/\*

\* CS122A\_LAB\_2.c

\*

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

\* Author : juanruelas

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <avr/io.h>

#include "io.c"

#include <avr/interrupt.h>

#include "usart.h"

//#include "lcd.h"

#include "keypad.h"

char\* concat(const char \*s1, const char \*s2)

{

const size\_t str1len = strlen(s1);

const size\_t str2len = strlen(s2);

char \*result = malloc(str1len + str2len + 1); // null-terminator

memcpy(result, s1, str1len);

memcpy(result + str1len, s2, str2len + 1); // null-terminator

return result;

}

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

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

/\* for the slave\*/

unsigned char curr\_speed = 0x01;

unsigned char curr\_pattern = 0x01;

/\* for the master\*/

unsigned char pattern = 0x30;

unsigned char speed = 0x03;

unsigned char transmit = 0x00;

// 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

char\* temp; // variable for string concat

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))) {

//wait

}

/\* Return Data Register \*/

return SPDR;

}

char \* getpattern(){

char \* string = malloc(1);

switch( (pattern) >> 4){

case 0x01:

strcpy(string, "1");

return string;

case 0x02:

strcpy(string, "2");

return string;

case 0x03:

strcpy(string, "3");

return string;

case 0x04:

strcpy(string, "4");

return string;

}

}

char \* getspeed(){

char \* string = malloc(1);

switch( speed ){

case 0x01:

strcpy(string, "1");

return string;

case 0x02:

strcpy(string, "2");

return string;

case 0x03:

strcpy(string, "3");

return string;

case 0x04:

strcpy(string, "4");

return string;

case 0x05:

strcpy(string, "5");

return string;

case 0x06:

strcpy(string, "6");

return string;

}

}

void print(unsigned char k){

switch(k){

case '1':

speed = 0x01; // we set the speed to 2 second period

transmit = pattern | speed; // merge the speed and pattern together

temp = concat("Ptrn:", getpattern());

temp = concat(temp, " Spd:");

temp = concat(temp, getspeed());

temp = concat(temp, " uC:1");

LCD\_DisplayString(1, temp);

SPI\_MasterTransmit(transmit);

break;

case '2':

speed = 0x02; // we set the speed to 2 second period

transmit = pattern | speed; // merge the speed and pattern together

temp = concat("Ptrn:", getpattern());

temp = concat(temp, " Spd:");

temp = concat(temp, getspeed());

temp = concat(temp, " uC:1");

LCD\_DisplayString(1, temp);

SPI\_MasterTransmit(transmit);

break;

case '3':

speed = 0x03; // we set the speed to 2 second period

transmit = pattern | speed; // merge the speed and pattern together

temp = concat("Ptrn:", getpattern());

temp = concat(temp, " Spd:");

temp = concat(temp, getspeed());

temp = concat(temp, " uC:1");

LCD\_DisplayString(1, temp);

SPI\_MasterTransmit(transmit);

break;

case 'A':

pattern = 0x10; // we set the speed to 2 second period

transmit = pattern | speed; // merge the speed and pattern together

temp = concat("Ptrn:", getpattern());

temp = concat(temp, " Spd:");

temp = concat(temp, getspeed());

temp = concat(temp, " uC:1");

LCD\_DisplayString(1, temp);

SPI\_MasterTransmit(transmit);

break;

case '4':

speed = 0x04; // we set the speed to 2 second period

transmit = pattern | speed; // merge the speed and pattern together

temp = concat("Ptrn:", getpattern());

temp = concat(temp, " Spd:");

temp = concat(temp, getspeed());

temp = concat(temp, " uC:1");

LCD\_DisplayString(1, temp);

SPI\_MasterTransmit(transmit);

break;

case '5':

speed = 0x05; // we set the speed to 2 second period

transmit = pattern | speed; // merge the speed and pattern together

temp = concat("Ptrn:", getpattern());

temp = concat(temp, " Spd:");

temp = concat(temp, getspeed());

temp = concat(temp, " uC:1");

LCD\_DisplayString(1, temp);

SPI\_MasterTransmit(transmit);

break;

case '6':

speed = 0x06; // we set the speed to 2 second period

transmit = pattern | speed; // merge the speed and pattern together

temp = concat("Ptrn:", getpattern());

temp = concat(temp, " Spd:");

temp = concat(temp, getspeed());

temp = concat(temp, " uC:1");

LCD\_DisplayString(1, temp);

SPI\_MasterTransmit(transmit);

break;

case 'B':

pattern = 0x20; // we set the speed to 2 second period

transmit = pattern | speed; // merge the speed and pattern together

temp = concat("Ptrn:", getpattern());

temp = concat(temp, " Spd:");

temp = concat(temp, getspeed());

temp = concat(temp, " uC:1");

LCD\_DisplayString(1, temp);

SPI\_MasterTransmit(transmit);

break;

case '7':

// do nothing since there is no speed 7

break;

case '8':

// do nothing since there is no speed 8

break;

case '9':

// do nothing since there is no speed 9

break;

case 'C':

pattern = 0x30; // we set the speed to 2 second period

transmit = pattern | speed; // merge the speed and pattern together

temp = concat("Ptrn:", getpattern());

temp = concat(temp, " Spd:");

temp = concat(temp, getspeed());

temp = concat(temp, " uC:1");

LCD\_DisplayString(1, temp);

SPI\_MasterTransmit(transmit);

break;

case '\*':

// do nothing since this is not an option

break;

case '0':

// do nothing since this is not a speed option

break;

case '#':

// do nothing since this is not a speed options

break;

case 'D':

pattern = 0x40; // we set the speed to 2 second period

transmit = pattern | speed; // merge the speed and pattern together

temp = concat("Ptrn:", getpattern());

temp = concat(temp, " Spd:");

temp = concat(temp, getspeed());

temp = concat(temp, " uC:1");

LCD\_DisplayString(1, temp);

SPI\_MasterTransmit(transmit);

break;

}

}

/\* THE FOLLOWING CODE WILL ONLY BE EXECUTED BY THE SLAVE(S)\*/

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;

//unsigned char data = SPI\_SlaveReceive();

unsigned char data = SPDR;

// first half of byte (first four bits) are the pattern

// second half of byte (last four bits) are the pattern

curr\_speed = data & 0x0F;

curr\_pattern = (data & 0xF0) >> 4;

//PORTA = data;

}

enum LEFT\_RIGHT { LEFT, RIGHT} LEFT\_RIGHT;

enum OFFSET {OFFSET\_LEFT, OFFSET\_RIGHT} OFFSET;

enum COUNT { ONE, TWO, FOUR, EIGHT, SIXTEEN, THIRTY\_TWO, SIXTY\_FOUR, ONE\_TWENTY\_EIGHT} COUNT;

enum CUSTOM { A\_PATTERN, B\_PATTERN } CUSTOM;

void left\_right\_tick(){

// transitions

switch(LEFT\_RIGHT){

case LEFT:

LEFT\_RIGHT = RIGHT;

break;

case RIGHT:

LEFT\_RIGHT = LEFT;

break;

default:

LEFT\_RIGHT = LEFT;

break;

}

// actions

switch(LEFT\_RIGHT){

case LEFT:

PORTA = 0xF0;

break;

case RIGHT:

PORTA = 0x0F;

break;

}

}

void offset\_tick(){

// transitions

switch(OFFSET){

case OFFSET\_LEFT:

OFFSET = OFFSET\_RIGHT;

break;

case OFFSET\_RIGHT:

OFFSET = OFFSET\_LEFT;

break;

default:

OFFSET = OFFSET\_LEFT;

}

// actions

switch(OFFSET){

case OFFSET\_LEFT:

PORTA = 0b10101010;

break;

case OFFSET\_RIGHT:

PORTA = 0b01010101;

break;

}

}

void counter\_tick(){

// transitions

switch(COUNT){

case ONE:

COUNT = TWO;

break;

case TWO:

COUNT = FOUR;

break;

case FOUR:

COUNT = EIGHT;

break;

case EIGHT:

COUNT = SIXTEEN;

break;

case SIXTEEN:

COUNT = THIRTY\_TWO;

break;

case THIRTY\_TWO:

COUNT = SIXTY\_FOUR;

break;

case SIXTY\_FOUR:

COUNT = ONE\_TWENTY\_EIGHT;

break;

case ONE\_TWENTY\_EIGHT:

COUNT = ONE;

break;

}

// actions

switch(COUNT){

case ONE:

PORTA = 0x01;

break;

case TWO:

PORTA = 0x02;

break;

case FOUR:

PORTA = 0x04;

break;

case EIGHT:

PORTA = 0x08;

break;

case SIXTEEN:

PORTA = 0x10;

break;

case THIRTY\_TWO:

PORTA = 0x20;

break;

case SIXTY\_FOUR:

PORTA = 0x40;

break;

case ONE\_TWENTY\_EIGHT:

PORTA = 0x80;

break;

}

}

void custom\_tick(){

// transitions

switch(CUSTOM){

case A\_PATTERN:

CUSTOM = B\_PATTERN;

break;

case B\_PATTERN:

CUSTOM = A\_PATTERN;

break;

default:

CUSTOM = A\_PATTERN;

}

switch(CUSTOM){

case A\_PATTERN:

PORTA = 0b00110011;

break;

case B\_PATTERN:

PORTA = 0b11001100;

break;

}

// actions

}

void master\_main(){

LCD\_init(); // Intialize the LCD Display

//LCD\_DisplayString(1, "Ptrn:1 Spd:1 uC:1");

unsigned char key;

SPI\_MasterInit();

while(1){

key = GetKeypadKey();

print(key);

}

}

void output\_curr\_pattern(){

switch(curr\_pattern){

case 0x01:

left\_right\_tick();

break;

case 0x02:

offset\_tick();

break;

case 0x03:

counter\_tick();

break;

case 0x04:

custom\_tick();

break;

}

}

void slave\_main(){

TimerSet(50);

TimerOn();

SPI\_SlaveInit();

unsigned int t = 0;

while(1){

while (!TimerFlag);

TimerFlag = 0;

t+= 50;

switch(curr\_speed){

case 0x01:

if( (t % 2000) == 0){

output\_curr\_pattern(); // tick the 2s period

}

break;

case 0x02:

if( (t % 1000) == 0){

output\_curr\_pattern(); // tick the 1s period

}

break;

case 0x03:

if( (t % 500) == 0){

output\_curr\_pattern(); // tick the 500ms period

}

break;

case 0x04:

if( (t % 250) == 0){

output\_curr\_pattern();; // tick the 250ms period

}

break;

case 0x05:

if( (t % 100) == 0){

output\_curr\_pattern(); // tick the 100ms period

}

break;

case 0x06:

if( (t % 50) == 0){

output\_curr\_pattern(); // tick the 50ms period

}

break;

}

if(t > 8000){

t = 0;

}

}

}

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

while(1)

{

//master\_main();

slave\_main();

}

}