MIE438 – Microprocessors and Embedded Microcontrollers

Laboratory Experiment 6

Lab Guide

Department of Mechanical and Industrial Engineering

University of Toronto

Introduction

In this week's lab, we will be looking at a number of topics related to analog to digital (A/D) conversion, as well as basic digital input, switch de-bouncing, and hysteresis. For this lab, we will examine the analog-to-digital converter (ADC) available on the 68HC12 itself. Although not the top-of-the-line in terms of performance, it is nonetheless useful for a wide variety of applications – not all tasks require extreme resolution or sampling speed.

A/D Conversion and the 68HC12

Like most modern microcontrollers, the 68HCS12 family implements an internal analog to digital converter that one can use without any external parts connected. If you plan to make use of the A to D converter in your project, you may wish to read the ATD_10B8C Guide that is part of the 'Detailed Datasheet' link (9S12DT256.ZIP) on Blackboard. This guide explains the operation of all the control registers, including the process needed to sample from more than one A/D channel.

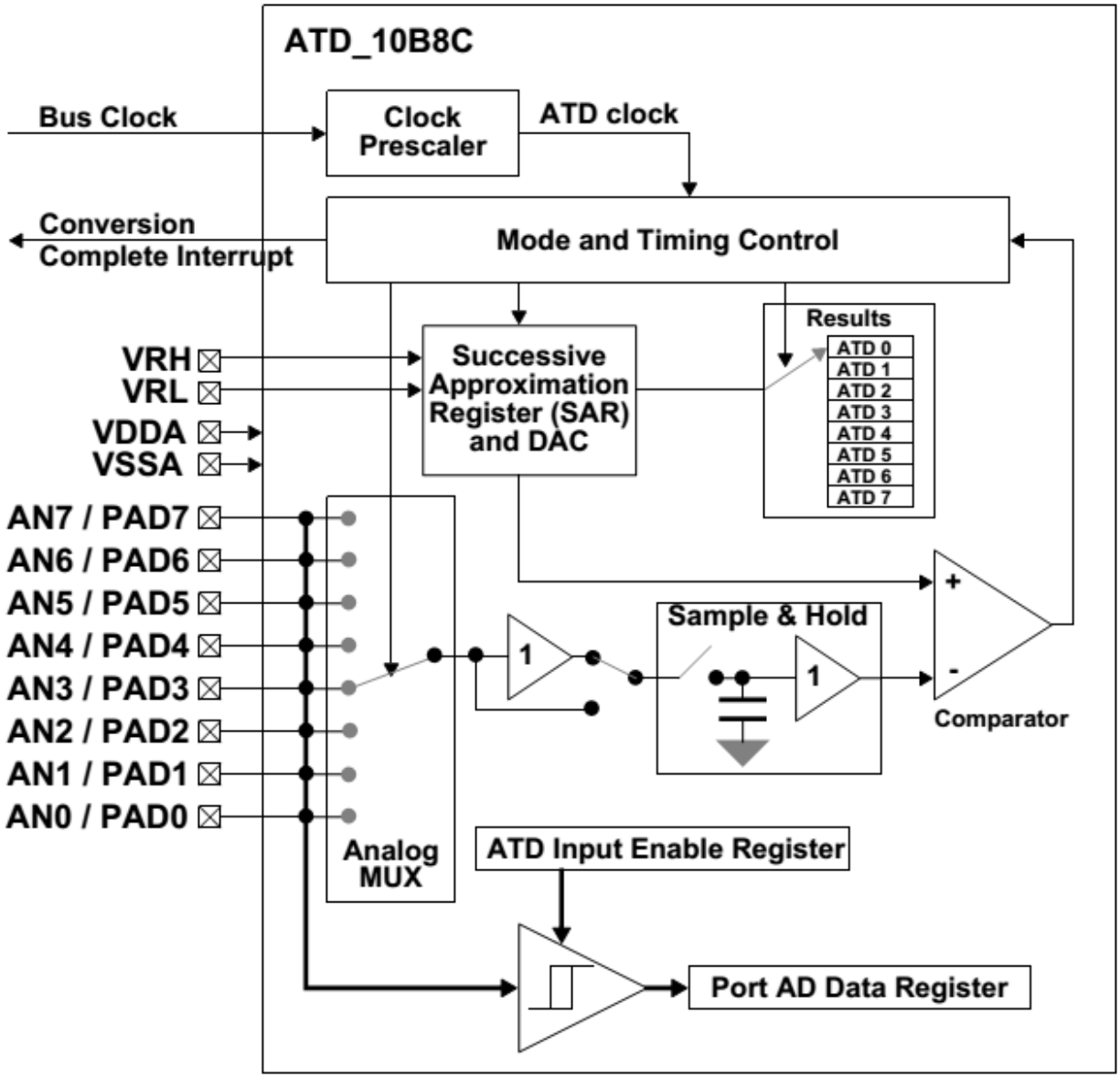
Conversion Basics

The internal ADC for the 68HC12 is a **successive approximation** type. Because of various optimizations, it can achieve a 7 μs sample time, and it is selectable between 8-bit and 10-bit sampling modes. Using lower resolution modes allows for faster conversion times. In addition, there are a total of **eight** analog to digital channels. These can be used flexibly; the 68HC12 allows for the sampling of a single channel or multiple channels through **multiplexing** of the input pins. Note that sampling multiple channels will reduce the effective sample rate per-channel, as mentioned in class. The 68HC12 ADC also has provision for continuous multi-sampling – the successive, rapid collection of multiple samples for a single channel. This mode is valuable; we can collect and **average** multiple close samples of the same signal to reduce electrical and other sources of **noise**, or to implement a simple form of **low-pass filtering**. The topic of multi-sampling will be addressed in class in a later lecture on implementing **control loops**.

Block Diagram

On the following page, a block diagram of the A to D conversion block is shown. Note the similarities to the diagrams shown in lecture for successive approximation. A free running block ('Bus Clock', with pre-scaling) drives a binary up/down counting module called the Successive Approximation Register. The count stored in this register drives a digital to analog converter (DAC). The output of the DAC is constantly compared to the analog value captured by the 'Sample & Hold' block, which we said captures a stable analog value for conversion. When the counting SAR value and sampled analog value are equal, the count from the SAR is 'snapshotted' and stored in one of the result registers, ATD0..7. Finally, one can note the **external power & reference supplies**, which allow us to use a more stable

and low-noise external power source (VDDA/VSSA) and reference voltage (VRH/VRL) for the ADC block. And, one can notice the eight potential analog input pins, AN0..7, which are **multiplexed** (selectively connected) to the input of the ADC. These same pins can be selectively used as regular digital input pins as 'PortAD'.



Quantization Type and Process Details

The 68HC12 implements a modified method of successive approximation which allows it to achieve its (comparatively) rapid sampling rate. In particular, the microcontroller is able to start reading the MSB from the SAR almost immediately after a sample is taken. It is able

to do so because of the way this ADC works; it is a specialized type of successive approximation converter. Recall from the last lecture that we could create an ADC with an advanced search pattern, rather than simply using a simple single or double-slop triangle-wave as a reference. This ADC works by implementing a form of binary search. The process is roughly as follows:

1. The ADC starts with the SAR (successive approximation register) initialized to a value in the middle of the quantization range. The table shown below is for an 8-bit mode conversion, so the middle value is 127 decimal, or [0111 1111] in binary.

2. The digital value in SAR is converted to an analog value using the DAC shown in the diagram. This is checked against the analog signal level from the sample and hold block using a comparator circuit, as we expect for all ADCs.

3. If the output indicated the sample is higher than the DAC output, we **reduce our search range** to the upper half of the old range and set SAR equal to the **midpoint of this new range**. If it is less than the DAC output, we take the lower range and do the same.

4. The ADC can repeat steps 2-4 until the range has lowered to below a single bit of precision, and the conversion is stable and complete.

In Step 3, the output of the comparator tells the ADC what the next search range will be. However, since binary only has two values for a bit, this comparator output is **also** the MSB of the quantized output. To illustrate this, consider the following example conversion of a sampled input of 2.03V.

Lower Limit	SAR	Upper Limit	DAC Output	Input Signal
[0000 0000] = 0	[0111 1111] = 127	[1111 1111] = 255	1.245 V	2.03 V
[0111 1111] = 127	[**1**011 1111] = 191	[1111 1111] = 255	1.873 V	2.03 V
[1011 1111] = 191	[**11**01 1111] = 223	[1111 1111] = 255	2.186 V	2.03 V
[1011 1111] = 191	[**110**0 1111] = 207	[1101 1111] = 223	2.029 V	2.03 V
[1011 1111] = 191	[**1100** 0111] = 199	[1100 1111] = 207	1.951 V	2.03 V
[1100 0111] = 199	[**1100 1**011] = 203	[1100 1111] = 207	1.990 V	2.03 V
[1100 1011] = 203	[**1100 11**01] = 205	[1100 1111] = 207	2.010 V	2.03 V
[1100 1101] = 205	[**1100 111**0] = 206	[1100 1111] = 207	2.020 V	2.03 V
[1100 1110] = 206	[**1100 1111**] = 207	[1100 1111] = 207	2.029 V	2.03 V

For an n-bit resolution, the above process takes **exactly** n clock cycles. This removes one of the biggest issues with naïve successive approximation – variation in conversion time. It is also quite fast; compare this to the 2^n cycles needed to count through the range using a single or dual slope ADC. Finally, if the output is read serially (one bit at a time) starting with the MSB, there would be virtually no delay from sampling to data being available.

There are two important things to notice about the above conversion. First, we can clearly see that at each stage, we constrain the search area by the MSB, meaning it remains the

same for all subsequent steps (as shown by the bold highlighted bits). This is what allows us to begin reading the quantized value almost immediately. If we wanted a parallel output, we would have to wait for the final comparison to finish. Also, you can see that this process is sometimes still inefficient – we tested 207 as a value at comparison four, but we do not know at that point that we can stop. Instead, we arrive back at the same value four steps later.

Positives: Conversion time is always the same, and the quantized value is available immediately. Our maximum sampling rate (assuming clock is constant) has a period of $(4 + 8)t_{clock}$ seconds (t_{clock} is the period of a single clock pulse); eight clock cycles for the data output, and four for setup.

Negatives: A parallel output will be slower, as it will require waiting for a sample to be ready. As we also saw, a dual slope converter could potentially be much faster in obtaining the quantized value in some cases.

The quantized value follows the following formula:

$$Analog\ Value = \left(\frac{Digital\ Value}{255}\right) V_{REF}$$

Notice that this uses the simpler of the two conversion schemes from class, with the downside being increased maximum quantization error. We are now ready to examine the details of the ADC controls.

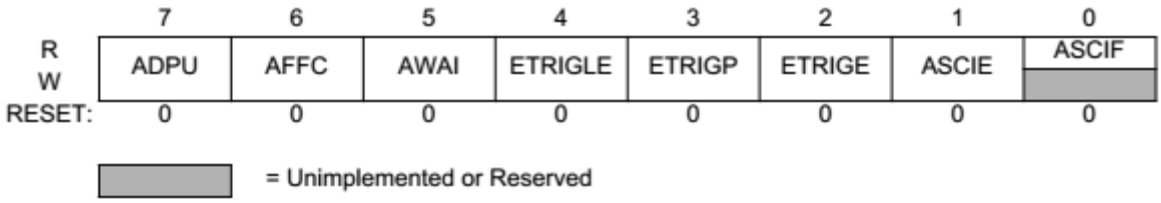
ADC Setup and Use

The ADC on the 68HC12 uses a number of registers. There are **six** primary control registers, ATDCTL0..5, and two status registers. There are also a number of specialized control registers that are used only in certain conversion modes. We will cover only the basics of conversion setup here; all registers are covered in the block guide.

Control Registers

ATDCTL0, **ATDCTL1** – The first two control registers are considered **reserved**; if read in normal operation, they simply return zero. And, if written to they will do nothing. In certain special modes (not supported by all 68HC12 variants), writing to these registers can modify functionality. We will not use them in this lab.

ATDCTL2 – This register controls **power down**, **interrupt**, and **external triggering** functionality. It can be written or read at any time.



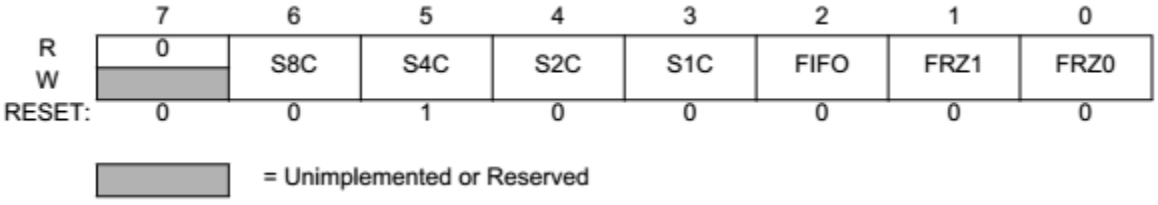
The default value of this register is zero on reset. At a minimum, we need to set ADPU ('ATD Power Up') to one to physically turn on power to the ADC inside the chip. Most other bits will not be touched. The AFFC bit turns on 'fast flag clear,' causing any access to a result register to automatically clear the CCF flag generated by a successful conversion. We will manually clear this flag for better control in most cases. The AWAI bit affect power down behaviour in 'Wait Mode,' a type of power conservation mode – it is only needed for low-power applications, and will not be set here. The ETRIGLE, ETRIGP, and ETRIGE bits affect external triggering, allowing the converter to trigger events on certain comparison levels. Note that while this is similar to the functionality we will write in code for this lab, we will not use this mode yet for simplicity – you may wish to investigate it later. Lastly, ASCIE and ASCIF are used to enable and disable interrupt generation:

ASCIE – ATD Sequence Complete Interrupt Enable; set to '1' to enable an interrupt request whenever a sample is complete. Can be used for selective masking of the ATD source when interrupts are enabled.

ASCIF – ATD Sequence Complete Interrupt Flag; read-only, is set to one when a sample is complete and an interrupt request is pending. Can be used to check the source of an interrupt inside your ISR code.

For basic analog to digital conversion, with no interrupt generation, we only need to turn on power to the ADC block, meaning we will initialize ATDCTL2 = 0x80.

ATDCTL3 – This register controls **sequence length** (number of conversions), FIFO for result registers, and Freeze Mode behaviour.



Of the individual bits, the sequence length bits are most important to us; they determine how many **samples** are captured in one conversion sequence:

S8C	S4C	S2C	S1C	Number of Conversions per Sequence
0	0	0	0	8
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	X	X	X	8

For this lab, we will select only one conversion per sequence. However, if we want to collect either multiple samples from one input, or a single sample from multiple inputs, we will need to set S1C, S2C, S4C, and S8C appropriately.

The remaining bits will not be used in this lab. The FIFO bit, if set to one, changes the ordering of samples in the result registers when multiple samples are taken. FIFO behaviour is complex, and will be omitted from this lab – see the guide if you plan to take multiple samples; typically, the simplest option even with multiple samples is to leave FIFO off. Lastly, FRZ1 and FRZ0 control behaviour of the ADC during freeze mode, a special debugging mode. If both are set to one, the analog to digital conversion process can actually be 'paused' when a debug breakpoint is reached. This could be useful in debugging a fast-running control loop or similar structure by allowing you to actually step through the code without disrupting the A/D process.

For this lab, we will set S1C to one and all other bits zero, enabling single sampling of one input with no other special operation. Thus, ATDCTL3 = 0x08 for this lab.

ATDCTL4 – This register affects the **rate of conversion** through the clock frequency, and is used to select **conversion resolution** and other parameters.

	7	6	5	4	3	2	1	0
R W	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
RESET:	0	0	0	0	0	1	0	1

= Unimplemented or Reserved

In the above, SRES8 selects 8-bit or 10-bit resolution – we will set it to '0' for the default, full-resolution width of 10 bits.

At full resolution, the ADC in the 68HC12 has a maximum clock input of $2\,MHz$ and a minimum of $500\,KHz$. The remaining bits in this register affect sampling rate through adjustment of the apparent sampling clock. We must set them based on the bus clock to achieve an apparent sampling clock within the allowable range.

First, the SMP0 and SMP1 bits affect the conversion length in the second phase. We will set them for fastest conversion, which is SMP0 = SMP1 = 0, resulting in a 2-cycle second phase of conversion.

SMP1	SMP0	Length of 2nd phase of sample time
0	0	2 A/D conversion clock periods
0	1	4 A/D conversion clock periods
1	0	8 A/D conversion clock periods
1	1	16 A/D conversion clock periods

Finally, the bits PRS0..PRS4 control the clock pre-scaling. These bits directly affect how long a single conversion takes, and thus the maximum achievable sampling rate. Since the Dragon12 EVB uses a bus clock of 24 MHz, we will set pre-scaling to 12 (PRS4..PRS0 = 00101), resulting in a $2\,MHz$ apparent sampling clock. This is the maximum allowed by the ADC. If pre-scaling were set higher, a slower apparent sampling clock, and thus a slower sampling rate would result. Thus, ATDCTL4 = 0x05 for this lab.

Prescale Value	Total Divisor Value	Max. Bus Clock[1]	Min. Bus Clock[2]
00000	divide by 2	4 MHz	1 MHz
00001	divide by 4	8 MHz	2 MHz
00010	divide by 6	12 MHz	3 MHz
00011	divide by 8	16 MHz	4 MHz
00100	divide by 10	20 MHz	5 MHz
00101	divide by 12	24 MHz	6 MHz
00110	divide by 14	28 MHz	7 MHz
00111	divide by 16	32 MHz	8 MHz
01000	divide by 18	36 MHz	9 MHz
01001	divide by 20	40 MHz	10 MHz
• • •	• • •	• • •	• • •

ATDCTL5 – The final control register selects the conversion sequence type and any multiplexing effects.

	7	6	5	4	3	2	1	0
R	DJM	DSGN	SCAN	MULT	0	CC	CB	CA
W								
RESET:	0	0	0	0	0	0	0	0

In the above, the bits DJM (justification) and DSGN (signed output) affect the format of the output, in combination with SRES8 from the previous control register:

SRES8	DJM	DSGN	Result Data Formats Description and Bus Bit Mapping
1	0	0	8-bit / left justified / unsigned - bits 8-15
1	0	1	8-bit / left justified / signed - bits 8-15
1	1	X	8-bit / right justified / unsigned - bits 0-7
0	0	0	10-bit / left justified / unsigned - bits 6-15
0	0	1	10-bit / left justified / signed - bits 6-15
0	1	X	10-bit / right justified / unsigned - bits 0-9

We will be using 10-bit conversion that is **unsigned** (DJM = 1). Justification refers to the ordering of bits in the output (result) registers – recall that this is a 10-bit value being stored in a 16-bit register. We can control if the leftmost bits (0-9) or the rightmost (6-15) are used. The other bits are zero. We will typically select right justification (DSGN = 0), giving us a decimal range of values of 0 to 1023 for 10-bit resolution.

The SCAN bit controls if samples are taken continuously, or one at a time. We will control when samples are taken in this lab, so SCAN = 0 (single-sample mode) here.

The MULT bit controls if samples are taken from one channel (MULT = 0) or across multiple channels (MULT = 1). When MULT = 0, the ADC samples only from one specified input channel, selected by the bits CC/CB/CA. If multiple channels are sampled (MULT = 1), these bits choose which channel to start with – sampling happens in sequence, wrapping around with channel AN7.

CC	CB	CA	Analog Input Channel
0	0	0	AN0
0	0	1	AN1
0	1	0	AN2
0	1	1	AN3
1	0	0	AN4
1	0	1	AN5
1	1	0	AN6
1	1	1	AN7

Since we are sampling only a single channel in this lab, we will leave CC = CB = CA = 0, selecting channel one, and thus pin AN0/PAD00 as the analog input. Thus, ATDCTL5 = 0x80. In this case, a single sample will be taken per conversion process. This sample will be placed in the first result register, ATD0DR0.

One final yet important point is that writing to a number of the control registers will **reset** the ADC process. This register in particular both resets **and** starts a new conversion. We will use this as our method of initiating an ADC sample in code. Thus, we now have the basic tools needed to perform analog to digital conversion using the 68HC12.

The lab will proceed as follows:

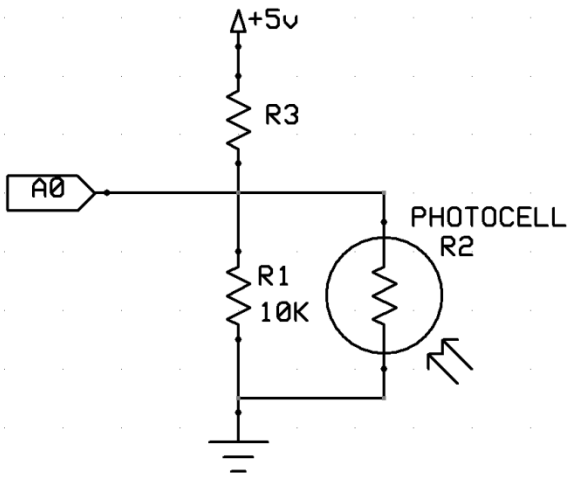
Part I – You will connect a light-sensitive resistor to the A/D input and visualize the sensed light level and quantization process using the LEDs as a bar graph. Later, you will try making a light-controlled switch using the sensor.

Part II – You will investigate methods of dealing with noise in the sensed signal; you will try implementing a digital version of the **multivibrator** circuit, and **hysteresis** circuit.

It is strongly recommended that you read over the lecture slides for these two topics as well, so that you have some initial ideas about how to implement them in software.

Lab – Part 1: Basic A/D Conversion

Step 1. To begin, we will need to construct a simple sensor circuit to provide a source of interesting data for the ADC to read. We will use a photocell (a light-varying resistance) to sense light level. Construct the circuit below:



Make all connections and changes to the circuit **with the power disconnected**. Both +5V and ground connections are available on the bottom pin header, near the 'RESET' button. The connection 'A0' on the above diagram should be connected to **PAD00**, located on the right-hand pin header, near the keypad.

Note that the value for R3 should be decided based on the ambient light level. The nominal value during testing was around $10K\Omega$ – do not select R3 below ~$1\,K\Omega$. Before connecting the circuit to PAD00, you should use the oscilloscope or a multimeter to verify that voltage at A0 does vary with light level. If R3 is too low, it will stay near 5V at all times. If it is too high, it will stay near zero. The TA will have several values of resistors available to try.

Step 2. Once you have the circuit connected, open the 'Lab6_1.mcp' project and read through 'main.c'. Notice that the code is set up as a polling loop; it will continually request

new samples from the ADC. The results of the conversion are presented back to you in two forms. The first is a bar graph using the LEDs connected to Port B. The second is the buzzer: a simple speaker is connected to PortT at Pin 5. When this pin is brought from low to high **or** from high to low, the speaker emits a short 'tick' sound. If we continually switched from high to low we could create different frequencies of sound, but in this lab we actually only want to hear when a transition occurs.

Step 3. Before working on the code, verify that your circuit does produce a varying output voltage which (depending on light level – use your hand to cover the sensor) does go above **and** below 2.5 V.

Next, connect the circuit to PAD00. In the space highlighted in code, add the **initialization code** needed to operate the A/D converter. You will need to initialize ATD0CTL2, ATD0CTL3, and ATD0CTL4 to the correct values outside the 'for' loop. And, inside the 'for' loop, you will need to set ATD0CTL5 to generate a new sample. Note that you can either set each register directly, as in:

ATD0CTL3 = <Value>

Or, you can set individual bits in the register:

ATD0CTL4_SRES8 = <1 or 0>

In the latter case, unmodified bits will be left at their 'reset' value indicated in the tables in the previous section. Write your completed code for the routine in the available space for **Question 1**.

Step 4. Download and test your code. If everything is working correctly, you should be able to move your hand above the sensor and see the bar graph on Port B change. The range should go from most of the LEDs off to most of them on. At a minimum, it must cross PB3, otherwise you have not reached the threshold of the 'if' statement in the main loop. In this case, adjust the value of R3 to fix the issue.

Step 5. Start with your hand completely covering the light sensor. Raise your hand away from the sensor quickly. Remember that the speaker on the board will produce a short 'tick' sound when the threshold is crossed, and the LED bar graph will change as well.

Step 6. Repeat the previous step, except move your hand very slowly, and pause as close to the crossover point as you can. Answer **Question 2** to report your findings.

Lab – Part 2: 1-bit Conversion Techniques

Step 7. To deal with issues with threshold-based comparisons, we can apply the techniques that we used for **1-bit A/D conversion**. In particular, we will try to apply both the monostable multivibrator technique, and the hysteresis technique.

First, modify the code for the main loop to implement the concept of a **monostable multivibrator**. That is:

1. Start the 'for' loop as normal. Keep taking samples and checking them against the threshold.
2. When you detect a transition from [Input < Threshold] to [Input > Threshold], disable the comparison. In other words, keep the output in one state for a **short but fixed time time**. For example, you can disable the comparison check for n loop cycles.

Note that you do not need to use the timer to do the above; you can simply add code to ignore the comparison for a **fixed** number of ADC samples.

Record your code for **Question 3**. Experiment with a few different values for the delay time, and then answer **Question 4**.

Step 8. Now we will try hysteresis. To begin, you will need to determine suitable low and high reference levels. To do so, first cover the sensor completely. Look at the bar graph code, and see what range of values the ADC is outputting, based on the bar graph display on PortB. Pick the lower reference to be slightly above this range, say +20 to +30 above.

Now, completely uncover the sensor and repeat the process, this time picking the high reference to be slightly below the level of output when the sensor is fully uncovered. Lastly, modify the 'if' statement to implement hysteresis:

1. If output is currently **high** (closer to 1023), compare the input to the **lower** reference level (closer to 0).
2. Otherwise, if the output is currently **low** (closer to 0), compare the input to the **higher** reference level (closer to 1023).
3. Whenever the output changes, be sure to change the reference used for comparison.

Record your code for **Question 5**, and answer **Question 6**.

Congratulations, you are now finished with Lab 6!

MIE438 – Microprocessors and Embedded Microcontrollers

Laboratory Experiment 6

Deliverable

Department of Mechanical and Industrial Engineering

University of Toronto

Last Name, First Name	Student Number

1. Record your code changes here.

2. What happens to the buzzer when your hand is near the cross-over point? What is happening in our A/D conversion process / program code, and why (i.e., what does the sound you hear represent)?

3. Monostable code listing (**just the code with the 'for' loop is sufficient**).

4. Has this solution resolved the issue found in Question 2, or has it merely reduced its effect? Speculate what situation or problem would this solution be more useful for.

5. Hysteresis code listing:

6. Why did adding hysteresis solve this problem? From the lecture, what other solutions might have worked?

This lab hand-in is due at the start of the subsequent lab section.