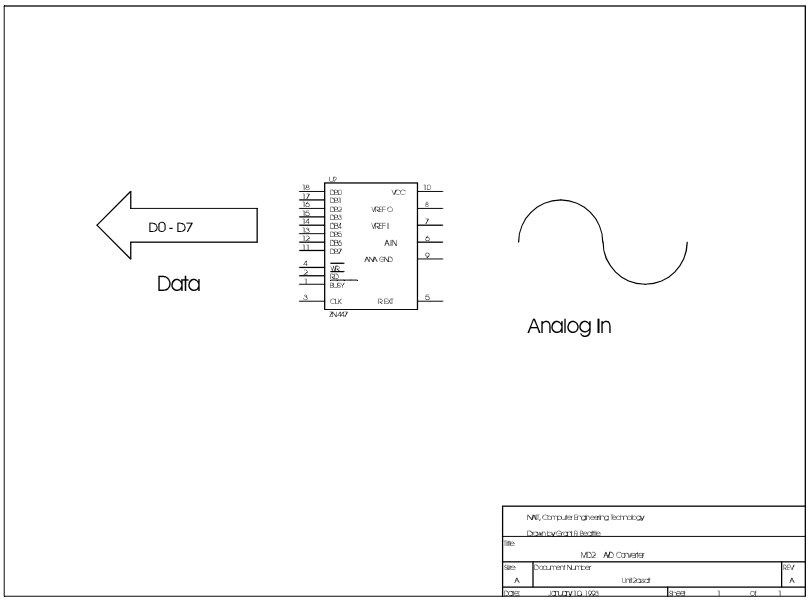
Micro Design 2

Lab #3



Analog to Digital Converters

Introduction

Unit 2 covered D/A converters and the methods used to connect such devices to the 68HC11's Serial Peripheral Interface. In order to build a complete data acquisition system, we need an A/D converter. Fortunately, the HC11 has a 4-channel A/D built in! This will enable the micro to assess real world conditions via some sensor, process the data with the CPU and then control an actuator with the D/A.

Unit 3 will cover several general techniques for performing analog to digital conversion. Following that will be a study of the HC11's A/D subsystem. The HC11 uses one A/D, but by implementing an input multiplexer, four discrete channels of analog input can be converted to digital.

For the last lab you created several test waveforms which could serve as the heart of a computer controlled function generator. In this lab, another Test and Measurement tool will be created, a simple digital voltmeter. Implementing the voltmeter will require A/D conversion, conversion from binary to decimal and finally conversion to ASCII for sending to the terminal screen.

Lab Outline

PRE-LAB: A/D Circuit

Several small changes must be made to your original CPU schematic to include the support circuitry for the A/D. Although the wiring is minimal, it is important to properly filter the analog V_{REF} and to provide current limiting protection for the A/D input(s).

SECTION 1: Wiring and Testing the A/D Circuit

Section 1 will consist of wiring of your circuit and verifying it's operation. A "sample" subroutine will be written and will act as a device driver for the A/D. Also, initialization code must be written which properly "powers up" the A/D subsystem.

SECTION 2: A/D Programming

One simple application for an A/D is the digital voltmeter. Although its range will be limited, the voltmeter presents an interesting exercise as it introduces a good use of BCD math.

Pre-Lab

A/D Circuit Schematic

( ) Step 1: Modifications to CPU Schematic

Based on your notes from class, modify your CPU schematic to include the support wiring for the A/D converter. In particular:

- Add the necessary filtering on V_{REF}.
- Add the current limiting resistor used to protect the CPU's AN0 input.

*** Have your schematic (pre-lab) checked off prior to construction! ***

Procedure

SECTION 1: WIRING AND TESTING THE A/D CIRCUIT

The next task is to build the circuit and test it's function. Also, you must write an A/D power-up routine and a SAMPLE subroutine and/or function which will be used in Section 2 (and later on in the course).

( ) Step 1: Wiring

Wire the A/D parts in the location specified by your instructor. Follow all of the wiring suggestions given in Lab #2 (D/A) in order to minimize noise in the analog circuit.

Please note that there are some pads and via's available on the pcb for the purpose of adding this circuit. Any resistors needed can be mounted "vertically" to fit the hole spacing provided.

Add another wire leaving the board to act as the A/D input.

( ) Step 2: A/D Power-Up Code

Write some short initialization code in C or assembly (your choice) which powers-up the A/D subsystem and selects the clock source which will introduce the least noise into the analog circuit. Also, you must include a delay of at least 100us to allow the charge pump and comparator circuits to stabilize.

( ) Step 3: Test Program and Subroutine SAMPLE

Write a short test program in the language of your choice which does the following:

i. Power up the A/D and initialize the SPI.
ii. Call your SAMPLE subroutine or function (see the header below).
iii. Echo the returned value to the LED Board and to the D/A. Shift the value four times before sending the value to the DAC.
iv. Return to ii (endless loop).

Create an ADC_LIB.ASM or ADC_LIB.C that contains the A/D equates and your SAMPLE subroutine or function. SAMPLE should do the following:

i. Start the conversion.
ii. Wait until the conversion is complete.
iii. Read the converted value from the appropriate result register.
iv. Return the converted value to the calling code.

The subroutine header for the assembly version of SAMPLE might look like this:

```
;SAMPLE          Requests a conversion from the A/D.  Returns the value upon
;                completion.
;REQUIRES -      Nothing.
;RETURNS -       Result of conversion returned in accumulator A.
;REGS AFFECTED - No other registers affected.
```

The prototype for the C version might look like:

```
unsigned char Sample(void);
```

Or you may prefer a signed or unsigned int return type such as:

```
unsigned int Sample(void);
```

( ) Step 4: A/D Verification

Using a variable voltage source (Anatek power supply) verify that your A/D outputs the correct value to the LED board over the operating range of 0.00v to +5.00v. Have your program checked off when it is working.

*** Have your calibration checked off at this point! ***

SECTION 2: A/D Digital Voltmeter Program

Once the A/D is in working order we can proceed to put it through its paces. Since we can use the A/D to read the value of some unknown analog voltage, it should be an easy matter to present that value to a human in a readable manner (in volts).

The step size of the A/D is 19.5mv, but for this lab we will assume that 19.5mv is approximately equal to 20mv[1]. We could either add calibration circuitry or else "massage" the data to make it exact, but for this lab the approximation is good enough. With a convenient step size of 20mv we can make the following "conversion" from the binary value to a voltage:

HEX VALUE	BCD VALUE	VOLTAGE
0x00	0000	0.00v
0x01	0001	0.02v
0x02	0002	0.04v
:	:	:
:	:	:
0xFE	0254	5.08v
0xFF	0255	5.10v

The change from the HEX VALUE column to the BCD VALUE column is obviously HEXTOBCD. The change from BCD VALUE to VOLTAGE is simply multiplying by two (which is the same as adding a number to itself). For instance, 255 + 255 = 510. The decimal point is not a concern. When displaying the result to a human simply send the five, then a decimal point, and finally the 10. Therefore the user sees 5.10, but internally the value 510 has been used. Isn't life wonderful?

Read the following information carefully and then decide whether you want to solve the lab using assembly or C.

( ) Step 1: ASSEMBLY VERSION

Write a subroutine which satisfies the following header:

```
;HEXTOBCD       Converts an 8 bit hex number into it's BCD equivalent.
;
;REQUIRES -      8 bit value to convert must be in accumulator A.
;               Accumulator A may hold any value from 0x00 to 0xFF.
;RETURNS -       16 bit BCD value is returned in accumulator D.
;               Accumulator D will return a value in the range 0x0000 to 0x0255.
;REGS AFFECTED - No other registers affected.
```

1 If you adjust the 5v power supply to +5.12v, the step size is 20mv and the voltmeter is accurate.

There are many ways to accomplish the task, several popular methods follow:

1. ITERATIVE SOLUTION

This method is described for your liberal education only! It is too slow to be used in this course!

This solution is the easiest to implement, but it takes the CPU the longest to execute. It is the software equivalent of RAMP A/D conversion. Simply put, two variables are required, the 8 bit input and the 16 bit output. The pseudocode might resemble:

```
HexToBCD
{
output = 0000;
while(input)
    {
    output + = 1;
    Decimal Adjust (output);
    input -=1;
    }
return(output);
}
```

The only other consideration is that the DAA operation only works on Acc A and also You must maintain a 16 bit result (which means moving carries from the LSB to the MSB). This implementation is relatively painless, just slow.

2. LOOK-UP TABLE

A look-up table is MUCH faster, but writing the code is not trivial. There are two possible sub-approaches. If you have no life at all, you can build a table of all of the possible BCD values from 0000 to 0255 and use the input value as an index into the table.

A shorter method is to employ two small look-up tables. Have a look at the HEX to DECIMAL conversion table in “Little Pink”. If I asked you to tell me the decimal equivalent of 0x9C you would look up the 9 in the third column and see “144". Then you would look up the C in the fourth column and see “12". Then 144 + 12 = 156.

To implement this in software, you need two look-up tables which are the same as the two columns on the programming card. Note that column four would be done with a byte-wide table and column three would be done with a word-wide table:

```
ByteTable:    .db     0x00, 0x01,       . . .   0x15

WordTable:    .dw     0x0000, 0x0016,   . . .   0x0240
```

The implementation is still tricky since there are a few details missing, but overall this version is FAST!

3. DIVISION

Conversion to BCD can also be accomplished using a couple of integer divisions. If The input number range is 0x00 - 0xFF, then we can divide by 100 to determine how many hundreds there are. The remainder after that division can then be divided by 10 To determine how many tens there are. The final remainder then becomes the ones. This method is not too difficult, but it is not a "screamer" either. At 41 cycles per division, it Will probably be slower than a look-up table, but it will still be much faster than the Iterative method.

4. BINARY WEIGHTED BCD ADDITION

If you realize that each bit position has a decimal value (1, 2, 4, 8, 16, ...) then it is possible to perform a series of additions (one for each bit that is true) to come up with A final BCD value.

( ) Step 1: C VERSION

Although you are required to have an understanding of BCD, the C version can be completed with or without using any BCD data manipulation. The easiest way of converting to decimal from an unsigned char is to first double the sample value (watch your data type here). Then, determine the hundreds, tens and ones using division similar to technique "3" above.

( ) Step 2: Program Specification (Both Versions)

Write a program which does the following:

1. Print a pleasant and colourful static screen (once) such as:

Micro Design 2 - Lab #3
Digital Voltmeter
by YourName Here

Voltage Reading: _

2. Get a conversion by calling your SAMPLE routine.

3. Convert this value as required so that it may be displayed in the form 0.00 thru 5.10.

4. Send the value to the screen. You will have to position the cursor, make the conversion to ASCII and place the decimal point.

5. Return to step 2 in an endless loop.

*** Have your program checked off when it is working! ***

LAB #3: LABCHECK SUMMARY

NAME: _____

SECTION: _____

PRE-LAB: Schematic

Make sure you hand in the A/D (CPU) schematic.

Date:_____ Instructor: _____ /5

LABCHECK #1: Wiring and Testing the A/D

Wiring (workmanship, color codes, analog grounding, etc.) /5

Verify A/D operation /5

Date:_____ Instructor: _____

LABCHECK #2: Digital Voltmeter

Operation /10

Program Documentation /10

Date:_____ Instructor: _____

YOU MUST HAND IN THE LAST PROGRAM COMPLETE WITH THE APPROPRIATE INCLUDE FILE(S).

TOTAL: /35