/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Project 5 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File: Proj5.c

\* Author name: Seth Cram

\* Rev. Date: 10/05/2021

\*

\* Project Description:

\*

\*

\* Notes:

\* CW = down table

\* SM1-SM4 = stepper motor outputs (other LEDs in same port so read-modify-write)

\* now set step\_delay based on speed + mode in decode\_btns()

\* Timer1 ISR (sm) is checked once every 1ms bc of PR1 value

\* Changed motor\_cntr to static var

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <plib.h>

#include "CerebotMX7cK.h"

#include "Proj5.h"

int main()

{

int btns;

system\_init(); /\* Setup system Hardware and SW for this lab. \*/

//setup timer1:

t1\_intr\_init();

//setup change notice intrs for btn1 and btn2:

cn\_intr\_init();

//read and decode the btns so 'step\_delay' set:

btns = read\_buttons();

decode\_buttons( btns, &step\_delay, &dir, &mode );

//Timer1\_ISR(); //tried calling ISR from code

while(1)

{

//empty while(1) loop

}

return 0;

}

//Sets up the configuration:

void system\_init(void)

{

// Setup processor board

Cerebot\_mx7cK\_setup();

PORTSetPinsDigitalOut(IOPORT\_B, SM\_LEDS);/\* Set PmodSTEP LEDs A-H outputs \*/

LATBCLR = SM\_LEDS; /\* Turn off LEDA through LEDH \*/

//set btn1 and btn2 to digital inputs:

PORTSetPinsDigitalIn(IOPORT\_G, BTN1 | BTN2);

//interrupt setup:

INTEnableSystemMultiVectoredInt();

INTEnableInterrupts();

}

//init timer 1 for interrupts:

void t1\_intr\_init(void)

{

//configure Timer 1 with internal clock, 1:1 prescale, PR1 for 1 ms period

OpenTimer1(T1\_ON | T1\_SOURCE\_INT | T1\_PS\_1\_1, T1\_TICK-1);

// set up the timer interrupt with a priority of 2, sub priority 0

mT1SetIntPriority(2); // Group priority range: 1 to 7

mT1SetIntSubPriority(0); // Subgroup priority range: 0 to 3

mT1IntEnable(1); // Enable T1 interrupts

// Global interrupts must enabled to complete the initialization.

}

//ISR for timer1: (triggered once every 1ms bc PR1 val)

void \_\_ISR(\_TIMER\_1\_VECTOR, IPL2) Timer1\_ISR(void)

{

static int motor\_cntr = 0; //declared static bc only needed in this ISR, zero so initialized below

LATBINV = LEDA; //toggle LEDA every ms

unsigned int sm\_code; //local var

motor\_cntr = motor\_cntr - 1; //decrement sm cntr

if( motor\_cntr <= 0 ) //if time to set motor

{

sm\_code = sw\_fsm( dir, mode );

output\_sm\_code( sm\_code );

LATBINV = LEDB; //toggle LEDB every motor write

motor\_cntr = step\_delay; //reset cntr

}

mT1ClearIntFlag(); //has to be at the very end?

}

/\* Initialization of CN peripheral for interrupt level 1 \*/

void cn\_intr\_init(void)

{

unsigned int dummy; // used to hold PORT read value

// BTN1 and BTN2 pins set for input by Cerebot header file

// PORTSetPinsDigitalIn(IOPORT\_G, BIT\_6 | BIT7); //

// Enable CN for BTN1 and BTN2

mCNOpen(CN\_ON,(CN8\_ENABLE | CN9\_ENABLE), 0);

// Set CN interrupts priority level 1 sub priority level 0

mCNSetIntPriority(1); // Group priority (1 to 7)

mCNSetIntSubPriority(0); // Subgroup priority (0 to 3)

// read port to clear difference

dummy = PORTReadBits(IOPORT\_G, BTN1 | BTN2);

mCNClearIntFlag(); // Clear CN interrupt flag

mCNIntEnable(1); // Enable CN interrupts

// Global interrupts must enabled to complete the initialization.

}

void \_\_ISR(\_CHANGE\_NOTICE\_VECTOR, IPL1) CN\_ISR(void)

{

LATBSET = LEDC; //start of CN ISR

unsigned int btns; //local var

//debounce the btns for 20 ms:

sw\_msDelay( DEBOUNCE\_TIME );

btns = read\_buttons();

decode\_buttons( btns, &step\_delay, &dir, &mode );

LATBCLR = LEDC; //end of CN ISR

mCNClearIntFlag(); //has to be at the very end?

}

//software delay for the specified number of ms:

void sw\_msDelay (unsigned int mS)

{

int i; //for loop cnting variable

while(mS --) // SW Stop breakpoint

{

for (i = 0; i< COUNTS\_PER\_MS; i++) // 1 ms delay loop

{

// do nothing

}

//LATBINV = LEDA; // Toggle LEDA each ms for instrumentation

//already flipped in T1 ISR

}

} //SW Stop breakpoint

//reads the status of BTN1 and BTN2:

int read\_buttons(void)

{

//return if btn1 and btn2 are set or not:

return PORTReadBits( IOPORT\_G, BTN1 | BTN2 );

}

//determines the values of motor direction, motor mode, and

// step delay using the rules specified in Table 2:

void decode\_buttons( unsigned int buttons, unsigned int \*step\_delay,

unsigned int \*dir, unsigned int \*mode)

{

//new step\_delay calced w/:

//T\_delay (ms/step) = 60000 (ms/min) / (X rev/min \* 100 steps/rev \* MODE)

switch( buttons )

{

case BTN1:

\*dir = CW;

\*mode = FS;

\*step\_delay = 40; //rpm = 15, FS

break;

case BTN2:

\*dir = CCW;

\*mode = HS;

\*step\_delay = 30; //rpm = 10, HS

break;

case BTN1 | BTN2: //both btns pressed

\*dir = CCW;

\*mode = FS;

\*step\_delay = 24; //rpm = 25, FS

break;

default: //neither btn pressed, or more than 2 pressed

\*dir = CW;

\*mode = HS;

\*step\_delay = 20; //rpm = 15, HS

break;

}

}

//determines the new output code for the stepper motor:

unsigned int sw\_fsm( unsigned int dir, unsigned int mode )

{

enum { S0 = 0, S0\_5, S1, S1\_5, S2, S2\_5, S3, S3\_5 }; /\* Declaration of states \*/

static unsigned int presState;

const unsigned int sm\_code[] = { 0x02, 0x0A, 0x08, 0x09, 0x01, 0x05, 0x04, 0x06 };

switch ( presState ) //NSL

{

case S0:

if ( dir == CW ) //if rotting CW

{

if( mode == HS )

{

presState = S0\_5;

}

else //full stepping

{

presState = S1;

}

}

else //CCW

{

if( mode == HS )

{

presState = S3\_5;

}

else //full stepping

{

presState = S3;

}

}

break;

case S0\_5:

if ( dir == CW ) //if rotting CW

{

if( mode == HS )

{

presState = S1;

}

else //full stepping

{

presState = S1\_5;

}

}

else //CCW

{

if( mode == HS )

{

presState = S0;

}

else //full stepping

{

presState = S3\_5;

}

}

break;

case S1:

if ( dir == CW ) //if rotting CW

{

if( mode == HS )

{

presState = S1\_5;

}

else //full stepping

{

presState = S2;

}

}

else //CCW

{

if( mode == HS )

{

presState = S0\_5;

}

else //full stepping

{

presState = S0;

}

}

break;

case S1\_5:

if ( dir == CW ) //if rotting CW

{

if( mode == HS )

{

presState = S2;

}

else //full stepping

{

presState = S2\_5;

}

}

else //CCW

{

if( mode == HS )

{

presState = S1;

}

else //full stepping

{

presState = S0\_5;

}

}

break;

case S2:

if ( dir == CW ) //if rotting CW

{

if( mode == HS )

{

presState = S2\_5;

}

else //full motorping

{

presState = S3;

}

}

else //CCW

{

if( mode == HS )

{

presState = S1\_5;

}

else //full stepping

{

presState = S1;

}

}

break;

case S2\_5:

if ( dir == CW ) //if rotting CW

{

if( mode == HS )

{

presState = S3;

}

else //full stepping

{

presState = S3\_5;

}

}

else //CCW

{

if( mode == HS )

{

presState = S2;

}

else //full stepping

{

presState = S1\_5;

}

}

break;

case S3:

if ( dir == CW ) //if rotting CW

{

if( mode == HS )

{

presState = S3\_5;

}

else //full stepping

{

presState = S0;

}

}

else //CCW

{

if( mode == HS )

{

presState = S2\_5;

}

else //full stepping

{

presState = S2;

}

}

break;

case S3\_5:

if ( dir == CW ) //if rotting CW

{

if( mode == HS )

{

presState = S0;

}

else //full stepping

{

presState = S0\_5;

}

}

else //CCW

{

if( mode == HS )

{

presState = S3;

}

else //full stepping

{

presState = S2\_5;

}

}

break;

}

return sm\_code[ presState ]; /\* Return next state \*/

}

//sends the four bit code to the stepper motor IO pins:

void output\_sm\_code( unsigned int sm\_code )

{

//READ-WRITE-MODIFY: ( LEDA-D which are Port B pins 2-6 kept same )

int temp, new\_data, mask;

mask = SM\_COILS; //SM\_COILS = (SM1 | SM2 | SM3 | SM4)

new\_data = (sm\_code << 7); //our new data is the step motor code;

// we shift it 7 bits to left since

// motor code is pins 7-10

// bc sm\_code = 1st 4 bits

temp = LATB; //read all 16 port B bits

temp = temp & ~mask; //clear all bits that need to be set by our stepper motor code

new\_data = new\_data & mask; //clear all non SM1-4 bits

temp = temp | new\_data; //recombine curr bits with new bits

LATB = temp;

}