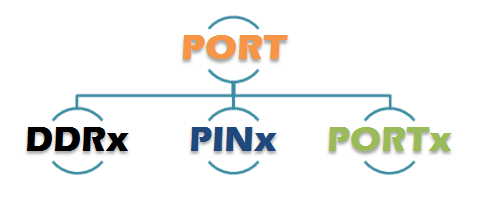
Posted by [Mayank](http://maxembedded.com/author/maxmiaggi/) on Jun 10, 2011 in [Atmel AVR](http://maxembedded.com/category/microcontrollers-2/atmel-avr/), [Microcontrollers](http://maxembedded.com/category/microcontrollers-2/) | [168 comments](http://maxembedded.com/2011/06/port-operations-in-avr/#respond)

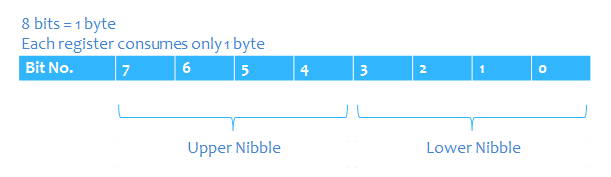


**I/O Port Operations in AVR**

[](http://maxembedded.files.wordpress.com/2011/06/avr_series.png?resize=140%2C105)Hello friends! In this post, we will discuss about the port operations in AVR. Before going further, I suggest that you read my previous post regarding [AVR Basics](http://maxembedded.wordpress.com/2011/06/09/avr-basics/). **The examples discussed here are in accordance with ATMEGA16/32 MCU. However, the concepts are equally good for any AVR MCU**.

**Register**

Okay, now I hope you are familiar with the term register. If not, then you must have heard of it. Basically, a [processor register](http://en.wikipedia.org/wiki/Processor_register) is a memory space within the CPU itself so that they can be accessed very frequently and fast. These registers are linked with the operation of the MCU. Let’s consider the following memory space.

[](http://maxembedded.files.wordpress.com/2011/06/register.png?resize=611%2C177)

Register Memory Space

Here, you can see that I have represented 8 bits together to form a memory of 1 byte. Note the sequence in which the bits are numbered. They are numbered as 7, 6, 5, … , 1, 0. This is because the bits are numbered from the **Least Significant Bit (LSB)** to the **Most Significant Bit (MSB)**. From the knowledge of digital logic, we know that the last bit is the LSB whereas the first bit is the MSB. Hence, **Bit 0 = LSB** and **Bit 7 = MSB**.

**Register Concept Explained**

Let me explain you why LSB is the last bit. Let’s take an example. Please note that 0b stands for binary and 0x stands for hexadecimal. If nothing is prefixed, it means that it is in decimal number system.

A = 0b 0110 0111 = 103

Now here, let’s change the value of the last bit (orange colour bit) from 1 to 0. This makes

B = 0b 0110 0110 = 102

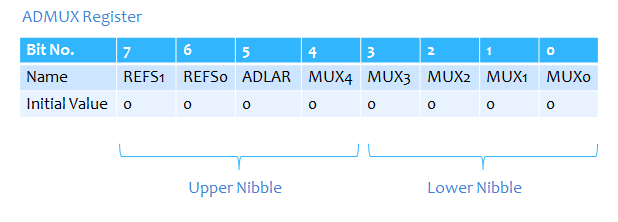
Now, once again in A, let’s change the first bit (magenta colour bit) from 0 to 1. This makes

C = 0b 1110 0111 = 231

We see that by changing the last bit, the result (B) is very close to the original data (A), whereas by changing the first bit, the result (C) varies quite significantly from the original data (A). Hence, the last bit is the LSB (as the data doesn’t change significantly) whereas the first bit is the MSB (as the data changes significantly).

Now, we also know that 1 nibble = 4 bits. Hence, bits 0,1,2,3 are called **lower nibble** whereas bits 4,5,6,7 are called **upper nibble**. So basically, a register is a memory allocated in the CPU, usually having a size of 1 byte (8 bits).

Next, every register has a name, and every bit of it also has a name. Take the following example.

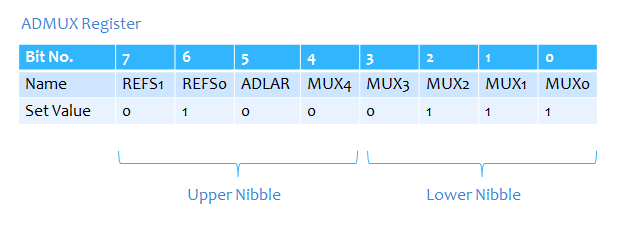
[](http://maxembedded.files.wordpress.com/2011/06/admux.png?resize=618%2C222)

ADMUX Register

Here, the name of the register is ADMUX (don’t worry about the name, we will discuss it later). Also, note that each bit also has a name and an initial value. Each bit of the register can have a value of either 0 or 1 (initially it is 0). Now suppose, I write

|  |
| --- |
| ADMUX = 0b01000111; |

This means that the ADMUX register has been updated as follows:

[](http://maxembedded.files.wordpress.com/2011/06/admux_1.png?resize=618%2C231)

ADMUX Register after setting values

This can also be achieved by the following codes:

|  |
| --- |
| ADMUX = (1<<REFS0)|(1<<MUX2)|(1<<MUX1)|(1<<MUX0);  ADMUX = 0x47;    //Hex form |

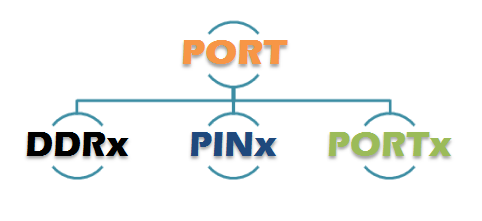
So, got an idea of what registers are and how are they defined/initialized? We will discuss about various registers one by one as required. Right now, we are concerned with only three registers, DDR, PIN and PORT.

**Port Operation Registers**

The following registers are related to the various port operations that we can perform with the GPIO pins.

* DDRx – Data Direction Register
* PORTx – Pin Output Register
* PINx – Pin Input Register

where x = GPIO port name (A, B, C or D)

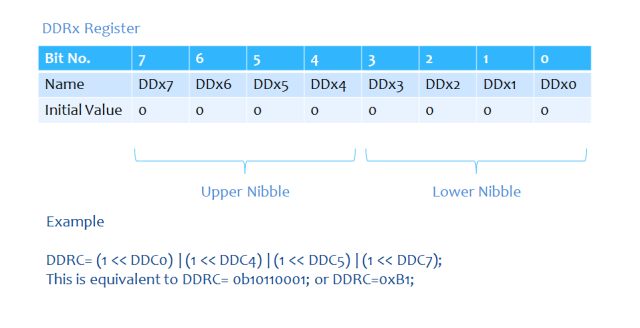
[](http://maxembedded.files.wordpress.com/2011/06/different-port-operations.png?resize=482%2C221)

Different Port Operations

**DDRx Register**

As I mentioned earlier, the GPIO pins are the digital I/O pins i.e. they can act as both input and output. Now, how do we know that the pin is an output pin or input? The DDRx (Data Direction Register) helps in it.

DDRx initializes the port. Have a look at it’s bit structure.

[](http://maxembedded.files.wordpress.com/2011/06/ddrx1.png?resize=640%2C328)

DDRx Register

The ‘x’ in DDRx is just for representation. It is replaced by the corresponding port i.e. x = A, B, C, D. Say for the example shown in the diagram above

|  |
| --- |
| DDRC = (1<<DDC0)|(1<<DDC4)|(1<<DDC5)|(1<<DDC7); |

This is equivalent to

|  |
| --- |
| DDRC = 0b10110001; |

and as well as

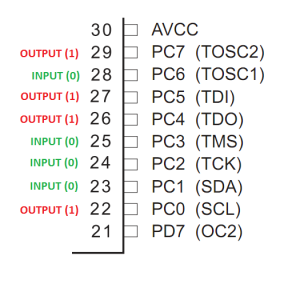
|  |
| --- |
| DDRC = 0xB1; |

and even

|  |
| --- |
| DDRC = (1<<0)|(1<<4)|(1<<5)|(1<<7); |

So, did you get how to declare it? Suppose I would like to initialize my port B, then I would have written

|  |
| --- |
| DDRB = (1<<DDB0)|(1<<DDB4)|(1<<DDB5)|(1<<DDB7); |

[](http://maxembedded.files.wordpress.com/2011/06/ddrc_port.png?resize=300%2C284)

DDRC Example

All right, now that we are done with the declaration, let me explain you what it does. Always remember, in the case of DDRx, **1 stands for output and 0 stands for input**. In the following statement (given below), port C is initialized such that the pins PC0, PC4, PC5 and PC7 are output pins whereas pins PC1, PC2, PC3 and PC6 are input pins.

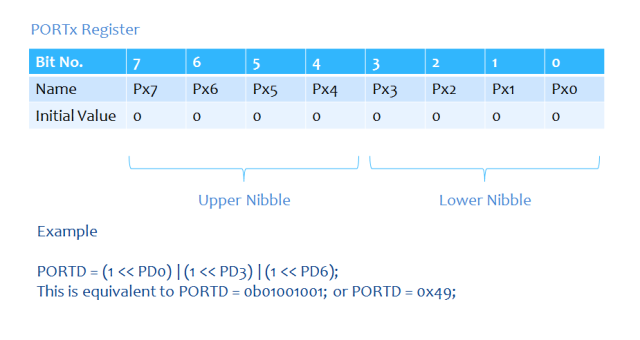
This is represented in the adjoining figure. The pins marked as input now has the capability to *read* the voltage level at that pin and then treat it as HIGH if it is above the threshold level, or else it will treat it as LOW. Generally, the threshold level is half the VCC.

Similarily, the pins marked as output have the capability to either *become* HIGH (voltage = VCC) or LOW (voltage = zero) as directed/written in the code.

|  |
| --- |
| DDRC = (1<<DDC0)|(1<<DDC4)|(1<<DDC5)|(1<<DDC7); |

**PORTx Register**

The PORTx register determines whether the output should be HIGH or LOW of the output pins. In simplified terms, once the DDRx register has defined the output pins, we need to set them to give an output of HIGH or LOW. The PORTx register goes as follows.

[](http://maxembedded.files.wordpress.com/2011/06/portx.png?resize=640%2C340)

PORTx Register

The register declaration is similar to the DDRx register, except that we change the names, that’s all! One such example is given above in the diagram. ***The following declarations are one and the same***.

|  |
| --- |
| PORTD = (1 << PD0)|(1 << PD3)|(1 << PD6);  PORTD = (1 << 0)|(1 << 3)|(1 << 6);  PORTD = 0b01001001;  PORTD = 0x49; |

Now, let’s consider the following statements:

DDRC = 0b10110001;

PORTC = 0b10010001;

OUTPUT = 0b10010001; /\*This is not a C executable line, this line is just for explanation\*/

The port C is initialized using the DDRx register. The highlighted bits correspond to the output pins. Now, just concentrate on the highlighted bits only. Since they are output pins, wherever I state ‘1’ in PORTC, that pin goes HIGH (1), giving an output voltage of VCC at that pin.

Now consider the following set of statements:

DDRC = 0b10110001;

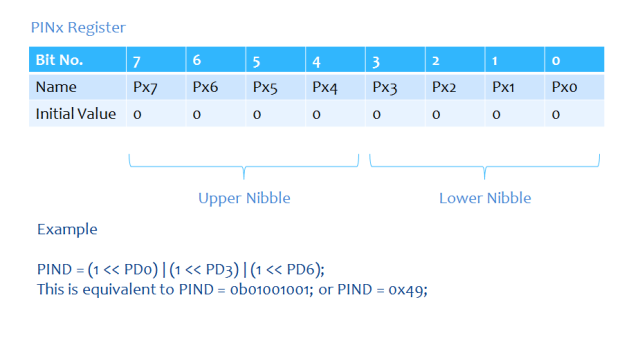
PORTC = 0b10010101;

OUTPUT = 0b10010001; /\*This is not a C executable line, this line is just for explanation\*/

Once again, the bits highlighted in orange correspond to the output pins. So, whatever value (0 or 1) desired in the orange region is reflected in the output. Now, look at the magenta highlighted bit. Inspite of being set as HIGH in the PORTx register, the output is LOW. This is because that pin is initialized as an input pin by DDRx. **Hence, PORTx cannot change the properties of that pin. Hence, in general, PORTx cannot modify the properties of a pin which is initialized as input by DDRx**.

**PINx Register**

The PINx register gets the reading from the input pins of the MCU. The register goes as follows:

[](http://maxembedded.files.wordpress.com/2011/06/pinx.png?resize=640%2C340)

PINx Register

The register declaration procedure stands the same. Also note that the names of the bits of PORTx and PINx registers are same.

Now, let’s consider the following statements:

DDRC = 0b10110001;

PINC = 0b01001011;

INPUT = 0b01001011; /\*This is not a C executable line, this line is just for explanation\*/

Here, the highlighted bits correspond to the pins that are initialized as input by the DDRx. In the second line, the PINx register is defined. Well, this line is just to explain the concept, practically, we always use PINx as a condition (like in IF or in WHILE loop). As per the second statement, the PINx command reads the values only at the input pins.

Now, consider the following set of statements:

DDRC = 0b10110001;

PINC = 0b01011010;

INPUT = 0b01001010; /\*This is not a C executable line, this line is just for explanation\*/

Here, you can compare it with the example I gave for PORTx. Since the magenta-highlighted bit is an output pin, PINx cannot change it’s properties. **Hence, in general, PINx cannot modify the properties of a pin which is initialized as output by DDRx and vice versa**.

**Example Code Snippet**

Let’s demonstrate the use of the DDRx, PORTx and PINx registers from the following code snippet:

|  |
| --- |
| DDRC = 0x0F;  PORTC = 0x0C;    // lets assume a 4V supply comes to PORTC.6 and Vcc = 5V  if (PINC == 0b01000000)      PORTC = 0x0B;  else      PORTC = 0x00; |

**Code Explained:**

* DDRC = 0x0F; is equivalent to DDRC = 0b00001111; This means that the pins PC0…PC3 are output pins (can be manipulated using PORTC) and pins PC4…PC7 are input pins (whose levels determine the value of PINC).
* PORTC = 0x0C; is equivalent to PORTC = 0b00001100; This means that the pins PC2 and PC3 have a HIGH voltage (Vcc = 5V) and pins PC0 and PC1 have LOW voltage (0V). The other pins have low voltage by default.
* if (PINC = 0b01000000) checks the input voltage at pin PC6. Since it is mentioned in the comment that a 4V is supplied to PORTC.6 (same as pin PC6), this condition is true (as 4 > 2.5, where 2.5V is the threshold, 5/2 = 2.5).
* Since the if condition is true, PORTC = 0x0B; is executed.
* If the if condition is not satisfied, PORTC = 0x00; will be executed.

We can also put it inside a while loop to run it continuously i.e. it always checks for the voltage at pin PC6, and the outputs at PC0, PC1 and PC3 go high only if the voltage at PC6 is greater than 4V.

|  |
| --- |
| DDRC = 0x0F;    while(1)  {      // a condition of 4V supply to PORTC.6 and Vcc = 5V      if (PINC == 0b01000000)          PORTC = 0x0B;      else          PORTC = 0x00;  } |

To learn how to code and simulate using AVR Studio 5, visit [this page](http://maxembedded.wordpress.com/2011/06/12/using-avr-studio-5/).

So, here we end up with the port operations that can be performed with an AVR MCU. Though the concept has been explained using ATMEGA16/32, the concepts are equally good for any AVR MCU! Just go through the datasheet in order to get an idea of the registers.

Thank you for reading this! The comments’ box is down below! ;)

sahil January 27, 2016