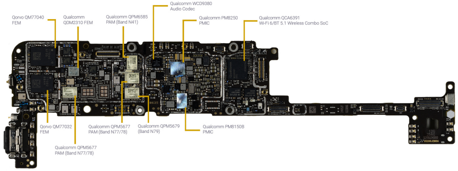

Qorvo QM77040 FEM
Qualcomm QDM2310 FEM
Qualcomm QPM6585 PAM (Band N41)
Qualcomm WCD9380 Audio Codec
Qualcomm PM8250 PMIC
Qualcomm QCA6391 Wi-Fi 6/BT 5.1 Wireless Combo SoC
Qorvo QM77032 FEM
Qualcomm QPM5677 PAM (Band N77/78)
Qualcomm QPM5679 (Band N79)
Qualcomm PM8150B PMIC
Qualcomm QPM5677 PAM (Band N77/78)

Interfacing to the external world

Kizito NKURIKIYEYEZU, Ph.D.

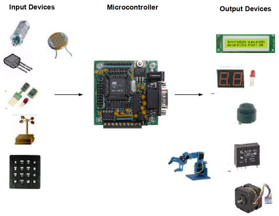
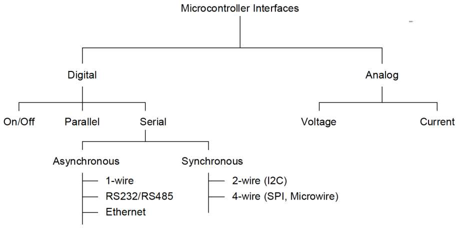


FIG 1. Micro-controllers are used to communicate with other devices, such as sensors, motors, switches, keypads, displays, memory and even other micro-controllers. In a very simplistic form, a micro-controller system can be viewed as a system that reads from (monitors) inputs, performs processing and writes to (controls) outputs.

Microcontroller Interface types



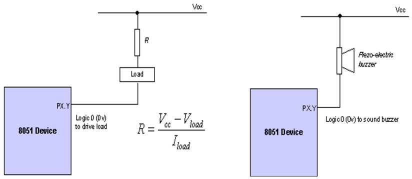
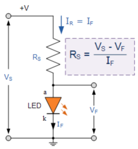

Vcc
R
Load
PX.Y
8051 Device
Logic 0 (0v) to drive load
$R = \frac{V_{cc} - V_{load}}{I_{load}}$
Vcc
Piezo-electric buzzer
PX.Y
8051 Device
Logic 0 (0v) to sound buzzer

FIG 2. **Driving a low-power load**
It is possible to drive low-power loads directly from the port pins. Note that we usually need a resistor between the supply and the port pin. This resistor is required to limit the current flow to the port. Remember that Each I/O pin can sink or source a maximum current of 40mA. Although each pin can sink or source 40mA current, it must be ensured that the current sourced or sunk from all the ports combined, should not exceed 200mA.

LED Interface



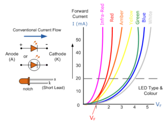
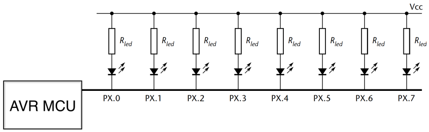


FIG 3. **Light Emitting Diodes I-V Characteristics** An LED is a diode; thus, its forward current to voltage characteristics curves can be plotted for each diode color. Note a voltage greater than Vf must be applied across the leads of the LED, from anode to cathode, in order for the LED to turn on.

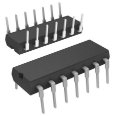
The need for IC buffer

- Suppose we try to control eight 10 mA LEDs from a single port using the techniques



- This approach won’t work because each I/O pin can sink or source a maximum current of 40mA. Although each pin can sink or source 40mA current, it must be ensured that the current sourced or sunk from all the ports combined, should not exceed 200mA.

The need for IC buffer



- Buffers can be needed if we need to drive multiple lowpower loads from one microcontroller
- With (5V) TTL buffer, the Logic 0 output is in the range 0 to 1.5V; the Logic 1 output is 3.5 to 5V.
- With (5V) CMOS, the Logic 0 output is 0V; the Logic 1 output is 5V

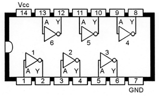


FIG 4. Functional diagram of the 7404, 74LS04, 74HC04, 74HCU04, or 4069UB Hex inverter ICs.

TTL versus CMOS buffers

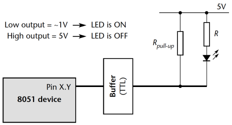


FIG 5. Using a TTL buffer

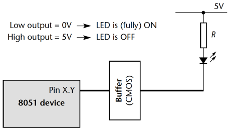

Low output = 0V → LED is (fully) ON
High output = 5V → LED is OFF
5V
R
Pin X.Y
8051 device
Buffer (CMOS)

FIG 6. Using a CMOS buffer

Notes:

- It almost always makes sense to use CMOS logic in your buffer designs wherever possible.
- When working with port pins that do not have internal pull-up resistors, you need to include such resistors (10K will do) at the input to the IC buffer, whatever kind of logic you are using.

Driving LED with a buffer

A buffer acts as a driver for an LED, as it can provide more drive current than the GPIO pins.

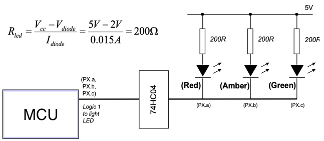


FIG 7. **Buffering three LEDs with a 74HC04**

In this approach, there is no need for pull-up resistors with the HC (CMOS) buffers. In this case, we assume that the LEDs are to be driven at 15 mA each. A buffer can provide up to 50 mA total.

BJT driver

How do you drive DC loads requiring currents of up to around 2A, from the port of a microcontroller?

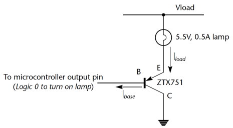


FIG 8. A PNP transistor driving a lamp.

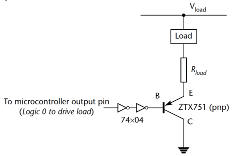


FIG 9. Adding a buffer protects the MCU in case the load changes (e.g., is physically damaged and creates a short-circuit)

Switching off inductive DC loads

- An inductive load is anything containing a coil of wire: common examples are electromechanical relays and motors.
- When the current is removed (i.e., the load is turned off) the voltage across the inductor will increase rapidly as shown in Equation (1). This voltage spike might destroy the circuit connected to the load.

$$V = L\frac{dI}{dt} \qquad (1)$$

  - V is the voltage across the inductor
  - L is the inductance
  - dI/dt is the rate of change of current flow

Switching off inductive DC loads

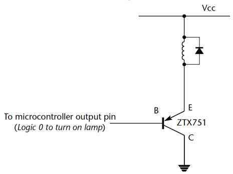


FIG 10. Protecting transistors against inductive kick with a diode

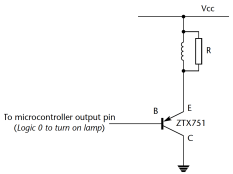


FIG 11. Protecting transistors against inductive kick with a resistor. This is best used for faster current decay after switch off, a resistor can be used in place of the diode

Example: Driving a high-power IR LED

- Infra-red (IR) LEDs are widely used remote control. They are also a component in many security systems.
- IR LEDs often have higher current requirements than conventional LEDs, but are otherwise connected in the same way.
- For example, the Siemens SFH485 IR LED requires a current of 100 mA, at a forward voltage of 1.5V.
- If we use a 2N2905 transistor—which can handle a maximum load current of 600 mA—$I_{LED} = 100mA$, a supply voltage of 5V and an LED forward voltage of 1.5V, the required value of R2 is:

$$R2 = \frac{V_{CC} - V_T - V_{CE}}{I_{LED}} = \frac{5V - 1.5V - 0.4V}{0.1A} = 31\,\Omega \quad (2)$$

The saturation voltage, $VCE = 0.4V$ is from the transistor

Example: Driving a high-power IR LED

FIG 12. Infrared 5mm T1-3/4 LED Emitter Osram SFH 485 SFH485

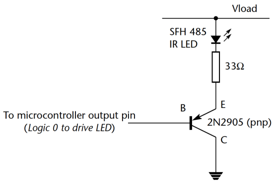


FIG 13. Driving an IR LED via a transistor (PNP) driver

Example: Driving a large buzzer

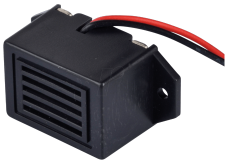


FIG 14. SUMMER 12V Miniature buzzer, 9–15 V

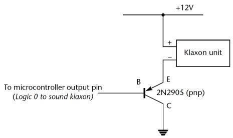


FIG 15. Controlling the buzzer with a 2N2905 BJT

MOSFET driver

How do you control a DC load with high current requirements (up to around 100A) using a microcontroller?

- MOSFETs provide a flexible and cost-effective solution to many high-power applications, able to switch currents of up to 100A or more
- Some MOSFETS can be driven directly from a processor port. However, in most cases it is necessary to drive the device with a higher voltage (12V +) than is available from the port itself.
- In addition,it is generally good policy to have an extra layer of protection between the ports and any high-power load.

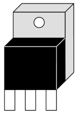


FIG 16. A typical TO-220 power MOSFET

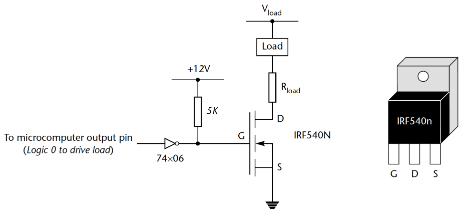


FIG 17. **Using a MOSFET driver**
Note that a Logic 0 output is required to switch on the load. Also, note that, as we are using the 74×06 driver for level shifting, the pull-up resistor is required

Example: Open-loop DC motor control

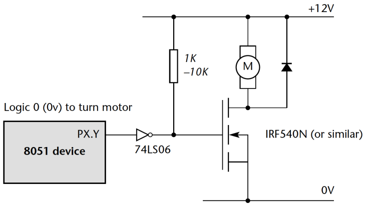


FIG 18. **Control of a DC motor using a MOSFET driver**
Circuitry control a 12V, 2A DC motor using a MOSFET driver.Note the use of the diode
to protect against inductive kick when the motor is switched off

Solid state relay driver

How do you “switch on” or “switch off” a piece of high-voltage (DC) electrical equipment using a microcontroller?

- A solid state relay (SSR) is an electronic switching device that switches on or off when an external voltage (AC or DC) is applied across its control terminals.
- It serves the same function as an electromechanical relay, but has no moving parts and therefore results in a longer operational lifetime.

FIG 19. **A 40A, 3-32V solid state relay**
Each one of these relays is equipped with four screw terminals (for use with ring or fork connectors) and a plastic cover that slides over the top of the relay to protect the terminals.

Example: Open-loop DC motor control with a SSR

- If we use an appropriate SSR we can simplify this circuit considerably.
- For example, previous example’s motor required 2A at up to 12V for correct operation.
- Here we can use an IOR PVNO12 SSR. Unlike the majority of SSRs, this can switch AC or DC loads, of up to 20 V, 4.5A. It has no zero-crossing detection. The control current is a

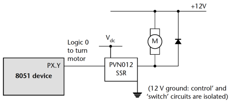
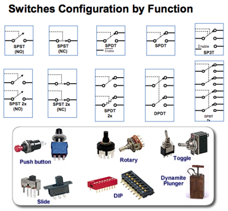


FIG 20. Open-loop DC motor control using a solid-state relay

Interface with switch inputs

Types of switches



[1] https://www.electronicshub.org/switches/

Interface with switches

One way to interface a switch to a microcontroller is shown in Figure 21

- The pin P1.0 must be first programmed as an input pin by writing 1 to its Latch.
- When the switch is open, then the current will flow from VCC to P1.0 and P1.0 will read 1.
- When the switch is closed, then the current will flow from VCC to GND and P1.0 will read 0

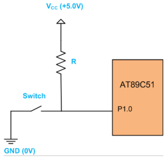


FIG 21. Switch interfacing

Interface with switches

- The DC design requirement for the interface is to is to make sure that $IR < I_{IH}$
- To do this, we need to determine the value of the current limiting resistor R in order to protect the P1.0. Assuming $I_{IH} = 650uA$, the value of R is calculated as shown in Equation (3):

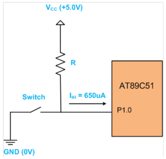
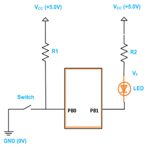


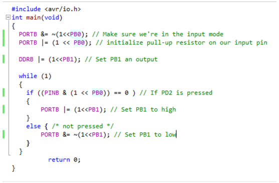
FIG 22. Switch interfacing

$$R = \frac{V_{CC}}{I_{IH}} = \frac{5\,V}{650\mu A} = 7692.3\,\Omega \tag{3}$$

Example: Switch interface

The code below turns on the LED when the switch is pressed. The LED is off otherwise



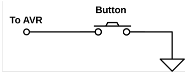

```c
#include <avr/io.h>
int main(void)
{
  PORTB &= ~(1<<PB0); // Make sure we're in the input mode
  PORTB |= (1 << PB0); // initialize pull-up resistor on our input pin

  DDRB |= (1<<PB1); // Set PB1 an output

  while (1)
  {
    if ((PINB & (1 << PB0)) == 0 ) // If PD2 is pressed
    {
        PORTB |= (1<<PB1); // Set PB1 to high
    }
    else { /* not pressed */
        PORTB &= ~(1<<PB1); // Set PB1 to low
    }
  }
          return 0;
}
```

Naive button circuit Interface

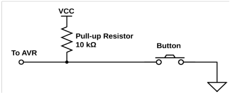
What's wrong with this approach?



- When the button is released (or open the switch), what is the voltage at the AVR end of the switch?
- Short answer: nobody knows. A wire that's just dangling in the air, on the unconnected side of a switch, can act as an antenna. The voltage on that wire will wiggle around between high and low logic states at whatever frequency the strongest local radio stations (or even "noisy" electrical appliances) broadcast.
- The only thing you do know about this voltage is that it's unreliable.

Solution – Use a pull-up resistor

- A pullup resistor keeps the voltage at a digital input pin as close as possible to a high state.
- A pull-up resistor is a resistor of a relatively high value that "pulls up" the no ground side of the button to the high voltage level in the absence of a button press.
- Think of pull-up (or pull-down) resistors as setting a default electrical value for the button when it's



The need for a pull-up resistor

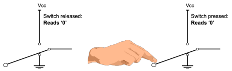


FIG 23. When there is no pull-up, the state on the pin is not known when the button is not pressed

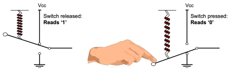


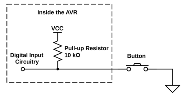
FIG 24. With a pull-up resistor, the input pin will read a high state when the button is not pressed.

[1] For switch and resistive sensor applications, the typical pull-up resistor value is 1-10 kΩ. If in doubt, a good starting point when using a switch is 4.7 kΩ. Some digital circuits, such as CMOS families, have a small input leakage current, allowing much higher resistance values, from around 10kΩ up to 1MΩ. The following links shows how such a resistor might be calculated https://learn.sparkfun.com/tutorials/pull-up-resistors/all

[2] Many GPIO pins on an AVR MCU have internal pull-up resistors. Their value of the pull up resistor are specified in the MCU's datasheet. Typical is they range from 20-50K, IIRC. A common, safe maximum value will be 4.7KΩ, maybe as high as 10kΩ

AVR Internal pull-up resistor

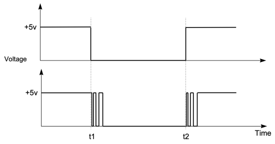
- The AVR chips provide a built-in pull-up resistor for each pin, and you have the option to activate it for any pin that is in input mode.
- That way the AVR side reads a well-defined high voltage level (pulled up by the internal resistor) until you press the button, when the lower-resistance connection to ground through the button dominates, and the pin reads 0 V.
- To enable a pull-up, you need to set to HIGH the output pin hardware register. That is, if you'd



Switch bouncing

Dealing with switch bounce

In practice, all mechanical switch contacts bounce (that is, turn on and off, repeatedly, for a short period of time) after the switch is closed or opened[1],[2]



[1] https://www.eejournal.com/article/ultimate-guide-to-switch-debounce-part-1/
[2] https://circuitcellar.com/research-design-hub/design-solutions/how-to-eliminate-switch-bounce/

Dealing with switch bounce

Without appropriate software design, this can give rise to a number of problems, not least:

- Rather than reading "A" from a keypad, we may read "AAAAA"
- Counting the number of times that a switch is pressed becomes extremely difficult.
- If a switch is depressed once, and then released some time later, the "bounce" may make it appear as if the switch has

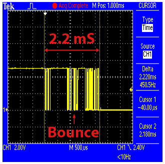


FIG 25. Example of a push-button switch being pressed, as seen on the oscilloscope:

Switch debouncing- Solution

Debouncing switch input is, in essence, straightforward[3]:

- We read the relevant port pin.
- If we think we have detected a switch depression, we read the pin again (say) 20 ms later.
- If the second reading confirms the first reading, then we assume the switch really has been depressed.

Notes

- The 20 ms delay depends on the switch used: the data sheet of the switch will provide this information. If you have no data sheet, you can either experiment with different figures or measure directly using an oscilloscope.
- There exists hardware methods to switch debouncing. However, they involve more circuitry (thus, cost) and lead to increase in power consumption. They should be avoided if possible.

[3] https://www.embedded.com/my-favorite-software-debouncers/

Liquid Crystal Display (LCD)

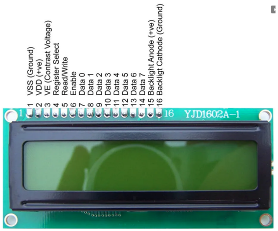


FIG 26. A 16×2 LCD based on HD44780

LCD pin descriptions

- VEE —used for controlling LCD contrast.
- RS —register select
  - RS = 0 to select command register (e.g., clear screen)
  - RS = 1 to select data register (e.g., write something on the LCD)
- R/W, read/write
  - R/W input allows to write information to the LCD or read information from it.
  - R/W = 1 when reading; R/W = 0 when writing.
- E enable—Used to enable the display
  - When $E = 0$, the LCD ignores R/W, RS, and data bus lines operations
  - When $E = 1$, the LCD processes the incoming data.
- D0–D7—8-bit data pins used to send information to the LCD or read the contents of the LCD's internal registers.

NOTE: Please read the LCD datashet[4]. More info can also be found online[5]

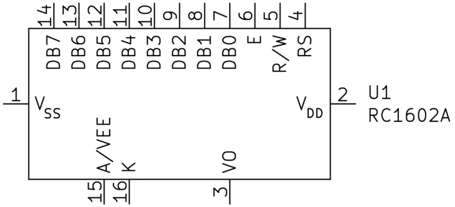
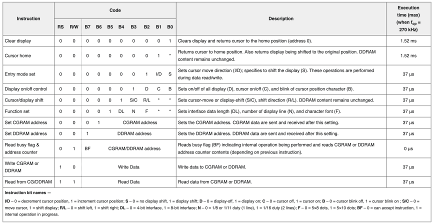


FIG 27. Schematic symbol for a 16×2-character LCD

TAB 1. HD44780U based LCD commands set[6]



Instruction	Code										Description	Execution time (max) (when f_{cp} = 270 kHz)
	RS	R/W	B7	B6	B5	B4	B3	B2	B1	B0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears display and returns cursor to the home position (address 0).	1.52 ms
Cursor home	0	0	0	0	0	0	0	0	1	*	Returns cursor to home position. Also returns display being shifted to the original position. DDRAM content remains unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction (I/D); specifies to shift the display (S). These operations are performed during data read/write.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets on/off of all display (D), cursor on/off (C), and blink of cursor position character (B).	37 µs
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	*	*	Sets cursor-move or display-shift (S/C), shift direction (R/L). DDRAM content remains unchanged.	37 µs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display line (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	CGRAM address						Sets the CGRAM address. CGRAM data are sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	DDRAM address							Sets the DDRAM address. DDRAM data are sent and received after this setting.	37 µs
Read busy flag & address counter	0	1	BF	CGRAM/DDRAM address							Reads busy flag (BF) indicating internal operation being performed and reads CGRAM or DDRAM address counter contents (depending on previous instruction).	0 µs
Write CGRAM or DDRAM	1	0	Write Data								Write data to CGRAM or DDRAM.	37 µs
Read from CG/DDRAM	1	1	Read Data								Read data from CGRAM or DDRAM.	37 µs

Instruction bit names —

I/D – 0 = decrement cursor position, 1 = increment cursor position; **S** – 0 = no display shift, 1 = display shift; **D** – 0 = display-off, 1 = display on; **C** – 0 = cursor off, 1 = cursor on; **B** – 0 = cursor blink off, 1 = cursor blink on ; **S/C** – 0 = move cursor, 1 = shift display; **R/L** – 0 = shift left, 1 = shift right; **DL** – 0 = 4-bit interface, 1 = 8-bit interface; **N** – 0 = 1/8 or 1/11 duty (1 line), 1 = 1/16 duty (2 lines); **F** – 0 = 5×8 dots, 1 = 5×10 dots; **BF** – 0 = can accept instruction, 1 = internal operation in progress.

[6] https://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller

LCD connection to an MCU

An LCD can be interfaced either with an 8-bit mode or in a 4-bit mode

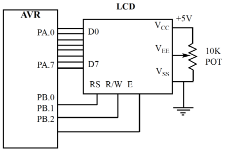


FIG 28. LCD Connections for 8-bit Data

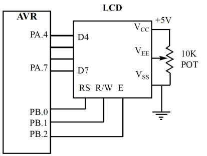


FIG 29. LCD Connections Using 4-bit Data

LCD Connections Using 4-bit Data

- In 4-bit mode, data/command is sent in a 4-bit (nibble) format.
- In this case, first a higher 4-bit is sent then the lower 4-bit of data/command is sent
- Only 4 data pins of 16x2 of LCD are connected to the microcontroller and other control pins RS (Register select), RW (Read/write), E (Enable) is connected to other GPIO Pins of the

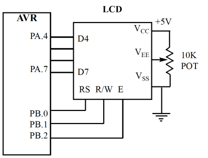


FIG 30. LCD Connections Using 4-bit Data

Sending commands and data to LCDs

To send data and commands to LCDs you should do the following steps:[7]

- **Step 1** —Initialize the LCD.
- **Step 2** —Send any of the commands from Table 1 to the LCD.
- **Step 3** —Send the character to be shown on the LCD.

	Code										
Instruction	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.

FIG 31. A numerical description of the LCD's "clear screen" command

[7]Notice that steps 2 and 3 can be repeated many times

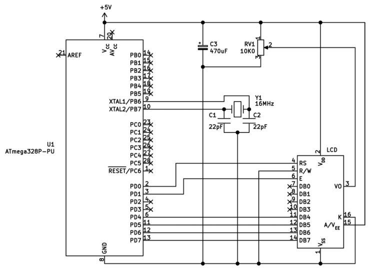


FIG 32. LCD connection to an ATMega328 using a 4-bit data configuration

```
void LCD_send_command(uint8_t _command )   {
  /*sends upper nibble, then lowert nibble to LCD*/
  PORTD = (PORTD & 0x0F) | (_command & 0xF0);
  /*RS off - selects instruction register (not data
     register)*/
  PORTD &= ~(1<<PIN_RS);
   /*Enable E_PIN  and wait for the changes to take
      effect */
  PORTD |= (1<<PIN_EN);
  _delay_us(1);
  /*Disable the enable pin*/
  PORTD &= ~(1<<PIN_EN);  _delay_us(200);
  /*sends lower nibble of command byte*/
  PORTD = (PORTD & 0x0F) | (_command << 4);
    /*Enable E_PIN  and wait for the changes to take
       effect */
  PORTD |= (1<<PIN_EN); _delay_us(1);
    /*Disable the  enable pin*/
```

```
void LCD_init(void)
{
  DDRD = 0b11111111;
  LCD_command(0x02);  // return cursor home
/*use 4-bit control via PD7~PD4, 2 line LCD, font 0
   (5x8 pixel)*/
  LCD_command(0x28);
/*display on, block cursor off, blinking cursor off
   */
  LCD_command(0x0C);
/*tells cursor to move in incremental stages*/
  LCD_command(0x06);
/*clear the LCD and wait a bit*/
  LCD_command(0x01);  _delay_ms(2);
}
```

Listing 2: Initializing the LCD

- The course website has a complete implementation the code