AVR I/O Ports

1. INTRODUCTION

- The AVR I/O ports are **the** path to the outside world. Understand how to use them and life is good. Failure to understand how the ports are used **will cause grief and possibly cost $'s.**
- An abused I/O port is fairly easy to burn out with excessive current or static damage: Almost all the I/O ports are floating inputs that can build up large static charge

2. AVR PORT REGISTERS

The Atmel ATmega is equipped with four 8-bit general-purpose, digital I/O ports designated PORTA, PORTB, PORTC, and PORTD. All of these ports also have alternate functions, which will be described later. In this section, we concentrate on the basic digital I/O port features.

As shown in Figure 1.1, each port has three registers associated with it:

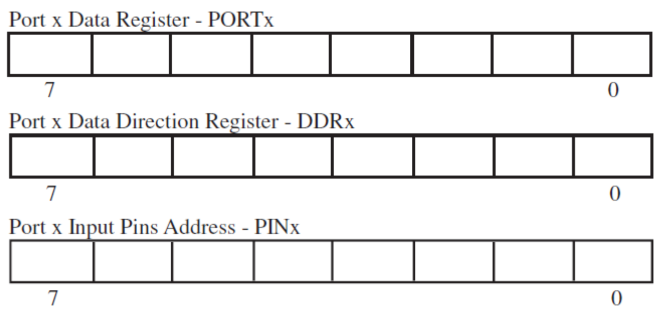

Port x Data Register - PORTx
7
0
Port x Data Direction Register - DDRx
7
0
Port x Input Pins Address - PINx
7
0

Fig 1.1 (a) port-associated registers

DDxn	PORTxn	I/O	Comment
0	0	input	Tri-state (Hi-Z)
0	1	input	source current if externally pulled low
1	0	output	Output Low (Sink)
1	1	output	Output High (Source)

x: port designator (A, B, C, D)
n: pin designator (0 - 7)

Fig 1.1 (b) port pin configuration.

All Atmega Port in general have:

- bit-selectable pull-up resistors
- bit-selectable tri-state outputs
- schmitt trigger input buffers
- synchronized to the system clock to prevent metastability
- symmetrical DC drive capability
- All ports have read-modify-write capability, i.e., you can change pin *direction*, pin *value*, or pin *pull-up resistor* without effecting any other pins in the port

Control of all ports and pins is done with three registers

- DDRx (i.e., DDRB is *data direction register* port B)
- PORTx (i.e. PORTB is the *output register* for port B)
- PINx (i.e. PINB in the *input register* for port B)

All of these ports may be read. Writing the PINx register does nothing[1].

Figure 1.1(b) describes the settings required to configure a specific port pin to either input or output.

- If selected for input, the pin may be selected for either an input pin or to operate in the high-impedance (Hi-Z) mode. In Hi-Z mode, the input appears as high impedance to a particular pin.
- If selected for output, the pin may be further configured for either logic low or logic high.

Port pins are usually configured at the beginning of a program for either input or output, and their initial values are then set. Usually, all eight pins for a given port are configured simultaneously.

A code example is provided in the code listing 1.1. below to show how ports are configured. Note that because we are using the C programming language with a compiler include file, the register contents are simply referred to by name. Note that the data direction register (DDRx) is first used to set the pins as either input or output, and then the data register (PORTx) is used to set the initial value of the output port pins

To read the value from a port pin configured as input, the following code could be used. Note the variable used to read the value from the input pins is declared as an unsigned char because both the port and this variable type are 8 bits wide.

```
unsigned char new_PORTB; //new values of PORTB

new_PORTB = PINB; //read PORTB
```

[1] High-end Atmega chips are exceptional and writing to PINx has some other implication.

```
//*****************************************************************
//initialize_ports: provides initial configuration for I/O ports
//*****************************************************************
void initialize_ports(void)
{
       DDRA=0xfc; //set PORTA[7:2] as output, PORTA[1:0]
       //as input (1111_1100)
       PORTA=0x03; //initialize PORTA[7:2] low, PORTA[1:0]
       //current source
       DDRB=0xa0; //PORTB[7:4] as output, set PORTB[3:0] as input
       PORTB=0x00; //disable PORTB pull-up resistors
       DDRC=0xff; //set PORTC as output
       PORTC=0x00; //initialize low
       DDRD=0xff; //set PORTD as output
       PORTD=0x00; //initialize low
}
```

Code Listing 1.1. Port Configuration Example

3. AVR PORT STRUCTURE

As shown in figure 1.2, the data direction DDRx register allows the data to go from PORTx of the pin either as an input or an output.

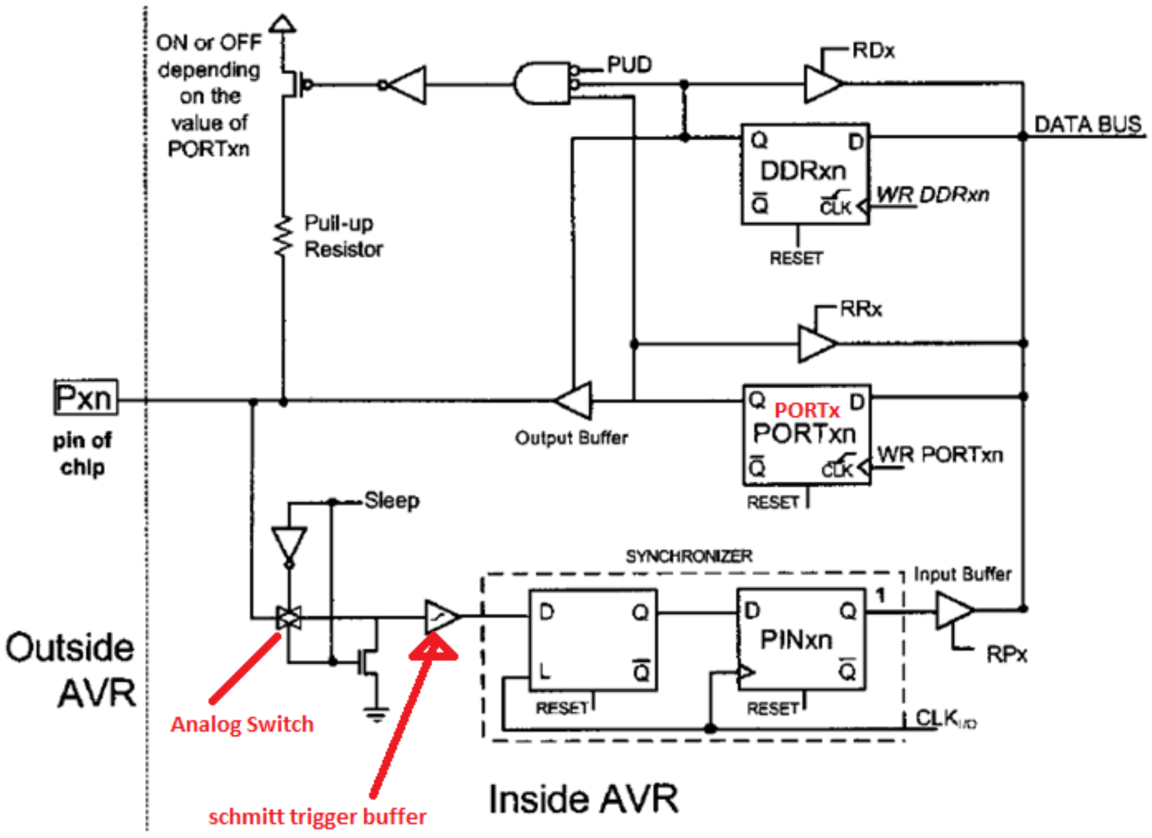


Figure 1.2. The AVR Port Structure

3.1. PORTx Register

As shown in figure 1.2, the PORTx register can be accessed using read and write operations.

- When one wants to write a value Rx to PORTx, he/she must set the port to an output first (use the OUT PORTx, Rx instruction in assembly). When this is done, the WR-PORTx pin is set high and the value, Rx is loaded into PORTx.
- When one wants to read from PORTx, he/she must configure the PORT as an input first (use IN Rd, PORTx instruction in assembly). In this case, the PRx pin is set to HIGH, which enables the buffe and makes it possible to read from PORTx.

3.2. The DDRx Register

In figure 1.2, the DDRx registerr can be accessed using read and write operations.

- When one wants to write to DDRx, he/she must use the "OUT DDRx, Rx" instruction. In this case, the WR-DDRx pin is set to HIGH and enables writing to DDRx.
- When one wants to read from DDRx, he/she must use the "IN Rd, DDRx" instruction. In this case, the RDx pin is set to LOW, which enables the buffe and makes it possible to read from DDRx.

The DDRx register controls the output buffer and the pull-up resistor:

- When the Q of DDRx is HIGH, it enables the output buffe and connects the Q of the PORTx register to Px pin of the chip. In this case, the pin is configured as an output.
- When Q of DDx is LOW, it disables the output buffe and configures the Px pin of the chip as an input. In this case, assuming that the PUD bit is LOW, the Q of PORTx controls the pull-up resistor.
- When the Q of PORTx is HIGH, it enables the pull-up resistor, and when it is LOW it disables the pull-up resistor.

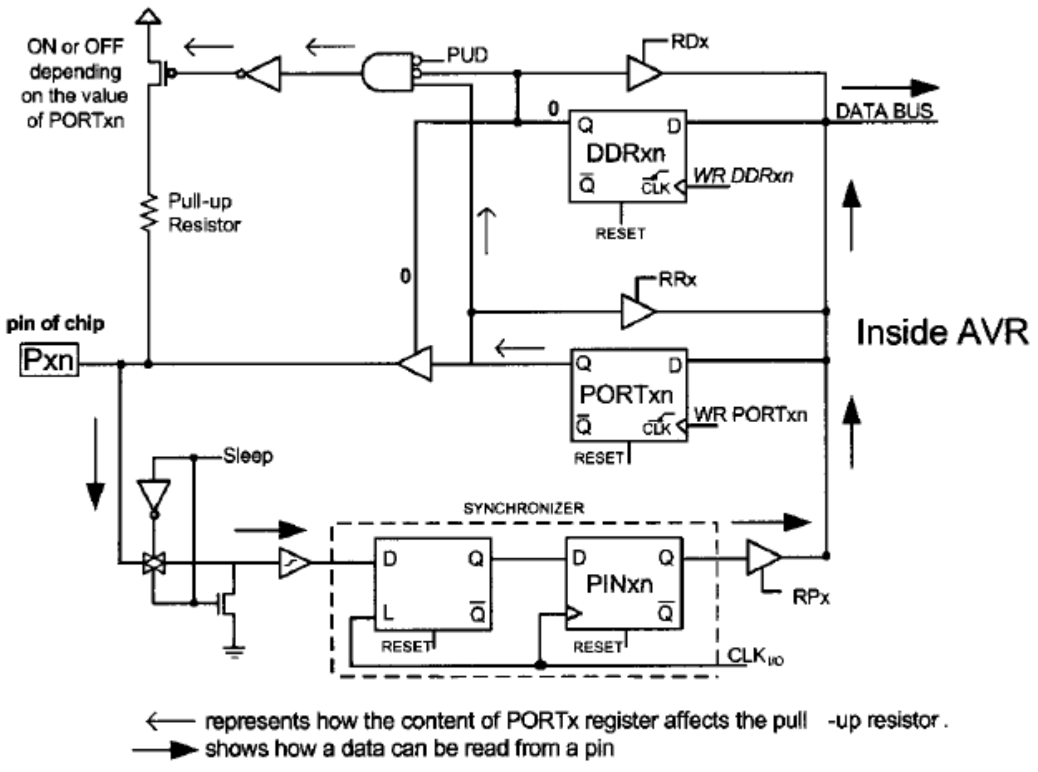
3.3. PINx Register

As shown in figure 1.2, when an AVR is not in sleep mode (we'll talk about this toward the end of the course), the PINxn flip-flop is loaded with the value of the AVR pin of the chip; thus, it is possible to read the content of the PINx register. To do so, one must use "IN Rd, PINx" instruction, which sets PRx high and enables the input buffer. In this case, the value of PINx passes through the internal data bus of AVR and will be loaded into the Rd Register.

3.3.1. Reading the pin when DDRx.n=LOW (is an input)

In our previous discussion, it was noted that to make any bits of any port of the AVR an input port, one must first write a 0 (logic LOW) to the DDRx.n BIT. This is due to the following:

- As shown in figure 1.3, if one write a 0 to the DDRx.n, it will have a LOW on its Q. This turns off the tri-state buffer.
- When the tri-state buffer is off it blocks the path from Q of PORTx.n to the pin of the chip, and the input signal is directed to the PINx.n buffer
- When reading the input port in instruction such as "IN R16, PINB", one is reading the data present in the pin. In other words, it is bringing into the CPU the status of the external pin. This instruction activates the read pin of the buffer and lets data at the pins flow into the CPU's internal bus.



ON or OFF depending on the value of PORTxn
PUD
RDx
DATA BUS
DDRxn
WR DDRxn
RESET
Pull-up Resistor
RRx
pin of chip
Pxn
Inside AVR
PORTxn
WR PORTxn
Sleep
SYNCHRONIZER
PINxn
RPx
CLK
represents how the content of PORTx register affects the pull -up resistor.
shows how a data can be read from a pin

Figure 1.3 Reading from a pin via PINx Register

3.3.2. Writing to pin when DDRx.n=HIGH (Output)

In section 3.3.1, it was showed why one must write a LOW to a port's DDRx.n bits in order to make it an input port. However, what would happen if one writes a HIGH to a DDRx.n that was configured as an input port? Taking a close look at figure 1.4 below, it is noticeable that when DDRx.n=HIGH, the DDRx.n latch has "HIGH" on its Q. This turns on the tri-state buffer, and the data of PORTx.n is transferred to the pin chip.

From figure 1.4, when DDRx.n=1, if one writes a 0 to the PORTx.n latch, then PORTx.n has "LOW" on its Q. This provides 0 to the pin of the chip. Therefore, any attempt to read the input pin will always get the "LOW" ground signal.

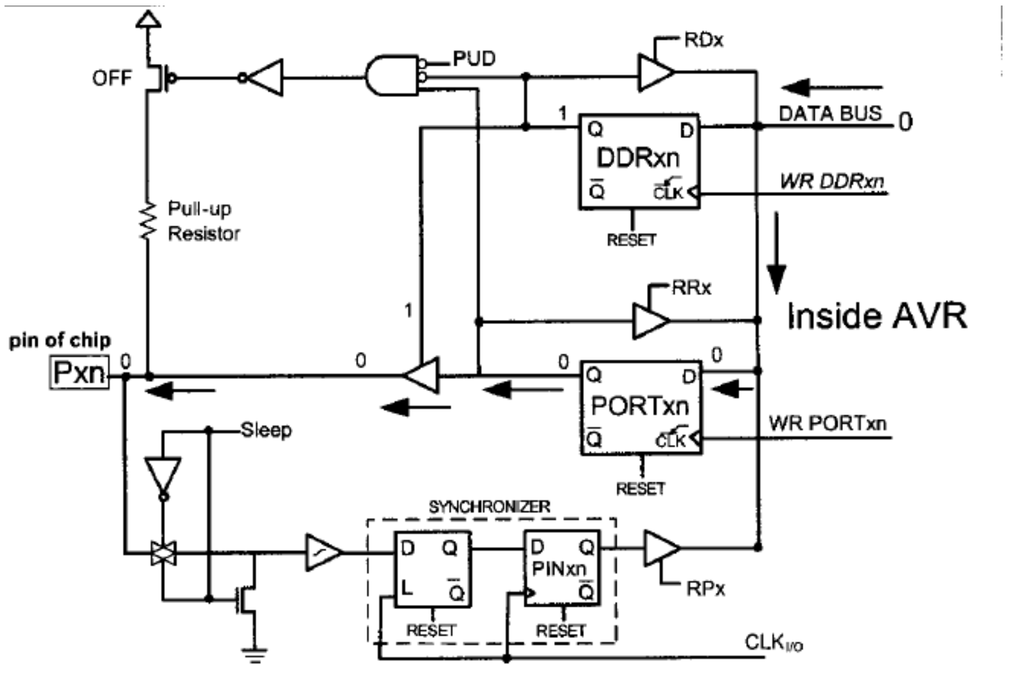


Figure 1.4. Writing a 0 to Pin in of the AVR

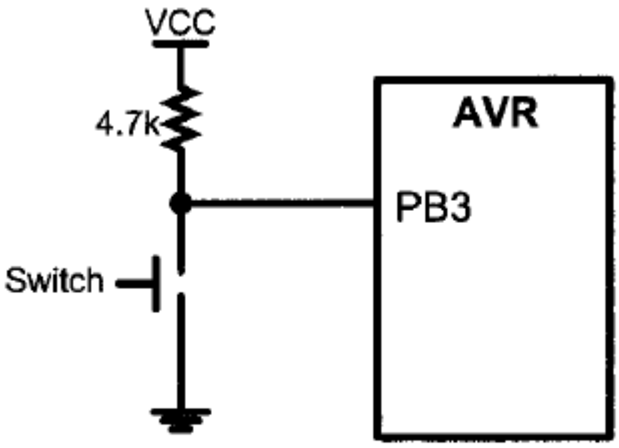
Figure 1.5 below, shows what happens if one writes a "1" to PORTx.n when DDRx.n=1. Writing 1 to the PORTx.n makes Q=1. As a result, a 1 is provided to the pin of the chip. Therefore, any attempt to read the input pin will always get the HIGH signal.

SAFETY NOTE:

One should not make an I/O port output while it is externally connected to a voltage; otherwise, he/she might damage the port. For example, in figure

```
// Set PB3 an output
DDRB |= (1<<PB3);
// Make PB3 High
PORTB |= (1 << PB3);
```

This code fragment may potentially damage the I/O port since PB3 is connected to a switch, it should note be configured as an output.



4. DIGITAL LOGIC AND OPEN COLLECTOR

Normal digital logic has two states: high and low. There are four currents of interest, as shown in Figure 1.5, when analyzing whether the inputs of the next stage are loading the output. IIH and IIL are the currents required of an input when high and low, respectively.

Similarly, I_{OH} and I_{OL} are the maximum currents available at the output when high and low. In order for the output to properly drive all the inputs of the next stage, the maximum available output current must be larger than the sum of all the required input currents for both the high and low conditions.

$$|I_{OL}| \geq \sum |I_{IL}| \quad \text{and} \quad |I_{OH}| \geq \sum |I_{IH}|$$

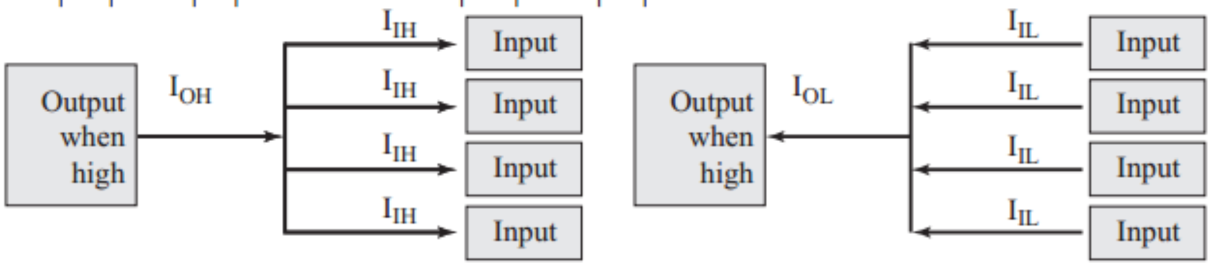


Figure 1.5. One output driving multiple inputs.

When we design circuits using devices all from a single logic family, we can define fan out as the maximum number of inputs one output can drive. For transistor-transistor logic (TTL) logic we can calculate fan out from the input and output currents:

$$\text{fan out} = \text{minimum}\ ((I_{OH}/I_{IH}), (I_{OL}/I_{IL}))$$

Note however that the fan out of high speed complementary metal-oxide semiconductor (CMOS) devices is determined by capacitive loading and not by the currents.

Figure 1.6 shows a simple model of a CMOS interface. The ideal voltage of the output device is labeled V1. For interfaces in close proximity, the resistance R results from the output impedance of the output device, and the capacitance C results from the input capacitance of the input device. However, if the interface requires a cable to connect the two devices, both the resistance and capacitance will be increased by the cable. The voltage labeled V2 is the effective voltage as seen by the input. The slew rate of a signal is the slope of the voltage versus time during the time when the logic level switches between low and high. A similar parameter is the transition time, which is the time it takes for an output to switch from one logic level to another. In Figure 1.13, the transition time is defined as the time it takes V2 to go from 1.75 to 3.25 V. There is a capacitive load for each CMOS input connected to a CMOS output. As this capacitance increases, the slew rate decreases, which will increase the transition time. The time constant of this simple circuit is τ = R*C.

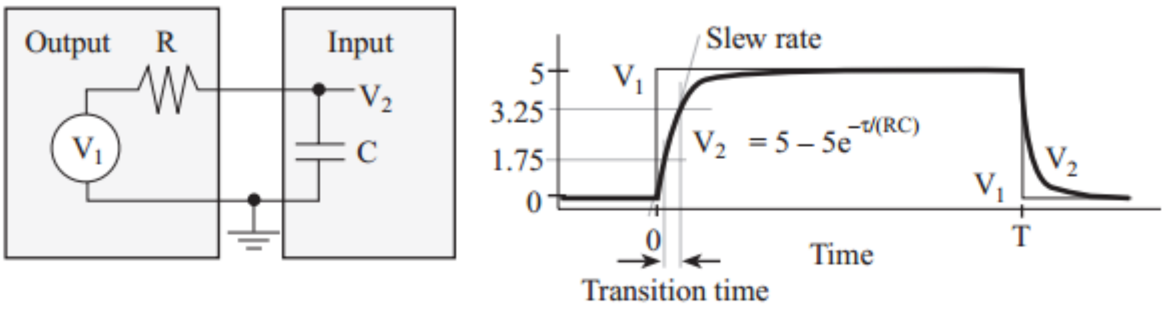


Figure 1.6 Capacitance loading is an important factor when interfacing CMOS devices.

Let T be the pulse width of the digital signal. If T is large compared to τ, then the CMOS interface functions properly. For circuits that mix devices from one family with another, we must look individually at the input and output currents, voltages, and capacitive loads. Table 1 shows typical current values for the various digital logic families. The MC9S12C32 allows full drive (a low R providing 10 mA) and reduced drive (a larger R providing a smaller 2 mA) modes.

Family	Example	I_{OH}	I_{OL}	I_{IH}	I_{IL}	Fan Out
Standard TTL	7404	0.4 mA	16 mA	40 µA	1.6 mA	10
Schottky TTL	74S04	1 mA	20 mA	50 µA	2 mA	10
Low-power Schottky TTL	74LS04	0.4 mA	4 mA	20 µA	0.4 mA	10
High-speed CMOS	74HC04	4 mA	4 mA	1 µA	1 µA	
Freescale microcomputer	MC68HC11E	0.8 mA	1.6 mA	1 µA	1 µA	
Freescale microcomputer	MC9S12C32	10 mA	10 mA	1 µA	1 µA	
Intel microcomputer	87C51 P0	7 mA	3.2 mA	10 µA	10 µA	
	87C51 P1,P2,P3	60 µA	1.6 mA		50 µA	

Table 1 The input and output currents of various digital logic families and microcomputers.

Figure 1.7 compares the input and output voltages for many of the digital logic families. V_{IL} is the voltage below which an input is considered a logic low. Similarly, V_{IH} is the voltage above which an input is considered a logic high. V_{OH} is the output voltage when the signal is high. In particular, if the output is a logic high, and the current is less than I_{OH}, then the voltage will be greater than V_{OH}. Similarly, V_{OL} is the output voltage when the signal is low. In particular, if the output is a logic low and the current is less than I_{OL}, then the voltage will be less than V_{OL}.

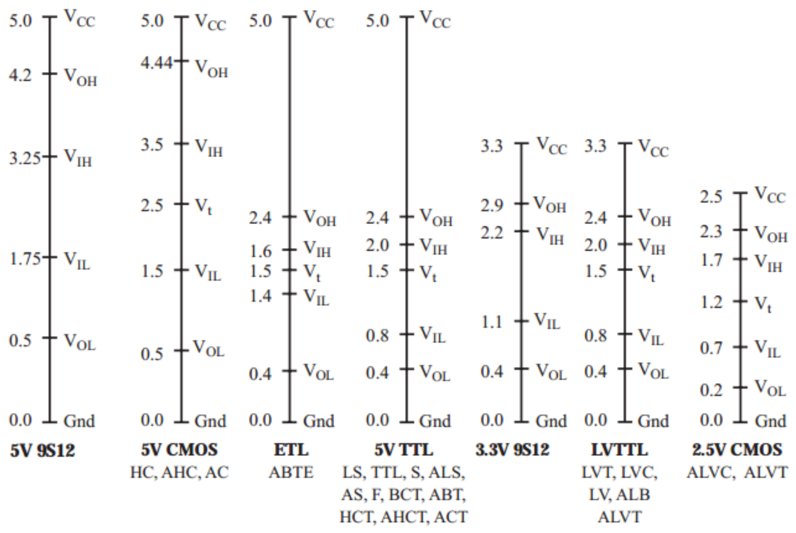


Figure 1.8. Voltage thresholds for various digital logic families.

The maximum output current specification on the Atmeaga128 is 40 mA, which is the current above which it will cause damage. Normally, we design the system so the output currents are less than I_{OH}

and I_{OL}. V_t is the typical threshold voltage, which is the voltage at which the input usually switches between logic low and high. Formally however, an input is considered as indeterminate for voltages between VIL and VIH. The five parameters that affect our choice of logic families are:

- Power supply voltage (e.g., 5V, 3.3 V etc.)
- Power supply current (e.g., will the system need to run on batteries?)
- Speed (e.g., clock frequency and propagation delays)
- Output drive, I_{OL}, I_{OH} (e.g., does it need to drive motors or lights?)
- Noise immunity (e.g., electromagnetic field interference)
- Working Temperature (e.g., 0 to 70 C)