Basic Input/Output Operations

Posted on May 9, 2008, by Ibrahim KAMAL, in Micro-controllers, tagged

In this third part of the 89s52 tutorial, we are going to study the basic structure and configuration of I/O ports. Then we are going to apply this theory on simple experimental projects, using a LED and switch, to experiment with the different I/O features of the micro-controller.

At this point of the tutorial, we are going to transfer programs to the microcontroller, using an ***ISP (In System Programmer)****. If you don't have one, you can build one* ***here****. Along all the tutorial, we are going to use our ISP connector.*

I/O port detailed structure

It is important to have some basic notions about the structure of an I/O port in the 8051 architecture. You will notice along this tutorial how this will affect our choices when it comes to connect I/O devices to the ports. Actually, the I/O ports configuration and mechanism of the 8051 can be confusing, due to the fact that a pin acts as an output pin as well as an input pin in the same time.

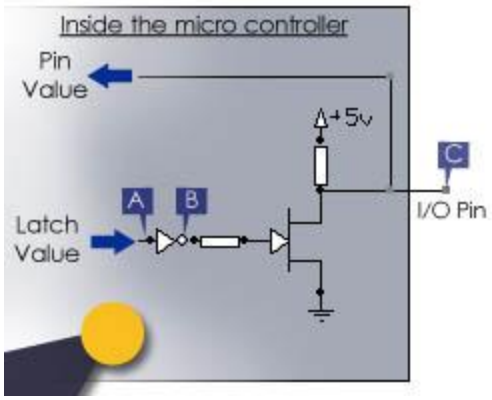

figure 3.1.A

Figure 3.1.A shows the internal diagram of a single I/O pin of port 1. The first thing you have to notice, is that there are two different direction for the data flow from the microcontroller's processor and the external pin: The Latch value and the Pin value. The latch value is the value that the microcontroller tries to output

on the pin, while the pin value, is the actual logic state of the pin, regardless of the latch value that was set by the processor in the first place. The microcontroller reads the state of a pin through the Pin value line, and writes through the latch value line. If you imagine the behavior of the simple circuit in ***figure 3.1.A***, you'll notice that the I/O pin should follow the voltage of the Latch value, providing 5V through the pull-up resistor, or 0V by connecting the pin directly to the GND through the transistor.

When the pin is pulled high by the pull-up resistor, the pin can output 5V but can also be used as an input pin, because there is no any risk of short-circuit due to the presence of a resistor. This can be easily verified by connecting the pin to 0V or to 5V, the two possible outcomes are both unharmful for the microcontroller, and the **PIN value** line will easily follow the value imposed by the external connection.

Now imagine the opposite configuration, where the latch value would be low, causing the pin to provide 0V, being directly connected to GND through the transistor. If in this situation, an external device tries to raise the pin's voltage to 5V, a short circuit will occur and some damage may be made to the micro-controller's port or to the external device connected to that pin.

To summarize, in the 8051 architecture, to use a PIN as an input pin, you have to output '1', and the pin value will follow the value imposed by the device connected to it (switch, sensor, etc…). If you plan to use the pin as an output pin, then just output the required value without taking any of this in consideration.

Even if some ports like P3 and P0 can have a slightly different internal composition than P1, due to the dual functions they assure, understanding the structure and functioning of port 1 as described above is fairly enough to use all the ports for basic I/O operations.

Simple output project: Blinking a led

A first simple project to experiment with the output operations is to blink a LED. Assuming you have successfully written and compiled the code as explained in the previous part of the tutorial, now we are going to transfer the HEX file corresponding to that code on the 89s52 microcontroller. Let us recall that the HEX file is a machine language file, generated by the compiler, originally from a C code.

The code for blinking a LED is as follow:

```
#include <REGX52.h>
#include <math.h>


delay(unsigned int y){
    unsigned int i;
    for(i=0;i<y;i++){;}
```

```
}


main(){
    while(1){
        delay(30000);
        P1_0 = 0;
        delay(30000);
        P1_0 = 1;
    }
}
```

Before transferring the HEX file to the target micro-controller, the hardware have to be constructed. First you have to provide a clean (noiseless) 5V power supply, by connecting the Vcc pin (40) to 5V and the GND pin (20) to 0V. Then you have provide a mean of regulating or generating the clock of the processor. The easiest and most efficient way to do this is to add a crystal resonator and two decoupling capacitors of approximately 30 pF (see the crystal X_1 and the capacitors C_1 and C_2 on ***figure 3.2.A***). Then, you have connect pin 31 (EA) to 5V. The EA pin is an active low' pin that indicate the presence of an external memory. Activating this pin by providing 0V on it will tell the internal processor to use external memories and ignore the internal built-in memory of the chip. By providing 5V on the EA pin, its functionality is deactivated and the processor uses the internal memories (RAM and FLASH). At last, you have to connect a standard reset circuitry on pin 9 composed of the 10 Kohm resistor R_2 and the 10 uF capacitor C_3, as you can see in the schematic. You can also add a switch to short-circuit pin 9 (RST) and 5V giving you the ability to reset the microcontroller by pressing on the switch (the processor resets in a high level is provided on the RST pin for more than 2 machine cycles).

Those were the minimum connections to be made for the microcontroller to be functional and able to operate correctly. According to the fact that we are going to use an ISP programmer, A connector is added by default to allow easy in system programming.

For our simple output project, a LED is connected to P1.0 through a 220 ohm resistor R1, as you can see in ***figure 3.2.***A below. Note that there are other ways to connect the LED, but now that you understand the internal structure of the port, you can easily deduce that this is the only way to connect the LED so that the current is fully controlled by the external resistor R_1. Any other connection scheme would involve the internal resistor of the port, which is 'uncontrollable'.

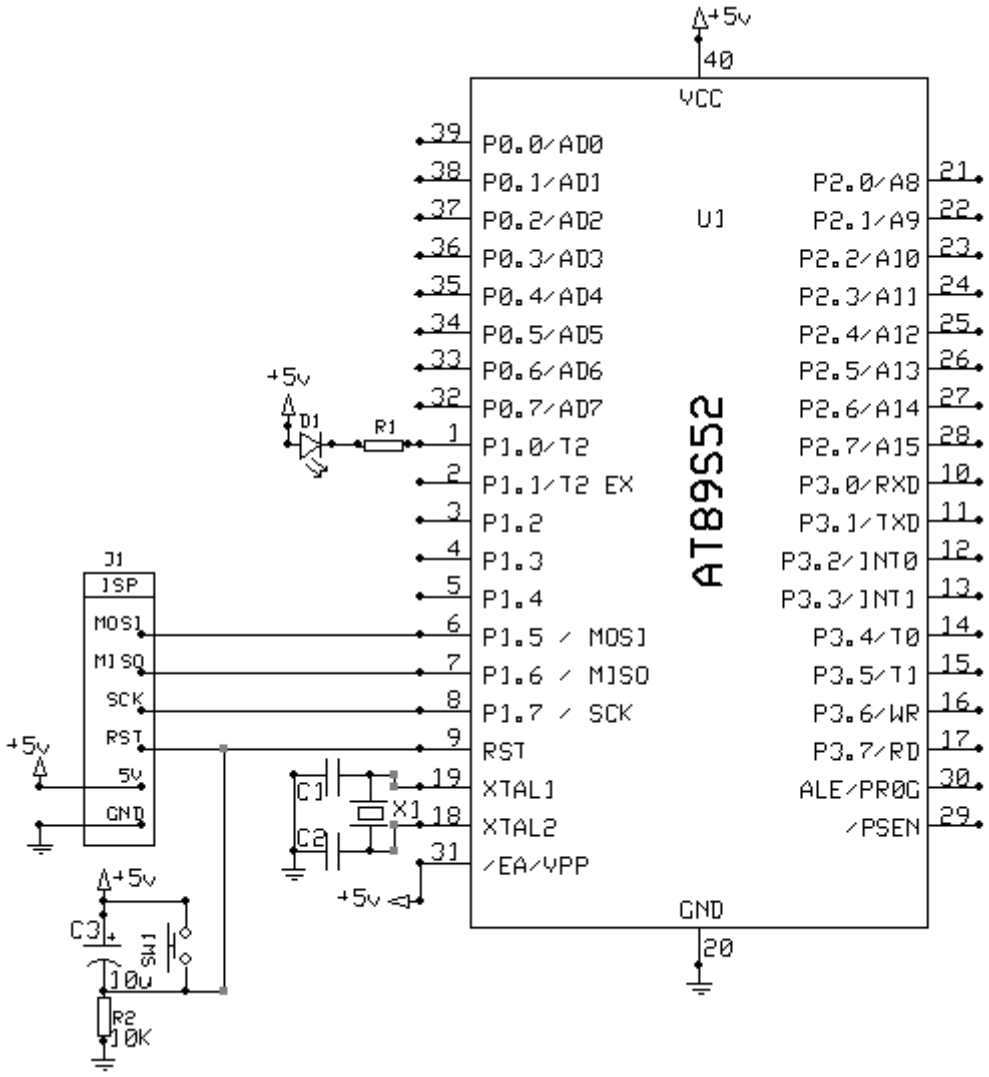

figure 3.2.A

In order get rid of any confusion, a picture of the implementation of this simple project on a bread board is provided to help you visualize the hardware part of the project:

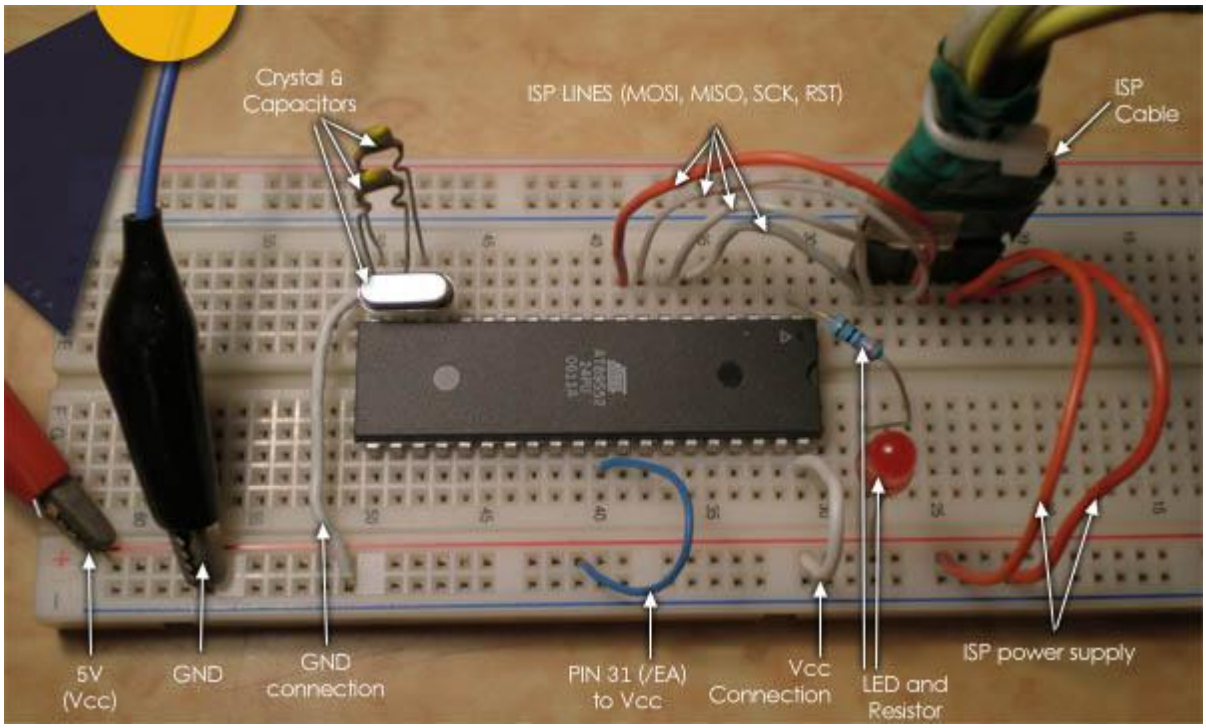

figure 3.2.B

Note that the reset switch and R/C filter are not present on this breadboard, the **reset** functionality of the ISP cable was used instead.

At this stage, you can finally connect your ISP programmer, launch the ISPprog software, browse the HEX file for programming the FLASH, and transfer it to the micro-controller, as described in the ISP page. You can eventually use any other available programming hardware and/or software.

If all your connections are correct, you should see the LED blinking as soon as the programming (transfer) is finished. You can experiment with different delay in the code to change the blinking frequency. Don't forget that for any change to take place, you have to rebuild your source code, generating a new hex file (replacing the old one) and retransfer the freshly generated HEX file to the micro-controller.

Simple Input/Output project

The most simple input operation you can implement to the previous project is a push button, to control the LED. The schematic below (***figure 3.3***) shows how a switch is added on another pin of Port 1. We could have connected the switch on another port, but i preferred to stress on capability of the 8051 architecture to share input and output pins on the same port. Notice how the switch is connected to ground, and without a pull up resistor. recalling the internal structure of a pin, you'll notice that this is the simplest way to connect a switch, and also the most adapted to the 8051 architecture, making use of the internal pull up resistor, and preventing any eventual short circuits if the port is not well configured.

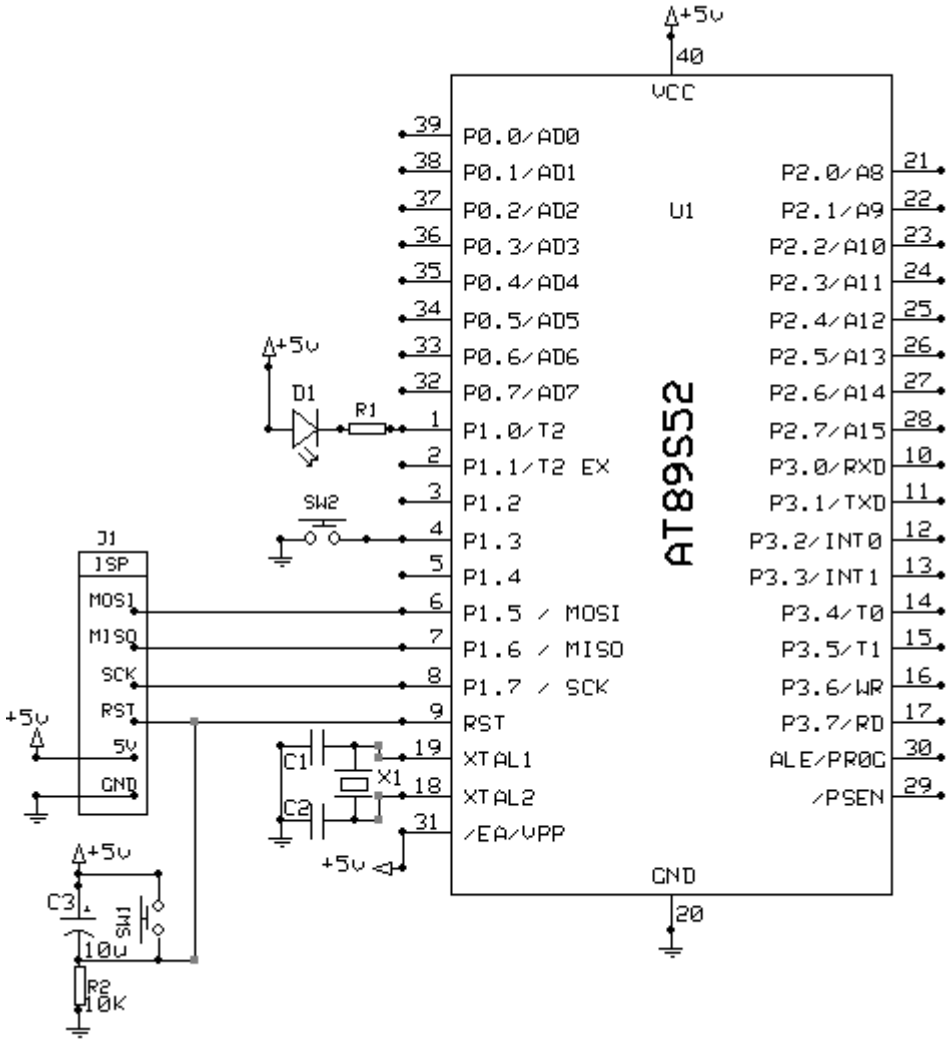

figure 3.3

Now, that the hardware is finalized, an adequate software have to be designed and written to assure the correct functioning of the system. To control a led, there are many possible solutions. The first one I propose is the simplest one: a software that turns on the LED as long as the push button is pressed and turn it off otherwise and whose source code would be as the following:

```
#include <REGX52.h>

#include <math.h>


delay(unsigned int y){

    unsigned int i;

    for(i=0;i<y;i++){;}
```

```
}


main(){
    P1_3 = 1; //Setup P1_3 as input pin
    while(1){
        if(P1_3 == 0){
            P1_0 = 0; //Turn ON the LED
        }else{
            P1_0 = 1; //Turn OFF the LED
        }
    }
}
```

The other solution I propose is a software that turns ON the LED for a couple of seconds each time the switch is pressed, then turn it off automatically. The source code would be as the following:

```
#include <REGX52.h>
#include <math.h>


unsigned long time, ON_time; //Global Variables


void main(){
    P1_3 = 1;     //Set up P1_3 as input pin
    ON_time = 100000;
    while(1){
         if (time < ON_time){
            time++;    // start or continue counting
            P1_0 = 0;     //Turn ON the LED
         }else{
            P1_0 = 1;    // Turn OFF the LED
```

```
        }
        if (P1_3 == 0){     // if the switch is pressed,
            time = 0;    // reset 'time' to 0
        }
    }
}
```

The source code above may need some explanation: First you can notice that there is no ‘delay’ function, as it is not needed anymore. Two variables are defined ‘time’ and ‘ON_time’, they are both ‘usigned long’ type, so that they can manage relatively huge numbers, required to generate dozens of seconds delays. The variable ‘time’ will be used to count the elapsed time (in number of code cycles), while the ‘ON_time’ is used to store the fixed time period which the LED should stay ON after the push button is released. Then those two variables as constantly compared, and as soon as the elapsed time reaches the required ‘ON_time’, the led switches off, and the ‘time’ counting stops, to prevent eventual overflow. A push on the button would set the ‘time’ back to 0, and the whole process can start again.

You can try on your own to figure out other ways of optimizing the control of a LED or a number of LEDs.

Exercise:

To conclude this part of the tutorial, i suggest this simple exercise:

“*Using the same schematic (**figure 3.3**), build a software that would allow you to toggle the state of a led on simple button press. A press on the button to turn the LED ON or OFF, depending on its initial state*“

You can try your source code by simulating it in KEIL IDE, or by testing it directly on your breadboard. IF you can’t find the solution, you can seek for help in the **forums**.

Source: http://www.ikalogic.com/part-3-basic-inputoutput-operations/