#include "function\_prototypes.h"

#include "definitions.h"

#include "peripheral\_registers.h"

void initializeGlobalVariables();

void initializeSerialCommunicationInterface();

void initializeTimers();

void initializeLEDs();

void initializeDACInterface();

void initializeVariables();

void initIsr ();

void initializeExternalInterrupts();

void initializeTripCloseGPIOs(void);

void initialize\_unit()

{

int i;

DACPresent = 1; //change to 1 to make it so DACs will work

initIsr ();

//InitializeFlashInterface();

initializeGlobalVariables();

initializeSerialCommunicationInterface();

InitializeSerialPeripheralInterface();

initializeTripCloseGPIOs();

initializeTimers();

initializeLEDs();

initializeExternalInterrupts();

if(DACPresent == 1)

initializeDACInterface();

// Start LED Timer

timerC0\_control.bit.count\_mode = 1;

timerC1\_control.bit.count\_mode = 0;

//DAC timers - C3 is the latch timer

///////////////////////////////////////////////////////

//Test

timerD0\_control.bit.count\_mode = 1;

timerC2\_control.bit.count\_mode = 1;

timerC3\_control.bit.count\_mode = 1;

///////////////////////////////////////////////////////

/\*

timerD0\_control.bit.count\_mode = 0;

timerC2\_control.bit.count\_mode = 0;

timerC3\_control.bit.count\_mode = 0;

\*/

//start program

program\_running = TRUE;

}

void initializeSerialCommunicationInterface()

{

//To Check

SCI0\_baud\_rate.bit.SBR = 195; // Baud rate = 19200 for 60MHz Z clock

SCI0\_control\_reg.bit.TE = 1; // Enable transmitter

SCI0\_control\_reg.bit.RE = 1; // Enable receiver

//Intrpt\_Priority\_Reg4.bit.GPIOA = 7; //I don't get this

SCI0\_control\_reg.bit.RFIE = 1;

Intrpt\_Priority\_Reg8.bit.SCI0\_xmit = 3; // SCI0 level 2 intrpts

Intrpt\_Priority\_Reg8.bit.SCI0\_rcv = 2;

// Setup TX, RX and msg queue pointers

txQ\_in = &txQ[0]; txQ\_out = txQ\_in; testQ\_in = (char \*)txQ\_in;

rxQ\_in = &rxQ[0]; rxQ\_out = rxQ\_in;

msgQ\_in = &msgQ[0]; msgQ\_out = msgQ\_in;

SCI1\_baud\_rate.bit.SBR = 98; // Baud rate = 19200 for 60MHz Z clock

SCI1\_control\_reg.bit.TE = 1; // Enable transmitter

SCI1\_control\_reg.bit.RE = 1; // Enable receiver

//Intrpt\_Priority\_Reg4.bit.GPIOA = 7; //I don't get this

SCI1\_control\_reg.bit.RFIE = 1;

Intrpt\_Priority\_Reg5.bit.SCI1\_xmit = 3; // SCI0 level 2 intrpts

Intrpt\_Priority\_Reg5.bit.SCI1\_rcv = 2; //TEST WERE BOTH ZERO!

// Setup TX, RX and msg queue pointers

txQ1\_in = &txQ1[0]; txQ1\_out = txQ1\_in;

rxQ1\_in = &rxQ1[0]; rxQ1\_out = rxQ1\_in;

msgQ1\_in = &msgQ1[0]; msgQ1\_out = msgQ1\_in;

}

void initializeTimers()

{

//Timer for Blinking light

timerC0\_load = 0xE4E1;//E4E1 = 1/8 s

//0 = don't count

timerC0\_control.bit.primary\_count\_src = 0b1111; //F = divide clk by 64

timerC0\_control.bit.second\_src = 0; //0 = none

timerC0\_control.bit.once = 0; //0 = count repeatedly

timerC0\_control.bit.length = 1; //1 = reload load after count

timerC0\_control.bit.count\_direction = 1; //1 = count down

timerC0\_control.bit.co\_chan\_init = 0; //0 = use just primary count

timerC0\_control.bit.output\_mode = 0b011; //toggle OFLAG

timerC0\_compare\_down = 0x0000;

timerC0\_status.word = 0x4000;

Intrpt\_Priority\_Reg6.bit.TMRC0 = 1;

//Timer For Switch Debounce

timerD2\_load = 0x6B8D; //??

timerD2\_control.bit.output\_mode = 3; //Toggle oflag

timerD2\_control.bit.co\_chan\_init = 0; //disabled

timerD2\_control.bit.count\_direction = 1; //down

timerD2\_control.bit.length = 1; //reinit after match

timerD2\_control.bit.once = 1; //1 - count once

timerD2\_control.bit.second\_src = 0; //no secondary source

timerD2\_control.bit.primary\_count\_src = 0xF; //F = divide IPBUs by 128

timerD2\_control.bit.count\_mode = 1; //don't count (will be 1)

timerD2\_status.word = 0x4000; //comp interrupt enabled

Intrpt\_Priority\_Reg6.bit.TMRD2 = 1;

if(DACPresent == 1)

{

timerC2\_load = 30000000/(60\*samp\_per\_cycle);// 0x9C4; //0x56D;

timerC2\_control.bit.output\_mode = 3;

timerC2\_control.bit.co\_chan\_init = 0;

timerC2\_control.bit.count\_direction = 1;

timerC2\_control.bit.length = 1;

timerC2\_control.bit.once = 0;

timerC2\_control.bit.second\_src = 0;

timerC2\_control.bit.primary\_count\_src = 0x9;

timerC2\_status.word = 0x4000;

Intrpt\_Priority\_Reg7.bit.TMRC2 = 1;

// Init timer C3 for DAC output points

timerC3\_load = 30000000/(60\*samp\_per\_cycle);// 0x9C4; //0x56D; ///\* 46.286 us sync strobe \*/

timerC3\_control.bit.output\_mode = 3;

timerC3\_control.bit.co\_chan\_init = 0;

timerC3\_control.bit.count\_direction = 1;

timerC3\_control.bit.length = 1;

timerC3\_control.bit.once = 0;

timerC3\_control.bit.second\_src = 0;

timerC3\_control.bit.primary\_count\_src = 0x9;

timerC3\_status.word = 0x4000;

Intrpt\_Priority\_Reg7.bit.TMRC3 = 1;

timerD0\_load = 30000000/(60\*samp\_per\_cycle);// 0x9C4; //0x56D;

timerD0\_control.bit.output\_mode = 3;

timerD0\_control.bit.co\_chan\_init = 0;

timerD0\_control.bit.count\_direction = 1;

timerD0\_control.bit.length = 1;

timerD0\_control.bit.once = 0;

timerD0\_control.bit.second\_src = 0;

timerD0\_control.bit.primary\_count\_src = 0x9;

timerD0\_status.word = 0x4000;

Intrpt\_Priority\_Reg6.bit.TMRD0 = 1;

}

/\* Init timer C1 for random number gen \*/

timerC1\_load = 0xFFFF; //0x03E8;/\* 46.286 us sync strobe \*/

timerC1\_control.bit.output\_mode = 0;

timerC1\_control.bit.co\_chan\_init = 0;

timerC1\_control.bit.count\_direction = 0;

timerC1\_control.bit.length = 0;

timerC1\_control.bit.once = 0;

timerC1\_control.bit.second\_src = 0;

timerC1\_control.bit.primary\_count\_src = 0x8;

timerC1\_status.word = 0x0000;

//timerC3\_counter = 0x0080;//0x56D;

//General\_intrpt\_Priority\_Reg8.bit.TMRC1 = 001;

timerD1\_load = 0x6DDD; //.06 seconds > max amount of time to transfer all data from GUI

timerD1\_control.bit.output\_mode = 3; //Toggle oflag

timerD1\_control.bit.co\_chan\_init = 0; //disabled

timerD1\_control.bit.count\_direction = 1; //down

timerD1\_control.bit.length = 1; //reinit after match

timerD1\_control.bit.once = 1; //1 - count once

timerD1\_control.bit.second\_src = 0; //no secondary source

timerD1\_control.bit.primary\_count\_src = 0xF; //F = divide IPBUs by 128

timerD1\_control.bit.count\_mode = 0; //don't count (will be 1)

timerD1\_status.word = 0x4000; //comp interrupt enabled

Intrpt\_Priority\_Reg6.bit.TMRD1 = 1;

}

void initializeLEDs()

{

//To Do

//GPIOE13 TD3 to LED D9 Amplifiers off

GPIOE\_peripheral\_enable.bit.TD3 = 0; //

GPIOE\_Data\_direction.bit.DB13 = 1; //sets to output

GPIOE\_Data\_reg.bit.DB13 = 0; //turn LED off

//GPIOD2 ~CS4 to LED D10 Blinky

GPIOD\_peripheral\_enable.bit.CS4 = 0; //

GPIOD\_Data\_direction.bit.DB2 = 1; //sets to output

GPIOD\_Data\_reg.bit.DB2 = 1; //turn LED on

GPIOD\_peripheral\_enable.bit.CS5 = 0; //

GPIOD\_Data\_direction.bit.DB3 = 1; //sets to output

GPIOD\_Data\_reg.bit.DB3 = 1; //turn LED on

//GPIOD3 ~CS5 to LED D11

GPIOD\_peripheral\_enable.bit.CS5 = 0; //

GPIOD\_Data\_direction.bit.DB3 = 1; //sets to output

GPIOD\_Data\_reg.bit.DB3 = 0; //turn LED off

/\*

GPIOD\_peripheral\_enable.bit.CS3 = 1; //sets to normal Mode

GPIOD\_Data\_direction.bit.CS3 = 1; //sets to output

GPIOD\_Data\_reg.bit.CS3 = 0; //turn LED on

\*/

//TEST FOR DEV BOARD!!!!

//GPIOC\_peripheral\_enable.bit.SCLK1 = 0;

//GPIOC\_Data\_direction.bit.DB0 = 1;

//GPIOC\_Data\_reg.bit.DB0 = 0;

}

void initializeDACInterface()

{

//GPIOE TD0 = LDAC

//GPIOE TD1 = ENABLE\_N

//GPIOE TD2 = SYNC\_N

//GPIOE MOSI0 = Din (on DAC)

//GPIOE SCLK0 = SCLK0

int test1 = 0, test2 = 0;

GPIOE\_peripheral\_enable.bit.TD0 = 0; //LDAC GPIO initialize

GPIOE\_Data\_direction.bit.DB10 = 1;

GPIOE\_Data\_reg.bit.DB10 = 0;

GPIOE\_peripheral\_enable.bit.TD1 = 0; //Chip enable init

GPIOE\_Data\_direction.bit.DB11 = 1;

GPIOE\_Data\_reg.bit.DB11 = 1;

GPIOE\_peripheral\_enable.bit.TD2 = 0; //SYNC enable init

GPIOE\_Data\_direction.bit.DB12 = 1;

GPIOE\_Data\_reg.bit.DB12 = 1;

DACEnable = 0;

//Enable Interrupts

// Setup serial peripheral interface

sendDACData(LoadBufA, 0x7FFF);

sendDACData(LatchAll, 0x0000);

}

void initializeVariables()

{

//variable init

}

void InitializeSerialPeripheralInterface()

{

//SS1 initialization

/\* out for TEST///////////////////////////

GPIOC\_peripheral\_enable.bit.SSI1 = 1; //1 - Makes It Peripheral (for slave, init will fix for master)

GPIOC\_Data\_direction.bit.SSI1 = 0; //Input (for slave, init will change for master.

GPIOC\_Data\_reg.bit.SSI1 = 1; //active low, so set high

\*/

//SPI0 for DAC control

SPI0\_status\_control\_reg.bit.SPR = 0x0; //0 - divide clock rate by 2

SPI0\_status\_control\_reg.bit.DSO = 0; //MSB first

SPI0\_status\_control\_reg.bit.EERIE = 0; //no error interrupt enabled

SPI0\_status\_control\_reg.bit.MODFEN = 0; //check this

SPI0\_status\_control\_reg.bit.SPRIE = 0; //No SPRF interrupt

SPI0\_status\_control\_reg.bit.SPMSTR = 1; //Master Operation

SPI0\_status\_control\_reg.bit.CPHA = 1; //Clock changes with bit change

SPI0\_status\_control\_reg.bit.CPOL = 0; //rise first

SPI0\_status\_control\_reg.bit.SPE = 0; //SPI Disabled

SPI0\_status\_control\_reg.bit.SPTIE = 0; //No transmit interrupt

Intrpt\_Priority\_Reg4.bit.SPI0\_RCV = 0;

Intrpt\_Priority\_Reg5.bit.SPI0\_xmit = 0;

SPI0\_data\_size = 0x7; //0xF for 16, 0x7 for 8 bit transfer size

SPI0\_status\_control\_reg.bit.SPE = 1;

/////////////////////////////////////

//SPI1 for Inter Processor Communications

//Make SPI1 the peripheral controlling the pins

SIM\_GPS.bit.C0 = 1;

SIM\_GPS.bit.C1 = 1;

SIM\_GPS.bit.C2 = 1;

SIM\_GPS.bit.C3 = 1;

//control the SS line for normal use

GPIOC\_peripheral\_enable.bit.SSI1 = 1; //0 - GPIO, 1 - for normal

SPI1\_status\_control\_reg.bit.SPR = 0x0; //0 - divide clock rate by 2

SPI1\_status\_control\_reg.bit.DSO = 0; //MSB first

SPI1\_status\_control\_reg.bit.EERIE = 0; //no error interrupt enabled

SPI1\_status\_control\_reg.bit.MODFEN = 0; //check this

SPI1\_status\_control\_reg.bit.SPRIE = 1; //enable receive interrupt

SPI1\_status\_control\_reg.bit.SPMSTR = 0; //0 - Slave OP, will change through SPI if master

SPI1\_status\_control\_reg.bit.CPHA = 0; //Clock changes with bit change

SPI1\_status\_control\_reg.bit.CPOL = 0; //Fall First

SPI1\_status\_control\_reg.bit.SPE = 0; //SPI Disabled

SPI1\_status\_control\_reg.bit.SPTIE = 0; //No transmit interrupt

Intrpt\_Priority\_Reg4.bit.SPI1\_rcv = 2;

Intrpt\_Priority\_Reg4.bit.SPI1\_xmit = 2;

SPI1\_data\_size = 0x7; //0xF for 16, 0x7 for 8 bit transfer size

SPI1\_status\_control\_reg.bit.SPE = 1;

//control line for return

GPIOD\_peripheral\_enable.bit.ISB2 = 0; //0 - GPIO

GPIOD\_Data\_direction.bit.DB12 = 1; //1 - output, will change with master

GPIOD\_Data\_reg.bit.DB12 = 1; //1 - active high, maybe

//GPIOD\_PPMODE.bit.ISB2 = 0; //0 - open drain

//SS line for PhaseB

GPIOD\_peripheral\_enable.bit.ISB0 = 0; //0 - GPIO

GPIOD\_Data\_direction.bit.DB10 = 1; //1 - output, will change with master

GPIOD\_Data\_reg.bit.DB10 = 1; //1 - active high

//GPIOD\_PPMODE.bit.ISB2 = 0; //0 - open drain

//SS line for PhaseC

GPIOD\_peripheral\_enable.bit.ISB1 = 0; //0 - GPIO

GPIOD\_Data\_direction.bit.DB11 = 1; //1 - output, will change with master

GPIOD\_Data\_reg.bit.DB11 = 1; //1 - active high

}

void initializeGlobalVariables()

{

int i;

//ReadFlashToRAM();

PhA.Vn.ConversionFactor.Full = RAMParameters.NetworkCalibration.Full;

PhA.Vt.ConversionFactor.Full = RAMParameters.TransformerCalibration.Full;

PhA.I.ConversionFactor.Full = RAMParameters.CurrentCalibration.Full;

PhA.I.offsetCorrection = RAMParameters.PhaseCorrection;

//if( PhA.Vn.ConversionFactor.Full == 0xFFFF || PhA.Vn.ConversionFactor.Full == 0 ||

// PhA.Vt.ConversionFactor.Full == 0xFFFF || PhA.Vt.ConversionFactor.Full == 0 ||

// PhA.I.ConversionFactor.Full == 0xFFFF || PhA.I.ConversionFactor.Full == 0)

//{

// PhA.Vn.ConversionFactor.Full = 0x4000; //1.00 in 4\_12 fixed point

// PhA.I.ConversionFactor.Full = 0x4000; //1.00

// PhA.Vt.ConversionFactor.Full = 0x4000;

//}

PhA.Vn.inputAmplitude.Full = 0;

PhA.Vn.amplitude.Full = 0;

PhA.Vn.inputOffset = 0;

PhA.Vn.rotate\_sig = 0;

PhA.Vn.rotate\_count = 0;

PhA.Vn.timing\_register = &timerC2\_load;

PhA.Vn.harmonic\_array = PhAVnHarmonicArray;

PhA.I.inputAmplitude.Full = 0;

PhA.I.amplitude.Full = 0;

PhA.I.inputOffset = 0;

PhA.I.rotate\_sig = 0;

PhA.I.rotate\_count = 0;

PhA.I.timing\_register = &timerC3\_load;

PhA.I.harmonic\_array = PhAIHarmonicArray;

PhA.Vt.inputAmplitude.Full = 0;

PhA.Vt.amplitude.Full = 0;

PhA.Vt.inputOffset = 0;

PhA.Vt.rotate\_sig = 0;

PhA.Vt.rotate\_count = 0;

PhA.Vt.timing\_register = &timerD0\_load;

PhA.Vt.harmonic\_array = PhAVtHarmonicArray;

PhA.I.frequencyComponents.thirdHarmonic = 0;

PhA.I.frequencyComponents.fifthHarmonic = 0;

PhA.I.frequencyComponents.seventhHarmonic = 0;

PhA.I.frequencyComponents.ninthHarmonic = 0;

PhA.Vn.frequencyComponents.thirdHarmonic = 0;

PhA.Vn.frequencyComponents.fifthHarmonic = 0;

PhA.Vn.frequencyComponents.seventhHarmonic = 0;

PhA.Vn.frequencyComponents.ninthHarmonic = 0;

for(i = 0; i < samp\_per\_cycle; ++i)

{

//PhAVnHarmonicArray[i] = 0;

//PhAVtHarmonicArray[i] = 0;

//PhAIHarmonicArray[i] = 0;

PhA.I.harmonic\_array[i] = 0;

PhA.Vn.harmonic\_array[i] = 0;

PhA.Vt.harmonic\_array[i] = 0;

}

generateHarmonicArray(PhA.Vn.frequencyComponents, PhA.Vn.harmonic\_array);

TempPhase = PhA;

}

void initializeExternalInterrupts()

{

//IRQA

GPIOE\_peripheral\_enable.bit.TC0 = 0; //GPIO Mode

GPIOE\_Data\_direction.bit.DB8 = 1; //Output

GPIOE\_Data\_reg.bit.DB8 = 0; //Off

Intrpt\_Priority\_Reg2.bit.IRQAIPL = 1; //interrupt for IRQ enabled

//IRQB

GPIOE\_peripheral\_enable.bit.TC1 = 0; //GPIO Mode

GPIOE\_Data\_direction.bit.DB9 = 1; //Output

GPIOE\_Data\_reg.bit.DB9 = 0; //Off

Intrpt\_Priority\_Reg2.bit.IRQBIPL = 1;

ICTL.bit.IRQA\_edge = 1; //cause of freeze error not sure why yet

ICTL.bit.IRQB\_edge = 1; //make edge-triggered

}

void initializeTripCloseGPIOs(void)

{

GPIOB\_peripheral\_enable.bit.A17 = 0; //GPIOMode

GPIOB\_Data\_direction.bit.DB1 = 0; //Input

GPIOBInterruptEnable.bit.A17 = 1; //Enable Interrupt

GPIOBInterruptPolarity.bit.A17 = 1; //Falling edge

GPIOB\_peripheral\_enable.bit.A18 = 0; //GPIOMode

GPIOB\_Data\_direction.bit.DB2 = 0; //Input

//GPIOBInterruptEnable.bit.A18 = 1; //Enable Interrupt

//GPIOBInterruptPolarity.bit.A18 = 1; //Falling edge

Intrpt\_Priority\_Reg4.bit.GPIOB = 0; //rising edge int

}