//#############################################################################

// 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 3.1415926535897932384626433832795

**#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;

**float** sinvalue = 0;

**float** time = 0;

**float** ampl = 3.0;

**float** frequency = 0.05;

**float** offset = 0.25;

int32\_t timeint = 0;

**float** satvalue = 0;

**float** **saturate**(**float** input, **float** saturation\_limit);

**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);

**ConfigCpuTimer**(&CpuTimer2, LAUNCHPAD\_CPU\_FREQUENCY, 50000); // Changed the period to 50ms

// Enable CpuTimer Interrupt bit TIE

CpuTimer0Regs.TCR.all = 0x4000;

CpuTimer1Regs.TCR.all = 0x4000;

CpuTimer2Regs.TCR.all = 0x4000;

init\_serialSCIA(&SerialA,115200);

// init\_serialSCIB(&SerialB,115200);

// init\_serialSCIC(&SerialC,115200);

// init\_serialSCID(&SerialD,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;

// 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)

{

timeint = timeint+1;

time = timeint\*0.25;

sinvalue = ampl\***sin**(2\*PI\*frequency\*time) + offset;

satvalue = saturate(sinvalue,2.65);

**if** (UARTPrint == 1 ) {

serial\_printf(&SerialA,"Timeinit:%ld Time:%.2f Input:%.3f SatOut:%.2f ",CpuTimer2.InterruptCount,timeinit, time, sinvalue, satvalue);

UARTPrint = 0; //This must be set back to 0 in order to restart incrementing from 0 to 50.

}

}

}

// 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%250) == 0) {

displayLEDletter(LEDdisplaynum);

LEDdisplaynum++;

**if** (LEDdisplaynum == 0xFFFF) { // prevent roll over exception

LEDdisplaynum = 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 % 5) == 0) { //changed the number from 50 to 5 to get the period of 250ms. 50ms \* 5 = 250ms

UARTPrint = 1;

}

// If CpuTimer2.InterruptCount has incremented to 2457, how much time has gone by?

// 2455/5 = 491

// 2457%5 = 2

// 491+2 = 493

// 493 \* 50ms = 24.65 seconds

}

// Create saturate funciton with two float inputs and one float output

**float** **saturate**(**float** input, **float** saturation\_limit){

**if** (**fabs**(input) >= saturation\_limit){

// If the modulus of input is greater than or equal to the saturation limit, return the saturation limit.

**if** (input<0){ // If the input is a negative value, return negative saturation limit.

**float** x = -saturation\_limit; //Create a local variable that is returned at the end of the function.

}

**else** { // If the input is positive, return the positive saturation limit.

**float** x = saturation\_limit;

}

}

**else** **if** (**fabs**(input) < saturation\_limit){

**float** x = input; // If the modulus of input is less than the saturation limit, return the input.

}

**return** x

}