**Tutorial**

**Overview**

You cannot imagine to use microcontroller without using any of its i/o pins. Finally its all about  : taking input , processing it and generating output ! Thus i/o registers and their correct settings is indispensable part while learning to program any uC.

We will learn how to use AVR ports and actually ‘code’  for writing/reading data to/from port pins. It is slightly confusing for beginners, however once you understand it, you will certainly appreciate the way it is designed.

NOTE : I will frequently refer to  ‘configuring pin’ or simply ‘pin’. Remember, a port has multiple pins. Thus in order to change setting for one port, you have to change setting for all port pins of that port. To change setting for one single pin of the port, you have to change a particular bit in associated register. Got that ? If not read this para again.

**Registers**

AVR is 8 bit microcontroller. All its ports are 8 bit wide. Every port has 3 registers associated with it each one with 8 bits. Every bit in those registers configure pins of particular port. Bit0 of these registers is associated with Pin0 of the port, Bit1 of these registers is associated with Pin1 of the port, …. and like wise for other bits.

These three registers are as follows :   
(x can be replaced by A,B,C,D as per the AVR you are using)  
- DDRx register   
- PORTx register  
- PINx register

**DDRx register**

DDRx (Data Direction Register) configures data direction of port pins. Means its setting determines whether port pins will be used for input or output. Writing 0 to a bit in DDRx makes corresponding port pin as input, while writing 1 to a bit in DDRx makes corresponding port pin as output.

example:

* to make all pins of port A as input pins :  
  DDRA = 0b00000000;
* to make all pins of port A as output pins :  
  DDRA = 0b11111111;
* to make lower [**nibble**](http://wiki.answers.com/Q/How_many_nibbles_are_there_in_a_byte) of port B as output and higher [**nibble**](http://wiki.answers.com/Q/How_many_nibbles_are_there_in_a_byte) as input :  
  DDRB = 0b00001111;

**PINx register**

PINx (Port IN) used to read data from port pins. In order to read the data from port pin, first you have to change port’s data direction to input. This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give you data that has been output on port pins.

Now there are two input modes. Either you can use port pins as tri stated inputs or you can activate internal pull up. It will be explained shortly.

example :

* to read data from port A.
* DDRA = 0x00; //Set port a as input
* x = PINA; //Read contents of port a

**PORTx register**

PORTx is used for two purposes.

1) To output data  :  when port is configured as output

When you set bits in DDRx to 1, corresponding pins becomes output pins. Now you can write data into respective bits in PORTx register. This will immediately change state of output pins according to data you have written.

In other words to output data on to port pins, you have to write it into PORTx register. However do not forget to set data direction as output.

example :

* to output 0xFF data on port b
* DDRB = 0b11111111; //set all pins of port b as outputs
* PORTB = 0xFF; //write data on port
* to output data in variable x on port a
* DDRA = 0xFF; //make port a as output
* PORTA = x;                //output variable on port
* to output data on only 0th bit of port c
* DDRC.0 = 1;        //set only 0th pin of port c as output
* PORTC.0 = 1; //make it high.

2) To activate/deactivate pull up resistors – when port is configures as input

When you set bits in DDRx to 0, i.e. make port pins as inputs, then corresponding bits in PORTx register are used to activate/deactivate pull-up registers associated with that pin. In order to activate pull-up resister, set bit in PORTx to 1, and to deactivate (i.e to make port pin tri stated) set it to 0.

In input mode, when pull-up is enabled, default state of pin becomes ’1′. So even if you don’t connect anything to pin and if you try to read it, it will read as 1. Now, when you externally drive that pin to zero(i.e. connect to ground / or pull-down), only then it will be read as 0.

However, if you configure pin as tri state. Then pin goes into state of high impedance. We can say, it is now simply connected to input of some OpAmp inside the uC and no other circuit is driving it from uC. Thus pin has very high impedance. In this case, if pin is left floating (i.e. kept unconnected) then even small static charge present on surrounding objects can change logic state of pin. If you try to read corresponding bit in pin register, its state cannot be predicted. This may cause your program to go haywire, if it depends on input from that particular pin.

Thus while, taking inputs from pins / using micro-switches to take input, always enable pull-up resistors on input pins.

NOTE : while using on chip ADC, ADC port pins must be configured as tri stated input.

example :

to make port a as input with pull-ups enabled and read data from port a

DDRA = 0x00; //make port a as input

PORTA = 0xFF; //enable all pull-ups

y = PINA; //read data from port a pins

to make port b as tri stated input

DDRB = 0x00; //make port b as input

PORTB = 0x00; //disable pull-ups and make it tri state

to make lower nibble of port a as output, higher nibble as input with pull-ups enabled

DDRA = 0x0F; //lower nib> output, higher nib> input

PORTA = 0xF0; //lower nib> set output pins to 0,

//higher nib> enable pull-ups

**Summery**

Following table lists register bit settings and resulting function of port pins

|  |  |  |  |
| --- | --- | --- | --- |
| **register bits → pin function↓** | **DDRx.n** | **PORTx.n** | **PINx.n** |
| tri stated input | 0 | 0 | read data bit x = PINx.n; |
| pull-up input | 0 | 1 | read data bit x = PINx.n; |
| Output | 1 | write data bit PORTx.n = x; | n/a |

(This data has been taken from [link](http://elecrom.wordpress.com/2008/02/12/avr-tutorial-2-avr-input-output/))