ECE 4999: INDEPENDENT STUDY

Syed Tahmid Mahbub (NetID: sm893)

CORNELL UNIVERSITY

Under the supervision of: Professor Bruce R. Land

Fall 2013

Exploring and comparing the peripheral modules of the Microchip PIC32MX250F128B (on the Microstick II platform) and the Atmel AT91SAM3X8E (on the Arduino Due platform). The Microchip PIC32MX250F128B is a 32-bit RISC microcontroller based on the MIPS M4K core. In this independent study, I have used the Microstick II platform provided by Microchip. The Atmel AT91SAM3X8E is a 32-bit ARM microcontroller based on the Cortex-M3 core. In this independent study, I have used the Arduino Due platform (which is based on the Atmel AT91SAM3X8E microcontroller) provided by Arduino.

TABLE OF CONTENTS

1. INTRODUCTION

The Microchip PIC32MX250F128B is a 32-bit RISC microcontroller based on the MIPS M4K core. In this independent study, I have used the Microstick II platform (http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en556208) provided by Microchip. The Microstick II platform consists of a microcontroller socket mounted on the board (on which I mounted the PIC32MX250F128B for my study) along with an on-board programmer and debugger. The XC32 compiler is used in the MPLABX IDE.

The Atmel AT91SAM3X8E is a 32-bit ARM microcontroller based on the Cortex-M3 core. In this independent study, I have used the Arduino Due platform (http://arduino.cc/en/Main/arduinoBoardDue) provided by Arduino. The Arduino Due is a board that has the AT91SAM3X8E soldered on to it, along with an on board programmer. However, unlike the Microchip PIC32MX250F128B (where the compiler and IDE used are those provided by Microchip), the Arduino compiler is used instead of Atmel's GCC compiler on the Atmel Studio IDE.

The purpose of this independent study is to get introduced to and learn about two 32-bit microcontroller series and the corresponding platforms developed by two manufacturers, based on different architectures. The focus was to explore the different peripheral modules and draw upon a comparison of the peripheral modules based on the experience.

2. MICROCONTROLLER VS PLATFORM

The PIC32MX250F128B and the AT91SAM3X8E microcontrollers are not used directly in this study. As mentioned above, the Microstick II and Arduino Due platforms (based on the abovementioned microcontrollers) are used.

The Microstick II platform is essentially just an add-on to the microcontroller itself. The board provides a programmer and a debugger. For programming, the XC32 compiler is used on the MPLABX IDE. The microcontroller is directly programmed, whereas the Microstick II board acts as just an additional programmer hardware.

On the other hand, the Arduino Due is a complete platform with its own compiler and IDE (also named Arduino). So while the AT91SAM3X8E microcontroller *is* being used, it is easy to forget so and even ignore this, as the entire platform focuses on the Arduino *name*. The significance of this is that it makes it extremely easy to use the Arduino Due (with a focus not on the microcontroller itself, but rather on *application development* based on a powerful microcontroller) due to the in-built functions. However, the use of these in-built function defeats the purpose of exploring the peripherals of the AT91SAM3X8E. For example, a library function can be used to simply do an analog to digital conversion in one statement. While this is convenient and makes it easy for anyone to pick up on this platform, it does not provide insight into the hardware itself. This being said, the compiler and platform do not restrict the user from manually accessing the “low-levels” of the microcontroller, by directly programming the special function registers (SFRs) instead of using the internal library functions. This is what I aimed to do in order to explore the peripherals of the AT91SAM3X8E. Additionally, the library functions provided by Arduino were interesting to have as well, as they could be analyzed and compared with the low-level programs, and sometimes even be used when necessary.

3. PERIPHERALS

The AT91SAM3X8E and the PIC32MX250F128B are both very capable microcontrollers with a plethora of powerful versatile internal peripherals. Over the course of this semester, in my independent study, I have played with the following peripherals:

- IO ports
- Internal Timers/Counters
- Compare modules
- Capture modules
- Analog-to-Digital Converter (ADC)
- Analog Comparator
- Digital-to-Analog Converter (DAC)
- Interrupt controller
- UART module

4. KEY COMPARISONS BETWEEN PERIPHERAL MODULES

a. IO PORTS

Both the PIC32MX250F128B and the AT91SAM3X8E come with several IO pins. The AT91SAM3X8E, being the larger microcontroller (in terms of pin count), obviously has a larger number of IO pins. However, given the pin count on both devices and considering that they will both be used in applications where they will have enough IO lines, there are sufficient pins for application.

The PIC32MX250F128B has pull-ups and pull-downs on all of its IO pins. On the other hand, the AT91SAM3X8E only has internal pull-ups, but no internal pull-downs. However, this is just a nice design feature to have; the circuit and software can be changed to not require a pull-down, but suffice with only a pull-up. Nonetheless, it is definitely a nice feature to have.

The PIC32MX250F128B IO pins all have the feature to enable or disable open-drain configuration for individual pins. The AT91SAM3X8E similarly offers the multi-drive feature which is similar to the open-drain configuration.

The PIC32MX250F128B IO pins all allow interrupt-on-change (interrupt generation when there is a change of state on a pin). The AT91SAM3X8E offers this feature as well. However, in addition to this, it also offers interrupt generation on rising edge, falling edge, high level or low level detection. In this regard, the AT91SAM3X8E offers a larger selection of interrupt sources on the IO ports and thus gives greater flexibility.

Additionally, the AT91SAM3X8E offers glitch filtering (rejection of “glitches” lower than one-half of system clock cycle) and debounce filtering (rejection of unwanted pulses from, for example, switch presses) on input pins. These features make the AT91SAM3X8E inputs more powerful as these can be employed to reduce external hardware and accomplish those tasks (such as switch debouncing or glitch rejection) using the in-built hardware.

An interesting and useful feature that the PIC32MX250F128B provides is the support of remappable inputs/outputs. Certain peripherals have input and/or output pins that can be mapped to one of certain remappable pins. For example, the INT4 input pin (for external interrupt) can be mapped to one of eight pins. Similarly the U1TX output pin (TX pin for UART1) can also be mapped to one of eight pins. Which peripheral input/output can be mapped to which pins is provided in the datasheet. This feature of the PIC32MX250F128B allows great flexibility in application development as the microcontroller has a low pin count. So this allows selection of optimal pins for external hardware interface (for example, in PCB design).

The AT91SAM3X8E output pins are not 5V-tolerant when configured as inputs or outputs. The AT91SAM3X8E, on the other hand, can only handle up to a maximum of +0.3V higher than supply voltage (supply voltage is in the range of 3.0V to 3.6V) on input lines. The PIC32MX250F128B has 4 pins, which when configured as open-drain outputs, are 5V-tolerant (can be pulled up, with a resistor, to 5V). The PIC32 reference manual mentions that the PIC32MX250F128B IO pins are 5V tolerant when configured as inputs. However, this is something I found confusing and did not test yet due to the discrepancy between the specifications in the reference manual and the datasheet. The datasheet "Electrical Characteristics" section states that the maximum input tolerable voltage level on an IO pin that is not 5V tolerant is equal to VDD (the absolute maximum is stated as VDD + 0.3V). This means that only 4 pins that are 5V tolerant can withstand up to 5V as inputs. Since the reference manual is written in general for the entire PIC32 series but the PIC32MX250F128B datasheet is written specifically for the device, in this case, I trusted the datasheet and did not proceed to test 5V input tolerance on a "non 5V tolerant" IO pin.

b. **TIMERS**

The AT91SAM3X8E comes with 3 Timer/Counter channels, each of which has 3 32-bit Timers/Counters. So, the AT91SAM3X8E effectively has 9 32-bit Timers/Counters. Besides the quadrature decoder logic (which I have not yet gotten around to using) and the write-protection setting, the three Timers/Counters in one channel are mostly independent.

On the other hand, the PIC32MX250F128B has 5 16-bit Timers (Timers 1, 2, 3, 4 and 5) and a 32-bit Core Timer. However, Timers 2 and 3 can be paired to form a single 32-bit Timer. Similarly, Timers 4 and 5 can be paired to form a single 32-bit Timer. The Core Timer is a 32-bit Timer that is incremented every 2 clock cycles. It does not have a configurable prescaler setting, unlike all other timers on the PIC32MX250F128B and the AT91SAM3X8E. However, it can be enabled and disabled by the user. Additionally, as the Core Timer is a 32-bit Timer, it is possible to make do without a configurable prescaler. Like the other timers, the Core Timer also has a compare module associated with it that can be used to generate an interrupt.

On a related note, when I had been initially trying to get the timers on the AT91SAM3X8E running, I had a difficult time. Even after a few hours, I couldn’t do so. After looking online, I found a piece of code that worked. There was one difference between that and mine. The code posted online had used the analogWrite(pin,value) Arduino function and passed 255 to the argument “value”. Initially, I had mistakenly thought that this statement turned on the DAC and sent a corresponding analog value to the pin. This puzzled me as to why this would make a Timer module work? However, I later found out from Professor Bruce Land that it did not turn on the DAC, but instead turned on PWM. Upon further investigation, I then found the problem. I had then figured out that the clock to the peripherals in the AT91SAM3X8E (the Timers included) is initially disabled. Every peripheral module requires the corresponding clock to be enabled in the Power Management Controller (PMC). The Arduino function analogWrite(pin,value) enables clock to the Timer module when turning on PWM.

The PIC32MX250F128B, on the other hand, has the clock to the peripherals enabled by default and does not allow (as far as I have found out) clock to be disabled to individual select peripherals in a non-power-saving mode.

c. COMPARE MODULES

The PIC32MX250F128B has 5 “Output Compare” modules which are modules that compare the value of a Timer against that in a Compare register. Each of the Compare modules can use either Timer 2 or Timer 3 as a time base if they are configured as individual 16-bit timers. However, if they are configured together as a 32-bit timer, than the Compare modules act as 32-bit comparators against the common 32-bit timer. Upon comparison match, these modules can do a horde of functions such as generating single or continuous pulses, pulse width modulation (PWM) and interrupt generation. The Core Timer has a separate Compare module associated with it that can only be used to generate an interrupt upon match of the Core Timer value against that of the Compare register.

The AT91SAM3X8E offers the “Waveform Operating Mode”. Each Timer offers a Waveform Operating Mode, which can be used for generating 1 or 2 PWM signals with same frequency but independently controllable duty cycles. It can also be used for generating single or continuous/multiple/repeated pulses. There are 3 Compare registers associated with each Timer. These registers can be used to configure outputs on two output pins (for each Timer) – TIOA and TIOB – and also set the period of the Timer. Thus, in total, the AT91SAM3X8E provides 9 “Compare” modules – one on each timer. Furthermore, each compare module provides three Compare registers to control the operation of the Compare module. Each of these Compare registers can generate a Compare event that can trigger an interrupt. This is about equivalent to 18 or 27 PIC32MX250F128B Compare modules if they are all working on the common 32-bit Timer (Timers 2 and 3 combined). The reason I say 18 or 27 is that, in the PIC32MX250F128B the Timers have dedicated Period Registers (PRx) whereas in the AT91SAM3X8E, the TOP of the Timer count is either 0xFFFFFFFF or the value set by the corresponding RC register which acts as the Period Register. So, if RC register is used to set the Period/TOP (as the Period Register), then this is equivalent to 18 PIC32MX250F128B Compare Modules. However, if the Period/TOP is set to the maximum value of 0xFFFFFFFF (which in the PIC32MX250F128B would be setting PRx to 0xFFFFFFFF, if in 32-bit mode), then the RC register is also a compare register. So, then, this is equivalent to 27 PIC32MX250F128B Compare Modules.

The one advantage of the PIC32MX250F128B Compare modules is that, all five modules can be used on a single 16-bit Timer, whereas in the AT91SAM3X8E, only up to three Compare modules can be used on a Timer. Of course then the comparison is between 16-bit Compare modules on the PIC32MX250F128B and 32-bit Compare modules on the AT91SAM3X8E.

d. CAPTURE MODULES

The PIC32MX250F128B has 5 “Input Capture” modules that can save the value on the associated Timer upon an event, such as a rising edge, a falling edge, on every edge or on every specified number of edges (rising/falling/all). The Input Capture modules can use Timer 2 or Timer 3 as the time base if they are configured as 16-bit timers. When configured as a 32-bit Timer, the Input Capture modules act as 32-bit Capture modules saving the value of the 32-bit Timer upon the specified event.

The Core Timer on the PIC32MX250F128B does *not* have any Capture module associated with it.

Each of the Timers on the AT91SAM3X8E has an associated Capture module that can be used to save the value on the associated Timer on two different events onto two different Capture registers. Each event occurs on every edge, every falling edge or every rising edge, as set by the user. Thus the AT91SAM3X8E has a total of 9 Capture modules that can “save” the value of the associated Timer at two events onto two registers. This is about the equivalent of 18 PIC32MX250F128B Capture modules if they are working on a 32-bit Timer (Timers 2 and 3 combined). The advantage to the PIC32MX250F128B Capture Modules is that they can all be used on one Timer if required.

e. **ANALOG TO DIGITAL CONVERTER (ADC)**

The AT91SAM3X8E has a 12-bit analog-to-digital converter (ADC) with a 1MHz conversion rate capacity. With the 84MHz clock on the Arduino Due, I have managed to reach a conversion rate of 660.6kHz. I used a prescaler of 6 for the ADC, as the maximum specified ADC clock frequency is 20MHz: using a prescaler of 4 (the next faster option) gives an ADC clock frequency of 21MHz (which is higher than the specified 20MHz). Thus the ADC clock frequency is 14MHz. Thus the theoretical maximum conversion rate at this clock would be (14/20) * 1MHz = 700kHz (assuming that the maximum conversion rate of the ADC is 1MHz). The conversion rate I achieved (660.6kHz) is slightly lower than this. One other thing I learnt about the ADC module in the AT91SAM3X8E is that, using the 10-bit conversion mode instead of the 12-bit mode does *not* speed up conversion rate.

The PIC32MX250F128B has a 10-bit ADC, also with a specified maximum conversion rate of 1MHz. I have not been able to test how fast I can get with the PIC32MX250F128B. (See section 4 j).

Two very useful features that the AT91SAM3X8E ADC has are that it allows differential-mode analog-to-digital conversion and it has configurable programmable gain (0.5, 1, 2 and 4) for both single-ended and differential-mode conversions. The PIC32MX250F128B does not have differential-mode conversion or programmable gain on conversions.

There is one interesting operational difference between the two ADC modules that I noticed when using them. Assuming that the user did not specify a scan sequence, in the AT91SAM3X8E, the ADC module does the conversions for the channels that are enabled and then loads the results into the corresponding result data registers (ADC_CDRx). So, if channels 0, 6 and 11 were enabled (for example), the ADC converts the analog levels on these channels and then loads them into the registers ADC_CDR0, ADC_CRD6 and ADC_CDR11 as well as the ADC_LCDR for the latest conversion. In the PIC32MX250F128B, all the channels can be enabled, but only the channels defined in the AD1CHS register will be used for analog-to-digital conversion (assuming scan sequence is disabled). So, for the AT91SAM3X8E, to change the channel where conversion is to be done, the channels upon which the previous conversion was done must be disabled and the new required channels have to be enabled. In the

PIC32MX250F128B, all required analog channels can be kept on, while only selecting which channels should be used for conversion. I am not saying that either is a big disadvantage, but that it is just an interesting difference that I noticed.

f. ANALOG COMPARATOR AND INTERNAL VOLTAGE REFERENCE

The PIC32MX250F128B has an on-board analog comparator module that has 3 analog comparators, each of which can be used to compare an external signal against an external or internal voltage reference. The internal voltage reference can be provided from an internal voltage reference section that consists of a 16-tap resistor ladder network with high and low range selection. This allows for selection of a voltage reference from one of 32 voltage levels. Using this internally as the reference to a comparator eliminates the need for an external voltage reference, both reducing external hardware requirements and saving IO pins. Moreover, this voltage reference can also be output on a pin as an analog voltage level, if required for external use. This is interesting because this means that this resistor ladder network can also be used as a "crude" DAC – a 5-bit DAC considering that there are 32 possible levels. I have not yet been able to test how fast I could get this running as a DAC. (See Section 4 j).

The AT91SAM3X8E does not have an on-board analog comparator module or a similar voltage reference module. While the internal comparators and the voltage reference circuitry are definitely nice features to have (on the PIC32MX250F128B) and add to the versatility of the microcontroller, the fast internal ADC module in the AT91SAM3X8E can also be utilized to perform analog comparisons if required (provided all ADC channels are not already being used for analog-to-digital conversion).

g. DIGITAL TO ANALOG CONVERTER (DAC)

The AT91SAM3X8E comes with a digital-to-analog converter (DAC) module that can drive two independent analog outputs. The Digital to Analog Converter Controller (DACC) supports 12-bit resolution digital-to-analog conversion.

I have managed to get the DACC running (at 12-bit resolution) at a conversion rate in excess of 500kHz while generating two analog sine wave outputs simultaneously.

The DAC is definitely a very useful peripheral module to have on-board. While the AT91SAM3X8E provides a powerful fast 12-bit DAC, the PIC32MX250F128B does not have a dedicated DAC module on-board.

h. **INTERRUPT CONTROLLERS**

The PIC32MX250F128B and the AT91SAM3X8E both provide very powerful nested vector interrupt controllers, supporting a large number of interrupt sources and providing a large number of interrupt vectors, along with interrupt priority and subpriority settings (the AT91SAM3X8E also supports "grouping of interrupt priorities" although I have not yet gotten around to this).

For now, I have not gotten to playing around too much with the interrupt controller specifically, but have just used interrupts for some of the other modules.

From personal experience, for me, it was much easier to get the PIC32MX250F128B interrupts running than the AT91SAM3X8E interrupts, which were a great source of headache for a long time. I had failed repeatedly to get any of the AT91SAM3X8E interrupts, besides the timer interrupts, running. However, after a long while, I finally figured out how to configure the interrupts for other interrupt sources.

i. UART

The UART module was something I was interested in just using – not too concerned about performance or speed. The PIC32MX250F128B UART was simple to set up and use. For the AT91SAM3X8E, I just used the provided Arduino library functions.

A useful feature of the Arduino Due board is that it has an on-board serial-to-USB converter so that the UART module can be used to send data to the computer (for data capture, debugging, etc) and to receive data from the computer. The Microstick II board does not come with such an on-board converter. Since the PIC32MX250F128B (and the AT91SAM3X8E too) has an internal USB OTG module, this can be used for communicating with a computer. However, I have not gotten around to using the USB OTG module. So, I had used the UART module to output data to a TX pin which I fed into the Arduino Due RX pin (for one of the UART modules). I used the AT91SAM3X8E to read this input and then transmit this on to the computer over one of its UART modules (through its on-board serial-to-USB converter). The reason I needed a serial-to-USB conversion for communicating with the computer is that, like a lot of computers today, my laptop did not come with a serial port, but only with USB ports.

j. PIC32MX250F128B CURRENT PROBLEM

While using the PIC32MX250F128B, I have come across a problem (this is where I am currently stuck). When I have the PIC32MX250F128B running at a specified oscillator frequency (8MHz for example), I can set the peripheral clock to be running at that same frequency (8MHz for example). However, the time between two instructions is not the time difference I anticipate (125ns for 8MHz, for example). To test how fast the PIC32MX250F128B core is running, I had a counter variable incrementing continuously in the main() loop in the program – ie, in the infinite loop, this was the only statement: incrementing the counter. I also configured Timer 2 such that an interrupt occurs one second after the Timer is turned on – the Timer is turned on right before the counter increment loop starts. In the interrupt service routine, I sent the value of the counter variable over the serial port (used as mentioned above) so that I could check the value of the counter. When the PIC32MX250F128B is running at 8MHz, I expect that the value of the counter would be at least 3 million. However, the counter value was around 400000 (four hundred thousand). This was significantly less than what I anticipated.

I started reading up in the datasheet and searching online to figure out why this was. On the Microchip forums, some of the users had mentioned that the PIC32MX250F128B is initialized with 7 flash wait states, no cache, etc. It was mentioned on the forum that these are not optimal for maximum performance. It was recommended to use the macro SYSTEMConfigPerformance(clock_in_Hz), which configured the flash wait states, cache, peripheral bus clock, etc. for maximum performance at the specified clock.

After doing this, the value of the counter went up from about 400000 (four hundred thousand) to about 700000 (seven hundred thousand). Still this was significantly lower than expected. Thus, I concluded that it is necessary for me to explore the architecture of the PIC32MX250F128B to understand why this is happening and how I would go on to get the PIC32MX250F128B operating at maximum performance. Due to this, I have not been able to measure how fast I can get the ADC and resistor ladder network based "DAC" (see Section 4 f) running.

I will go on to read more about this and understand more about the architecture over the winter break and next semester as I continue to experiment more with the PIC32MX250F128B and the AT91SAM3X8E (on the Arduino Due).

5. RESOURCES AND SUPPORT

The AT91SAM3X8E and the PIC32MX250F128B both have good support communities in the form of support groups and forums online – the Arduino Due support community as well as the AT91SAM support community (http://www.at91.com) being the most notable for the AT91SAM3X8E, and the support community offered on Microchip forums for the PIC32MX250F128B.

The AT91SAM3X8E gains tremendous support from being on the Arduino Due in the form of library functions and routines provided by the Arduino community, as well as hardware shields and boards (along with accompanying software). However, most of the times these all make use of the Arduino library functions. While this is not wrong for application development, this does not allow me to go to the low-levels of the microcontroller to directly access and explore the peripheral modules.

In this regard, the PIC32MX250F128B is easier to work with (to go down to all the low-levels of the microcontroller) as the Microstick II board used is essentially just a programmer/debugger add-on to the microcontroller. Additionally, the library functions and macros provided by Microchip are all low-level functions. I avoided using these as much as possible to understand how to use the peripherals directly by accessing the relevant special function registers (SFRs). The Microchip forums are all very helpful in this regard as there is a significant amount of discussion on how to use the microcontroller at the "low-level".

In terms of learning, the PIC32MX250F128B was a lot easier to pick up than the AT91SAM3X8E. While having previous experience in the PIC series did help (Microchip did maintain a good amount of compatibility with the previous generations of PIC microcontrollers), the PIC32MX250F128B also had significantly better documentation that the AT91SAM3X8E in the form of the PIC32 Reference Manual supplied by Microchip for free. While the AT91SAM3X8E datasheet is written with an attempt to be thorough, it does take much more effort than for the PIC32MX250F128B to understand the workings of a given peripheral module. The Reference Manual for the PIC32 makes this extremely easy, as it clearly lays out most of what is required to get a peripheral module working and how it works.

Beyond manufacturer-provided documentation, the PIC32 also has a few books that one can buy in order to understand and learn how to use it. A quick Amazon book search on “pic32” returns (as of 17th December 2013) at least four books that are written about the PIC32. A similar search on “at91sam” returns no relevant books. A search on “arduino due” returns no books written for the Arduino Due. The Arduino Due does have some tutorials and documentation on the Arduino website, though.

6. FINAL COMPARISON

I have talked about the peripheral modules of both the AT91SAM3X8E and the PIC32MX250F128B above. One thing I did not do was to compare pricing of the two microcontrollers *and* of the microcontroller platforms that I used.

As of 17th December 2013, Mouser lists the unit price of the AT91SAM3X8E as $14.26 and the unit price – if purchased in a quantity of 100 – as $8.09. Similar figures for the PIC32MX250F128B are $4.56 (if one is purchased) and $3.15 (if 100 is purchased).

As of 17th December 2013, the Microstick II is on sale on MicrochipDirect for $34.95. The Arduino Due is on sale at the Arduino Store for $55.56.

From this, it is clear that the AT91SAM3X8E is a much more expensive microcontroller than the PIC32MX250F128B, just as the Arduino Due is more expensive than the Microstick II. A direct implication of this is that the AT91SAM3X8E is a relatively high-end much superior microcontroller than the PIC32MX250F128B.

From my above peripheral module comparisons, this is clear. The AT91SAM3X8E provides a greater number of more powerful peripheral resources than the PIC32MX250F128B. However, this should not undermine the capability of the PIC32MX250F128B microcontroller, which in its own right is a powerful microcontroller with sufficiently powerful peripheral modules.

Comparing the AT91SAM3X8E directly to the PIC32MX250F128B thus seems to be an unfair comparison. The AT91SAM3X8E is significantly superior in terms of processing capability and on-board peripherals.

However, the AT91SAM3X8E is more difficult to learn, understand and get working (at least as far as I have found myself). Moreover, it comes in an unfriendly BGA144 package as compared to the friendlier PDIP28 package of the PIC32MX250F128B. Thus, when a microcontroller (from these two) is to be used for hobbyist or educational purposes, I would recommend using the PIC32MX250F128B. Even though it is not as powerful as the AT91SAM3X8E, it is far more than sufficient for most hobbyist-related tasks or tasks that (that I expect are) to be done in a microcontroller class.

Then comes the Arduino Due, as a platform. The Arduino Due is a powerful and user-friendly platform that is easy to work with when not much detail is given to the microcontroller itself or to low-level implementation, but the use is application-focused (ie, the aim is to just get the job done). In such cases, where the aim is to just get the job done, the Arduino Due is extremely versatile, and I recommend using it in such a case. Such an application could be, for example, in a bioengineering class where a powerful microcontroller (or microcontroller platform) is required, but the attention is given to the task at hand (bioengineering or biomedical application, for example) rather than a focus on the embedded system aspect (ie, on the microcontroller itself, as would be in a microcontroller class for example).

AT91SAM3X8E: If one can learn how to use this microcontroller (which isn't too difficult if sufficient time is spent) and is need of an extremely powerful microcontroller, I recommend the AT91SAM3X8E.

PIC32MX250F128B: For those who will be using a powerful microcontroller for hobbyist purposes or for regular everyday tasks, and for use in an educational institution in a class (for example, in the ECE 4760 course at Cornell University), I highly recommend the PIC32MX250F128B microcontroller.

Arduino Due: When a powerful microcontroller is required for a specific application without much focus on the microcontroller aspect itself, but rather on getting the task done, I recommend the Arduino Due for educational or prototyping purposes. The cost of the platform itself, as compared to the discrete microcontrollers (even the AT91SAM3X8E on which the Arduino Due is based), is prohibitive to using the Arduino Due platform in most commercial applications.

7. CONCLUSION

The AT91SAM3X8E and the PIC32MX250F128B are both powerful microcontrollers with an immense number of peripheral modules. Above, I have talked about some of these peripheral modules that I have worked with. This semester, I have learnt an immense amount through the use of the two microcontroller platforms. This was my first introduction to 32-bit microcontrollers, as well as the ARM Cortex and MIPS architectures.

Having previously worked with several PIC and AVR microcontrollers, I realized that the transition to using the PIC32MX250F128B was not too difficult. However, it was quite challenging to get a grip of the AT91SAM3X8E, especially in the initial stages. Now I feel a lot more confident in both these microcontrollers and should be able to continue exploring more of the peripherals and the microcontrollers more easily than before. I have made my general comparison of the two microcontrollers above.

I will be continuing my Independent Study next semester when I hope to experiment more with the peripheral modules. I also hope to explore the architectures of the two controllers; I have a few things in mind to additionally "play with" (such as using the Arduino Due with Atmel Studio, and expanding on the small Arduino Due based oscilloscope I was designing, for example).