sOS Source Code

Build date: Saturday, December 9, 2023.

GIO CHOI

2023

TOOLBOX.h

Default Header File we use.

/\*

\* TOOLBOX.h

\*

\* Created: 7/26/2023 1:41:46 AM

\* Author: 6gioc

\*/

#pragma once

#ifndef TOOLBOX\_H\_

#define TOOLBOX\_H\_

#endif /\* TOOLBOX\_H\_ \*/

#define F\_CPU 16000000

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define T2INTITIAL 6

enum \_adc\_list{

\_x\_joystick,

\_y\_joystick,

\_tempature,

\_light,

};

enum MODELIST{

\_stop,

\_auto,

\_manual,

};

enum CONAMICODE\_JOYSITICK\_ENTER{

\_up,

\_down,

\_left,

\_right,

\_nodirection,

};

void startup\_(void);

void update\_(void);

void display\_setup\_(void);

void display\_update\_(void);

void stepmotor\_setup\_(void);

void stepmotor\_update\_(signed int \_xval, signed int \_yval);

void r\_a\_comm\_setup\_(void);

void r\_a\_comm\_update\_(void);

void portacontroler\_(unsigned char \_bit, unsigned char \_logical\_value);

void portbcontroler\_(unsigned char \_bit, unsigned char \_logical\_value);

void portccontroler\_(unsigned char \_bit, unsigned char \_logical\_value);

void portgcontroler\_(unsigned char \_bit, unsigned char \_logical\_value);

unsigned char getbitofbyte\_(unsigned char \_byte, unsigned char \_bit);

unsigned int abs\_(signed int \_value);

extern unsigned char \_mind[8];

extern unsigned char \_mode;

extern unsigned char \_mode\_list[10][50];

extern unsigned int \_dotmatrix\_animation\_move;

extern unsigned char \_dotmatrix\_brightness;

extern unsigned int \_auto\_manual\_switch\_timer\_miliseconds;

extern unsigned char \_tutorial\_stage;

extern unsigned char \_conami\_code\_progress;

MAESTRO.cpp

Main Function is in here.

/\*

\* sOS.cpp

\*

\* Created: 7/26/2023 1:37:32 AM

\* Author : 6gioc

\* Designed to operate on Atmega128

\*/

#include "TOOLBOX.h"

int main(void)

{

startup\_();

while (TRUE){

update\_();

}

}

CORE.cpp

Common functions are defined here.

/\*

\* DEFAULT\_FUNCTION.cpp

\*

\* Created: 7/26/2023 1:40:26 AM

\* Author: 6gioc

\*/

#include "TOOLBOX.h"

#define JOYSTICK\_DEAD\_ZONE 32

unsigned char \_dotmatrix\_animation\_counter=0;

unsigned int \_dotmatrix\_animation\_move=0;

unsigned char \_adc\_target=\_x\_joystick;

unsigned int \_x\_read=0;

unsigned int \_y\_read=0;

unsigned char \_adc\_counter=0;

unsigned int \_tempature\_read=0;

unsigned int \_light\_read=0;

unsigned int \_x\_motor\_ocr=0;

unsigned int \_y\_motor\_ocr=0;

unsigned char \_mind[8];

unsigned char \_mode=\_manual;

unsigned char \_mode\_list[10][50]={"Stop", "Auto", "Manual", "Use Joystick to aim target", "Press button to fire"};

unsigned char \_tutorial\_stage = 0;

unsigned char \_conami\_code\_is\_entered=FALSE;

const unsigned char \_conami\_code[] = {\_up,\_up,\_down,\_down,\_left,\_right,\_left,\_right};

unsigned char \_conami\_code\_progress=0;

unsigned char \_conami\_code\_entered\_value=\_nodirection;

unsigned char \_fan\_pwm\_boolean = FALSE;

unsigned int \_auto\_manual\_switch\_timer\_miliseconds = 0;

void startup\_(void){

DDRA=0x00;

DDRB=0xff;

DDRC=0xff;

DDRD=0x00;

DDRE=0x00;

DDRF=0x00;

DDRG=0xff;

ADMUX=0;

ADCSRA=(TRUE<<ADEN)|(TRUE<<ADPS0)|(TRUE<<ADPS1)|(TRUE<<ADPS2)|(TRUE<<ADIE);

TCCR0=(TRUE<<WGM01)|(TRUE<<CS02)|(TRUE<<CS01)|(TRUE<<CS00);

OCR0=0;//todo last

TCCR1A=0;

TCCR1B=(TRUE<<WGM12)|(TRUE<<CS10)|(TRUE<<CS12);

TCCR1C=0;

OCR1AH=0;//todo last

OCR1AL=0;//todo last

TCNT2=T2INTITIAL;

TCCR2=(TRUE<<CS21)|(TRUE<<CS20);

TCCR3A=0;

TCCR3B=(TRUE<<WGM32)|(TRUE<<CS30)|(TRUE<<CS32);

TCCR3C=0;

OCR3AH=0;//todo last

OCR3AL=0;//todo last

TIMSK=(TRUE<<OCIE1A)|(TRUE<<TOIE2);

ETIMSK=(TRUE<<OCIE3A);

sei();

display\_setup\_();

stepmotor\_setup\_();

}

void update\_(void){

display\_update\_();

ADCSRA=ADCSRA|(TRUE<<ADSC);

while((ADCSRA&(TRUE<<ADEN))==TRUE);

r\_a\_comm\_update\_();

}

void portacontroler\_(unsigned char \_bit, unsigned char \_logical\_value){

unsigned char \_porta=(TRUE<<\_bit);

\_porta=~\_porta;

\_porta=\_porta&PORTA;

\_porta=\_porta|(\_logical\_value<<\_bit);

PORTA=\_porta;

return;

}

void portbcontroler\_(unsigned char \_bit, unsigned char \_logical\_value){

unsigned char \_portb=(TRUE<<\_bit);

\_portb=~\_portb;

\_portb=\_portb&PORTB;

\_portb=\_portb|(\_logical\_value<<\_bit);

PORTB=\_portb;

return;

}

void portccontroler\_(unsigned char \_bit, unsigned char \_logical\_value){

unsigned char \_newportc=0;

for(unsigned char \_digit=0;\_digit<8;\_digit++){

if(\_digit==\_bit){

\_newportc=\_newportc|(\_logical\_value<<\_bit);

}

else{

\_newportc=\_newportc|(PORTC&(TRUE<<\_digit));

}

}

PORTC=\_newportc;

}

void portgcontroler\_(unsigned char \_bit, unsigned char \_logical\_value){

unsigned char \_portg=(TRUE<<\_bit);

\_portg=~\_portg;

\_portg=\_portg&PORTG;

\_portg=\_portg|(\_logical\_value<<\_bit);

PORTG=\_portg;

return;

}

unsigned char getbitofbyte\_(unsigned char \_byte, unsigned char \_bit){

return (\_byte>>\_bit)&0x01;

}

void mindwizard\_(unsigned int \_xval, unsigned int \_yval){

for(unsigned char \_erasey=0;\_erasey<8;\_erasey++){

\_mind[\_erasey]=0;

}

signed int \_xvalsigned=(signed int)\_xval-512;

signed int \_yvalsigned=(signed int)\_yval-512;

unsigned char \_xline;

if(abs\_(\_xvalsigned)<JOYSTICK\_DEAD\_ZONE){

\_xval=512;

\_xvalsigned=0;

}

if(abs\_(\_yvalsigned)<JOYSTICK\_DEAD\_ZONE){

\_yval=512;

\_yvalsigned=0;

}

if(\_xvalsigned == 0 && \_yvalsigned == 0 && \_conami\_code\_entered\_value != \_nodirection){

if(\_conami\_code\_progress>=8){

\_conami\_code\_progress=0;

}

if(\_conami\_code[\_conami\_code\_progress]==\_conami\_code\_entered\_value){

\_conami\_code\_progress++;

}

else{

\_conami\_code\_progress=0;

}

\_conami\_code\_entered\_value = \_nodirection;

\_conami\_code\_is\_entered = FALSE;

}

if(\_yvalsigned !=0 || \_xvalsigned !=0){

\_mode=\_manual;

\_auto\_manual\_switch\_timer\_miliseconds = 0;

}

if(\_mode == \_auto){

return;

}

\_xval=\_xval/64;

if(\_xval%2==0){

\_xline=3<<(7-(\_xval/2));

}

else{

\_xline=1<<(7-(\_xval/2));

}

\_yval=\_yval/64;

if(\_yval%2==0){

\_mind[\_yval/2]=\_xline;

if(\_yval>=1){

\_mind[\_yval/2-1]=\_xline;

}

}

else{

\_mind[\_yval/2]=\_xline;

}

stepmotor\_update\_(\_xvalsigned, \_yvalsigned);

if(\_xvalsigned>500 && \_conami\_code\_is\_entered == FALSE){

\_conami\_code\_entered\_value = \_right;

\_conami\_code\_is\_entered = TRUE;

}

else if(\_xvalsigned<-500 && \_conami\_code\_is\_entered == FALSE){

\_conami\_code\_entered\_value = \_left;

\_conami\_code\_is\_entered = TRUE;

}

else if(\_yvalsigned>500 && \_conami\_code\_is\_entered == FALSE){

\_conami\_code\_entered\_value = \_down;

\_conami\_code\_is\_entered = TRUE;

}

else if(\_yvalsigned<500 && \_conami\_code\_is\_entered == TRUE){

\_conami\_code\_entered\_value = \_up;

\_conami\_code\_is\_entered = TRUE;

}

return;

}

unsigned int abs\_(signed int \_value){

if(\_value<0){

return \_value \* -1;

}

else{

return \_value;

}

}

ISR(TIMER0\_COMP\_vect){//fan control

const unsigned char \_fan\_speed = \_tempature\_read>>2;

if(\_fan\_pwm\_boolean == TRUE){

OCR0=\_fan\_speed;

\_fan\_pwm\_boolean = FALSE;

portbcontroler\_(5,TRUE);

}

else{

\_fan\_pwm\_boolean = TRUE;

OCR0=255-\_fan\_speed;

if(\_fan\_speed == 255){

OCR0=1;

}

portbcontroler\_(5,FALSE);

}

}

ISR(TIMER2\_OVF\_vect){//1ms

TCNT2=T2INTITIAL;

\_dotmatrix\_animation\_counter++;

if(\_dotmatrix\_animation\_counter==100){//100ms

\_dotmatrix\_animation\_counter=0;

\_dotmatrix\_animation\_move++;

if(\_dotmatrix\_animation\_move>=7\**strlen*((char\*)\_mode\_list[\_mode])){

\_dotmatrix\_animation\_move=0;

}

}

\_adc\_counter++;

if(\_adc\_counter==10){

\_adc\_counter=0;

}

if(\_mode == \_manual && \_tutorial\_stage == 2){

\_auto\_manual\_switch\_timer\_miliseconds++;

}

if(\_auto\_manual\_switch\_timer\_miliseconds == 5000){//5seconds

\_auto\_manual\_switch\_timer\_miliseconds=0;

\_mode=\_auto;

}

}

ISR(ADC\_vect){

switch(\_adc\_target){

case \_x\_joystick:

\_adc\_target=\_x\_joystick;

\_x\_read=ADCW;

ADMUX=\_y\_joystick;

break;

case \_y\_joystick:

\_y\_read=1023-ADCW;

ADMUX=\_tempature;

break;

case \_tempature:

\_tempature\_read=ADCW;

ADMUX=\_light;

break;

case \_light:

\_light\_read=ADCW;

ADMUX=\_x\_joystick;

break;

}

\_adc\_target=(\_adc\_target+1)%4;

mindwizard\_(\_x\_read, \_y\_read);

}

DISPLAY\_ENGINE.cpp

Dot Matrix Driver.

/\*

\* DISPLAY\_ENGINE.cpp

\*

\* Created: 7/26/2023 1:43:57 AM

\* Author: 6gioc

\*/

#include "TOOLBOX.h"

//MAX7219 PIN

#define CS PB0

#define CLK PB1

#define DIN PB2

//MAX7219 CONTROL

#define MAX7219\_NOP 0x00

#define MAX7219\_DIGIT0 0x01

#define MAX7219\_DIGIT1 0x02

#define MAX7219\_DIGIT2 0x03

#define MAX7219\_DIGIT3 0x04

#define MAX7219\_DIGIT4 0x05

#define MAX7219\_DIGIT5 0x06

#define MAX7219\_DIGIT6 0x07

#define MAX7219\_DIGIT7 0x08

#define MAX7219\_DECODE 0x09

#define MAX7219\_INTEN 0x0A

#define MAX7219\_SCANLIM 0x0B

#define MAX7219\_SHUTDOWN 0x0C

#define MAX7219\_TEST 0x0F

//TEXTDISPLAY SPACING

#define CHARSPACE 6

enum DOTMATRIXUPDATESEQUENCE{

\_dimming,

\_line0,

\_line1,

\_line2,

\_line3,

\_line4,

\_line5,

\_line6,

\_line7,

};

void spitrans\_(unsigned char \_data);

void display\_commander\_(unsigned char \_adress, unsigned char \_value);

void cwrotate\_(unsigned char\* \_\_origin);

void ccwrotate\_(unsigned char\* \_\_origin);

unsigned char mirror\_(unsigned char \_origin);

void contentupdate\_(unsigned char \_mind[], unsigned char \_line);

unsigned char getlineofchar\_(unsigned char \_char, unsigned char \_linenumber, unsigned char \_bold);

unsigned long int unsignedlongintbitshiftleftadjustcreatorviaunsignedchar\_(unsigned char \_value, unsigned char \_rightbitshiftvalue);

unsigned char \_light\_alphabet[52][8]={//0-25 uppercase / 26-51 lowercase

{0x00, 0xfc, 0x12, 0x12, 0x12, 0xfc, 0x00, 0x00},//A

{0x00, 0xfe, 0x92, 0x92, 0x92, 0x7c, 0x00, 0x00},//B

{0x00, 0x7c, 0x82, 0x82, 0x82, 0x82, 0x00, 0x00},//C

{0x00, 0xfe, 0x82, 0x82, 0x82, 0x7c, 0x00, 0x00},//D

{0x00, 0xfe, 0x92, 0x92, 0x92, 0x92, 0x00, 0x00},//E

{0x00, 0xfe, 0x12, 0x12, 0x12, 0x12, 0x00, 0x00},//F

{0x00, 0x7c, 0x82, 0x82, 0x92, 0x72, 0x00, 0x00},//G

{0x00, 0xfe, 0x10, 0x10, 0x10, 0xfe, 0x00, 0x00},//H

{0x00, 0x00, 0x00, 0xfe, 0x00, 0x00, 0x00, 0x00},//I

{0x00, 0x40, 0x80, 0x82, 0x82, 0x7e, 0x00, 0x00},//J

{0x00, 0xfe, 0x10, 0x28, 0x44, 0x82, 0x00, 0x00},//K

{0x00, 0xfe, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00},//L

{0x00, 0xfe, 0x08, 0x10, 0x08, 0xfe, 0x00, 0x00},//M

{0x00, 0xfe, 0x08, 0x10, 0x20, 0xfe, 0x00, 0x00},//N

{0x00, 0x7c, 0x82, 0x82, 0x82, 0x7c, 0x00, 0x00},//O

{0x00, 0xfe, 0x12, 0x12, 0x12, 0x0c, 0x00, 0x00},//P

{0x00, 0x7c, 0x82, 0x82, 0x42, 0xbc, 0x00, 0x00},//Q

{0x00, 0xfe, 0x12, 0x12, 0x12, 0xec, 0x00, 0x00},//R

{0x00, 0x8c, 0x92, 0x92, 0x92, 0x62, 0x00, 0x00},//S

{0x00, 0x02, 0x02, 0xfe, 0x02, 0x02, 0x00, 0x00},//T

{0x00, 0x7e, 0x80, 0x80, 0x80, 0x7e, 0x00, 0x00},//U

{0x00, 0x0e, 0x30, 0xc0, 0x30, 0x0e, 0x00, 0x00},//V

{0x00, 0xfe, 0x40, 0x38, 0x40, 0xfe, 0x00, 0x00},//W

{0x00, 0xc6, 0x28, 0x10, 0x28, 0xc6, 0x00, 0x00},//X

{0x00, 0x06, 0x08, 0xf0, 0x08, 0x06, 0x00, 0x00},//Y

{0x00, 0xc2, 0xa2, 0x92, 0x8a, 0x86, 0x00, 0x00},//Z

{0x00, 0x40, 0xa8, 0xa8, 0xa8, 0xf0, 0x00, 0x00},//a

{0x00, 0xfe, 0x88, 0x88, 0x88, 0x70, 0x00, 0x00},//b

{0x00, 0x70, 0x88, 0x88, 0x88, 0x88, 0x00, 0x00},//c

{0x00, 0x70, 0x88, 0x88, 0x88, 0xfe, 0x00, 0x00},//d

{0x00, 0x70, 0xa8, 0xa8, 0xa8, 0x30, 0x00, 0x00},//e

{0x00, 0x88, 0x88, 0x7c, 0x0a, 0x0a, 0x00, 0x00},//f

{0x00, 0x18, 0xa4, 0xa4, 0xa4, 0x78, 0x00, 0x00},//g

{0x00, 0xfe, 0x08, 0x08, 0x08, 0xf0, 0x00, 0x00},//h

{0x00, 0x00, 0x00, 0xfa, 0x00, 0x00, 0x00, 0x00},//i

{0x00, 0x00, 0x80, 0x80, 0x7a, 0x00, 0x00, 0x00},//j

{0x00, 0xfe, 0x20, 0x20, 0x50, 0x88, 0x00, 0x00},//k

{0x00, 0x00, 0x00, 0x7e, 0x80, 0x00, 0x00, 0x00},//l

{0x00, 0xf0, 0x08, 0xf0, 0x08, 0xf0, 0x00, 0x00},//m

{0x00, 0xf0, 0x08, 0x08, 0x08, 0xf0, 0x00, 0x00},//n

{0x00, 0x70, 0x88, 0x88, 0x88, 0x70, 0x00, 0x00},//o

{0x00, 0xfc, 0x24, 0x24, 0x24, 0x18, 0x00, 0x00},//p

{0x00, 0x18, 0x24, 0x24, 0x24, 0xfc, 0x00, 0x00},//q

{0x00, 0xf0, 0x08, 0x08, 0x08, 0x08, 0x00, 0x00},//r

{0x00, 0x90, 0xa8, 0xa8, 0xa8, 0x48, 0x00, 0x00},//s

{0x00, 0x00, 0x08, 0x7e, 0x88, 0x80, 0x00, 0x00},//t

{0x00, 0x78, 0x80, 0x80, 0x80, 0x78, 0x00, 0x00},//u

{0x00, 0x18, 0x60, 0x80, 0x60, 0x18, 0x00, 0x00},//v

{0x00, 0x78, 0x80, 0x78, 0x80, 0x78, 0x00, 0x00},//w

{0x00, 0x88, 0x50, 0x20, 0x50, 0x88, 0x00, 0x00},//x

{0x00, 0x1c, 0xa0, 0xa0, 0xa0, 0x7c, 0x00, 0x00},//y

{0x00, 0x88, 0xc8, 0xa8, 0x98, 0x88, 0x00, 0x00},//z

};

unsigned char \_bold\_alphabet[52][8]={//0-25 uppercase / 26-51 lowercase

{0x00, 0xfc, 0xfe, 0x12, 0xfe, 0xfc, 0x00, 0x00},//A

{0x00, 0xfe, 0xfe, 0x92, 0xfe, 0x6c, 0x00, 0x00},//B

{0x00, 0x7c, 0xfe, 0x82, 0xc6, 0x44, 0x00, 0x00},//C

{0x00, 0xfe, 0xfe, 0x82, 0xfe, 0x7c, 0x00, 0x00},//D

{0x00, 0xfe, 0xfe, 0x92, 0x92, 0x82, 0x00, 0x00},//E

{0x00, 0xfe, 0xfe, 0x12, 0x12, 0x02, 0x00, 0x00},//F

{0x00, 0x7c, 0xfe, 0x82, 0xf6, 0xf4, 0x00, 0x00},//G

{0x00, 0xfe, 0xfe, 0x10, 0xfe, 0xfe, 0x00, 0x00},//H

{0x00, 0x82, 0xfe, 0xfe, 0x82, 0x00, 0x00, 0x00},//I

{0x00, 0x40, 0xc0, 0x82, 0xfe, 0x7e, 0x02, 0x00},//J

{0x00, 0xfe, 0xfe, 0x38, 0xee, 0xc6, 0x00, 0x00},//K

{0x00, 0xfe, 0xfe, 0x80, 0x80, 0x80, 0x00, 0x00},//L

{0x00, 0xfe, 0xfe, 0x0c, 0x0c, 0xfe, 0xfe, 0x00},//M

{0x00, 0xfe, 0xfe, 0x18, 0x60, 0xfe, 0xfe, 0x00},//N

{0x00, 0x7c, 0xfe, 0x82, 0xfe, 0x7c, 0x00, 0x00},//O

{0x00, 0xfe, 0xfe, 0x12, 0x1e, 0x0c, 0x00, 0x00},//P

{0x00, 0x3e, 0x7f, 0x41, 0xff, 0xbe, 0x00, 0x00},//Q

{0x00, 0xfe, 0xfe, 0x12, 0xfe, 0xec, 0x00, 0x00},//R

{0x00, 0x8c, 0x9e, 0x92, 0xf2, 0x62, 0x00, 0x00},//S

{0x00, 0x02, 0x02, 0xfe, 0xfe, 0x02, 0x02, 0x00},//T

{0x00, 0x7e, 0xfe, 0x80, 0xfe, 0x7e, 0x00, 0x00},//U

{0x00, 0x1e, 0x7e, 0xe0, 0xe0, 0x7e, 0x1e, 0x00},//V

{0x00, 0xfe, 0xfe, 0x60, 0x60, 0xfe, 0xfe, 0x00},//W

{0x00, 0xc6, 0xee, 0x38, 0x38, 0xee, 0xc6, 0x00},//X

{0x00, 0xc6, 0xee, 0x38, 0x18, 0x0e, 0x06, 0x00},//Y

{0x00, 0xc2, 0xe2, 0xb2, 0x9e, 0x8e, 0x00, 0x00},//Z

{0x00, 0x40, 0xe8, 0xa8, 0xf8, 0xf0, 0x00, 0x00},//a

{0x00, 0xfe, 0xfe, 0x88, 0xf8, 0x70, 0x00, 0x00},//b

{0x00, 0x70, 0xf8, 0x88, 0xd8, 0x50, 0x00, 0x00},//c

{0x00, 0x70, 0xf8, 0x88, 0xfe, 0xfe, 0x00, 0x00},//d

{0x00, 0x70, 0xf8, 0xa8, 0xb8, 0x30, 0x00, 0x00},//e

{0x00, 0x10, 0xfc, 0xfe, 0x12, 0x00, 0x00, 0x00},//f

{0x00, 0x9c, 0xbe, 0xa2, 0xfe, 0x7e, 0x00, 0x00},//g

{0x00, 0xfe, 0xfe, 0x08, 0xf8, 0xf8, 0x00, 0x00},//h

{0x00, 0x88, 0xfa, 0xfa, 0x80, 0x00, 0x00, 0x00},//i

{0x00, 0x40, 0xc0, 0x88, 0xfa, 0x7a, 0x00, 0x00},//j

{0x00, 0xfe, 0xfe, 0x70, 0xd8, 0x88, 0x00, 0x00},//k

{0x00, 0x82, 0xfe, 0xfe, 0x80, 0x00, 0x00, 0x00},//l

{0xf8, 0xf8, 0x08, 0xf0, 0x08, 0xf8, 0xf0, 0x00},//m

{0x00, 0xf8, 0xf8, 0x08, 0xf8, 0xf0, 0x00, 0x00},//n

{0x00, 0x70, 0xf8, 0x88, 0xf8, 0x70, 0x00, 0x00},//o

{0x00, 0xfe, 0xfe, 0x22, 0x3e, 0x1c, 0x00, 0x00},//p

{0x00, 0x1c, 0x3e, 0x22, 0xfe, 0xfe, 0x00, 0x00},//q

{0x00, 0xf8, 0xf8, 0x18, 0x08, 0x08, 0x00, 0x00},//r

{0x00, 0x90, 0xb8, 0xa8, 0xe8, 0x48, 0x00, 0x00},//s

{0x00, 0x10, 0x7e, 0xfe, 0x90, 0x00, 0x00, 0x00},//t

{0x00, 0x78, 0xf8, 0x80, 0xf8, 0xf8, 0x00, 0x00},//u

{0x00, 0x38, 0x78, 0xc0, 0xc0, 0x78, 0x38, 0x00},//v

{0x00, 0xf8, 0xf8, 0x60, 0x60, 0xf8, 0xf8, 0x00},//w

{0x00, 0x88, 0xd8, 0x70, 0x70, 0xd8, 0x88, 0x00},//x

{0x00, 0x9e, 0xbe, 0xa0, 0xfe, 0x7e, 0x00, 0x00},//y

{0x00, 0xc8, 0xe8, 0xb8, 0x98, 0x00, 0x00, 0x00},//z

};

unsigned char \_light\_number[10][8]={

{0x00, 0x7c, 0xa2, 0x92, 0x8a, 0x7c, 0x00, 0x00},//0

{0x00, 0x00, 0x04, 0xfe, 0x00, 0x00, 0x00, 0x00},//1

{0x00, 0x82, 0xc2, 0xa2, 0x92, 0x8c, 0x00, 0x00},//2

{0x00, 0x92, 0x92, 0x92, 0x92, 0x6c, 0x00, 0x00},//3

{0x00, 0x1e, 0x10, 0x10, 0x10, 0xfe, 0x00, 0x00},//4

{0x00, 0x9e, 0x92, 0x92, 0x92, 0x62, 0x00, 0x00},//5

{0x00, 0x7c, 0x92, 0x92, 0x92, 0x60, 0x00, 0x00},//6

{0x00, 0x02, 0x02, 0xc2, 0x32, 0x0e, 0x00, 0x00},//7

{0x00, 0x6c, 0x92, 0x92, 0x92, 0x6c, 0x00, 0x00},//8

{0x00, 0x0c, 0x92, 0x92, 0x92, 0x7c, 0x00, 0x00},//9

};

unsigned char \_bold\_number[10][8]={

{0x00, 0x7c, 0x7c, 0x82, 0x7c, 0x7c, 0x00, 0x00},//0

{0x00, 0x84, 0xfe, 0xfe, 0x80, 0x00, 0x00, 0x00},//1

{0x00, 0xc4, 0xe6, 0xb2, 0x9e, 0x8c, 0x00, 0x00},//2

{0x00, 0x44, 0xc6, 0x92, 0xfe, 0x6c, 0x00, 0x00},//3

{0x00, 0x18, 0x1c, 0x16, 0xfe, 0xfe, 0x00, 0x00},//4

{0x00, 0x4e, 0xce, 0x8a, 0xfa, 0x72, 0x00, 0x00},//5

{0x00, 0x7c, 0xfe, 0x8a, 0xfa, 0x70, 0x00, 0x00},//6

{0x00, 0xc2, 0xe2, 0x32, 0x1e, 0x0e, 0x00, 0x00},//7

{0x00, 0x6c, 0xfe, 0x92, 0xfe, 0x6c, 0x00, 0x00},//8

{0x00, 0x6c, 0xfe, 0x92, 0xfe, 0x6c, 0x00, 0x00},//9

};

unsigned long long int \_dotmatrix\_screen[8];

unsigned char \_dotmatrix\_update\_phase=0;

unsigned char \_dotmatrix\_brightness=0;

unsigned char \_animation\_shift=0;

void display\_setup\_(void){

for(unsigned char \_rotate=0;\_rotate<52;\_rotate++){

cwrotate\_(\_light\_alphabet[\_rotate]);

cwrotate\_(\_bold\_alphabet[\_rotate]);

for(unsigned char \_mirror=0;\_mirror<8;\_mirror++){

\_light\_alphabet[\_rotate][\_mirror]=mirror\_(\_light\_alphabet[\_rotate][\_mirror]);

\_bold\_alphabet[\_rotate][\_mirror]=mirror\_(\_bold\_alphabet[\_rotate][\_mirror]);

}

}

for(unsigned char \_rotate=0;\_rotate<10;\_rotate++){

cwrotate\_(\_light\_number[\_rotate]);

cwrotate\_(\_bold\_number[\_rotate]);

for(unsigned char \_mirror=0;\_mirror<8;\_mirror++){

\_light\_number[\_rotate][\_mirror]=mirror\_(\_light\_number[\_rotate][\_mirror]);

\_bold\_number[\_rotate][\_mirror]=mirror\_(\_bold\_number[\_rotate][\_mirror]);

}

}

DDRB=DDRB|(1<<CS)|(1<<DIN)|(1<<CLK);

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

unsigned char \_max7219\_startup\_sequence[6][2]={

{MAX7219\_NOP, 0x00},

{MAX7219\_TEST, 0x00},

{MAX7219\_SCANLIM, 0x07},

{MAX7219\_DECODE, 0x00},

{MAX7219\_INTEN, 0x00},

{MAX7219\_SHUTDOWN, 0x01},

};

for(unsigned char \_phase=0;\_phase<6;\_phase++){

portbcontroler\_(CS,FALSE);

for(unsigned char \_module=0;\_module<4;\_module++){

display\_commander\_(\_max7219\_startup\_sequence[\_phase][0], \_max7219\_startup\_sequence[\_phase][1]);

}

portbcontroler\_(CS,TRUE);

}

for(unsigned char \_eraser=0;\_eraser<8;\_eraser++){

\_dotmatrix\_screen[\_eraser]=0;

}

}

void display\_update\_(void){

contentupdate\_(\_mind, \_dotmatrix\_update\_phase-1);

portbcontroler\_(CS,FALSE);

for(unsigned char \_digit=0;\_digit<4;\_digit++){

if(\_dotmatrix\_update\_phase==\_dimming){

display\_commander\_(MAX7219\_INTEN, \_dotmatrix\_brightness);

}

else{

display\_commander\_(\_dotmatrix\_update\_phase, mirror\_((unsigned char)(\_dotmatrix\_screen[\_dotmatrix\_update\_phase-1]>>(8\*\_digit))));

}

}

portbcontroler\_(CS,TRUE);

\_dotmatrix\_update\_phase=(\_dotmatrix\_update\_phase+1)%9;

}

void spitrans\_(unsigned char \_data){

SPDR=\_data;

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

}

void display\_commander\_(unsigned char \_adress, unsigned char \_value){

spitrans\_(\_adress);

spitrans\_(\_value);

}

void cwrotate\_(unsigned char\* \_\_origin){

unsigned char \_drawboard[8]={0,};

for(unsigned char \_y=0;\_y<8;\_y++){

for(unsigned char \_x=0;\_x<8;\_x++){

\_drawboard[\_y]=\_drawboard[\_y]|(getbitofbyte\_(\_\_origin[\_x],(7-\_y))<<\_x);

}

}

for(unsigned char \_duplicate=0;\_duplicate<8;\_duplicate++){

\_\_origin[\_duplicate]=\_drawboard[\_duplicate];

}

}

void ccwrotate\_(unsigned char\* \_\_origin){

unsigned char \_drawboard[8]={0,};

for(unsigned char \_y=0;\_y<8;\_y++){

for(unsigned char \_x=0;\_x<8;\_x++){

\_drawboard[\_y]=\_drawboard[\_y]|(getbitofbyte\_(\_\_origin[7-\_x], \_y)<<\_x);

}

}

for(unsigned char \_duplicate=0;\_duplicate<8;\_duplicate++){

\_\_origin[\_duplicate]=\_drawboard[\_duplicate];

}

}

unsigned char mirror\_(unsigned char \_origin){

unsigned char \_clipboard=\_origin;

\_origin=0;

for(unsigned char \_bit=0;\_bit<8;\_bit++){

\_origin=\_origin|(((\_clipboard&(1<<\_bit))>>\_bit)<<(7-\_bit));

}

return \_origin;

}

void contentupdate\_(unsigned char \_mind[], unsigned char \_line){

\_dotmatrix\_screen[\_line]=0;

if(\_tutorial\_stage == 0){

for(unsigned char \_wordx=0;\_wordx<4;\_wordx++){

\_dotmatrix\_screen[\_line]=\_dotmatrix\_screen[\_line] | unsignedlongintbitshiftleftadjustcreatorviaunsignedchar\_(getlineofchar\_(\_mode\_list[3][\_wordx],\_line, TRUE), (CHARSPACE\*\_wordx));

}

\_dotmatrix\_animation\_move=0;

}

else if(\_tutorial\_stage == 1){

for(unsigned char \_wordx=0;\_wordx<4;\_wordx++){

\_dotmatrix\_screen[\_line]=\_dotmatrix\_screen[\_line] | unsignedlongintbitshiftleftadjustcreatorviaunsignedchar\_(getlineofchar\_(\_mode\_list[4][\_wordx],\_line, TRUE), (CHARSPACE\*\_wordx));

}

\_dotmatrix\_animation\_move=0;

}

else if(*strlen*((char\*)\_mode\_list[\_mode])<=4){

for(unsigned char \_wordx=0;\_wordx<4;\_wordx++){

\_dotmatrix\_screen[\_line]=\_dotmatrix\_screen[\_line] | unsignedlongintbitshiftleftadjustcreatorviaunsignedchar\_(getlineofchar\_(\_mode\_list[\_mode][\_wordx],\_line, TRUE), (CHARSPACE\*\_wordx));

}

\_dotmatrix\_animation\_move=0;

}

else{

for(unsigned char \_wordx=0;\_wordx<*strlen*((char\*)\_mode\_list[\_mode]);\_wordx++){

\_dotmatrix\_screen[\_line]=\_dotmatrix\_screen[\_line] | unsignedlongintbitshiftleftadjustcreatorviaunsignedchar\_(getlineofchar\_(\_mode\_list[\_mode][\_wordx],\_line, TRUE), (CHARSPACE\*\_wordx)-(\_dotmatrix\_animation\_move));

}

}

\_dotmatrix\_screen[\_line]=(\_dotmatrix\_screen[\_line]&0xffffff00)|\_mind[\_line];

/\*

for(unsigned char \_wordx=0;\_wordx<3;\_wordx++){

for(unsigned char \_wordy=0;\_wordy<8;\_wordy++){

\_dotmatrix\_screen[\_wordy]=\_dotmatrix\_screen[\_wordy]|(((unsigned long long int)\_testalpha[\_wordx][\_wordy])<<(8\*(3-\_wordx)));

}

}

\*/

}

unsigned char getlineofchar\_(unsigned char \_char, unsigned char \_linenumber, unsigned char \_bold){

if(\_char>='A' && \_char<='Z'){//upper case

\_char = \_char - 'A';

if(\_bold){

return \_bold\_alphabet[\_char][\_linenumber];

}

else{

return \_light\_alphabet[\_char][\_linenumber];

}

}

else if(\_char>='a' && \_char<='z'){//lower case

\_char = \_char - 71;

if(\_bold){

return \_bold\_alphabet[\_char][\_linenumber];

}

else{

return \_light\_alphabet[\_char][\_linenumber];

}

}

else if(\_char>='0' && \_char<='9'){//number

\_char = \_char - '0';

if(\_bold){

return \_bold\_number[\_char][\_linenumber];

}

else{

return \_light\_number[\_char][\_linenumber];

}

}

else{

return 0;

}

}

unsigned long int unsignedlongintbitshiftleftadjustcreatorviaunsignedchar\_(unsigned char \_value, unsigned char \_rightbitshiftvalue){

if(\_value==0){

return 0;

}

unsigned long int \_simpleunsignedlongint = \_value;

\_simpleunsignedlongint=\_simpleunsignedlongint<<(24-\_rightbitshiftvalue);

return \_simpleunsignedlongint;

}

MOTOR\_ENGINE.cpp

Motor Driver.

/\*

\* MOTOR\_ENGINE.cpp

\*

\* Created: 8/3/2023 4:29:37 PM

\* Author: 6gioc

\*/

#include "TOOLBOX.h"

enum \_stepmotor\_phase\_choose{

\_1phase,

\_2phase,

\_12phase,

\_phase\_count,

};

enum \_stepmotor\_direction\_choose{

\_retrograde=-1,

\_hold=0,

\_prograde=1,

};

enum \_stepmotor\_xy\_define{

\_xmotor=0,

\_ymotor=1,

\_motor\_count=2,

};

const unsigned char \_stepmotor\_sequence[\_phase\_count][8]={

{0x11, 0x22, 0x44, 0x88,0x11,0x22,0x44,0x88},//\_1phase

{0x33, 0x66, 0xcc, 0x99,0x33,0x66,0xcc,0x99},//\_2phase

{0x11, 0x33, 0x22, 0x66, 0x44, 0xcc, 0x88, 0x99},//\_12phase

};

signed char \_stepmotor\_direction[\_motor\_count];

unsigned int \_stepmotor\_rpm[\_motor\_count];

const unsigned char \_stepmotor\_control\_invert[\_motor\_count]={FALSE, FALSE};

signed char \_stepmotor\_current\_step[\_motor\_count]={0,};

unsigned char \_stepmotor\_current\_phase\_choose[\_motor\_count];

unsigned char \_portc\_output;

void stepmotor\_setup\_(void){

DDRA=0xff;

\_stepmotor\_direction[\_xmotor]=\_hold;

\_stepmotor\_direction[\_ymotor]=\_hold;

\_stepmotor\_current\_phase\_choose[\_xmotor]=\_2phase;

\_stepmotor\_current\_phase\_choose[\_ymotor]=\_2phase;

\_stepmotor\_rpm[\_xmotor]=0;

\_stepmotor\_rpm[\_ymotor]=0;

}

void stepmotor\_update\_(signed int \_xval, signed int \_yval){

if(\_xval==0){

\_stepmotor\_direction[\_xmotor]=\_hold;

\_stepmotor\_rpm[\_xmotor]=\_hold;

}

else if(\_xval<0){

if(\_stepmotor\_control\_invert[\_xmotor]==FALSE){

\_stepmotor\_direction[\_xmotor]=\_retrograde;

}

else{

\_stepmotor\_direction[\_xmotor]=\_prograde;

}

\_stepmotor\_rpm[\_xmotor]=abs\_(\_xval);

}

else{

if(\_stepmotor\_control\_invert[\_xmotor]==FALSE){

\_stepmotor\_direction[\_xmotor]=\_prograde;

}

else{

\_stepmotor\_direction[\_xmotor]=\_retrograde;

}

\_stepmotor\_rpm[\_xmotor]=abs\_(\_xval);

}

if(\_yval==0){

\_stepmotor\_direction[\_ymotor]=\_hold;

\_stepmotor\_rpm[\_ymotor]=\_hold;

}

else if(\_yval<0){

if(\_stepmotor\_control\_invert[\_ymotor]==FALSE){

\_stepmotor\_direction[\_ymotor]=\_retrograde;

}

else{

\_stepmotor\_direction[\_ymotor]=\_prograde;

}

\_stepmotor\_rpm[\_ymotor]=abs\_(\_yval);

}

else{

if(\_stepmotor\_control\_invert[\_ymotor]==FALSE){

\_stepmotor\_direction[\_ymotor]=\_prograde;

}

else{

\_stepmotor\_direction[\_ymotor]=\_retrograde;

}

\_stepmotor\_rpm[\_ymotor]=abs\_(\_yval);

}

if(\_stepmotor\_direction[\_xmotor]==\_hold){

TCCR1B=TCCR1B&(~((1<<CS10)|(1<<CS12)));

}else{

TCCR1B=TCCR1B|((1<<CS10)|(1<<CS12));

}

if(\_stepmotor\_direction[\_ymotor]==\_hold){

TCCR3B=TCCR3B&(~((1<<CS30)|(1<<CS32)));

}else{

TCCR3B=TCCR3B|((1<<CS30)|(1<<CS32));

}

OCR1A=(1100-\_stepmotor\_rpm[\_xmotor]\*2);

OCR3AH=((1100-\_stepmotor\_rpm[\_ymotor]\*2)&0xff00)>>8;

OCR3AL=(1100-\_stepmotor\_rpm[\_ymotor]\*2)&0x00ff;

if(TCNT1>OCR1A){

TCNT1 = 0;

}

unsigned int \_tcnt3 = TCNT3H;

\_tcnt3 = (\_tcnt3<<8)|TCNT3L;

if(\_tcnt3>(1100-\_stepmotor\_rpm[\_ymotor]\*2)){

TCNT3H=0;

TCNT3L=0;

}

if(\_xval==\_hold){

portacontroler\_(0,FALSE);

}

else{

portacontroler\_(0,TRUE);

}

if(\_yval==\_hold){

portacontroler\_(1,FALSE);

}

else{

portacontroler\_(1,TRUE);

}

}

ISR(TIMER1\_COMPA\_vect){//x

\_portc\_output=(\_portc\_output&0xf0)|(\_stepmotor\_sequence[\_stepmotor\_current\_phase\_choose[\_xmotor]][\_stepmotor\_current\_step[\_xmotor]]&0x0f);

if(\_stepmotor\_current\_phase\_choose[\_xmotor]==\_1phase||\_stepmotor\_current\_phase\_choose[\_xmotor]==\_2phase){

if(\_stepmotor\_direction[\_xmotor]==\_prograde){

\_stepmotor\_current\_step[\_xmotor]=(\_stepmotor\_current\_step[\_xmotor]+1)%4;

}

else if(\_stepmotor\_direction[\_xmotor]==\_retrograde){

if(\_stepmotor\_current\_step[\_xmotor]==0){

\_stepmotor\_current\_step[\_xmotor]=3;

}

else{

\_stepmotor\_current\_step[\_xmotor]--;

}

}

}

else if(\_stepmotor\_current\_phase\_choose[\_xmotor]==\_12phase){

if(\_stepmotor\_direction[\_xmotor]==\_prograde){

\_stepmotor\_current\_step[\_xmotor]=(\_stepmotor\_current\_step[\_xmotor]+1)%8;

}

else if(\_stepmotor\_direction[\_xmotor]==\_retrograde){

if(\_stepmotor\_current\_step[\_xmotor]==0){

\_stepmotor\_current\_step[\_xmotor]=7;

}

else{

\_stepmotor\_current\_step[\_xmotor]--;

}

}

}

PORTC=\_portc\_output;

}

ISR(TIMER3\_COMPA\_vect){//y

\_portc\_output=(\_portc\_output&0x0f)|(\_stepmotor\_sequence[\_stepmotor\_current\_phase\_choose[\_ymotor]][\_stepmotor\_current\_step[\_ymotor]]&0xf0);

if(\_stepmotor\_current\_phase\_choose[\_ymotor]==\_1phase||\_stepmotor\_current\_phase\_choose[\_ymotor]==\_2phase){

if(\_stepmotor\_direction[\_ymotor]==\_prograde){

\_stepmotor\_current\_step[\_ymotor]=(\_stepmotor\_current\_step[\_ymotor]+1)%4;

}

else if(\_stepmotor\_direction[\_ymotor]==\_retrograde){

if(\_stepmotor\_current\_step[\_ymotor]==0){

\_stepmotor\_current\_step[\_ymotor]=3;

}

else{

\_stepmotor\_current\_step[\_ymotor]--;

}

}

}

else if(\_stepmotor\_current\_phase\_choose[\_ymotor]==\_12phase){

if(\_stepmotor\_direction[\_ymotor]==\_prograde){

\_stepmotor\_current\_step[\_ymotor]=(\_stepmotor\_current\_step[\_ymotor]+1)%8;

}

else if(\_stepmotor\_direction[\_ymotor]==\_retrograde){

if(\_stepmotor\_current\_step[\_ymotor]==0){

\_stepmotor\_current\_step[\_ymotor]=7;

}

else{

\_stepmotor\_current\_step[\_ymotor]--;

}

}

}

PORTC=\_portc\_output;

}

R\_A\_COMM\_ENGINE.cpp

Raspberry Pi and Atmega Communicating Engine.

/\*

\* R\_A\_COMM\_ENGINE.cpp

\*

\* Created: 12/5/2023 9:07:12 AM

\* Author: 6gioc

\*/

#include "TOOLBOX.h"

void r\_a\_comm\_setup\_(void){

DDRD=0x00;

}

void r\_a\_comm\_update\_(){

unsigned char \_pine = PINE;

const unsigned char \_manual\_gun\_shot\_button = 1;

unsigned char \_manual\_gun\_shot\_button\_value = \_pine & (1<<(\_manual\_gun\_shot\_button + 3));

\_manual\_gun\_shot\_button\_value = \_manual\_gun\_shot\_button\_value !=0;

if(\_manual\_gun\_shot\_button\_value){

if(\_conami\_code\_progress == 8){

\_conami\_code\_progress = 0;

\_tutorial\_stage = 0;

return;

}

\_mode = \_manual;

\_auto\_manual\_switch\_timer\_miliseconds=0;

}

unsigned char \_automatic\_gun\_shot\_trigger\_value = \_pine & (1<<6);

if(\_mode == \_auto){

\_automatic\_gun\_shot\_trigger\_value = \_automatic\_gun\_shot\_trigger\_value !=0;

}

else{

\_automatic\_gun\_shot\_trigger\_value=0;

}

portgcontroler\_(0,\_manual\_gun\_shot\_button\_value || \_automatic\_gun\_shot\_trigger\_value);

if(\_mode == \_auto){

unsigned char \_pind = PIND;

unsigned int \_abs\_ymotor\_speed = 0;

unsigned int \_abs\_xmotor\_speed = 0;

if(\_pind&(1<<3)){//y motor speed control

\_abs\_ymotor\_speed = 100;

}

else{

\_abs\_ymotor\_speed = 500;

}

if(\_pind & (1<<0)){//x motor speed control

\_abs\_xmotor\_speed = 100;

}

else{

\_abs\_xmotor\_speed = 500;

}

signed int \_xmotor\_auto\_speed=0;

signed int \_ymotor\_auto\_speed=0;

const signed char \_xmotor\_auto\_speed\_invert=1;

const signed char \_ymotor\_auto\_speed\_invert=1;

if(\_pind&(1<<4)){//x+r\_a

\_xmotor\_auto\_speed = \_abs\_xmotor\_speed \* \_xmotor\_auto\_speed\_invert;

}

if(\_pind&(1<<5)){//x-r\_a

\_xmotor\_auto\_speed = -1 \* \_abs\_xmotor\_speed \* \_xmotor\_auto\_speed\_invert;

}

if(\_pind&(1<<6)){//y+r\_a

\_ymotor\_auto\_speed = \_abs\_ymotor\_speed \* \_ymotor\_auto\_speed\_invert;

}

if(\_pind&(1<<7)){//y-r\_a

\_ymotor\_auto\_speed = -1 \* \_abs\_ymotor\_speed \* \_ymotor\_auto\_speed\_invert;

}

stepmotor\_update\_(\_xmotor\_auto\_speed,\_ymotor\_auto\_speed);

}

}