MPSD

A/D Lab Exercise

Analog-to-Digital Converter

Student's name & ID (1): _____

Partner's name & ID (2): _____

Your Section number & TA's name _____

Notes:

You must work on this assignment with your partner.

Hand in a printer copy of your software listings for the team.
Hand in a neat copy of your circuit schematics for the team.

These will be returned to you so that they may be used for reference.

---------- do not write below this line ----------

	POINTS (1)	POINTS (2)	TA init.
Grade for performance verification (50% max.)			
Grade for answers to TA's questions (20% max.)			
Grade for documentation and appearance (30% max.)			
			TOTAL

Grader's signature: _____

Date: _____

Analog-to-Digital Converter

GOAL

By doing this lab assignment, you will learn to use:

1. The Analog-to-Digital converter.
2. The basics of polling.

PREPARATION

- Read Sections 12.1 to 12.9 from *Software and Hardware Engineering* by Cady & Sibigtroth. (Read Section 7.11 from *Microcomputer Engineering* by Gene H. Miller.)
- Write a C program that is free from syntax errors (i.e., it should compile without error messages and produce a running .S19 file).

1. INTRODUCTION TO THE A-to-D CONVERTER

The A-to-D converter (ADC) uses successive approximation to produce an 8-bit unsigned number (0-255) to represent the analog voltage applied to one of the ADC input pins. This number, ADResult, represents the input voltage V_{ADC} in terms of two reference voltages, V_{RH} (high reference voltage) and V_{RL} (low reference voltage). The ADResult can be computed using this expression.

$$\text{ADResult} = round\left[255 \cdot \left(\frac{V_{ADC} - V_{RL}}{V_{RH} - V_{RL}}\right)\right] \text{ for } V_{RL} \le V_{ADC} \le V_{RH}$$

$$\text{with } V_{RH} \le 6\text{V},\ V_{RL} \ge 0\text{V},\ \text{and } V_{RH} - V_{RL} \ge 2.5\text{V}$$

The ADC converts analog voltages applied to Port AD pins 0 through 7 (PAD0, PAD1, …, PAD7). The following paragraphs explain how the ADC is operated through the use of the internal registers.

TURNING THE ADC ON/OFF. The ATDCTL2 register (address $0062) uses bit 7 (ADPU) to control the power to the ADC. ADPU = A to D Power Up (0 = ADC power off, 1 = ADC power on). Upon RESET, the ATDCTL2 register is set to $00. That is, ADPU = 0 and the ADC power is off (to save energy).

	7	6	5	4	3	2	1	0	
$0062	ADPU	AFFC	AWAI	0	0	0	ASCIE	ASCIF	ATDCTL2
RESET =	0	0	0	0	0	0	0	0	

{The D-Bug12 monitor puts $00 into the ATDCTL2 register as part of its RESET initialization. That is, D-Bug12 turns the ADC power off.} If you want to control ADC power, you can write a 1 or a 0 to ADPU (ATDCTL2 bit 7).

ATD CLOCK FREQUENCY. The P-clock must be scaled to keep the ADT clock within the required frequency limits of from 500 KHz to 2 MHz. The prescale value is specified in the ATDCTL4 register (address $0064). The 68HC12 EVBs have P-clock frequencies of 8 MHz so the default scale factor of 4 allows the converter to operate at its fastest rate.

	7	6	5	4	3	2	1	0	
$0064	0	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0	ATDCTL4
RESET =	0	0	0	0	0	0	0	1	

PRS4	PRS3	PRS2	PRS1	PRS0	P-clock Divisor
0	0	0	0	0	2
0	0	0	0	1	4
0	0	0	1	0	6
0	0	0	1	1	8
0	0	1	0	0	10
0	0	1	0	1	12
0	0	1	1	0	14
0	0	1	1	1	16
0	1	x	x	x	Do not use
1	x	x	x	x	Do not use

CONFIGURING THE ADC. The ADC can convert analog voltages from any of eight channels, Port AD input pins (PAD0, PAD1, …, PAD7). Each conversion cycle consists of four or eight conversions with the results put into four or eight of the A/D Result registers: ADR0H (address $0070), ADR1H (address $0072), . . . ADR7H (address $007E). The user selects, via software settings, which pin (PEn, n = 0, …, 7) or group of pins (PE0:3 or PE4:7) are converted.

The ATDCTL5 register (address $0065) is the A/D Control/Status Register. The user sets three parameters and indicates the channels to be used in the ADCTL register to configure the ADC:

- S8CM = 0 conversion sequence consists of four conversions
  S8CM = 1 perform eight conversions in the conversion sequence
- SCAN = 0 perform a single conversion cycle and stop
  SCAN = 1 perform continuous conversions without stopping after each cycle
- MULT = 0 convert a single channel
  MULT = 1 convert a four or eight channel group
- CD – CA = Channel Select D through A

	7	6	5	4	3	2	1	0	
$0065	0	S8CM	SCAN	MULT	CD	CC	CB	CA	ATDCTL5
RESET =	0	0	0	0	0	0	0	0	

When MULT = 0, set CD – CA for the desired channel. Each single channel is converted four or eight times per cycle (depending on S8CM). The four or eight conversion results are put into the A/D Result registers in sequence: ADR0, ADR1, ADR2, and ADR3 or ADR0, ADR1, ADR2, ADR3, … ADR7.

CD	CC	CB	CA	Channel
0	0	0	0	0
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

When MULT = 1 and S8CM = 0, set CD and CC to select either of two groups of four channels (0:3 or 4:7). Each channel in the group is converted once and the result put into one of the A/D Result registers as given in the table. With S8CN = 1, all 8 channels are sampled once and the values put into each corresponding result register.

CD	CC	CB	CA	Channel	Result Register
0	0	X	X	0	ADR0
0	0	X	X	1	ADR1
0	0	X	X	2	ADR2
0	0	X	X	3	ADR3
0	1	X	X	4	ADR0
0	1	X	X	5	ADR1
0	1	X	X	6	ADR2
0	1	X	X	7	ADR3

2. POLLING

An A/D conversion cycle starts whenever any value is written to the ATDCTL5 register. When SCAN =

0, the program must poll (test) the CCFn (Conversion Complete Flag) bits in ATDSTAT. These bits are reset (CCFn = 0) when the conversion cycle is started. The bits are set (CCFn = 1) when the results of the four or eight conversions have been written to the four or eight A/D Result registers. Since a single conversion takes from 72 to 128 clock cycles (9 to 16 μs), the CCFn is set 72 to 128 clock cycles after writing to the ATDSTAT to start the process. The user may use this time to perform some simple routine, such as average the results. In any case, the user must test each CCFn to know when the data in the corresponding A/D Result registers are valid. Alternatively, the SCF (sequence complete flag), bit 7 in the ATDSTAT ($0066) register may be used to indicate that all four or eight values are available from the four or eight conversions. Normally the CCFn flags are cleared when the associated result registers are read (AFFC = 0 in ATDCTL2).

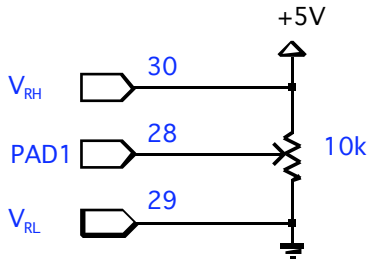
When SCAN = 1 (continuous conversion mode), each of the A/D Result registers is updated in a circular fashion: ADR0, ADR1, ADR2, ADR3, ADR0, etc. Thus a new result is available every 72 to 128 clock cycles.

3. PROGRAMMING ASSIGNMENT

For this lab you will write your program with a circuit on a prototyping board. This builds on what you have learned so far.

This lab uses the EVB and a protoboard. You will need to wire up an LED to any available output port pin and a push button switch to Port AD pin 7 for input to pull the pin low when pushed. The TAs will caution you about the power supply and its connections. NEVER WORK WITH THE POWER ON!

THE PROTOBOARD. On the protoboard you will find PAD1 on pin 28 of the 60-pin J9 connection block. You will need to connect +5 V to V_{RH}, pin 30, and ground V_{RL}, pin 29. To obtain an analog voltage to convert, use a potentiometer (10 kΩ). Connect the potentiometer between +5 V and ground. Connect the wiper to PAD1. Only when you are sure you have no shorts should you turn on the power supply. PAD7 is pin 22 on J9.



PART I – Simple Voltmeter

Write an analog-to-digital conversion program to operate the EVB as a digital voltmeter. Display the

hexadecimal result and the decimal voltage value on the terminal display.

Some of what you will need to do:

- Write a main program that calls the necessary routines to implement your program design.
- Write a routine to configure the ADC to convert the single analog voltage on Port AD pin 1 (PAD1). Operate in 8 conversions and stop mode. That is, S8CM = 1, SCAN = 0, MULT = 0, and CD–CA = 0001. Conversion should start when bit 7 on Port AD is pulled low. The program should poll this bit waiting for the signal.
- Write a routine to poll the CCFn bits or the SCF bit and average the eight results.
- Write a routine to display the low, high, and average result byte in hexadecimal on the display as well as the decimal value of the **voltage** to 6 decimal places and update the display value whenever bit 7 on Port AD is pulled low.
- Check your results with a voltmeter and hand calculate a few voltage readings to verify that your program is working correctly.

PART II – Voltage Comparator

Configure an available parallel port for output to light an LED. Make sure a current limiting resistor is included. A/D channel 1 will be connected to the potentiometer as in Part I and A/D channel 0 will be connected to a signal generator set to a low frequency (~0.5 Hz) with a D.C. offset and amplitude adjusted to keep the output within the 0 to +5 V limit of the converter.

Sample at the highest frequency possible but still use the longest sampling aperture achievable (16 ATD clock periods). Be prepared to explain the purpose of the programmable sample time aperture in your report. Use an A/D interrupt ISR rather than polling to obtain samples when they are available.

Program the A/D converter to sample channels 0 and 1 continuously. If the difference between the 2 channels is within ±2 bits, light the LED for approximately 0.2 seconds (may be done with a countdown delay loop). Verify operation by adjusting the potentiometer voltage and signal generator output while observing both signals on an oscilloscope.

PART III – Advanced Voltmeter and Frequency Counter

This program will use some of the self-testing features of the A/D converter as well as processor mathematic operations to calculate some values for an input waveform. After initialization of the A/D, perform a set of eight reading on each of the 0 V, +2.5 V, and +5 V settings using the internal checking ability of the converter. For each set of eight samples, compare the low, high, and average to the expected values and warn the operator about any possible malfunctions.

Next, the normal operation of the program should continuously sample a waveform input on A/D channel

0 with the shortest sampling aperture window (2 ATD clock periods) and display the minimum, maximum, and a running weighted average based on $V'_{avg} = \frac{V_{in} + 127 \cdot V_{avg}}{128}$, or weighting the current input with $\frac{1}{128}$ and the previous average with $\frac{127}{128}$. (Make sure you do these calculations using 16 bit numbers.) The display should be updated approximately every second. A clear pushbutton should be provided to reset the stored minimum and maximum readings.

Assuming the maximum frequency of the input waveforms are much below half the sampling frequency of the processor system, calculate the frequency of the waveform by measuring the time between repeat patterns on the signal (i.e. peak to peak) for sine and triangle waves. This can be done by counting the number of samples in one period and multiplying the number by the sample period or using timers. Test your program on frequencies in the range of ~0.3 Hz to 30 kHz and document it's useable range. Better accuracy on lower frequency signals may be achieved by adjusting the timer and A/D converter clock frequencies and redoing the measurements. Display the frequency in Hz on the console, but do not update it more frequently than about once a second. Use an oscilloscope to verify measurements of each parameter.

Optional Feature: Have the program calculate the A.C. RMS value of the waveform. This is $\sqrt{\overline{(V - V_{avg})^2}}$ where V_{avg} is the running weighted average. The $\overline{(V - V_{avg})^2}$ mean should be calculated using some form of integration over one period.

Good programmer's tip. Design the program top-down. Then write the routines bottom-up. Write them one at a time and thoroughly test each one before integrating them. This way you will have isolated any errors to the routine that you are currently writing. Good programmers follow this method.

NOTE:

There are typographical errors in the Introl version 4.0 C compiler include file HC912B32.H. The A/D registers are named _H12ADTxxxx instead of _H12ATDxxxx as seen in the segment from the file below.

```
__BYTEreg__ _H12ADTCTL2;   // ATD Control 2
__BYTEreg__ _H12ADTCTL3;   // ATD Control 3
__BYTEreg__ _H12ADTCTL4;   // ATD Control 4
__BYTEreg__ _H12ADTCTL5;   // ATD Control 5
__WORDreg__ _H12ADTSTAT;   // ATD Status
```

There is another bug in the DB12->printf() function you will find related to outputting floating point numbers. These values do not display correctly. There are many simple work-arounds that are worth noting since many smaller microcontrollers do not support floating point arithmetic, but many applications use scaled integer calculations to perform the necessary control functions. For display and debugging purposes it is sometimes necessary to display floating point values without the use of floating point I/O routines. The following short program demonstrates how the decimal digits can be extracted and displayed using only integers.

```
/*****************************************************************/
/* Test program to print a floating point number using integers */
/*                                                               */
/* This gets around the floating point printing problem with    */
/* the Introl/D-Bug12 compiler                                   */
/*                                                               */
/* NOTE: this can't run on the HC12 due to the printf problems  */
/*****************************************************************/

main()
{
    float x,y,s;
    char i, f[7];

    x=1.234567;
    s=2.0;

    y=x=x*s;          //scale x value and save copy in y

    for(i=0; i<7; i++)
    {
        f[i]=(int)y;
        y=(y-f[i])*10.;
    }

/* f[0] is the integer part of x, f[i] (i=1,2,3...) are the    */
/* decimal digits (10^-i values)                               */
    printf("x  = %9.6f  = %2d.%1d%1d%1d%1d%1d%1d \n\r", x, f[0],
        f[1], f[2], f[3], f[4], f[5], f[6]);

}
```