AVR Studio 5 Debugging

Hello friends. Since you now aware of creating and building your project using AVR Studio 5, you are in a position to know how to debug and simulate your code using the AVR Simulator. The AVR Studio 5 Simulator has the following features:

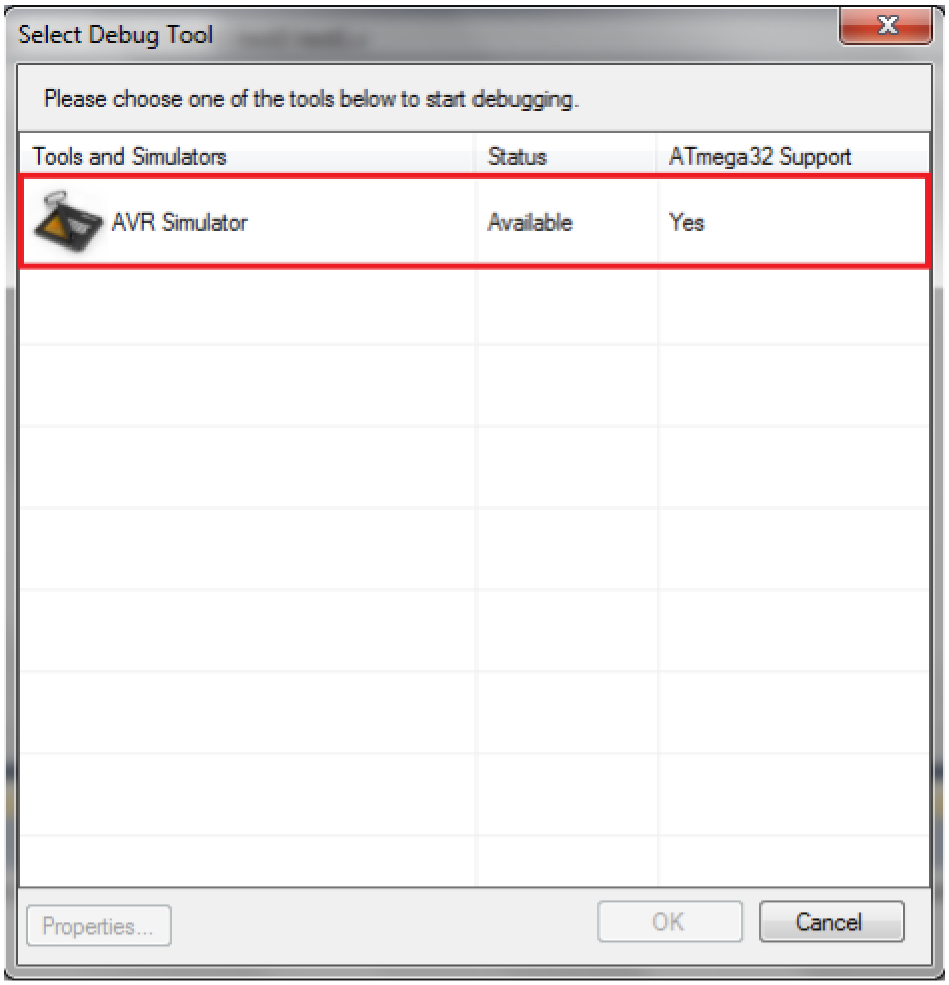
- It supports software emulation of any real AVR device without actually connecting it.
- It gives access to all the peripherals of the real MCU but no external devices.
- So if you want to give external signals, you need to do it yourself, either by manually updating the registers or creating a stimuli file.

Now, let's take the following code example to explain the functionality of the AVR Simulator.

```
#include <avr/io.h>



int main(void)

{

    uint8_t counter;

    DDRB = 0xFF;

    while(1)

    {
        counter++;    // insert breakpoint here

        PORTB = counter;

    }

}
```

Debugging the code using Simulator

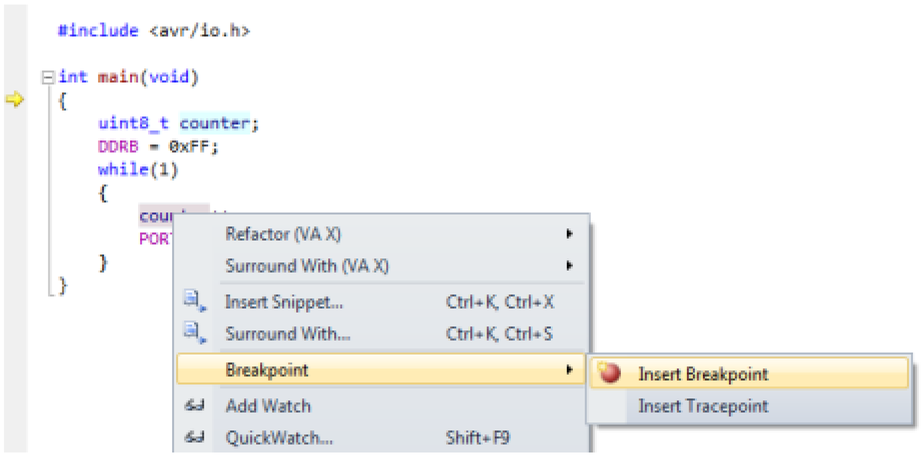
- Now, click on the **Debug** menu and then click on **Start Debugging and Break**. If initially no debugger is chosen, AVR Studio 5 will ask you to choose a **Debug Tool**. **AVR Simulator** is always an option there. Choose it and click **OK**.


Select Debug Tool

Please choose one of the tools below to start debugging.

Tools and Simulators	Status	ATmega32 Support
AVR Simulator	Available	Yes

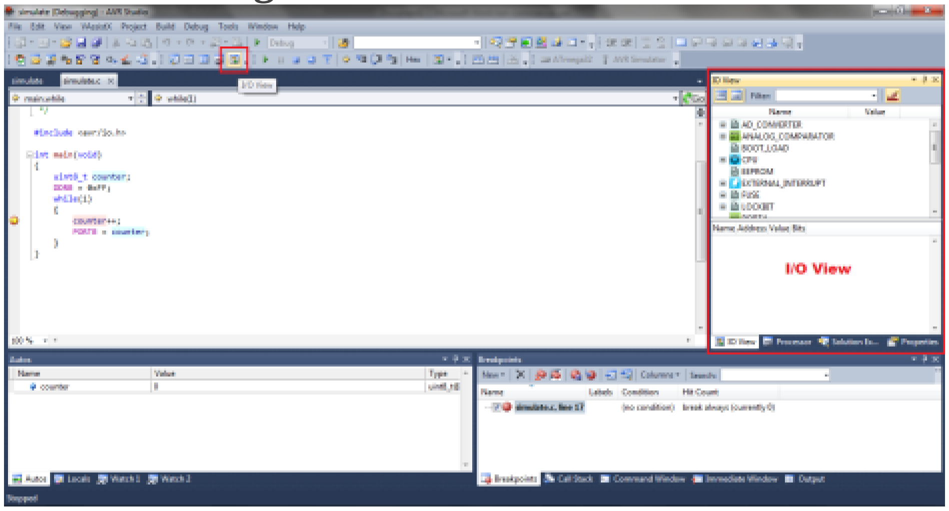
- After this, debugging starts and halts in the beginning of main(). You can see a yellow arrow mark determining the current executing line.
- Λet's place a breakpoint in the main and start execution. Highlight the variable counter in counter++, right click it, go to **Breakpoint** and then click on **Insert Breakpoint**.



```
#include <avr/io.h>

int main(void)
{
    uint8_t counter;
    DDRB = 0xFF;
    while(1)
    {
        coun
        POR
    }
}
```

Insert Breakpoint

- Now press the **play** button (**F5**) or click on **Continue** from the **Debug**menu to run to the breakpoint.
- Now look at the affected registers in the **I/O view**. If you don’t have the I/O View open, you can select it from the **Debug** toolbar or from the **Debug**windows menu.

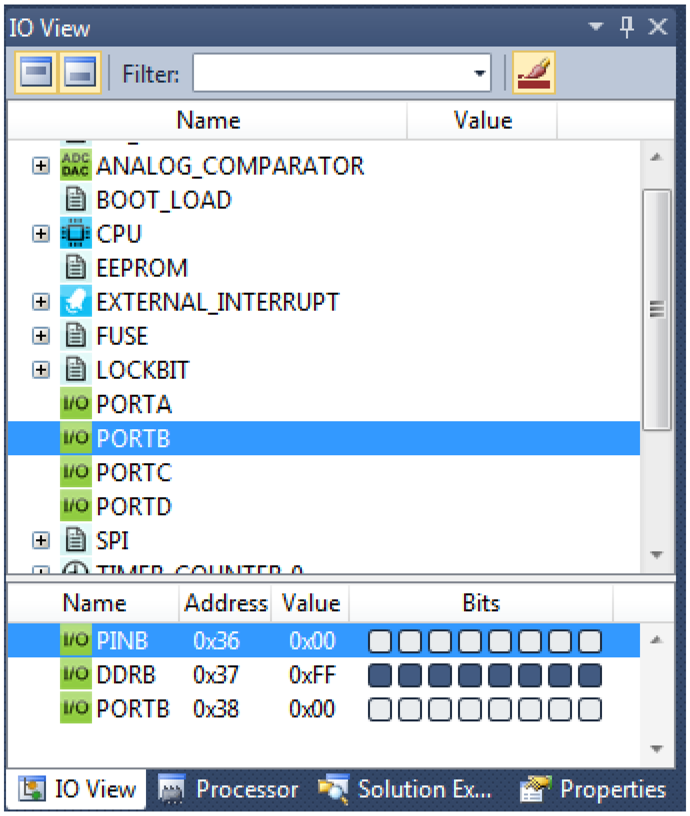


```
#include <avr/io.h>

int main(void)
{
    uint8_t counter;
    DDRB = 0xFF;
    while(1)
    {
        counter++;
        PORTB = counter;
    }
}
```

I/O View

- All the peripheral features are mentioned over here. We can monitor any changes from the software and also manipulate the values to provide input.

- Now, since the counter changes the value of PORTB, scroll down in the I/O View and click on PORTB.

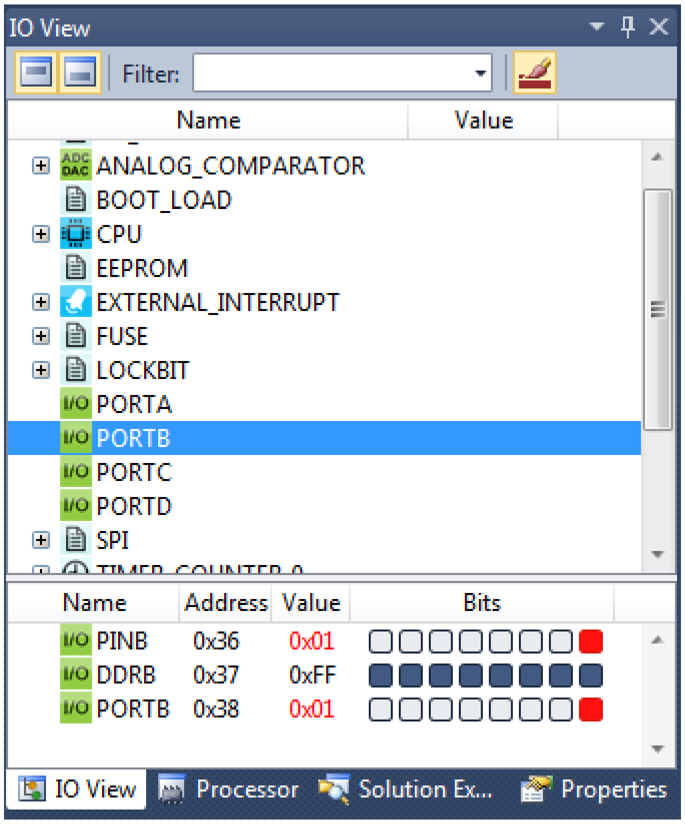


Name	Value
ANALOG_COMPARATOR	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
FUSE	
LOCKBIT	
PORTA	
PORTB	
PORTC	
PORTD	
SPI	

Name	Address	Value	Bits
PINB	0x36	0x00	□□□□□□□□
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0x00	□□□□□□□□

Choose PORTB in I/O View

- Upon clicking PORTB, you can see the three registers assigned for PORTB operations, PINB, DDRB and PORTB. You can also view their current values.
- A solid block represents ‘1’ whereas a blank block represents ‘0’.
- Since it the beginning of main(), we defined DDRB = 0xFF, all the blocks are filled. You can also look at its value there.
- Now, press the play button. The loop iterates once and stops at the breakpoint. You can see that values of PINB and PORTB have changed to 0×01. This is because after one iteration, counter = 1.


IO View

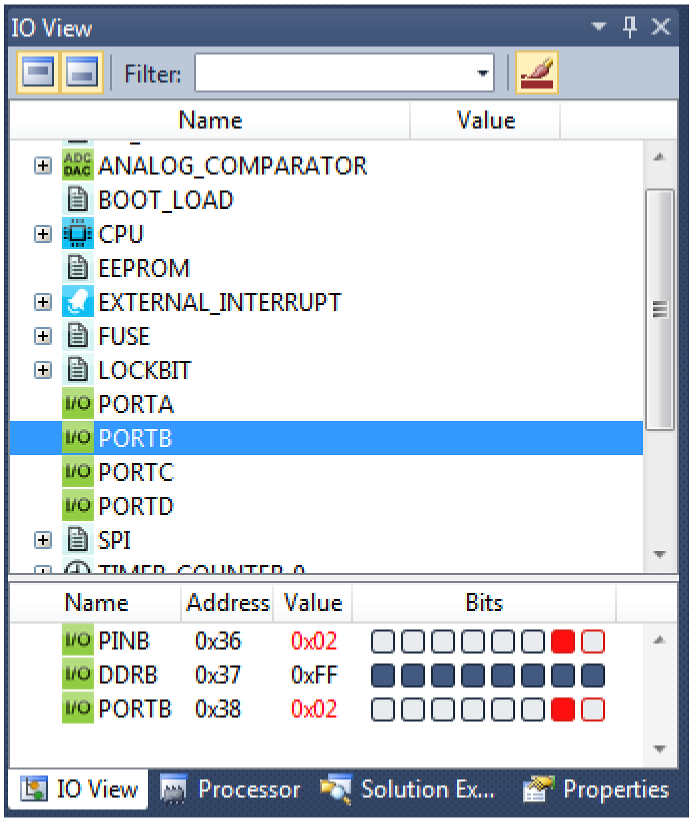
Filter:

Name	Value
ANALOG_COMPARATOR	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
FUSE	
LOCKBIT	
PORTA	
PORTB	
PORTC	
PORTD	
SPI	

Name	Address	Value	Bits
PINB	0x36	0x01	□□□□□□□🟥
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0x01	□□□□□□□🟥

Choose PORTB in I/O View

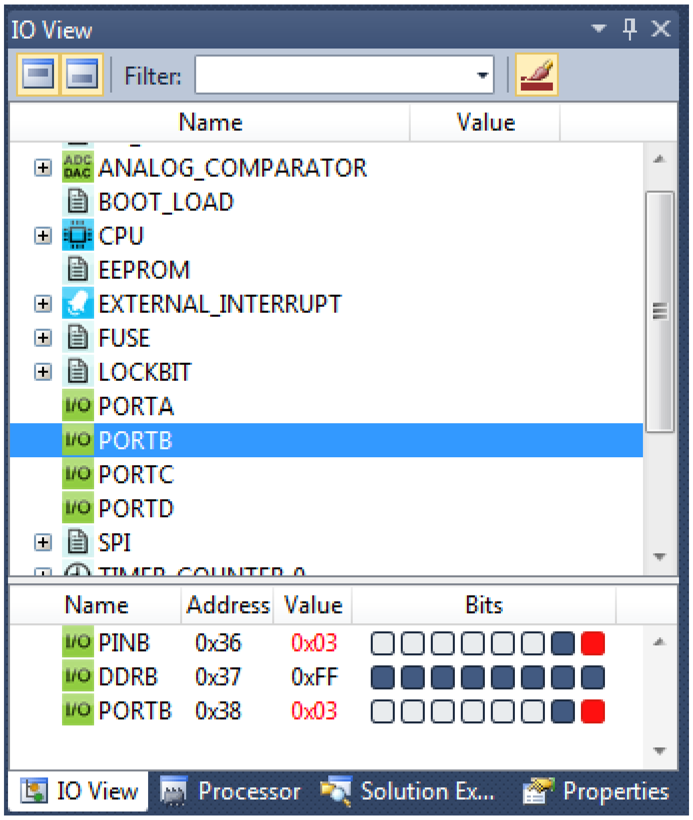
- Upon clicking PORTB, you can see the three registers assigned for PORTB operations, PINB, DDRB and PORTB. You can also view their current values.
- A solid block represents ‘1’ whereas a blank block represents ‘0’.
- Since it the beginning of main(), we defined DDRB = 0xFF, all the blocks are filled. You can also look at its value there.
- Now, press the play button. The loop iterates once and stops at the breakpoint. You can see that values of PINB and PORTB have changed to 0×01. This is because after one iteration, counter = 1.



Name	Value
ANALOG_COMPARATOR	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
FUSE	
LOCKBIT	
PORTA	
PORTB	
PORTC	
PORTD	
SPI	
TIMER_COUNTER_0	

Name	Address	Value	Bits
PINB	0x36	0x02	□□□□□□🟥□
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0x02	□□□□□□🟥□

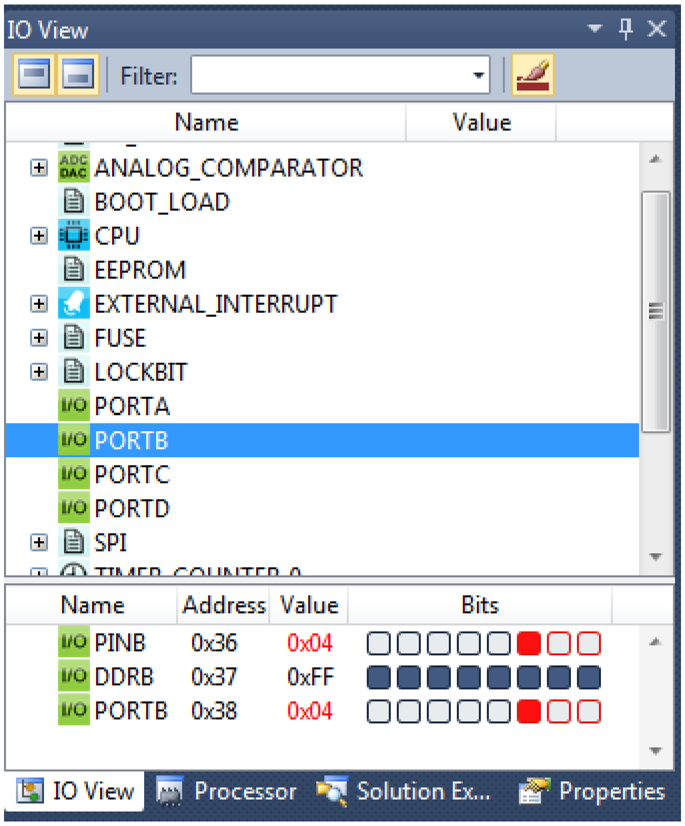
Debug in Progress (2)



Name	Value
ANALOG_COMPARATOR	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
FUSE	
LOCKBIT	
PORTA	
PORTB	
PORTC	
PORTD	
SPI	
TIMER_COUNTER_0	

Name	Address	Value	Bits
PINB	0x36	0x03	□□□□□□■🟥
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0x03	□□□□□□■🟥

Debug in Progress (3)

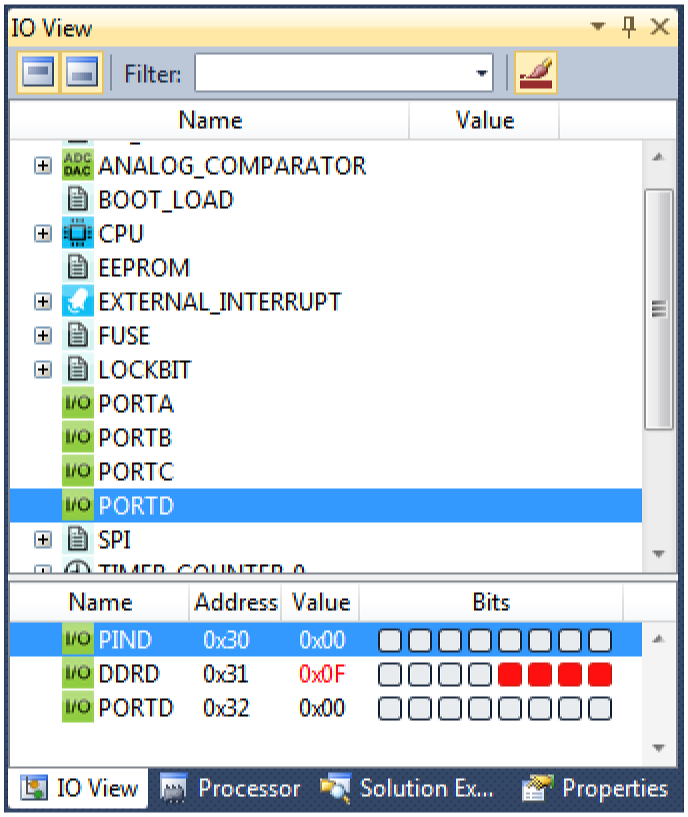


Name	Value
ANALOG_COMPARATOR	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
FUSE	
LOCKBIT	
PORTA	
PORTB	
PORTC	
PORTD	
SPI	

Name	Address	Value	Bits
PINB	0x36	0x04	□□□□□■□□
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0x04	□□□□□■□□

Debug in Progress (4)

- Now, if you want to change some other registers (apart from the ones changed by the code), simply click on the corresponding register and change its value.
- Say for example you want to change the value of DDRD. Click on PORTD and then give any value you want. You can also click on the corresponding bits to toggle the values


IO View

Filter:

Name	Value
ANALOG_COMPARATOR	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
FUSE	
LOCKBIT	
PORTA	
PORTB	
PORTC	
PORTD	
SPI	
TIMER_COUNTER_0	

Name	Address	Value	Bits
PIND	0x30	0x00	☐☐☐☐☐☐☐☐
DDRD	0x31	0x0F	☐☐☐☐🟥🟥🟥🟥
PORTD	0x32	0x00	☐☐☐☐☐☐☐☐

IO View | Processor | Solution Ex... | Properties

Making External Changes

So now we are done with the basics of AVR Studio 5. There are more advanced features of debugging in AVR Studio 5 which includes In-System Debugging which is a kind of runtime debugging unlike the software emulation that we learnt in this post. However, we are not interested in discussing these concept here as it is possible only with AVR Programmers.

So for now, enjoy!