/\*

\* 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"

/\* 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;

}

/\*\*/

enum LEADER\_STATES { ON\_L, OFF\_L} LEADER\_STATE;

enum FOLLOWER\_STATES { ON\_F, OFF\_F} FOLLOWER\_STATE;

void leader\_tick(){

// transitions for leader

switch(LEADER\_STATE){

case ON\_L :

LEADER\_STATE = OFF\_L;

break;

case OFF\_L :

LEADER\_STATE = ON\_L;

break;

default:

LEADER\_STATE = OFF\_L;

break;

}

// actions for leader

switch (LEADER\_STATE){

case ON\_L:

// here we transmit a 1

// here we set LED to high

if(USART\_IsSendReady(0)){

USART\_Send(0xFF, 0);

}

PORTA = 0x01;

break;

case OFF\_L:

// here we transmit a 0

// here we set LED to low

if(USART\_IsSendReady(0)){

USART\_Send(0x00, 0);

}

PORTA = 0x00;

break;

default:

// here we don't transmit anything

// here we don't do anything

break;

}

}

void follower\_tick(){

// transitions for leader

if(USART\_HasReceived(0)){

unsigned char data = USART\_Receive(0);

switch(FOLLOWER\_STATE){

case ON\_F :

// if we receive

if(data == 0x00){

FOLLOWER\_STATE = OFF\_F;

}

break;

case OFF\_F :

if(data == 0xFF){

FOLLOWER\_STATE = ON\_F;

}

break;

default:

// do nothing

break;

}

// actions for leader

switch (FOLLOWER\_STATE){

case ON\_F:

PORTA = 0x01;

break;

case OFF\_L:

PORTA = 0x00;

break;

default:

// here we don't do anything

break;

}

}

}

int main(void) {

/\* Replace with your application code \*/

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

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

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

TimerSet(100);

TimerOn();

// initialize usart 0 for both leader and follower

initUSART(0);

unsigned int t = 0;

while (1) {

while (!TimerFlag);

TimerFlag = 0;

t += 100;

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

// leader\_tick();

// t = 0;

//}

follower\_tick();

}

}