**Microprocessor Systems Lab 4**

Analog Conversion and Mac

Shane Conaboy

Justin Yost

ECSE-4790 Microprocessor Systems

November 2, 2015

**Introduction**

Lab 4 investigated the use of the 8051 Microcontroller’s analog-to-digital converter (ADC) and digital-to-analog converter (DAC), as well as the multiply and accumulate (MAC) engine. The ADC was used to make a digital voltmeter that would measure the applied voltage and display it on the terminal screen. The DAC was then incorporated to convert a digital signal coming from the ADC back to analog and send it to an oscilloscope to compare the original signal to the processed one. Lastly, the MAC was used to create a notch filter that was observable through use of the oscilloscope and changing the frequency of the input signal.

**Procedure**

The first part of the lab was to set up and use the 8051’s ADC. The ADC was set for single-ended mode, the input port on AIN0.0, the clock rate less than 2.5MHz, and an internal reference voltage of 2.4V by using the internal bias generator and buffer as instructed by the reference manual. A push-button was wired as an external interrupt to trigger an analog-to-digital conversion. If an external interrupt from the button was detected, the program would clear the “conversion done” flag, AD0INT, and set the AD0BUSY flag to 1 to begin the conversion. The program would then wait for AD0INT to be set high before reading the data from the ADC0L and ADC0H buffers. The input voltage was output to an accuracy of six decimal places. The program also recorded the highest and lowest voltage readings and computed the average of the last sixteen voltages. The high, low, and average numbers were output in hexadecimal format.

After the ADC was functioning, the DAC was incorporated next. There was almost no initial setup needed for the DAC; it was set to update the output on writes to DAC0H and then enabled. It was important that DAC0L be set before DAC0H in the code since the output updated upon setting DAC0H. The 8051 was programmed to read analog values using the ADC and convert them to digital values, and then using the DAC to convert them back to analog and update the output. A function generator was used to generate a sine wave as the input, and an oscilloscope was used to monitor both the input and output waveforms. Since the sine wave had both positive and negative values, and the ADC that was set up did not support negative voltages, the function generator was wired through a variable voltage source to add a DC offset to the signal and keep it positive.

With both the ADC and DAC working, the notch filter was then implemented using both a software implementation and a MAC implementation. Implementing the filter within the code was straightforward and allowed the notch to be viewed on the oscilloscope before using the MAC. The MAC was then setup in multiply and accumulate mode, fractional mode, non-saturation mode, and to right-shift the results. Using fractional mode, the MAC represents numbers using 2’s compliment as shown in Figure 1 below. Using this mode, the coefficients in the filter equation needed to be converted to 2’s compliment form. The hexadecimal 2’s compliment representation for the numbers are shown below. Once the MAC was set up, the procedure given in the lab was followed, loading the MAC with the appropriate values and reading from the rounding engine, before outputting the resulting signal through the DAC. The oscilloscope was used again to monitor both input and output signals. Once the MAC was working, the frequency of the input signal was changed until the notch frequency was found.



Figure : MAC Fractional Mode Data Representation

**Results**

All parts of the lab were successfully implemented. For the voltmeter, calculating the accuracy to 6 decimal places and avoiding overflow was a small issue. Since the printf command to interface the ANSI terminal does not output decimal values, the strategy was to multiple by 10N, where N is the number of places to shift the decimal over (6 in this case), and then convert the value to and integer. An unsigned integer is 16 bits long, representing numbers up to 65535, which is less than 106 and would cause overflow. To remedy this, the first three decimal places were obtained first using the above method, the integer value was subtracted off, and then the next three decimal places were obtained from that number. After the overflow issue was resolved, the voltmeter worked as expected and was verified using an actual multimeter.

There were little problems with setting up the DAC. The most time-consuming portion was setting up the function generator, oscilloscope, and DC offset. Once everything was wired appropriately, the input and output waveforms were visible on the oscilloscope and matched as expected.

Getting the notch filter set up also posed little problems. First, it was implemented in software and observed on the oscilloscope. It took a class period to figure out what was supposed to be seen since the lab did not mention if there was a target frequency to achieve and the oscilloscope was not displaying the output waveform in an easily-viewable manner. After everything was understood and the MAC was set up in fractional mode with the appropriate 2’s compliment values for the coefficients, the filter was implemented successfully, matching what the software implementation showed but at a higher frequency due to the increased running speed from using the MAC. The software and MAC notch filter frequencies were 1.55kHz and approximately 24kHz respectively. The MAC-implemented notch frequency is shown in Figure 2 below.

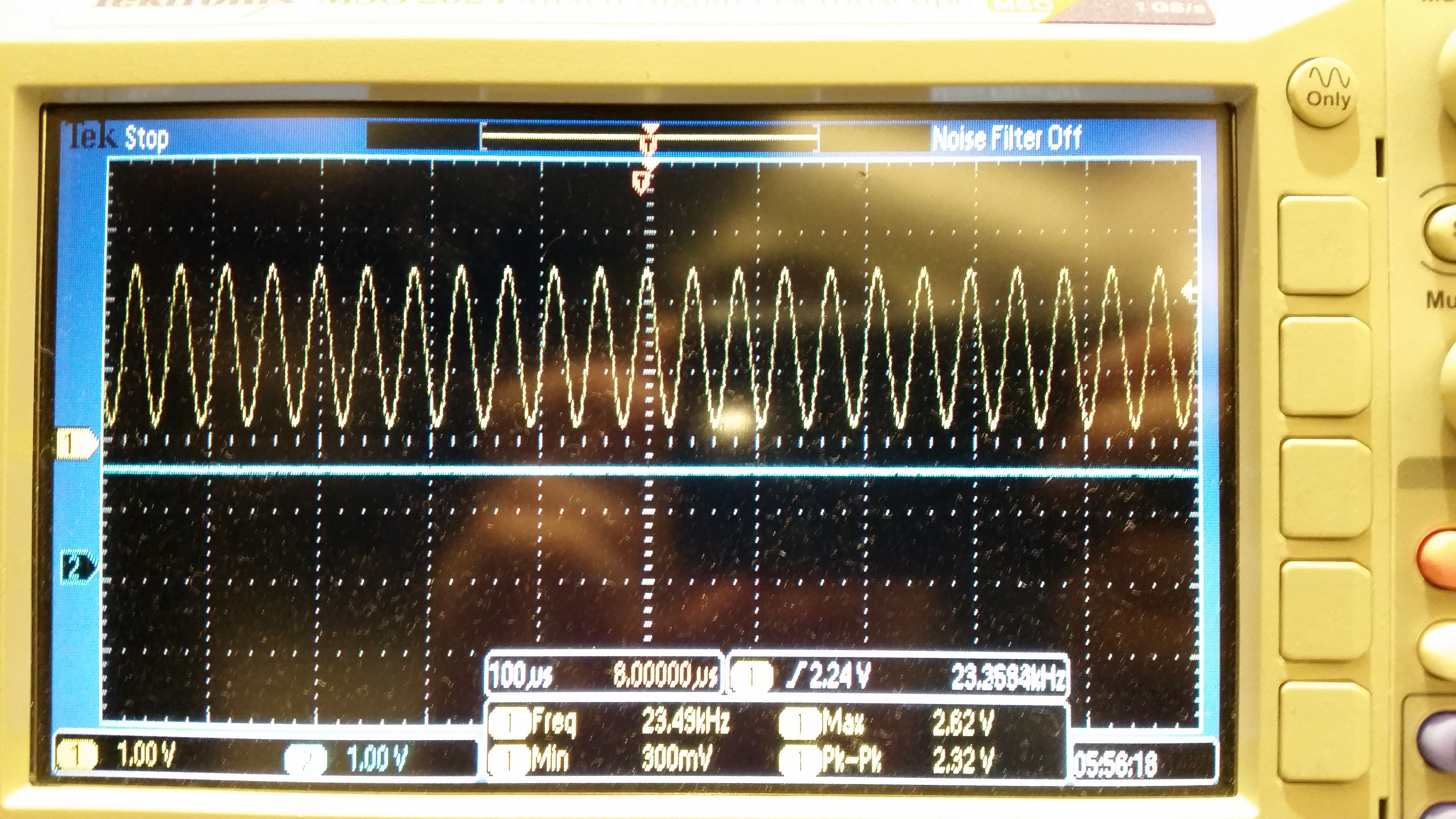


Figure : MAC-Implemented Notch Filter

**Conclusion**

The lab investigated the use of the 8051’s analog-to-digital and digital-to-analog converters, as well as the multiply and accumulate engine. Several different reference voltages, both internal and external, were applied to the ADC and DAC, and how the reference affected the output values was tested. In the last part of the lab, the speed at which the software runs was directly observed by comparing the notch frequencies of the software- and MAC-implemented filter. This lab demonstrated how to leverage different hardware components of the 8051 microcontroller and how using the built-in components can dramatically increase the running speed of the program.

**Appendix A: Circuit Schematics**

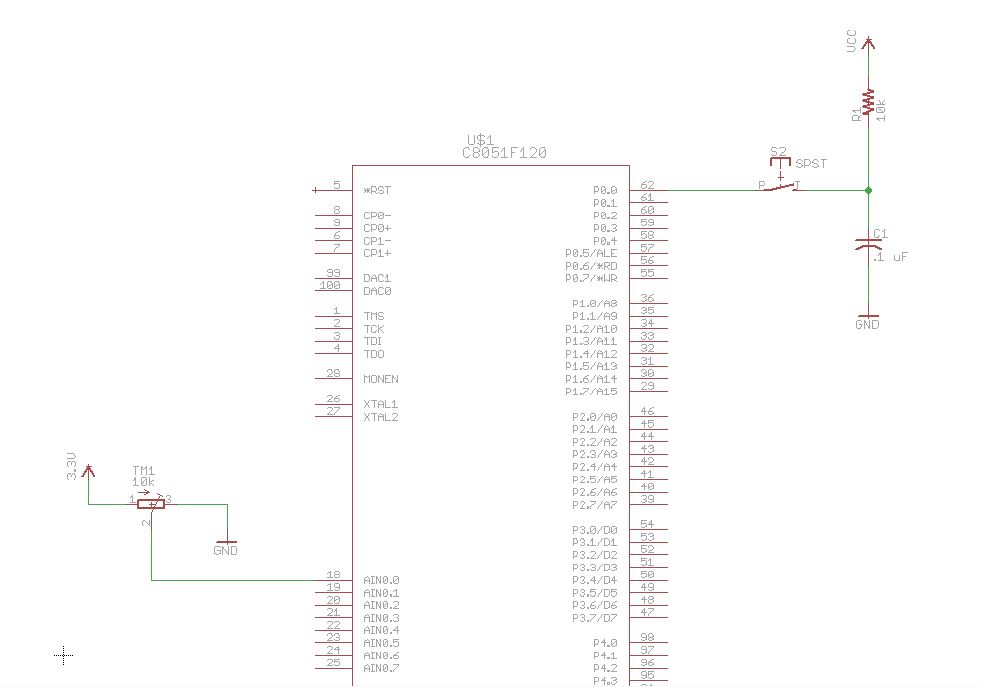


Figure : Part 1 Circuit Schematic

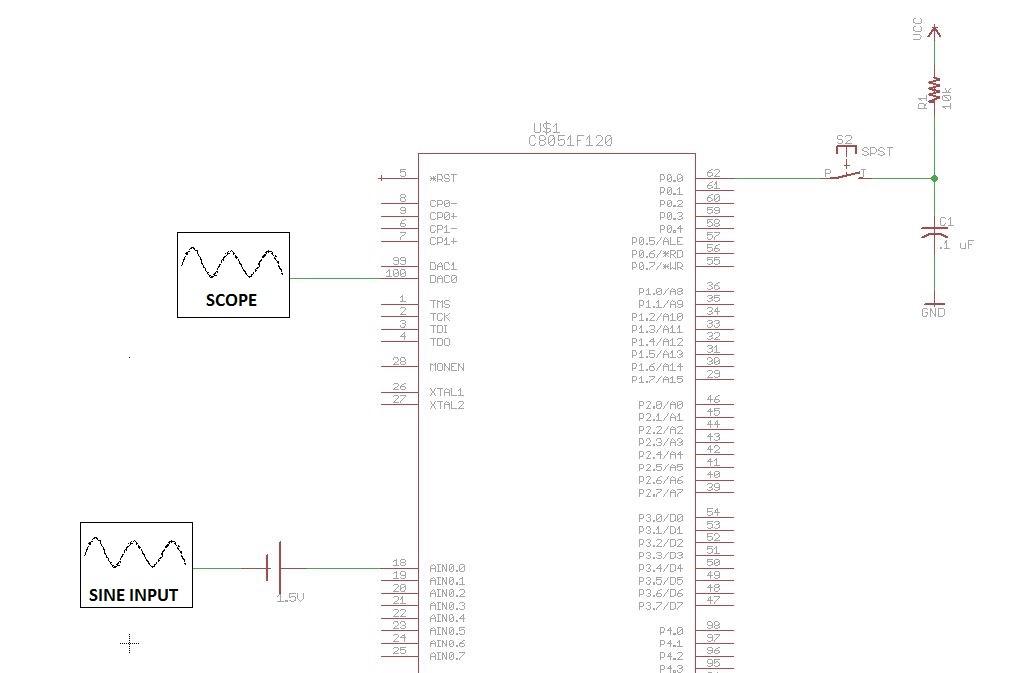


Figure : Parts 3 and 4 Circuit Schematic

**Appendix B: Code**

Part 1

//------------------------------------------------------------------------------------

// lab4\_part1.c

//------------------------------------------------------------------------------------

// Part 1 acts like a digital voltmeter, taking in an analong input on AIN0.0 and

// displays the voltage on a terminal when a push button is pressed.

//------------------------------------------------------------------------------------

// Includes

//------------------------------------------------------------------------------------

#include <c8051f120.h>

#include <stdio.h>

#include “putget.h”

//------------------------------------------------------------------------------------

// Global Constants

//------------------------------------------------------------------------------------

#define EXTCLK 22118400 // External oscillator frequency in Hz

#define SYSCLK 22118400

#define BAUDRATE 115200 // UART baud rate in bps

//------------------------------------------------------------------------------------

// Function Prototypes

//------------------------------------------------------------------------------------

void main(void);

void SYSCLK\_INIT(void);

void PORT\_INIT(void);

void UART0\_INIT(void);

void ADC\_INIT(void);

void INTERRUPT\_INIT(void);

// Global Variables

char start\_conversion = 0; // Flag to indicate when to start the ADC

//------------------------------------------------------------------------------------

// MAIN Routine

//------------------------------------------------------------------------------------

void main(void)

{

unsigned short int analogval;

unsigned char \*analoghi, \*analoglow;

float VREF = 2.4; // Reference voltage

float result; // Voltage measurement result

short int result\_high = 0;

short int result\_low = 4095;

char samples = 0; // Use the last 16 samples to take an average

short int accum[16] = {0}; // Array to hold the samples

char i;

unsigned long int sum = 0;

unsigned int avg;

float result\_dec1; // Need 2 variables to hold the decimal value to prevent overflow

float result\_dec2;

analoglow = (char \*) &analogval;

analoghi = analoglow + 1;

WDTCN = 0xDE; // Disable the watchdog timer

WDTCN = 0xAD;

PORT\_INIT(); // Initialize the Crossbar and GPIO

SYSCLK\_INIT(); // Initialize the oscillator

UART0\_INIT(); // Initialize UART0

ADC\_INIT(); // Initialize ADC0

INTERRUPT\_INIT();

while(1)

{

if (start\_conversion)

{

SFRPAGE = ADC0\_PAGE;

AD0INT = 0; // Clear the “conversion done” flag

AD0BUSY = 1; // Start A/D Conversion

while (AD0INT == 0); // Wait for the conversion to finish

\*analoglow = ADC0L; // Read the low byte

\*analoghi = ADC0H; // Read the high byte

SFRPAGE = UART0\_PAGE;

result = analogval\*2.4/(float)4095;

// Maintain high and low hex values

if (analogval > result\_high) result\_high = analogval;

if (analogval < result\_low) result\_low = analogval;

// Maintain the array of voltage readings

if (samples < 16) {

accum[samples] = analogval;

samples++;

}

else {

for (i = 0; i < samples-1; i++) {

accum[i] = accum[i+1];

}

accum[samples-1] = analogval;

}

// Compute the average

sum = 0;

for (i = 0; i < samples; i++) {

sum = sum + accum[i];

}

avg = (int)(sum/samples);

result\_dec1 = (1000\*result-1000\*(int)result);

result\_dec2 = (1000\*(result\_dec1-(int)result\_dec1));//-100\*(int)(result\_dec1-(int)result\_dec1);

// Erase screen and move cursor to home position

printf(“\033[2JVoltage reading: %d.%03d%03d\n\r”, (int)result, (int)result\_dec1, (int)result\_dec2);

printf(“Hex value: 0x%03X\n\r”, analogval);

printf(“High value hex: 0x%03X\n\r”, result\_high);

printf(“Low value hex: 0x%03X\n\r”, result\_low);

printf(“Average value hex: 0x%03X\n\r”, avg);

start\_conversion = 0;

}

}

}

//External Interrupt (Push Button) ISR

void int0\_interrupt(void) \_\_interrupt 0

{

start\_conversion = 1;

}

//------------------------------------------------------------------------------------

// SYSCLK\_Init

//------------------------------------------------------------------------------------

// Initialize the system clock to use a 22.1184MHz crystal as its clock source

void SYSCLK\_INIT(void)

{

int i = 0;

char SFRPAGE\_SAVE = SFRPAGE;

SFRPAGE = CONFIG\_PAGE;

OSCXCN = 0x67;

for (i = 0; i < 3000; i++); // Wait 1ms for initialization

while ((OSCXCN & 0x80) == 0);

CLKSEL = 0x01;

OSCICN &= ~0x80;

SFRPAGE = SFRPAGE\_SAVE;

}

//------------------------------------------------------------------------------------

// PORT\_Init

//------------------------------------------------------------------------------------

// Configure the Crossbar and GPIO ports

void PORT\_INIT(void)

{

SFRPAGE = CONFIG\_PAGE;

WDTCN = 0xDE; // Disable watchdog timer.

WDTCN = 0xAD;

XBR0 = 0x04;

XBR1 = 0x04; // Enable external interrupt

XBR2 = 0x40; //Enable crossbar

P0MDOUT = 0x01;// Set TX0 pin to push-pull, RX and Push Button to open drain

P0 = 0x06; // Inputs RX and Push Button set to high impedence

SFRPAGE = LEGACY\_PAGE;

IT0 = 0; // /INT0 is edge triggered, falling-edge.

}

//------------------------------------------------------------------------------------

// UART0\_Init

//------------------------------------------------------------------------------------

// Configure the UART0 using Timer1, for <baudrate> and 8-N-1

void UART0\_INIT(void)

{

SFRPAGE = UART0\_PAGE;

SCON0 = 0x50; // Mode 1, 8-bit UART, enable RX

SSTA0 = 0x10; // SMOD0 = 1

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;

TI0 = 1; // Indicate TX0 ready

}

void ADC\_INIT(void)

{

char SFRPAGE\_SAVE = SFRPAGE;

SFRPAGE = ADC0\_PAGE;

AMX0CF = 0x00; // All input ports set to single-ended mode

AMX0SL = 0x00; // Select port AIN0.0 for inpit

ADC0CF = 0x20; // ADC clock rate < 2.5MHz, internal gain of 1

ADC0CN = 0X00;

REF0CN = 0X03; // Voltage reference from VREF0 pin; enable bias generator and reference buffer

AD0EN = 1; //Enable ADC

}

void INTERRUPT\_INIT(void) {

EA = 1;

EX0 = 1;

}

Part 3

//------------------------------------------------------------------------------------

// lab4\_part3.c

//------------------------------------------------------------------------------------

// Part 3 takes in an analog signal on AIN0.0, converts it to a digital signal using

// the ADC, converts it back to an analog using the DAC and outputs it on the DAC0

// pin.

//------------------------------------------------------------------------------------

// Includes

//------------------------------------------------------------------------------------

#include <c8051f120.h>

#include <stdio.h>

#include "putget.h"

//------------------------------------------------------------------------------------

// Global Constants

//------------------------------------------------------------------------------------

#define EXTCLK 22118400 // External oscillator frequency in Hz

#define SYSCLK 22118400

#define BAUDRATE 115200 // UART baud rate in bps

//------------------------------------------------------------------------------------

// Function Prototypes

//------------------------------------------------------------------------------------

void main(void);

void SYSCLK\_INIT(void);

void PORT\_INIT(void);

void UART0\_INIT(void);

void ADC\_INIT(void);

void INTERRUPT\_INIT(void);

void DAC\_INIT(void);

void TIMER\_INIT(void);

// Global Variables

char start\_conversion = 0; // Flag to indicate when to start the ADC

unsigned int dig\_val = 0;

//------------------------------------------------------------------------------------

// MAIN Routine

//------------------------------------------------------------------------------------

void main(void)

{

unsigned short int analogval;

unsigned char \*analoghi, \*analoglow;

float VREF = 3; //Voltage reference

float result; // ADC voltage result

short int result\_high = 0;

short int result\_low = 4095;

char samples = 0;

short int accum[16] = {0}; // Still compute the average

char i;

unsigned long int sum = 0;

unsigned int avg;

float result\_dec1; // Need 2 variables to hold the decimal value to prevent overflow

float result\_dec2;

analoglow = (char \*) &analogval;

analoghi = analoglow + 1;

WDTCN = 0xDE; // Disable the watchdog timer

WDTCN = 0xAD;

PORT\_INIT(); // Initialize the Crossbar and GPIO

SYSCLK\_INIT(); // Initialize the oscillator

UART0\_INIT(); // Initialize UART0

ADC\_INIT(); // Initialize ADC0

INTERRUPT\_INIT();

DAC\_INIT();

while(1)

{

//dig\_val = dig\_val + 50; // Generate the sawtooth

SFRPAGE = ADC0\_PAGE;

AD0INT = 0; // Clear the "conversion done" flag

AD0BUSY = 1; // Start A/D Conversion

while (AD0INT == 0); // Wait for the conversion to finish

\*analoglow = ADC0L; // Read the low byte

\*analoghi = ADC0H; // Read the high byte

SFRPAGE = DAC0\_PAGE; // Output the digital value through the DAC

DAC0L = analogval; //digval;

DAC0H = analogval >> 8; //digval >> 8;

SFRPAGE = UART0\_PAGE;

result = analogval\*2.4/(float)4095;

// Maintain high and low hex values

if (analogval > result\_high) result\_high = analogval;

if (analogval < result\_low) result\_low = analogval;

// Maintain the array of voltage readings

if (samples < 16) {

accum[samples] = analogval;

samples++;

}

else {

for (i = 0; i < samples-1; i++) {

accum[i] = accum[i+1];

}

accum[samples-1] = analogval;

}

// Compute the average

sum = 0;

for (i = 0; i < samples; i++) {

sum = sum + accum[i];

}

avg = (int)(sum/samples);

result\_dec1 = (1000\*result-1000\*(int)result);

result\_dec2 = (1000\*(result\_dec1-(int)result\_dec1));

// Erase screen and move cursor to home position

printf("\033[2JVoltage reading: %d.%03d%03d\n\r", (int)result, (int)result\_dec1, (int)result\_dec2);

printf("Hex value: 0x%03X\n\r", analogval);

printf("High value hex: 0x%03X\n\r", result\_high);

printf("Low value hex: 0x%03X\n\r", result\_low);

printf("Average value hex: 0x%03X\n\r", avg);

}

}

//External Interrupt (Push Button) ISR

void int0\_interrupt(void) \_\_interrupt 0

{

start\_conversion = 1;

}

//------------------------------------------------------------------------------------

// SYSCLK\_Init

//------------------------------------------------------------------------------------

// Initialize the system clock to use a 22.1184MHz crystal as its clock source

void SYSCLK\_INIT(void)

{

int i = 0;

char SFRPAGE\_SAVE = SFRPAGE;

SFRPAGE = CONFIG\_PAGE;

OSCXCN = 0x67;

for (i = 0; i < 3000; i++); // Wait 1ms for initialization

while ((OSCXCN & 0x80) == 0);

CLKSEL = 0x01;

OSCICN &= ~0x80;

SFRPAGE = SFRPAGE\_SAVE;

}

//------------------------------------------------------------------------------------

// PORT\_Init

//------------------------------------------------------------------------------------

// Configure the Crossbar and GPIO ports

void PORT\_INIT(void)

{

SFRPAGE = CONFIG\_PAGE;

WDTCN = 0xDE; // Disable watchdog timer.

WDTCN = 0xAD;

XBR0 = 0x04;

XBR1 = 0x04; // Enable external interrupt

XBR2 = 0x40; //Enable crossbar

P0MDOUT = 0x01;// Set TX0 pin to push-pull, RX and Push Button to open drain

P0 = 0x06; // Inputs RX and Push Button set to high impedence

SFRPAGE = LEGACY\_PAGE;

IT0 = 0; // /INT0 is edge triggered, falling-edge.

}

//------------------------------------------------------------------------------------

// UART0\_Init

//------------------------------------------------------------------------------------

// Configure the UART0 using Timer1, for <baudrate> and 8-N-1

void UART0\_INIT(void)

{

char SFRPAGE\_SAVE = SFRPAGE; // Save Current SFR page

SFRPAGE = UART0\_PAGE;

SCON0 = 0x50; // Mode 1, 8-bit UART, enable RX

SSTA0 = 0x10; // SMOD0 = 1

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;

TI0 = 1; // Indicate TX0 ready

SFRPAGE = SFRPAGE\_SAVE; // Restore SFR page

}

void ADC\_INIT(void)

{

char SFRPAGE\_SAVE = SFRPAGE;

SFRPAGE = ADC0\_PAGE;

AMX0CF = 0x00; // All input ports set to single-ended mode

AMX0SL = 0x00; // Select port AIN0.0 for inpit

ADC0CF = 0x20; // ADC clock rate < 2.5MHz, internal gain of 1

ADC0CN = 0X00;

REF0CN = 0X02; // Voltage reference from VREF0 pin; enable bias generator and disable reference buffer

AD0EN = 1; //Enable ADC

}

void DAC\_INIT(void)

{

SFRPAGE = DAC0\_PAGE;

DAC0CN = 0x80;

}

void INTERRUPT\_INIT(void) {

EA = 1;

//ET0 = 1; // Timer0 interrupts

EX0 = 1; //External interrupts

}

void TIMER\_INIT(void) {

TMOD = 0x01; // Timer0 in 16-bit mode

CKCON = 0x08; // Timer0 uses sysclk

TR0 = 1; // Enable Timer0

}

Part 4

//------------------------------------------------------------------------------------

// lab4\_part4.c

//------------------------------------------------------------------------------------

// Part 4 builds upon parts 1 and 3 of lab 4 by using the ADC and DAC, as well as the

// MAC, to implement a notch filter.

//------------------------------------------------------------------------------------

// Includes

//------------------------------------------------------------------------------------

#include <c8051f120.h>

#include <stdio.h>

#include "putget.h"

//------------------------------------------------------------------------------------

// Global Constants

//------------------------------------------------------------------------------------

#define EXTCLK 22118400 // External oscillator frequency in Hz

#define SYSCLK 22118400

#define BAUDRATE 115200 // UART baud rate in bps

//------------------------------------------------------------------------------------

// Function Prototypes

//------------------------------------------------------------------------------------

void main(void);

void SYSCLK\_INIT(void);

void PORT\_INIT(void);

void UART0\_INIT(void);

void ADC\_INIT(void);

void INTERRUPT\_INIT(void);

void DAC\_INIT(void);

void TIMER\_INIT(void);

void MAC\_INIT(void);

// Global Variables

char start\_conversion = 0; // Flag to indicate when to start the ADC

unsigned int dig\_val = 0;

//------------------------------------------------------------------------------------

// MAIN Routine

//------------------------------------------------------------------------------------

void main(void)

{

unsigned char adcValH[3];

unsigned char adcValL[3];

unsigned int result;

unsigned short int analogval;

unsigned char analoghi, analoglow;

float VREF = 3;

WDTCN = 0xDE; // Disable the watchdog timer

WDTCN = 0xAD;

PORT\_INIT(); // Initialize the Crossbar and GPIO

SYSCLK\_INIT(); // Initialize the oscillator

ADC\_INIT(); // Initialize ADC0

DAC\_INIT();

MAC\_INIT();

adcValH[2] = adcValH[1] = adcValL[2] = adcValL[1] = 0;

result = 0;

while(1)

{

SFRPAGE = ADC0\_PAGE;

AD0INT = 0; // Clear the "conversion done" flag

AD0BUSY = 1; // Start A/D Conversion

while (AD0INT == 0); // Wait for the conversion to finish

analoglow = ADC0L; // Read the low byte

analoghi = ADC0H; // Read the high byte

analogval = analoghi<<8 | analoglow;

// Update the variables in the filter equation

adcValH[2] = adcValH[1]; // x(k-2) high byte

adcValH[1] = adcValH[0]; // x(k-1) high byte

adcValL[2] = adcValL[1]; // x(k-2) low byte

adcValL[1] = adcValL[0]; // x(k-1) low byte

adcValH[0] = analoghi; // x(k) high and low bytes

adcValL[0] = analoglow;

SFRPAGE = MAC0\_PAGE;

MAC0CF |= 0x08; // Clear MAC

// Load the MAC with the correct values and compute the filter equation

MAC0AH = 0x28;

MAC0AL = 0x00;

MAC0BH = adcValH[0];

MAC0BL = adcValL[0];

MAC0BH = adcValH[2];

MAC0BL = adcValL[2];

MAC0AH = 0x1E;

MAC0AL = 0xC4;

MAC0BH = adcValH[1];

MAC0BL = adcValL[1];

MAC0AH = 0x26;

MAC0AL = 0x00;

SFRPAGE = MAC0\_PAGE; // Delay with any dummy command

result = (int)MAC0RNDH<<8 | MAC0RNDL; // Read the result from the rounding engine

SFRPAGE = DAC0\_PAGE; // Output the result through the DAC

DAC0L = result;

DAC0H = result>>8;

}

}

//------------------------------------------------------------------------------------

// SYSCLK\_Init

//------------------------------------------------------------------------------------

// Initialize the system clock to use a 22.1184MHz crystal as its clock source

void SYSCLK\_INIT(void)

{

int i = 0;

char SFRPAGE\_SAVE = SFRPAGE;

SFRPAGE = CONFIG\_PAGE;

OSCXCN = 0x67;

for (i = 0; i < 3000; i++); // Wait 1ms for initialization

while ((OSCXCN & 0x80) == 0);

CLKSEL = 0x01;

OSCICN &= ~0x80;

SFRPAGE = SFRPAGE\_SAVE;

}

//------------------------------------------------------------------------------------

// PORT\_Init

//------------------------------------------------------------------------------------

// Configure the Crossbar and GPIO ports

void PORT\_INIT(void)

{

SFRPAGE = CONFIG\_PAGE;

WDTCN = 0xDE; // Disable watchdog timer.

WDTCN = 0xAD;

XBR0 = 0x04;

XBR1 = 0x04; // Enable external interrupt

XBR2 = 0x40; //Enable crossbar

P0MDOUT = 0x01;// Set TX0 pin to push-pull, RX and Push Button to open drain

P0 = 0x06; // Inputs RX and Push Button set to high impedence

SFRPAGE = LEGACY\_PAGE;

IT0 = 0; // /INT0 is edge triggered, falling-edge.

}

//------------------------------------------------------------------------------------

// UART0\_Init

//------------------------------------------------------------------------------------

// Configure the UART0 using Timer1, for <baudrate> and 8-N-1

void UART0\_INIT(void)

{

SFRPAGE = UART0\_PAGE;

SCON0 = 0x50; // Mode 1, 8-bit UART, enable RX

SSTA0 = 0x10; // SMOD0 = 1

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;

TI0 = 1; // Indicate TX0 ready

}

void ADC\_INIT(void)

{

SFRPAGE = ADC0\_PAGE;

AMX0CF = 0x00; // All input ports set to single-ended mode

AMX0SL = 0x00; // Select port AIN0.0 for inpit

ADC0CF = 0x20; // ADC clock rate < 2.5MHz, internal gain of 1

ADC0CN = 0X00;

REF0CN = 0X02; // Voltage reference from VREF0 pin; enable bias generator and disable reference buffer

AD0EN = 1; //Enable ADC

}

void DAC\_INIT(void)

{

SFRPAGE = DAC0\_PAGE;

DAC0CN = 0x80;

}

void INTERRUPT\_INIT(void) {

EA = 1;

EX0 = 1; //External interrupts

}

void TIMER\_INIT(void) {

TMOD = 0x01; // Timer0 in 16-bit mode

CKCON = 0x08; // Timer0 uses sysclk

TR0 = 1; // Enable Timer0

}

void MAC\_INIT(void) {

SFRPAGE = MAC0\_PAGE;

MAC0CF = 0x1A; //MAC mode, fractional mode, rounding will not saturate, reset MAC, shift right

}