#include "DSP28x\_Project.h" // Device Headerfile and Examples Include File

#include "f2802x\_common/include/adc.h"

#include "f2802x\_common/include/clk.h"

#include "f2802x\_common/include/flash.h"

#include "f2802x\_common/include/gpio.h"

#include "f2802x\_common/include/pie.h"

#include "f2802x\_common/include/pll.h"

#include "f2802x\_common/include/timer.h"

#include "f2802x\_common/include/wdog.h"

#include "f2802x\_common/include/pwr.h"

#include "f2802x\_common/include/pwm.h"

void pwm\_Init\_();

void InitEPwm1();

void InitEPwm2();

unsigned int TBPRD = 256;

unsigned int CMPA=0; //32500;

ADC\_Handle myAdc;

CLK\_Handle myClk;

FLASH\_Handle myFlash;

GPIO\_Handle myGpio;

PIE\_Handle myPie;

TIMER\_Handle myTimer;

CPU\_Handle myCpu;

PLL\_Handle myPll;

WDOG\_Handle myWDog;

PWM\_Handle myPwm1, myPwm2, myPwm4;

PWR\_Handle myPwr;

uint16\_t Digital\_Result =0;

void disable();

void enable();

void ADC\_INIT\_Fn();

void ADC\_SETUP\_Fn();

void set\_duty(int a);

int adcresult=2048;

interrupt void adc\_isr(void)

{

//discard ADCRESULT0 as part of the workaround to the 1st sample errata for rev0

Digital\_Result = ADC\_readResult(myAdc, ADC\_ResultNumber\_0);

adcresult =Digital\_Result;

set\_duty(adcresult);

ADC\_clearIntFlag(myAdc, ADC\_IntNumber\_1); // Clear ADCINT1 flag reinitialize for next SOC

PIE\_clearInt(myPie, PIE\_GroupNumber\_10);// Acknowledge interrupt to PIE

return;

}

void main(void)

{

myAdc = ADC\_init((void \*)ADC\_BASE\_ADDR, sizeof(ADC\_Obj));

myClk = CLK\_init((void \*)CLK\_BASE\_ADDR, sizeof(CLK\_Obj));

myCpu = CPU\_init((void \*)NULL, sizeof(CPU\_Obj));

myFlash = FLASH\_init((void \*)FLASH\_BASE\_ADDR, sizeof(FLASH\_Obj));

myGpio = GPIO\_init((void \*)GPIO\_BASE\_ADDR, sizeof(GPIO\_Obj));

myPie = PIE\_init((void \*)PIE\_BASE\_ADDR, sizeof(PIE\_Obj));

myPll = PLL\_init((void \*)PLL\_BASE\_ADDR, sizeof(PLL\_Obj));

myTimer = TIMER\_init((void \*)TIMER0\_BASE\_ADDR, sizeof(TIMER\_Obj));

myWDog = WDOG\_init((void \*)WDOG\_BASE\_ADDR, sizeof(WDOG\_Obj));

myPwm1 = PWM\_init((void \*)PWM\_ePWM1\_BASE\_ADDR, sizeof(PWM\_Obj));

myPwm2 = PWM\_init((void \*)PWM\_ePWM2\_BASE\_ADDR, sizeof(PWM\_Obj));

myPwm4 = PWM\_init((void \*)PWM\_ePWM3\_BASE\_ADDR, sizeof(PWM\_Obj));

myPwr = PWR\_init((void \*)PWR\_BASE\_ADDR, sizeof(PWR\_Obj));

disable();

// Perform basic system initialization

WDOG\_disable(myWDog);

CLK\_enableAdcClock(myClk);

CLK\_setOscSrc(myClk, CLK\_OscSrc\_Internal); //Select the internal oscillator 1 as the clock source

PLL\_setup(myPll, PLL\_Multiplier\_10, PLL\_DivideSelect\_ClkIn\_by\_2); // Setup the PLL for x10 /2 which will yield 50Mhz = 10Mhz \* 10 / 2

enable();

ADC\_INIT\_Fn();

ADC\_SETUP\_Fn();

//the Buck-Boost pwm is either 4 or 5

GPIO\_setMode(myGpio, GPIO\_Number\_4, GPIO\_4\_Mode\_EPWM3A);

GPIO\_setMode(myGpio, GPIO\_Number\_5, GPIO\_5\_Mode\_EPWM3B);

//disable pullup

GPIO\_setPullUp(myGpio, GPIO\_Number\_0, GPIO\_PullUp\_Disable);

GPIO\_setPullUp(myGpio, GPIO\_Number\_1, GPIO\_PullUp\_Disable);

GPIO\_setPullUp(myGpio, GPIO\_Number\_2, GPIO\_PullUp\_Disable);

GPIO\_setPullUp(myGpio, GPIO\_Number\_3, GPIO\_PullUp\_Disable);

//the inverter PWMs will be PWM1A/B and PWM2A/B on J6 pin 1,2,3,4 respectively

// GPIO0, GPIO1, GPIO2, GPIO3, respectively

GPIO\_setMode(myGpio, GPIO\_Number\_0, GPIO\_0\_Mode\_EPWM1A);

GPIO\_setMode(myGpio, GPIO\_Number\_1, GPIO\_1\_Mode\_EPWM1B);

GPIO\_setMode(myGpio, GPIO\_Number\_2, GPIO\_2\_Mode\_EPWM2A);

GPIO\_setMode(myGpio, GPIO\_Number\_3, GPIO\_3\_Mode\_EPWM2B);

CLK\_disableTbClockSync(myClk);

//initialize PWM

pwm\_Init\_();

InitEPwm1();

InitEPwm2();

CLK\_enableTbClockSync(myClk);

while(1)

{

ADC\_forceConversion(myAdc, ADC\_SocNumber\_0);// Wait for ADC interrupt

}

}

void disable()

{

// Disable the PIE and all interrupts

PIE\_disable(myPie);

PIE\_disableAllInts(myPie);

CPU\_disableGlobalInts(myCpu);

CPU\_clearIntFlags(myCpu);

}

void enable()

{

PIE\_enable(myPie);

// Register interrupt handlers in the PIE vector table

CPU\_enableInt(myCpu, CPU\_IntNumber\_10); // Enable CPU Interrupt 1

CPU\_enableGlobalInts(myCpu); // Enable Global interrupt INTM

CPU\_enableDebugInt(myCpu); // Enable Global realtime interrupt DBGM

// Enable XINT1 in the PIE: Group 1 interrupt 4 & 5

// Enable INT1 which is connected to WAKEINT

PIE\_enableInt(myPie, PIE\_GroupNumber\_3, PIE\_InterruptSource\_XINT\_1);

CPU\_enableInt(myCpu, CPU\_IntNumber\_1);

// GPIO0 is XINT1, GPIO1 is XINT2

GPIO\_setExtInt(myGpio, GPIO\_Number\_12, CPU\_ExtIntNumber\_1);

// Configure XINT1

PIE\_setExtIntPolarity(myPie, CPU\_ExtIntNumber\_1, PIE\_ExtIntPolarity\_RisingEdge);

// Enable XINT1 and XINT2

PIE\_enableExtInt(myPie, CPU\_ExtIntNumber\_1);

}

void ADC\_INIT\_Fn()

{

ADC\_enableBandGap(myAdc);

ADC\_enableRefBuffers(myAdc);

ADC\_powerUp(myAdc);

ADC\_enable(myAdc);

ADC\_setVoltRefSrc(myAdc, ADC\_VoltageRefSrc\_Int);

}

void ADC\_SETUP\_Fn()

{

PIE\_registerPieIntHandler(myPie, PIE\_GroupNumber\_10, PIE\_SubGroupNumber\_1, (intVec\_t)&adc\_isr);

PIE\_enableAdcInt(myPie, ADC\_IntNumber\_1); // Enable ADCINT1 in PIE

//Note: Channel ADCINA1 will be double sampled to workaround the ADC 1st sample issue for rev0 silicon errata

ADC\_setIntPulseGenMode(myAdc, ADC\_IntPulseGenMode\_Prior); //ADCINT1 trips after AdcResults latch

ADC\_enableInt(myAdc, ADC\_IntNumber\_1); //Enabled ADCINT1

ADC\_setIntMode(myAdc, ADC\_IntNumber\_1, ADC\_IntMode\_ClearFlag); //Disable ADCINT1 Continuous mode

ADC\_setIntSrc(myAdc, ADC\_IntNumber\_1, ADC\_IntSrc\_EOC0); //setup EOC0 to trigger ADCINT1 to fire

ADC\_setSocChanNumber (myAdc, ADC\_SocNumber\_0, ADC\_SocChanNumber\_A4); //set SOC0 channel select to ADCINA4

ADC\_setSocTrigSrc(myAdc, ADC\_SocNumber\_0, ADC\_SocTrigSrc\_Sw); //set SOC0 start trigger on EPWM4A, due to round-robin SOC0 converts first then SOC1

ADC\_setSocSampleWindow(myAdc, ADC\_SocNumber\_0, ADC\_SocSampleWindow\_7\_cycles); //set SOC0 S/H Window to 7 ADC Clock Cycles, (6 ACQPS plus 1)

}

void pwm\_Init\_()

{

CLK\_enablePwmClock(myClk, PWM\_Number\_3);

// Setup TBCLK

PWM\_setPeriod(myPwm4, TBPRD); // Set timer period

PWM\_setPhase(myPwm4, 0x0000); // Phase is 0

PWM\_setCount(myPwm4, 0x0000); // Clear counter

// Setup counter mode

PWM\_setCounterMode(myPwm4, PWM\_CounterMode\_UpDown); // Count up and down

PWM\_disableCounterLoad(myPwm4); // Disable phase loading

PWM\_setHighSpeedClkDiv(myPwm4, PWM\_HspClkDiv\_by\_1); // Clock ratio to SYSCLKOUT

PWM\_setClkDiv(myPwm4, PWM\_ClkDiv\_by\_1);

///setup for GPIO6 PWM

// Setup shadowing

PWM\_setShadowMode\_CmpA(myPwm4, PWM\_ShadowMode\_Shadow);

PWM\_setLoadMode\_CmpA(myPwm4, PWM\_LoadMode\_Zero);

// Set actions

PWM\_setActionQual\_CntUp\_CmpA\_PwmA(myPwm4, PWM\_ActionQual\_Clear); // Set PWM1A on event A, up count

PWM\_setActionQual\_CntDown\_CmpA\_PwmA(myPwm4, PWM\_ActionQual\_Set); // Clear PWM1A on event A, down count

//SETUP for GPIO 7 PWM

// Setup shadowing

PWM\_setShadowMode\_CmpB(myPwm4, PWM\_ShadowMode\_Shadow);

PWM\_setLoadMode\_CmpB(myPwm4, PWM\_LoadMode\_Zero);

// Set actions

PWM\_setActionQual\_CntUp\_CmpB\_PwmB(myPwm4, PWM\_ActionQual\_Clear); // Set PWM1B on event B, up count

PWM\_setActionQual\_CntDown\_CmpB\_PwmB(myPwm4, PWM\_ActionQual\_Set); // Clear PWM1B on event B, down count

}

void InitEPwm1()

{

// Enables the pwm clock

CLK\_enablePwmClock(myClk, PWM\_Number\_1);

// Sets the pulse width modulation (PWM) sync mode

// TBCTL Time-Base Control Register - syncmode PWM\_TBCTL\_SYNCOSEL\_BITS

PWM\_setSyncMode(myPwm1, PWM\_SyncMode\_CounterEqualZero);

// Sets the pulse width modulation (PWM) period

// TBPRD Time Base Period Register - period

PWM\_setPeriod(myPwm1, 40960);

// Sets the pulse width modulation (PWM) phase

// TBPHS Time-Base Phase Register - phase (0)

PWM\_setPhase(myPwm1, 0x0000);

// Sets the pulse width modulation (PWM) count

// TBCTR Time-Base Counter - count

PWM\_setCount(myPwm1, 0x0000);

// Sets the pulse width modulation (PWM) counter mode

// TBCTL Time-Base Control Register - counterMode PWM\_TBCTL\_CTRMODE\_BITS

PWM\_setCounterMode(myPwm1, PWM\_CounterMode\_UpDown);

// Disables the pulse width modulation (PWM) counter loading from the phase register

// TBCTL Time-Base Control Register - PWM\_TBCTL\_PHSEN\_BITS

PWM\_disableCounterLoad(myPwm1);

// Sets the pulse width modulation (PWM) high speed clock divisor

// TBCTL Time-Base Control Register - clkDiv PWM\_TBCTL\_HSPCLKDIV\_BITS

PWM\_setHighSpeedClkDiv(myPwm1, PWM\_HspClkDiv\_by\_10);

// Sets the pulse width modulation (PWM) clock divisor

// TBCTL Time-Base Control Register - WM\_TBCTL\_CLKDIV\_BITS

PWM\_setClkDiv(myPwm1, PWM\_ClkDiv\_by\_1);

//adds deadtime by setting CMP values to <50% duty

//with the current AQ configuration: decrease A duty by increasing CMPA; decrease B duty by decreasing CMPB

PWM\_setCmpA(myPwm1, 20462); //20480-18

PWM\_setCmpB(myPwm1, 20528); //20480+18

PWM\_setActionQual\_CntUp\_CmpA\_PwmA(myPwm1, PWM\_ActionQual\_Set);

PWM\_setActionQual\_CntDown\_CmpA\_PwmA(myPwm1, PWM\_ActionQual\_Clear);

PWM\_setActionQual\_CntUp\_CmpB\_PwmB(myPwm1, PWM\_ActionQual\_Clear);

PWM\_setActionQual\_CntDown\_CmpB\_PwmB(myPwm1, PWM\_ActionQual\_Set);

}

void InitEPwm2()

{

// Enables the pwm clock

CLK\_enablePwmClock(myClk, PWM\_Number\_2);

// Sets the pulse width modulation (PWM) sync mode

// TBCTL Time-Base Control Register - syncmode PWM\_TBCTL\_SYNCOSEL\_BITS

PWM\_setSyncMode(myPwm2, PWM\_SyncMode\_EPWMxSYNC);

// Sets the pulse width modulation (PWM) period

// TBPRD Time Base Period Register - period

PWM\_setPeriod(myPwm2, 40960);

// Enables the pulse width modulation (PWM) counter loading from the phase register

// TBCTL Time-Base Control Register - PWM\_TBCTL\_PHSEN\_BITS

PWM\_enableCounterLoad(myPwm2);

// Sets the pulse width modulation (PWM) phase

// TBPHS Time-Base Phase Register - phase (27307/period\*2)\*360 = 120deg)

PWM\_setPhase(myPwm2, 27307); //120 degree shift

// Sets the pulse width modulation (PWM) phase direction

// TBCTL Time-Base Control Register - phase direction PWM\_TBCTL\_PHSDIR\_BITS

PWM\_setPhaseDir(myPwm2, PWM\_PhaseDir\_CountUp);

// Sets the pulse width modulation (PWM) counter mode

// TBCTL Time-Base Control Register - counterMode PWM\_TBCTL\_CTRMODE\_BITS

PWM\_setCounterMode(myPwm2, PWM\_CounterMode\_UpDown);

// Sets the pulse width modulation (PWM) high speed clock divisor

// TBCTL Time-Base Control Register - clkDiv PWM\_TBCTL\_HSPCLKDIV\_BITS

PWM\_setHighSpeedClkDiv(myPwm2, PWM\_HspClkDiv\_by\_10);

// Sets the pulse width modulation (PWM) clock divisor

// TBCTL Time-Base Control Register - WM\_TBCTL\_CLKDIV\_BITS

PWM\_setClkDiv(myPwm2, PWM\_ClkDiv\_by\_1);

//adds deadtime by setting CMP values to 49.5% duty

//with the current AQ configuration: decrease A duty by increasing CMPA; decrease B duty by decreasing CMPB

PWM\_setCmpA(myPwm2, 20462); //20480-18

PWM\_setCmpB(myPwm2, 20528); //20480+18

PWM\_setActionQual\_CntUp\_CmpA\_PwmA(myPwm2, PWM\_ActionQual\_Set);

PWM\_setActionQual\_CntDown\_CmpA\_PwmA(myPwm2, PWM\_ActionQual\_Clear);

PWM\_setActionQual\_CntUp\_CmpB\_PwmB(myPwm2, PWM\_ActionQual\_Clear);

PWM\_setActionQual\_CntDown\_CmpB\_PwmB(myPwm2, PWM\_ActionQual\_Set);

}

void set\_duty( int a)

{

CMPA = a/16;

if(CMPA > 205) //cap duty ratio at 80%

{

CMPA = 205;

}

PWM\_setCmpA(myPwm4, CMPA);

PWM\_setCmpB(myPwm4, CMPA);

}