/\*

\* CS122A\_LAB\_2.c

\*

\* Created: 10/4/18 9:14:57 AM

\* Author : juanruelas

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

#include "usart.h"

#include "lcd.h"

#include "new\_keypad.h"

/\* USED FOR CREATING A SYNCRONOUS STATE MACHINE\*/

volatile unsigned char TimerFlag = 0; // TimerISR() sets this to 1. C programmer should clear to 0.

// Internal variables for mapping AVR's ISR to our cleaner TimerISR model.

unsigned long \_avr\_timer\_M = 1; // Start count from here, down to 0. Default 1 ms.

unsigned long \_avr\_timer\_cntcurr = 0; // Current internal count of 1ms ticks

void TimerOn() {

// AVR timer/counter controller register TCCR1

TCCR1B = 0x0B;// bit3 = 0: CTC mode (clear timer on compare)

// bit2bit1bit0=011: pre-scaler /64

// 00001011: 0x0B

// SO, 8 MHz clock or 8,000,000 /64 = 125,000 ticks/s

// Thus, TCNT1 register will count at 125,000 ticks/s

// AVR output compare register OCR1A.

OCR1A = 125; // Timer interrupt will be generated when TCNT1==OCR1A

// We want a 1 ms tick. 0.001 s \* 125,000 ticks/s = 125

// So when TCNT1 register equals 125,

// 1 ms has passed. Thus, we compare to 125.

// AVR timer interrupt mask register

TIMSK1 = 0x02; // bit1: OCIE1A -- enables compare match interrupt

//Initialize avr counter

TCNT1=0;

\_avr\_timer\_cntcurr = \_avr\_timer\_M;

// TimerISR will be called every \_avr\_timer\_cntcurr milliseconds

//Enable global interrupts

SREG |= 0x80; // 0x80: 1000000

}

void TimerOff() {

TCCR1B = 0x00; // bit3bit1bit0=000: timer off

}

void TimerISR() {

TimerFlag = 1;

}

// In our approach, the C programmer does not touch this ISR, but rather TimerISR()

ISR(TIMER1\_COMPA\_vect) {

// CPU automatically calls when TCNT1 == OCR1 (every 1 ms per TimerOn settings)

\_avr\_timer\_cntcurr--; // Count down to 0 rather than up to TOP

if (\_avr\_timer\_cntcurr == 0) { // results in a more efficient compare

TimerISR(); // Call the ISR that the user uses

\_avr\_timer\_cntcurr = \_avr\_timer\_M;

}

}

// Set TimerISR() to tick every M ms

void TimerSet(unsigned long M) {

\_avr\_timer\_M = M/10;

\_avr\_timer\_cntcurr = \_avr\_timer\_M;

}

#define DD\_MOSI 4

#define DD\_SCK 5

#define DD\_MISO 6

#define DD\_SS 7

void SPI\_MasterInit(void) {

/\* Set MOSI and SCK output, all others input \*/

/\*

DDR\_SPI = (1<<DD\_MOSI)|(1<<DD\_SCK);

\*/

DDRB = ( 1 << DD\_MOSI ) | (1 << DD\_SCK) | (1 << DD\_SS);

DDRB = DDRB | (0 << DD\_MISO);

/\* Enable SPI, Master, set clock rate fck/16 \*/

SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);

}

void SPI\_MasterTransmit(char cData) {

/\* Start transmission \*/

DDRB = DDRB | (0 << DD\_SS);

SPDR = cData;

/\* Wait for transmission complete \*/

while(!(SPSR & (1<<SPIF))){

;

}

// set SS high

DDRB = DDRB | (1 << DD\_SS);

}

void SPI\_SlaveInit(void) {

/\* Set MISO output, all others input \*/

DDRB = ( 1 << DD\_MISO );

DDRB = DDRB | ( 0 << DD\_MOSI) | (0 << DD\_SCK) | (0 << DD\_SS);

/\* Enable SPI \*/

SPCR = (1<<SPE) | (1 << SPIE);

//asm("sei ;");

sei();

}

char SPI\_SlaveReceive(void) {

/\* Wait for reception complete \*/

while(!(SPSR & (1<<SPIF)))

;

/\* Return Data Register \*/

return SPDR;

}

void print(unsigned char k){

switch(k){

case '1':

LCD\_DisplayString(1, "1");

SPI\_MasterTransmit(0x01);

break;

case '2':

LCD\_DisplayString(1, "2");

SPI\_MasterTransmit(0x02);

break;

case '3':

LCD\_DisplayString(1, "3");

SPI\_MasterTransmit(0x03);

break;

case 'A':

LCD\_DisplayString(1, "A");

SPI\_MasterTransmit(0x0A);

break;

case '4':

LCD\_DisplayString(1, "4");

SPI\_MasterTransmit(0x04);

break;

case '5':

LCD\_DisplayString(1, "5");

SPI\_MasterTransmit(0x05);

break;

case '6':

LCD\_DisplayString(1, "6");

SPI\_MasterTransmit(0x06);

break;

case 'B':

LCD\_DisplayString(1, "B");

SPI\_MasterTransmit(0x0B);

break;

case '7':

LCD\_DisplayString(1, "7");

SPI\_MasterTransmit(0x07);

break;

case '8':

LCD\_DisplayString(1, "8");

SPI\_MasterTransmit(0x08);

break;

case '9':

LCD\_DisplayString(1, "9");

SPI\_MasterTransmit(0x09);

break;

case 'C':

LCD\_DisplayString(1, "C");

SPI\_MasterTransmit(0x0C);

break;

case '\*':

LCD\_DisplayString(1, "\*");

SPI\_MasterTransmit(0x0E);

break;

case '0':

LCD\_DisplayString(1, "0");

SPI\_MasterTransmit(0x00);

break;

case '#':

LCD\_DisplayString(1, "#");

SPI\_MasterTransmit(0x0F);

break;

case 'D':

LCD\_DisplayString(1, "D");

SPI\_MasterTransmit(0x0D);

break;

}

}

ISR(SPI\_STC\_vect) { // this is enabled in with the SPCR register's "SPI

// Interrupt Enable"

// SPDR contains the received data, e.g. unsigned char receivedData =

// SPDR;

PORTA = SPI\_SlaveReceive();

}

int main(void) {

/\* Replace with your application code \*/

DDRA = 0xFF; PORTA = 0x00; // Configure port A's 8 pins at outputs

DDRB = 0xFF; PORTB = 0x00; // Configure port B's 8 pins as outputs

DDRC = 0xFF; PORTC = 0x00; // Configure port C's 8 pins as outputs

DDRD = 0xF0; PORTD = 0x0F; // Configure port D's 8 pins as inputs

//TimerSet(100);

//TimerOn();

// initialize usart 0 for both leader and follower

//initUSART(0);

LCD\_init();

LCD\_DisplayString(1, "Hello World");

unsigned char key = GetKeypadKey();

SPI\_MasterInit();

//SPI\_SlaveInit();

while (1) {

key = GetKeypadKey();

print(key);

//continue;

}

}