Land Surveyor-in-Training: Kentucky Registration #1504
   Microsoft Certified Professional: Registration #1386565

5. Professional Publications:

   N/A

Author: Matthew Wayne Everett

Date: 29 May 2004