



AVR[®] Simulator

AVR[®] Simulator

AVR[®] Simulator

The AVR[®] Simulator is a software simulator for Microchip AVR devices where the user can run and debug code without using any hardware. It simulates the CPU, including all instructions, interrupts, and most of the on-chip I/O modules.

The Simulator operates within the Microchip Studio application as a debug target. This enables the user to use the normal debug commands such as Run, Break, Reset, Single-step, set breakpoints, and watch variables. The I/O, memory, and register views are fully functional using the Simulator.

The simulator is based on software models of devices derived directly from the hardware designs, and are thus cycle-accurate per the real devices.

Table of Contents

AVR® Simulator.....	1
1. Using Simulator in Microchip Studio.....	3
1.1. Overview.....	3
1.2. Using Simulator in a Debugging Session.....	3
1.3. Using Simulator in the Programming Dialog.....	12
1.4. Key Differences Between Simulator and Hardware Tools.....	13
1.5. Key Differences from AVR® Studio 4 and AVR32 Studio.....	13
2. Known Issues in Simulator.....	15
2.1. General Issues.....	15
2.2. Device and Family Specific Issues.....	15
3. Revision History.....	17
The Microchip Website.....	18
Product Change Notification Service.....	18
Customer Support.....	18
Microchip Devices Code Protection Feature.....	18
Legal Notice.....	19
Trademarks.....	19
Quality Management System.....	20
Worldwide Sales and Service.....	21

1. Using Simulator in Microchip Studio

1.1 Overview

The simulator is a debugging tool made to behave similarly to any other debugging tool. It can be selected on the same list as the other tools, and it can be used immediately since it does not need any hardware connections. When a debug session is started, the simulator loads a simulator model of the selected device.

The performance of the simulator model is slow compared to a real device, but because it is made of software it gives the user some extra debugging possibilities that are not available with a real device.

1.2 Using Simulator in a Debugging Session

When a debugging session with the simulator is started, a simulator model of the selected device is loaded and programmed with the user's application. After the application is loaded into the application memory, a Power-on Reset (POR) is applied to the model. Hence, the simulator will start at the reset vector with the POR reset flag set. If the user used "Start Debugging/Continue (F5)" to start the debug session, the simulator model would now be running and start executing the application. If the user used "Start Debugging and Break (Alt+F5)", the simulator would start running until it reached the start of the main() function and then do a break.

The Reset button in Microchip Studio applies a POR to the simulator. The content of flash and EEPROM are not lost during Power-on Reset (POR). When doing a reset in Microchip Studio, the execution will normally break at the start of the main() function. By switching to the disassembly view before performing the reset, the execution will halt at the reset vector. If Watchdog is programmed to generate a reset, it can be caught through setting a breakpoint at the reset vector and let the simulator run.

There is no way to attach to a running simulator, or detach/disconnect from it and keep it running in the background. Like in the programming dialog, the simulated device ceases to exist when the user stops debugging. This is a restriction that may be lifted in future versions of Microchip Studio.

Because the simulator is a software model, it is not limited by the OCD system on the device, and has certain capabilities that the hardware tools do not share:

- Unlimited numbers of breakpoints regardless of device and OCD system limitations
- Set and delete breakpoints while the target is running
- Can debug and supports trace on all devices, even those that lack an OCD system
- Access to locations that can not be reached by the OCD system
- Provide features that have no counterpart in hardware, for instance, a cycle counter
- Unlike real hardware, the simulator allows flash and EEPROM contents to be changed directly using the Microchip Studio memory view
- Can simulate devices that do not exist yet (early support before samples are available)

1.2.1 Using the I/O View with Simulator

The I/O view generally works in the same manner as with other tools. However, due to the way I/O addresses are mapped to internal signals in the simulator model, there are a few peculiarities. These occur with registers or bit fields within registers that are read-only, write-only, read and write from different hardware registers, or have peculiar write modes. In general, different bit fields within the same register may behave differently because they are mapped to different hardware locations.

When a register is changed in the I/O view, the new value of the register is written to the target and then read back, and the read-back value is shown in the I/O view. For this reason, the value shown may be different from the value written for various reasons.

- In a regular register/bit field with read/write access, when changing its value, the new value is shown immediately
- In some cases, read and write access go to different hardware locations. The effect of this may be that attempting to change a register/bit field has no visible effect even if the register is written (like the USART `UDR`

register in many devices), or that the effect is delayed by one or more cycles (like registers with double buffering).

- If a register/bit field is read-only, attempting to change it will have no effect
- If a register/bit field is write-only, it will normally always read as zero
- Some registers/bit fields have special write modes like set, clear, or toggle. This means that writing a value of one to a bit will perform one of these operations on the bit while writing zero has no effect (this to eliminate the need for read-modify-write sequences to change single bits in a register). Often such registers are mirrored as a regular register at a different address. In these cases, the simulator normally will allow a write to this regular register even if it is documented as read-only. Using the regular register when making changes from the I/O view is much easier.
- When possible, the simulator will often allow full read/write access to registers/bit fields even though they are documented as read-only or write-only. One particularly important example of this is interrupt flags. These are often intended to be read-only, but allowing to write them from the simulator allows triggering the interrupt if the hardware is designed such that the interrupt flag is the cause of the interrupt, which is often the case. This feature is important for stimuli generation, e.g., ADC interrupts can be stimulated through writing the converted value to the ADC data register and then trigger the ADC interrupt through setting the ADC interrupt flag (both the ADC data register and the ADC interrupt flag are typically described as read-only in data sheets).
- In some 32-bit models, some of the peripheral module I/O register accesses use the on-chip bus instead of accessing internal signals directly. In these cases the, simulator will be subject to the same restrictions as the OCD-based emulators. For example, read-only registers remain read-only, some registers require a protection pattern to be written to a different register before the actual register can be written, some peripherals must have their clock explicitly enabled before they can be accessed, etc. Per-device details can be found in [2.2 Device and Family Specific Issues](#).
- Sometimes hardware design makes it impossible to map a bit field properly, with reasonable effort. This most frequently affects the write access, but sometimes even read. Such deficiencies are documented in [2.2 Device and Family Specific Issues](#).
- Finally, unresponsive or otherwise faulty registers in the I/O view can be caused by bugs in the I/O mapping in the simulator model. This is the most common type of bug in the simulator.

Notes:

Notes regarding the above:

1. Bugs or shortcomings in the simulator I/O mapping only affect the debugger view of the register, not the function of the model. An application running on the target will not be affected by such bugs.
2. All that is said about the I/O view in this section is equally true when the memory view is used to access I/O locations. The only difference between the two is the presentation.
3. The limitations listed for the I/O view will also affect stimuli described in [1.2.2 Simulator Stimuli](#).

1.2.2 Simulator Stimuli

1.2.2.1 Introduction

Simulator stimuli is a way to write simple script files that can read or write any register or memory location at a given time during the simulation. The stimuli file can be started at any time during the simulation and will continue until the end of the file or when the simulator session ends.

Note:

Starting with Atmel Studio 6.1, the File Stimulator found in AVR® Studio 4 has been reintroduced. There were two stimuli variants in AVR Studio 4. This is the most recent one used in Simulator 2. The older one is not supported.

Features of the File Stimulator:

- Timing is expressed in terms of delay instead of absolute cycle counter values
- Any I/O register can be assigned to (stimulated), not only port registers
- Individual bits in I/O registers can be stimulated using bitwise assignments
- Directives increase flexibility
- One stimuli file can open and execute another stimuli file

1.2.2.2 Use from Microchip Studio

The stimuli input file must be prepared in advance using a text editor like the Microchip Studio editor. It is recommended to use the extension `.stim` for stimuli files. Before starting debugging, the stimuli file is selected in the **Tool** tab in the project properties page. During a debug session, you can select a new stimfile by using the menu selection *Debug > Set Stimulifile*.

The stimuli generator can be started from Microchip Studio using the menu selection *Debug > Execute Stimulifile*. This option is only available during an active debug session. The time between this action and the stimuli input is exhausted by the last stimuli file being closed is referred to as a stimuli session.

During a stimuli session, all normal debug features like breakpoints, single-step, etc. can be used. Stimuli are applied as the application program is executed (free-running or single-stepped). If the debug session is continued after the stimuli session is ended, it will continue without stimuli (unless a new stimuli session is started).

There is no way to explicitly abort an active stimuli session, apart from ending the debug session.

The output from the stimuli session will be routed to the **Output** pane in Atmel Studio. Select **FileStimuliProvider** from the drop-down list **Show output from**. The FileStimuliProvider pane is created the first time a stimuli file is executed and will remain in place until Microchip Studio is closed. The output is retained across debug sessions. If this is undesired, it has to be cleared manually. See [1.2.2.2.1 The FileStimuliProvider Output Pane Format](#) for an explanation of the output pane format.

Logging can only be started using commands in stimuli files. There are presently no GUI facilities to set up or start logging. Microchip Studio currently only supports logging to file, not to the output pane.

1.2.2.2.1 The FileStimuliProvider Output Pane Format

When a stimuli session is running, all output from the session is printed to the FileStimuliProvider pane. All output lines start with a time stamp on the form `#00000028`. This is the value of the cycle counter in decimal. There are three types of output printed to this pane, which is shown below.

Stimuli File Open or Close and end of Session

Whenever a stimuli file is opened or closed, it is logged in the output. The first line of the output will be the opening of the initial stimuli file. It might look like this:

```
#00000000 Opened file 'C:\Project\Test\test.stim' as [ 0]
```

The number in square brackets is the file number assigned to the file, starting at 0. This number is used to refer to the file in other messages. If another stimuli file is opened, it will get number 1, and so on.

When the file is closed, a similar message appears:

```
#00000028 Closed file 'C:\Project\Test\test.stim' [ 0]
```

When the last stimuli file has been closed, the stimuli session is ended, and no more stimuli will be produced. Notice that the last closed file does not have to be the initial file. When the last file has been closed, the following message is produced:

```
#00000036 All stimuli files closed
```

Command Echo

All text from the stimuli files is echoed at the time it is executed. The echoed text is preceded with the time stamp and file number, and may look like this:

```
#00000016 [ 0] PINB ^= 0x03
#00000016 [ 0] #4
#00000020 [ 0] PINA = 0x01
```

In this example, the assignments to PINB and PINA are separated in time by four cycles, and all commands are read from file 0.

Error and Warning Messages

Error/warning messages refer explicitly to the file name and line number where the error was detected and may look like this:

```
#00000006 [ 0] log foo.bar
Error: L:\Project\Test\test.stim(6): Syntax error
```

1.2.2.3 Stimuli File Format

A stimuli file is a simple ASCII plain text file containing stimuli commands, one command per line. Apart from [comments](#), there are only three kinds of commands: [delays](#), [assignments](#), and [directives](#).

1.2.2.3.1 Delays

A delay is specified by an # character followed by the duration of the delay in CPU clock cycles. #20 means a delay of 20 clock cycles. Using delays is the only way of separating commands in time. Commands that are not separated by delays will be executed simultaneously, i.e., within the same clock cycle. In the current implementation, stimuli are only evaluated between CPU single-steps, meaning that the delay may be longer than specified if it would end in the middle of a multi-cycle instruction.

1.2.2.3.2 Assignments

Assignments are used to assign a new value to an I/O register. The operators are listed in [Table 1-1](#).

Table 1-1. Stimuli Assignments Operators

Statement	Description
<code>target = value</code>	Direct assignment; set target equal to value
<code>target = value</code>	Bitwise OR assignment; bits that are 1 in value will be set in target, remaining bits unchanged
<code>target &= value</code>	Bitwise AND assignment; bits that are 0 in value will be cleared in target, remaining bits unchanged
<code>target ^= value</code>	Bitwise XOR assignment; bits that are 1 in value will be toggled (inverted) in target, remaining bits unchanged

The target can be the numerical memory address of an I/O register in the I/O map. For simple devices with a flat I/O structure such as tinyAVR® and megaAVR®, the register name, as found in the data sheet, can also be used.

For devices with complex I/O structure (XMEGA®, UC3, SAM), it is for now recommended to use addresses. The easiest way to determine the address is to bring up the I/O view and select the desired register. The address can be copied from the I/O view (select the desired register, right-click, and select “Copy Address”).

The value can be either a numerical constant specified in decimal, octal, or hex according to C syntax, or it can have the form `*source`. When using this syntax `source` is the name or memory address of an I/O register. The current interpreter does not support expressions.

```
GPIOR0 = *GPIOR1 // Allowed!
GPIOR0 = *GPIOR1 + 1 // Not allowed!
```

1.2.2.3.3 Directives

A directive is initiated by a \$ character, followed by a command. Directives are used to control various aspects of stimuli execution and logging.

The currently supported directives are listed in [Table 1-2](#).

Table 1-2. Stimuli Directives

Directive	Arguments ¹	Description
<code>\$stimulate</code>	<i>filename</i>	Start reading of stimuli from a new file. The new file will be read in parallel with the current file. This is currently the only way of opening multiple stimuli files within a stimuli session.
<code>\$quit</code>		Close the current stimuli file. The remainder of the file will be discarded, and the file is closed. (The same as reaching the end of the file.)

¹ Multiple arguments are separated by space.

.....continued												
Directive	Arguments ¹	Description										
\$break		Break program execution. Stimuli file(s) remain open, and stimuli will be resumed when program execution is resumed.										
\$repeat	<i>number</i>	Start a repeat loop, repeat number times until \$endrep directive										
\$endrep		End of a repeat loop										
\$log	<i>IO-register mask</i>	Set up register logging. If <i>mask</i> is specified, the log will only update when the bits in the mask change. The <i>mask</i> will be OR'ed with any previous mask for the same address. Logging will not start until \$startlog directive is executed.										
\$unlog	<i>IO-register mask</i>	Stop register logging. If <i>mask</i> is specified, only the bits in the mask will stop being logged.										
\$startlog	<i>filename writemode</i>	Start logging to the named file. The <i>writemode</i> is optional, the default mode is to append to the file. Table 1-3. Log Writemodes										
		<table border="1" style="width: 100%;"> <thead> <tr> <th style="text-align: left;">Type</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>a</td> <td>Append to file (default)</td> </tr> <tr> <td>o</td> <td>Overwrite any existing file</td> </tr> </tbody> </table>	Type	Description	a	Append to file (default)	o	Overwrite any existing file				
Type	Description											
a	Append to file (default)											
o	Overwrite any existing file											
\$stoplog		Stop logging										
\$fuse	<i>address value</i>	Set fuse byte at <i>address</i> to <i>value</i> . ² Fuse addresses generally start at 0.										
\$reset	<i>type</i>	Reset device. Possible reset types are listed below. Table 1-4. Reset Types										
		<table border="1" style="width: 100%;"> <thead> <tr> <th style="text-align: left;">Type</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>p</td> <td>Power on reset (POR)</td> </tr> <tr> <td>e</td> <td>External reset (EXT)</td> </tr> <tr> <td>b</td> <td>Brown-out detection (BOD)</td> </tr> <tr> <td>s</td> <td>Spike (AVR XMEGA only, similar to external reset)</td> </tr> </tbody> </table>	Type	Description	p	Power on reset (POR)	e	External reset (EXT)	b	Brown-out detection (BOD)	s	Spike (AVR XMEGA only, similar to external reset)
Type	Description											
p	Power on reset (POR)											
e	External reset (EXT)											
b	Brown-out detection (BOD)											
s	Spike (AVR XMEGA only, similar to external reset)											

¹ Multiple arguments are separated by space.

² After the fuse is changed, a power-on reset must be applied to make the change effective.

³ This command was introduced in Atmel Studio 6.1 SP2.

.....continued

Directive	Arguments ¹	Description										
\$memload	<i>file segment nocheck</i>	<p>Load the contents of <i>file</i> into the memory. On AVR designs, you can specify <i>segment</i> to select where to load the data. If you add <i>nocheck</i> to the end of the command, any checksum errors in the file will be ignored.</p> <p>Table 1-5. Memory Segments</p> <table border="1"> <thead> <tr> <th>Segment</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>s</td> <td>Data memory (default)</td> </tr> <tr> <td>f</td> <td>Flash</td> </tr> <tr> <td>e</td> <td>EEPROM</td> </tr> <tr> <td>i</td> <td>I/O</td> </tr> </tbody> </table>	Segment	Description	s	Data memory (default)	f	Flash	e	EEPROM	i	I/O
Segment	Description											
s	Data memory (default)											
f	Flash											
e	EEPROM											
i	I/O											
\$memdump	<i>file adr size segment</i>	<p>Dump the contents of the memory to <i>file</i>, starting at <i>adr</i> and dumping <i>size</i> number of bytes. Optionally, specify <i>segment</i> to select which memory to dump.</p> <p>Table 1-6. Memory Segments</p> <table border="1"> <thead> <tr> <th>Segment</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>s</td> <td>Data memory (default)</td> </tr> <tr> <td>f</td> <td>Flash</td> </tr> <tr> <td>e</td> <td>EEPROM</td> </tr> <tr> <td>i</td> <td>I/O</td> </tr> </tbody> </table>	Segment	Description	s	Data memory (default)	f	Flash	e	EEPROM	i	I/O
Segment	Description											
s	Data memory (default)											
f	Flash											
e	EEPROM											
i	I/O											

A log entry is generated whenever a logged I/O register changes the value for whatever reason. The log format is compatible with the stimuli format, which means that log output can be used as stimuli input. The log file will consist of delay statements and assignments.

Note:

Relative paths are relative to the directory of the initial stimuli file.

1.2.2.3.4 Comments

Comments are initiated with // and last until the end of the line. Block comments (/* ... */) are not supported.

1.2.2.4 Known Issues

Stimuli Files

Note:

The mapping between register names and addresses only works reliably for devices with a flat I/O structure with unique register names. Dotted notation does not work. Use numeric addresses instead.

- Logging of some I/O registers on 32-bit devices may be unsupported. This will be documented on a per-device basis.
- In assignments, the operator (=, etc) must be surrounded by spaces
- The stimuli interpreter will fail if the last line of the stimuli input file is not terminated by a newline
- On 8-bit devices, it is not possible to assign values to 16- or 32-bit register tuples, e.g., to assign to `ADC`, one must assign to `ADCL` and `ADCH` separately. See the example in [1.2.2.5 Example Stimuli Session](#).

⁴ This command was introduced in Atmel Studio 6.2 SP1.

¹ Multiple arguments are separated by space.

- Error reporting leaves a lot to be desired
- The timing of stimuli can be a cycle or two off compared to delay specification because stimuli files are evaluated only between CPU single-steps in the current implementation
- Sharing violation if attempting to edit a stimuli file while open

1.2.2.5 Example Stimuli Session

Session 1; Simple AVR[®] Program

The following example should work with any ATtiny or ATmega device having at least two GPIO ports (`PORTA` and `PORTB`). It should be run with the I/O view active and set up to show the `PORTA` and `PORTB` modules.

This program sets up `PORTA` as output and `PORTB` as input, and then loops, reading whatever is present on `PINB`, increments it by one, and outputs it at `PORTA`.

```
reset:
    rjmp start
start:
    ldi r16, 0xff
    out DDRA, r16 // PORTA => output
    clr r0
    out DDRB, r0 // input <= PORTB
loop:
    in r0, PINB // Requires stimuli for any action
    inc r0
    out PORTA, r0
    rjmp loop
```

Without stimuli, this program will read 0 from `PINB`, and output 1 to `PORTA`, forever.

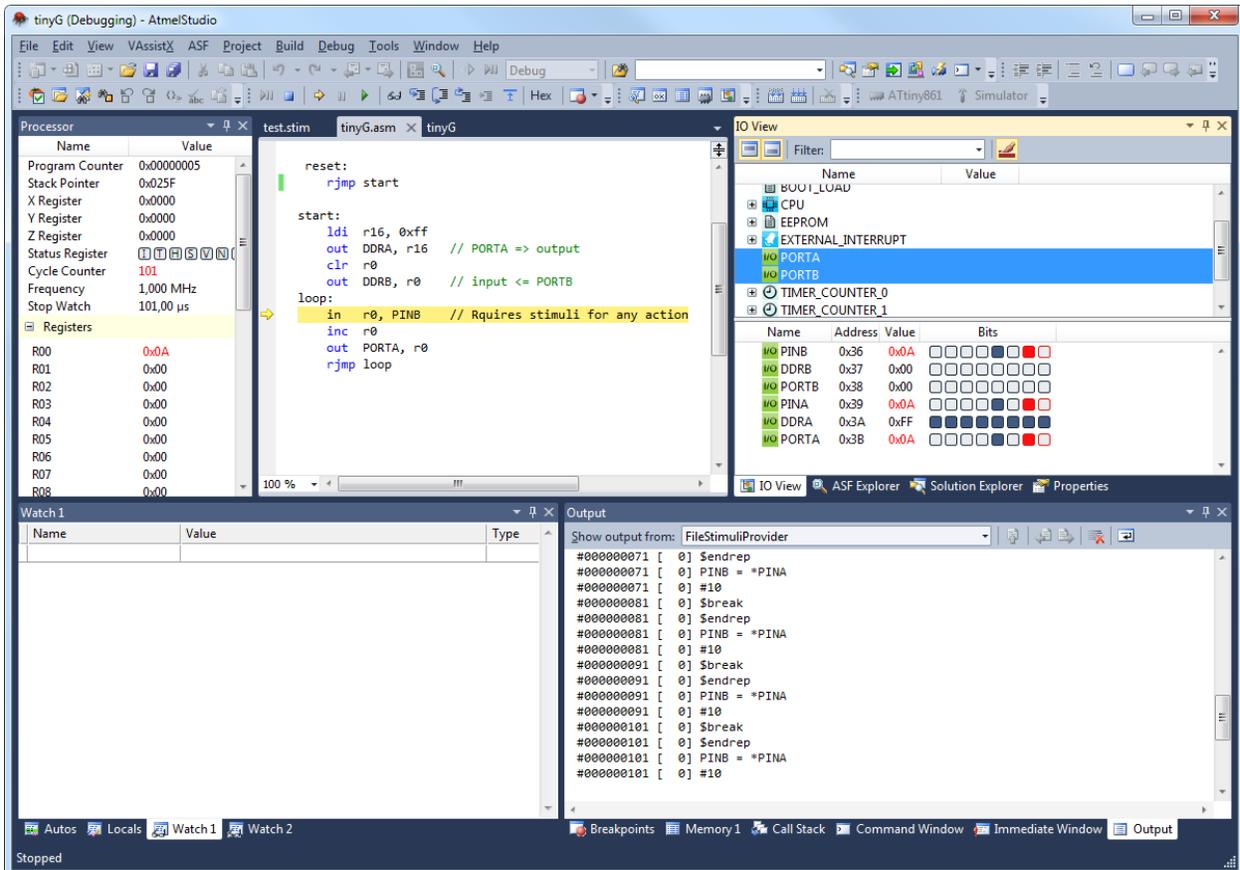
Now apply the following stimuli file:

```
// Example stimuli file, feeds any output on PINA back to PINB.
#10
$repeat 10
    PINB = *PINA
    #10
    $break
$endrep
```

`PINB` will now be driven by the output from `PORTA` every 10th cycle, and the effect will be directly visible in the I/O view. Thanks to the `$break` directive inside the loop, the program will stop every 10 cycles without any breakpoint in the program.

¹ Multiple arguments are separated by space.

Figure 1-1. Screenshot of the Stimuli Session from the Example



Session 2; Simple AVR[®] Logging

The following example is made for ATmega328P but should work with most ATtiny or ATmega devices. This program starts `TIMER0` and then enters a loop, constantly adding 1 to the value in `PIND` and assigning the result to `PORTD`.

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

int main(void)
{
    DDRD = 0xFF;
    TCCR0B = (1<<CS00);
    while(1)
    {
        PORTD = PIND + 1;
    }
}
```

When applying the following stimuli, it will create a log file that starts logging the value of `PORTD` and `PIND`. After 20 cycles, it changes to logging `PIND` and `TCNT0`, and after another 20 cycles, it continues to only log `PIND` for 60 cycles.

```
$log PORTD
$log PIND
$startlog mega328p_log_output.stim
#20
$unlog PORTD
$log TCNT0
#20
$unlog TCNT0
#60
$stoplog
$break
```

This will produce a file called `mega328p_log_output.stim` with the following content:

```
#6
PORTD = 0x01
#1
PIND = 0x01
#4
PORTD = 0x02
#1
PIND = 0x02
#4
PORTD = 0x03
#1
PIND = 0x03
#3
TCNT0 = 0x10
TCNT0 = 0x11
#1
TCNT0 = 0x12
#1
PIND = 0x04
TCNT0 = 0x13
#1
TCNT0 = 0x14
#1
TCNT0 = 0x15
#1
TCNT0 = 0x16
#1
TCNT0 = 0x17
#1
PIND = 0x05
TCNT0 = 0x18
#1
...
```

Session 3; Logging PWM Output

The following example is made for ATmega328P but should work with most ATtiny or ATmega devices. This program starts `TIMER0` in fast PWM mode. The output of the PWM signal on PD6 is controlled by PB0. 1 on PB0 enables PWM output, 0 on PB0 disables it.

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

int main(void)
{
    DDRB = 0x00;
    DDRD = 0xFF;
    PORTD = 0x00;
    OCR0A = 0x20;
    TCCR0A = (1<<COM0A0 | 1<<WGM01 | 1<<WGM00);
    TCCR0B |= (1<<CS00);
    while(1)
    {
        if(PINB & 0x01)
            TCCR0B |= (1<<WGM02);
        else
            TCCR0B &= ~(1<<WGM02);
    }
}
```

When applying the following stimuli, it will create a log file that starts logging the value of `PINB` and `PIND`. After 20 cycles, it also starts to log `TCNT0`. Five cycles later, it applies a 1 to PB0 to start the PWM output, and another five cycles later, it turns off logging of `TCNT0`. For the next 200 cycles, the PWM output on PD6 will be logged before it is turned off by setting PB0 to 0. Then it continues for another 200 cycles before it breaks.

```
$log PINB
$log PIND
$startlog mega328p_log_output.stim
#20
// Log TIMER0 counter
$log TCNT0
#5
```

```
// Set pin PB0 to '1', Start PWM output
PINB |= 0x01
#5
// Stop logging TIMER0 counter
$unlog TCNT0
#200
// Set pin PB0 to '0', Stop PWM output
PINB &= 0xFE
#200
$stoplog
$break
```

This will produce a file called `mega328p_log_output.stim` with the following content:

```
#20
TCNT0 = 0x09
TCNT0 = 0x0a
#1
TCNT0 = 0x0b
#1
TCNT0 = 0x0c
#1
TCNT0 = 0x0d
#1
TCNT0 = 0x0e
#1
TCNT0 = 0x0f
#1
PINB = 0x01
TCNT0 = 0x10
#1
TCNT0 = 0x11
#1
TCNT0 = 0x12
#1
TCNT0 = 0x13
#15
PIND = 0x40
#33
PIND = 0x00
#33
PIND = 0x40
#33
PIND = 0x00
#33
PIND = 0x40
#33
PIND = 0x00
#22
PINB = 0x00
#11
PIND = 0x40
#1
PIND = 0x00
```

1.3 Using Simulator in the Programming Dialog

Although it is possible, using the simulator in the programming dialog does not have that much practical use, due to the volatility of the simulator. However, it can be useful for new users to explore the programming dialog without the risk of damaging any real hardware, or even without having invested in any hardware at all. It can also be used to create a production file, which can include flash content, EEPROM content, user signatures, fuse settings, and lock bits settings.

The simulator supports the following operations:

- Read device ID
- Erase device/memories

- Program, read, and verify memories
- Program, read, and verify fuses
- Program, read, and verify lockbits

When the programming dialog is opened, and a device is selected, the simulated device starts with default factory settings and empty memories. Any data programmed into the simulator will be retained until a different device is selected, or the programming dialog is closed.

1.4 Key Differences Between Simulator and Hardware Tools

In Microchip Studio, the simulator is handled basically like any other hardware tool. It can be selected for both programming dialog and debug sessions. However, there are some key differences:

- The simulator is *volatile*, meaning it has no memory between sessions. Anything programmed into the simulator non-volatile memories (like flash contents or fuses) in one session will be forgotten after the session is quit. When a session is ended, the simulated device quite literally ceases to exist, and when a new session is started a new simulated device is created from scratch, starting its existence in its initial state. Specifically, this means you cannot program fuses in the programming dialog, and later start a debugging session with these fuse settings intact. A new session always starts with default fuse settings. Fuses and other options for use in debugging sessions with the simulator have to be set up using a simulator-specific property page active only during a simulator debugging session. A feature allowing parts of or the entire simulator state to be saved between sessions may be considered in the future.
- The simulator has no selectable programming or debugging interfaces. This because the simulated device is implemented by a software model, and all access of internals within the model is done via a software API; no physical interface is involved, and all access is completely non-intrusive since nothing needs to be clocked to retrieve or write data.
- Presently, only a single instance of the simulator can be run at a time. Also, the simulator lacks serial numbers like most of the hardware tools have.
- The simulator is *not* real-time. This means that the speed of the simulation (measured in simulated CPU cycles per second) is significantly lower than on a real device. The simulator can only utilize a single CPU core in a multi-core CPU, so upgrading your PC with more cores will not make it faster, but there are indications that a larger CPU cache gives better performance.
- The simulator is *not* a complete model of the device. While digital logic is simulated cycle-accurately, all analog periphery is presently lacking. Also, the modeling of NVM⁵ is incomplete. The degree of incompleteness varies between devices, see [2. Known Issues in Simulator](#).
- Device support is not complete. The simulator depends on a software model of each device/family to be simulated. Supported devices are shown in the device selector when the simulator is selected.
- The simulator runs in isolation, meaning the surroundings of the simulated device is not simulated. To run real-life applications on the simulator, stimuli must be provided to the inputs of the simulated device. Stimuli can be provided by simple stimuli files, as shown in [1.2.2 Simulator Stimuli](#).
- For the ATmega128 simulator model the ATmega103 compatibility fuse (M103C) is not programmed by default, and SUT is set to the shortest possible value in some devices
- In most tinyAVR® devices, the SELFPRGEN fuse is unprogrammed by default, preventing SPM from working. Set this fuse when working with SPM.
- Many tinyAVR devices have external reset as an alternative port function, and an RSTDISBL fuse to disable the external reset pin. When unprogrammed (the default setting), the corresponding port pin will not work as expected.
- The CKDIV8 fuse is normally unprogrammed to increase simulation speed

1.5 Key Differences from AVR® Studio 4 and AVR32 Studio

The model-based simulator uses the same technology used in AVR Studio 4, also known as “Simulator 2”. It has been extended to support modeling of 32-bit UC3 and Arm® devices as well, but Arm models are not publicly distributed.

⁵ Flash, EEPROM

AVR[®] Simulator

Using Simulator in Microchip Studio

The instruction set simulator from AVR32 Studio is presently not available in Microchip Studio. The model-based simulator is used instead.

2. Known Issues in Simulator

This section lists known bugs and shortcomings of the current simulator. Numbers shown in parentheses refer to bug numbers in the bug tracking system.

2.1 General Issues

- The simulator settings dialog is not yet implemented (#13412)
- AVR XMEGA[®] B1, UC3 except for UC3A and UC3L series, and some tinyAVR[®] and megaAVR[®] devices are not yet supported in Microchip Studio.
- AT90CAN*/ATmega*C*, AT90USB*/ATmega*U*, AT90PWM*, and ATtiny87/167 devices will never be supported by simulator models
- External memory is not supported by the simulator (#7570, #9442)
- When writing to I/O PORT registers and the port is configured as an output, the change may show up immediately in the PIN register, not one cycle delayed as on the real chip (#7188). This only affects the Microchip Studio I/O view, not the execution of programs.
- The AVR32 Studio instruction set simulator still not implemented in Microchip Studio 7.0 (#11557)
- If the simulated device is put to sleep, single-stepping will not promote the program counter. Like a real device, this state will remain until something wakes it up from sleep. The reason for this behavior is that the CPU is not executing code when asleep. The alternative would be having the single-step hang until the device wakes up, which would hang the entire Microchip Studio until that happens (maybe never).
- Setting fuses in the code of the simulated program may interfere with the operation of the simulator. Especially the OCDEN fuse and the SUT fuse may cause the simulator to abort the session. This is caused by a time-out in the simulator that counts the number of clock ticks it takes to get the chip out of reset. If this time-out is exceeded, the simulator will fail the session.

2.2 Device and Family Specific Issues

2.2.1 tinyAVR[®] Devices

- ATtiny40 RAMDR register cannot be written from Microchip Studio's I/O view. Workaround: Write SRAM directly via the memory view.
- Selecting ATtiny10 external clock in the simulator will cause Microchip Studio to hang (#9349)
- PRR register of ATtiny25/45/85 does not work (#5584, it works with the other devices having PRR)
- ATtiny25/45/85: Watchdog time-out too long (~64 times longer than it should be @1 MHz)
- System clock prescaler not included in simulator models. Writing CLKPR will not affect the system clock. (All devices except ATtiny10/9/5/4, ATtiny20, and ATtiny40.)
- CLKPR is not updated when debugging if the CKDIV8 fuse is programmed (#10515)

2.2.2 megaAVR[®] and Smart Battery Devices

- ATmega16HVB stack pointer does not initialize correctly and must be initialized by the application. This is an issue with the actual chip, and the simulator just reflects reality, see data sheet Errata section 38.1.1 (Rev. B).
- ATmega169PA/165PA/329P/325P/3250P/3290P/649P/645P/6490P/6450P watchdog timer does not work (#9301)
- The system clock prescaler is not included in simulator models. Writing CLKPR will not affect system clock.
- External interrupt rising flank triggers on both flanks on some ATmega devices (#13434)
- CLKPR is not updated when debugging if the CKDIV8 fuse is programmed (#10515)

2.2.3 AVR[®] XMEGA[®] Devices

- Writing/erasing FLASH using SPM and EEPROM from an application is not yet implemented in ATxmega devices (#7611)

- Simulation is slow compared to other devices
- I/O view: Enabling the watchdog by setting ENABLE bits in WDT.CTRL register doesn't work
- Attempting to change CLK.CTRL register through I/O view will cause Studio to hang (mapped read-only in XMEGA E5)
- XMEGA E5: TCCOM compare/capture register and TCxn_CTRLD read-only in I/O view
- XMEGA E5: FAULTn.CTRLG registers bit 5-7 read-only in I/O view
- XMEGA E5: SPI.DATA register read-only in I/O view
- XMEGA E5: CRC.DATIN register read-only in I/O view

2.2.4 32-bit AVR® UC3 Devices

- The UC3A model I/O map is not yet completed. The following I/O modules will always show zero in the Microchip Studio I/O view: SMC, HMATRIX, FLASHC, MACB, SMC, DRAMC, INTC, PM, and RTC.
- Simulation of UC3 devices is slow compared to other devices due to complexity and size of the UC3 designs

3. Revision History

Doc. Rev.	Date	Comments
A	12/2020	Initial document release

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication is provided for the sole purpose of designing with and using Microchip products. Information regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Klear, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICKit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQL, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2020, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-6742-7

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
<p>Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Tel: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com</p> <p>Atlanta Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455</p> <p>Austin, TX Tel: 512-257-3370</p> <p>Boston Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088</p> <p>Chicago Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075</p> <p>Dallas Addison, TX Tel: 972-818-7423 Fax: 972-818-2924</p> <p>Detroit Novi, MI Tel: 248-848-4000</p> <p>Houston, TX Tel: 281-894-5983</p> <p>Indianapolis Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380</p> <p>Los Angeles Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800</p> <p>Raleigh, NC Tel: 919-844-7510</p> <p>New York, NY Tel: 631-435-6000</p> <p>San Jose, CA Tel: 408-735-9110 Tel: 408-436-4270</p> <p>Canada - Toronto Tel: 905-695-1980 Fax: 905-695-2078</p>	<p>Australia - Sydney Tel: 61-2-9868-6733</p> <p>China - Beijing Tel: 86-10-8569-7000</p> <p>China - Chengdu Tel: 86-28-8665-5511</p> <p>China - Chongqing Tel: 86-23-8980-9588</p> <p>China - Dongguan Tel: 86-769-8702-9880</p> <p>China - Guangzhou Tel: 86-20-8755-8029</p> <p>China - Hangzhou Tel: 86-571-8792-8115</p> <p>China - Hong Kong SAR Tel: 852-2943-5100</p> <p>China - Nanjing Tel: 86-25-8473-2460</p> <p>China - Qingdao Tel: 86-532-8502-7355</p> <p>China - Shanghai Tel: 86-21-3326-8000</p> <p>China - Shenyang Tel: 86-24-2334-2829</p> <p>China - Shenzhen Tel: 86-755-8864-2200</p> <p>China - Suzhou Tel: 86-186-6233-1526</p> <p>China - Wuhan Tel: 86-27-5980-5300</p> <p>China - Xian Tel: 86-29-8833-7252</p> <p>China - Xiamen Tel: 86-592-2388138</p> <p>China - Zhuhai Tel: 86-756-3210040</p>	<p>India - Bangalore Tel: 91-80-3090-4444</p> <p>India - New Delhi Tel: 91-11-4160-8631</p> <p>India - Pune Tel: 91-20-4121-0141</p> <p>Japan - Osaka Tel: 81-6-6152-7160</p> <p>Japan - Tokyo Tel: 81-3-6880-3770</p> <p>Korea - Daegu Tel: 82-53-744-4301</p> <p>Korea - Seoul Tel: 82-2-554-7200</p> <p>Malaysia - Kuala Lumpur Tel: 60-3-7651-7906</p> <p>Malaysia - Penang Tel: 60-4-227-8870</p> <p>Philippines - Manila Tel: 63-2-634-9065</p> <p>Singapore Tel: 65-6334-8870</p> <p>Taiwan - Hsin Chu Tel: 886-3-577-8366</p> <p>Taiwan - Kaohsiung Tel: 886-7-213-7830</p> <p>Taiwan - Taipei Tel: 886-2-2508-8600</p> <p>Thailand - Bangkok Tel: 66-2-694-1351</p> <p>Vietnam - Ho Chi Minh Tel: 84-28-5448-2100</p>	<p>Austria - Wels Tel: 43-7242-2244-39 Fax: 43-7242-2244-393</p> <p>Denmark - Copenhagen Tel: 45-4485-5910 Fax: 45-4485-2829</p> <p>Finland - Espoo Tel: 358-9-4520-820</p> <p>France - Paris Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79</p> <p>Germany - Garching Tel: 49-8931-9700</p> <p>Germany - Haan Tel: 49-2129-3766400</p> <p>Germany - Heilbronn Tel: 49-7131-72400</p> <p>Germany - Karlsruhe Tel: 49-721-625370</p> <p>Germany - Munich Tel: 49-89-627-144-0 Fax: 49-89-627-144-44</p> <p>Germany - Rosenheim Tel: 49-8031-354-560</p> <p>Israel - Ra'anana Tel: 972-9-744-7705</p> <p>Italy - Milan Tel: 39-0331-742611 Fax: 39-0331-466781</p> <p>Italy - Padova Tel: 39-049-7625286</p> <p>Netherlands - Drunen Tel: 31-416-690399 Fax: 31-416-690340</p> <p>Norway - Trondheim Tel: 47-72884388</p> <p>Poland - Warsaw Tel: 48-22-3325737</p> <p>Romania - Bucharest Tel: 40-21-407-87-50</p> <p>Spain - Madrid Tel: 34-91-708-08-90 Fax: 34-91-708-08-91</p> <p>Sweden - Gothenberg Tel: 46-31-704-60-40</p> <p>Sweden - Stockholm Tel: 46-8-5090-4654</p> <p>UK - Wokingham Tel: 44-118-921-5800 Fax: 44-118-921-5820</p>