/\*

\* PS2.c

\*

\* Created: 19-06-2018 17:56:38

\* Author : Sahil

\*/

#define *F\_CPU* 14745600UL

#include <avr/io.h>

#include <util/delay.h>

#include <avr/eeprom.h>

#include <math.h>

#include <avr/interrupt.h>

#define BAUD 9600 //serial com;

#define BAUDRATE ((*F\_CPU*/(BAUD\*16UL)-1))

*uint8\_t* RX[16]={100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100};

int RX\_range=200,RX\_raw=255,RX\_ad=255,RX\_count=0;

*uint8\_t* TX[16]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};

int flag[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

*uint8\_t* TX\_raw=200,TX\_ad=201,TX\_flag=0;

#define indicator\_1 PORTB

//variable for ps2

#define PS\_L1 10

#define PS\_R1 11

#define PS\_L2 8

#define PS\_R2 9

#define PS\_L3 6

#define PS\_R3 7

#define PS\_SQUARE 14

#define PS\_TRIANGLE 12

#define PS\_CIRCLE 13

#define PS\_CROSS 15

#define PS\_UP 2

#define PS\_DOWN 5

#define PS\_LEFT 4

#define PS\_RIGHT 3

#define PS\_START 0

#define PS\_SELECT 1

int xj1=0,yj1=0,xj2=0,yj2=0,pot1=0,pot2=0,pot3=0,pot4=0; //analog values(serially received from remote);

int butt[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //digital values(serially received from remote);

int js\_error=20;

int pwm\_range=255;

void receive();

void serialstart\_3();

*uint16\_t* ADC\_read(*uint8\_t* ch);

void ADC\_initiate();

void port\_init();

long map\_value(long in\_value, long in\_min, long in\_max, long out\_min, long out\_max);

long limit\_var(long in\_var, long l\_limit, long h\_limit);

int main(void)

{

port\_init();

sei();

ADC\_initiate();

serialstart\_3();

while (1)

{

receive();

}

}

void port\_init()

{

DDRB=0xFF; //indicator\_1

}

void receive()

{

yj1=map\_value(RX[0],0,RX\_range,(-pwm\_range),pwm\_range);

xj1=map\_value(RX[1],0,RX\_range,pwm\_range,(-pwm\_range));

yj2=map\_value(RX[2],0,RX\_range,(-pwm\_range),pwm\_range);

xj2=map\_value(RX[3],0,RX\_range,pwm\_range,(-pwm\_range));

if (butt[PS\_START]==1)

{

indicator\_1^=0xFF;

butt[PS\_START]=0;

}

if (butt[PS\_SELECT]==1)

{

indicator\_1^=0xFF;

butt[PS\_SELECT]=0;

}

if (butt[PS\_UP]==1)

{

indicator\_1^=0xFF;

butt[PS\_UP]=0;

}

if (butt[PS\_DOWN]==1)

{

indicator\_1^=0xFF;

butt[PS\_DOWN]=0;

}

if (butt[PS\_LEFT]==1)

{

indicator\_1^=0xFF;

butt[PS\_LEFT]=0;

}

if (butt[PS\_RIGHT]==1)

{

indicator\_1^=0xFF;

butt[PS\_RIGHT]=0;

}

if (butt[PS\_SQUARE]==1)

{

indicator\_1^=0xFF;

butt[PS\_SQUARE]=0;

}

if (butt[PS\_CIRCLE]==1)

{

indicator\_1^=0xFF;

butt[PS\_CIRCLE]=0;

}

if (butt[PS\_TRIANGLE]==1)

{

indicator\_1^=0xFF;

butt[PS\_TRIANGLE]=0;

}

if (butt[PS\_CROSS]==1)

{

indicator\_1^=0xFF;

butt[PS\_CROSS]=0;

}

if (butt[PS\_L1]==1)

{

indicator\_1^=0xFF;

butt[PS\_L1]=0;

}

if (butt[PS\_R1]==1)

{

indicator\_1^=0xFF;

butt[PS\_R1]=0;

}

if (butt[PS\_L2]==1)

{

indicator\_1^=0xFF;

butt[PS\_L2]=0;

}

if (butt[PS\_R2]==1)

{

indicator\_1^=0xFF;

butt[PS\_R2]=0;

}

if (butt[PS\_L3]==1)

{

indicator\_1^=0xFF;

butt[PS\_L3]=0;

}

if (butt[PS\_R3]==1)

{

indicator\_1^=0xFF;

butt[PS\_R3]=0;

}

}

void ADC\_initiate()

{

ADMUX = (0<<REFS1)|(1<<REFS0)|(0<<ADLAR); // AVcc // right adjusted

ADCSRA = (1<<ADEN)|(0<<ADATE)|(0<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); // bit4 ADC EOC flag // prescalar- 111 - 128 division factor

ADCSRB = 0x00;

}

*uint16\_t* ADC\_read(*uint8\_t* ch)

{

ADMUX = ADMUX & 0b11100000; //Clearing all the mux;

ADCSRB = ADCSRB & 0b11110111; //------"-"-----------

ch = ch & 0b00001111;

if ( ch <= 7 )

{

ch = ch & 0b00000111; //

ADMUX = ADMUX | ch;

ADCSRB=0x00;

}

else

{

ch = ch-8;

ch = ch & 0b00000111;

ADMUX = ADMUX | ch;

ADCSRB=0x00;

ADCSRB = ADCSRB | (1<<MUX5);

}

ADCSRA = ADCSRA | (1<<ADSC); //Bit 6 to start conversion-ADSC

while( !(ADCSRA & (1<<ADIF)) ); // Wait for conversion to complete

return(ADC);

}

void serialstart\_3()

{

UBRR3H=BAUDRATE>>8;

UBRR3L=BAUDRATE;

UCSR3B=0b10011000;//enable RXEN TXEN

UCSR3C=0b00000110;// UCSZ1 UCSZ0

}

ISR(USART3\_RX\_vect)

{

RX\_count=1;

RX\_raw=UDR3;

if ((RX\_raw>200) && (RX\_raw<255)) //201 to 216 for addresses of analog values, 231 to 246 for buttons;

{

RX\_ad=RX\_raw;

if ((RX\_raw>230) && (RX\_raw<247))

{

*uint8\_t* r\_temp0=(RX\_raw-231);

butt[r\_temp0]=1;

}

}

else if ((RX\_raw>=0) && (RX\_raw<201))

{

*uint8\_t* r\_temp1=(RX\_ad-201);

if (r\_temp1<16)

{

RX[r\_temp1]=RX\_raw;

}

}

}

long map\_value(long in\_value, long in\_min, long in\_max, long out\_min, long out\_max)

{

return (in\_value - in\_min) \* (out\_max - out\_min) / (in\_max - in\_min) + out\_min;

}

long limit\_var(long in\_var, long l\_limit, long h\_limit)

{

if (in\_var>h\_limit)

{

in\_var=h\_limit;

}

else if (in\_var<l\_limit)

{

in\_var=l\_limit;

}

return in\_var;

}