```c
//###########################################################################
// FILE:   LABstarter_main.c
//
// TITLE:  Lab Starter
//###########################################################################

// 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          3.14159265358979323846264338327795
#define TWOPI       6.28318530717958647692528676655900
#define HALFPI      1.57079632679489661923132169163980
// 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;

// JS create a global variable
int32_t numTimer2calls = 0;
int16_t checkLEDs = 0;
int16_t checkButtons = 0;

float x1 = 6.0;
float x2 = 2.3;
float x3 = 7.3;
float x4 = 7.1;
void setLEDRowsOnOff (int16_t);
int16_t ReadPushButtons(void);
void main(void)
```

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

    InitGpio();

       // 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;
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);
// JS Change the period to 250000 for exercise 1, and then to 1000 for exercise 4
ConfigCpuTimer(&CpuTimer2, LAUNCHPAD_CPU_FREQUENCY, 1000);

// Enable CpuTimer Interrupt bit TIE
// JS CPU timer 0 & 1 out
//CpuTimer0Regs.TCR.all = 0x4000;
//CpuTimer1Regs.TCR.all = 0x4000;
CpuTimer2Regs.TCR.all = 0x4000;

   init_serialSCIA(&SerialA,115200);

// 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;

      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 ) {
            // JS add 32 bit numTimer2calls, 16 bit int checkLEDs pass to
SetLEDRowsOnOff, 16 bit int checkButtons returned by ReadPushButtons to serial_print
with coresponding formatters %ld for 32 bit int, %d for 16 bit int

                    serial_printf(&SerialA,"Num Timer2:%ld Num SerialRX: %ld,
numTimer2calls:%ld, checkLEDs:%d,
ReadPushButton:%d\r\n",CpuTimer2.InterruptCount,numRXA,numTimer2calls,checkLEDs,
checkButtons);
            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)
{
    //JS global variables change to see a fact of breakpoints
       x4 = x3 +2.0;
       x3 = x4 + 1.3;
       x1 = 9* x2;
       x2 = 34* x3;
// JS for exercise 1, increment every time cpu timer 2 is called
    numTimer2calls++;
       // test for small period printing UARTPrint = 1;
    // Blink LaunchPad Blue LED
    GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;
    CpuTimer2.InterruptCount++;
```

```
    // JS for exercise 1, to turn on LED rows based on button pressed
    //setLEDRowsOnOff(checkButtons);

    // JS for exercise 2, add bitwise and to check if button 2 and 3 are press, if
so, stop incrementing global count integer.
    // JS checkButtons is 16 bit int returned from ReadPushButtons()
    checkButtons= ReadPushButtons();

    if(!(checkButtons & 0x6)){
        checkLEDs++;
    }

// JS change to % 100 to slow down blinking LEDs and serial_print every 100 times to
CPU Timer 2 interrupt
      if ((CpuTimer2.InterruptCount % 100) == 0) {
            UARTPrint = 1;
            setLEDRowsOnOff(checkLEDs);
      }
}
// JS put in 16 bit int rows
// JS If rows is high at bit 0, then (rows & 0x1) will equal 0x1 and 1st row of LED
will be turned on using GpioDataRegs.GPASET.bit.GPIO? from LEDPatterns.c
// JS otherwise, turn off
void setLEDRowsOnOff (int16_t rows)
{
    if ((rows & 0x1) ==0x1){
        GpioDataRegs.GPASET.bit.GPIO22 = 1;
        GpioDataRegs.GPESET.bit.GPIO130 = 1;
        GpioDataRegs.GPBSET.bit.GPIO60 = 1;
    }
    else{
        GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;
        GpioDataRegs.GPECLEAR.bit.GPIO130 = 1;
        GpioDataRegs.GPBCLEAR.bit.GPIO60 = 1;
    }
    // JS If rows is high at bit 1, then (rows & 0x2) will equal 0x2 and 2nd row of
LED will be turned on using GpioDataRegs.GPASET.bit.GPIO? from LEDPatterns.c
    // JS otherwise, turn off
    if ((rows & 0x2) == 0x2){
        GpioDataRegs.GPCSET.bit.GPIO94 = 1;
        GpioDataRegs.GPESET.bit.GPIO131 = 1;
        GpioDataRegs.GPBSET.bit.GPIO61 = 1;
    }
    else{
        GpioDataRegs.GPCCLEAR.bit.GPIO94 = 1;
        GpioDataRegs.GPECLEAR.bit.GPIO131 = 1;
        GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1;
    }
    // JS If rows is high at bit 2, then (rows & 0x4) will equal 0x4 and 3rd row of
LED will be turned on using GpioDataRegs.GPASET.bit.GPIO? from LEDPatterns.c
    // JS otherwise, turn off
    if ((rows & 0x4) == 0x4){
        GpioDataRegs.GPCSET.bit.GPIO95 = 1;
        GpioDataRegs.GPASET.bit.GPIO25 = 1;
        GpioDataRegs.GPESET.bit.GPIO157 = 1;
```

```
    }
    else{
        GpioDataRegs.GPCCLEAR.bit.GPIO95 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO25 = 1;
        GpioDataRegs.GPECLEAR.bit.GPIO157 = 1;
    }
    // JS If rows is high at bit 3, then (rows & 0x8) will equal 0x8 and 4th row of
LED will be turned on using GpioDataRegs.GPASET.bit.GPIO? from LEDPatterns.c
    // JS otherwise, turn off
    if ((rows & 0x8) == 0x8){
        GpioDataRegs.GPDSET.bit.GPIO97 = 1;
        GpioDataRegs.GPASET.bit.GPIO26 = 1;
        GpioDataRegs.GPESET.bit.GPIO158 = 1;
    }
    else{
        GpioDataRegs.GPDCLEAR.bit.GPIO97 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO26 = 1;
        GpioDataRegs.GPECLEAR.bit.GPIO158 = 1;
    }
    // JS If rows is high at bit 4, then (rows & 0x10) will equal 0x10 and 5th row of
LED will be turned on using GpioDataRegs.GPASET.bit.GPIO? from LEDPatterns.c
    // JS otherwise, turn off
    if ((rows & 0x10) == 0x10){
        GpioDataRegs.GPDSET.bit.GPIO111 = 1;
        GpioDataRegs.GPASET.bit.GPIO27 = 1;
        GpioDataRegs.GPESET.bit.GPIO159 = 1;
    }
    else{
        GpioDataRegs.GPDCLEAR.bit.GPIO111 = 1;
        GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
        GpioDataRegs.GPECLEAR.bit.GPIO159 = 1;
    }


}
// JS get 16 bit int Buttons

int16_t ReadPushButtons(void)
{
    int16_t Buttons = 0;
    // JS If button 1 is pressed, GPIO connects to ground, and read 0
    // JS Buttons|0x1 turn bit 0 of Buttons high
    if (GpioDataRegs.GPADAT.bit.GPIO4 == 0){
        Buttons = Buttons|0x1;

    }
    // JS If button 2 is pressed, GPIO connects to ground, and read 0
    // JS Buttons|0x2 turn bit 1 of Buttons high
    if (GpioDataRegs.GPADAT.bit.GPIO5 == 0){
        Buttons = Buttons|0x2;
    }
    // JS If button 3 is pressed, GPIO connects to ground, and read 0
    // JS Buttons|0x4 turn bit 2 of Buttons high
    if (GpioDataRegs.GPADAT.bit.GPIO6 == 0){
        Buttons = Buttons|0x4;
```

```
    }
    // JS If button 4 is pressed, GPIO connects to ground, and read 0
    // JS Buttons|0x8 turn bit 3 of Buttons high
    if (GpioDataRegs.GPADAT.bit.GPIO7 == 0){
        Buttons = Buttons|0x8;
    }
    return Buttons;
}
```