```c
//#############################################################################
// FILE:   LAB 3.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

//JS add definations for note period values and SONG_LENGTH
#define C4NOTE ((uint16_t)(((50000000/2)/2)/261.63))
#define D4NOTE ((uint16_t)(((50000000/2)/2)/293.66))
#define E4NOTE ((uint16_t)(((50000000/2)/2)/329.63))
#define F4NOTE ((uint16_t)(((50000000/2)/2)/349.23))
#define G4NOTE ((uint16_t)(((50000000/2)/2)/392.00))
#define A4NOTE ((uint16_t)(((50000000/2)/2)/440.00))
#define B4NOTE ((uint16_t)(((50000000/2)/2)/493.88))
#define C5NOTE ((uint16_t)(((50000000/2)/2)/523.25))
#define D5NOTE ((uint16_t)(((50000000/2)/2)/587.33))
#define E5NOTE ((uint16_t)(((50000000/2)/2)/659.25))
#define F5NOTE ((uint16_t)(((50000000/2)/2)/698.46))
#define G5NOTE ((uint16_t)(((50000000/2)/2)/783.99))
#define A5NOTE ((uint16_t)(((50000000/2)/2)/880.00))
#define B5NOTE ((uint16_t)(((50000000/2)/2)/987.77))
#define F4SHARPNOTE ((uint16_t)(((50000000/2)/2)/369.99))
#define G4SHARPNOTE ((uint16_t)(((50000000/2)/2)/415.3))
#define A4FLATNOTE ((uint16_t)(((50000000/2)/2)/415.3))
#define C5SHARPNOTE ((uint16_t)(((50000000/2)/2)/554.37))
#define A5FLATNOTE ((uint16_t)(((50000000/2)/2)/830.61))
#define OFFNOTE 0
#define SONG_LENGTH 61
//JS songarray for short happy birthday
//uint16_t songarray[SONG_LENGTH] = {
//E4NOTE,
//OFFNOTE,
//E4NOTE,
```

```
//OFFNOTE,
//F4SHARPNOTE,
//F4SHARPNOTE,
//F4SHARPNOTE,
//F4SHARPNOTE,
//E4NOTE,
//E4NOTE,
//E4NOTE,
//E4NOTE,
//A4NOTE,
//A4NOTE,
//A4NOTE,
//A4NOTE,
//G4SHARPNOTE,
//G4SHARPNOTE,
//G4SHARPNOTE,
//G4SHARPNOTE,
//G4SHARPNOTE,
//G4SHARPNOTE,
//G4SHARPNOTE,
//G4SHARPNOTE,
//E4NOTE,
//OFFNOTE,
//E4NOTE,
//OFFNOTE,
//F4SHARPNOTE,
//F4SHARPNOTE,
//F4SHARPNOTE,
//F4SHARPNOTE,
//E4NOTE,
//E4NOTE,
//E4NOTE,
//E4NOTE,
//B4NOTE,
//B4NOTE,
//B4NOTE,
//B4NOTE,
//A4NOTE,
//A4NOTE,
//A4NOTE,
//A4NOTE,
//A4NOTE,
//A4NOTE,
//A4NOTE,
//A4NOTE};
//JS songarray for mary had a little lamb
uint16_t songarray[SONG_LENGTH] = {
E4NOTE,
OFFNOTE,
D4NOTE,
OFFNOTE,
C4NOTE,
OFFNOTE,
D4NOTE,
OFFNOTE,
```

```
E4NOTE,
OFFNOTE,
E4NOTE,
OFFNOTE,
E4NOTE,
E4NOTE,
OFFNOTE,
D4NOTE,
OFFNOTE,
D4NOTE,
OFFNOTE,
D4NOTE,
D4NOTE,
OFFNOTE,
E4NOTE,
OFFNOTE,
G4NOTE,
OFFNOTE,
G4NOTE,
G4NOTE,
OFFNOTE,
E4NOTE,
OFFNOTE,
D4NOTE,
OFFNOTE,
C4NOTE,
OFFNOTE,
D4NOTE,
OFFNOTE,
E4NOTE,
OFFNOTE,
E4NOTE,
OFFNOTE,
E4NOTE,
OFFNOTE,
E4NOTE,
OFFNOTE,
D4NOTE,
OFFNOTE,
D4NOTE,
OFFNOTE,
E4NOTE,
OFFNOTE,
D4NOTE,
OFFNOTE,
C4NOTE,
OFFNOTE,
C4NOTE,
OFFNOTE,
C4NOTE,
OFFNOTE,
C4NOTE,
OFFNOTE};

// Interrupt Service Routines predefinition
```

```c
__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 define a global 16 bit variable to trigger increase or decrease of duty cycle
int16_t updown = 1;
//JS define a global 16 bit variable for duty cycle
int16_t mycount = 0;
//JS define a global float variable for motor control effort and servo angle
float motorcount = 0.0;
//JS define a global 16 bit variable for songarray index
int16_t songindex = 0;

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
    //JS PinMux to set GPIO22 to EPWM12A
    GPIO_SetupPinMux(22, GPIO_MUX_CPU1, 5);
    //GPIO_SetupPinOptions(22, GPIO_OUTPUT, GPIO_PUSHPULL);
    //GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;

    //JS PinMux to set GPIO2 to EPWM2A
    GPIO_SetupPinMux(2, GPIO_MUX_CPU1, 1);
    //JS PinMux to set GPIO3 to EPWM2B
    GPIO_SetupPinMux(3, GPIO_MUX_CPU1, 1);
    //JS PinMux to set GPIO14 to EPWM8A
    GPIO_SetupPinMux(14, GPIO_MUX_CPU1, 1);
    //JS PinMux to set GPIO15 to EPWM8B
    GPIO_SetupPinMux(15, GPIO_MUX_CPU1, 1);
    //JS PinMux to set GPIO16 to EPWM9A
    GPIO_SetupPinMux(16, GPIO_MUX_CPU1, 5);
    //JS PinMux to set GPIO104 to SCITXDD for UARTD's transmit
```

```
GPIO_SetupPinMux(104, GPIO_MUX_CPU1, 6);
   // 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;
```

```c
    // 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;
```

```c
    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);
    //JS for exercise 4, change to play happy birthday
    ConfigCpuTimer(&CpuTimer1, LAUNCHPAD_CPU_FREQUENCY, 125000);
    ConfigCpuTimer(&CpuTimer2, LAUNCHPAD_CPU_FREQUENCY, 10000);

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

       init_serialSCIA(&SerialA,115200);
       //same for the following EPWM, all EPWM settings are the same as EPWM12,
unless specified
       //JS Count up Mode bit is 00
    EPwm12Regs.TBCTL.bit.CTRMODE = 0;
    //JS 2/3 free run bit is 1x, 10 or 11
    EPwm12Regs.TBCTL.bit.FREE_SOFT = 2;
    //JS disable the phase loading bit is 0
    EPwm12Regs.TBCTL.bit.PHSEN = 0;
    //JS CLKDIV is 1, 2 to the power of 0
    EPwm12Regs.TBCTL.bit.CLKDIV = 0;
    //JS Start the timer at 0
    EPwm12Regs.TBCTR = 0;
    //JS Signal needs to be 20KHz, to have a period of 50 microseconds, TBPRD value
get divided by carrier frequency
    EPwm12Regs.TBPRD = 2500;
    //JS 0%*TBPRD for duty cycle
    EPwm12Regs.CMPA.bit.CMPA = 0;
    //JS when TBCTR=CMPA, clear to set to low
    EPwm12Regs.AQCTLA.bit.CAU = 1;
    //JS when TBCTR =0,set it to high
    EPwm12Regs.AQCTLA.bit.ZRO = 2;
    //JS copy from guideline
    EPwm12Regs.TBPHS.bit.TBPHS = 0;
```

```
    EPwm2Regs.TBCTL.bit.CTRMODE = 0;
    EPwm2Regs.TBCTL.bit.FREE_SOFT = 2;
    EPwm2Regs.TBCTL.bit.PHSEN = 0;
    EPwm2Regs.TBCTL.bit.CLKDIV = 0;
    EPwm2Regs.TBCTR = 0;
    EPwm2Regs.TBPRD = 2500;
    EPwm2Regs.CMPA.bit.CMPA = 0;
    //JS needs CMPB for EPWM2B
    EPwm2Regs.CMPB.bit.CMPB = 0;
    EPwm2Regs.AQCTLA.bit.CAU = 1;
    //JS needs CBU for EPWM2B
    EPwm2Regs.AQCTLB.bit.CBU = 1;
    EPwm2Regs.AQCTLA.bit.ZRO = 2;
    //JS needs AQCTLB.ZRO for EPWM2B
    EPwm2Regs.AQCTLB.bit.ZRO = 2;
    EPwm2Regs.TBPHS.bit.TBPHS = 0;

    EPwm8Regs.TBCTL.bit.CTRMODE = 0;
    EPwm8Regs.TBCTL.bit.FREE_SOFT = 2;
    EPwm8Regs.TBCTL.bit.PHSEN = 0;
    //JS for exercise 3, to set the carrier frequency to 50Hz. Firstly, divide the
orginal by 2^4
    EPwm8Regs.TBCTL.bit.CLKDIV = 4;
    EPwm8Regs.TBCTR = 0;
    //JS Secondly, to get the period divide TBPRD by carrier frequency value after
CLKDIV
    EPwm8Regs.TBPRD = 65535;
    EPwm8Regs.CMPA.bit.CMPA = 0;
    //JS needs CMPB for EPWM8B
    EPwm8Regs.CMPB.bit.CMPB = 0;
    EPwm8Regs.AQCTLA.bit.CAU = 1;
    //JS needs CBU for EPWM8B
    EPwm8Regs.AQCTLB.bit.CBU = 1;
    EPwm8Regs.AQCTLA.bit.ZRO = 2;
    //JS needs AQCTLB.ZRO for EPWM8B
    EPwm8Regs.AQCTLB.bit.ZRO = 2;
    EPwm8Regs.TBPHS.bit.TBPHS = 0;

    EPwm9Regs.TBCTL.bit.CTRMODE = 0;
    //JS 2/3
    EPwm9Regs.TBCTL.bit.FREE_SOFT = 2;

    EPwm9Regs.TBCTL.bit.PHSEN = 0;
    EPwm9Regs.TBCTL.bit.CLKDIV = 1;
    EPwm9Regs.TBCTR = 0;
    EPwm9Regs.TBPRD = 2500;
    //JS no need for CMPA
    //EPwm9Regs.CMPA.bit.CMPA = 0;
    //JS CAU and ZRO value create square wave
    EPwm9Regs.AQCTLA.bit.CAU = 0;
    EPwm9Regs.AQCTLA.bit.ZRO = 3;
    EPwm9Regs.TBPHS.bit.TBPHS = 0;

    //JS add following codes based on guidelines
    EALLOW; // Below are protected registers
```

```
    GpioCtrlRegs.GPAPUD.bit.GPIO2 = 1; // For EPWM2A
    GpioCtrlRegs.GPAPUD.bit.GPIO3 = 1; // For EPWM2B
    GpioCtrlRegs.GPAPUD.bit.GPIO14 = 1; // For EPWM8A
    GpioCtrlRegs.GPAPUD.bit.GPIO15 = 1; // For EPWM8B
    GpioCtrlRegs.GPAPUD.bit.GPIO16 = 1; // For EPWM9A
    GpioCtrlRegs.GPAPUD.bit.GPIO22 = 1; // For EPWM12A
    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;

      init_serialSCIC(&SerialC,115200);
      // JS uncomment for UART demo
      //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);
            UARTPrint = 0;
        }
    }
}
//JS create function setEPWM2A
//JS saturate controleffort between -10 to 10 for right motor
void setEPWM2A(float controleffort)
{
    if (controleffort > 10){
        controleffort = 10;
    }
    if (controleffort < -10){
        controleffort = -10;
    }
```

```
    //JS control duty cycle to 0% when controleffort is -10, 50% when controleffort
is 0, 100% when controleffort is 10, converts controleffort value between -10 and 10
to duty cycle 0% to 100%
    EPwm2Regs.CMPA.bit.CMPA = ((controleffort + (float)10))/ ((float)20)*
EPwm2Regs.TBPRD;
}
//JS saturate controleffort between -10 to 10 for left motor
void setEPWM2B(float controleffort){
    if (controleffort > 10){
        controleffort = 10;
    }
    if (controleffort < -10){
        controleffort = -10;
    }
    //JS control duty cycle to 0% when controleffort is -10, 50% when controleffort
is 0, 100% when controleffort is 10, converts controleffort value between -10 and 10
to duty cycle 0% to 100%
    EPwm2Regs.CMPB.bit.CMPB = ((controleffort + (float)10))/ ((float)20)*
EPwm2Regs.TBPRD;
}
//JS saturate angle between -90 and 90 for RCServo1
void setEPWM8A_RCServo(float angle)
{
    if (angle > 90){
        angle = 90;
    }
    if (angle < -90){
        angle = -90;
    }
    //JS control duty cycle to 4% when angle is -90, 8% when angle is 0, 12% when
angle is 90, converts angle value between -90 and 90 to duty cycle 4% to 12%
    EPwm8Regs.CMPA.bit.CMPA = ((((float)8/(float)180)*angle+(float)8)/100)*
EPwm8Regs.TBPRD;
}
//JS saturate angle between -90 and 90 for RCServo2
void setEPWM8B_RCServo(float angle)
{
    if (angle > 90){
        angle = 90;
    }
    if (angle < -90){
        angle = -90;
    }
    //JS control duty cycle to 4% when angle is -90, 8% when angle is 0, 12% when
angle is 90, converts angle value between -90 and 90 to duty cycle 4% to 12%
    EPwm8Regs.CMPB.bit.CMPB = ((((float)8/(float)180)*angle+(float)8)/100)*
EPwm8Regs.TBPRD;
}
// 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)

{
    if (songindex >= SONG_LENGTH){
        //JS when songindex get to the SONG_LENGTH, change the mux of pin to GPIO16
from EPWM9A
        GPIO_SetupPinMux(16, GPIO_MUX_CPU1, 0);
        //JS set GPIO16 to low
        GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;
    }
```

```
    if (songindex<SONG_LENGTH){
        //JS when songindex less than SONG_LENGTH, set TBPRD to the period of note in songarray, and increase songindex
        EPwm9Regs.TBPRD = songarray[songindex];
        songindex++;
    }


    CpuTimer1.InterruptCount++;

}

// cpu_timer2_isr CPU Timer2 ISR
__interrupt void cpu_timer2_isr(void)
{
//JS comment the code below used for exercise 1 and 2
//jS when updown is 1, the value set to the duty cycle mycount and controleffort value motorcount increases by 1 and 0.1 respectively until the duty cycle is 100% or motorcount reaches 10
//    if (updown == 1){
//        mycount++;
//        motorcount+=0.1;
    //JS pass motorcount to setEPWM2A to run the right motor
//        setEPWM2A(motorcount);
    //JS pass motorcount to setEPWM2B to run the left motor
//        setEPWM2B(motorcount);
//
//         EPwm12Regs.CMPA.bit.CMPA = mycount;
    //JS when motorcount is larger or equal to 10,set updowm to 0 to start decreasing motorcount
//         if (motorcount >= 10.0){
//             updown =0;
//         }
    //JS when duty cycle value equal to TBPRD, set updown to 0 to decrease the duty cycle mycount
//        if (mycount == EPwm12Regs.TBPRD){
//            updown = 0;
//        }
//     }
//    else{
//        mycount--;
//        motorcount-=0.1;
    //JS pass motorcount to setEPWM2A to run the right motor
//        setEPWM2A(motorcount);
    //JS pass motorcount to setEPWM2B to run the left motor
//        setEPWM2B(motorcount);
//
//        EPwm12Regs.CMPA.bit.CMPA = mycount;
    //JS when motorcount is less or equal to -10,set updowm to 1 to start increasing motorcount
//        if (motorcount <= -10.0){
//            updown = 1;
//        }
    //JS when duty cycle equals to 0, set updown to 1 to increase the duty cycle mycount
```

```
//        if (mycount == 0){
//            updown = 1;
//        }
//    }

    //JS for exercise 3, copied code from exercise 1 and 2 to change the angle of
servo
    if (updown == 1){
            mycount++;
            motorcount+=0.1;
            //JS pass motorcount to setEPWM8A_RCServo and setEPWM8B_RCServo
            setEPWM8A_RCServo(motorcount);
            setEPWM8B_RCServo(motorcount);
            EPwm12Regs.CMPA.bit.CMPA = mycount;
            if (motorcount >= 90.0){
                updown =0;
            }
    //        if (mycount == EPwm12Regs.TBPRD){
    //            updown = 0;
    //        }
        }
        else{
            mycount--;
            motorcount-=0.1;
            //JS pass motorcount to setEPWM8A_RCServo and setEPWM8B_RCServo
            setEPWM8A_RCServo(motorcount);
            setEPWM8B_RCServo(motorcount);
            EPwm12Regs.CMPA.bit.CMPA = mycount;
            if (motorcount <= -90.0){
                updown = 1;
            }
        }

    // Blink LaunchPad Blue LED
    GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;

    CpuTimer2.InterruptCount++;

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