```c
//#############################################################################
// FILE:   LAB4.c
//
// TITLE:  Lab4
//#############################################################################

// Included Files
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <limits.h>
#include "F28x_Project.h"
#include "driverlib.h"
#include "device.h"
#include "F28379dSerial.h"
#include "LEDPatterns.h"
#include "song.h"
#include "dsp.h"
#include "fpu32/fpu_rfft.h"

#define PI          3.1415926535897932384626433832795
#define TWOPI       6.283185307179586476925286766559
#define HALFPI      1.5707963267948966192313216916398
// The Launchpad's CPU Frequency set to 200 you should not change this value
#define LAUNCHPAD_CPU_FREQUENCY 200


// Interrupt Service Routines predefinition
__interrupt void cpu_timer0_isr(void);
__interrupt void cpu_timer1_isr(void);
__interrupt void cpu_timer2_isr(void);
__interrupt void SWI_isr(void);

// Count variables
uint32_t numTimer0calls = 0;
uint32_t numSWIcalls = 0;
extern uint32_t numRXA;
uint16_t UARTPrint = 0;
uint16_t LEDdisplaynum = 0;
// global variable use for lab4
int16_t adcd0result =0;
int16_t adcd1result =0;
int16_t adca2result =0;
int16_t adca3result =0;
int16_t adcb1result =0;
float volts0 = 0.0;
int32_t ADCD1count =0;
int32_t ADCA1count =0;
int32_t ADCB1count =0;
float xka2=0;
float xka3=0;
float yka2=0;
float yka3=0;
```

```
float xkb1 =0;
float ykb1=0;

//This function sets DACA to the voltage between 0V and 3V passed to this function.
//If outside 0V to 3V the output is saturated at 0V to 3V
//Example code
//float myu = 2.25;
//setDACA(myu); // DACA will now output 2.25 Volts
//JS DAC output functions
void setDACA(float dacouta0) {
    int16_t DACOutInt = 0;
    //JS divided by the resolution to scale volts to 0-4095
    DACOutInt = (dacouta0/3.0)*4096.0; // perform scaling of 0 – almost 3V to 0 -
4095
    if (DACOutInt > 4095) DACOutInt = 4095;
    if (DACOutInt < 0) DACOutInt = 0;
    DacaRegs.DACVALS.bit.DACVALS = DACOutInt;
}
void setDACB(float dacouta1) {
    int16_t DACOutInt = 0;
    //JS divided by the resolution to scale volts to 0-4095
    DACOutInt = (dacouta1/3.0)*4096.0; // perform scaling of 0 – almost 3V to 0 -
4095
    if (DACOutInt > 4095) DACOutInt = 4095;
    if (DACOutInt < 0) DACOutInt = 0;
    DacbRegs.DACVALS.bit.DACVALS = DACOutInt;
}

//xk is the current ADC reading, xk_1 is the ADC reading one millisecond ago, xk_2
two milliseconds ago, etc
float xk = 0;
float xk_1 = 0;
float xk_2 = 0;
float xk_3 = 0;
float xk_4 = 0;
//yk is the filtered value
float yk = 0;
//b is the filter coefficients
//JS defined b arrays, using values from matlab
//float b[5] = {3.3833240118424500e-02,
//              2.4012702387971543e-01,
//              4.5207947200372001e-01,
//              2.4012702387971543e-01,
//              3.3833240118424500e-02}; // 0.2 is 1/5th therefore a 5 point average
//float b[5] = {0.2,0.2,0.2,0.2,0.2};

//float b[] = {3.3833240118424500e-02,
//                            2.4012702387971543e-01,
//                            4.5207947200372001e-01,
//                            2.4012702387971543e-01,
//                            3.3833240118424500e-02};
//float b[22]={   -2.3890045153263611e-03,
//    -3.3150057635348224e-03,
//    -4.6136191242627002e-03,
//    -4.1659855521681268e-03,
```

```
//    1.4477422497795286e-03,
//    1.5489414225159667e-02,
//    3.9247886844071371e-02,
//    7.0723964095458614e-02,
//    1.0453473887246176e-01,
//    1.3325672639406205e-01,
//    1.4978314227429904e-01,
//    1.4978314227429904e-01,
//    1.3325672639406205e-01,
//    1.0453473887246176e-01,
//    7.0723964095458614e-02,
//    3.9247886844071371e-02,
//    1.5489414225159667e-02,
//    1.4477422497795286e-03,
//    -4.1659855521681268e-03,
//    -4.6136191242627002e-03,
//    -3.3150057635348224e-03,
//    -2.3890045153263611e-03};

//float b[32]={   -6.3046914864397922e-04,
//    -1.8185681242784432e-03,
//    -2.5619416124584822e-03,
//    -1.5874939943956356e-03,
//    2.3695126689747326e-03,
//    8.3324969783531780e-03,
//    1.1803612855040625e-02,
//    6.7592967793297151e-03,
//    -9.1745119977290398e-03,
//    -2.9730906886035850e-02,
//    -3.9816452266421651e-02,
//    -2.2301647638687881e-02,
//    3.1027965907247105e-02,
//    1.1114350049251465e-01,
//    1.9245540210070616e-01,
//    2.4373020388648489e-01,
//    2.4373020388648489e-01,
//    1.9245540210070616e-01,
//    1.1114350049251465e-01,
//    3.1027965907247105e-02,
//    -2.2301647638687881e-02,
//    -3.9816452266421651e-02,
//    -2.9730906886035850e-02,
//    -9.1745119977290398e-03,
//    6.7592967793297151e-03,
//    1.1803612855040625e-02,
//    8.3324969783531780e-03,
//    2.3695126689747326e-03,
//    -1.5874939943956356e-03,
//    -2.5619416124584822e-03,
//    -1.8185681242784432e-03,
//    -6.3046914864397922e-04};

float b[101]={  -2.9880028485706014e-18,
    1.0014161157911048e-04,
    -5.2618551547237234e-04,
```

-7.7962413218288327e-04,
3.8326299947443642e-04,
1.4468607075027277e-03,
4.7641201701853882e-04,
-1.1966077184856539e-03,
-9.8442612457462801e-04,
2.2426543490299702e-04,
-1.3055511470703196e-18,
-2.8856900257159991e-04,
1.6267183735412948e-03,
2.5290719210008640e-03,
-1.2795608153376812e-03,
-4.8925823855322497e-03,
-1.6113987990414897e-03,
4.0107106316287265e-03,
3.2480258555224279e-03,
-7.2516689926356299e-04,
1.8056386897280851e-17,
8.9007602241939149e-04,
-4.8948933522602461e-03,
-7.4262043229105694e-03,
3.6693671696644108e-03,
1.3718600126208420e-02,
4.4244260206582689e-03,
-1.0801717429888814e-02,
-8.5964585113028728e-03,
1.8899182131726397e-03,
-5.3057330072547164e-18,
-2.2642270131377727e-03,
1.2346792985290733e-02,
1.8623150986016437e-02,
-9.1751382337289834e-03,
-3.4312641521119479e-02,
-1.1108970111065547e-02,
2.7336258194538145e-02,
2.2030316909781834e-02,
-4.9314814569541088e-03,
7.0946687196014922e-18,
6.2561782550849951e-03,
-3.5604150484977087e-02,
-5.6784697958734962e-02,
3.0104607026376420e-02,
1.2416425489455964e-01,
4.5989918232812586e-02,
-1.3743489446447751e-01,
-1.5046109818463846e-01,
6.0593548159411273e-02,
1.9952895080570546e-01,
6.0593548159411273e-02,
-1.5046109818463846e-01,
-1.3743489446447751e-01,
4.5989918232812586e-02,
1.2416425489455964e-01,
3.0104607026376420e-02,
-5.6784697958734962e-02,

```
    -3.5604150484977087e-02,
    6.2561782550849951e-03,
    7.0946687196014922e-18,
    -4.9314814569541088e-03,
    2.2030316909781834e-02,
    2.7336258194538145e-02,
    -1.1108970111065547e-02,
    -3.4312641521119479e-02,
    -9.1751382337289834e-03,
    1.8623150986016437e-02,
    1.2346792985290733e-02,
    -2.2642270131377727e-03,
    -5.3057330072547164e-18,
    1.8899182131726397e-03,
    -8.5964585113028728e-03,
    -1.0801717429888814e-02,
    4.4244260206582689e-03,
    1.3718600126208420e-02,
    3.6693671696644108e-03,
    -7.4262043229105694e-03,
    -4.8948933522602461e-03,
    8.9007602241939149e-04,
    1.8056386897280851e-17,
    -7.2516689926356299e-04,
    3.2480258555224279e-03,
    4.0107106316287265e-03,
    -1.6113987990414897e-03,
    -4.8925823855322497e-03,
    -1.2795608153376812e-03,
    2.5290719210008640e-03,
    1.6267183735412948e-03,
    -2.8856900257159991e-04,
    -1.3055511470703196e-18,
    2.2426543490299702e-04,
    -9.8442612457462801e-04,
    -1.1966077184856539e-03,
    4.7641201701853882e-04,
    1.4468607075027277e-03,
    3.8326299474436420e-04,
    -7.7962413218288327e-04,
    -5.2618551547237234e-04,
    1.0014161157911048e-04,
    -2.9880028485706014e-18};

//JS define arrays use to calculate filter signals
float volts1 = 0.0;
float xkarray[22]={};
float xka2array[22]={};
float xka3array[22]={};
float xkb1array[101]={};
//adcd1 pie interrupt
__interrupt void ADCD_ISR (void) {
    adcd0result = AdcdResultRegs.ADCRESULT0;
    adcd1result = AdcdResultRegs.ADCRESULT1;
    // Here covert ADCIND0, ADCIND1 to volts
```

```
    //JS convert to volts by multiply by resolution
    xk = (adcd0result/4096.0)*3.0;
    //JS convert to volts by multiply by resolution
    volts1 = (adcd1result/4096.0)*3.0;
    //JS set yk back to 0, avoid keeping on adding
    yk = 0;
    //JS implement a for loop to use previous xk state
    for (int i = 21; i>0; i--){
        //JS set xk values from the previous index xk values
        xkarray[i]=xkarray[i-1];
        //JS sum xk values * filter coefficients except the first value of array
        yk += xkarray[i]*b[i];
    }
    //JS set the first array value to xk
    xkarray[0]=xk;
    //JS add 1st xk value*the first filter coefficient
    yk += xkarray[0]*b[0];
    //yk = b[0]*xk + b[1]*xk_1 + b[2]*xk_2 + b[3]*xk_3 + b[4]*xk_4;

    //Save past states before exiting from the function so that next sample they are
the older state
//    xk_4 = xk_3;
//    xk_3 = xk_2;
//    xk_2 = xk_1;
//    xk_1 = xk;
    // Here write yk to DACA channel
    setDACA(yk);
    // Print ADCIND0 and ADCIND1's voltage value to TeraTerm every 100ms
    //JS increase ADCD1count with 1ms
    ADCD1count++;
    //JS print to TeraTerm with 100ms
    if (ADCD1count % 100 == 0){
        UARTPrint = 1;
    }
    AdcdRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}


__interrupt void ADCA_ISR (void) {
    adca2result = AdcaResultRegs.ADCRESULT0;
    adca3result = AdcaResultRegs.ADCRESULT1;
    //JS similar to ADCD, but uses two xkarrays and yks,  two signals input in filter
    xka2 = (adca2result/4096.0)*3.0;
    xka3 = (adca3result/4096.0)*3.0;
    yka2 = 0;
    yka3=0;
    for (int i = 21; i>0; i--){
        xka2array[i]=xka2array[i-1];
        xka3array[i]=xka3array[i-1];
        yka2 += xka2array[i]*b[i];
        yka3 += xka3array[i]*b[i];
    }
    xka2array[0]=xka2;
    xka3array[0]=xka3;
```

```
    yka2 += xka2array[0]*b[0];
    yka3 += xka3array[0]*b[0];
    // Here write yk to DACA channel
//    setDACA(yk);
//
    // Print  voltage value to TeraTerm every 100ms
    ADCA1count++;
    if (ADCA1count % 100 == 0){
        UARTPrint = 1;
    }
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

__interrupt void ADCB_ISR (void) {
    //JS set GPIO52 as an output
    GpioDataRegs.GPBSET.bit.GPIO52 = 1;
    //JS similar to ADCD, but use 100 th order FIR filter
    adcb1result = AdcbResultRegs.ADCRESULT0;
    xkb1 = (adcb1result/4096.0)*3.0;
    ykb1 = 0;
    for (int i = 100; i>0; i--){
        xkb1array[i]=xkb1array[i-1];
        ykb1 += xkb1array[i]*b[i];
    }
    xkb1array[0]=xkb1;
    ykb1 += xkb1array[0]*b[0];
    // Here write yk to DACA channel
    //JS add 1.5 oscilloscope offset
    setDACA(ykb1+1.5);
//
    // Print  voltage value to TeraTerm every 100ms
    ADCB1count++;
    if (ADCB1count % 100 == 0){
        UARTPrint = 1;
    }
    AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
    //JS turn off GPIO52
    GpioDataRegs.GPBCLEAR.bit.GPIO52 = 1;
}

void main(void)
{
    // PLL, WatchDog, enable Peripheral Clocks
    // This example function is found in the F2837xD_SysCtrl.c file.
    InitSysCtrl();

    InitGpio();

    GPIO_SetupPinMux(52, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(52, GPIO_OUTPUT, GPIO_PUSHPULL);

      // Blue LED on LaunchPad
    GPIO_SetupPinMux(31, GPIO_MUX_CPU1, 0);
```

```
GPIO_SetupPinOptions(31, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO31 = 1;

   // Red LED on LaunchPad
GPIO_SetupPinMux(34, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(34, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPBSET.bit.GPIO34 = 1;

   // LED1 and PWM Pin
GPIO_SetupPinMux(22, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(22, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;

   // LED2
GPIO_SetupPinMux(94, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(94, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPCCLEAR.bit.GPIO94 = 1;

   // LED3
GPIO_SetupPinMux(95, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(95, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPCCLEAR.bit.GPIO95 = 1;

   // LED4
GPIO_SetupPinMux(97, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(97, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPDCLEAR.bit.GPIO97 = 1;

   // LED5
GPIO_SetupPinMux(111, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(111, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPDCLEAR.bit.GPIO111 = 1;

   // LED6
GPIO_SetupPinMux(130, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(130, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPECLEAR.bit.GPIO130 = 1;

   // LED7
GPIO_SetupPinMux(131, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(131, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPECLEAR.bit.GPIO131 = 1;

   // LED8
GPIO_SetupPinMux(25, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(25, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPACLEAR.bit.GPIO25 = 1;

   // LED9
GPIO_SetupPinMux(26, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(26, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPACLEAR.bit.GPIO26 = 1;

   // LED10
GPIO_SetupPinMux(27, GPIO_MUX_CPU1, 0);
```

```
GPIO_SetupPinOptions(27, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;

   // LED11
GPIO_SetupPinMux(60, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(60, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPBCLEAR.bit.GPIO60 = 1;

   // LED12
GPIO_SetupPinMux(61, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(61, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1;

   // LED13
GPIO_SetupPinMux(157, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(157, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPECLEAR.bit.GPIO157 = 1;

   // LED14
GPIO_SetupPinMux(158, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(158, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPECLEAR.bit.GPIO158 = 1;

   // LED15
GPIO_SetupPinMux(159, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(159, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPECLEAR.bit.GPIO159 = 1;

   // LED16
GPIO_SetupPinMux(160, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(160, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPFCLEAR.bit.GPIO160 = 1;

//WIZNET Reset
GPIO_SetupPinMux(0, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(0, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO0 = 1;

//ESP8266 Reset
GPIO_SetupPinMux(1, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(1, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO1 = 1;

   //SPIRAM  CS  Chip Select
GPIO_SetupPinMux(19, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(19, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO19 = 1;

//DRV8874 #1 DIR  Direction
GPIO_SetupPinMux(29, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(29, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO29 = 1;

//DRV8874 #2 DIR  Direction
GPIO_SetupPinMux(32, GPIO_MUX_CPU1, 0);
```

```
GPIO_SetupPinOptions(32, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPBSET.bit.GPIO32 = 1;

//DAN28027  CS  Chip Select
GPIO_SetupPinMux(9, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(9, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO9 = 1;

//MPU9250  CS  Chip Select
GPIO_SetupPinMux(66, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(66, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;

   //WIZNET  CS  Chip Select
GPIO_SetupPinMux(125, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(125, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPDSET.bit.GPIO125 = 1;

//PushButton 1
GPIO_SetupPinMux(4, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(4, GPIO_INPUT, GPIO_PULLUP);

//PushButton 2
GPIO_SetupPinMux(5, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(5, GPIO_INPUT, GPIO_PULLUP);

//PushButton 3
GPIO_SetupPinMux(6, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(6, GPIO_INPUT, GPIO_PULLUP);

//PushButton 4
GPIO_SetupPinMux(7, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(7, GPIO_INPUT, GPIO_PULLUP);

   //Joy Stick Pushbutton
GPIO_SetupPinMux(8, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(8, GPIO_INPUT, GPIO_PULLUP);


// Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the F2837xD_PieCtrl.c file.
InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
IER = 0x0000;
IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
```

```
    // This will populate the entire table, even if the interrupt
    // is not used in this example.  This is useful for debug purposes.
    // The shell ISR routines are found in F2837xD_DefaultIsr.c.
    // This function is found in F2837xD_PieVect.c.
    InitPieVectTable();

    // Interrupts that are used in this example are re-mapped to
    // ISR functions found within this project
    EALLOW;  // This is needed to write to EALLOW protected registers
    PieVectTable.TIMER0_INT = &cpu_timer0_isr;
    PieVectTable.TIMER1_INT = &cpu_timer1_isr;
    PieVectTable.TIMER2_INT = &cpu_timer2_isr;
    //JS for ADCD to call the memory address of ADCD_ISR
//    PieVectTable.ADCD1_INT = &ADCD_ISR;
    //JS for ADCA to call the memory address of ADCA_ISR
//    PieVectTable.ADCA1_INT = &ADCA_ISR;
    //JS for ADCB to call the memory address of ADCB_ISR
    PieVectTable.ADCB1_INT = &ADCB_ISR;
    PieVectTable.SCIA_RX_INT = &RXAINT_recv_ready;
    PieVectTable.SCIB_RX_INT = &RXBINT_recv_ready;
    PieVectTable.SCIC_RX_INT = &RXCINT_recv_ready;
    PieVectTable.SCID_RX_INT = &RXDINT_recv_ready;
    PieVectTable.SCIA_TX_INT = &TXAINT_data_sent;
    PieVectTable.SCIB_TX_INT = &TXBINT_data_sent;
    PieVectTable.SCIC_TX_INT = &TXCINT_data_sent;
    PieVectTable.SCID_TX_INT = &TXDINT_data_sent;

    PieVectTable.EMIF_ERROR_INT = &SWI_isr;
    EDIS;    // This is needed to disable write to EALLOW protected registers


    // Initialize the CpuTimers Device Peripheral. This function can be
    // found in F2837xD_CpuTimers.c
    InitCpuTimers();

    // Configure CPU-Timer 0, 1, and 2 to interrupt every given period:
    // 200MHz CPU Freq,                       Period (in uSeconds)
    ConfigCpuTimer(&CpuTimer0, LAUNCHPAD_CPU_FREQUENCY, 10000);
    ConfigCpuTimer(&CpuTimer1, LAUNCHPAD_CPU_FREQUENCY, 20000);
    ConfigCpuTimer(&CpuTimer2, LAUNCHPAD_CPU_FREQUENCY, 40000);

    // Enable CpuTimer Interrupt bit TIE
    CpuTimer0Regs.TCR.all = 0x4000;
    CpuTimer1Regs.TCR.all = 0x4000;
    CpuTimer2Regs.TCR.all = 0x4000;

      init_serialSCIA(&SerialA,115200);

      EALLOW;
      EPwm5Regs.ETSEL.bit.SOCAEN = 0; // Disable SOC on A group
      EPwm5Regs.TBCTL.bit.CTRMODE = 3; // freeze counter
      //JS only bit 1 is high,Enable event time-base counter equal to period
      EPwm5Regs.ETSEL.bit.SOCASEL = 2; // Select Event when counter equal to PRD
      //JS when only bit 0 is hig, Generate the EPWM5SOCA pulse on the first event
```

```
	EPwm5Regs.ETPS.bit.SOCAPRD = 1; // Generate pulse on 1st event ("pulse" is the same as "trigger")
	EPwm5Regs.TBCTR = 0x0; // Clear counter
	EPwm5Regs.TBPHS.bit.TBPHS = 0x0000; // Phase is 0
	EPwm5Regs.TBCTL.bit.PHSEN = 0; // Disable phase loading
	EPwm5Regs.TBCTL.bit.CLKDIV = 0; // divide by 1 50Mhz Clock
	//JS for exerise 1, set period to 1ms 50MHz * 0.001s
//	EPwm5Regs.TBPRD = 50000; // Set Period to 1ms sample. Input clock is 50MHz.
	//
//	EPwm5Regs.TBPRD = 12500;
    EPwm5Regs.TBPRD = 5000;
	// Notice here that we are not setting CMPA or CMPB because we are not using the PWM signal
	EPwm5Regs.ETSEL.bit.SOCAEN = 1; //enable SOCA
	//JS up-count mode, set it to 0
	EPwm5Regs.TBCTL.bit.CTRMODE = 0; //unfreeze, and enter up count mode
	EDIS;

	EALLOW;
	//write configurations for all ADCs ADCA, ADCB, ADCC, ADCD
	AdcaRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
	AdcbRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
	AdccRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
	AdcdRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
	AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
	AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
	AdcSetMode(ADC_ADCC, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
	AdcSetMode(ADC_ADCD, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
	//Set pulse positions to late
	AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
	AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
	AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;
	AdcdRegs.ADCCTL1.bit.INTPULSEPOS = 1;
	//power up the ADCs
	AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
	AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
	AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;
	AdcdRegs.ADCCTL1.bit.ADCPWDNZ = 1;
	//delay for 1ms to allow ADC time to power up
	DELAY_US(1000);
	//Select the channels to convert and end of conversion flag
	//Many statements commented out, To be used when using ADCA or ADCB
	//ADCA
	//JS set SOC0 to pin2
	AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2; //SOC0 will convert Channel you choose Does not have to be A0
	AdcaRegs.ADCSOC0CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK cycles = 500ns
	//JS EPWM5 ADCSOCA is 13
	AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 13;// EPWM5 ADCSOCA or another trigger you choose will trigger SOC0
```

```
	//JS set SOC1 to pin3
	AdcaRegs.ADCSOC1CTL.bit.CHSEL = 3; //SOC1 will convert Channel you choose Does not have to be A1
	AdcaRegs.ADCSOC1CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK cycles = 500ns
	//JS EPWM5 ADCSOCA is 13
	AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 13;// EPWM5 ADCSOCA or another trigger you choose will trigger SOC1
	//JS set to the last converted SOC1
	AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //set to last SOC that is converted and it will set INT1 flag ADCA1
	AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1; //enable INT1 flag
	AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared
	//ADCB
	//JS set SOC0 to pin4
	AdcbRegs.ADCSOC0CTL.bit.CHSEL = 4; //SOC0 will convert Channel you choose Does not have to be B0
	AdcbRegs.ADCSOC0CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK cycles = 500ns
	//JS EPWM5 ADCSOCA is 13
	AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 13; // EPWM5 ADCSOCA or another trigger you choose will trigger SOC0
//	AdcbRegs.ADCSOC1CTL.bit.CHSEL = ???; //SOC1 will convert Channel you choose Does not have to be B1
//	AdcbRegs.ADCSOC1CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK cycles = 500ns
//	AdcbRegs.ADCSOC1CTL.bit.TRIGSEL = ???; // EPWM5 ADCSOCA or another trigger you choose will trigger SOC1
//	AdcbRegs.ADCSOC2CTL.bit.CHSEL = ???; //SOC2 will convert Channel you choose Does not have to be B2
//	AdcbRegs.ADCSOC2CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK cycles = 500ns
//	AdcbRegs.ADCSOC2CTL.bit.TRIGSEL = ???; // EPWM5 ADCSOCA or another trigger you choose will trigger SOC2
//	AdcbRegs.ADCSOC3CTL.bit.CHSEL = ???; //SOC3 will convert Channel you choose Does not have to be B3
//	AdcbRegs.ADCSOC3CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK cycles = 500ns
//	AdcbRegs.ADCSOC3CTL.bit.TRIGSEL = ???; // EPWM5 ADCSOCA or another trigger you choose will trigger SOC3
	//JS set to the last converted SOC0
	AdcbRegs.ADCINTSEL1N2.bit.INT1SEL = 0; //set to last SOC that is converted and it will set INT1 flag ADCB1
	AdcbRegs.ADCINTSEL1N2.bit.INT1E = 1; //enable INT1 flag
	AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared

	//ADCD
	//JS use only SOC0 and SOC1 for ADCD
	//JS set SOC0 to pin0
	AdcdRegs.ADCSOC0CTL.bit.CHSEL = 0; // set SOC0 to convert pin D0
	AdcdRegs.ADCSOC0CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK cycles = 500ns
	//JS EPWM5 ADCSOCA is 13
	AdcdRegs.ADCSOC0CTL.bit.TRIGSEL = 13; // EPWM5 ADCSOCA will trigger SOC0
	//JS set SOC1 to pin1
```

```
	AdcdRegs.ADCSOC1CTL.bit.CHSEL = 1; //set SOC1 to convert pin D1
	AdcdRegs.ADCSOC1CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK cycles
= 500ns
	//JS EPWM5 ADCSOCA is 13
	AdcdRegs.ADCSOC1CTL.bit.TRIGSEL = 13; // EPWM5 ADCSOCA will trigger SOC1
	//AdcdRegs.ADCSOC2CTL.bit.CHSEL = ???; //set SOC2 to convert pin D2
	//AdcdRegs.ADCSOC2CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK
cycles = 500ns
	//AdcdRegs.ADCSOC2CTL.bit.TRIGSEL = ???; // EPWM5 ADCSOCA will trigger SOC2
	//AdcdRegs.ADCSOC3CTL.bit.CHSEL = ???; //set SOC3 to convert pin D3
	//AdcdRegs.ADCSOC3CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK
cycles = 500ns
	//AdcdRegs.ADCSOC3CTL.bit.TRIGSEL = ???; // EPWM5 ADCSOCA will trigger SOC3
	//JS set to the last converted SOC1
	AdcdRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //set to SOC1, the last converted, and
it will set INT1 flag ADCD1
	AdcdRegs.ADCINTSEL1N2.bit.INT1E = 1; //enable INT1 flag
	AdcdRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared
	EDIS;

	//JS initialize the DACs copied from guideline
	// Enable DACA and DACB outputs
	EALLOW;
	DacaRegs.DACOUTEN.bit.DACOUTEN = 1; //enable dacA output-->uses ADCINA0
	DacaRegs.DACCTL.bit.LOADMODE = 0; //load on next sysclk
	DacaRegs.DACCTL.bit.DACREFSEL = 1; //use ADC VREF as reference voltage
	DacbRegs.DACOUTEN.bit.DACOUTEN = 1; //enable dacB output-->uses ADCINA1
	DacbRegs.DACCTL.bit.LOADMODE = 0; //load on next sysclk
	DacbRegs.DACCTL.bit.DACREFSEL = 1; //use ADC VREF as reference voltage
	EDIS;



    // Enable CPU int1 which is connected to CPU-Timer 0, CPU int13
    // which is connected to CPU-Timer 1, and CPU int 14, which is connected
    // to CPU-Timer 2:  int 12 is for the SWI.
    IER |= M_INT1;
    IER |= M_INT8;  // SCIC SCID
    IER |= M_INT9;  // SCIA
    IER |= M_INT12;
    IER |= M_INT13;
    IER |= M_INT14;

    // Enable TINT0 in the PIE: Group 1 interrupt 7
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
	// Enable SWI in the PIE: Group 12 interrupt 9
    PieCtrlRegs.PIEIER12.bit.INTx9 = 1;
    //JS Enable ADCD1 in the PIE: Group 1 interrupt 6
    //PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
    //JS Enable ADCA1 in the PIE: Group 1 interrupt 1
//	PieCtrlRegs.PIEIER1.bit.INTx1 = 1;
    //JS Enable ADCB1 in the PIE: Group 1 interrupt 2
    PieCtrlRegs.PIEIER1.bit.INTx2 = 1;

	init_serialSCIC(&SerialC,115200);
```

```
        init_serialSCID(&SerialD,115200);
    // Enable global Interrupts and higher priority real-time debug events
    EINT;  // Enable Global interrupt INTM
    ERTM;  // Enable Global realtime interrupt DBGM


    // IDLE loop. Just sit and loop forever (optional):
    while(1)
    {
        if (UARTPrint == 1 ) {
                    //serial_printf(&SerialA,"Num Timer2:%ld Num SerialRX:
%ld\r\n",CpuTimer2.InterruptCount,numRXA);
            //JS print ADCD0 volt to TeraTerm
            //serial_printf(&SerialA,"ADCD0 voltage= %.3f\r\n", volts0);
            //JS print filter ADCD0 and ADCD1 to TeraTerm
//              serial_printf(&SerialA,"ADCD0 voltage= %.3f ADCD1 voltage= %.3f\r\n",
xk, volts1);
            //JS print filter values from pin2 and pin3
            serial_printf(&SerialA,"filter value 2= %.3f filter value 3= %.3f\r\n",
yka2, yka3);
            UARTPrint = 0;
        }
    }
}


// SWI_isr,  Using this interrupt as a Software started interrupt
__interrupt void SWI_isr(void) {

    // These three lines of code allow SWI_isr, to be interrupted by other interrupt
functions
        // making it lower priority than all other Hardware interrupts.
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
    asm("       NOP");                    // Wait one cycle
    EINT;                                 // Clear INTM to enable interrupts



    // Insert SWI ISR Code here.......


    numSWIcalls++;

    DINT;

}

// cpu_timer0_isr - CPU Timer0 ISR
__interrupt void cpu_timer0_isr(void)
{
    CpuTimer0.InterruptCount++;

    numTimer0calls++;

//    if ((numTimer0calls%50) == 0) {
```

```
//        PieCtrlRegs.PIEIFR12.bit.INTx9 = 1;  // Manually cause the interrupt for
the SWI
//    }

    if ((numTimer0calls%25) == 0) {
        displayLEDletter(LEDdisplaynum);
        LEDdisplaynum++;
        if (LEDdisplaynum == 0xFFFF) {  // prevent roll over exception
            LEDdisplaynum = 0;
        }
    }

    if ((numTimer0calls%50) == 0) {
            // Blink LaunchPad Red LED
            GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;
    }


    // Acknowledge this interrupt to receive more interrupts from group 1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

// cpu_timer1_isr - CPU Timer1 ISR
__interrupt void cpu_timer1_isr(void)
{

    CpuTimer1.InterruptCount++;
}

// cpu_timer2_isr CPU Timer2 ISR
__interrupt void cpu_timer2_isr(void)
{
     // Blink LaunchPad Blue LED
    GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;

    CpuTimer2.InterruptCount++;

     if ((CpuTimer2.InterruptCount % 10) == 0) {
            UARTPrint = 1;
     }
}
```