EEL4511C

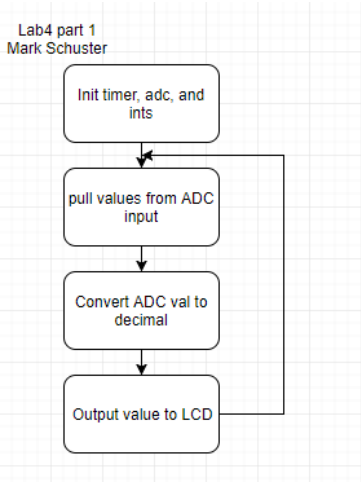
Schuster, Mark

Lab 4

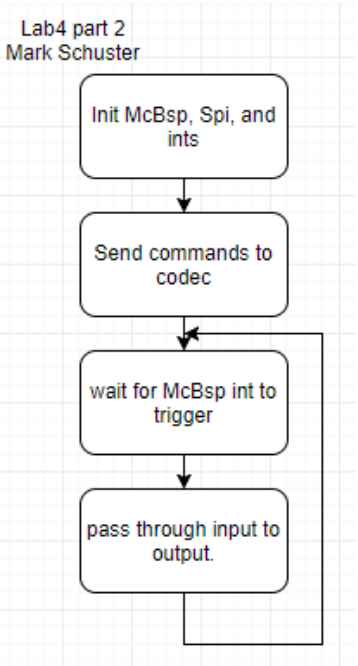
02/26/2018

1.1 Flowcharts:

Ohmmeter flowchart:



Codec flowchart:



1.2 Schematics/Decoding Logic:

None for this lab.

1.3 Problems Encountered:

In this lab I had no major problems. The codec took forever to set up☹

1.4 Program Code:

Headers:

ADC_Utils.h

```
// File:   I2C_LCD_Utils.h
// Date:   02/24/2018
// Name:   Mark Schuster
// Class:  EEE4511C (DSP)

#ifndef ADCUTILS_H_
#define ADCUTILS_H_

void initADC(void);

#endif
```

Codec_Utils.h:

```
// File:   Codec_Utils.h
// Date:   02/24/2018
// Name:   Mark Schuster
// Class:  EEE4511C (DSP)

#ifndef CODECUTILS_H_
#define CODECUTILS_H_

extern interrupt void codecIsr();
void initCodec(void);

#endif
```

I2C_LCD_Utils.h

```
// File:   I2C_LCD_Utils.h
// Date:   02/24/2018
// Name:   Mark Schuster
// Class:  EEE4511C (DSP)

#ifndef I2CLCDUTILS_H_
#define I2CLCDUTILS_H_


// Initializes the LCD and clears the display.
```

```
void initLCD(void);

// Sends a command to the LCD over I2C.
void sendCmdListLCD(Uint16 *, Uint16);

// Writes a character to the LCD after it has been
// initialized with "initLCD".
void sendCharLCD(char);

// Writes a string to the LCD using "sendCharLCD".
void sendStringLCD(char*);

// Send the clear screen to the LCD,
// wiping the display.
void clearLCD(void);

// Set the LCD's cursor the first
// position in the top right corner.
void cursorHomeLCD(void);

#endif
```

Interrupt_ISRs.h

```
//  File:   Interrupt_ISRs.h
//  Date:   02/24/2018
//  Name:   Mark Schuster
//  Class:  EEE4511C (DSP)

#ifndef INTERRUPTISRS_H_
#define INTERRUPTISRS_H_

interrupt void timer1Isr(void);
interrupt void audioIsr(void);

#endif
```

Interrupt_Utils.h

```
//  File:   Interrupt_Utils.h
//  Date:   02/24/2018
//  Name:   Mark Schuster
//  Class:  EEE4511C (DSP)

#ifndef INTERRUPTUTILS_H_
```

```
#define INTERRUPTUTILS_H_


void initInterrupts(void);


#endif
```

OneToOneI2CDriver.h

```
/*
 * OneToOneI2CDriver.h
 *
 *  Created on: Sep 24, 2016
 *      Author: Raz Aloni
 */


#ifndef ONETOONEI2CDRIVER_H_
#define ONETOONEI2CDRIVER_H_


/*
 * <summary>
 *                  Initializes the I2C to run in Master Mode for a One-To-One connection
 * </summary>
 * <param="slaveAddress">Address of the slave device to write to</param>
 * <param="sysClkMhz">System Clock Frequency in Mhz</param>
 * <param="I2CClkKHz">Desired I2C Clock Frequency in KHz</param>
 */
void I2C_O2O_Master_Init(Uint16 slaveAddress, float32 sysClkMhz, float32 I2CClkKHz);


/*
 * <summary>
 *                  Sends bytes via I2C
 * </summary>
 * <param="values">Pointer to array of bytes to send</param>
 * <param-"length">Length of array</param>
 */
void I2C_O2O_SendBytes(Uint16 * const values, Uint16 length);



#endif /* ONETOONEI2CDRIVER_H_ */
```

Timer1_Utils.h

```
//  File:   Timer1_Utils.h
```

```
// Date:   02/24/2018
// Name:   Mark Schuster
// Class:  EEE4511C (DSP)

#ifndef TIMER1UTILS_H_
#define TIMER1UTILS_H_

extern interrupt void timer1Isr();
void initTimer1(void);

#endif
```

ADC_Utils.c

```
// File:   ADC_Utils.c
// Date:   02/24/2018
// Name:   Mark Schuster
// Class:  EEE4511C (DSP)

#include <DSP2833x_Device.h>
#include "DSP28x_Project.h"
#include "I2C_LCD_Utils.h"
#include "ADC_Utils.h"

void initADC(void){
   InitAdc();
   EALLOW;
   PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
   AdcRegs.ADCTRL1.all = 0x0170;
   AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0;
   AdcRegs.ADCREFSEL.bit.REF_SEL = 0x0;
   AdcRegs.ADCMAXCONV.all = 0;
   AdcRegs.ADCTRL1.bit.CPS = 1;
   AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 1;

   return;
}
```

Codec_Utils.c

```
// File:   Codec_Utils.c
// Date:   02/24/2018
// Name:   Mark Schuster
```

```
// Class:  EEE4511C (DSP)


#include <DSP2833x_Device.h>
#include "DSP28x_Project.h"
#include "Codec_Utils.h"
#include "Interrupt_ISRs.h"


void initCodec(void){
                        EALLOW;
   PieVectTable.MRINTB = audioIsr; //link it to my interrupt
   PieCtrlRegs.PIEIER6.bit.INTx3 = 1;
   IER |= M_INT6;
   EDIS;
}
```

I2C_LCD_Utils.c

```
// File:   I2C_LCD_Utils.c
// Date:   02/24/2018
// Name:   Mark Schuster
// Class:  EEE4511C (DSP)


#include <DSP2833x_Device.h>
#include "DSP28x_Project.h"
#include "OneToOneI2CDriver.h"
#include "I2C_LCD_Utils.h"
#include "I2C_LCD_Utils.h"
#include "I2C_LCD_Utils.h"
#include "I2C_LCD_Utils.h"



// Initializes the LCD and clears the display.
void initLCD(void)
{
   Uint16 LCDCommandList[] = {0x33,0x32,0x28,0x0F,0x01};
   I2C_O2O_Master_Init(0x3F, 150, 400);
   sendCmdListLCD(LCDCommandList, sizeof(LCDCommandList)/sizeof(Uint16));
   cursorHomeLCD();


   return;
}
```

```
// Sends a command to the LCD over I2C.
void sendCmdListLCD(Uint16 *cmdList, Uint16 cmdListLen)
{
   for(Uint16 itter=0; itter<cmdListLen; itter++)
   {
      Uint16 upperNibble, lowerNibble;
      lowerNibble = ((cmdList[itter] & 0x000F) << 4) | 0x8;
      upperNibble = (cmdList[itter] & 0x00F0) | 0x8;
      Uint16 nibbleEnableCmds[] = { (upperNibble | 0x4), upperNibble,
                                    (lowerNibble | 0x4), lowerNibble};
      I2C_O2O_SendBytes(nibbleEnableCmds, sizeof(nibbleEnableCmds)/sizeof(Uint16));
   }
   DELAY_US(1000);

   return;
}



// Writes a character to the LCD after it has been
// initialized with "initLCD".
void sendCharLCD(char c){



   Uint16 upperNibble, lowerNibble;
   lowerNibble = (((Uint16)c & 0x000F) << 4) | 0x9;
   upperNibble = ((Uint16)c & 0x00F0) | 0x9;
   Uint16 nibbleEnableCmds[] = { (upperNibble | 0x4), upperNibble,
                                 (lowerNibble | 0x4), lowerNibble};
   I2C_O2O_SendBytes(nibbleEnableCmds, sizeof(nibbleEnableCmds)/sizeof(Uint16));

   return;
}

// Writes a string to the LCD using "sendCharLCD".
void sendStringLCD(char* str){
   for(Uint16 itter=0; str[itter]!='\0'; itter++)
      sendCharLCD(str[itter]);

   return;
}
```

```
// Send the clear screen to the LCD,
// wiping the display.
void clearLCD(void)
{
   Uint16 clearVal[] = {0x01};
   sendCmdListLCD(clearVal, sizeof(clearVal)/sizeof(Uint16));

   return;
}

// Set the LCD's cursor the first
// position in the top right corner.
void cursorHomeLCD(void){
   Uint16 homeVal[] = {0x02};
   sendCmdListLCD(homeVal, sizeof(homeVal)/sizeof(Uint16));

   return;
}
```

Interrupt_ISRs.c

```
//  File:   Interrupt_ISRs.c
//  Date:   02/24/2018
//  Name:   Mark Schuster
//  Class:  EEE4511C (DSP)

#include <DSP2833x_Device.h>
#include "DSP28x_Project.h"
#include "Interrupt_ISRs.h"
#include "I2C_LCD_Utils.h"

interrupt void timer1Isr(){
   CpuTimer1Regs.TCR.bit.TIF = 1;
   double voltage = AdcMirror.ADCRESULT0;
   voltage *= (3.0/4096.0);
   voltage *= 100.0;
   Uint16 voltageOnes = (Uint16)voltage / 100;
   Uint16 voltageTenths = ((Uint16)voltage / 10) - voltageOnes*10;
   Uint16 voltageHunths = (Uint16)voltage - voltageOnes*100 - voltageTenths*10;
```

```
    cursorHomeLCD();
    sendStringLCD("Voltmeter=");
    sendCharLCD(voltageOnes+'0');
    sendCharLCD('.');
    sendCharLCD(voltageTenths+'0');
    sendCharLCD(voltageHunths+'0');
    sendCharLCD('V');

    return;
}



interrupt void audioIsr(void){
    enum{LEFT, RIGHT,};
    static Uint16 state = LEFT;
    if(state == LEFT){
        McbspbRegs.DXR1.all = McbspbRegs.DRR1.all;
        McbspbRegs.DXR2.all = McbspbRegs.DRR2.all;
        state = RIGHT;
    }
    else if(state == RIGHT)
    {
       McbspbRegs.DXR1.all = McbspbRegs.DRR1.all;
       McbspbRegs.DXR2.all = McbspbRegs.DRR2.all;
       state = LEFT;
    }
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;
}
```

Interrupt_Utils.c

```
// File:   Interrupt_Utils.c
// Date:   02/24/2018
// Name:   Mark Schuster
// Class:  EEE4511C (DSP)

#include <DSP2833x_Device.h>
#include "DSP28x_Project.h"
#include "Interrupt_Utils.h"

void initInterrupts(void){
```

```
    InitPieCtrl();
    InitPieVectTable();
    EnableInterrupts();

}
```

OneToOneI2CDriver.c

```
/*
 * OneToOneI2CDriver.c
 *
 *  Created on: Sep 24, 2016
 *      Author: Raz Aloni
 */

#include <DSP2833x_Device.h>
#include "OneToOneI2CDriver.h"

/* Ideal module clock frequency for I2C */
static const Uint16 IdealModClockFrqMHz = 12;

/*
 * <summary>
 *                  Initializes the GPIO for the I2C
 * </summary>
 */
static void InitI2CGpio();

/*
 * <summary>
 *                  Calculates and sets the ClockDivides for the I2C Module
 * </summary>
 * <param="sysClkMhz">System Clock Frequency in Mhz</param>
 * <param="I2CClkKHz">Desired I2C Clock Frequency in KHz</param>
 */
static inline void SetClockDivides(float32 sysClkMHz, float32 I2CClkKHz);

/*
 * <summary>
 *                  Initializes the I2C to run in Master Mode for a One-To-One connection
 * </summary>
```

```
 * <param="slaveAddress">Address of the slave device to write to</param>
 * <param="sysClkMhz">System Clock Frequency in Mhz</param>
 * <param="I2CClkKHz">Desired I2C Clock Frequency in KHz</param>
 */
void I2C_O2O_Master_Init(Uint16 slaveAddress, float32 sysClkMhz, float32 I2CClkKHz)
{
		// Init GPIO
		InitI2CGpio();

		EALLOW;

		// Enable Clock for I2C
		SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 1;

		// Put I2C into Reset Mode
		I2caRegs.I2CMDR.bit.IRS = 0;

		// Set Slave Address
		I2caRegs.I2CSAR = slaveAddress;

		// Set Clocks
		SetClockDivides(sysClkMhz, I2CClkKHz);

		// Release from Reset Mode
		I2caRegs.I2CMDR.bit.IRS = 1;

		EDIS;
}



/*
 * <summary>
 *		Sends bytes via I2C
 * </summary>
 * <param="values">Pointer to array of bytes to send</param>
 * <param-"length">Length of array</param>
 */
void I2C_O2O_SendBytes(Uint16 * const values, Uint16 length)
{
		// Set to Master, Repeat Mode, TRX, FREE, Start
		I2caRegs.I2CMDR.all = 0x66A0;
```

```
		// Write values to I2C
		for (Uint16 i = 0; i < length; i++)
		{
			// Wait if Transmit is not ready
			while(!I2caRegs.I2CSTR.bit.ARDY);
			I2caRegs.I2CDXR = values[i];
		}

		// Stop Bit
		I2caRegs.I2CMDR.bit.STP = 1;
}

/*
 * <summary>
 *		Calculates and sets the ClockDivides for the I2C Module
 * </summary>
 * <param="sysClkMhz">System Clock Frequency in Mhz</param>
 * <param="I2CClkKHz">Desired I2C Clock Frequency in KHz</param>
 */
static inline void SetClockDivides(float32 sysClkMHz, float32 I2CClkKHz)
{
		/* Calculate Module Clock Frequency - Must be between 7-12 MHz
		 * Module Clock Frequency = sysClkMhz/(IPSC + 1)
		 */
		Uint16 IPSC = (Uint16)(sysClkMHz/IdealModClockFrqMHz);

		/* Calculate Divide Downs for SCL
		 * FreqMClk = sysClkMHz/((IPSC + 1)[(ICCL + d) + (ICCH + d)])
		 *
		 * Assume an even clock size -> ICCH == ICCL
		 * ICCL = ICCH = sysclkMHz/(2000 * I2CClkKHz * (IPSC + 1)) - d
		 */

		// Find value for d
		Uint16 d = 5;

		if (IPSC < 2)
		{
			d++;
			if (IPSC < 1)
```

```
                {
                        d++;
                }
        }

        Uint16 ICCLH = (Uint16)(1000 * sysClkMHz/(2 * I2CClkKHz * (IPSC + 1)) - d);

        // Set values
        I2caRegs.I2CPSC.all = IPSC;
        I2caRegs.I2CCLKL = ICCLH;
        I2caRegs.I2CCLKH = ICCLH;
}



/*
 * <summary>
 *      Initializes the GPIO for the I2C
 * </summary>
 */
static void InitI2CGpio()
{

  EALLOW;
/* Enable internal pull-up for the selected pins */
// Pull-ups can be enabled or disabled disabled by the user.
// This will enable the pullups for the specified pins.
// Comment out other unwanted lines.

        GpioCtrlRegs.GPBPUD.bit.GPIO32 = 0;    // Enable pull-up for GPIO32 (SDAA)
        GpioCtrlRegs.GPBPUD.bit.GPIO33 = 0;          // Enable pull-up for GPIO33 (SCLA)

/* Set qualification for selected pins to asynch only */
// This will select asynch (no qualification) for the selected pins.
// Comment out other unwanted lines.

        GpioCtrlRegs.GPBQSEL1.bit.GPIO32 = 3;  // Asynch input GPIO32 (SDAA)
   GpioCtrlRegs.GPBQSEL1.bit.GPIO33 = 3;  // Asynch input GPIO33 (SCLA)

/* Configure SCI pins using GPIO regs*/
// This specifies which of the possible GPIO pins will be I2C functional pins.
// Comment out other unwanted lines.
```

```
        GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 1;   // Configure GPIO32 for SDAA operation
        GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 1;   // Configure GPIO33 for SCLA operation

    EDIS;
}
```

Timer1_Utils.c

```
// File:   Timer1_Utils.c
// Date:   02/24/2018
// Name:   Mark Schuster
// Class:  EEE4511C (DSP)

#include <DSP2833x_Device.h>
#include "DSP28x_Project.h"
#include "Timer1_Utils.h"
#include "Interrupt_ISRs.h"

void initTimer1(void){
    InitCpuTimers();
    ConfigCpuTimer(&CpuTimer1, 150, 1E5);
    EALLOW;
    PieVectTable.XINT13 = timer1Isr;
    IER |= M_INT13;
    EDIS;
    CpuTimer1.RegsAddr->TCR.bit.TSS = 0;
}
```

main_Part1.c

```
// File:   main_Part1.c
// Date:   02/24/2018
// Name:   Mark Schuster
// Class:  EEE4511C (DSP)

#include <DSP2833x_Device.h>
#include "DSP28x_Project.h"

#include "ADC_Utils.h"
#include "Interrupt_Utils.h"
#include "I2C_LCD_Utils.h"
#include "Timer1_Utils.h"
```

```
#include "Interrupt_ISRs.h"


// Simple enum for bool values.
enum{
   FALSE,
   TRUE,
};



// Function prototypes:



void initLEDSAndSwitches(void);


Uint16 main(void)
{
   // Disable the WDT, and init the phase lock loop.
   DisableDog();
   InitPll(10,3);


   initLEDSAndSwitches();
   initLCD();
   initADC();
   initInterrupts();
   initTimer1();


   while(TRUE);
   return 0;
}



void initLEDSAndSwitches(void){
   // Allow write to protected regs.
   EALLOW;
   GpioCtrlRegs.GPAMUX1.all &= 0x0003FFF0;
   GpioCtrlRegs.GPADIR.all  |= 0x00007F80;
   GpioCtrlRegs.GPAMUX1.all &= 0xFFFFF30F;
   GpioCtrlRegs.GPAPUD.all  &= 0xFFFFFF8B;
   GpioCtrlRegs.GPADIR.all  &= 0xFFFFFF8B;


   return;
```

```
};
```

main_Part2.c

```
// File:   main_Part2.c
// Date:   02/24/2018
// Name:   Mark Schuster
// Class:  EEE4511C (DSP)

#include <DSP2833x_Device.h>
#include "DSP28x_Project.h"
#include "AIC23.h"
#include "InitAIC23.h"
#include "ADC_Utils.h"
#include "Codec_Utils.h"
#include "Interrupt_Utils.h"
#include "I2C_LCD_Utils.h"
#include "Timer1_Utils.h"
#include "Interrupt_ISRs.h"

#define VMAP asm(" CLRC VMAP");

// Simple enum for bool values.
enum{
   FALSE,
   TRUE,
};



// Function prototypes:

void initSpi(void);

Uint16 main(void)
{
   // Disable the WDT, and init the phase lock loop.
   InitSysCtrl();

   InitMcBSPb();
   InitSPIA();
   InitAIC23();

   initCodec();
```

```
    EALLOW;
    EnableInterrupts();
    McbspbRegs.SPCR1.bit.RRST = 0;
    McbspbRegs.SPCR1.bit.RRST = 1;
    EDIS;

    while(TRUE);

    return 0;
}
```

1.5 Program Description;

The Ohmmeter simply takes in an analog value into the ADC input and outputs the value to the LCD.

The codec application simply exercised my patience and initialized the codec. It then performed an endless loop of outputting the audio input to the audio output.