Noor blum

Design Assignment DA6

**DO NOT REMOVE THIS PAGE DURING SUBMISSION:**

The student understands that all required components should be submitted in complete for grading of this assignment.

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **SUBMISSION ITEM** | **COMPLETED (Y/N)** | **MARKS**  **(/MAX)** |
| 0. | C CODE WITH COMMENTS FOR TASK | Y |  |
| 1. | SCHEMATIC | Y |  |
| 2. | Flow Chart | Y |  |
| 3. | SNAPSHOT OF THE BOARD WITH CONNECTED COMPONENTS | Y |  |
| 4. | SNAPSHOT OF THE COMPUTER SCREEN WITH TERMINAL | Y |  |
| 5. | YouTube Video Link | Y |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 0. | C CODE WITH COMMENTS FOR TASK |  |  |

#ifndef F\_CPU

#define F\_CPU 1600000UL

#endif

#include <avr/io.h>

#include <util/delay.h>

#include "lcd.h"

void adc\_init()

{

ADMUX = (1<<REFS0);

ADCSRA = (1<<ADEN);

}

// read adc value

uint16\_t adc\_read(uint8\_t ch)

{

ch &= 0b00000111; // AND operation with 7

ADMUX = (ADMUX & 0xF7)|ch;

ADCSRA |= (1<<ADSC);

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

return (ADC);

}

int main()

{

uint16\_t adc\_result0;

int temp;

char buffer[10];

// initialize adc and lcd

adc\_init();

lcd\_init(LCD\_DISP\_ON\_CURSOR);

lcd\_clrscr();

lcd\_gotoxy(0,0);

\_delay\_ms(50);

while(1)

{

adc\_result0 = adc\_read(0);// read adc value at PC0

temp=adc\_result0/16.0; // finding the temperature

lcd\_gotoxy(0,0);

itoa(temp,buffer,10);

lcd\_puts("Temperature="); //display temperature

lcd\_puts(buffer);

lcd\_puts(" "); //display temperature

lcd\_puts(" "); //display temperature

lcd\_gotoxy(0,1);

lcd\_puts(" "); //display temperature

lcd\_puts("By Noor Blum ");//display temperature

\_delay\_ms(500);

}

}

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

Title : HD44780U LCD library

Author: Peter Fleury <pfleury@gmx.ch> http://jump.to/fleury

File: $Id: lcd.c,v 1.14.2.2 2012/02/12 07:51:00 peter Exp $

Software: AVR-GCC 3.3

Target: any AVR device, memory mapped mode only for AT90S4414/8515/Mega

DESCRIPTION

Basic routines for interfacing a HD44780U-based text lcd display

Originally based on Volker Oth's lcd library,

changed lcd\_init(), added additional constants for lcd\_command(),

added 4-bit I/O mode, improved and optimized code.

Library can be operated in memory mapped mode (LCD\_IO\_MODE=0) or in

4-bit IO port mode (LCD\_IO\_MODE=1). 8-bit IO port mode not supported.

Memory mapped mode compatible with Kanda STK200, but supports also

generation of R/W signal through A8 address line.

USAGE

See the C include lcd.h file for a description of each function

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

#include <inttypes.h>

#include <avr/io.h>

#include <avr/pgmspace.h>

#include "lcd.h"

/\*

\*\* constants/macros

\*/

#define DDR(x) (\*(&x - 1)) /\* address of data direction register of port x \*/

#if defined(\_\_AVR\_ATmega64\_\_) || defined(\_\_AVR\_ATmega128\_\_)

/\* on ATmega64/128 PINF is on port 0x00 and not 0x60 \*/

#define PIN(x) ( &PORTF==&(x) ? \_SFR\_IO8(0x00) : (\*(&x - 2)) )

#else

#define PIN(x) (\*(&x - 2)) /\* address of input register of port x \*/

#endif

#if LCD\_IO\_MODE

#define lcd\_e\_delay() \_\_asm\_\_ \_\_volatile\_\_( "rjmp 1f\n 1:" ); //#define lcd\_e\_delay() \_\_asm\_\_ \_\_volatile\_\_( "rjmp 1f\n 1: rjmp 2f\n 2:" );

#define lcd\_e\_high() LCD\_E\_PORT |= \_BV(LCD\_E\_PIN);

#define lcd\_e\_low() LCD\_E\_PORT &= ~\_BV(LCD\_E\_PIN);

#define lcd\_e\_toggle() toggle\_e()

#define lcd\_rw\_high() LCD\_RW\_PORT |= \_BV(LCD\_RW\_PIN)

#define lcd\_rw\_low() LCD\_RW\_PORT &= ~\_BV(LCD\_RW\_PIN)

#define lcd\_rs\_high() LCD\_RS\_PORT |= \_BV(LCD\_RS\_PIN)

#define lcd\_rs\_low() LCD\_RS\_PORT &= ~\_BV(LCD\_RS\_PIN)

#endif

#if LCD\_IO\_MODE

#if LCD\_LINES==1

#define LCD\_FUNCTION\_DEFAULT LCD\_FUNCTION\_4BIT\_1LINE

#else

#define LCD\_FUNCTION\_DEFAULT LCD\_FUNCTION\_4BIT\_2LINES

#endif

#else

#if LCD\_LINES==1

#define LCD\_FUNCTION\_DEFAULT LCD\_FUNCTION\_8BIT\_1LINE

#else

#define LCD\_FUNCTION\_DEFAULT LCD\_FUNCTION\_8BIT\_2LINES

#endif

#endif

#if LCD\_CONTROLLER\_KS0073

#if LCD\_LINES==4

#define KS0073\_EXTENDED\_FUNCTION\_REGISTER\_ON 0x2C /\* |0|010|1100 4-bit mode, extension-bit RE = 1 \*/

#define KS0073\_EXTENDED\_FUNCTION\_REGISTER\_OFF 0x28 /\* |0|010|1000 4-bit mode, extension-bit RE = 0 \*/

#define KS0073\_4LINES\_MODE 0x09 /\* |0|000|1001 4 lines mode \*/

#endif

#endif

/\*

\*\* function prototypes

\*/

#if LCD\_IO\_MODE

static void toggle\_e(void);

#endif

/\*

\*\* local functions

\*/

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

delay loop for small accurate delays: 16-bit counter, 4 cycles/loop

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

static inline void \_delayFourCycles(unsigned int \_\_count)

{

if ( \_\_count == 0 )

\_\_asm\_\_ \_\_volatile\_\_( "rjmp 1f\n 1:" ); // 2 cycles

else

\_\_asm\_\_ \_\_volatile\_\_ (

"1: sbiw %0,1" "\n\t"

"brne 1b" // 4 cycles/loop

: "=w" (\_\_count)

: "0" (\_\_count)

);

}

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

delay for a minimum of <us> microseconds

the number of loops is calculated at compile-time from MCU clock frequency

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

#define delay(us) \_delayFourCycles( ( ( 1\*(XTAL/4000) )\*us)/1000 )

#if LCD\_IO\_MODE

/\* toggle Enable Pin to initiate write \*/

static void toggle\_e(void)

{

lcd\_e\_high();

lcd\_e\_delay();

lcd\_e\_low();

}

#endif

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

Low-level function to write byte to LCD controller

Input: data byte to write to LCD

rs 1: write data

0: write instruction

Returns: none

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

#if LCD\_IO\_MODE

static void lcd\_write(uint8\_t data,uint8\_t rs)

{

unsigned char dataBits ;

if (rs) { /\* write data (RS=1, RW=0) \*/

lcd\_rs\_high();

} else { /\* write instruction (RS=0, RW=0) \*/

lcd\_rs\_low();

}

lcd\_rw\_low();

if ( ( &LCD\_DATA0\_PORT == &LCD\_DATA1\_PORT) && ( &LCD\_DATA1\_PORT == &LCD\_DATA2\_PORT ) && ( &LCD\_DATA2\_PORT == &LCD\_DATA3\_PORT )

&& (LCD\_DATA0\_PIN == 0) && (LCD\_DATA1\_PIN == 1) && (LCD\_DATA2\_PIN == 2) && (LCD\_DATA3\_PIN == 3) )

{

/\* configure data pins as output \*/

DDR(LCD\_DATA0\_PORT) |= 0x0F;

/\* output high nibble first \*/

dataBits = LCD\_DATA0\_PORT & 0xF0;

LCD\_DATA0\_PORT = dataBits |((data>>4)&0x0F);

lcd\_e\_toggle();

/\* output low nibble \*/

LCD\_DATA0\_PORT = dataBits | (data&0x0F);

lcd\_e\_toggle();

/\* all data pins high (inactive) \*/

LCD\_DATA0\_PORT = dataBits | 0x0F;

}

else

{

/\* configure data pins as output \*/

DDR(LCD\_DATA0\_PORT) |= \_BV(LCD\_DATA0\_PIN);

DDR(LCD\_DATA1\_PORT) |= \_BV(LCD\_DATA1\_PIN);

DDR(LCD\_DATA2\_PORT) |= \_BV(LCD\_DATA2\_PIN);

DDR(LCD\_DATA3\_PORT) |= \_BV(LCD\_DATA3\_PIN);

/\* output high nibble first \*/

LCD\_DATA3\_PORT &= ~\_BV(LCD\_DATA3\_PIN);

LCD\_DATA2\_PORT &= ~\_BV(LCD\_DATA2\_PIN);

LCD\_DATA1\_PORT &= ~\_BV(LCD\_DATA1\_PIN);

LCD\_DATA0\_PORT &= ~\_BV(LCD\_DATA0\_PIN);

if(data & 0x80) LCD\_DATA3\_PORT |= \_BV(LCD\_DATA3\_PIN);

if(data & 0x40) LCD\_DATA2\_PORT |= \_BV(LCD\_DATA2\_PIN);

if(data & 0x20) LCD\_DATA1\_PORT |= \_BV(LCD\_DATA1\_PIN);

if(data & 0x10) LCD\_DATA0\_PORT |= \_BV(LCD\_DATA0\_PIN);

lcd\_e\_toggle();

/\* output low nibble \*/

LCD\_DATA3\_PORT &= ~\_BV(LCD\_DATA3\_PIN);

LCD\_DATA2\_PORT &= ~\_BV(LCD\_DATA2\_PIN);

LCD\_DATA1\_PORT &= ~\_BV(LCD\_DATA1\_PIN);

LCD\_DATA0\_PORT &= ~\_BV(LCD\_DATA0\_PIN);

if(data & 0x08) LCD\_DATA3\_PORT |= \_BV(LCD\_DATA3\_PIN);

if(data & 0x04) LCD\_DATA2\_PORT |= \_BV(LCD\_DATA2\_PIN);

if(data & 0x02) LCD\_DATA1\_PORT |= \_BV(LCD\_DATA1\_PIN);

if(data & 0x01) LCD\_DATA0\_PORT |= \_BV(LCD\_DATA0\_PIN);

lcd\_e\_toggle();

/\* all data pins high (inactive) \*/

LCD\_DATA0\_PORT |= \_BV(LCD\_DATA0\_PIN);

LCD\_DATA1\_PORT |= \_BV(LCD\_DATA1\_PIN);

LCD\_DATA2\_PORT |= \_BV(LCD\_DATA2\_PIN);

LCD\_DATA3\_PORT |= \_BV(LCD\_DATA3\_PIN);

}

}

#else

#define lcd\_write(d,rs) if (rs) \*(volatile uint8\_t\*)(LCD\_IO\_DATA) = d; else \*(volatile uint8\_t\*)(LCD\_IO\_FUNCTION) = d;

/\* rs==0 -> write instruction to LCD\_IO\_FUNCTION \*/

/\* rs==1 -> write data to LCD\_IO\_DATA \*/

#endif

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

Low-level function to read byte from LCD controller

Input: rs 1: read data

0: read busy flag / address counter

Returns: byte read from LCD controller

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

#if LCD\_IO\_MODE

static uint8\_t lcd\_read(uint8\_t rs)

{

uint8\_t data;

if (rs)

lcd\_rs\_high(); /\* RS=1: read data \*/

else

lcd\_rs\_low(); /\* RS=0: read busy flag \*/

lcd\_rw\_high(); /\* RW=1 read mode \*/

if ( ( &LCD\_DATA0\_PORT == &LCD\_DATA1\_PORT) && ( &LCD\_DATA1\_PORT == &LCD\_DATA2\_PORT ) && ( &LCD\_DATA2\_PORT == &LCD\_DATA3\_PORT )

&& ( LCD\_DATA0\_PIN == 0 )&& (LCD\_DATA1\_PIN == 1) && (LCD\_DATA2\_PIN == 2) && (LCD\_DATA3\_PIN == 3) )

{

DDR(LCD\_DATA0\_PORT) &= 0xF0; /\* configure data pins as input \*/

lcd\_e\_high();

lcd\_e\_delay();

data = PIN(LCD\_DATA0\_PORT) << 4; /\* read high nibble first \*/

lcd\_e\_low();

lcd\_e\_delay(); /\* Enable 500ns low \*/

lcd\_e\_high();

lcd\_e\_delay();

data |= PIN(LCD\_DATA0\_PORT)&0x0F; /\* read low nibble \*/

lcd\_e\_low();

}

else

{

/\* configure data pins as input \*/

DDR(LCD\_DATA0\_PORT) &= ~\_BV(LCD\_DATA0\_PIN);

DDR(LCD\_DATA1\_PORT) &= ~\_BV(LCD\_DATA1\_PIN);

DDR(LCD\_DATA2\_PORT) &= ~\_BV(LCD\_DATA2\_PIN);

DDR(LCD\_DATA3\_PORT) &= ~\_BV(LCD\_DATA3\_PIN);

/\* read high nibble first \*/

lcd\_e\_high();

lcd\_e\_delay();

data = 0;

if ( PIN(LCD\_DATA0\_PORT) & \_BV(LCD\_DATA0\_PIN) ) data |= 0x10;

if ( PIN(LCD\_DATA1\_PORT) & \_BV(LCD\_DATA1\_PIN) ) data |= 0x20;

if ( PIN(LCD\_DATA2\_PORT) & \_BV(LCD\_DATA2\_PIN) ) data |= 0x40;

if ( PIN(LCD\_DATA3\_PORT) & \_BV(LCD\_DATA3\_PIN) ) data |= 0x80;

lcd\_e\_low();

lcd\_e\_delay(); /\* Enable 500ns low \*/

/\* read low nibble \*/

lcd\_e\_high();

lcd\_e\_delay();

if ( PIN(LCD\_DATA0\_PORT) & \_BV(LCD\_DATA0\_PIN) ) data |= 0x01;

if ( PIN(LCD\_DATA1\_PORT) & \_BV(LCD\_DATA1\_PIN) ) data |= 0x02;

if ( PIN(LCD\_DATA2\_PORT) & \_BV(LCD\_DATA2\_PIN) ) data |= 0x04;

if ( PIN(LCD\_DATA3\_PORT) & \_BV(LCD\_DATA3\_PIN) ) data |= 0x08;

lcd\_e\_low();

}

return data;

}

#else

#define lcd\_read(rs) (rs) ? \*(volatile uint8\_t\*)(LCD\_IO\_DATA+LCD\_IO\_READ) : \*(volatile uint8\_t\*)(LCD\_IO\_FUNCTION+LCD\_IO\_READ)

/\* rs==0 -> read instruction from LCD\_IO\_FUNCTION \*/

/\* rs==1 -> read data from LCD\_IO\_DATA \*/

#endif

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

loops while lcd is busy, returns address counter

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

static uint8\_t lcd\_waitbusy(void)

{

register uint8\_t c;

/\* wait until busy flag is cleared \*/

while ( (c=lcd\_read(0)) & (1<<LCD\_BUSY)) {}

/\* the address counter is updated 4us after the busy flag is cleared \*/

delay(2);

/\* now read the address counter \*/

return (lcd\_read(0)); // return address counter

}/\* lcd\_waitbusy \*/

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

Move cursor to the start of next line or to the first line if the cursor

is already on the last line.

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

static inline void lcd\_newline(uint8\_t pos)

{

register uint8\_t addressCounter;

#if LCD\_LINES==1

addressCounter = 0;

#endif

#if LCD\_LINES==2

if ( pos < (LCD\_START\_LINE2) )

addressCounter = LCD\_START\_LINE2;

else

addressCounter = LCD\_START\_LINE1;

#endif

#if LCD\_LINES==4

#if KS0073\_4LINES\_MODE

if ( pos < LCD\_START\_LINE2 )

addressCounter = LCD\_START\_LINE2;

else if ( (pos >= LCD\_START\_LINE2) && (pos < LCD\_START\_LINE3) )

addressCounter = LCD\_START\_LINE3;

else if ( (pos >= LCD\_START\_LINE3) && (pos < LCD\_START\_LINE4) )

addressCounter = LCD\_START\_LINE4;

else

addressCounter = LCD\_START\_LINE1;

#else

if ( pos < LCD\_START\_LINE3 )

addressCounter = LCD\_START\_LINE2;

else if ( (pos >= LCD\_START\_LINE2) && (pos < LCD\_START\_LINE4) )

addressCounter = LCD\_START\_LINE3;

else if ( (pos >= LCD\_START\_LINE3) && (pos < LCD\_START\_LINE2) )

addressCounter = LCD\_START\_LINE4;

else

addressCounter = LCD\_START\_LINE1;

#endif

#endif

lcd\_command((1<<LCD\_DDRAM)+addressCounter);

}/\* lcd\_newline \*/

/\*

\*\* PUBLIC FUNCTIONS

\*/

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

Send LCD controller instruction command

Input: instruction to send to LCD controller, see HD44780 data sheet

Returns: none

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

void lcd\_command(uint8\_t cmd)

{

lcd\_waitbusy();

lcd\_write(cmd,0);

}

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

Send data byte to LCD controller

Input: data to send to LCD controller, see HD44780 data sheet

Returns: none

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

void lcd\_data(uint8\_t data)

{

lcd\_waitbusy();

lcd\_write(data,1);

}

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

Set cursor to specified position

Input: x horizontal position (0: left most position)

y vertical position (0: first line)

Returns: none

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

void lcd\_gotoxy(uint8\_t x, uint8\_t y)

{

#if LCD\_LINES==1

lcd\_command((1<<LCD\_DDRAM)+LCD\_START\_LINE1+x);

#endif

#if LCD\_LINES==2

if ( y==0 )

lcd\_command((1<<LCD\_DDRAM)+LCD\_START\_LINE1+x);

else

lcd\_command((1<<LCD\_DDRAM)+LCD\_START\_LINE2+x);

#endif

#if LCD\_LINES==4

if ( y==0 )

lcd\_command((1<<LCD\_DDRAM)+LCD\_START\_LINE1+x);

else if ( y==1)

lcd\_command((1<<LCD\_DDRAM)+LCD\_START\_LINE2+x);

else if ( y==2)

lcd\_command((1<<LCD\_DDRAM)+LCD\_START\_LINE3+x);

else /\* y==3 \*/

lcd\_command((1<<LCD\_DDRAM)+LCD\_START\_LINE4+x);

#endif

}/\* lcd\_gotoxy \*/

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

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

int lcd\_getxy(void)

{

return lcd\_waitbusy();

}

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

Clear display and set cursor to home position

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

void lcd\_clrscr(void)

{

lcd\_command(1<<LCD\_CLR);

}

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

Set cursor to home position

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

void lcd\_home(void)

{

lcd\_command(1<<LCD\_HOME);

}

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

Display character at current cursor position

Input: character to be displayed

Returns: none

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

void lcd\_putc(char c)

{

uint8\_t pos;

pos = lcd\_waitbusy(); // read busy-flag and address counter

if (c=='\n')

{

lcd\_newline(pos);

}

else

{

#if LCD\_WRAP\_LINES==1

#if LCD\_LINES==1

if ( pos == LCD\_START\_LINE1+LCD\_DISP\_LENGTH ) {

lcd\_write((1<<LCD\_DDRAM)+LCD\_START\_LINE1,0);

}

#elif LCD\_LINES==2

if ( pos == LCD\_START\_LINE1+LCD\_DISP\_LENGTH ) {

lcd\_write((1<<LCD\_DDRAM)+LCD\_START\_LINE2,0);

}else if ( pos == LCD\_START\_LINE2+LCD\_DISP\_LENGTH ){

lcd\_write((1<<LCD\_DDRAM)+LCD\_START\_LINE1,0);

}

#elif LCD\_LINES==4

if ( pos == LCD\_START\_LINE1+LCD\_DISP\_LENGTH ) {

lcd\_write((1<<LCD\_DDRAM)+LCD\_START\_LINE2,0);

}else if ( pos == LCD\_START\_LINE2+LCD\_DISP\_LENGTH ) {

lcd\_write((1<<LCD\_DDRAM)+LCD\_START\_LINE3,0);

}else if ( pos == LCD\_START\_LINE3+LCD\_DISP\_LENGTH ) {

lcd\_write((1<<LCD\_DDRAM)+LCD\_START\_LINE4,0);

}else if ( pos == LCD\_START\_LINE4+LCD\_DISP\_LENGTH ) {

lcd\_write((1<<LCD\_DDRAM)+LCD\_START\_LINE1,0);

}

#endif

lcd\_waitbusy();

#endif

lcd\_write(c, 1);

}

}/\* lcd\_putc \*/

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

Display string without auto linefeed

Input: string to be displayed

Returns: none

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

void lcd\_puts(const char \*s)

/\* print string on lcd (no auto linefeed) \*/

{

register char c;

while ( (c = \*s++) ) {

lcd\_putc(c);

}

}/\* lcd\_puts \*/

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

Display string from program memory without auto linefeed

Input: string from program memory be be displayed

Returns: none

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

void lcd\_puts\_p(const char \*progmem\_s)

/\* print string from program memory on lcd (no auto linefeed) \*/

{

register char c;

while ( (c = pgm\_read\_byte(progmem\_s++)) ) {

lcd\_putc(c);

}

}/\* lcd\_puts\_p \*/

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

Initialize display and select type of cursor

Input: dispAttr LCD\_DISP\_OFF display off

LCD\_DISP\_ON display on, cursor off

LCD\_DISP\_ON\_CURSOR display on, cursor on

LCD\_DISP\_CURSOR\_BLINK display on, cursor on flashing

Returns: none

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

void lcd\_init(uint8\_t dispAttr)

{

#if LCD\_IO\_MODE

/\*

\* Initialize LCD to 4 bit I/O mode

\*/

if ( ( &LCD\_DATA0\_PORT == &LCD\_DATA1\_PORT) && ( &LCD\_DATA1\_PORT == &LCD\_DATA2\_PORT ) && ( &LCD\_DATA2\_PORT == &LCD\_DATA3\_PORT )

&& ( &LCD\_RS\_PORT == &LCD\_DATA0\_PORT) && ( &LCD\_RW\_PORT == &LCD\_DATA0\_PORT) && (&LCD\_E\_PORT == &LCD\_DATA0\_PORT)

&& (LCD\_DATA0\_PIN == 0 ) && (LCD\_DATA1\_PIN == 1) && (LCD\_DATA2\_PIN == 2) && (LCD\_DATA3\_PIN == 3)

&& (LCD\_RS\_PIN == 4 ) && (LCD\_RW\_PIN == 5) && (LCD\_E\_PIN == 6 ) )

{

/\* configure all port bits as output (all LCD lines on same port) \*/

DDR(LCD\_DATA0\_PORT) |= 0x7F;

}

else if ( ( &LCD\_DATA0\_PORT == &LCD\_DATA1\_PORT) && ( &LCD\_DATA1\_PORT == &LCD\_DATA2\_PORT ) && ( &LCD\_DATA2\_PORT == &LCD\_DATA3\_PORT )

&& (LCD\_DATA0\_PIN == 0 ) && (LCD\_DATA1\_PIN == 1) && (LCD\_DATA2\_PIN == 2) && (LCD\_DATA3\_PIN == 3) )

{

/\* configure all port bits as output (all LCD data lines on same port, but control lines on different ports) \*/

DDR(LCD\_DATA0\_PORT) |= 0x0F;

DDR(LCD\_RS\_PORT) |= \_BV(LCD\_RS\_PIN);

DDR(LCD\_RW\_PORT) |= \_BV(LCD\_RW\_PIN);

DDR(LCD\_E\_PORT) |= \_BV(LCD\_E\_PIN);

}

else

{

/\* configure all port bits as output (LCD data and control lines on different ports \*/

DDR(LCD\_RS\_PORT) |= \_BV(LCD\_RS\_PIN);

DDR(LCD\_RW\_PORT) |= \_BV(LCD\_RW\_PIN);

DDR(LCD\_E\_PORT) |= \_BV(LCD\_E\_PIN);

DDR(LCD\_DATA0\_PORT) |= \_BV(LCD\_DATA0\_PIN);

DDR(LCD\_DATA1\_PORT) |= \_BV(LCD\_DATA1\_PIN);

DDR(LCD\_DATA2\_PORT) |= \_BV(LCD\_DATA2\_PIN);

DDR(LCD\_DATA3\_PORT) |= \_BV(LCD\_DATA3\_PIN);

}

delay(16000); /\* wait 16ms or more after power-on \*/

/\* initial write to lcd is 8bit \*/

LCD\_DATA1\_PORT |= \_BV(LCD\_DATA1\_PIN); // \_BV(LCD\_FUNCTION)>>4;

LCD\_DATA0\_PORT |= \_BV(LCD\_DATA0\_PIN); // \_BV(LCD\_FUNCTION\_8BIT)>>4;

lcd\_e\_toggle();

delay(4992); /\* delay, busy flag can't be checked here \*/

/\* repeat last command \*/

lcd\_e\_toggle();

delay(64); /\* delay, busy flag can't be checked here \*/

/\* repeat last command a third time \*/

lcd\_e\_toggle();

delay(64); /\* delay, busy flag can't be checked here \*/

/\* now configure for 4bit mode \*/

LCD\_DATA0\_PORT &= ~\_BV(LCD\_DATA0\_PIN); // LCD\_FUNCTION\_4BIT\_1LINE>>4

lcd\_e\_toggle();

delay(64); /\* some displays need this additional delay \*/

/\* from now the LCD only accepts 4 bit I/O, we can use lcd\_command() \*/

#else

/\*

\* Initialize LCD to 8 bit memory mapped mode

\*/

/\* enable external SRAM (memory mapped lcd) and one wait state \*/

MCUCR = \_BV(SRE) | \_BV(SRW);

/\* reset LCD \*/

delay(16000); /\* wait 16ms after power-on \*/

lcd\_write(LCD\_FUNCTION\_8BIT\_1LINE,0); /\* function set: 8bit interface \*/

delay(4992); /\* wait 5ms \*/

lcd\_write(LCD\_FUNCTION\_8BIT\_1LINE,0); /\* function set: 8bit interface \*/

delay(64); /\* wait 64us \*/

lcd\_write(LCD\_FUNCTION\_8BIT\_1LINE,0); /\* function set: 8bit interface \*/

delay(64); /\* wait 64us \*/

#endif

#if KS0073\_4LINES\_MODE

/\* Display with KS0073 controller requires special commands for enabling 4 line mode \*/

lcd\_command(KS0073\_EXTENDED\_FUNCTION\_REGISTER\_ON);

lcd\_command(KS0073\_4LINES\_MODE);

lcd\_command(KS0073\_EXTENDED\_FUNCTION\_REGISTER\_OFF);

#else

lcd\_command(LCD\_FUNCTION\_DEFAULT); /\* function set: display lines \*/

#endif

lcd\_command(LCD\_DISP\_OFF); /\* display off \*/

lcd\_clrscr(); /\* display clear \*/

lcd\_command(LCD\_MODE\_DEFAULT); /\* set entry mode \*/

lcd\_command(dispAttr); /\* display/cursor control \*/

}/\* lcd\_init \*/

#ifndef LCD\_H

#define LCD\_H

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

Title : C include file for the HD44780U LCD library (lcd.c)

File: $Id: lcd.h,v 1.13.2.2 2006/01/30 19:51:33 peter Exp $

Software: AVR-GCC 3.3

Hardware: any AVR device, memory mapped mode only for AT90S4414/8515/Mega

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

/\*\*

@defgroup pfleury\_lcd LCD library

@code #include <lcd.h> @endcode

@brief Basic routines for interfacing a HD44780U-based text LCD display

Originally based on Volker Oth's LCD library,

changed lcd\_init(), added additional constants for lcd\_command(),

added 4-bit I/O mode, improved and optimized code.

Library can be operated in memory mapped mode (LCD\_IO\_MODE=0) or in

4-bit IO port mode (LCD\_IO\_MODE=1). 8-bit IO port mode not supported.

Memory mapped mode compatible with Kanda STK200, but supports also

generation of R/W signal through A8 address line.

@author Peter Fleury pfleury@gmx.ch http://jump.to/fleury

@see The chapter <a href="http://homepage.sunrise.ch/mysunrise/peterfleury/avr-lcd44780.html" target="\_blank">Interfacing a HD44780 Based LCD to an AVR</a>

on my home page.

\*/

/\*@{\*/

#if (\_\_GNUC\_\_ \* 100 + \_\_GNUC\_MINOR\_\_) < 303

#error "This library requires AVR-GCC 3.3 or later, update to newer AVR-GCC compiler !"

#endif

#include <inttypes.h>

#include <avr/pgmspace.h>

/\*\*

\* @name Definitions for MCU Clock Frequency

\* Adapt the MCU clock frequency in Hz to your target.

\*/

#define XTAL 16000000 /\*\*< clock frequency in Hz, used to calculate delay timer \*/

/\*\*

\* @name Definition for LCD controller type

\* Use 0 for HD44780 controller, change to 1 for displays with KS0073 controller.

\*/

#define LCD\_CONTROLLER\_KS0073 0 /\*\*< Use 0 for HD44780 controller, 1 for KS0073 controller \*/

/\*\*

\* @name Definitions for Display Size

\* Change these definitions to adapt setting to your display

\*/

#define LCD\_LINES 2 /\*\*< number of visible lines of the display \*/

#define LCD\_DISP\_LENGTH 16 /\*\*< visibles characters per line of the display \*/

#define LCD\_LINE\_LENGTH 0x40 /\*\*< internal line length of the display \*/

#define LCD\_START\_LINE1 0x00 /\*\*< DDRAM address of first char of line 1 \*/

#define LCD\_START\_LINE2 0x40 /\*\*< DDRAM address of first char of line 2 \*/

#define LCD\_START\_LINE3 0x14 /\*\*< DDRAM address of first char of line 3 \*/

#define LCD\_START\_LINE4 0x54 /\*\*< DDRAM address of first char of line 4 \*/

#define LCD\_WRAP\_LINES 0 /\*\*< 0: no wrap, 1: wrap at end of visibile line \*/

#define LCD\_IO\_MODE 1 /\*\*< 0: memory mapped mode, 1: IO port mode \*/

#if LCD\_IO\_MODE

/\*\*

\* @name Definitions for 4-bit IO mode

\* Change LCD\_PORT if you want to use a different port for the LCD pins.

\*

\* The four LCD data lines and the three control lines RS, RW, E can be on the

\* same port or on different ports.

\* Change LCD\_RS\_PORT, LCD\_RW\_PORT, LCD\_E\_PORT if you want the control lines on

\* different ports.

\*

\* Normally the four data lines should be mapped to bit 0..3 on one port, but it

\* is possible to connect these data lines in different order or even on different

\* ports by adapting the LCD\_DATAx\_PORT and LCD\_DATAx\_PIN definitions.

\*

\*/

#define LCD\_PORT PORTD /\*\*< port for the LCD lines \*/

#define LCD\_DATA0\_PORT LCD\_PORT /\*\*< port for 4bit data bit 0 \*/

#define LCD\_DATA1\_PORT LCD\_PORT /\*\*< port for 4bit data bit 1 \*/

#define LCD\_DATA2\_PORT LCD\_PORT /\*\*< port for 4bit data bit 2 \*/

#define LCD\_DATA3\_PORT LCD\_PORT /\*\*< port for 4bit data bit 3 \*/

#define LCD\_DATA0\_PIN 0 /\*\*< pin for 4bit data bit 0 \*/

#define LCD\_DATA1\_PIN 1 /\*\*< pin for 4bit data bit 1 \*/

#define LCD\_DATA2\_PIN 2 /\*\*< pin for 4bit data bit 2 \*/

#define LCD\_DATA3\_PIN 3 /\*\*< pin for 4bit data bit 3 \*/

#define LCD\_RS\_PORT LCD\_PORT /\*\*< port for RS line \*/

#define LCD\_RS\_PIN 4 /\*\*< pin for RS line \*/

#define LCD\_RW\_PORT LCD\_PORT /\*\*< port for RW line \*/

#define LCD\_RW\_PIN 5 /\*\*< pin for RW line \*/

#define LCD\_E\_PORT LCD\_PORT /\*\*< port for Enable line \*/

#define LCD\_E\_PIN 6 /\*\*< pin for Enable line \*/

#elif defined(\_\_AVR\_AT90S4414\_\_) || defined(\_\_AVR\_AT90S8515\_\_) || defined(\_\_AVR\_ATmega64\_\_) || \

defined(\_\_AVR\_ATmega8515\_\_)|| defined(\_\_AVR\_ATmega103\_\_) || defined(\_\_AVR\_ATmega128\_\_) || \

defined(\_\_AVR\_ATmega161\_\_) || defined(\_\_AVR\_ATmega162\_\_)

/\*

\* memory mapped mode is only supported when the device has an external data memory interface

\*/

#define LCD\_IO\_DATA 0xC000 /\* A15=E=1, A14=RS=1 \*/

#define LCD\_IO\_FUNCTION 0x8000 /\* A15=E=1, A14=RS=0 \*/

#define LCD\_IO\_READ 0x0100 /\* A8 =R/W=1 (R/W: 1=Read, 0=Write \*/

#else

#error "external data memory interface not available for this device, use 4-bit IO port mode"

#endif

/\*\*

\* @name Definitions for LCD command instructions

\* The constants define the various LCD controller instructions which can be passed to the

\* function lcd\_command(), see HD44780 data sheet for a complete description.

\*/

/\* instruction register bit positions, see HD44780U data sheet \*/

#define LCD\_CLR 0 /\* DB0: clear display \*/

#define LCD\_HOME 1 /\* DB1: return to home position \*/

#define LCD\_ENTRY\_MODE 2 /\* DB2: set entry mode \*/

#define LCD\_ENTRY\_INC 1 /\* DB1: 1=increment, 0=decrement \*/

#define LCD\_ENTRY\_SHIFT 0 /\* DB2: 1=display shift on \*/

#define LCD\_ON 3 /\* DB3: turn lcd/cursor on \*/

#define LCD\_ON\_DISPLAY 2 /\* DB2: turn display on \*/

#define LCD\_ON\_CURSOR 1 /\* DB1: turn cursor on \*/

#define LCD\_ON\_BLINK 0 /\* DB0: blinking cursor ? \*/

#define LCD\_MOVE 4 /\* DB4: move cursor/display \*/

#define LCD\_MOVE\_DISP 3 /\* DB3: move display (0-> cursor) ? \*/

#define LCD\_MOVE\_RIGHT 2 /\* DB2: move right (0-> left) ? \*/

#define LCD\_FUNCTION 5 /\* DB5: function set \*/

#define LCD\_FUNCTION\_8BIT 4 /\* DB4: set 8BIT mode (0->4BIT mode) \*/

#define LCD\_FUNCTION\_2LINES 3 /\* DB3: two lines (0->one line) \*/

#define LCD\_FUNCTION\_10DOTS 2 /\* DB2: 5x10 font (0->5x7 font) \*/

#define LCD\_CGRAM 6 /\* DB6: set CG RAM address \*/

#define LCD\_DDRAM 7 /\* DB7: set DD RAM address \*/

#define LCD\_BUSY 7 /\* DB7: LCD is busy \*/

/\* set entry mode: display shift on/off, dec/inc cursor move direction \*/

#define LCD\_ENTRY\_DEC 0x04 /\* display shift off, dec cursor move dir \*/

#define LCD\_ENTRY\_DEC\_SHIFT 0x05 /\* display shift on, dec cursor move dir \*/

#define LCD\_ENTRY\_INC\_ 0x06 /\* display shift off, inc cursor move dir \*/

#define LCD\_ENTRY\_INC\_SHIFT 0x07 /\* display shift on, inc cursor move dir \*/

/\* display on/off, cursor on/off, blinking char at cursor position \*/

#define LCD\_DISP\_OFF 0x08 /\* display off \*/

#define LCD\_DISP\_ON 0x0C /\* display on, cursor off \*/

#define LCD\_DISP\_ON\_BLINK 0x0D /\* display on, cursor off, blink char \*/

#define LCD\_DISP\_ON\_CURSOR 0x0E /\* display on, cursor on \*/

#define LCD\_DISP\_ON\_CURSOR\_BLINK 0x0F /\* display on, cursor on, blink char \*/

/\* move cursor/shift display \*/

#define LCD\_MOVE\_CURSOR\_LEFT 0x10 /\* move cursor left (decrement) \*/

#define LCD\_MOVE\_CURSOR\_RIGHT 0x14 /\* move cursor right (increment) \*/

#define LCD\_MOVE\_DISP\_LEFT 0x18 /\* shift display left \*/

#define LCD\_MOVE\_DISP\_RIGHT 0x1C /\* shift display right \*/

/\* function set: set interface data length and number of display lines \*/

#define LCD\_FUNCTION\_4BIT\_1LINE 0x20 /\* 4-bit interface, single line, 5x7 dots \*/

#define LCD\_FUNCTION\_4BIT\_2LINES 0x28 /\* 4-bit interface, dual line, 5x7 dots \*/

#define LCD\_FUNCTION\_8BIT\_1LINE 0x30 /\* 8-bit interface, single line, 5x7 dots \*/

#define LCD\_FUNCTION\_8BIT\_2LINES 0x38 /\* 8-bit interface, dual line, 5x7 dots \*/

#define LCD\_MODE\_DEFAULT ((1<<LCD\_ENTRY\_MODE) | (1<<LCD\_ENTRY\_INC) )

/\*\*

\* @name Functions

\*/

/\*\*

@brief Initialize display and select type of cursor

@param dispAttr \b LCD\_DISP\_OFF display off\n

\b LCD\_DISP\_ON display on, cursor off\n

\b LCD\_DISP\_ON\_CURSOR display on, cursor on\n

\b LCD\_DISP\_ON\_CURSOR\_BLINK display on, cursor on flashing

@return none

\*/

extern void lcd\_init(uint8\_t dispAttr);

/\*\*

@brief Clear display and set cursor to home position

@param void

@return none

\*/

extern void lcd\_clrscr(void);

/\*\*

@brief Set cursor to home position

@param void

@return none

\*/

extern void lcd\_home(void);

/\*\*

@brief Set cursor to specified position

@param x horizontal position\n (0: left most position)

@param y vertical position\n (0: first line)

@return none

\*/

extern void lcd\_gotoxy(uint8\_t x, uint8\_t y);

/\*\*

@brief Display character at current cursor position

@param c character to be displayed

@return none

\*/

extern void lcd\_putc(char c);

/\*\*

@brief Display string without auto linefeed

@param s string to be displayed

@return none

\*/

extern void lcd\_puts(const char \*s);

/\*\*

@brief Display string from program memory without auto linefeed

@param s string from program memory be be displayed

@return none

@see lcd\_puts\_P

\*/

extern void lcd\_puts\_p(const char \*progmem\_s);

/\*\*

@brief Send LCD controller instruction command

@param cmd instruction to send to LCD controller, see HD44780 data sheet

@return none

\*/

extern void lcd\_command(uint8\_t cmd);

/\*\*

@brief Send data byte to LCD controller

Similar to lcd\_putc(), but without interpreting LF

@param data byte to send to LCD controller, see HD44780 data sheet

@return none

\*/

extern void lcd\_data(uint8\_t data);

/\*\*

@brief macros for automatically storing string constant in program memory

\*/

#define lcd\_puts\_P(\_\_s) lcd\_puts\_p(PSTR(\_\_s))

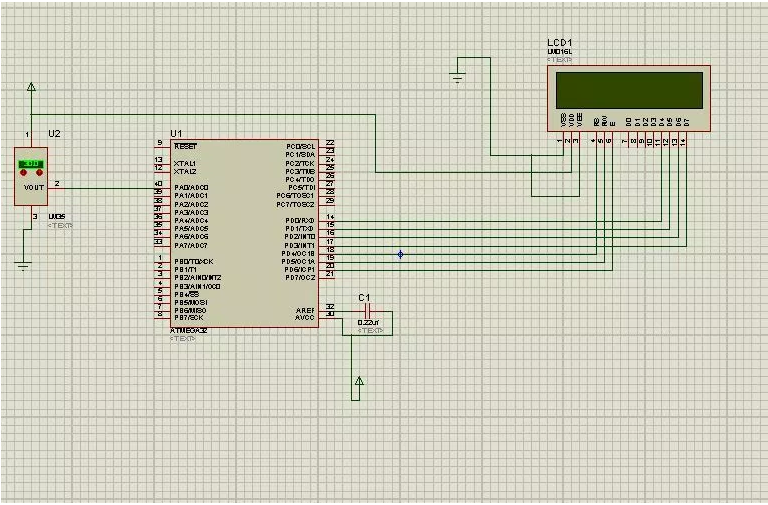
/\*@}\*/

#endif //LCD\_H

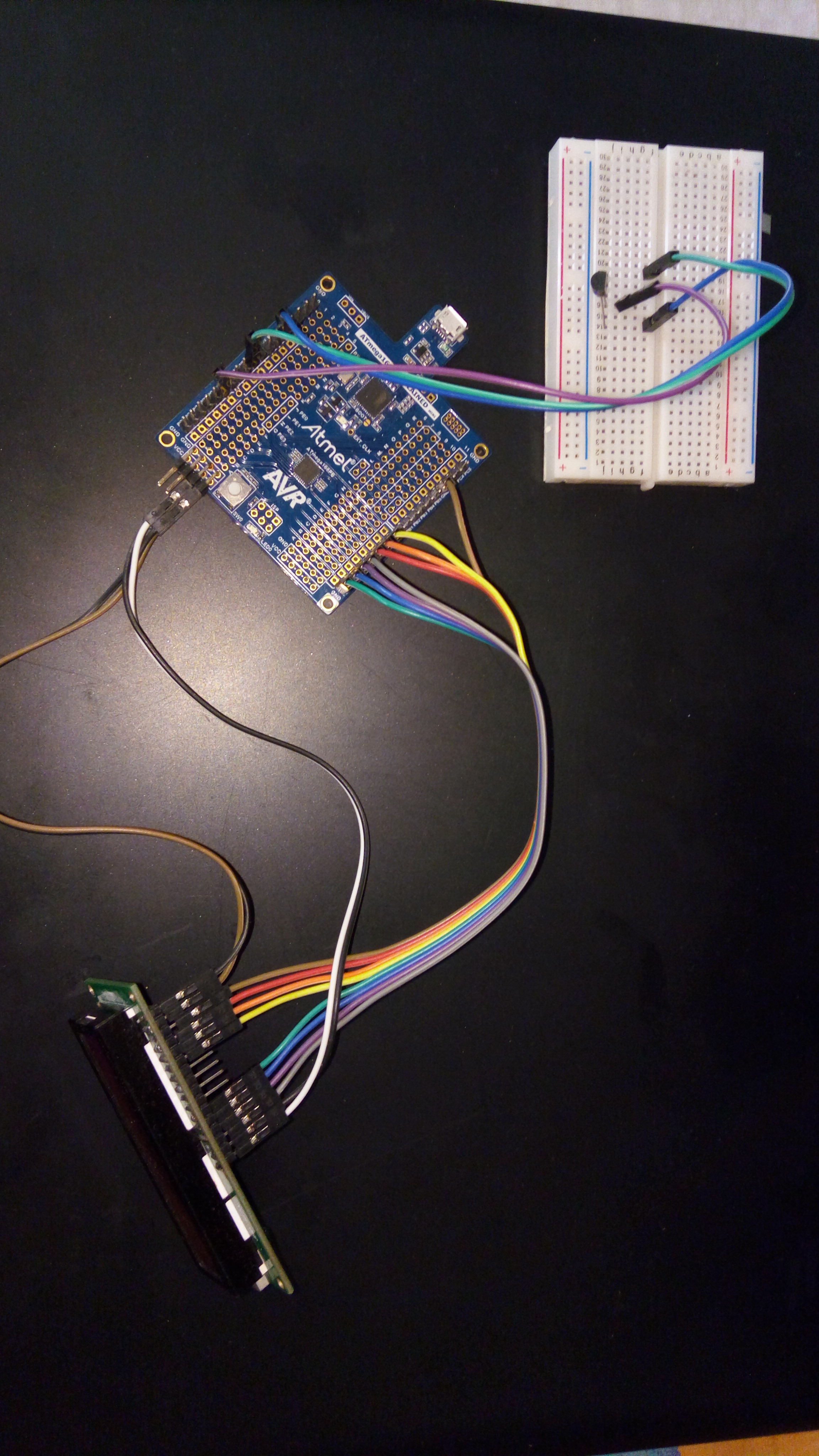
|  |  |  |  |
| --- | --- | --- | --- |
| 2. | Flow Chart |  |  |



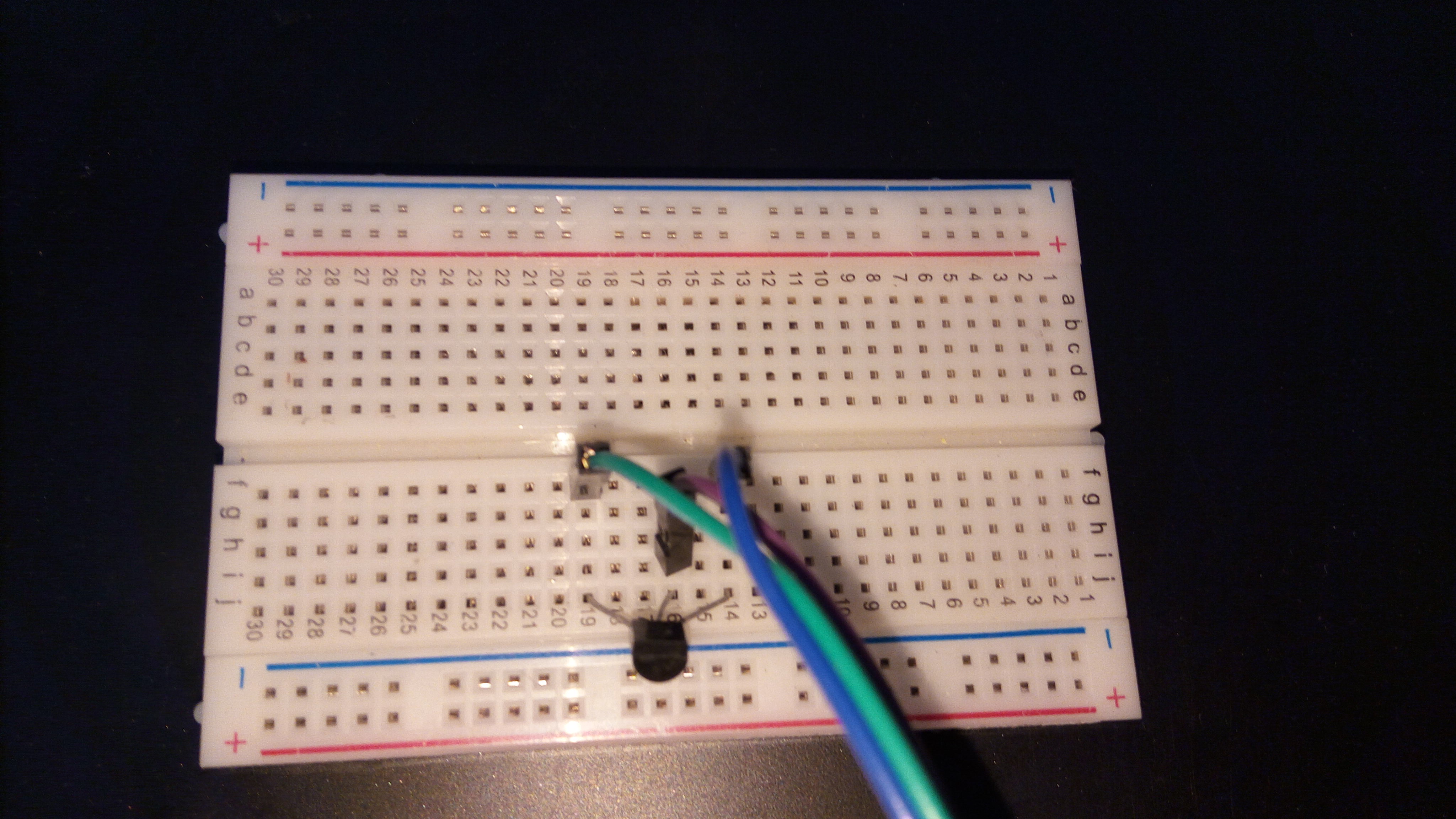
|  |  |  |  |
| --- | --- | --- | --- |
| 1. | SCHEMATIC |  |  |

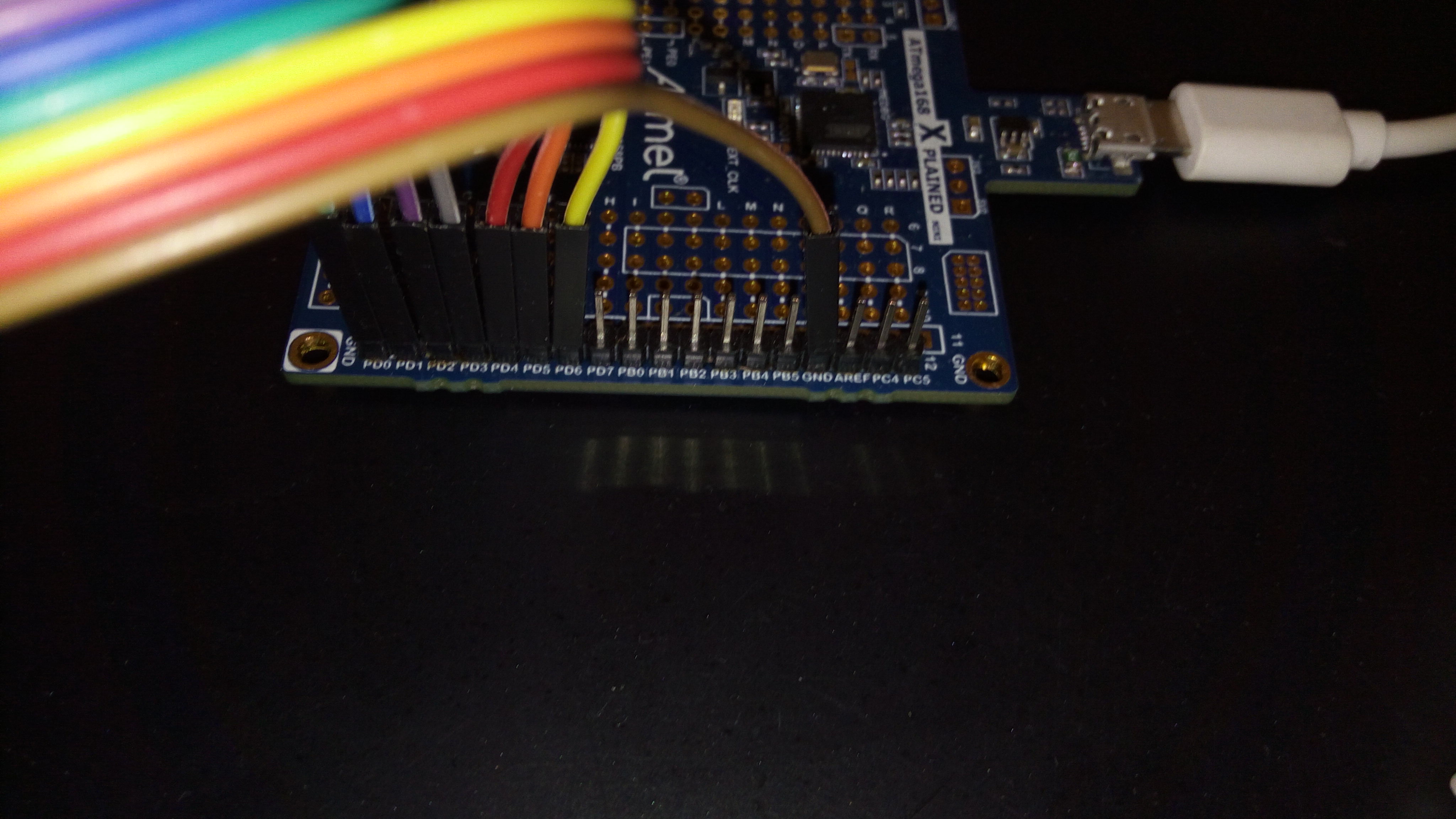


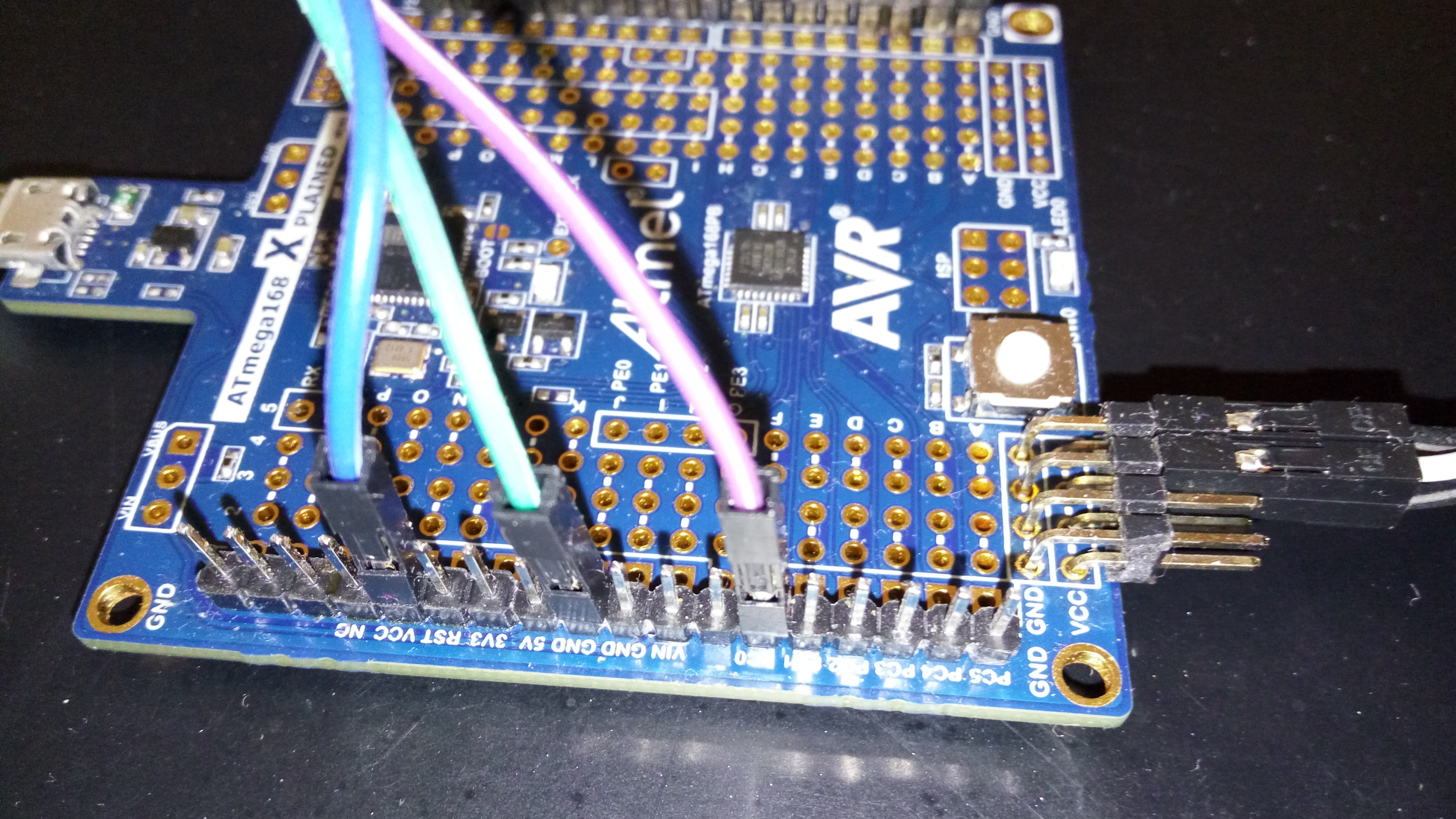
|  |  |  |  |
| --- | --- | --- | --- |
| 2. | SNAPSHOT OF THE BOARD WITH CONNECTED COMPONENTS |  |  |
|  |  |  |  |



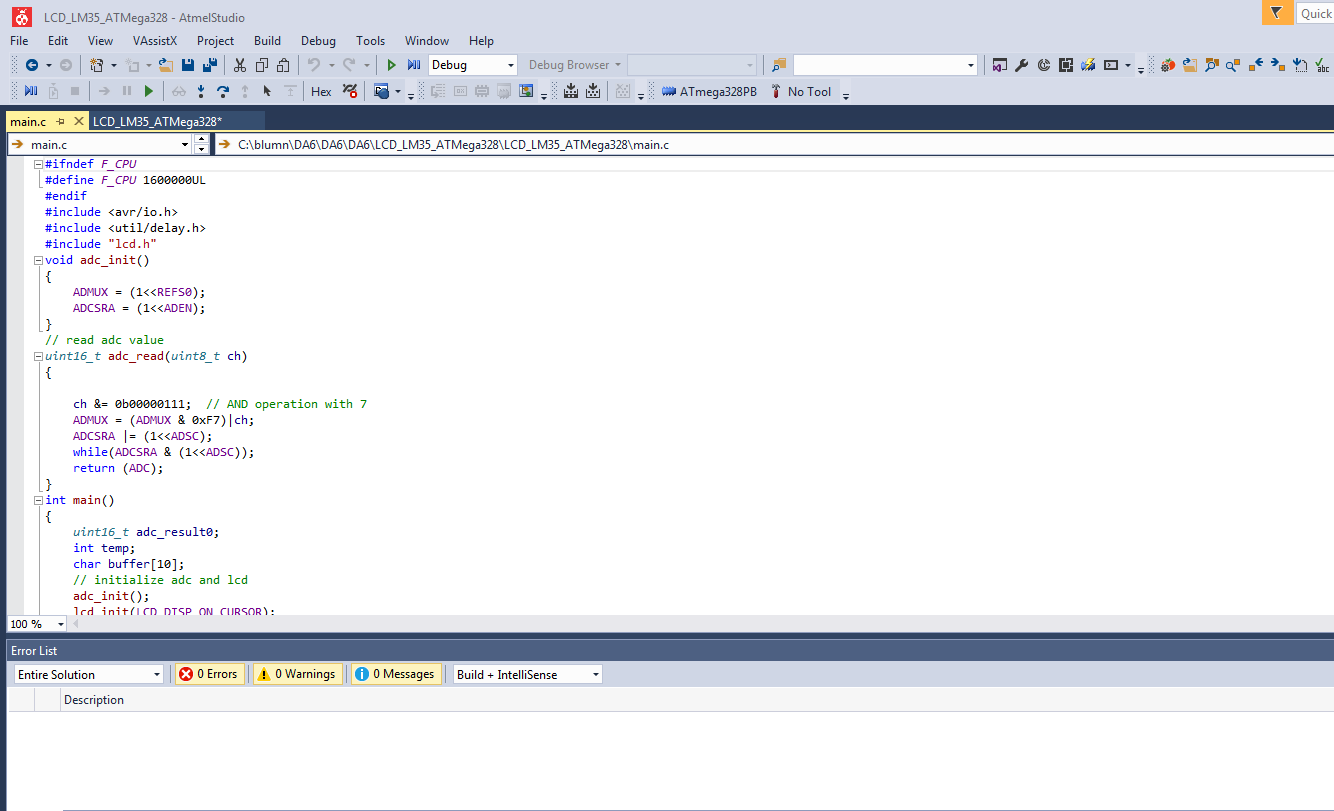








|  |  |  |  |
| --- | --- | --- | --- |
| 4. | SNAPSHOT OF THE COMPUTER SCREEN WITH TERMINAL |  |  |



|  |  |  |  |
| --- | --- | --- | --- |
| 5. | Video Link (You tube) |  |  |

<http://www.youtube.com/watch?v=jVzBVwQW5VM>

**Student Academic Misconduct Policy**

“This assignment submission is my own, original work”.

Noor Blum