/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* AUTHOR : MOSTAFA MUHAMED ABDOU \*/

/\* VERSION : V01 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef ATMEGA32\_REGISTERS

#define ATMEGA32\_REGISTERS

#include "standard\_types.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PORTA Registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define PORTA (\*(volatile u\_int8\*) 0x3B)

#define DDRA (\*(volatile u\_int8\*) 0x3A)

#define PINA (\*(volatile u\_int8\*) 0x39)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PORTB Registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define PORTB (\*(volatile u\_int8\*) 0x38)

#define DDRB (\*(volatile u\_int8\*) 0x37)

#define PINB (\*(volatile u\_int8\*) 0x36)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PORTC Registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define PORTC (\*(volatile u\_int8\*) 0x35)

#define DDRC (\*(volatile u\_int8\*) 0x34)

#define PINC (\*(volatile u\_int8\*) 0x33)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PORTD Registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define PORTD (\*(volatile u\_int8\*) 0x32)

#define DDRD (\*(volatile u\_int8\*) 0x31)

#define PIND (\*(volatile u\_int8\*) 0x30)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Interrupt Registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SREG (\*(volatile u\_int8\*) 0x5F)

#define GICR (\*(volatile u\_int8\*) 0x5B)

#define GIFR (\*(volatile u\_int8\*) 0x5A)

#define MCUCR (\*(volatile u\_int8\*) 0x55)

#define MCUCSR (\*(volatile u\_int8\*) 0x54)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC Registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SFIOR (\*(volatile u\_int8\*) 0x50)

#define ADCSRA (\*(volatile u\_int8\*) 0x26)

#define ADMUX (\*(volatile u\_int8\*) 0x27)

#define ADCH (\*(volatile u\_int8\*) 0x25)

#define ADCL (\*(volatile u\_int8\*) 0x24)

#define ADC\_VAL (\*(volatile u\_int16\*) 0x24)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TIMER0 Registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define TCCR0 (\*(volatile u\_int8\*) 0x53)

#define TCNT0 (\*(volatile u\_int8\*) 0x52)

#define OCR0 (\*(volatile u\_int8\*) 0x5C)

#define TIMSK (\*(volatile u\_int8\*) 0x59)

#define TIFR (\*(volatile u\_int8\*) 0x58)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* UART Registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define UDR (\*(volatile u\_int8\*) 0x2C)

#define UCSRA (\*(volatile u\_int8\*) 0x2B)

#define UCSRB (\*(volatile u\_int8\*) 0x2A)

#define UBRRL (\*(volatile u\_int8\*) 0x29)

#define UBRRH (\*(volatile u\_int8\*) 0x40)

#define UCSRC (\*(volatile u\_int8\*) 0x40)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SPI Registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* NORMAL WAY \*/

#define SPCR (\*(volatile u\_int8\*) 0x2D)

#define SPSR (\*(volatile u\_int8\*) 0x2E)

#define SPDR (\*(volatile u\_int8\*) 0x2F)

/\*\* USING STRUCT \*\*/

/\*

typedef struct

{

u\_int8 SPCR;

u\_int8 SPSR;

u\_int8 SPDR;

}SPI\_REGS;

#define SPI ((volatile SPI\_REGS\*)(0x2D))

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* I2C Registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define TWCR (\*(volatile u\_int8\*) 0X65)

#define TWDR (\*(volatile u\_int8\*) 0x23)

#define TWAR (\*(volatile u\_int8\*) 0x22)

#define TWSR (\*(volatile u\_int8\*) 0x21)

#define TWBR (\*(volatile u\_int8\*) 0X20)

/\*\* WAY TO BUILD A REGISTER MANUALLY \*\*

struct ADCSRA

{

u\_int8 ADPS0 : 1 ;

u\_int8 ADPS1 : 1 ;

u\_int8 ADPS2 : 1 ;

u\_int8 ADIE : 1 ;

u\_int8 ADIF : 1 ;

u\_int8 ADATE : 1 ;

u\_int8 ADCS : 1 ;

u\_int8 ADEN : 1 ;

};

\*/

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* AUTHOR : MOSTAFA MUHAMED ABDOU \*/

/\* VERSION : V01 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef COMMON\_MACROS\_H\_INCLUDED

#define COMMON\_MACROS\_H\_INCLUDED

/\*macro to set a specific bit\*/

#define set\_bit(byte,bit\_num) (byte |= (1<<bit\_num))

/\*macro to clear a specific bit\*/

#define clear\_bit(byte,bit\_num) (byte &= (~(1<<bit\_num)))

/\*macro to toggle a specific bit\*/

#define toggle\_bit(byte,bit\_num) (byte ^= (1<<bit\_num))

/\*macro to check if a bit is set\*/

#define bit\_is\_set(byte,bit\_num) (byte & (1<<bit\_num))

/\*macro to check if bit is cleared\*/

#define bit\_is\_clear(byte,bit\_num) (!(byte & (1<<bit\_num)))

/\*macro to perform right circular\*/

#define cir\_right(byte,bits) (byte = (byte >> bits) | (byte << (8-bits)))

/\*macro to perform left circular\*/

#define cir\_left(byte,bits) (byte = (byte << bits) | (byte >> (8-bits)))

/\*macro to display a specific flag\*/

#define get\_bit(byte,flag\_num) (byte = (byte >> flag\_num) & 0X01)

#endif // COMMON\_MACROS\_H\_INCLUDED

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* AUTHOR : MOSTAFA MUHAMED ABDOU \*/

/\* DATE : 14 DEC 2021 \*/

/\* VERSION : V01 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "SPI.h"

void SPI\_INIT\_MASTER()

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MASTER SIDE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* MOSI -> OUPTPUT\_PIN \*

\* SS -> OUTPUT\_PIN \*

\* MISO -> INPUT\_PIN \*

\* SCK -> OUTPUT\_PIN \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DIO\_SETPINDIR(DIO\_PORTB , DIO\_PIN4 , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(DIO\_PORTB , DIO\_PIN5 , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(DIO\_PORTB , DIO\_PIN6 , DIO\_PIN\_INPUT);

DIO\_SETPINDIR(DIO\_PORTB , DIO\_PIN7 , DIO\_PIN\_OUTPUT);

/\* CURRENT CONFIGURATION \*/

SPCR = 0x53;

/\*

#if SPI\_INTERRUPT\_MODE == ENABLE\_SPI\_INTERRUPT

set\_bit(SPCR , 7);

#elif SPI\_INTERRUPT\_MODE == DISABLE\_SPI\_INTERRUPT

clear\_bit(SPCR , 7);

#endif

\*/

/\* ENABLE SPI MODULE \*/

//set\_bit(SPCR , 6);

/\*

#if SPI\_DATA\_ORDER == SEND\_LSB\_FIRST

set\_bit(SPCR , 5);

#elif SPI\_DATA\_ORDER == SEND\_MSB\_FIRST

clear\_bit(SPCR , 5);

#endif

\*/

/\* MASTER MODE \*/

//set\_bit(SPCR , 4);

/\*

#if SPI\_CLOCK\_POLARITY == IDLE\_HIGH\_POLARITY

set\_bit(SPCR , 3);

#elif SPI\_CLOCK\_POLARITY == IDLE\_LOW\_POLARITY

clear\_bit(SPCR , 3);

#endif

#if SPI\_CLOCK\_PHASE == READ\_WRITE\_PHASE

clear\_bit(SPCR , 2);

#elif SPI\_CLOCK\_PHASE == WRITE\_READ\_PHASE

set\_bit(SPCR , 2);

#endif

#if DOUBLE\_SPEED\_MODE == ENABLE\_DOUBLE\_SPEED\_MODE

set\_bit(SPSR , 0);

#elif DOUBLE\_SPEED\_MODE == DISABLE\_DOUBLE\_SPEED\_MODE

clear\_bit(SPSR , 0);

#endif

#if SPI\_PRESCALER\_VALUE == PRESCALLER\_4

clear\_bit(SPCR , 0);

clear\_bit(SPCR , 1);

clear\_bit(SPSR , 0);

#elif SPI\_PRESCALER\_VALUE == PRESCALLER\_16

set\_bit(SPCR , 0);

clear\_bit(SPCR , 1);

clear\_bit(SPSR , 0);

#elif SPI\_PRESCALER\_VALUE == PRESCALLER\_32

clear\_bit(SPCR , 0);

set\_bit(SPCR , 1);

set\_bit(SPSR , 0);

#elif SPI\_PRESCALER\_VALUE == PRESCALLER\_128

set\_bit(SPCR , 0);

set\_bit(SPCR , 1);

clear\_bit(SPSR , 0);

#elif SPI\_PRESCALER\_VALUE == PRESCALLER\_2

clear\_bit(SPCR , 0);

clear\_bit(SPCR , 1);

set\_bit(SPSR , 0);

#elif SPI\_PRESCALER\_VALUE == PRESCALLER\_8

set\_bit(SPCR , 0);

clear\_bit(SPCR , 1);

set\_bit(SPSR , 0);

#endif

\*/

}

void SPI\_INIT\_SLAVE()

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SLAVE SIDE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* MOSI -> INPUT\_PIN \*

\* SS -> INPUT\_PIN \*

\* MISO -> OUTPUT\_PIN \*

\* SCK -> INPUT\_PIN \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DIO\_SETPINDIR(DIO\_PORTB , DIO\_PIN4 , DIO\_PIN\_INPUT);

DIO\_SETPINDIR(DIO\_PORTB , DIO\_PIN5 , DIO\_PIN\_INPUT);

DIO\_SETPINDIR(DIO\_PORTB , DIO\_PIN6 , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(DIO\_PORTB , DIO\_PIN7 , DIO\_PIN\_INPUT);

/\* CHOOSE SLAVE MODE \*/

clear\_bit(SPCR , 4);

/\* ENABLE SPI MODULE \*/

set\_bit(SPCR , 6);

/\*

\* CLOCK POLARITY AND CLOCK PHASE ARE

\* THE SAME IN MASTER SIDE AND SLAVE SIDE

\*/

#if SPI\_CLOCK\_POLARITY == IDLE\_HIGH\_POLARITY

set\_bit(SPCR , 3);

#elif SPI\_CLOCK\_POLARITY == IDLE\_LOW\_POLARITY

clear\_bit(SPCR , 3);

#endif

#if SPI\_CLOCK\_PHASE == READ\_WRITE\_PHASE

clear\_bit(SPCR , 2);

#elif SPI\_CLOCK\_PHASE == WRITE\_READ\_PHASE

set\_bit(SPCR , 2);

#endif

}

void SPI\_MASTER\_INIT\_TRANSMIT()

{

/\* DRIVE THE SS PIN LOW \*/

DIO\_SETPINVAL(DIO\_PORTB , DIO\_PIN4 , DIO\_PIN\_LOW);

}

void SPI\_MASTER\_END\_TRANSMIT()

{

/\* MAKE THE SS PIN HIGH \*/

DIO\_SETPINVAL(DIO\_PORTB , DIO\_PIN4 , DIO\_PIN\_HIGH);

}

u\_int8 SPI\_TRANSCEIV\_BYTE(u\_int8 DATA)

{

/\*

\* THIS FUNCTION SEND DATA AND RETURN THE

\* RECEIVED DATA FROM SLAVE

\*/

u\_int8 REC\_DATA = 0;

SPDR = DATA ; /\* TRANSMITTED DATA \*/

while(get\_bit(SPSR , 7) == 0); /\* DO NOT MOVE UNTILL DATA RECEIVED \*/

REC\_DATA = SPDR ; /\* RECEIVED DATA \*/

return REC\_DATA;

}

void SPI\_SEND\_BYTE(u\_int8 data)

{

SPDR = data ;

while(get\_bit(SPSR , 7) == 0);

}

u\_int8 SPI\_RECEIVE\_BYTE()

{

u\_int8 REC\_DATA;

while(get\_bit(SPSR , 7) == 0);

REC\_DATA = SPDR ;

return REC\_DATA;

}

void SPI\_SEND\_STRING(const u\_int8\* str)

{

u\_int8 counter = 0;

while (str[counter] != '\0')

{

SPI\_SEND\_BYTE(str[counter]);

counter++; /\* MOVE TO THE SECOND LETTER \*/

}

}

void SPI\_RECEIVE\_STRING(u\_int8\* str)

{

u\_int8 counter = 0 ;

str[counter] = SPI\_RECEIVE\_BYTE();

while (str[counter] != '#')

{

counter++;

str[counter] = SPI\_RECEIVE\_BYTE();

}

str[counter] = '\0' ; /\* TERMINATOR OF STRING \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* AUTHOR : MOSTAFA MUHAMED ABDOU \*/

/\* DATE : 11 FEB 2020 \*/

/\* VERSION : V01 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "uart.h"

void UART\_init(void)

{

/\* directions of TX and RX pins \*/

DIO\_SETPINDIR(DIO\_PORTD , DIO\_PIN1 , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(DIO\_PORTD , DIO\_PIN0 , DIO\_PIN\_INPUT);

/\* U2X = 1 for double transmission speed \*/

UCSRA = (1<<U2X);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* UCSRB Description \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* RXCIE = 0 Disable USART RX Complete Interrupt Enable

\* TXCIE = 0 Disable USART Tx Complete Interrupt Enable

\* UDRIE = 0 Disable USART Data Register Empty Interrupt Enable

\* RXEN = 1 Receiver Enable

\* RXEN = 1 Transmitter Enable

\* UCSZ2 = 0 For 8-bit data mode

\* RXB8 & TXB8 not used for 8-bit data mode

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

UCSRB = (1<<RXEN) | (1<<TXEN);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* UCSRC Description \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* URSEL = 1 The URSEL must be one when writing the UCSRC

\* UMSEL = 0 Asynchronous Operation

\* UPM1:0 = 00 Disable parity bit

\* USBS = 0 One stop bit

\* UCSZ1:0 = 11 For 8-bit data mode

\* UCPOL = 0 Used with the Synchronous operation only

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

UCSRC = (1<<URSEL) | (1<<UCSZ0) | (1<<UCSZ1);

/\* baud rate=9600 & Fosc=1MHz --> UBBR=( Fosc / (8 \* baud rate) ) - 1 = 12 \*/

UBRRH = 0;

UBRRL = 207;

}

void UART\_sendByte(const u\_int8 data)

{

/\* UDRE flag is set when the Tx buffer (UDR) is empty and ready for

\* transmitting a new byte so wait until this flag is set to one \*/

while(bit\_is\_clear(UCSRA,UDRE)){}

/\* Put the required data in the UDR register and it also clear the UDRE flag as

\* the UDR register is not empty now \*/

UDR = data;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Another Method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

UDR = data;

while(BIT\_IS\_CLEAR(UCSRA,TXC)){} // Wait until the transimission is complete TXC = 1

SET\_BIT(UCSRA,TXC); // Clear the TXC flag

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

u\_int8 UART\_recieveByte(void)

{

/\* RXC flag is set when the UART receive data so wait until this

\* flag is set to one \*/

while(bit\_is\_clear(UCSRA,RXC)){}

/\* Read the received data from the Rx buffer (UDR) and the RXC flag

will be cleared after read this data \*/

return UDR;

}

void UART\_sendString(const u\_int8 \*Str)

{

u\_int8 i = 0;

while(Str[i] != '\0')

{

UART\_sendByte(Str[i]);

i++;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Another Method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

while(\*Str != '\0')

{

UART\_sendByte(\*Str);

Str++;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

void UART\_receiveString(u\_int8 \*Str)

{

u\_int8 i = 0;

Str[i] = UART\_recieveByte();

while(Str[i] != '#')

{

i++;

Str[i] = UART\_recieveByte();

}

Str[i] = '\0';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* AUTHOR : MOSTAFA MUHAMED ABDOU \*/

/\* VERSION : V01 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "DIO.h"

#include "common\_macros.h"

void DIO\_SETPINDIR(u\_int8 port , u\_int8 pin , u\_int8 dir)

{

switch(dir)

{

case DIO\_PIN\_INPUT: //in case of input

switch(port)

{

case DIO\_PORTA:

clear\_bit(DDRA , pin);

break;

case DIO\_PORTB:

clear\_bit(DDRB , pin);

break;

case DIO\_PORTC:

clear\_bit(DDRC , pin);

break;

case DIO\_PORTD:

clear\_bit(DDRD , pin);

break;

default:

break;

} //end of switch(port)

break;

case DIO\_PIN\_OUTPUT: //in case of output

switch(port)

{

case DIO\_PORTA:

set\_bit(DDRA , pin);

break;

case DIO\_PORTB:

set\_bit(DDRB , pin);

break;

case DIO\_PORTC:

set\_bit(DDRC , pin);

break;

case DIO\_PORTD:

set\_bit(DDRD , pin);

break;

default:

break;

} //end of switch(port)

break;

} //end of swich(dir)

} //end of function

void DIO\_SETPORTDIR(u\_int8 port , u\_int8 dir)

{

switch(port)

{

case DIO\_PORTA:

DDRA = dir;

break;

case DIO\_PORTB:

DDRB = dir;

break;

case DIO\_PORTC:

DDRC = dir;

break;

case DIO\_PORTD:

DDRD = dir;

break;

default:

break;

} //end of switch(port)

} //end of the function

void DIO\_SETPINVAL(u\_int8 port , u\_int8 pin , u\_int8 val)

{

switch(val)

{

case DIO\_PIN\_LOW:

switch(port)

{

case DIO\_PORTA:

clear\_bit(PORTA , pin);

break;

case DIO\_PORTB:

clear\_bit(PORTB , pin);

break;

case DIO\_PORTC:

clear\_bit(PORTC , pin);

break;

case DIO\_PORTD:

clear\_bit(PORTD , pin);

break;

default:

break;

} //end of switch(port)

break;

case DIO\_PIN\_HIGH:

switch(port)

{

case DIO\_PORTA:

set\_bit(PORTA , pin);

break;

case DIO\_PORTB:

set\_bit(PORTB , pin);

break;

case DIO\_PORTC:

set\_bit(PORTC , pin);

break;

case DIO\_PORTD:

set\_bit(PORTD , pin);

break;

default:

break;

} //end of switch(port)

break;

} //end of switch(val)

} //end of the function

void DIO\_SETPORTVAL(u\_int8 port , u\_int8 val)

{

switch(port)

{

case DIO\_PORTA:

PORTA = val;

break;

case DIO\_PORTB:

PORTB = val;

break;

case DIO\_PORTC:

PORTC = val;

break;

case DIO\_PORTD:

PORTD = val;

break;

default:

break;

} //end of switch(port)

} //end of the function

void DIO\_READPIN(u\_int8 port , u\_int8 pin , u\_int8\* val)

{

switch(port)

{

case DIO\_PORTA:

\*val = get\_bit(PINA , pin);

break;

case DIO\_PORTB:

\*val = get\_bit(PINB , pin);

break;

case DIO\_PORTC:

\*val = get\_bit(PINC , pin);

break;

case DIO\_PORTD:

\*val = get\_bit(PIND , pin);

break;

default:

break;

} //end of switch(port)

}//end of the function

void DIO\_READPORT(u\_int8 port , u\_int8\* val)

{

switch(port)

{

case DIO\_PORTA:

\*val = PINA;

break;

case DIO\_PORTB:

\*val = PINB;

break;

case DIO\_PORTC:

\*val = PINC;

break;

case DIO\_PORTD:

\*val = PIND;

break;

default:

break;

} //end of switch(port)

} //end of the function

void DIO\_PINTOGGLE(u\_int8 port , u\_int8 pin)

{

switch(port)

{

case DIO\_PORTA:

toggle\_bit(PORTA , pin);

break;

case DIO\_PORTB:

toggle\_bit(PORTB , pin);

break;

case DIO\_PORTC:

toggle\_bit(PORTC , pin);

break;

case DIO\_PORTD:

toggle\_bit(PORTD , pin);

break;

default:

break;

} //end of switch(port)

} //end of the function

void DIO\_SETPULLUPS(u\_int8 port , u\_int8 pin)

{

switch (port)

{

case DIO\_PORTA:

set\_bit(PORTA , pin);

break;

case DIO\_PORTB:

set\_bit(PORTB , pin);

break;

case DIO\_PORTC:

set\_bit(PORTC , pin);

break;

case DIO\_PORTD:

set\_bit(PORTD , pin);

break;

default:

break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* AUTHOR : MOSTAFA MUHAMED ABDOU \*/

/\* DATE : 11 FEB 2020 \*/

/\* VERSION : V01 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "LCD.h"

void LCD\_INIT(void)

{

#if LCD\_MODE == 8

DIO\_SETPINDIR(LCD\_8BIT\_CMD\_PORT , LCD\_RS\_PIN , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(LCD\_8BIT\_CMD\_PORT , LCD\_RW\_PIN , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(LCD\_8BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_OUTPUT);

DIO\_SETPORTDIR(LCD\_8BIT\_DATA\_PORT , DIO\_PORT\_OUTPUT);

\_delay\_ms(100);

LCD\_SENDCMD(LCD\_8BIT\_MODE);

LCD\_SENDCMD(LCD\_CURSOR\_OFF);

LCD\_SENDCMD(LCD\_CURSOR\_INC);

LCD\_SENDCMD(LCD\_RETURN\_HOME);

LCD\_SENDCMD(LCD\_CLEAR);

#elif LCD\_MODE == 4

DIO\_SETPINDIR(LCD\_4BIT\_CMD\_PORT , LCD\_RS\_PIN , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(LCD\_4BIT\_CMD\_PORT , LCD\_RW\_PIN , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(LCD\_4BIT\_DATA\_PORT , LCD\_D4\_PIN , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(LCD\_4BIT\_DATA\_PORT , LCD\_D5\_PIN , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(LCD\_4BIT\_DATA\_PORT , LCD\_D6\_PIN , DIO\_PIN\_OUTPUT);

DIO\_SETPINDIR(LCD\_4BIT\_DATA\_PORT , LCD\_D7\_PIN , DIO\_PIN\_OUTPUT);

\_delay\_ms(100);

LCD\_SENDCMD(LCD\_4BIT\_MODE\_CMD1);

LCD\_SENDCMD(LCD\_4BIT\_MODE\_CMD2);

LCD\_SENDCMD(LCD\_4BIT\_MODE\_CMD3);

LCD\_SENDCMD(LCD\_CURSOR\_OFF);

LCD\_SENDCMD(LCD\_CURSOR\_INC);

LCD\_SENDCMD(LCD\_RETURN\_HOME);

LCD\_SENDCMD(LCD\_CLEAR);

#endif

}

void LCD\_SENDCMD(u\_int8 cmd)

{

#if LCD\_MODE == 8

DIO\_SETPINVAL(LCD\_8BIT\_CMD\_PORT , LCD\_RS\_PIN , DIO\_PIN\_LOW); //COMMAND MODE

DIO\_SETPINVAL(LCD\_8BIT\_CMD\_PORT , LCD\_RW\_PIN , DIO\_PIN\_LOW); //WRITE TO LCD MODE

DIO\_SETPINVAL(LCD\_8BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_LOW); //PREPARE FOR LATCH

DIO\_SETPORTVAL(PORTA , cmd); //SEND THE COMMAND

DIO\_SETPINVAL(LCD\_8BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_HIGH); //LATCH

\_delay\_ms(1);

DIO\_SETPINVAL(LCD\_8BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_LOW); //LATCH

\_delay\_ms(5);

#elif LCD\_MODE == 4

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_RS\_PIN , DIO\_PIN\_LOW); //COMMAND MODE

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_RW\_PIN , DIO\_PIN\_LOW); //WRITE TO LCD MODE

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_LOW); //PREPARE FOR LATCH

PORTA = ((cmd & 0xF0) | (PORTA & 0x0F)); //SEND THE MSB(HIGH NIBBLE)

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_HIGH); //LATCH

\_delay\_ms(1);

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_LOW); //LATCH

PORTA = ((cmd << 4) | (PORTA & 0x0F)); //SEND THE LSB(LOW NIBBLE)

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_HIGH); //LATCH

\_delay\_ms(1);

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_LOW); //LATCH

\_delay\_ms(5); //5 ms BEFORE SENDING THE NEXT COMMAND

#endif

}

void LCD\_WRITECHAR(u\_int8 chr)

{

#if LCD\_MODE == 8

DIO\_SETPINVAL(LCD\_8BIT\_CMD\_PORT , LCD\_RS\_PIN , DIO\_PIN\_HIGH); //DATA MODE

DIO\_SETPINVAL(LCD\_8BIT\_CMD\_PORT , LCD\_RW\_PIN , DIO\_PIN\_LOW); //WRITE TO LCD MODE

DIO\_SETPINVAL(LCD\_8BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_LOW); //DATA MODE

PORTA = chr; //SEND THE CHARACTER

DIO\_SETPINVAL(LCD\_8BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_HIGH); //LATCH

\_delay\_ms(1);

DIO\_SETPINVAL(LCD\_8BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_LOW); //LATCH

\_delay\_ms(5); //5 ms BEFORE SENDING THE NEXT CHARACTER

#elif LCD\_MODE == 4

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_RS\_PIN , DIO\_PIN\_HIGH); //DATA MODE

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_RW\_PIN , DIO\_PIN\_LOW); //WRITE TO LCD MODE

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_LOW); //DATA MODE

PORTA = ((chr & 0xF0) | (PORTA & 0x0F)); //SEND THE MSB(HIGH NIBBLE)

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_HIGH); //LATCH

\_delay\_ms(1);

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_LOW); //LATCH

PORTA = ((chr << 4) | (PORTA & 0x0F)); //SEND THE LSB(LOW NIBBLE)

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_HIGH); //LATCH

\_delay\_ms(1);

DIO\_SETPINVAL(LCD\_4BIT\_CMD\_PORT , LCD\_E\_PIN , DIO\_PIN\_LOW); //LATCH

\_delay\_ms(5); //5 ms BEFORE SENDING THE NEXT CHARACTER

#endif

}

void LCD\_WRITESTRING(u\_int8\* str)

{

u\_int8 i = 0 ;

while (str[i] != '\0')

{

LCD\_WRITECHAR(str[i]);

i++;

}

}

void LCD\_GOTO(u\_int8 row , u\_int8 column)

{

u\_int8 positions[4] = {0x80 , 0xC0 , 0x94 , 0xD4}; //POSITIONS OF 1st cell of each row

LCD\_SENDCMD(positions[row] + column);

}

void LCD\_CLear(void)

{

LCD\_SENDCMD(LCD\_CLEAR);

}

void LCD\_WRITEINT(s\_int32 num)

{

u\_int8 i = 0, k ;

u\_int8 number[10];

/\* In case of negative numbers \*/

if(num < 0)

{

num \*= -1;

while(num > 0)

{

number[i] = (num % 10) + 48; //Equivalent ASCII of number

num /= 10;

i++;

}

number[i] = '-'; // storing the negative sign

for(k = i+1 ; k > 0 ; k--)

{

LCD\_WRITECHAR(number[k - 1]); //Display each number

}

}

/\* Zero case \*/

else if(num == 0)

{

LCD\_WRITECHAR('0');

}

/\* In case of positive numbers \*/

else if(num > 0)

{

while(num > 0)

{

number[i] = (num % 10) + 48; //Equivalent ASCII of number

num /= 10;

i++;

}

for(k = i ; k > 0 ; k--)

{

LCD\_WRITECHAR(number[k-1]); //Display each number

}

}

}