AMS LAB exercise 2

Simulator and JTAG debugging

Purpose

1. To be able to use the Atmel Studio 7 simulator.
2. To be able to set up and use the Atmel - ICE for debugging purposes.

Material

- Atmel-ICE Users Guide (AMS Blackboard).
- Optional: The Mega2560 data book (the JTAG section).

The exercise

In this exercise we will:

- Create a project and write a C program (intended for later debugging).
- Use the Atmel Studio Simulator to simulate the program execution (only using your PC).
- Connect the Atmel-ICE to the "Arduino Mega2560" board.
- Use the Atmel-ICE for In-Circuit program debugging.

It is assumed that Atmel Studio is already installed at your PC.

Part 1: Write a C program in Atmel Studio

For this exercise we need a C program for later simulating and debugging.

Proposal: Start by creating a new AVR GCC C project and write this C-code (or copy/paste from the file “samplecode.c” at Blackboard):

```
/*******************************************
  AMS, LAB2
  Sample code:
  Using simulator and Atmel-ICE

  Henning Hargaard, February 1, 2019
*******************************************/
#include <avr/io.h>
#define F_CPU 16000000
#include <util/delay.h>

int main()
{
unsigned char i = 0;

  DDRA = 0;    //PORTA pins are inputs (SWITCHES)
  DDRB = 0xFF; //PORTB pins are outputs (LEDs)
  while (1)
  {
    PORTB = i; //Display "i" at the LEDs
    i++;
    _delay_ms(500);
    if ((PINA & 0b10000000)==0)
      i = 0;
  }
}
```

If you forgot how to create a C project, it is all explained in LAB1.

Take a little time to study the functionality of the program.

The sample code in the program is obviously very simple and bugs can often be found simply by running at the target hardware and observing its behavior.
Of course “real programs” are often a lot more complicated.

In some cases we can benefit from adding a few lines of “test code” to make debugging easier (for example code lines to control some LEDs at certain points in the code).

However there are disadvantages associated with using this method: We make changes to the object (the program) that we want to debug (it might in some cases disturb e.g. the timing), and we have to rebuild and download the program each time, we have added new test points to the program.

To do better debugging, using an In Circuit Emulator (ICE) would be a good solution.

Part 2: Using the Atmel Studio simulator

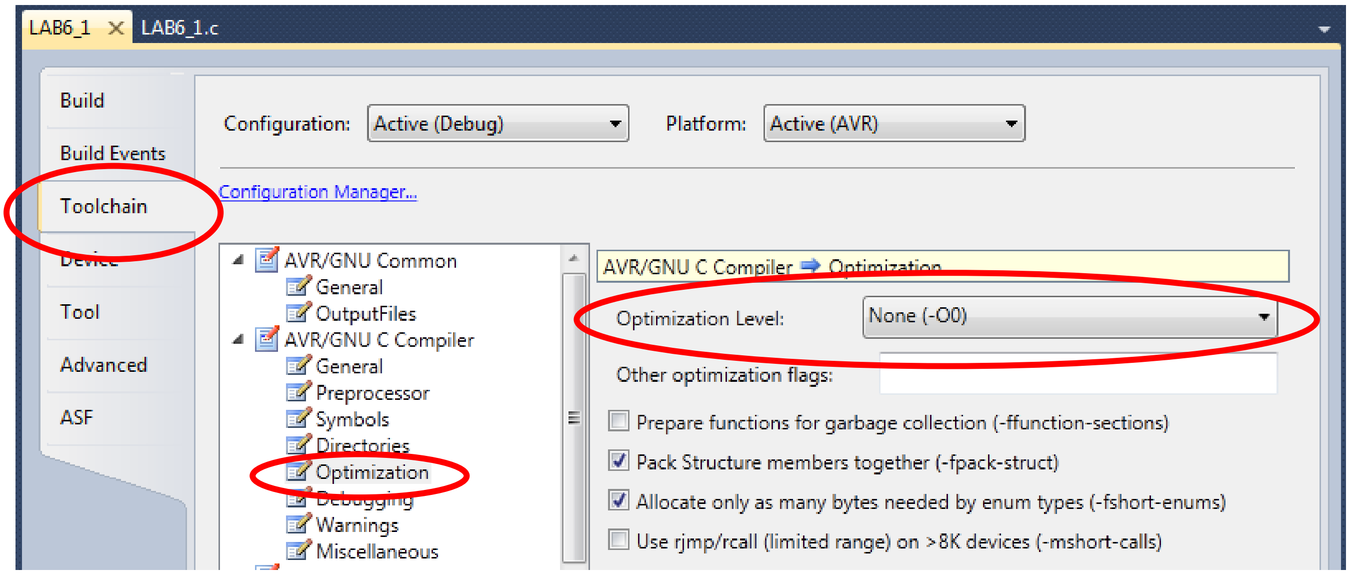
If you don't have an ICE available, an alternative method could be to simulate the program using the Atmel Studio Simulator (also called the "debugger").
The simulator is integrated in the Atmel Studio IDE.

When we use the simulator, the program execution can be simulated in Atmel Studio (not in real time and obviously you don't have to connect your target board).

The simulator "manual" is integrated in Atmel Studio Online Help ("Help" -> "View Help" -> "Atmel Studio" -> "Debugging").

A very important setting to be done, when we intend to debug/simulate the program, is to *disable the compiler optimization*.
If you don't, there will be no one-to-one correspondance between the C statement and a block of assembly code, and the simulator will execute uncorrectly!

Select "Project" -> "Properties" (or press Alt + F7).
Select the tab "Toolchain", and mark "Optimization".
Now select "Optimization Level" to "None" (eventually "Og"):



Configuration: Active (Debug)   Platform: Active (AVR)

Optimization Level: None (-O0)

Other optimization flags:

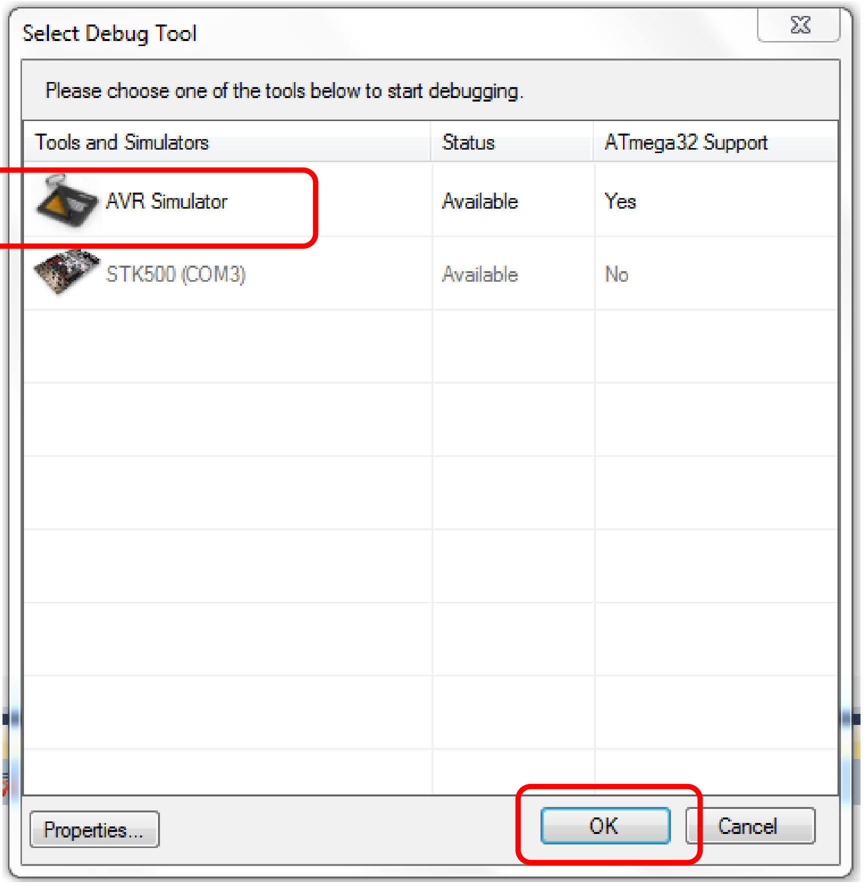
☐ Prepare functions for garbage collection (-ffunction-sections)
☑ Pack Structure members together (-fpack-struct)
☑ Allocate only as many bytes needed by enum types (-fshort-enums)
☐ Use rjmp/rcall (limited range) on >8K devices (-mshort-calls)

This link: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html will tell you the differences between the possible Optimization Levels.

Remember after debugging, normally the Optimization level should be set (different fra "None") to generate effective code.

Select "Debug" -> "Start Debugging and Break" (or press keys ALT + F5).

If this windows appears:



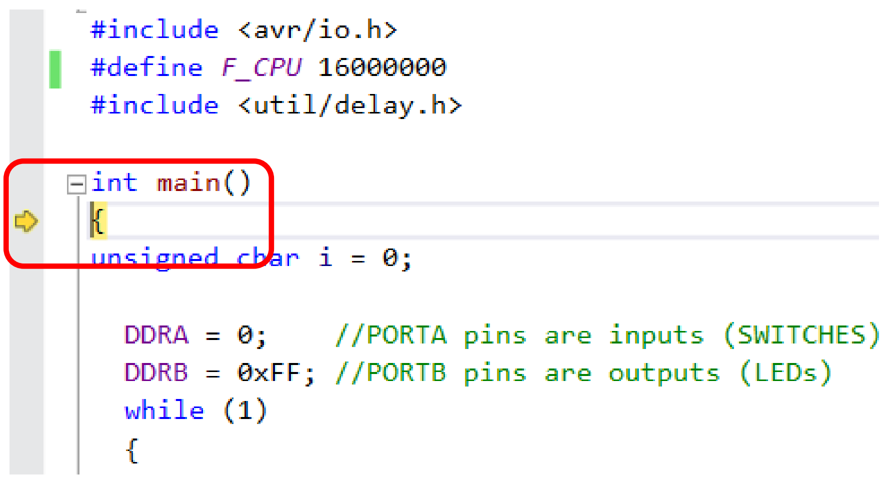
Select Debug Tool

Please choose one of the tools below to start debugging.

Tools and Simulators	Status	ATmega32 Support
AVR Simulator	Available	Yes
STK500 (COM3)	Available	No

- then mark "AVR Simulator" and click "OK".

Now several windows open, that can be used during the simulation process. In the code window a yellow arrow will indicate the next C statement ready to be executed (in this case the beginning of the main function):

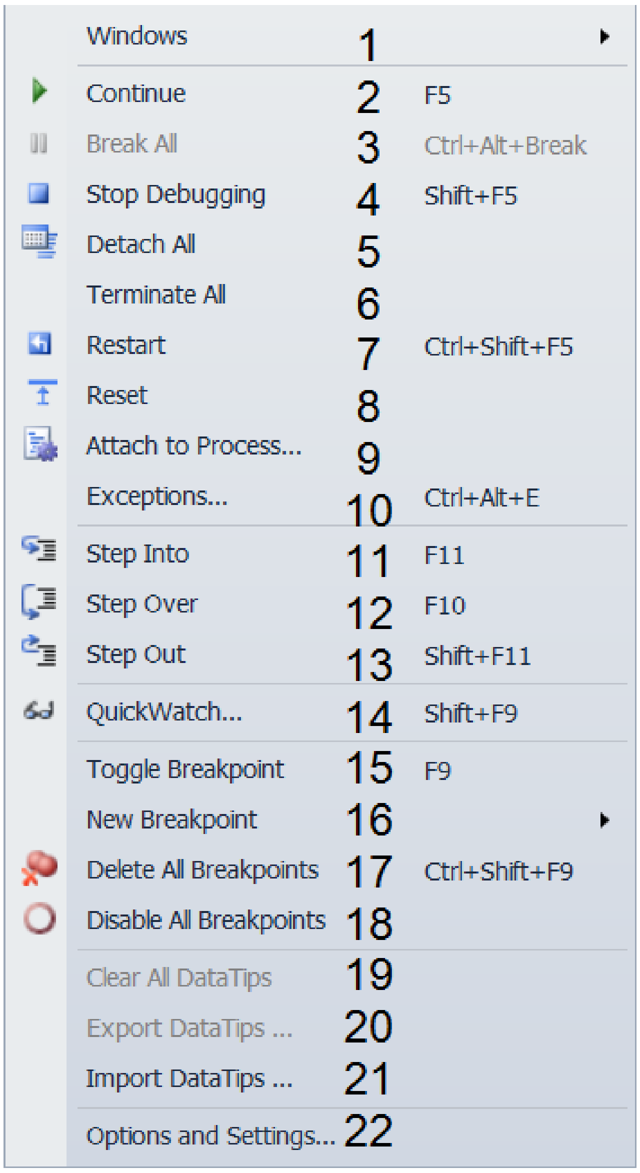


```
#include <avr/io.h>
#define F_CPU 16000000
#include <util/delay.h>

int main()
{
unsigned char i = 0;

  DDRA = 0;     //PORTA pins are inputs (SWITCHES)
  DDRB = 0xFF; //PORTB pins are outputs (LEDs)
  while (1)
  {
```

You can now simulate the program execution in many ways:



Windows	1	▸
Continue	2	F5
Break All	3	Ctrl+Alt+Break
Stop Debugging	4	Shift+F5
Detach All	5	
Terminate All	6	
Restart	7	Ctrl+Shift+F5
Reset	8	
Attach to Process...	9	
Exceptions...	10	Ctrl+Alt+E
Step Into	11	F11
Step Over	12	F10
Step Out	13	Shift+F11
QuickWatch...	14	Shift+F9
Toggle Breakpoint	15	F9
New Breakpoint	16	▸
Delete All Breakpoints	17	Ctrl+Shift+F9
Disable All Breakpoints	18	
Clear All DataTips	19	
Export DataTips ...	20	
Import DataTips ...	21	
Options and Settings...	22	

The most important functions used for simulation are:

Step Into The most simple is ”Single Step” (= ”Step Into”) , where *one* statement at the yellow arrow, will be executed (F11 key or click at).

Stop Debugging Stops the debugger.

Reset Resets the ”program counter”. The yellow arrow will move to the first statement.

Continue Starts ”execution” of the program (= F5). Can be stopped again using ”Break All”.

Break All Stops the ”program execution” (if started by F5).

If you wish the program “execution” to be continuous, but to stop at a specific statement, simply place the cursor at the instruction and press (Cntl + F10). Alternatively right-click at the statement and select “Run to Cursor”.

Be aware that the program is ”just” simulated.
This will take much more time than executing the program on the microcontroller.

You also have the possibility to set one or more ”breakpoints” in the code.
If a breakpoint is reached during simulation, the program will “pause”.
Breakpoint will normally be used in conjunction with “Continue” = F5.

Breakpoint can be inserted (or deleted) by placing the cursor at the relevant instruction and right-click.

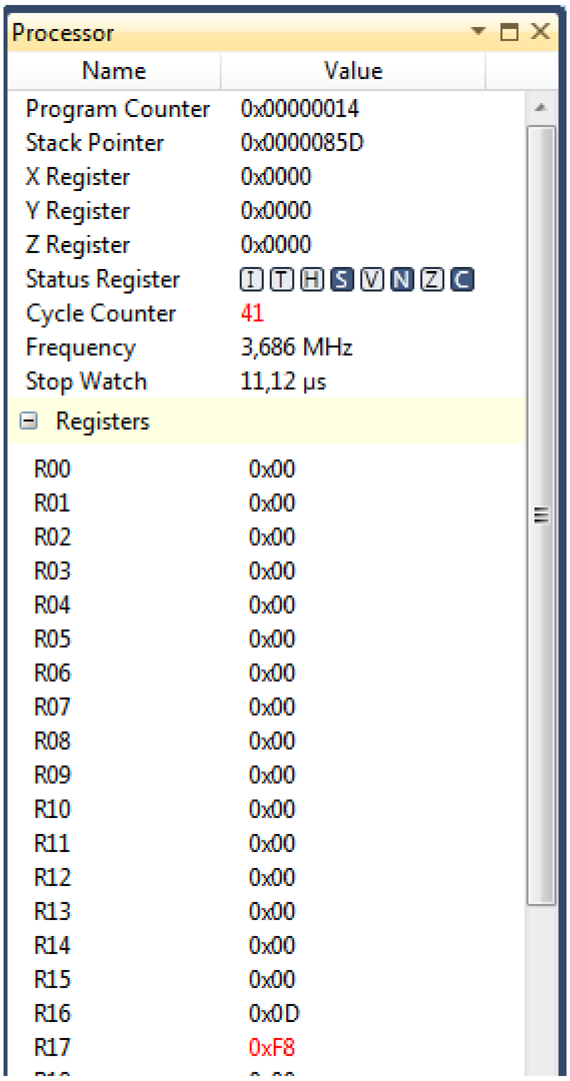
To insert a breakpoint (●), select ”Breakpoint” -> ”Insert Breakpoint”.
To delete a breakpoint, select ”Breakpoint” -> ”Delete Breakpoint”.

If you want to delete all breakpoint, select Delete All Breakpoints from the debug menu.

When simulating, not only the sequenze of program execution is interesting.

Especially when debugging assembly programs, one often wants to watch the register contents – for example when single stepping the program.

The register contents can be watched in the window ”Processor” (click at the ”+ “ in front of the word “Registers”):



Name	Value
Program Counter	0x00000014
Stack Pointer	0x0000085D
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	41
Frequency	3,686 MHz
Stop Watch	11,12 µs
⊟ Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x0D
R17	0xF8

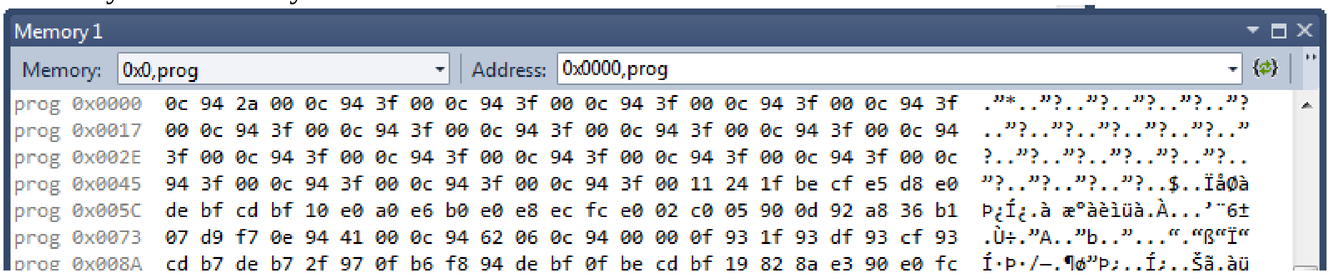
At the time a register contents has been changed by an instruction, it will be marked with a red color.

The ”Program counter” also can be seen in this window.

”Stop Watch” in the window shown the exact time elapsed since program execution started (if it was executed in the microcontroller – and not simulated). The stopwatch can be zeroed at any time by right-clicking it and select “Reset Stopwatch”.
The stopwatch is very useful for analyzing time delays in the program.

For the stopwatch to function properly, the CPU clock frequency has to be set to the right value (16 MHz for matching the Arduino Mega2560). Write this value directly to the field ”Frequency”.

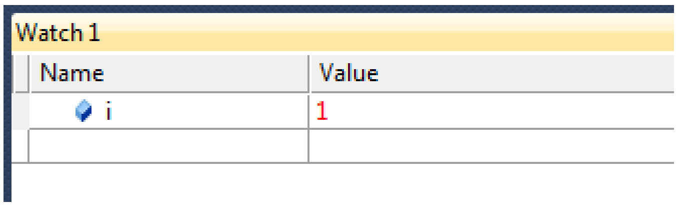
You will also be able to monitor the contents of all sort of processor memory (SRAM, registers, EPROM, Flash) using the memory window. It appear, when you select ”Debug” -> ”Windows” -> ”Memory” -> ”Memory1”:



```
prog 0x0000  0c 94 2a 00 0c 94 3f 00 0c 94 3f 00 0c 94 3f 00 0c 94 3f 00 0c 94 3f  .”*..”?..”?..”?..”?..”?
prog 0x0017  00 0c 94 3f 00 0c 94 3f 00 0c 94 3f 00 0c 94 3f 00 0c 94 3f 00 0c 94  ..”?..”?..”?..”?..”?..”
prog 0x002E  3f 00 0c 94 3f 00 0c 94 3f 00 0c 94 3f 00 0c 94 3f 00 0c 94 3f 00 0c  ?..”?..”?..”?..”?..”?..
prog 0x0045  94 3f 00 0c 94 3f 00 0c 94 3f 00 0c 94 3f 00 11 24 1f be cf e5 d8 e0  ”?..”?..”?..”?..$..ÏåØà
prog 0x005C  de bf cd bf 10 e0 a0 e6 b0 e0 e8 ec fc e0 02 c0 05 90 0d 92 a8 36 b1  Þ¿Í¿.à æ°àèìüà.À...’¨6±
prog 0x0073  07 d9 f7 0e 94 41 00 0c 94 62 06 0c 94 00 00 0f 93 1f 93 df 93 cf 93  .Ù÷.”A..”b..”...“.“ß“Ï“
prog 0x008A  cd b7 de b7 2f 97 0f b6 f8 94 de bf 0f be cd bf 19 82 8a e3 90 e0 fc  Í·Þ·/—.¶ø”Þ¿.¾Í¿..Šã.àü
```

Here the ”Program memory” is displayed. Others can be selected.

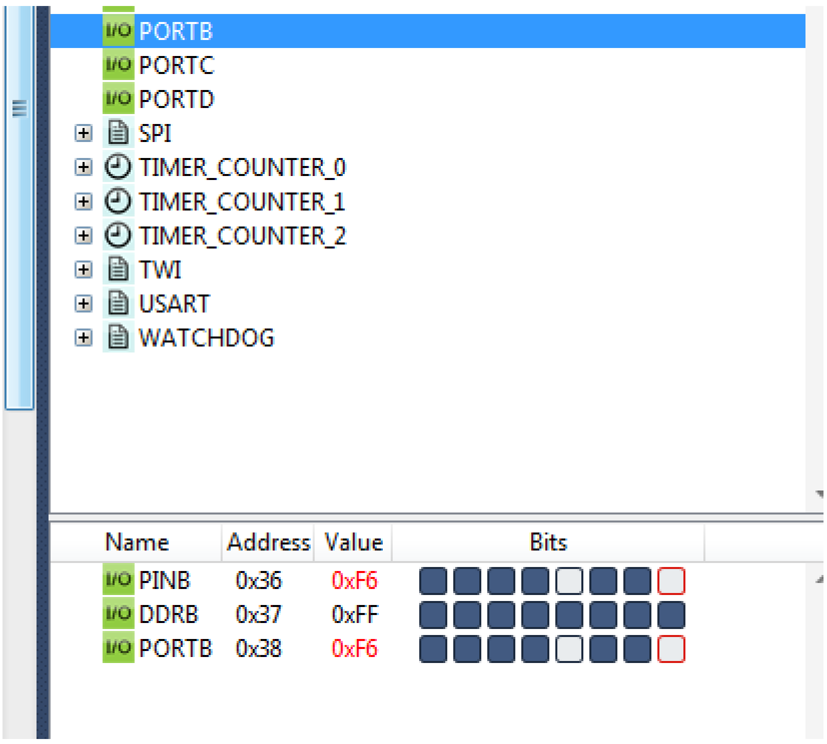
A smart way of monitoring specific variables and registers is to use the ”Watch Window”.
It is displayed by selecting ”Debug” -> ”Windows” -> ”Watch” -> ”Watch1”:



Name	Value
i	1

The elements (in this example the variable i), that you want to monitor, can be added to the window by writing the variable name – or you can simply add them to the window using ”drag-and-drop” from the code window.

To monitor the I/O registers of the microcontroller, a good way is to use the window ”I/O View” (select ”Debug” -> ”Windows” -> ”I/O View”):

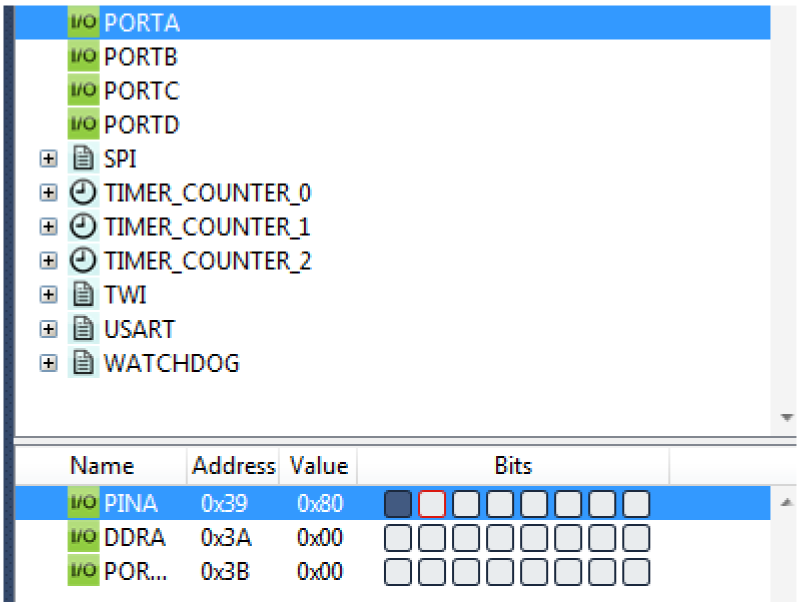

PORTB
PORTC
PORTD
SPI
TIMER_COUNTER_0
TIMER_COUNTER_1
TIMER_COUNTER_2
TWI
USART
WATCHDOG

Name	Address	Value	Bits
PINB	0x36	0xF6	■■■■□■■□
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0xF6	■■■■□■■□

In the example above, we can see that PORTB has the binary value 11110110 (the dark boxes are 1’s and the white are 0’s).

Notice, that you have the possibility to change the individual bits in the I/O registers - simply by clicking at them.

Actually you will need to use this feature for debugging our sample code, since the program reads PINA to read the status of pushbutton 7 (switch SW7 at the STK500 board). Since we simulate the program, we have to <u>simulate the pushbuttons</u> by changing the individual bits in the PINA register:


PORTA
PORTB
PORTC
PORTD
SPI
TIMER_COUNTER_0
TIMER_COUNTER_1
TIMER_COUNTER_2
TWI
USART
WATCHDOG

Name	Address	Value	Bits
PINA	0x39	0x80	■□□□□□□□
DDRA	0x3A	0x00	□□□□□□□□
POR...	0x3B	0x00	□□□□□□□□

Spend some time experimenting with the simulator to investigate the sample code execution.
Then continue with the next part, using the more advanced debug tool, the Atmel-ICE.

The simulator has many other, advanced features than mentioned in this LAB exercise.

You can read more by selecting the Atmel Studio menu ”Help” -> ”View Help” -> “Atmel Studio” -> ”Debugging”.

Part 3: Using the Atmel-ICE
In this part of the exercise we will use the Atmel -ICE for debugging the program while it is executing in target.

This is possible, since the Mega2560 has an on chip hardware JTAG interface.
Not all AVR devices have JTAG interfaces, but many have.

The Atmel-ICE Users Guide is available at AMS Blackboard.
Start by reading/skimming the User's Guide.
All the stuff about connecting is not very relevant, BUT table 4-10 at page 37 is important.

The ICE uses the same IDE ("frontend") for debugging as the simulator does (refer to part 2).
The great advantage is that the program is executed in real time (only with a very few exceptions) at the target microcontroller (even with ability to debug it).

One disadvance is that some microcontroller port pins are allocated for the JTAG interface.
The dedicated Mega2560 JTAG interface pins are 4 pins of PORTF.

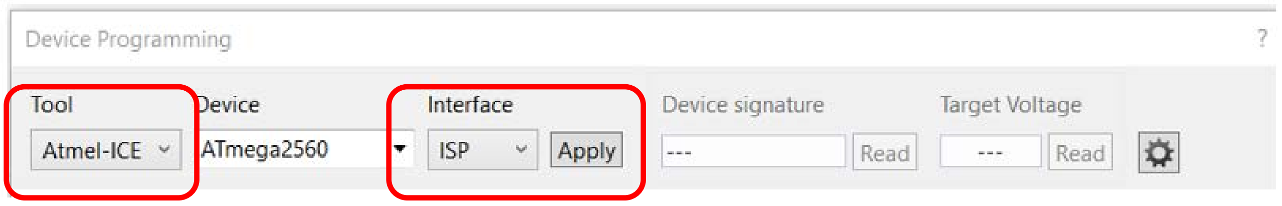
IMPORTANT: Before you can use the JTAG interface of the microcontroller (the Mega2560), a fuse internal the chip has to be set to enable the JTAG interface. This can only be done using ISP programming:

Connect the ISP 6 pin connector of the ICE to the Mega2560 ISP connector (put it the right way, meaning the plastic tab of the connector must be in the direction of the Mega2560 chip).

Connect both the Arduino and the ICE to USB ports of you PC.

When you connect the ICE, its driver normally will be automatically installed.

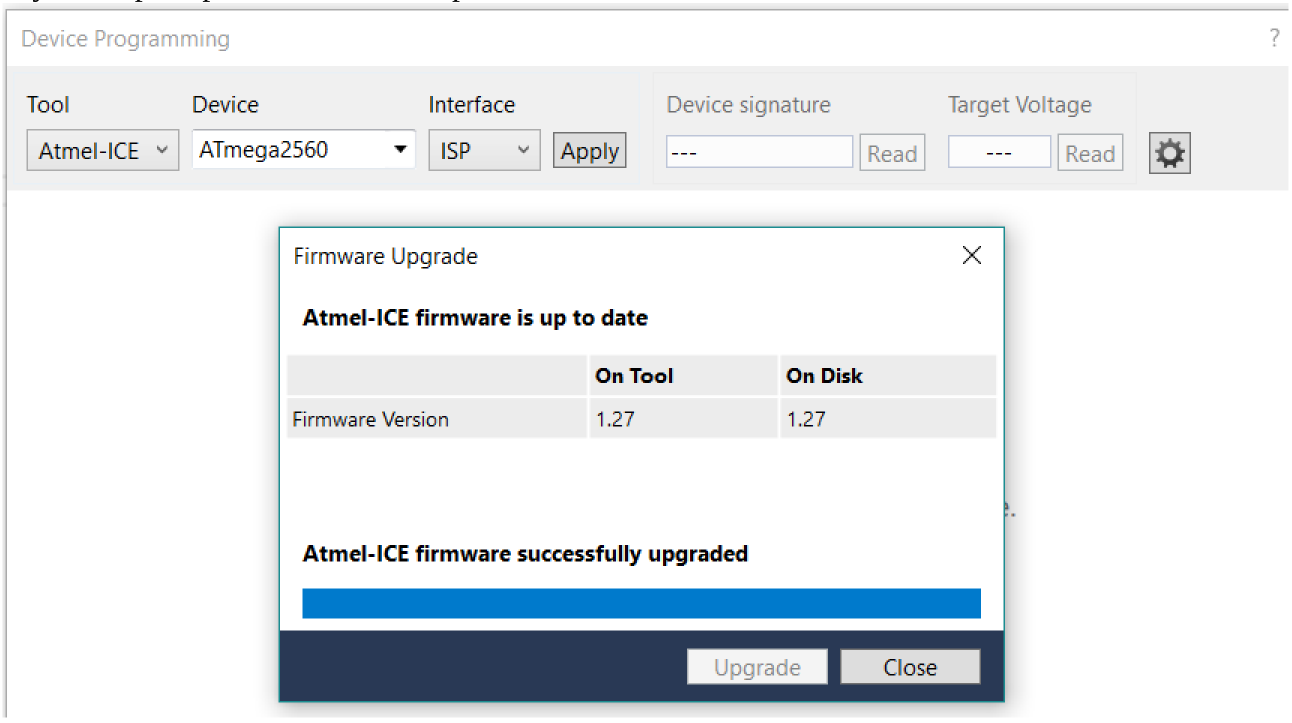
In Atmel Studio, go to "Device Programming" and select the Atmel-ICE as tool and the interface as ISP.:



Device Programming

Tool: Atmel-ICE
Device: ATmega2560
Interface: ISP
Device signature: ---
Target Voltage: ---

If you are prompted for firmware update, do that:



Device Programming

Tool: Atmel-ICE
Device: ATmega2560
Interface: ISP
Device signature: ---
Target Voltage: ---

Firmware Upgrade

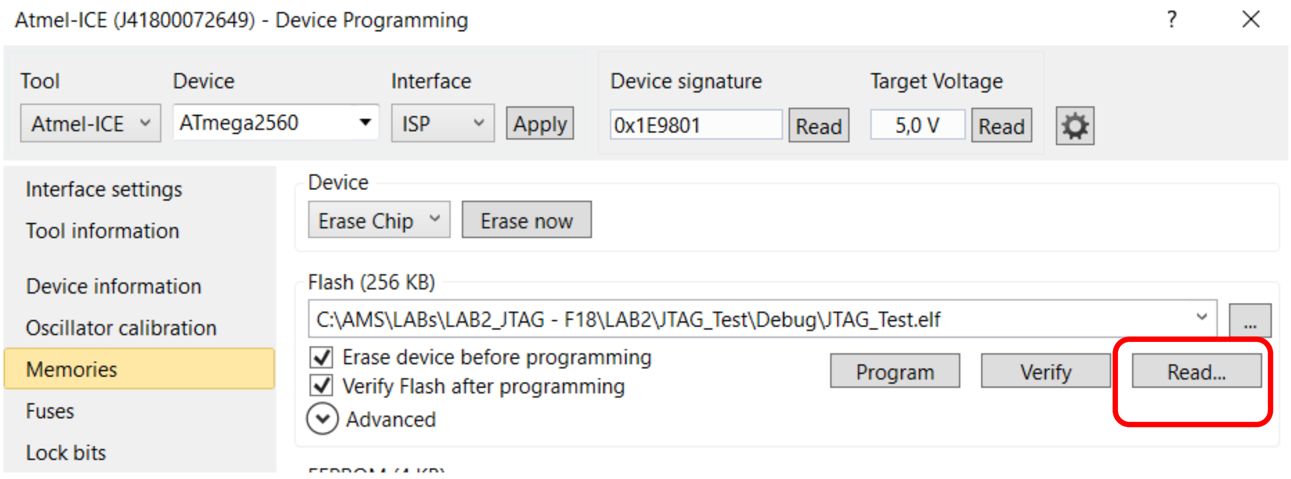
Atmel-ICE firmware is up to date

	On Tool	On Disk
Firmware Version	1.27	1.27

Atmel-ICE firmware successfully upgraded

Your Arduino board comes with a boot loader installed. When using the ICE, we will probably erase the boot loader – obviously not a wise thing to do.

Therefore, start by reading the flash content of your board to a hex file (call it "Mega2560.hex"), and store the file at your PC. Later we will use this file when restoring the boot loader.

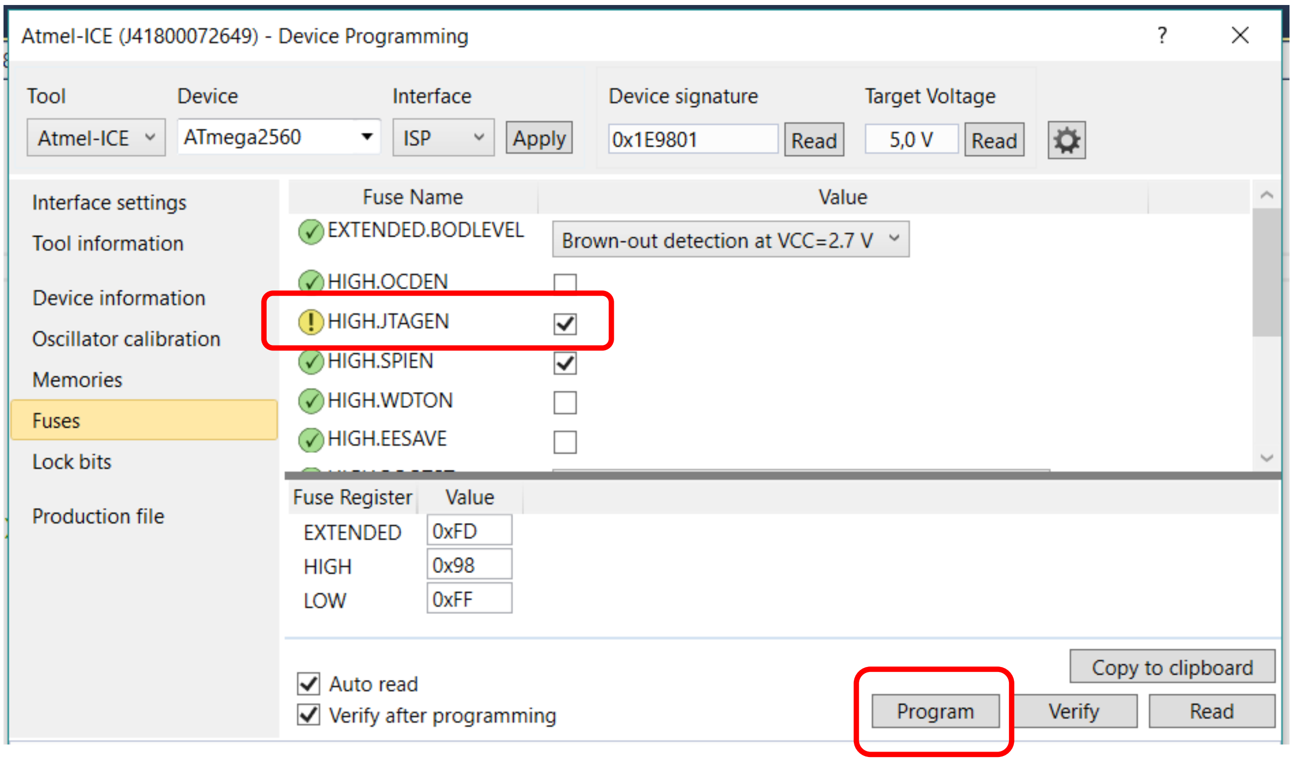


Device signature: 0x1E9801
Target Voltage: 5,0 V

Flash (256 KB)
C:\AMS\LABs\LAB2_JTAG - F18\LAB2\JTAG_Test\Debug\JTAG_Test.elf

☑ Erase device before programming
☑ Verify Flash after programming

Now go to tab “Fuses” and check “JTAGEN”. Click Program.



Device signature: 0x1E9801
Target Voltage: 5,0 V

Fuse Name	Value
EXTENDED.BODLEVEL	Brown-out detection at VCC=2.7 V
HIGH.OCDEN	☐
HIGH.JTAGEN	☑
HIGH.SPIEN	☑
HIGH.WDTON	☐
HIGH.EESAVE	☐

Fuse Register	Value
EXTENDED	0xFD
HIGH	0x98
LOW	0xFF

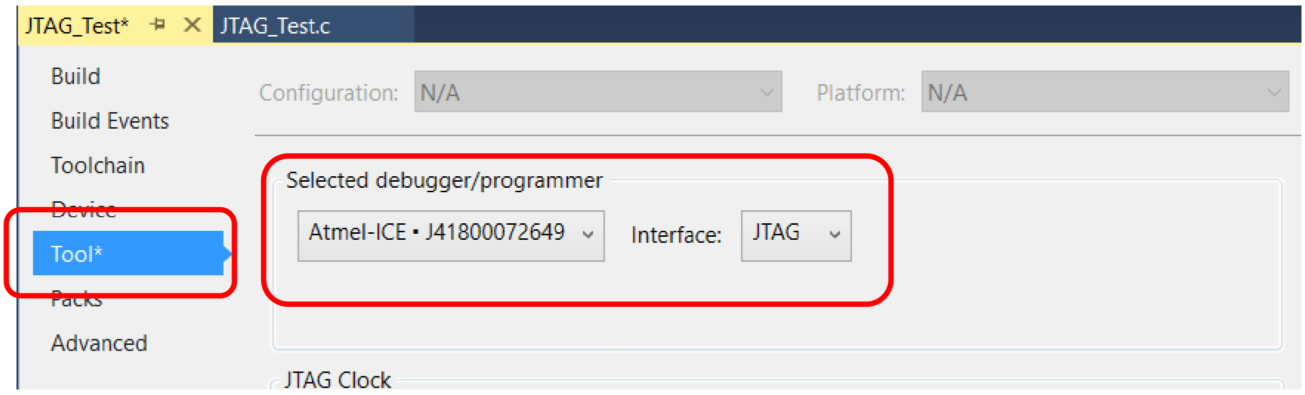
☑ Auto read
☑ Verify after programming

If you get a warning: Continue.

Notice:
After having enabled the JTAG interface of the microcontroller, 4 pins of PORTF will be reserved for JTAG communication (and therefore are not available for general purpose I/O any more).
For this reason: Remember to unchecke the JTAGEN fuse after having finished debugging, <u>if</u> you intend to use the pins in your application.

Set up Atmel Studio to use the Atmel-ICE for debugging (instead of the simulator that we used in part2). Go to the project properties (Alt + F7).

Go to the “Tool” tab, select the Atmel-ICE as debugger and set interface to “JTAG”:



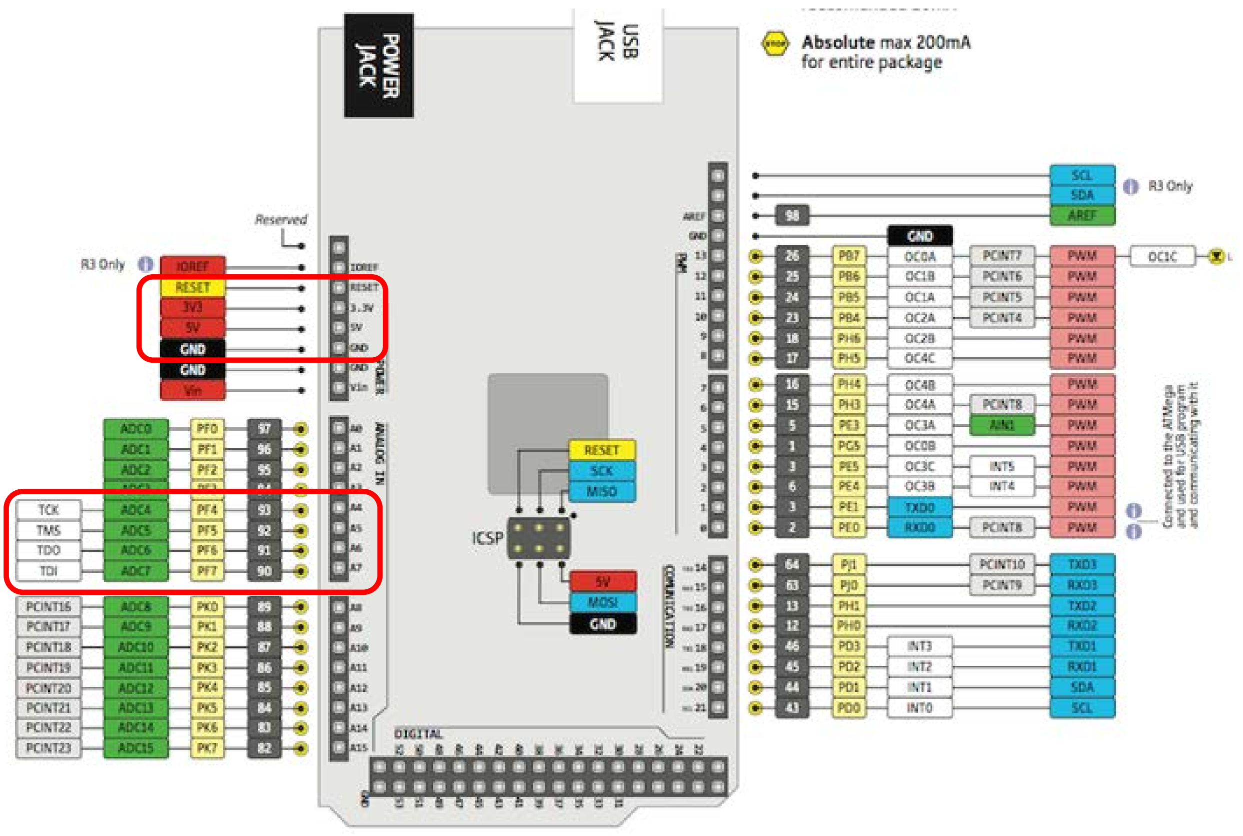
Selected debugger/programmer: Atmel-ICE • J41800072649
Interface: JTAG

Now connect the mini squid cable of the ICE (the one with single number labeled sockets) to the JTAG pins of Mega2560. Use some simple pins (male-male) to connect the squid cable to the sockets of the Arduino board / the IO shield. The RESET connection is optional.

Signal	ICE: AVR port pin	Mega2560 pin
TCK	1	PORTF, 4
TMS	5	PORTF, 5
TDO	3	PORTF, 6
TDI	9	PORTF, 7

Signal	ICE: AVR port pin	Mega2560 pin
(TRST)	(8)	(RESET)
Vcc	4	5V
GND	2	GND

The “Arduino Mega250” pinout is shown at the next page.

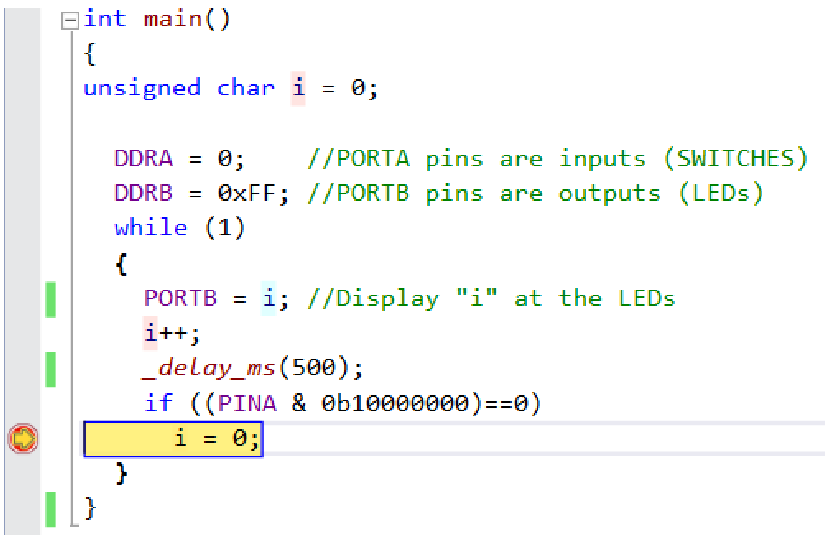


You can now use the JTAG ICE for downloading programs and for debugging purposes using "AVR Programming" (as we use to do).

Play around with the JTAG ICE and debug the program.

Do experiments similar to the experiments you did using the simulator. In this case you should of cause notice that the program is <u>executing at the target hardware</u> (not simulated any more).

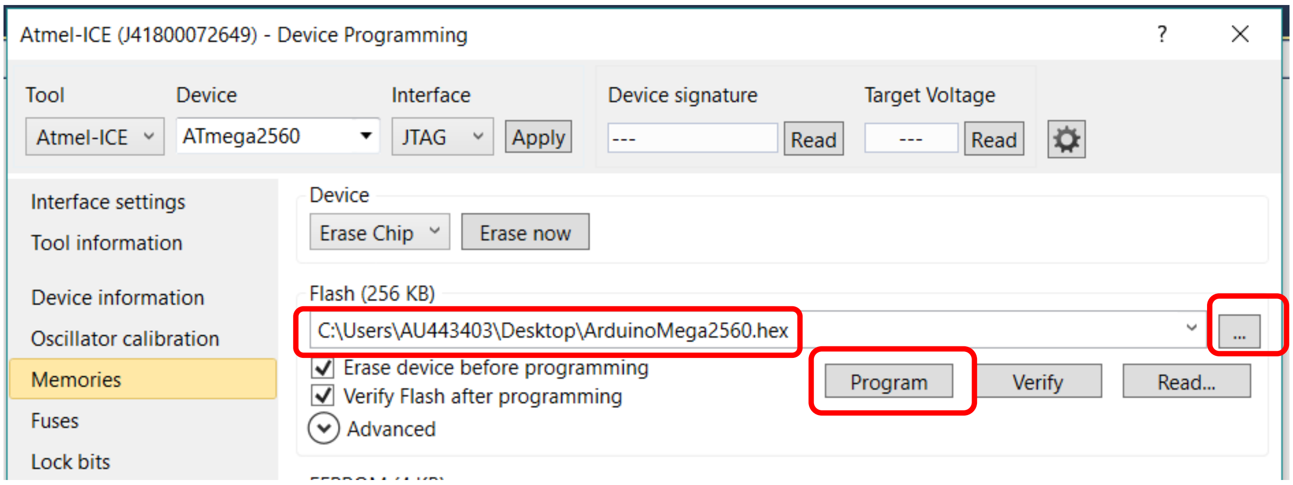
Try setting a breakpoint as shown underneath and press pushbutton SW7 as the program is debugged and running. You will see the debugger reacts upon the physical event.



```
int main()
{
unsigned char i = 0;

  DDRA = 0;     //PORTA pins are inputs (SWITCHES)
  DDRB = 0xFF; //PORTB pins are outputs (LEDs)
  while (1)
  {
    PORTB = i; //Display "i" at the LEDs
    i++;
    _delay_ms(500);
    if ((PINA & 0b10000000)==0)
      i = 0;
  }
}
```

If you like to restore the Arduino with the original boot loader, simply program the device with the hex file, you saved earlier in the exercise:


Atmel-ICE (J41800072649) - Device Programming

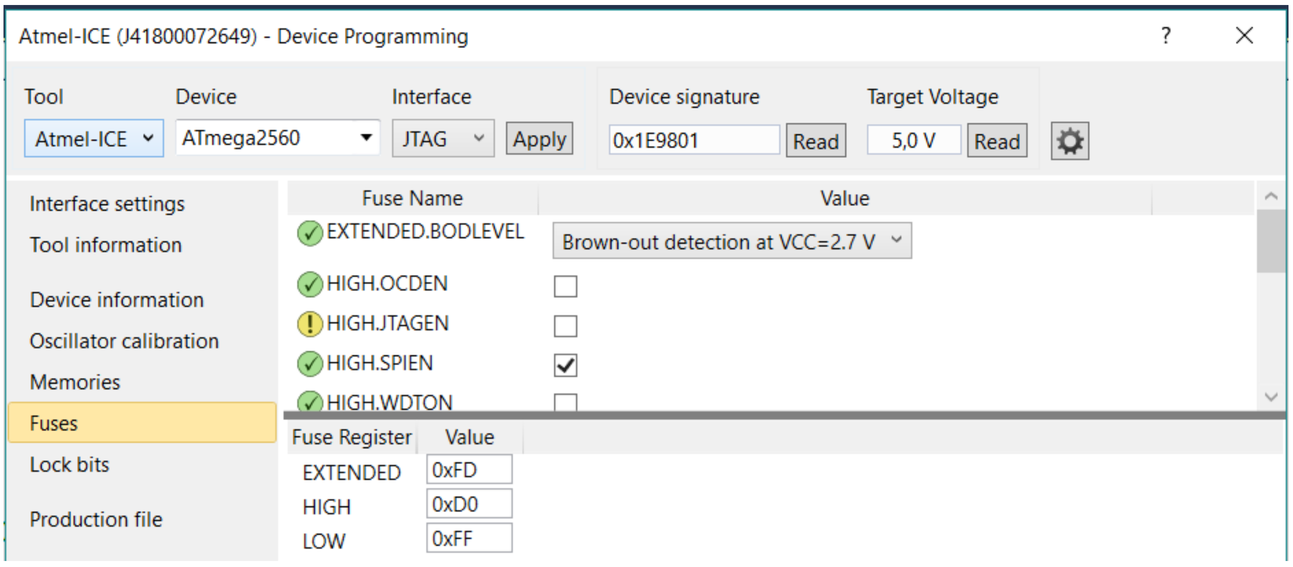
Tool: Atmel-ICE
Device: ATmega2560
Interface: JTAG
Device signature: ---
Target Voltage: ---

Interface settings
Tool information
Device information
Oscillator calibration
Memories
Fuses
Lock bits

Device: Erase Chip — Erase now

Flash (256 KB)
C:\Users\AU443403\Desktop\ArduinoMega2560.hex
☑ Erase device before programming
☑ Verify Flash after programming
Advanced

In you want to free the JTAG pins of PORT F for your application, you will have to clear the JTAG fuse:


Atmel-ICE (J41800072649) - Device Programming

Tool: Atmel-ICE
Device: ATmega2560
Interface: JTAG
Device signature: 0x1E9801
Target Voltage: 5,0 V

Interface settings
Tool information
Device information
Oscillator calibration
Memories
Fuses
Lock bits
Production file

Fuse Name	Value
EXTENDED.BODLEVEL	Brown-out detection at VCC=2.7 V
HIGH.OCDEN	☐
HIGH.JTAGEN	☐
HIGH.SPIEN	☑
HIGH.WDTON	☐

Fuse Register	Value
EXTENDED	0xFD
HIGH	0xD0
LOW	0xFF