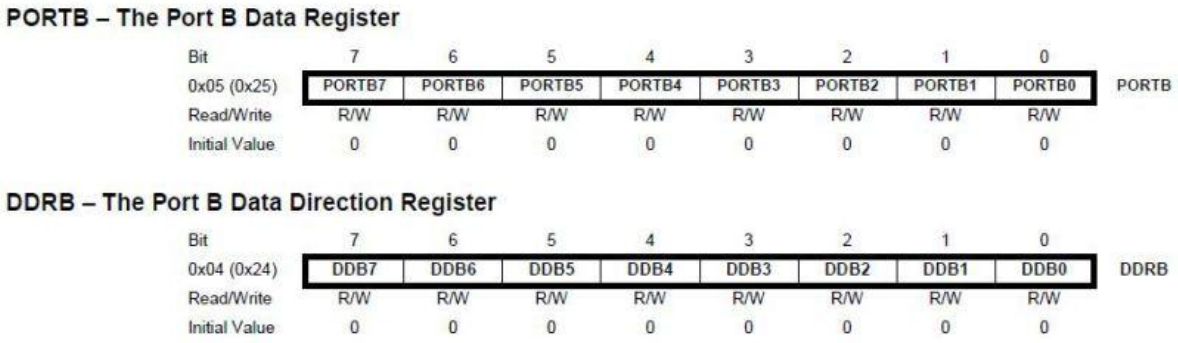
AVR: Configuring and Using Ports in C

There comes a time, when you want turn something on using a pin from your AVR chip. Today's tutorial discusses how to configure any port on any AVR microcontroller. In order to use the digital IO of any AVR chip, you must tell the chip which pins are output and input pins. In order to do this, you must first configure DDRn register, where n stands of the PORT you're configuring (e.g. DDRB configures PORTB). The number of DDRn registers and the bit length of the DDRn register depends on the number of PORTs and the bit length of each PORT on chip you're using. For example, the ATtiny85 only has one 6 bit long PORTB while an Atmega 328 chip has two 8 bit long ports (PORTB and PORTD) and one 6 bit long port(PORTC). Thus,the ATtiny has one DDRB register, while the Atmega328 has one DDRB register, one DDRD register, and one DDRC register. Fig.1 shows the DDRB and PORTB register for the ATmega328.



PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 1: DDRB and PORTB register from ATmega328 datasheet

For the rest of the tutorial, we'll be using PORTB and DDRB. Now, how do you tell the microcontroller which pins are inputs and outputs using DDRB? Let's say we want to only configure PB0 and PB2 as output pins. To make PB0 and PB2 output pins, you must write a bit to both DDRB0 and DDRB2 in DDRB. Thus we want to write 5 to the DDRB register (2^0+2^2=5=0×05). Here's a C code example of configuring the DDRB register.

```
DDRB=0x05; // Configures PB0 and PB2 as output pins
```

Keep in mind, we just configure PB0 and PB2 as output pins, the rest of the pins are configured to be input pins. Now, to turn on PB0 then PB2, look at the following C code.

```
PORTB=0x01; // Turns on just PB0
PORTB=0x04; // Turns on just PB2
```

```
PORTB=0x05; // Turns on PB0 and PB2
```

The previous code is overriding all of the bits in the PORTB register to make PB0 and PB2 turn on. In most projects, you only want to configure the pins you are using and preserve the states of the rest of the pins. To preserve the states of the previous bits while keeping the rest of the bits intact, we want to use bit masking. To do bitmasking, we simply use and/or logic operations.

```
PORTB=0x00; // Making sure 0 is in all pins of PORTB
PORTB|=0x05; // Turn on just PB0 and PB2
PORTB&=~0x05; // Turns off just PB0 and PB2 by taking the logical and of the complement of 5
```

Source: http://coolcapengineer.wordpress.com/2012/07/31/avr-configuring-and-using-ports-in-c/