/\*

\* Lab\_Elective\_1.c

\*

\* Created: 10/22/2018 8:19:15 PM

\* Author : David

\*/

#define F\_CPU 1000000UL // 1 MHz

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdbool.h>

#include <util/delay.h>

#include <avr/io.h>

#include <avr/interrupt.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;

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

}

unsigned short input;

unsigned char lit\_led = 0x01; // illuminated LED

// ADEN: Enables analog-to-digital conversion

// ADSC: Starts analog-to-digital conversion

// ADATE: Enables auto-triggering, allowing for constant

// analog to digital conversions.

// Note: Do not need to set the DDRA to enable the Analog to Digital circuitry

void A2D\_init() {

ADCSRA |= (1 << ADEN) | (1 << ADSC) | (1 << ADATE);

}

unsigned short adc\_read() {

// start single conversion

// write '1' to ADIF

ADCSRA |= (1<<ADIF);

// wait for conversion to complete

// ADIF becomes '0' again

while( ADCSRA & (1<<ADIF));

return (ADC);

}

// Pins on PORTA are used as input for A2D conversion

// The default channel is 0 (PA0)

// The value of pinNum determines the pin on PORTA

// used for A2D conversion

// Valid values range between 0 and 7, where the value

// represents the desired pin for A2D conversion

void Set\_A2D\_Pin(unsigned char pinNum)

{

ADMUX = (pinNum <= 0x07) ? pinNum : ADMUX;

// Allow channel to stabilize

static unsigned char i = 0;

for ( i=0; i<15; i++ )

{

asm("nop");

}

}

enum LED\_States {Wait, Left, Right} LED\_State;

void LED\_Tick()

{

// Transitions

switch (LED\_State)

{

case Wait:

if( input > 450 && input < 550 ) {

LED\_State = Wait;

}

else if( input < 450 ) {

LED\_State = Right;

}

else if( input > 550 ) {

LED\_State = Left;

}

break;

case Left:

if( input < 450 ) {

LED\_State = Right;

}

else if( input > 550 ) {

LED\_State = Left;

}

else {

LED\_State = Wait;

}

break;

case Right:

if( input < 450 ) {

LED\_State = Right;

}

else if( input > 550 ) {

LED\_State = Left;

}

else {

LED\_State = Wait;

}

break;

default:

LED\_State= Wait;

break;

}

// Actions

switch (LED\_State)

{

case Wait:

break;

case Left:

if( lit\_led == 0x80 ) {

lit\_led = 0x01;

}

else {

lit\_led = lit\_led << 1;

}

break;

case Right:

if( lit\_led == 0x01 ) {

lit\_led = 0x80;

}

else {

lit\_led = lit\_led >> 1;

}

break;

}

}

void displayCol(){

if(ADC <= 0x80){

// 128, 1

PORTC = 0x80;

} else if(ADC > 0x80 && ADC <= 0x100){

// 256, 2

PORTC = 0x40;

} else if(ADC > 0x100 && ADC <= 0x180){

// 384, 4

PORTC = 0x20;

} else if(ADC > 0x180 && ADC <= 0x200 ){

// 512, 8

PORTC = 0x10;

} else if(ADC > 0x200 && ADC <= 0x280){

// 640, 16

PORTC = 0x08;

} else if(ADC > 0x280 && ADC <= 0x300){

// 768, 32

PORTC = 0x04;

} else if (ADC > 0x300 && ADC <= 0x380){

// 896, 64

PORTC = 0x02;

} else if(ADC > 0x380 && ADC <= 1024){

// 1024, 128

PORTC = 0x01;

}

}

void selectRow(){

if(ADC <= 0x80){

// 128, 1

//PORTD = 0xFE;

PORTD = 0x7F;

} else if(ADC > 0x80 && ADC <= 0x100){

// 256, 2

//PORTD = 0xFD;

PORTD = 0xBF;

} else if(ADC > 0x100 && ADC <= 0x180){

// 384, 4

//PORTD = 0xFB;

PORTD = 0xDF;

} else if(ADC > 0x180 && ADC <= 0x200 ){

// 512, 8

//PORTD = 0xF7;

PORTD = 0xEF;

} else if(ADC > 0x200 && ADC <= 0x280){

// 640, 16

//PORTD = 0xEF;

PORTD = 0xF7;

} else if(ADC > 0x280 && ADC <= 0x300){

// 768, 32

//PORTD = 0xDF;

PORTD = 0xFB;

} else if (ADC > 0x300 && ADC <= 0x380){

// 896, 64

//PORTD = 0xBF;

PORTD = 0xFD;

} else if(ADC > 0x380 && ADC <= 1024){

// 1024, 128

//PORTD = 0x6F;

PORTD = 0xFE;

}

}

int main(void) {

A2D\_init();

TimerSet(10);

TimerOn();

/\* Replace with your application code \*/

DDRA = 0x00; PORTA = 0xFF; // PORTA Input

DDRD = 0xFF; PORTD = 0x00; // PORTD Output

DDRC = 0xFF; PORTC = 0x00; // PORTC Output

while (1) {

while(!TimerFlag);

TimerFlag = 0;

// first we must select the row

//Set\_A2D\_Pin(0x01);

//selectRow();

// must have at least 1ms delay to allow the mux to have the right values selected

//\_delay\_ms(5);

// then we select the column (i.e. show the LED)

//Set\_A2D\_Pin(0x00);

displayCol();

//PORTC = ADC;

PORTD = 0xFE;

}

}