Advanced Digital Signal Processing (DSP) Programming & Applications (March 2010)

Damian Szmulewicz

Abstract *— In a world of constant technological development, electronics have become an extremely attractive and competitive market. A microprocessor is the essential component of today's technology. Microprocessors are the heart of most devices used in an every-day basis, from supercomputers, cars and cell phones to simple house appliances such as toasters and coffee makers. For this reason, it is crucial for all electrical and computer engineers to have a deep understanding of microprocessors and their applications. The University of Florida currently offers a course called Microprocessor Applications (EEL 4744) as a specialization course for electrical engineers. The course is based on Motorola's 68HC12 fixed point microprocessor. This device was introduced in the mid 90s and it is presently outdated. As a result, a new microprocessor board based on Texas Instrument's latest TMS32028F335 Digital Signal Processor (DSP) device has been designed. Learning the basics and applications of this new integrated circuit will give students the tools needed to enter the industry of today. In this study, tutorial and labs are redesign for EEL 4744 with the new DSP. Data manuals were analyzed in great details and vital information was collected and documented. In addition, all designed labs were tested on the newest board. Topics of the project include but are not limited to assembly programming, debugging and compiling software, input/output interface, memory expansion, serial communication interface, and analog to digital and digital to analog conversion. It is in this way that the migration from the 68HC12 to the TMS320F28335 will be possible in the near future.*

I. INTRODUCTION

The department of Electrical and Computer Engineering at the University of Florida offers a course on microprocessors and their applications. The UF course catalog describes EEL 4744C as one in which the following topics are covered: elements of microprocessor-based systems, hardware interfacing and software design for their application. A correlative laboratory is associated to the class and required for every student enrolled in the course [1].

With the purpose of teaching students the functional and technological characteristics of microprocessor structures, memory components, peripheral support devices, and interface logic, the course syllabus covers a vast list of topics. The course's lectures begin with a review of number systems, including binary, decimal, and hexadecimal, and follows by an introduction to computer architecture during the first and second weeks. Students are then exposed to more elaborate topics including memory interfacing, assembly language programming, debugging, microprocessor's instruction set, software design concepts, computer busses, parallel input and output (I/O), software interrupts, real time events, and timers. The course finalizes with an introduction to serial I/O and analog I/O. The lectures occur 3 times a week and last fifty minutes each. The laboratory session takes place once a week and is intended to provide students a hands-on experience on the topics covered in the lectures. The instructor is not present in the laboratory sessions; however, a teacher assistance that works back-to-back with the professor is in charge of guiding students on their projects.

The faculty in charge of achieving the goals and objectives of EEL 4744C is formed by experienced experts in the fields of computer engineering: Dr. Karl Gugel (Figure 1), a senior lecturer with an MS degree in electrical engineering from Florida Atlantic University (1993) and a PhD in electrical and computer engineering from the University of Florida (1997); Dr. Eric Schwartz (Figure 2), a master lecturer with an MS degree (1989) and PhD (1995) in computer and electrical engineering from the University of Florida; and Dr. Tao Li (Figure 3), an assistant professor with an ME degree in computer engineering from the Beijing Institute of Data Processing Technology (1996) and a PhD in electrical and computer engineering from the University of Texas at Austin (2004). Dr. Gugel and Dr. Schwartz are also affiliated to the Machine Intelligence Lab (MIL) which "provides a synergistic environment dedicated to the study and development of intelligent, autonomous robots [2].

Fig. 1: Dr. Karl Gugel

Fig. 2: Dr. Eric Schwatz

Fig. 3: Dr. Tao Li

History shows that Dr. Gugel and Dr. Schwartz have worked together to maintain EEL 4744 up to date. In 2002, students began using Motorola's 68HC12 microprocessor in their lab sessions. Scott Kanowitz, one of Dr. Schwartz's MIL students, designed a 68HC12-based processor board for the course and Patrick O'Malley, another MIL student, wrote the software monitor for this processor. The main improvement of the 68HC12 board over its predecessor was the inclusion of a complex programmable logic device (CPLD) on board. In 2006, a new board also based on the same Motorola's device, was introduced to the course. This new board had an USB port, necessary for many students whose laptops lacked serial communication ports. Although this new board helped to face some of the obstacles generated by the advancement of technology, it did not take long for faculty to realize that the 68HC12 was on the path of becoming outdated. Consequently, Dr. Gugel and Dr. Schwartz began analyzing the possibility of designing a new microprocessor board based on an entirely new device. Texas Instrument's latest TMS32028F335 Digital Signal Processor (DSP) device was the chosen microprocessor for the task, due to its powerful features.

The rapid improvement of technology in the last ten years has left in the past many pieces of electronics that were once of great applications. An excellent example of such devices is the 68HC12, a 16-bit microcontroller produced by Freescale Semiconductors. This device was introduced in the mid 90s and is now considered obsolete. Its technology allows a maximum of 8Mhz bus speed at 5V and 5Mhz at 3V. 1024 bytes of RAM and 4096 Bytes Electrically Erasable Programmable Read-Only Memory are available within this device. In addition, it presents a 16-bit non-multiplexed address bus, an 8-bit non-multiplexed data bus, and a 20-bit Arithmetic Logic Unit (ALU). A 16-bit pulse accumulator and a real-time interrupt circuit are embedded on the 68HC12. Seven programmable chip selects with clock stretching are available in expanded modes. 8-channels configurable as input capture or output compare and enhanced 16-bit timer with programmable pre-scaler allows flexible choice of clock source. For properly operation a COP watchdog can be turned on in software. For communication, two enhanced asynchronous non-return to zero (NRZ) serial communication interfaces (SCI) and one enhanced synchronous serial peripheral interface (SPI) are also available. Finally, only one 8-channel, 8-bit analog-to-digital converter (ATD) was within the features of the 68HC12.

The TMS320F28335 is Texas Instrument's latest Digital Signal Processor device. It is designed with many high-tech features. Its high-performance Static CMOS technology allows up to a clock frequency of 150 MHz or 6.67-ns Cycle Time. Its high-performance 32-bit CPU, which is based on Harvard architecture and IEEE 754 Single-Precision Floating Point Unit, allows fast interrupt response and processing, 16 x 16 and 32 x 32 MAC Operations, 16 x 16 Dual MAC, and code efficiency (both in assembly and C/C++). It has a six-channel DMA controller for ADC, McBSP, ePWM, XINTF, and SARAM. It features a 16-bit or 32-bit external interface (XINTF) which allows an address reach of over 2M x 16. It has two on-chip volatile memories: a 256K x 16 Flash and a 34K x 16 SARAM and an 8k x 16 boot nonvolatile ROM. Its clock and system control supports Dynamic PLL ratio changes and is composed of and on-chip oscillator and a Watchdog Timer module. Eighty-eight individually programmable, multiplexed general purpose input output (GPIO) pins are found in the device. Sixty-four of them (GPIO0 to GPIO63) can be connected to one of the eight external core interrupts. A 28-bit security key is present and optional in the device to protect flash, OTP and RAM blocks and to prevent reverse engineering. The device is also fully equipped with enhanced control peripherals: Up to 2 CAN Modules, up to 3 SCI (UART) modules, up to 2 McBSP modules, 1 SPI module, and 1 inter-integrated-circuit module. For analog to digital and digital to analog conversion a 12-Bit ADC with 16 channels is integrated in the device. The ADC has a 80-ns conversion rate and features a 2 x 8 channel input multiplexer, 2 sample-and-hold, single simultaneous conversions and internal or external reference. Advance emulation features allow analysis and breakpoint functions and real-time debugging. These features make the TMS320F28335 DSP device 1000 times faster and more powerful than the 68HC12 microprocessor. Table 1 shows a comparison of both microprocessors under discussion. Therefore; it is vital, for the preparation of future electrical and computer engineers, to migrate to TI's device.

Table 1: DSP Comparisson

Feature	Motorola 68HC12	TI TMS320F28335
CPU	16-bit @ 8MHz	16-bit @ 150MHz
RAM	1KB	68KB
Flash	32KB	512KB
ADC channels	10-bit, 8 channels	12-bit, 16 channels
ADC conversion time	500ns	80ns
SCI	1x UART	3x UART
CPU timer	1x 16-bit	3x 32-bit
Watchdog Timer	Yes	Yes
Max GPIO pins	63	88
Core Voltage	5.0V	3.3V

II. PROCEDURE

A systematic approach was taken for this research project. The procedure was broken down into three main stages: A. Problem identification, B. Research and C. Development.

A. Problem Identification

During the first stage, the laboratory documentation based on Motorola's 68HC12 microprocessor created by faculty from the Electrical Engineering Department in previous years, was analyzed in detail. The purpose, procedure and solutions of each laboratory were identified. A total of nine laboratory handouts were studied in depth and the purpose and hardware needed for the completion of each assignment were then noted in Table 2. Stage 1 allowed researchers to visualize each of the problems that were to be solved.

Table 2

Microprocessor Applications Laboratory Description									
Laboratory Number	1	2	3	4	5	6	7	8	9
Laboratory Name	Soldering/Wire-wrapping Tutorial & Practice	F28335 Dev Board Construction & CC4 Assembly/Debugging	Assembly Programming & Elementary Wiring	Elementary Assembly Programming & Adding a Keypad	Software Switch Debounce & Adding an 8 Bit External Output Port	Input Buffer & Memory Expansion	LCD Display and Real Time Interrupt	SCI communication	A/D and D/A
Laboratory Puspose	Learn how to solder and wire-wrap	Get familiar with programming tools and add headers to the board for future use	Learn how to program in assembly language and to add LEDs and switches to the board	Additional programming practice and to learn how to add a keypad to a CPU	Create de-bounce software for the keypad and add and external output port	Add an external 8-bit input port and to add memory expansion	Add an LCD display to the CPU and learn how to write to it. Use interrupts to create a simple stop watch	Communicate serially with external devices	Add an A/D to create a digital voltmeter. Learn how to output a triangle wave to a speaker
Hardware Needed	Soldering iron, wire, solder, flux, wire-wrapping tool and dummy ICs	Soldering iron, wire, solder, flux, wire-wrapping tool	LEDs, resistors, headers, wire, solder, iron, wire-wrapping tool and switches	Keypad, headers, wire, solder, iron and wire-wrapping tool	D-flip flop, headers, wire, solder, iron and wire-wrapping tool	Tri-state buffer, SRAM , headers, wire, solder, iron and wire-wrapping tool	LCD, headers, wire, solder, iron and wire-wrapping tool	UART, RS232 header, headers, wire, solder, iron and wire-wrapping tool	Speaker, A/D, D/A, Op amp, capacitor, resistors, headers, wire, solder, iron and wire-wrapping tool

The first laboratory is intended to teach students how to solder parts, such as integrated circuits and headers, to an electronic board, and to wire-wrap pins of different ICs to create electrical connections. Students are introduced to various soldering tools including solder, flux, solder suckers, solder wick, and soldering irons, and wire-wrapping tools including wire, wire strippers, and wire cutters.

The second laboratory aims to help students get familiar with programming tools, such as MinIDE, and to begin the construction of the microprocessor board. Basic syntax for assembly programming is presented and discussed.

The purpose of the third laboratory is to reinforce programming techniques taught during the previous laboratory and to add switches and Light Emitting Diodes (LEDs) to the microprocessor board. The concepts of pull-up and pull-down resistors, as well as the difference between an anode and a cathode, are discussed. In addition, students are encouraged to learn how to read data sheets extracting important parameter, an essential ability for the design of electronic circuits.

In the fourth laboratory, students are expected to become experts in assembly programming. In addition, students learn how to add a keypad to a microprocessor for input purposes, and to constantly scan this keypad via software. Every computer has a microcontroller whose sole purpose is to scan the keyboard; therefore, this laboratory allows students to work on a "real-world" application of microprocessors.

The fifth laboratory introduces students to the concept of external interface. Adding an output port to the microprocessor board exposes students to timing diagrams, memory maps, and to the need for flip-flops as a mean of interfacing with a data bus. Moreover, students learn to debounce switches for obtaining accurate input readings.

The sixth laboratory is a supplement to the previous assignment. Three-state buffers and SRAMS are presented to students. The concepts of memory mapping, address decoding and external interfacing are reinforced, providing students with the knowledge and familiarity needed in this topic. Additionally, dynamic and statics busses and control signals are studied. A very important tool, the Logic State Analyzer, is presented to students as an essential debugging device.

The seventh laboratory is intended to teach students how to add and communicate to an external LCD display. LCDs have a microcontroller of their own that runs much slower than both the 68 HC12and the TMS320F28335; therefore, precise timing is required for proper communication. Timer interrupts are also presented in this assignment Interrupts are used to design a precise stopwatch on the LCD.

The eighth laboratory teaches students to communicate a CPU serially with another device, such as another DSP or a personal computer. Serial Communication Interface (SCI) hardware is presented including the Universal Asynchronous Receiver/Transmitter (UART) and the RS232 dB9 header. The RS232 standard is studied and the concepts of baud rate, parity bits, and stop and start bits are introduced.

The ninth and final laboratory introduces the topics of analog-to-digital (A/D) and digital-to-analog (D/A) conversion. The Shannon sampling theorem is studied and applied. Students use an A/D IC to create a digital voltmeter and a D/A IC to output a continuous triangle wave to a speaker. In addition, an oscilloscope is used to observe the output signal.

B. Research

The main source for this research was Texas Instruments' web site: www.ti.com. TMS320F28335 literature was analyzed. Specifically, the Data Manual (SPRS439f), the TMS320x2833x, 2823x DSC External Interface (XINTF) Reference Guide (SPRU949), the TMS320C28x CPU and Instruction Set Reference Guide (SPRU430E), the TMS320x2833x, 2823x System Control and Interrupts Reference Guide (SPRUFB0), and the TMS320x2833x, 2823x Serial Communications Interface (SCI) Reference Guide (SPRUFZ5A) were referenced for the completion of all nine laboratories.

The Data Manual provides an overview of all functions of the TMS320F28335 DSP, including a functional overview: pin assignments, signal description, memory maps, memory bus, peripheral bus, real-time JTAG and analysis, external interface, flash, M0 and M1 SARAMs, L0, L1, L2, L3, L4, L5, L6, L7 SARAMs, boot ROM, security, peripheral interrupt expansion (PIE) block, external interrupts (XINT1-XINT7, XNMI), oscillator and PLL, watchdog, peripheral clocking, low-power modes, peripheral frames 0, 1, 2, 3 (PFn), general-purpose input/output (GPIO) multiplexer, 32-Bit CPU-timers (0, 1, 2), control peripherals, serial port peripherals, register map, device emulation registers, interrupts, external interrupts, system control, OSC and PLL block , external reference oscillator clock option, PLL-based clock module, loss of input clock, watchdog block, and low-power modes block; available peripherals: DMA, 32-Bit CPU-timers (0/1/2, 3) enhanced PWM modules (ePWM1/2/3/4/5/6), high-resolution PWM (HRPWM), enhanced CAP modules (eCAP1/2/3/4/5/6), enhanced QEP modules (eQEP1/2), analog-to-digital converter (ADC) module, ADC registers, ADC calibration, multichannel buffered serial port (McBSP) module, enhanced controller area network (eCAN) modules (eCAN-A and eCAN-B), serial communications interface (SCI) modules (SCI-A, SCI-B, SCI-C), serial peripheral interface (SPI) module (SPI-A), GPIO MUX, and external interface (XINTF); device support: device and development support tool nomenclature, and documentation support; electrical specifications: absolute maximum ratings, recommended operating conditions, electrical characteristics, current consumption, clock requirements and characteristics, power sequencing, general-purpose input/output timing, external interrupt timing, serial peripheral interface timing, external interface timing, ADC power-up control bit timing, multichannel buffered serial port (McBSP) timing, and flash timing; and thermal/mechanical data [3]. This document served as a guide for laboratories 2 through 9.

The TMS320x2833x, 2823x DSC External Interface Reference Guide provides a deep description of the external interface functions including a functional description, a XINTF configuration overview, external DMA support, configuring lead, active, and trail wait states, configuring XBANK cycles, XINTF registers, signal descriptions, and waveforms [4]. This document served as a guide for laboratories 5 and 6.

The TMS320C28x CPU and Instruction Set Reference Guide provides a description of the central processing unit (CPU) and the assembly language instructions of the TMS320C28x 32-bit fixed-point CPU [5]. This document was mainly use for assembly language syntax for the software development of all laboratories.

The TMS320x2833x, 2823x System Control and Interrupts Reference Guide provides a description of system clocking, timer interrupts and Peripheral Interrupt Expansion [6]. This literature provided the needed material for the completion of labs 7, 8, and 9.

The TMS320x2833x, 2823x Serial Communications Interface (SCI) Reference Guide provides an enhanced SCI module overview and a profound description of SCI registers [7]. This document served as a guide for laboratory number 8.

C. Development

During the development stage, all laboratories previously described were reproduced using the TMS320F28335 DSP device. In this section the platform used for software development, Code Composer Studio is described. In addition, the bases for each laboratory are introduced.

I. Code Composer Studio

All laboratories were reproduced using Code Composer Studio version 4. Code Composer Studio v4 (CCS v4) is the integrated development environment for TI's DSPs, microcontrollers and application processors [8]. TI offers Code Composer Studio version 4 for free here: http://tiexpressdsp.com/index.php/Category:Code_Composer_Studio_v4.

Texas Instruments provides the following description for CCS v4: "Code Composer Studio includes a suite of tools used to develop and debug embedded applications. It includes compilers for each of TI's device families, source code editor, project build environment, debugger, profiler, simulators and many other features. The CCS IDE provides a single user interface taking you through each step of the application development flow. Familiar tools and interfaces allow users to get started faster than ever before and add functionality to their application thanks to sophisticated productivity tools. As of version 4 CCS is based on the Eclipse open source software framework. The Eclipse software framework is used for many different applications but it was originally developed as an open framework for creating development tools. We have chosen to base CCS on Eclipse as it offers an excellent software framework for building software development environments and is becoming a standard framework used by many embedded software vendors. CCS combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from Texas Instruments resulting in a compelling feature rich development environment for embedded developers" [8].

CCS v4 provides an intuitive integrated development environment (IDE) as shown in Figure 4. CCStudio's IDE allows the user to visualize, in a single console, all created projects, both active and inactive, ASM files, C files, and current problems with files, including syntax issues and illegal operand problems. This way the user has the capability of

switching among different projects and files in just a matter of seconds. When CCS is first run, the user is required to create a workspace in which all projects will be saved. Upon the completion of the workspace, users are ready to begin creating projects by simply selecting a project name and project directory within the workspace. One of the key features of CCS is the simulator: "Simulators provide a way for users to begin development prior to having access to a development board. Simulators also have the benefit of providing enhanced visibility into application performance and behavior. Several simulator variants are available allowing users to trade off cycle accuracy, speed and peripheral simulation, with some simulators being ideally suited to algorithm benchmarking and others for more detailed system simulation" [8]. By selecting different target configurations, the user can run programs in a development board or in a TI simulator.

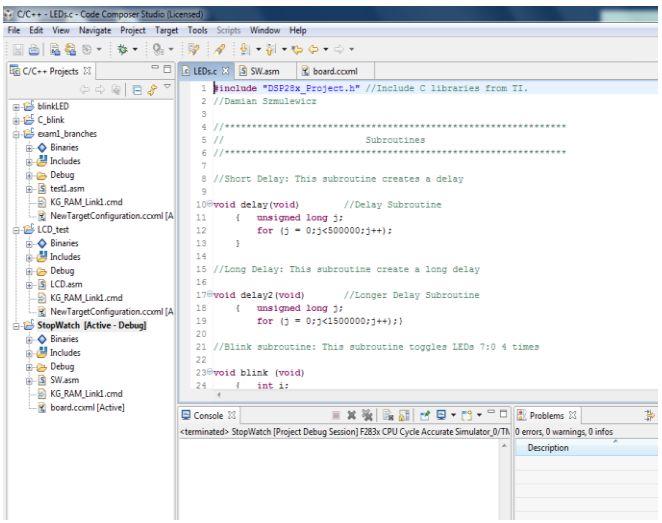


Fig. 4: CCS v4 IDE screenshot

CCStudio's debugger has been designed to simplify development. Within it vast features, an advanced register window, shown in Figure 5, and an advanced memory window, shown in Figure 6, provide debugging capabilities lacked by many design software such as MinIDE.

The simulation toolbar shown in Figure 7 allows users to run a program at full speed or step through each line of instructions. This feature is the most powerful for students learning to work with a microprocessor as they can clearly see what each instruction does to memory and to all the different registers.

Fig. 7: Simulation toolbar. Green 'play' button runs code at full speed, red 'stop' button terminates execution, yellow and green arrows step through instructions of code.

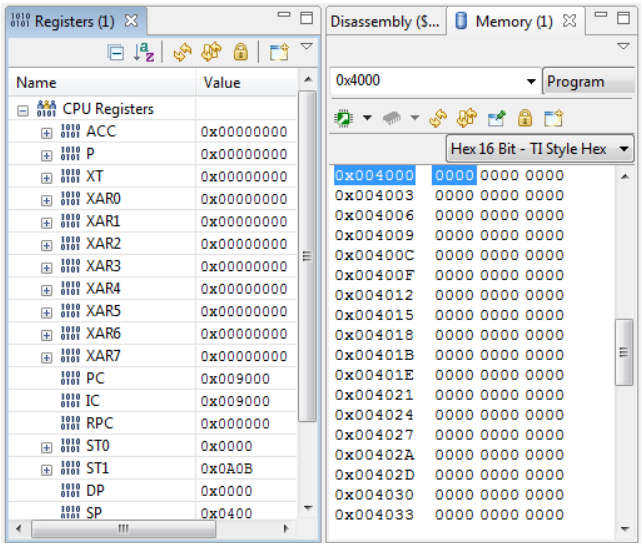


Fig. 5(left):advancedd register window. Fig. 6 (right): advanced memory window.

II. Watchdog Timer and 'Eallow' Protection

The watchdog timer is an error-control software or hardware that triggers a system reset if the CPU fails to regularly service the watchdog service routine. In order to run programs at full speed without having the watchdog timer resetting the CPU, the following lines of code must be included at the beginning of all ASM files:

```
MOV   AL,#0x0068
MOV   AR1,#0x7029
MOV   *AR1,AL
```

Some CPU registers are 'Eallow' protected and must be enabled before they can be modified. An example of such registers is the general input/output multiplexer registers. Eallow protects certain registers from spurious writes after configuration. Using the "EALLOW" assembly instruction prior to the register access allows access and the "EDIS" instruction disallows access to these registers.

III. Memory Map

The memory map for the TMS320F28335 DSP, which includes ranges of SRAM available, the peripheral frames, external interface zones and reserved spaces, is found in Appendix A.

IV.CPU Registers

The CPU architecture presents numerous registers. Refer to Appendix C for a complete Architecture block diagram.

The main working register is the 32-bit Accumulator (ACC). The ACC is formed of two halves, the AH (high 16 bits) and AL (low 16 bits), which can be accessed independently. All values from the ALU, except those that operate directly on other registers or memory, are received by the ACC. The ACC resets to the value 0x0000000.

The Multiplicand Register (XT) is mainly used for storing a 32-bit integer for use in a multiply operation. Two halves can be accessed independently: XT (high 16 bits) and T (low 16 bits). XT resets to the value 0x0000000.

The Product Register (P) is also a 32-bit register used to store the result of a multiplication. The P register is composed of PH (high 16 bits) and PL (low 16 bits), each can be accessed independently. P can also be loaded from a memory location, the ACC or a constant value. P resets to 0x0000000.

The Data Page Pointer Register (DP) is a 16-bit register used for direct addressing modes. Blocks of 64 words are defined as a page. DP stores the page number to be accessed. DP resets to 0x0000.

The Stack Pointer (SP) is a 16-bit register that can address only the low 64K of memory, from 0x0000000 to 0x0000FFFF. The stack grows from low to high in memory and is little endian. When the stack reaches the value 0xFFFF it overflows to 0x0000. SP resets to 0x0400.

The Auxiliary Registers (XAR0-XAR7) are 32-bit registers and can be used as pointers of as general purpose registers. The lower 16-bits of each auxiliary register (AR0-AR7) can be accessed independently. XAR0-XAR7 reset to 0x00000000.

The Program Counter (PC) is a 22-bit register which points to the instruction that is currently being processed. PC resets to 0x003FFFC0.

The Return Program Counter (RPC) is a 22-bit register that saves the return address when a call operation is made using the LCR instruction. When a return operation is made using the LRET instruction, the return address is loaded from the RPC register.

The Status Registers (ST0, ST1) are 16-bit registers that contain flag and control bits, including overflow, zero, carry and sign flags. ST0 resets to 0x0000 and ST1 resets to 0x080B.

The Interrupt Control Registers are 16-bit register consisting of the Interrupt Enable Register (IER), the Interrupt Flag Register (IFR), and the Debug Interrupt Enable Register (DBGIER). In order to enable an interrupt, a ‘0’ has to be written to the register. A ‘1’ clears the bit and disable the interrupt routine. Maskabe interrupts are accomplished using bits from the IFR. All registers reset to 0x0000.

III. RESULTS

All laboratories were reproduced with relative ease using the TMS320F28335 DSP. Sample code produced during the development phase of this project is found in Appendix B. The breakout board was built by William Goh, from Texas Instruments, and Dr. Karl Gugel. Several versions of the board were built until all errors were corrected. Figure 8 shows an unpopulated DSP board and Figure 9 shows the board after the development phase of this project.

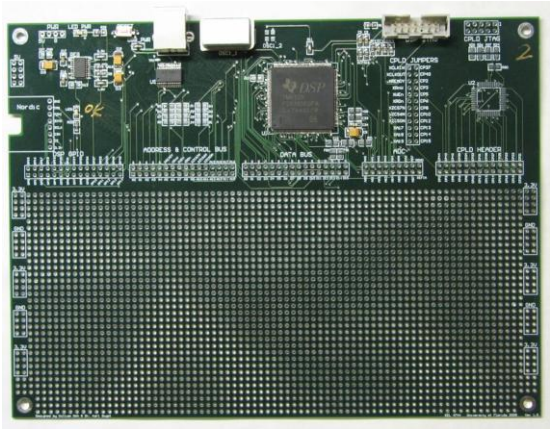


Fig. 8: Unpopulated DSP board

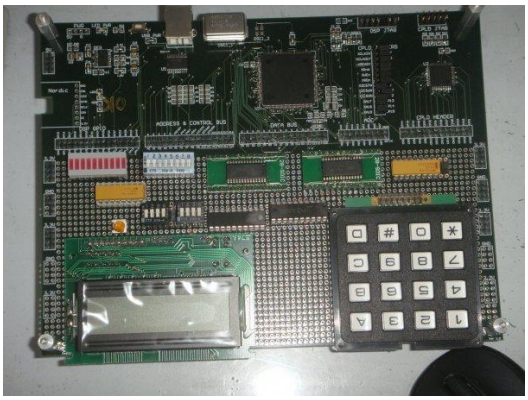


Fig. 9: Populated DSP board

All laboratories were then successfully tested to meet the requirements specified in the laboratory handouts. Dr. Gugel supervised this verification process. Updated laboratory handouts were created by faculty and revised by researchers.

The new DSP board was introduced to EEL 47744 in the Fall semester of 2009. Students showed a high degree of interest in the new processor, mostly due to the improved features and the user friendly interface. Figure 10 shows a group of students with complete DSP boards at the end of the Fall 2009 semester. TI’s TMS320F28335 DSP is still the processor used in the class at the time of the creation of this paper.

Fig. 10: Students of Microprocessor applications. From left to right: Kate Sutton, Erick Mecias, Aminatu Oyebanjo, and Franklin Cooper.

IV. REFERENCES

[1] University of Florida, Guides to Majors 2007-2008. Gainesville : Office of the University Registrar, 2007.

[2] " Machine Intelligence Laboratory". University of Florida. 01/2010 <http://mil.ufl.edu/>.

[3] Texas Instruments, Data Manual - SPRS439f. Dallas: Texas Instruments, 2007

[4] Texas Instruments, TMS320x2833x, 2823x External Interface (XINTF) Reference Guide (Rev. D) - SPRU949. Dallas: Texas Instruments, 2007.

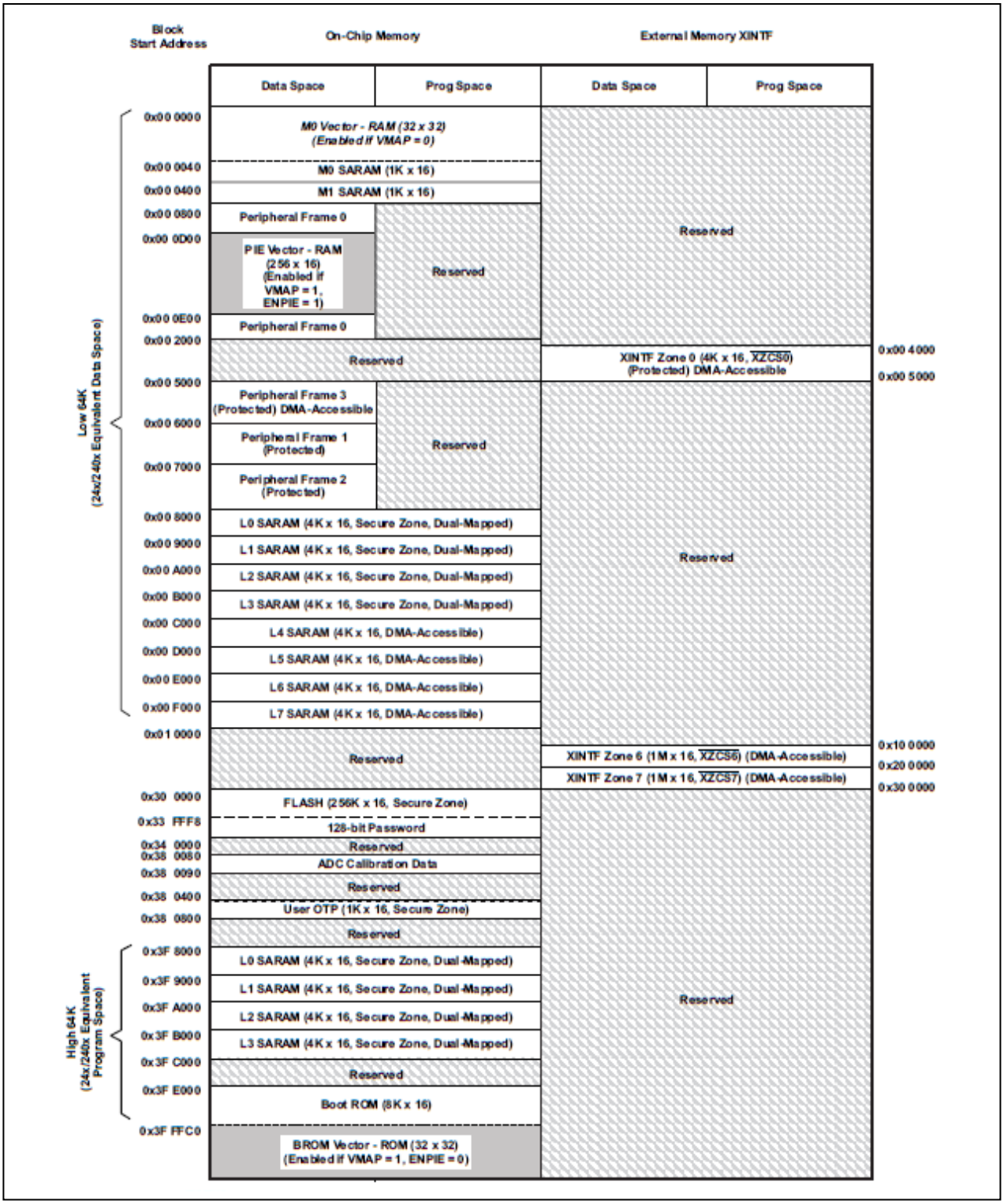
[5] Texas Instruments, TMS320C28x CPU and Instruction Set Reference Guide - SPRU430E. Dallas: Texas Instruments, 2001.

[6] Texas Instruments, System Control and Interrupts Reference Guide - SPRUFB0 . Dallas: Texas Instruments, 2007.

[7] Texas Instruments, TMS320x2833x, 2823x Serial Communications Interface (SCI) Reference Guide - SPRUFZ5A. Dallas: Texas Instruments, 2008.

[8] "Digital Signal Processors & ARM Microprocessors - Code Composer Studio IDE". Texas Instruments. 03/02/10 <http://focus.ti.com/dsp/docs/dspsupportatn.tsp?sectionId=3&tabId=415&familyId=44&toolTypeId=30>.

V. APPENDIX A: TMS320F28335 MEMORY MAP



VI. Appendix B.1: Sample Code: C Code

```
#include "DSP28x_Project.h" //Include C libraries from TI.

//*************************************************************
//                          Subroutines
//*************************************************************

//Short Delay: This subroutine creates a delay

void delay(void)     //Delay Subroutine
  { unsigned long j;
    for (j = 0;j<500000;j++);
  }

//Long Delay: This subroutine create a long delay

void delay2(void)    //Longer Delay Subroutine
  { unsigned long j;
    for (j = 0;j<1500000;j++);}

//Blink subroutine: This subroutine toggles LEDs 7:0 4 times

void blink (void)
  { int i;
    delay ();
    for (i = 0;i<4;i++)
    {GpioDataRegs.GPATOGGLE.all = 0xFF;
    delay2 ();
    }
  }

//Paint left: this subroutine turns on the LEDs one at a time
//from right to left creating the effect of a a brush painting
//the LEDs. This subroutine takes an unsigned long number X and
//returns the modified X to main.

unsigned long paint_left (unsigned long X)
{again:  GpioDataRegs.GPATOGGLE.all = X;
    X = X*2; //Shift right
    if (X == 0x100);
    else { delay();
      goto again; }
    return X;
}

//Erase right: this subroutine turns off the LEDs one at a time
//from left to right creating the effect of erasing
//the LEDs. This subroutine takes an unsigned long number Y and
//returns the modified Y to main.

unsigned long erase_paint (unsigned long Y)
{   again2:  delay();
    GpioDataRegs.GPATOGGLE.all = Y;
    Y = Y/2; //shift_left
    if (Y != 0x01)
      goto again2;
    return Y;
}
```

```
//*************************************************************
//                          Subroutines
//*************************************************************
//Sweep left: this subroutine turns on one LED at a time
//from left to right creating the a sweeping effect
//This subroutine takes an unsigned long number Z and
//returns the modified Z to main.

unsigned long sweep_left (unsigned long Z)
{   again3: delay();
      GpioDataRegs.GPADAT.all = Z;
    Z = Z*2; //Shift right
    if (Z != 0x80)
      goto again3;
    return Z;
}

//Sweep right: same as Sweep left but in the opposite direction

unsigned long sweep_right (unsigned long W)
{   again4: delay();
      GpioDataRegs.GPADAT.all = W;
    W = W/2; //Shift left
    if (W != 0x00) goto again4;
    else W = 0x02;
    return W;
}
//*************************************************************
//                          Main Program
//*************************************************************

void main( void )
{
  unsigned long  Number;    //Declare variables
  InitSysCtrl();     //Disable Watchdog, initialize SysCLK and Peripheral
Clock

  EALLOW;
  GpioCtrlRegs.GPAMUX1.all = 0x00;  //set GPIO7:0 pins to GPIO
  GpioCtrlRegs.GPADIR.all = 0xFF;   //set GPIO7:0 as output
  Number = 0x1;           //Initialize variable Number to 1
  while (1)            //Create an infinite loop.
  { delay();          //call short delay subroutine
    Number = paint_left (Number); //call paint_left. Argument: Number. Result:
X
    blink ();          //call blink. Argument: Void. Result: Void
    Number = erase_paint (Number);  //call erase_pain. Argument: X. Result:
Number
    Number = sweep_left (Number);   //call sweep_left. Argument: Number.
Result: Y
    Number = sweep_right (Number);  //call sweep_right. Argument:Y. Result:
Number
    Number = sweep_left (Number);   //call seep_left Argument: Number. Result:
Z
    Number = sweep_right (Number);  //call sweep_right. Argument: Z. Result:
Number

  }

}
```

VII. Appendix B.2: Sample Code: ASM

```
;Project 3.2: RAM_TEST
;Adam Mills, Damian Szmulewicz
;Description: loads ext RAM (0x100000-0x108000) with 8 bit count value starting
from 0 and going to FF and then looping back to 0

;*******************************************************************************
;constants section
;*******************************************************************************

GPAMUX2  .set   0x6F88    ;registers for pin function selection
GPBMUX1  .set   0x6F96
GPBMUX2  .set   0x6F98
GPCMUX1  .set   0x6FA6
GPCMUX2  .set   0x6FA8
PCLKCR3  .set   0x7020    ;register containing XTIMCLK enable
BEGIN    .set  0x100000  ;begining address of RAM

       .global   _c_int00
;*******************************************************************************
;program section
;*******************************************************************************

    .text
_c_int00:
     LC   INIT_CPU    ;SET OBJMODE, TURN OFF WATCHDOG, TURN ON PROTECTED WRITE
     LC   INIT_XINTF    ;SET PINS FOR XINTF, TURN ON XTIMCLK
     MOVL XAR1, #BEGIN    ;LOAD XAR1 WITH FIRST ADDRESS
     MOV  AL, #0      ;HOLDS COUNT
LOOP MOV  *XAR1++, AL     ;STORE LSB OF COUNT VALUE TO ADDRESS
     ADD  ACC, #1       ;INCREMENT COUNT
     CMP  AL, #0x8000     ;CHECK IF LAST ADDRESS
     B   LOOP, NEQ     ;REPEAT IF MORE ADDRESSES LEFT TO FILL
END  B   END, UNC    ;END PROGRAM
INIT_CPU:
     SETC OBJMODE       ;OBJMODE MUST BE SET OR INDIRECT ADDRESSING WILL NOT WORK
     EALLOW          ;TURN ON PROTECTED REGISTER ACCESS
     MOVZ DP,#0x7029>>6  ;TURN OFF WATCH DOG
     MOV  @0x7029,#0x68
     LRET
INIT_XINTF:
     MOV  AL, #0x0000    ;SETTING GPIO MUXES FOR XINTF CONTROL
     MOV  AH, #0xFF00    ;(SEE PAGES 93-97 IN SPRS439E FOR DETAILS)
     MOV  AR0, #GPAMUX2
     MOVL *XAR0, ACC
     MOV  AL, #0xFFFF
     MOV  AH, #0xFFFF
     MOV  AR0, #GPBMUX1
     MOVL *XAR0, ACC
     MOV  AR0, #GPBMUX2
     MOVL *XAR0, ACC
     MOV  AR0, #GPCMUX1
     MOVL *XAR0, ACC
     MOV  AR0, #GPCMUX2
     MOVL *XAR0, ACC
     MOV  AR0, #PCLKCR3  ;SET BIT 12 OF PCLKCR3 TO ENABLE XTIMCLK
     TSET *AR0, #12
     LRET
```

VIII. APPENDIX C: ARCHITECTURE BLOCK DIAGRAM