Microprocessor Systems
Lab 4: Analog Conversion & MAC

Nick Choi Samuel Deslandes

10/31/16

1 Introduction

The overall goal of this lab is to become familiar with configuring the ADC, DAC and MAC on the 8051 to perform various applications such as digital filtering and implementing a voltmeter.

The lab is divided into three sections. In the first section, a C program is created to configure the ADC on the 8051 to convert analog signals to digital signals when a pushbutton is pressed. The code polls a flag which is set when the pushbutton is pressed to determine whether or not to initialize an analog to digital conversion. When a conversion has been completed, the program displays the digital voltage reading rounded to six decimal places as well as the minimum voltage, maximum voltage and a moving average of the last 16 voltage readings via the ANSI terminal. The output is updated each time a conversion takes place.

In the third section, a C program is created to configure the DAC on the 8051 to convert digital signals to analog signals. This section builds upon the code in section 1 because the output of the ADC is used as the input of the DAC. The output of the DAC was observed using an oscilloscope.

In the fourth section, a new C program is written to implement an infinite impulse response (IIR) filter using the MAC on the 8051. The inputs of the digital filter are the present analog reading, the past two analog readings and the last digital output of the filter. All of these inputs are scaled by a coefficient and added together to reconstruct an analog waveform with digital signals using the 8051s MAC. The output of the filter was observed using an oscilloscope.

2 Methods

2.1 Software

The code for parts 1, 2 and 3 can be found in the appendix below. All code was uploaded and run on the 8051 through the programming/debugging USB port.

2.1.1 Part 1

The purpose of this program is to create a digital voltmeter using the analog to digital converter (ADC) on the 8051. When the pushbutton is pressed, an A-D conversion begins, and the result is used to calculate a voltage which is then displayed on the terminal.

Configurations for the ADC can be found in the REFOCN, ADC0CF, and ADC0CN registers, as well as the AMX0SL and AMX0CF registers for selecting the mode of operation of the analog input pins and which pins are selected for use. In the REF0CN SFR, bit1 enables the bias generator; this must be enabled. Bit0 of this SFR enables the internal reference buffer, which sets VREF0 to 2.4V. When using an external voltage source as a reference, this bit should be disabled. For this section the internal voltage reference was used. For the purposes of this lab, only one analog input operating in single-ended mode was needed, therefore the AMX0SL and AMX0CF registers did not require any modification; using the reset (default) values of these SFRs results in AIN0.0 selected with the desired mode of operation. The ADC0CF SFR, sets the gain for the input voltage (AIN0.0) and the

sampling rate for the conversion process. The sampling rate can be calculated as a function of SYSCLK and the value loaded into AD0SC (bit3-bit7 of this SFR) using the following equation:

$$AD0SC = \frac{SYSCLK}{2 \times CLK_{SAR0}} - 1$$

where CLK_{SAR0} is the ADC sampling rate.

Bits0-2 of this SFR are used to set the gain. For this lab a gain of 1 was used with AD0SC set to 23, resulting in an SAR_{CLK} of approximately 1MHz. Lastly ADC0 must be enabled via bit7 of the ADC0CN, which is also bit addressable as AD0EN.

For starting conversions, the pushbutton was connected as an external interrupt source as was previously done in lab 2. To do this, /INT0 must be routed to a port pin (P0.2 in this case) via the crossbar (XBR1) and external interrupts must be enabled. This can be done by setting the EA and EX0 bit addresses. Because only one conversion per button press was desired, /INT0 was set to be triggered on a negative falling edge by setting the IT0 bit address.

UART0 was also used to display the output onto the terminal, which is enabled in the XBR0 SFR. For configuring the ports, TX (P0.0) must be configured for push-pull, and RX (P0.1) and /INT0 (P0.2) must be set for open-drain with high impedance. This can be set in the P0MDOUT and P0 SFRs.

The code for this section follows a simple routine in which the program waits for a flag set by the /INT0 ISR to go high. This starts an A-D conversion, and the result is then used to calculate a voltage which is then printed to the terminal along with various statistics. Before starting the A-D conversion, the conversion complete interrupt flag (AD0INT) must be cleared; as its name implies, this bit is set by the hardware once the conversion is completed. To begin the conversion, set the AD0BUSY, then wait for the conversion to be completed by polling AD0INT. At this point the result of the conversion can be obtained by reading the 16bit SFR ADC0. The result of the conversion is the ratio of the input voltage between 0 and VREF0 mapped to a value between 0 and 4095. The result can be converted back into a voltage using the following equation:

$$V_{in} = \frac{ADC0}{4096} \times VREF0$$

where $VREF0 = 2.4\text{V}$

In order to display the max, min, and average voltage over the past 16 samples each value is stored in an array of integers of size 16. When the array is filled, using modular arithmetic the oldest entry in the array is updated with the newest sample. Using this array the max, min, and average can easily be calculated by traversing the array. Once these statistics have been calculated, the voltage values are displayed in decimal form, along with the raw result of the conversion displayed in hex.

2.1.2 Part 3

The program for this section follows a simple routine in which the result of an A-D conversion is sent directly to the DAC to be converted back to an analog signal.

The initialization for the ADC was similar to the previous section, the only difference being that VREF0 is now defined by an external 3V voltage source. To do this, bit0 of the REF0CN register must be cleared. Once REF0CN is configured, the only initialization necessary for DAC0 is setting bit7 of the DAC0CN register; this enables the DAC0 output pin.

The routine for this program continuously performs A-D conversions, and outputs the result to the DAC by loading it into the 16bit SFR "DAC0".

2.1.3 Part 4

The code for this section follows a similar procedure to that of the previous section, while introducing the MAC to implement a digital filter. As was done previously, an analog input is read in using the ADC, the result is passed through the filter, and the filtered result is then output through the DAC.

The IIR filter used can be described by the following difference equation:

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

where y(k) is the current output

y(k-1) is the previous output

x(k) is the current sample

x(k-1) is the previous sample

and x(k-2) is the sample before the previous sample

After the analog input was read in through the ADC, the variables storing the previous sample values were updated, and an offset value was subtracted from the current sample. This was needed because the function generator input was set to a 3Vpp wave with a 1.5V offset such that the wave ranged from $0 - 3$V.

Next, the accumulator was cleared, and the MAC was set to operate in integer multiply-accumulate mode. This can be set in the MAC0CF SFR. At this point, values can be loaded into the MAC. Since the MAC was operating in integer mode, the coefficients were multiplied by a large constant (2^{10}); the final result was normalized in the accumulator via right bit-shift operations once the calculations were complete.

First, the MAC0A register was loaded with the decimal value 320 (0.3125×2^{10}), then MAC0BH loaded with the high 8 bits of the current sample and MAC0BL with the lower 8 bits of the current sample. This represented the $0.31250000x(k)$ portion of the above difference equation. Next, MAC0BH and MAC0BL were similarly loaded with the sample before the previous sample, representing the $0.31250000x(k-1)$ portion of the equation. After this, MAC0A was loaded with the decimal value 246 ($\approx 0.24038462 \times 2^{10}$), and MAC0BH and MAC0BL with the previous sample, representing $0.24038462x(k-1)$. Lastly, MAC0A was loaded with the 304 (0.296875×2^{10}), and MAC0BH and MAC0BL with the previous output, representing the final part of the equation $0.29687500y(k-1)$. These products are summed in the accumulator and as mentioned before, were normalized by performing 10 right bit-shit operations. This was accomplished using a for loop and setting the "MAC0CF" register to `0x30`; this executes a single shift of the value in the accumulator to the right.

Values are stored in the accumulator using 5 8bit registers, “MAC0ACC0” - “MAC0ACC3” and an overflow register “MAC0OVR”. After the bit-shift operations, the desired result can be found in registers “MAC0ACC0” and “MAC0ACC1”. The values in these registers were combined into a single 16bit variable and the offset originally subtracted from the current sample is added back in. The final result is then output to DAC0.

2.2 Hardware

The hardware for section 1 involved using the potentiometer module to vary the voltage input to AIN0.0 for the A-D conversions. Rather than using the pins on the 60pin bus, the Analog I/O terminal block (J20) was used. A schematic for this can be viewed in the appendix below. The output of the potentiometer was connected to pin 5 on the terminal block (AIN0.0), and pin 8 (AGND) was connected to a common ground on the board.

Section 3 called for the use of a function generator and an oscilloscope, as well as an external voltage source for VREF0. The 3V reference voltage was generated using the potentiometer module, and was connected to pin 7 on the terminal block (VREF0). The function generator was set to produce a 3Vpp sinusoidal wave with 1.5V offset. This was connected to AIN0.0 on the terminal block (pin 5) using an alligator clip. The output from DAC0 (pin 3 on the terminal block) was connected to the oscilloscope also using an alligator clip. As was done before, AGND was connect to a universal ground. A schematic for this can be viewed in the appendix section below.

The hardware for section 4 was identical to that of the previous section.

3 Results

By completing section one of the lab, a C program was developed to convert variable analog voltages to digital values used to calculate a digital voltage. The program displayed the converted voltages, as well as statistics including the moving average of the last 16 samples, and the min/max voltage values. After completing section three of the lab, a C program was developed to convert digital voltages obtained from an ADC to the equivalent analog voltage values. By completing section four of the lab, a C program was created to use the MAC on the 8051 to reconstruct analog signals using the ADC and DAC written for the previous sections of the lab. The output of the MAC represented the output of an infinite impulse response digital filter.

4 Conclusion

The end results of the lab generally matched with the initial goals however there were numerous instances where the system did not behave as expected. In section 4 of the lab, the output of our digital filter was only able to accurately reconstruct the analog input signal up to a certain frequency threshold. This anomaly can be explained by the Nyquist sampling theorem which states that in order to accurately represent an analog signal, the sampling frequency must be at least double the frequency of the input signal. If the sampling frequency falls below this threshold, the 8051 is unable to obtain enough samples to accurately model

the input waveform. In this lab, the output signal did not accurately represent the input waveform after 25 kHz. Given that the sampling frequency for the 8051 was calculated to be 50 kHz, this result is what could be expected.

Prior to implementing the filter with the 8051's MAC, a software implementation of the IIR filter was used to make it easier to debug this section of the lab. Although the output signals of the software implementation are generally the same as the MAC, the algorithm's performance is limited compared to the MAC counterpart. This is because the code takes longer to compute the output values than the MAC due to the innate latency involved with running the code.

If more time was given to complete this lab assignment, the voltmeter created in section one would be modified to achieve the functionalities described in section two of the lab. This would create a digital voltmeter which would be capable of measuring the frequency of input waveforms as well as the AC RMS voltage value of the signal.

5 Appendices

5.1 Analog I/O Terminal block

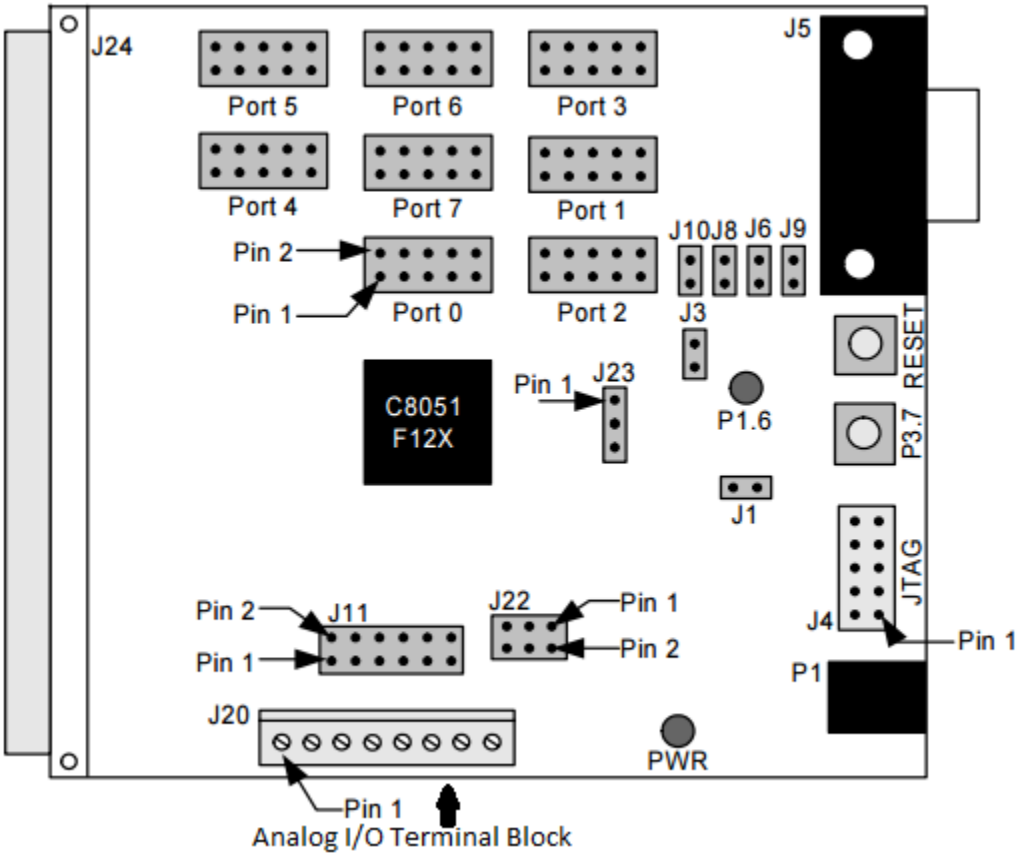

Figure 1: C8051F120 Targer Board layout

Pin#	Description
1	CP0+
2	CP0-
3	DAC0
4	DAC1
5	AIN0.0
6	AIN0.1
7	VREF0
8	AGND (Analog Ground)

Table 1: J20 Terminal Block reference table

5.2 Modified putget.h

```
//-----------------------------------------------------------------------------
// putget.h
//-----------------------------------------------------------------------------
// Title:                Microcontroller Development: putchar() & getchar() functions.
// Author:               Dan Burke
// Date Created:         03.25.2006
// Date Last Modified:   03.25.2006
//
// Description:          http://chaokhun.kmitl.ac.th/~kswichit/easy1/easy1_3.html
//
//
// Target:               C8051F120
// Tool Chain:           KEIL C51

//-----------------------------------------------------------------------------
// putchar()
//-----------------------------------------------------------------------------
void putchar(char c)
{
    while(!TI0);
    TI0=0;
    SBUF0 = c;
}

//-----------------------------------------------------------------------------
// getchar()
//-----------------------------------------------------------------------------
char getchar(void)
{
    char c;
    while(!RI0);
    RI0 =0;
    c = SBUF0;
// Echoing the get character back to the terminal is not normally part of getchar()
//    putchar(c);    // echo to terminal
    return SBUF0;
}
```

5.3 Part 1

5.3.1 Circuit Schematic for section 1

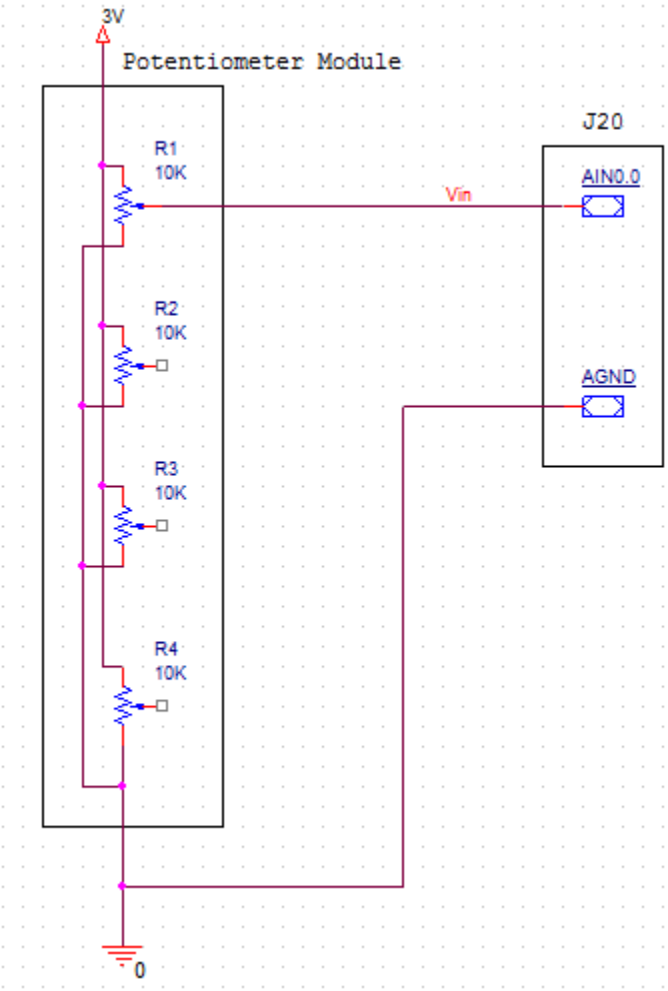

Figure 2: Circuit schematic for part 1

5.3.2 Code for part 1

```
// Nick Choi and Sam Deslandes
// Code for section 1 of Lab 4.
// This program acts as a digital voltmeter using the ADC, displaying the voltage in decimal
     and hex to the terminal.
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f120.h>
#include <stdio.h>
#include "putget.h"
//-----------------------------------------------------------------------------
// Global Constants
//-----------------------------------------------------------------------------
#define EXTCLK      22118400            // External oscillator frequency in Hz
#define SYSCLK      49766400            // Output of PLL derived from (EXTCLK * 9/4)
#define BAUDRATE    115200              // UART baud rate in bps
char butpress = 0;

//-----------------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------------
void main(void);
void SYSCLK_INIT(void);
void PORT_INIT(void);
void UART0_INIT(void);
void ADC0_INIT(void);
void display(int* start, char div);
void BUTPRESS_ISR(void) __interrupt 0;

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main(void)
{
    int ADCdata;
  int dataArray[16] = {0};
  char num_entries = 0;
  char size = 0;
  int i;

    WDTCN = 0xDE;                       // Disable the watchdog timer
    WDTCN = 0xAD;

    PORT_INIT();                        // Initialize the Crossbar and GPIO
    SYSCLK_INIT();                      // Initialize the oscillator
    UART0_INIT();                       // Initialize UART0
  ADC0_INIT();

  SFRPAGE = LEGACY_PAGE;
  IT0 = 1;                  // /INT0 triggered on negative falling edge

  SFRPAGE = UART0_PAGE;               // Direct output to UART0

    printf("\033[2J");                  // Erase screen & move cursor to home position
    printf("Test of the printf() function.\n\n\r");

  while(1)
    {
        if(butpress){
      butpress = 0;
      //printf("button pressed\n\r");

      AD0INT = 0;           // Clear conversion interrupt flag
      AD0BUSY = 1;            // Start conversion
      while(!AD0INT);         // Wait for conversion to end
```

```
      ADCdata = ADC0;         // Get result of conversion (using 16bit register)

      printf_fast_f("Result: %.6f\t0x%x\n\r",(ADCdata/4096.)*2.43,ADCdata);

      dataArray[num_entries%16] = ADCdata;
      num_entries += 1;

      // algorithm for # of relevant entries in array (for averaging)
      size = num_entries;
      if(num_entries > 15){
        size = 16;
      }

      display(dataArray,size);

    }
    }
}

void display(int* start, char div){
  float sum=0;
  char i;
  int min = 4095;
  int max = 0;

  for(i = 0; i<div; i++){
    // Get min value in array
    if(*(start+i) < min){
      min = *(start+i);
    }
    // Get max value in array
    if(*(start+i) > max){
      max = *(start+i);
    }
    // Sum entries in array
    sum += *(start+i);
  }

  printf_fast_f("Min is %.6f\t0x%x\n\r",(min/4096.)*2.43,min);
  printf_fast_f("Max is %.6f\t0x%x\n\r",(max/4096.)*2.43,max);
  printf_fast_f("Average is %.6f\t0x%x\n\n\r",((sum/div)/4096.)*2.43,  sum/div);
}

void BUTPRESS_ISR(void) __interrupt 0{
  butpress = 1;
}

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// Initialize the system clock to use a 22.1184MHz crystal as its clock source
//
void SYSCLK_INIT(void)
{
    int i;
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;             // Save Current SFR page

    SFRPAGE = CONFIG_PAGE;
    OSCXCN  = 0x67;                     // Start ext osc with 22.1184MHz crystal
    for(i=0; i < 256; i++);             // Wait for the oscillator to start up
    while(!(OSCXCN & 0x80));
    CLKSEL  = 0x01;
    OSCICN  = 0x00;
```

```
    SFRPAGE = CONFIG_PAGE;
    PLL0CN  = 0x04;
    SFRPAGE = LEGACY_PAGE;
    FLSCL   = 0x10;
    SFRPAGE = CONFIG_PAGE;
    PLL0CN |= 0x01;
    PLL0DIV = 0x04;
    PLL0FLT = 0x01;
    PLL0MUL = 0x09;
    for(i=0; i < 256; i++);
    PLL0CN |= 0x02;
    while(!(PLL0CN & 0x10));
    CLKSEL  = 0x02;

    SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;             // Save Current SFR page

    SFRPAGE  = CONFIG_PAGE;
    XBR0     = 0x04;                    // Enable UART0
    XBR1     = 0x04;          // /INT0 routed to port pin
    XBR2     = 0x40;                    // Enable Crossbar and weak pull-up
    P0MDOUT |= 0x01;                    // Set TX0 on P0.0 pin to push-pull
  P0       = 0x06;                // Additionally, set P0.0=0, P0.1=1, and P0.2=1.

  EA     = 1;            // Enable global interrupts
  EX0    = 1;            // Enable external interrupts

  SFRPAGE  = SFRPAGE_SAVE;            // Restore SFR page
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1
//
void UART0_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;             // Save Current SFR page

    SFRPAGE = TIMER01_PAGE;
    TMOD   &= ~0xF0;
    TMOD   |=  0x20;                    // Timer1, Mode 2, 8-bit reload
    TH1     = -(SYSCLK/BAUDRATE/16);    // Set Timer1 reload baudrate value T1 Hi Byte
    CKCON  |= 0x10;                     // Timer1 uses SYSCLK as time base
    TL1     = TH1;
    TR1     = 1;                        // Start Timer1

    SFRPAGE = UART0_PAGE;
    SCON0   = 0x50;                     // Mode 1, 8-bit UART, enable RX
    SSTA0   = 0x10;                     // SMOD0 = 1
    TI0     = 1;                        // Indicate TX0 ready

    SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
```

```
}

//-----------------------------------------------------------------------------
// ADC0_Init
//-----------------------------------------------------------------------------
void ADC0_INIT(void){
  char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;
  SFRPAGE = ADC0_PAGE;

  REF0CN |= 0x03;                    // turn on internal ref buffer and bias generator, vref0
      is ref voltage
  ADC0CF = 0xB8;               // AD0SC = 23 for SARclk of 1Mhz, Gain = 1
  AD0EN = 1;                  // enable ADC0

    SFRPAGE = SFRPAGE_SAVE;                // Restore SFR page
}
```

5.4 Schematic for parts 3 and 4

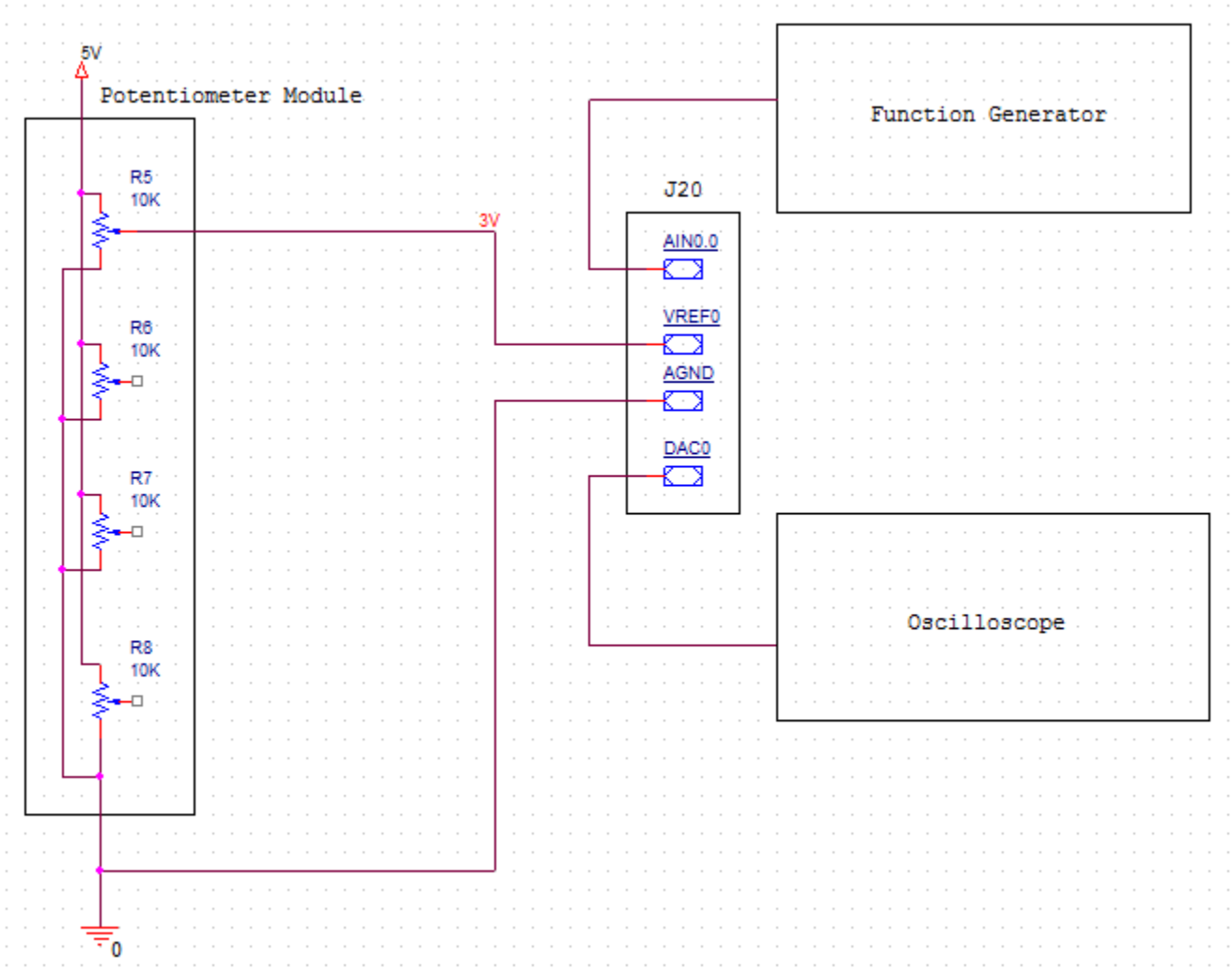

Figure 3: Circuit schematic for part 2

5.5 Part 3

5.5.1 Code for part 3

```
// Nick Choi and Sam Deslandes
// Code for section 3 of Lab 4.
// This program takes the conversions generated by ADC, and output them through DAC. The
    output mirrors the input.
//-----------------------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------------------
#include <c8051f120.h>
#include <stdio.h>
#include "putget.h"
//-----------------------------------------------------------------------------------------
// Global Constants
//-----------------------------------------------------------------------------------------
#define EXTCLK      22118400            // External oscillator frequency in Hz
#define SYSCLK      49766400            // Output of PLL derived from (EXTCLK * 9/4)
#define BAUDRATE    115200              // UART baud rate in bps
char butpress = 0;

//-----------------------------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------------------------
void main(void);
void SYSCLK_INIT(void);
void PORT_INIT(void);
void UART0_INIT(void);
void ADC0_INIT(void);
void DAC0_INIT(void);

//-----------------------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------------------
void main(void)
{

    WDTCN = 0xDE;                       // Disable the watchdog timer
    WDTCN = 0xAD;

    PORT_INIT();                        // Initialize the Crossbar and GPIO
    SYSCLK_INIT();                      // Initialize the oscillator
    UART0_INIT();                       // Initialize UART0
  ADC0_INIT();
  DAC0_INIT();

  SFRPAGE = UART0_PAGE;               // Direct output to UART0

    printf("\033[2J");                  // Erase screen & move cursor to home position
    printf("Test of the printf() function.\n\n\r");

  while(1)
    {
        AD0INT = 0;          // Clear conversion interrupt flag
    AD0BUSY = 1;           // Start conversion
    while(!AD0INT);        // Wait for conversion to end

    DAC0 = ADC0;         // Get data form conversion and output directly to DAC
  }
}
//-----------------------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------------------
//
// Initialize the system clock to use a 22.1184MHz crystal as its clock source
//
```

```
void SYSCLK_INIT(void)
{
    int i;
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;             // Save Current SFR page

    SFRPAGE = CONFIG_PAGE;
    OSCXCN  = 0x67;                     // Start ext osc with 22.1184MHz crystal
    for(i=0; i < 256; i++);             // Wait for the oscillator to start up
    while(!(OSCXCN & 0x80));
    CLKSEL  = 0x01;
    OSCICN  = 0x00;

    SFRPAGE = CONFIG_PAGE;
    PLL0CN  = 0x04;
    SFRPAGE = LEGACY_PAGE;
    FLSCL   = 0x10;
    SFRPAGE = CONFIG_PAGE;
    PLL0CN |= 0x01;
    PLL0DIV = 0x04;
    PLL0FLT = 0x01;
    PLL0MUL = 0x09;
    for(i=0; i < 256; i++);
    PLL0CN |= 0x02;
    while(!(PLL0CN & 0x10));
    CLKSEL  = 0x02;

    SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;             // Save Current SFR page

    SFRPAGE  = CONFIG_PAGE;
    XBR0     = 0x04;                    // Enable UART0
    XBR1     = 0x04;          // /INT0 routed to port pin
    XBR2     = 0x40;                    // Enable Crossbar and weak pull-up
    P0MDOUT |= 0x01;                    // Set TX0 on P0.0 pin to push-pull
  P0       = 0x02;                 // Additionally, set P0.0=0, P0.1=1

  SFRPAGE  = SFRPAGE_SAVE;             // Restore SFR page
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1
//
void UART0_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;             // Save Current SFR page

    SFRPAGE = TIMER01_PAGE;
    TMOD   &= ~0xF0;
    TMOD   |=  0x20;                    // Timer1, Mode 2, 8-bit reload
```

```
    TH1     = -(SYSCLK/BAUDRATE/16);     // Set Timer1 reload baudrate value T1 Hi Byte
    CKCON  |= 0x10;                      // Timer1 uses SYSCLK as time base
    TL1     = TH1;
    TR1     = 1;                         // Start Timer1

    SFRPAGE = UART0_PAGE;
    SCON0   = 0x50;                      // Mode 1, 8-bit UART, enable RX
    SSTA0   = 0x10;                      // SMOD0 = 1
    TI0     = 1;                         // Indicate TX0 ready

    SFRPAGE = SFRPAGE_SAVE;              // Restore SFR page
}

//-----------------------------------------------------------------------------
// ADC0_Init
//-----------------------------------------------------------------------------
void ADC0_INIT(void){
  char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;
  SFRPAGE = ADC0_PAGE;

  REF0CN |= 0x02;                  // turn on internal ref buffer and bias generator, vref0
      is ref voltage
  ADC0CF = 0xB8;              // AD0SC = 23 for SARclk of 1Mhz, Gain = 1
  AD0EN = 1;                // enable ADC0

    SFRPAGE = SFRPAGE_SAVE;              // Restore SFR page
}

void DAC0_INIT(void){
  char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;
  SFRPAGE = DAC0_PAGE;

  REF0CN |= 0x02;                  // turn on bias generator, but not internal reference
      buffer. vref comes from external source
  DAC0CN = 0x80;              // enable DAC0

    SFRPAGE = SFRPAGE_SAVE;              // Restore SFR page
}
```

5.6 Part 4

5.6.1 Code for part 4

```
// Nick Choi and Sam Deslandes
// Code for section 4 of Lab 4.
// This program utilizes the MAC to filter a signal received through the ADC, then sends the
     filtered signal out through the DAC.
// The digital filter acts as a notch filter, in whihch at a certain frequecy the frequency
    response goes to zero.
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f120.h>
#include <stdio.h>
#include "putget.h"
//-----------------------------------------------------------------------------
// Global Constants
//-----------------------------------------------------------------------------
#define EXTCLK       22118400            // External oscillator frequency in Hz
#define SYSCLK       49766400            // Output of PLL derived from (EXTCLK * 9/4)
#define BAUDRATE     115200              // UART baud rate in bps
#define OFFSET     1502                // Offset value = (Offset voltage/Vref)*4069
```

```
//-----------------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------------
void main(void);
void SYSCLK_INIT(void);
void PORT_INIT(void);
void UART0_INIT(void);
void ADC0_INIT(void);
void DAC0_INIT(void);
void ADC_read(void);


//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main(void)
{
  char i;
  signed int curr_ADC = 0;
  signed int  past_ADC = 0;
  signed int pastest_ADC = 0;
  signed int past_DAC = 0;
  signed int result;

    WDTCN = 0xDE;                          // Disable the watchdog timer
    WDTCN = 0xAD;

    PORT_INIT();                           // Initialize the Crossbar and GPIO
    SYSCLK_INIT();                         // Initialize the oscillator
    UART0_INIT();                          // Initialize UART0
  ADC0_INIT();
  DAC0_INIT();

  SFRPAGE = UART0_PAGE;                  // Direct output to UART0

    printf("\033[2J");                     // Erase screen & move cursor to home position
    printf("Test of the printf() function.\n\n\r");

  while(1)
    {
    SFRPAGE = ADC0_PAGE;
    ADC_read();                 // start new conversion


    pastest_ADC = past_ADC;      // update previous inputs
    past_ADC = curr_ADC;
    curr_ADC = ADC0;          // get new sample

    SFRPAGE = MAC0_PAGE;
    MAC0CF = 0x08;           // clear accumulator, integer mode, multiply and accumulate

    curr_ADC -= OFFSET;          // subtract 1.5V offset


    MAC0A = 320;           // .3125*2^10 (scaled integer)

    MAC0BH = curr_ADC>>8;      // load MAC0B with current sample (k)
    MAC0BL = curr_ADC;

    MAC0BH = pastest_ADC>>8;    // load MAC0B with 2nd previous sample (k-2)
    MAC0BL = pastest_ADC;

    MAC0A = 246;                      // .24038462*2^10 (scaled integer)

    MAC0BH = past_ADC>>8;      // load MAC0B with previous sample (k-1)
    MAC0BL = past_ADC;

    MAC0A = 304;           // load MAC0A with .296875*2^10 (scaled integer)
```

```
    MAC0BH = past_DAC>>8;      // load MAC0B with previous output (k-1)
    MAC0BL = past_DAC;

    SFRPAGE = MAC0_PAGE;
    for(i=0;i<10;i++){         // shift right 10 times
      MAC0CF = 0x30;
    }

    SFRPAGE = MAC0_PAGE;       // dummy op, then get results
    result = MAC0ACC1<<8 | MAC0ACC0;

    result += OFFSET;                  // add back in 1.5V offset

    SFRPAGE = DAC0_PAGE;
    DAC0 = result;           // output result to DAC

    past_DAC = DAC0;         // update previous output

  }
}

void ADC_read(void){
  AD0INT = 0;          // Clear conversion interrupt flag
  AD0BUSY = 1;           // Start conversion
  while(!AD0INT);        // Wait for conversion to end
}

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// Initialize the system clock to use a 22.1184MHz crystal as its clock source
//
void SYSCLK_INIT(void)
{
    int i;
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;              // Save Current SFR page

    SFRPAGE = CONFIG_PAGE;
    OSCXCN  = 0x67;                      // Start ext osc with 22.1184MHz crystal
    for(i=0; i < 256; i++);              // Wait for the oscillator to start up
    while(!(OSCXCN & 0x80));
    CLKSEL  = 0x01;
    OSCICN  = 0x00;

    SFRPAGE = CONFIG_PAGE;
    PLL0CN  = 0x04;
    SFRPAGE = LEGACY_PAGE;
    FLSCL   = 0x10;
    SFRPAGE = CONFIG_PAGE;
    PLL0CN |= 0x01;
    PLL0DIV = 0x04;
    PLL0FLT = 0x01;
    PLL0MUL = 0x09;
    for(i=0; i < 256; i++);
    PLL0CN |= 0x02;
    while(!(PLL0CN & 0x10));
    CLKSEL  = 0x02;

    SFRPAGE = SFRPAGE_SAVE;              // Restore SFR page
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
```

```
//
// Configure the Crossbar and GPIO ports
//
void PORT_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;             // Save Current SFR page

    SFRPAGE  = CONFIG_PAGE;
    XBR0     = 0x04;                    // Enable UART0
    XBR1     = 0x04;          // /INT0 routed to port pin
    XBR2     = 0x40;                    // Enable Crossbar and weak pull-up
    P0MDOUT |= 0x01;                    // Set TX0 on P0.0 pin to push-pull
  P0       = 0x02;                // Additionally, set P0.0=0, P0.1=1

  SFRPAGE  = SFRPAGE_SAVE;            // Restore SFR page
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1
//
void UART0_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;             // Save Current SFR page

    SFRPAGE = TIMER01_PAGE;
    TMOD   &= ~0xF0;
    TMOD   |=  0x20;                    // Timer1, Mode 2, 8-bit reload
    TH1     = -(SYSCLK/BAUDRATE/16);    // Set Timer1 reload baudrate value T1 Hi Byte
    CKCON  |= 0x10;                     // Timer1 uses SYSCLK as time base
    TL1     = TH1;
    TR1     = 1;                        // Start Timer1

    SFRPAGE = UART0_PAGE;
    SCON0   = 0x50;                     // Mode 1, 8-bit UART, enable RX
    SSTA0   = 0x10;                     // SMOD0 = 1
    TI0     = 1;                        // Indicate TX0 ready

    SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}


void ADC0_INIT(void){
  char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;
  SFRPAGE = ADC0_PAGE;

  REF0CN |= 0x02;                 // turn on internal ref buffer and bias generator, vref0
      is ref voltage
  ADC0CF = 0xB8;              // AD0SC = 23 for SARclk of 1Mhz, Gain = 1
  AD0EN = 1;                // enable ADC0

    SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}

void DAC0_INIT(void){
  char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;
  SFRPAGE = DAC0_PAGE;
```

```
  REF0CN |= 0x02;                     // turn on bias generator, but not internal reference
      buffer. vref comes from external source
  DAC0CN = 0x80;                // enable DAC0

    SFRPAGE = SFRPAGE_SAVE;               // Restore SFR page
}
```

6 References

"MPS Lab 4," in RPI ECSE Department, 2016. [Online]. Available: `http://www.rpi.edu/dept/ecse/mps/MPS_Lab_Ex4-ADC.pdf`. Accessed: Oct. 30, 2016.

"C8051 Manual," in RPI ECSE Department, 1.4 ed., 2005. [Online]. Available: `https://www.ecse.rpi.edu/courses/CStudio/Silabs/C8051F12x-13x.pdf`. Accessed: Oct. 30, 2016.

"C8051F12x Development Kit User's Guide ," in RPI ECSE Department, Rev. 0.6,May 2005. [Online]. Available: `https://www.ecse.rpi.edu/courses/CStudio/Silabs/C8051F12x-DK.pdf`. Accessed: Oct. 30, 2016.