

Analog Conversion and MAC

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 printed 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 -----

Grade for performance verification (50% max.)

Part 1 (13% max.)

Part 3 (7% max.)

Part 4 (30% max., 15% max. for non-MAC solution)

Grade for answers to TA's questions (20% max.)

Enhancement (5% max.)

Grade for documentation and appearance (25% max.)

POINTS
(1) (2) TA init.

		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.
3. The Digital-to-Analog converter.
4. The Multiply and Accumulate (MAC) engine.

PREPARATION

- References: C8051F12x-13x.pdf 8051 *C8051F12X Reference Manual, Ch. 5, 8, 9, 12*
- Write a C program that is free from syntax errors (i.e., it should assemble without error messages).

A-to-D CONVERTER

1. Introduction to the A-to-D Converter

The A-to-D converter (ADC) uses successive approximation to produce an 8, 10, or 12-bit signed or unsigned numbers 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 for $n = 8, 10, \text{ or } 12$ using this expression:

$$ADResult = \text{floor} \left[2^n \cdot \text{Gain} \cdot \left(\frac{V_{ADC} - V_{RL}}{V_{RH} - V_{RL}} \right) \right] \text{ for } V_{RL} \leq V_{ADC} \leq V_{RH}$$

with $V_{RH} \leq 3.3V$, $V_{RL} = 0V$ in single-ended mode and

with $V_{RH} \leq 3.3V$, $V_{RL} = -V_{RH}$ in differential mode

$\text{floor}[\cdot]$ indicates rounding down (truncating) to next lower integer

The ADC converts analog voltages applied to AIN0 pins 0 through 7 (AIN0.0, AIN0.1, ..., AIN0.7) in single-ended mode and paired up (AIN0.0/AIN0.1, AIN0.2/AIN0.3, ...) in differential mode. Two of these pins are located on J20 (pin 5 & 6 for AIN0.0 & AIN0.1) and two more on J11. In order for the ADC to operate properly, the reference for ground and the high voltage will need to be connected. These are located on the J20 with pin 7 being VREF0, and pin 8 being analog ground. **WARNING:** when an external reference is supplied, the x2 reference buffer (REFBE) **MUST** be disabled. A gain can also be set for amplification if necessary. The following paragraphs explain how the ADC is operated through the use of the internal registers.

CONFIGURING ADC. In order to configure ADC0 for 12-bit conversions, the following steps need to be taken:

1. Make sure ADC0_PAGE is selected.

2. Set the REF0CN register to use VREF0 as the voltage reference (AD0VRS), and turn on the internal bias generator (BIASE) as well as the internal reference buffer (REFBE). This will set VREF0 to 2.4 Volts. A 0.1 μ F and 4.7 μ F bypass capacitors is needed between VREF & AGND and they are included on the development board (see Figure 9.1 in the Reference Manual).
3. Set AMX0SL to the pin or pins on which you wish to perform A/D conversions.
4. Enable ADC0 through the ADC0CN register.

These steps will enable A/D conversions in polled mode. The 8051 also supports interrupts for ADC, which can be added as an enhancement to this lab.

2. Polling

After selecting 00 on the AD0CM multiplexer and clearing AD0INT, an A/D conversion cycle starts whenever a 1 is written to the AD0BUSY bit. The AD0INT bit will remain 0 until a conversion is completed, at which point it will be set to 1. In order to properly operate in polled mode, the program must wait for the AD0INT bit to be set to 1 before reading the A/D conversion value. After a conversion has completed, the resulting value can be read from the ADC0H and ADC0L registers, which are the high and low bytes respectively. AD0JST is used to left or right justify the 12 bits in the 16-bit register, but this must be set before the conversion.

3. 16-Bit Values from 8-bit Registers

An efficient and convenient way to convert the two 8-bit register values into a 16-bit short integer is needed that does not require arithmetic operations. One of the easiest ways to do this is to define variables that use the same memory address with one being a short integer and the others being two unsigned characters that are the low and high byte of the short integer. When needing the 8-bit values, use the char names and when needing the 16-bit value, use the short int name. The short int may be signed or unsigned, depending on the mode: differential or single-ended. Below are two possibilities.

```

unsigned short int analogval;
unsigned char *analoghi, *analoglo;
analoglo = (char *) &analogval; // *analoglo set to the lo byte of analogval
analoghi = analoglo + 1;        // *analoghi set to the hi byte of analogval
. . .
*analoglo = 0x55;
*analoghi = 0xAA;                // analogval will equal 0xAA55

unsigned short int *analogval;
unsigned char analogbyt[2];      // analogbyt[0] is lo, analogbyt[1] is hi
analogval = (int *)analogbyt;    // now *analogval is 16-bit combination

```

D-to-A CONVERTER

1. Introduction to the D-to-A Converter

The C8051F120 features two Digital to Analog Converters (DAC). DAC0 and DAC1 are both 12-bit DACs with an output ranging from 0V to VREF0. The DAC will convert a digital 12-bit value to an analog signal. The same equation given above with $n = 12$ relates the digital value to the analog voltage.

CONFIGURING DAC. In order to configure DAC0 for 12-bit conversions, the following steps need to be taken:

1. Make sure DAC0_PAGE is selected.
2. Configure REF0CN and enable the internal bias generator. If you have already enabled this for ADC, you do not need to do it again.
3. Enable DAC0 through the DAC0CN register.

2. Performing a Conversion

In order to perform a conversion, the DAC0H and DAC0L registers will need to be written to. These represent the high byte and low byte of the 12-bit number respectively. Once written, the analog output will be available on the DAC0 pin (pin 3 on the J11 connector). Once again the software will control when conversions occur through the mode bits in DAC0CN (DAC0MD1 & DAC0MD0). These bits should be set to convert on write to the high data byte rather than using any of the timers. Also note that the data format bits (DAC0DF2-0) in DAC0CN must indicate the proper 12 bits out of the 16-bit value that will be used by the 12-bit converter to create an analog voltage.

MULTIPLY AND ACCUMULATE

Introduction to the MAC

The Multiply and Accumulate (MAC) engine is an extremely fast and powerful hardware math accelerator for digital signal processing calculations. The MAC is the basis for DSP algorithms such as the FFT, and can be used to implement fast IIR and FIR filters. The MAC can multiply and add two 16-bit numbers (integer or float). In order to achieve this, the following steps need to be taken.

1. Set the MAC0CF register to clear the accumulator, perform multiply & accumulate, and set the mode for integer or fractional multiplication.
2. Set MAC0AH and MAC0AL, which are the high and low bytes of the first number.
3. Set MAC0BH and MAC0BL, which are the high and low bytes of the second number.

Upon writing to the MAC0BL register, the MAC is automatically performs the multiplication and addition. The entire process takes two system clock cycles.

PROGRAMMING TASKS

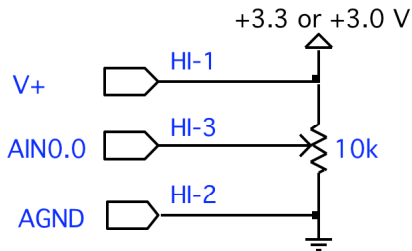
PART I – Simple Voltmeter

For this lab you will write your program for use 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 a push button switch to P1.0 for input to pull the pin low when pushed. The TAs will caution you about the power supply and its connections. NEVER WORK ON CIRCUIT WIRING WITH THE POWER ON!

THE PROTOBOARD. You will use 2.4 Volt VREF0, and connect AGND to ground as mentioned

earlier. To obtain an analog voltage to be used as the input to converter, use a potentiometer (10 k Ω). Connect the potentiometer between +3.3 V and AGND. Connect the wiper to AIN0.0. Only when you are sure you have no shorts should you turn on the power supply. If +3.3V is unavailable on the 8051 board, there are DC power supplies at most of the lab stations that will output an appropriate voltage. Dialing in 3.00 V will provide a safe voltage level that won't damage the converter. Make sure that the 10 - 20V button to the left of the dial and display is NOT pressed in. The 10 k potentiometer is one of 4 in the parts kit, mounted on a circuit board. Refer back to Lab Exercise 1 for the circuit schematic and pinouts.



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 AIN0.0 using single-ended mode on the input MUX. Conversion should start when P1.0 is pulled low. The program should poll this pin waiting for the signal.
- Write a routine to poll ADC0 until a conversion is complete.
- Write a routine to display the low, high, and average value in hexadecimal on the display as well as the decimal value of the **voltage** to 6 decimal places and update the display value whenever P1.0 is pulled low. This is an extension of the second routine that keeps track of the low and high values and maintains a running average (e.g., the average of last 16 or so samples).
- Check your results with a voltmeter and hand calculate a few voltage readings to verify that your program is working correctly. Note the maximum voltage this setup is able to convert correctly relative to the 3.3 Volt or 3.0 Volt potentiometer supply and the 2.4 Volt reference.

PART II – Advanced Voltmeter and Frequency Counter **{THIS PART IS OPTIONAL, but heed the VREF warning}**

THE PROTOBOARD. You will need to connect +3.0 V to VREF0 and AGND to ground (J20, pins 7 & 8, PWR LED end of connector), as mentioned earlier. The difference is that an external 3.0 Volt DC power supply will be used for the reference voltage rather than the internal band-gap generated 2.4 Volts and the REF0CN configuration REFBE bit must be changed to accommodate this. See section 9.3 in the Technical Reference manual. **Make sure the internal reference buffer has been disabled before**

applying power to VREF. The specification states that VREF must be 0.3 V below the +3.3 V supply voltage to the 8051, so 3.0 V is the maximum allowed value. To obtain an analog voltage to be used as the input to converter, use a potentiometer (10 k Ω) as in Part I. Connect the potentiometer between +3.0 V and ground. Connect the wiper to AIN0.0. Only when you are sure you have no shorts should you turn on the power supply.

This program will use some of the alternative features of the A/D converter as well as processor arithmetic operations to calculate some values for an input waveform. With differential input mode on the MUX, both positive and negative voltages can be converted in 2's complement notation. Configure channel 0 accordingly. In this mode with a 3.0 V reference, the input voltage may range from -3 V to about +3 V. A floating supply must be used. Due to a peculiar anomaly on the EVB, 2 separate adjustable supplies would be required for this setup, but only 1 is easily available. A little cheating will be allowed and the 3.3V VDD supply on the EVB (pin A-1 on J24, see C8051F12x-DK.pdf) may be used in a pinch. The adjustable supply can be used as a floating supply with the negative terminal grounded for 0 – +3V inputs and the positive terminal grounded for 0 – -3V inputs.

The normal operation of the program should continuously sample a waveform input on AIN0.0 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 reset pushbutton should be provided to reset the stored minimum and maximum readings.

Optional Feature: 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 its 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. 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.

PART III – DAC Output

Write a program to convert digital values to analog signals and output them to the DAC0 pin. First, write a short routine that continuously increments an unsigned integer starting from 0. Once the integer overflows, it will start back at 0 again. This should generate a sawtooth wave on the DAC output. Confirm this with the oscilloscope. Use a 3.0V VREF for this part.

Next, enable ADC0 and connect it to a signal generator. Set it up to input a sine wave. Take the conversions generated by ADC0, and output them through DAC0. The signal you see through the oscilloscope should be very close to the signal that is being sent from the signal generator. If you use the input from the program written for PART II, you will need to figure out how to deal with a bipolar input passed to a unipolar output. Probably the easiest way is to scale the signal so that the peak-to-peak

voltage will fit in the 0 – 3 Volt range of the output DAC and add an offset equivalent to half the range (1.5 Volt). Now the output will have the same shape as the input but will have an offset and be scaled down.

PART IV – FIR Filter

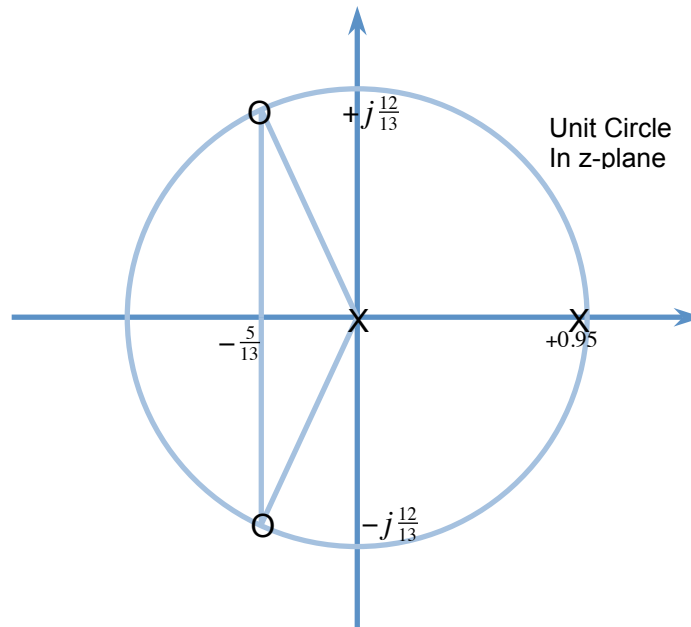
This program will build on the previous part, and filter the data coming in before outputting it through the DAC. Create a simple IIR (infinite impulse response) filter using the MAC. The filter to be implemented will have a response based on the transfer function and pole-zero diagram below:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{(z + \frac{5}{13} + j\frac{12}{13})(z + \frac{5}{13} - j\frac{12}{13})}{z(z - 0.95)} \times \frac{10}{32}$$

which leads to the difference equation:

$$y(k) = [x(k) + \frac{10}{13}x(k-1) + x(k-2) + 0.95y(k-1)] \frac{10}{32}$$

where $x(k)$ is the input from ADC0 at sample time k , $x(k-1)$ is the value of the previous sample etc., and $y(k)$ is the value being sent to the DAC. To implement this filter, you will need to store the current and previous two ADC0 readings as well as the previous output, and use the MAC to multiply $x(k-1)$ by $\frac{10}{13}$ and $y(k-1)$ by 0.95 as well as the overall scaling of $\frac{10}{32}$, necessary to limit the D.C. gain to a reasonable value. There are 2 mode that may be used to perform the multiplications on the MAC: in integer mode multiply everything by a large power of 2 (e.g. 2^{10}) & then divide the answer by the same power of 2 (right shifts), or in 2's complement, fraction values mode, normalize the coefficient to be less than 1.



If you use the 2's complement, fraction values mode and reordering for efficiency, the difference equation would be written as:

$$y(k) = 0.31250000x(k) + 0.31250000x(k-2) + 0.24038462x(k-1) + 0.29687500y(k-1)$$

The largest positive value less than 1 is $1-2^{-15} \approx 0.99996948$. The hexadecimal value is 0x7FFF. Use the

bit weightings in Fig. 12.3 in the 8051F120 technical reference manual to find the representations for the coefficients. ADC conversion should be set up to put the results in the low 12 bits of the 16-bit output.

After setting the MAC to mode (Fractional or Integer – your choice) & MAC Mode (not Multiply Only Mode), the sequence followed to evaluate the difference equation on the MAC is:

1. Start a new conversion
2. Update previous output & previous 2 inputs for the cycle
3. Clear the MAC Accumulator and set to MAC mode
4. Get the new sample
5. Subtract the 1.5V offset
6. Load MAC0A with 0.31250000 hexadecimal equivalent or scaled integer value
7. Load MAC0BH & MAC0BL with the 12 bits of current sample (k)
8. Load MAC0BH & MAC0BL with the 12 bits of second previous sample (k-2)
9. Load MAC0A with 0.24038462 hexadecimal equivalent or scaled integer value
10. Load MAC0BH & MAC0BL with the 12 bits of first previous sample (k-1)
11. Load MAC0A with 0.29687500 hexadecimal equivalent or scaled integer value
12. Load MAC0BH & MAC0BL with the 12 bits of previous output (k-1)
13. Do left or right bit shifts to align 12-bit answer on byte boundary
14. Check completion flag
15. Get the signed 16-bit results from MAC0ACC3 & MAC0ACC2 or 2 & 1 in integer mode
16. Add back in the 1.5V offset & scale (may be also done using the MAC if you are clever)
17. Output final result to DAC0
18. Repeat sequence

Before step 14. above it may be necessary to normalize the results with bit shifts in the MAC. The scaling in step 16. may be necessary if the DAC uses a different resolution than the ADC.

After implementing the filter, connect the output to the oscilloscope and input different frequencies from the signal generator. Explain your results. You should be able to show in the Z-plane that the discrete time system has a zero on the unit circle where the output is a minimum and a pole near the unit circle where the output is a maximum. The output for inputs near DC will have high gains and may cause data overflows. Don't worry about fixing this other than by reducing the input amplitude, but be ready to explain what is observed.

Due to the anomaly mentioned in PART II, a standard function generator connected to an oscilloscope and used as an input signal to the EVB will not be able to go to -3V. The board requires a floating differential signal, which the function generator, with common power supply grounds, is incapable of providing. So the signal is loaded and unable go below -1V. The workaround is to offset the input by 1.5V to keep it positive between 0V and 3V. Then subtract the offset from the digital value. Use this adjusted value for the digital filter $x(k)$ and then add an offset back in again before $y(k)$ is output to DAC0. Remember to ground the AIN- input and to get a full 12-bit value from a positive-only (0V – 3V) signal revert back to single-ended mode.

In digital filters, frequency increases counterclockwise along the unit circle with 0Hz at $z=1$ and $f_s/2$ (half the sample frequency) at $z=-1$. The sample frequency is determined by $1/T$ where T is the time for which an output voltage is held. The angle to the zero on the unit circle is 112.61986° so that frequency corresponds proportionally to $\frac{112.61986^\circ}{180^\circ} \cdot \frac{f_s}{2}$. Verify that the notch appears at the correct frequency. This is the frequency of an input sine wave that gives an output amplitude of zero. $H(1)$ is the D.C. gain of a

filter. Calculate this value and verify experimentally that it is correct with the rounded coefficients used in the implementation compared to the ideal coefficients.

It is easy to implement the filter with a standard C expression calculating $y(k)$ instead of using the MAC. This filter will behave the same, except at a much slower sampling rate. This is useful for debugging and comparison to verifying correct MAC and normalizing operation. You will receive partial credit for demonstrating a working software implementation if you are unable to get the MAC working. A software implementation should still be able to achieve a sampling frequency around 30kHz or more.

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.

It is highly recommended that you start using the MAC with a very simple difference equation, such as:

$$y(k) = [2x(k)]/2$$

Use the MAC to multiply by 2 and a right shift to do the divide. When this is working try increasing the size of the coefficients and add another term, but increase the complexity just a little bit at a time. In the assembly language examples in the manual, NOPs follow the last MAC command since the answer is not immediately available in the accumulator. You may need to insert a dummy statement in your program for this. If your answers look incorrect it is probably a timing issue.

Remember to check the SFR page before attempting an operation. The MAC is on MAC0_PAGE and the analog converters are on ADC0_PAGE. Also, there is a peculiar issue with DAC0. Make sure you output the low byte BEFORE the high byte. You will see periodic incorrect voltages output on regular intervals if this is not followed.

It is not clear what is happening when pointers are used to overlay a 16-bit int with two 8-bit chars using the sample code at the bottom of page 3 above, but testing has shown that execution speed takes a hit with a performance drop of about 40%. It is significantly faster to use

```
analogval = analoghi<<8 | analoglo;
```

or

```
analoghi = analogval>>8; analoglo = analogval;
```

to pull out the 16-bit or 8-bit values.

If you use integer mode MAC operations and create the fractional coefficients by a final divide, remember that the divisor can only be a power of 2 and implemented by right shifts. The larger the power of 2 the more accuracy there will be in your rounded coefficients since $\frac{10}{13}$ cannot be represented exactly. Also, instead of doing 7 right shifts to get the 12-bit answer into MAC0ACC1 & MAC0ACC0 where they will be read, it is faster to do a single left shift and read the answer from MAC0ACC2 & MAC0ACC1.