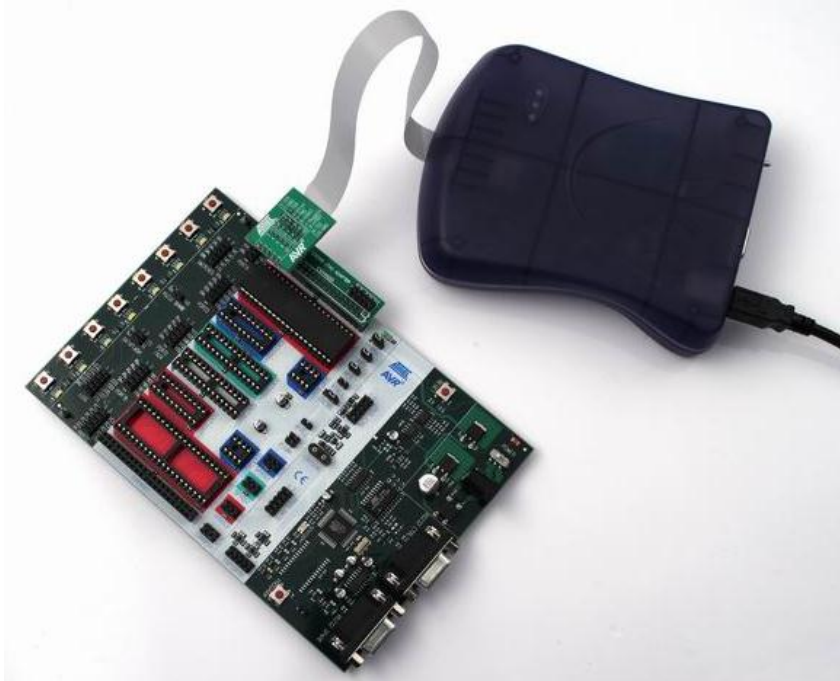


### **Purpose**

1. To be able to use the AVR Studio 5 simulator.
2. To be able to set up and use the Atmel JTAG ICE mkII for debugging purposes.



### **Material**

- The Mega32 data book (the JTAG section).
- AVR Studio 5 Online Help.

### **The exercise**

In this exercise we will:

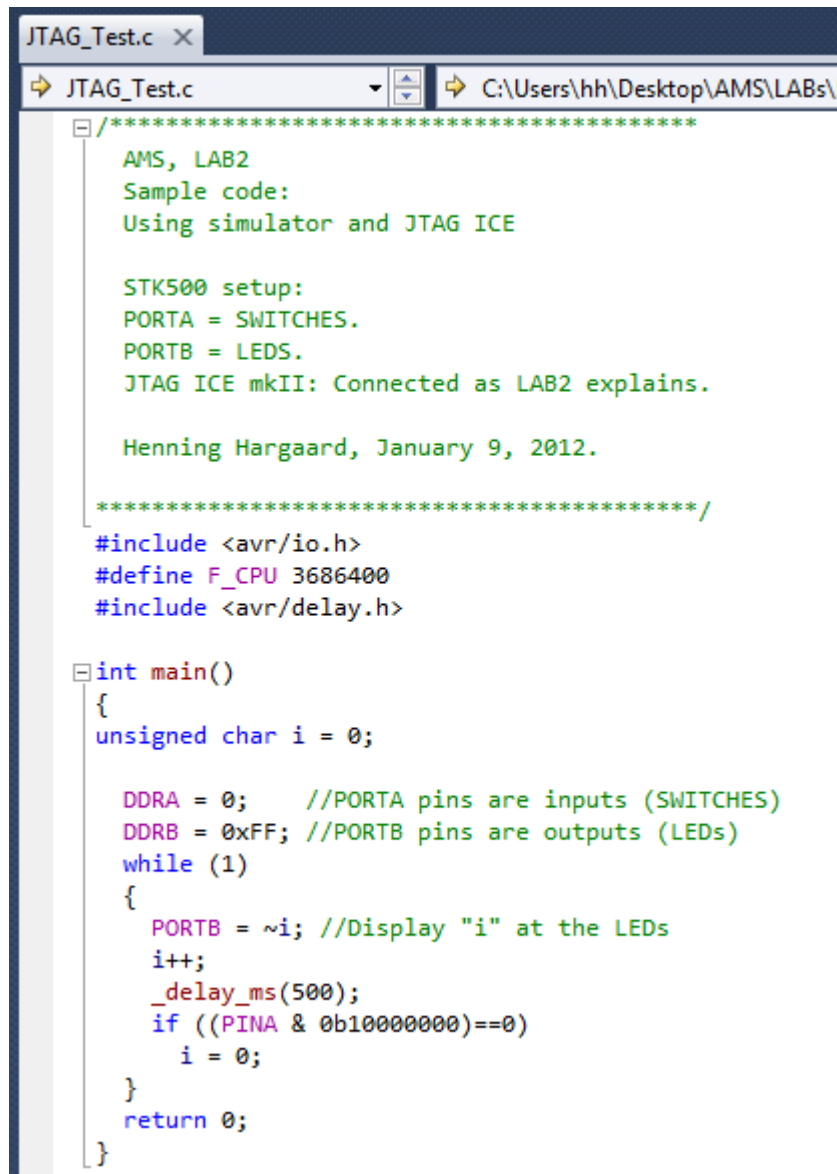
- Create an AVR GCC project and write a C program (intended for later debugging).
- Use the AVR Studio Simulator to simulate the program execution.
- Connect the JTAG In Circuit Emulator mkII to the STK500 board.
- Use the JTAG ICE for program debugging.

It is assumed that AVR Studio 5 is already installed at your PC.

#### Part 1: Write a C program in AVR Studio 5

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

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



```
JTAG_Test.c
JTAG_Test.c C:\Users\hh\Desktop\AMS\LABs\
/*****
AMS, LAB2
Sample code:
Using simulator and JTAG ICE

STK500 setup:
PORTA = SWITCHES.
PORTB = LEDS.
JTAG ICE mkII: Connected as LAB2 explains.

Henning Hargaard, January 9, 2012.

*****/
#include <avr/io.h>
#define F_CPU 3686400
#include <avr/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;
    }
    return 0;
}
```

If you forgot how to create an AVR GCC project, it is all explained in LAB1.

Take a little time to study the programs functionality.

Then download it to STK500 and test it (remember first to mount the cables for the switches and the LEDS).

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 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 proper solution.

#### Part 2: Using the AVR Studio simulator

If you don't have an ICE available, an alternative method could be to simulate the program using the AVR Studio Simulator (also called the "debugger").

The simulator is integrated in the AVR Studio IDE.

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

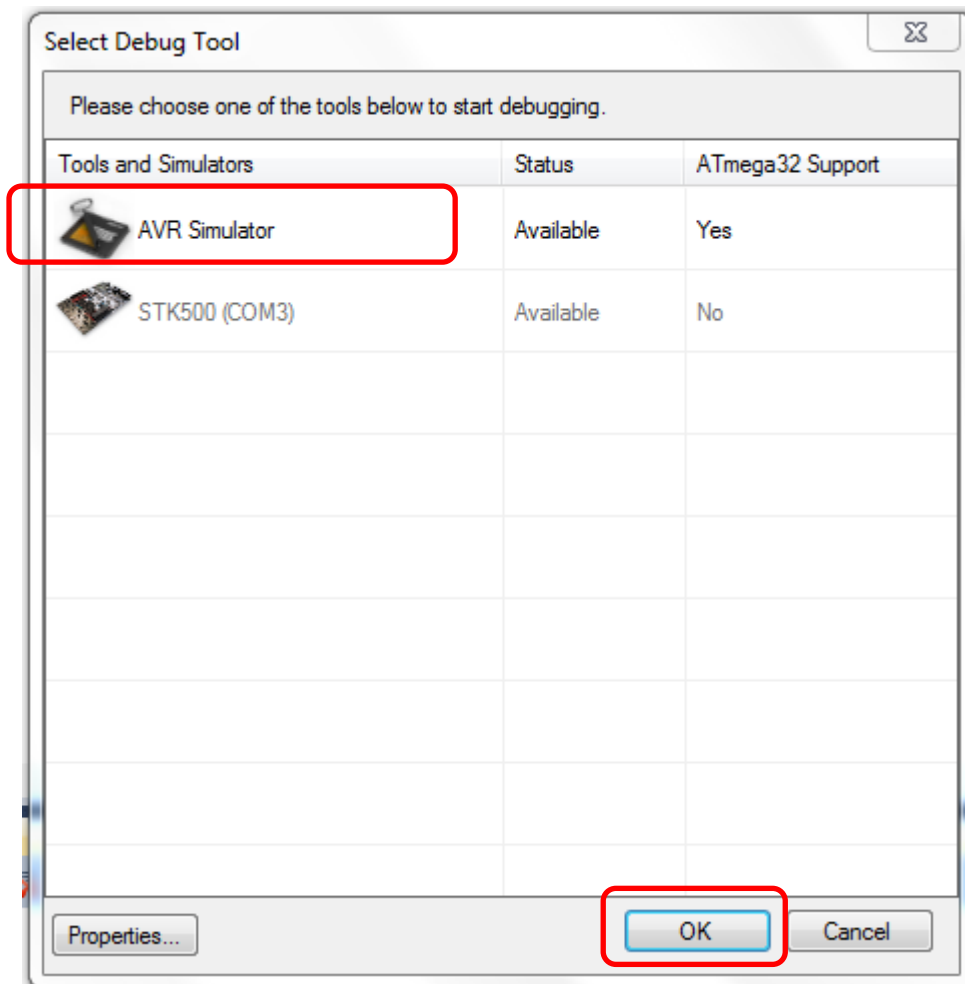
The simulator "manual" is integrated in AVR Studio 5 Online Help ("Help" -> "View Help" -> "AVR Studio User Guide" -> "Debugging in AVR Studio 5").

We are now going to simulate the program execution of the sample code.

In AVR Studio 5, build the project.

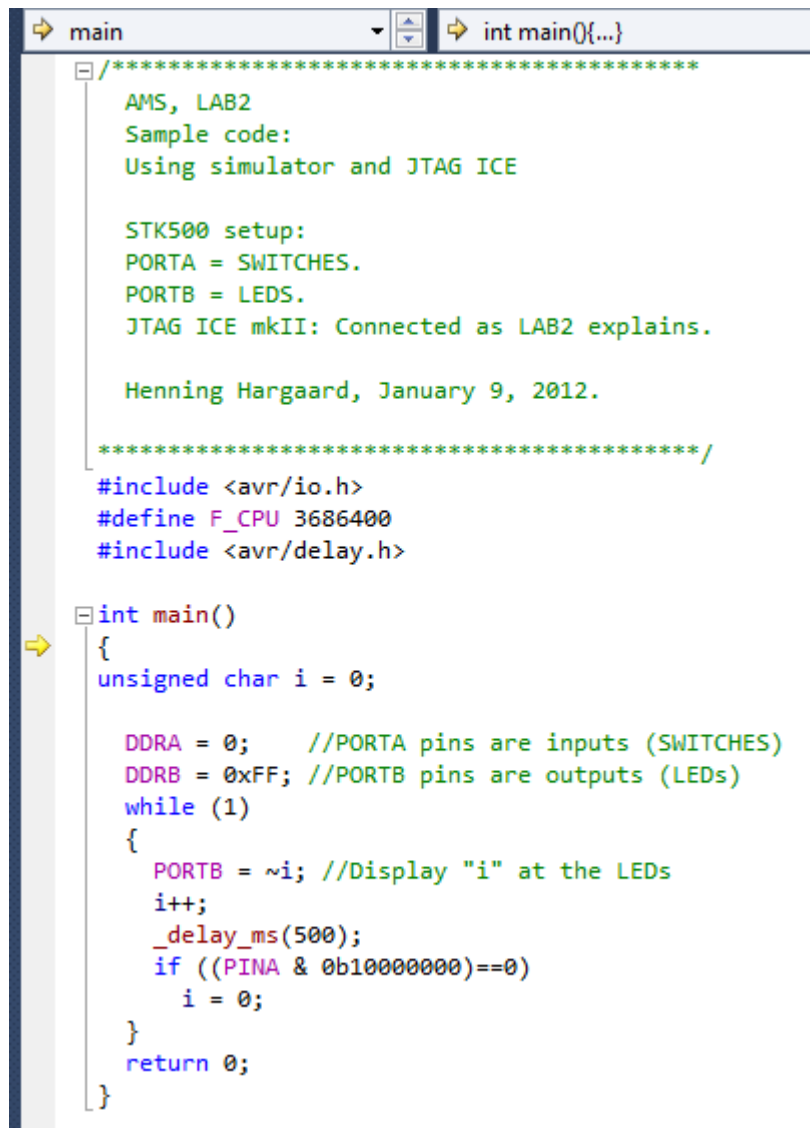
Then select "Debug" -> "Start Debugging and Break" (or press keys ALT + F5).

If this windows appears:



- then mark "AVR Simulator" og 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):



```
main ▾ int main(){...}
/*****
AMS, LAB2
Sample code:
Using simulator and JTAG ICE

STK500 setup:
PORTA = SWITCHES.
PORTB = LEDS.
JTAG ICE mkII: Connected as LAB2 explains.

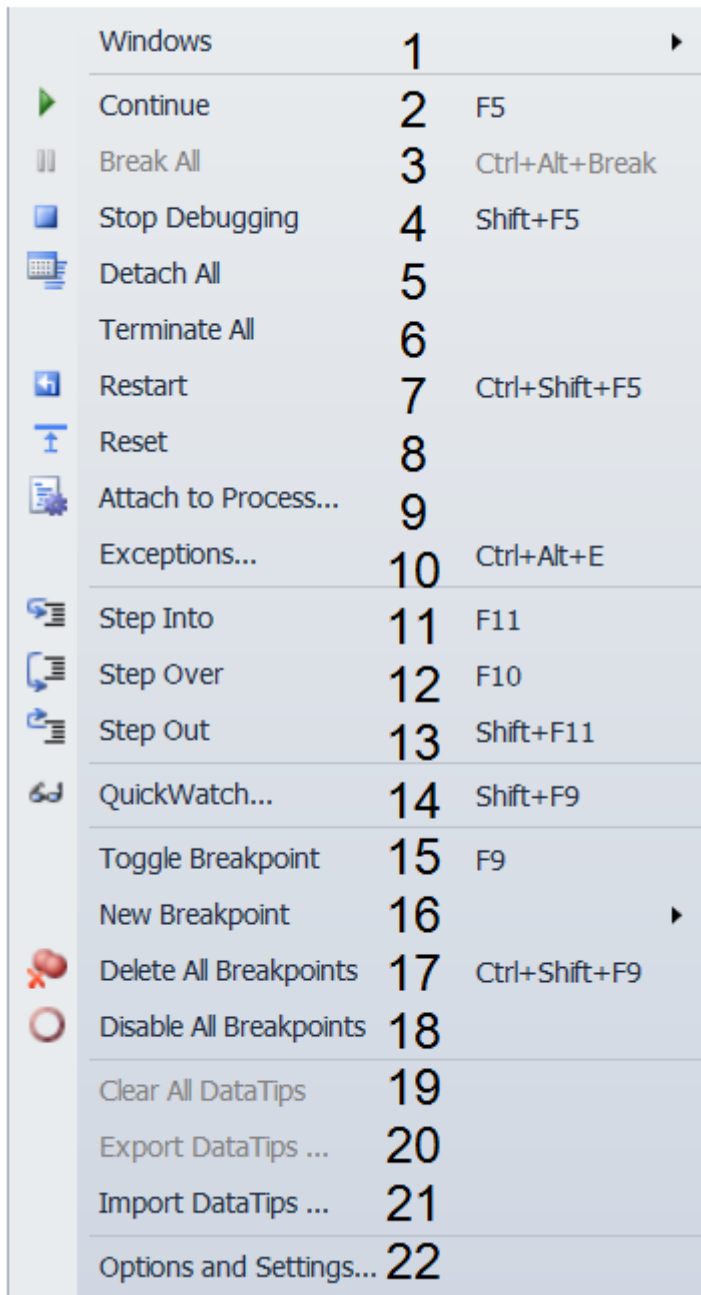
Henning Hargaard, January 9, 2012.

*****/
#include <avr/io.h>
#define F_CPU 3686400
#include <avr/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;
    }
    return 0;
}
```

You can now simulate the program execution in many ways:



The most important functions used for simulation are:

 Step Into	The most simple is "Single Step" (= "Step Into") , where <u>one</u> 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).


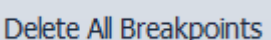
If you wish the program “execution” to be continuous, but to stop at a specific statement, simply place the cursor at the instruction and press (Cntl + F10). Alternatively right-click at the statement and select “Run to Cursor”.

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.

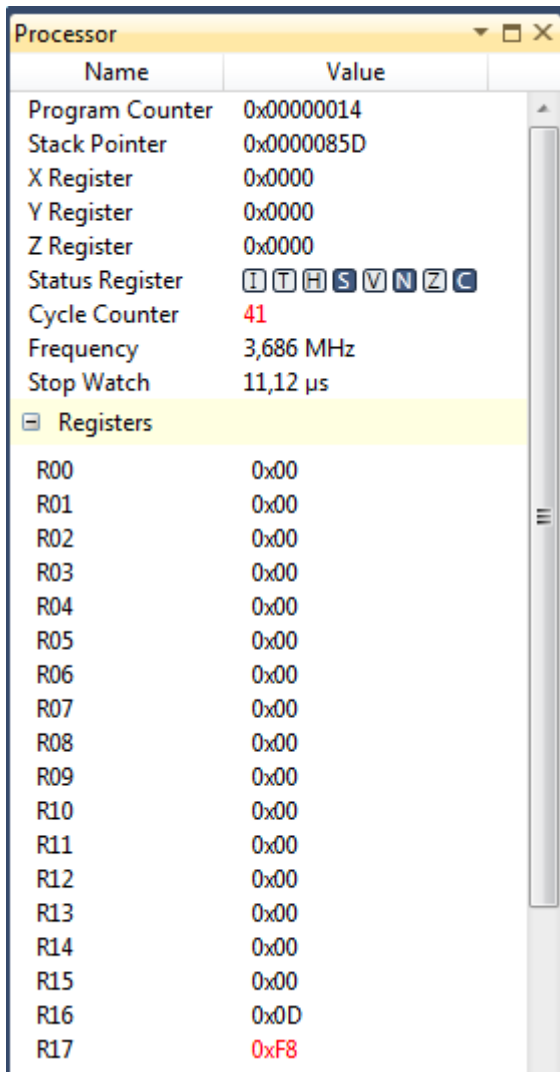
To insert a breakpoint (  ), select ”Breakpoint” -> ”Insert Breakpoint”.  
To delete a breakpoint, select ”Breakpoint” -> ”Delete Breakpoint”.

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

When simulating, not only the order 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"):



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.

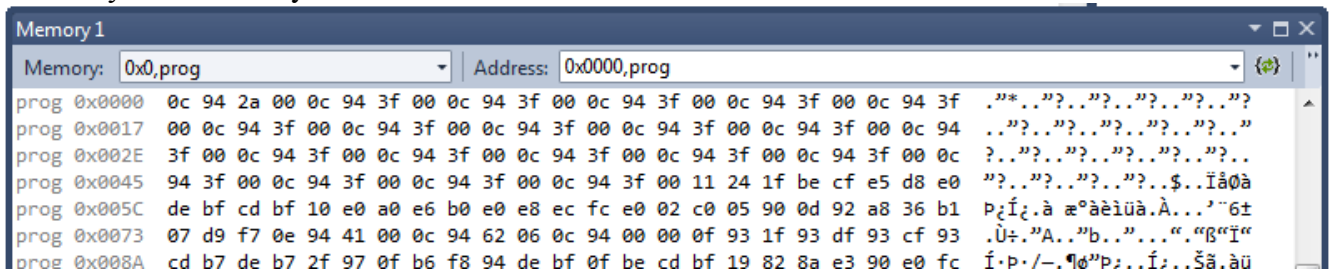
"Stop Watch" in the window shows the exact time elapsed since program execution started (if it was executed in the microcontroller – and not simulated). The stopwatch can be zeroed at any time by right-clicking it and select "Reset Stopwatch".

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 (3,686 MHz for matching the STK500). Write this value directly to the field "Frequency".

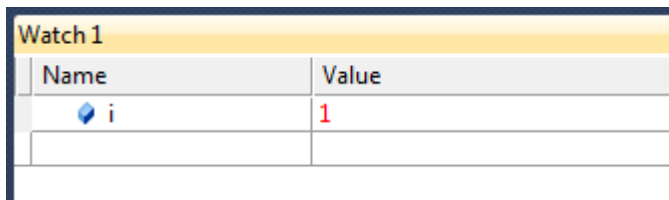


You will also be able to monitor the contents of all sort of processor memory (SRAM, registers, EEPROM, Flash) using the memory window. It appear, when you select "Debug" -> "Windows" -> "Memory" -> "Memory1":



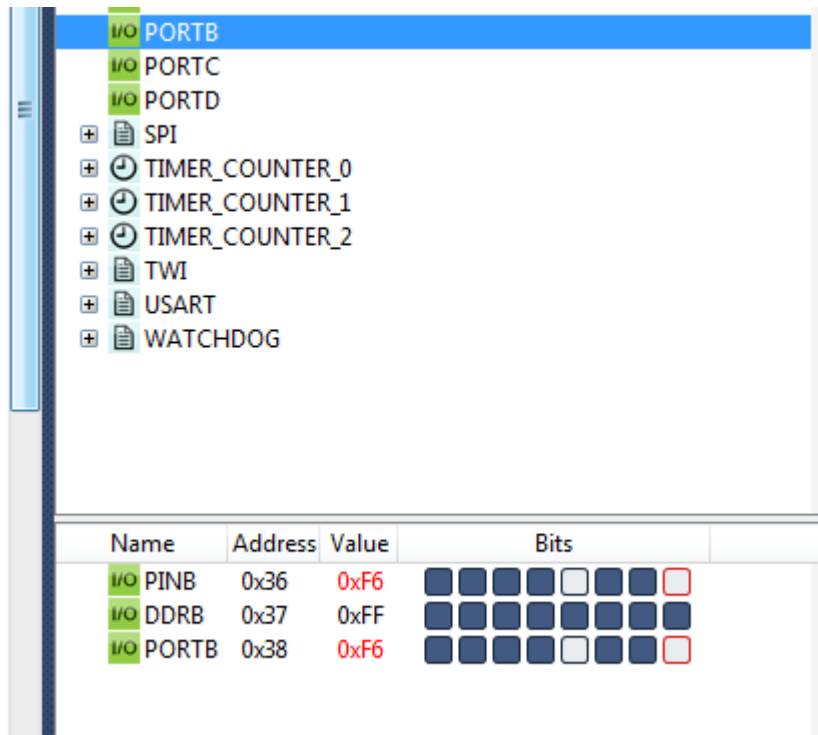
Here the "Program memory" is displayed. Others can be selected.

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



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.

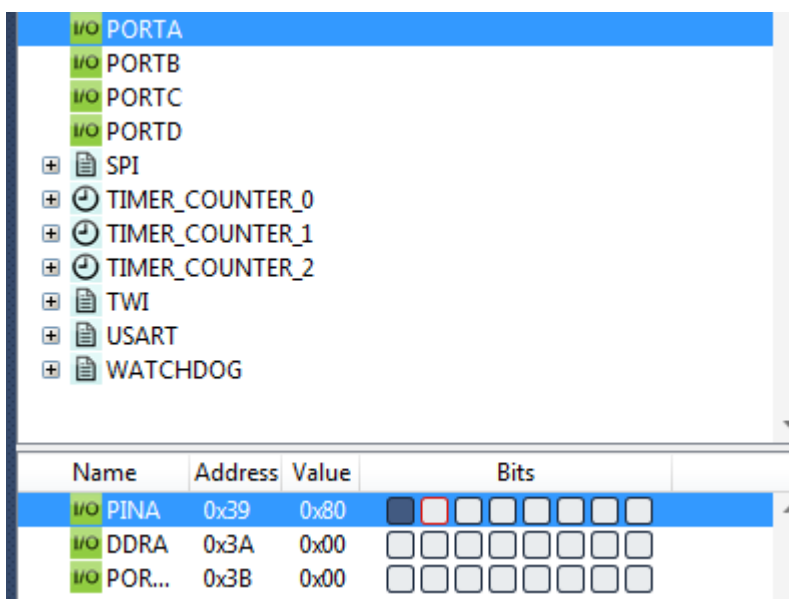
To monitor the I/O registers of the microcontroller, a good way is to use the window "I/O View" (select "Debug" -> "Windows" -> "I/O View"):



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 simulate the pushbuttons by changing the individual bits in the PINA register:



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 JTAG ICE.

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

You can read more by selecting the AVR Studio menu "Help" -> "View Help" -> "AVR Studio User Guide" -> "Debugging in AVR Studio 5".

### **Part 3: Using the JTAG ICE**

In this part of the exercise we will use the Atmel JTAG ICE mkII for debugging the program while it is executing in target.

This is possible, since the Mega32 has an on chip JTAG interface.  
Not all AVR devices have JTAG interface.



The documentation for the JTAG ICE can be found at AVR Studio 5 Online Help:  
“Help” -> “View Help” -> “JTAG ICE mkII User Guide”.

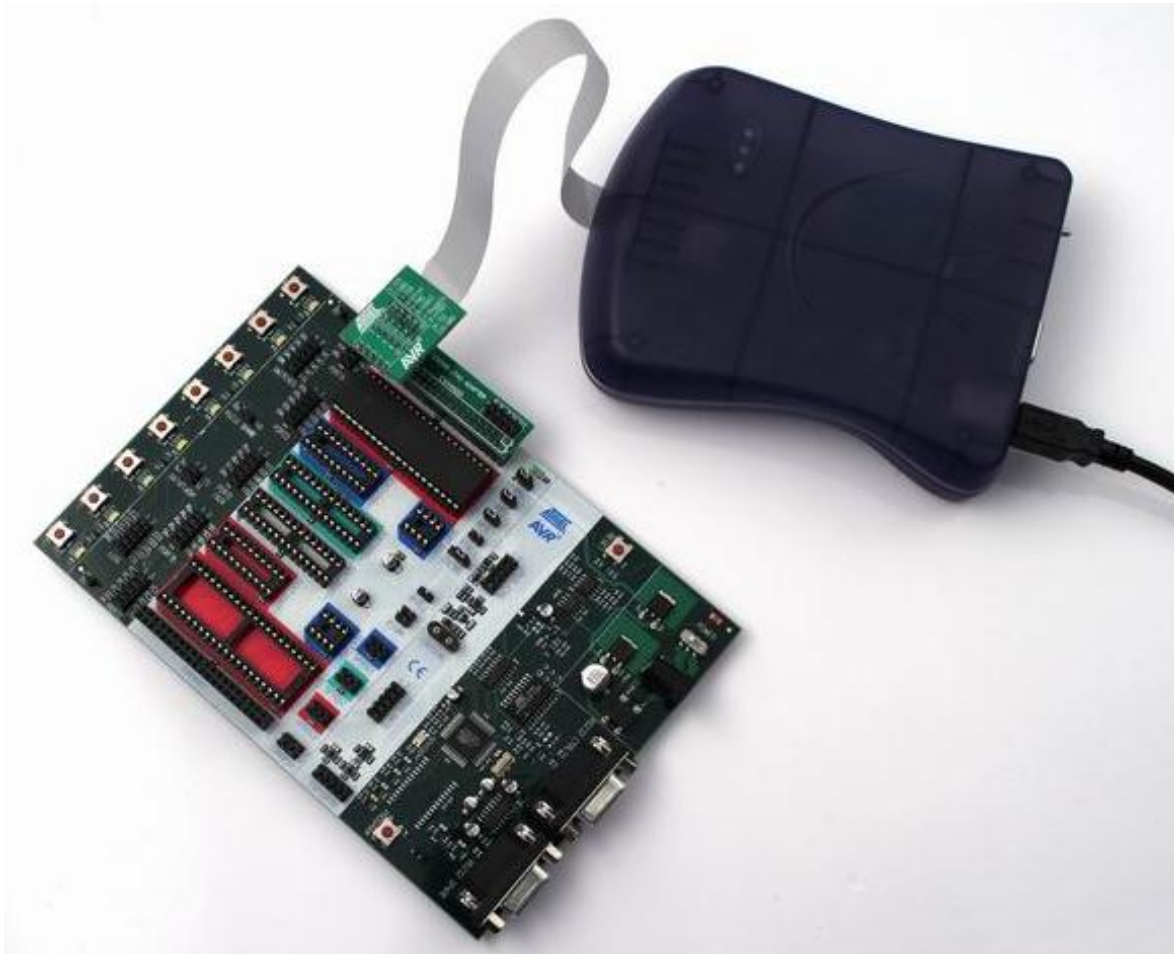
The JTAG ICE uses the same IDE (“frontend”) for debugging as the simulator does (refer to part 2).  
The great advantage is that the program is executed in real time (only with a very few exceptions) at the target microcontroller (even with ability to debug it).

The dedicated Mega32 JTAG interface pins are 4 pins of PORTC.

To be able to connect them properly to the JTAG ICE, first mount the “STK500 JTAG ADAPTER” at the STK500 connector labeled “EXPANDO” (see the figure at next page).



Connect the JTAG ICE to the STK500 via the “STK500 JTAG ADAPTER” exactly as shown:



Then connect the JTAG ICE to a USB port at your PC.

The JTAG ICE will be powered from USB (no external power supply is needed).

Turn on the JTAG ICE (power switch at device front).

The USB driver for the JTAG ICE is embedded in AVR Studio and therefore driver installation should be automatic when you USB plug the ICE.

## AMS LAB exercise 2

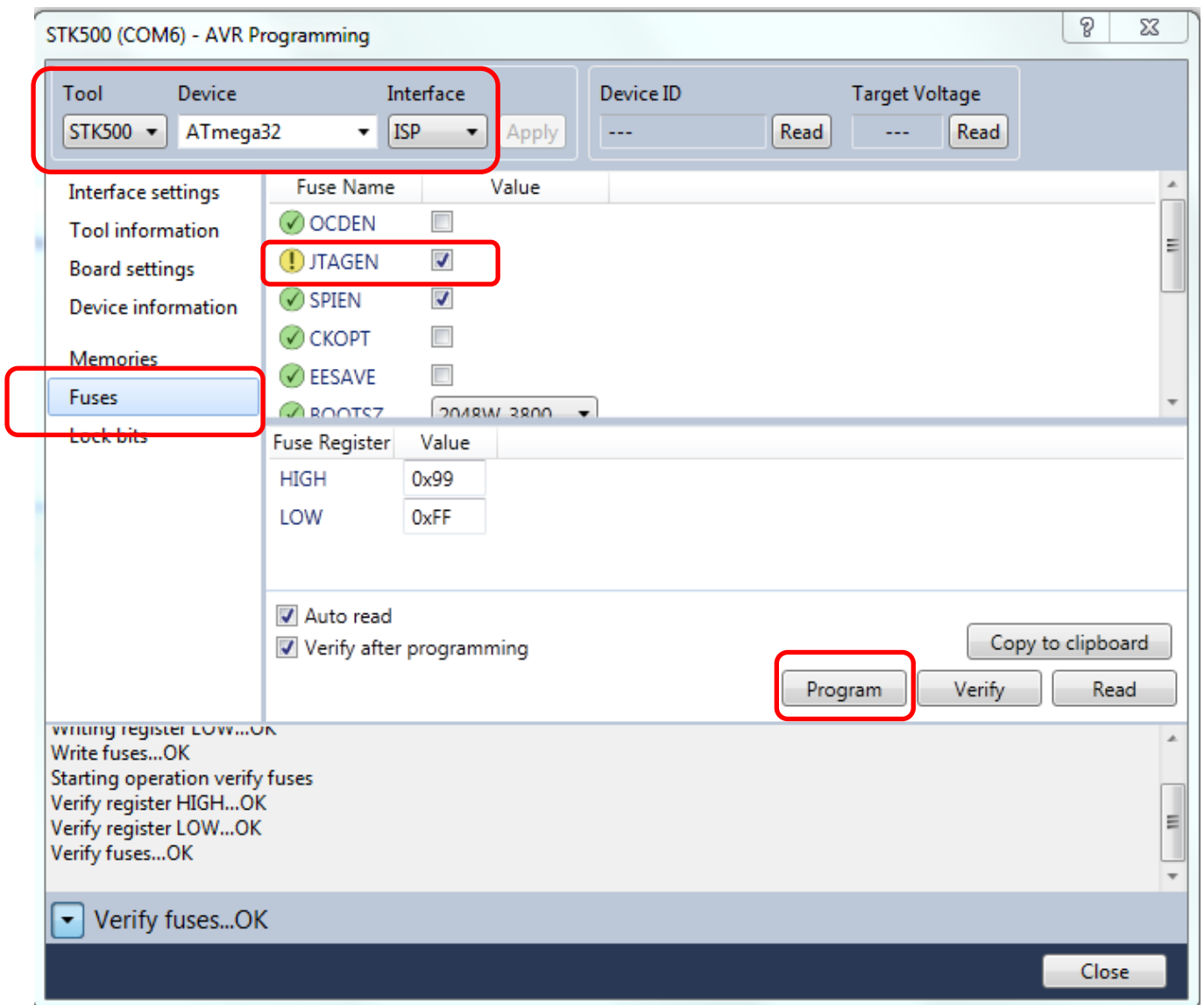
### Simulator and JTAG debugging

HH, January 9, 2012  
Page 14 of 17

IMPORTANT: Before you can use the JTAG interface of the microcontroller (the Mega32), a fuse internal the chip has to be set to enable the JTAG interface:

Connect to the STK500 using “normal” ISP connection and a COM-port.

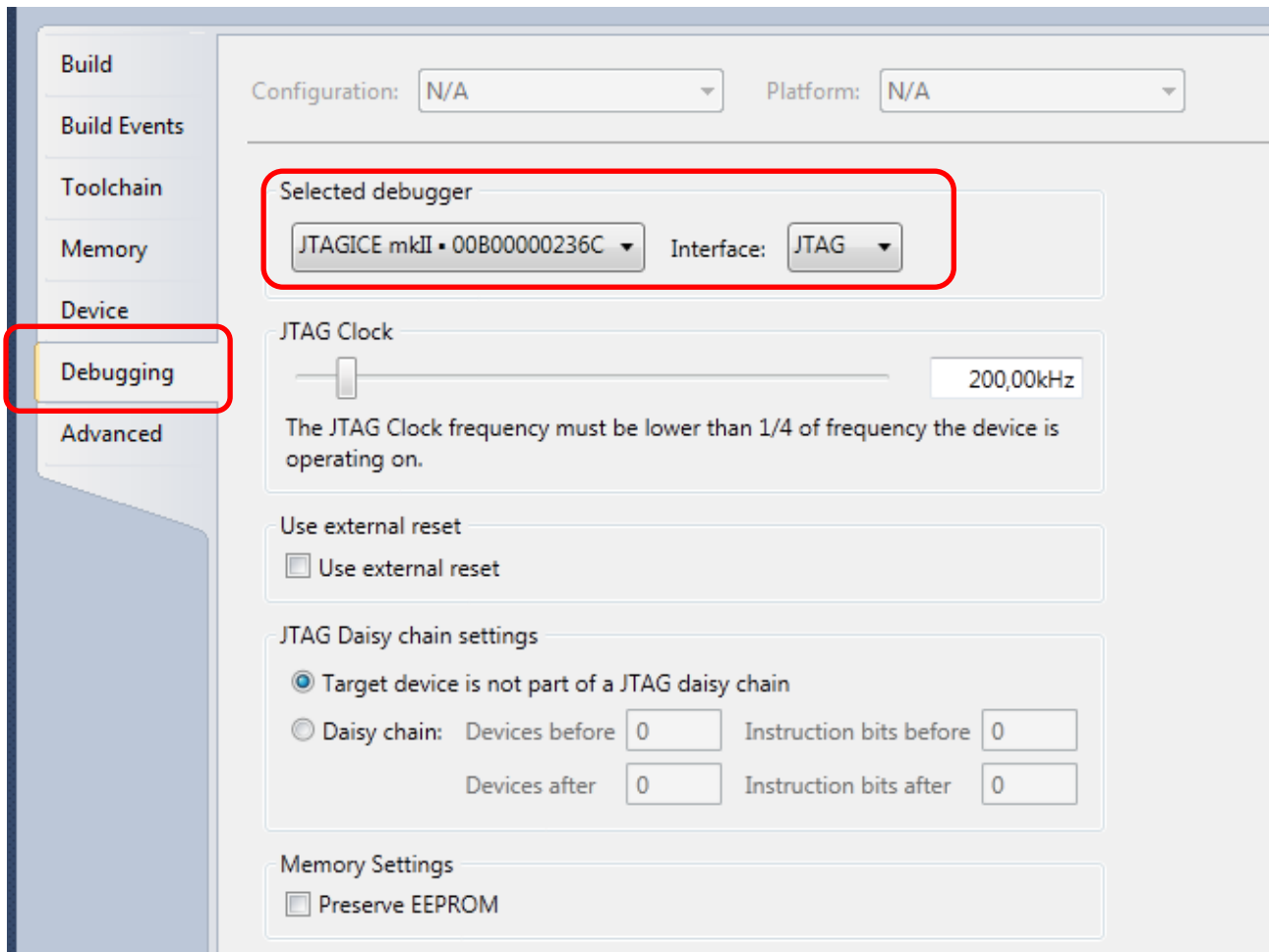
In the tab “Fuses” put a checkmark at “JTAGEN” and click “PROGRAM”:



*Notice:*

*After having enabled the JTAG interface of the microcontroller, 4 pins of PORTB will be reserved for JTAG communication (and therefore are not available for general purpose I/O any more).*

Now set up AVR Studio to use the JTAG ICE for debugging (instead of the simulator that we used in part2). Go to the project properties (Alt + F7) and select the JTAG ICE:



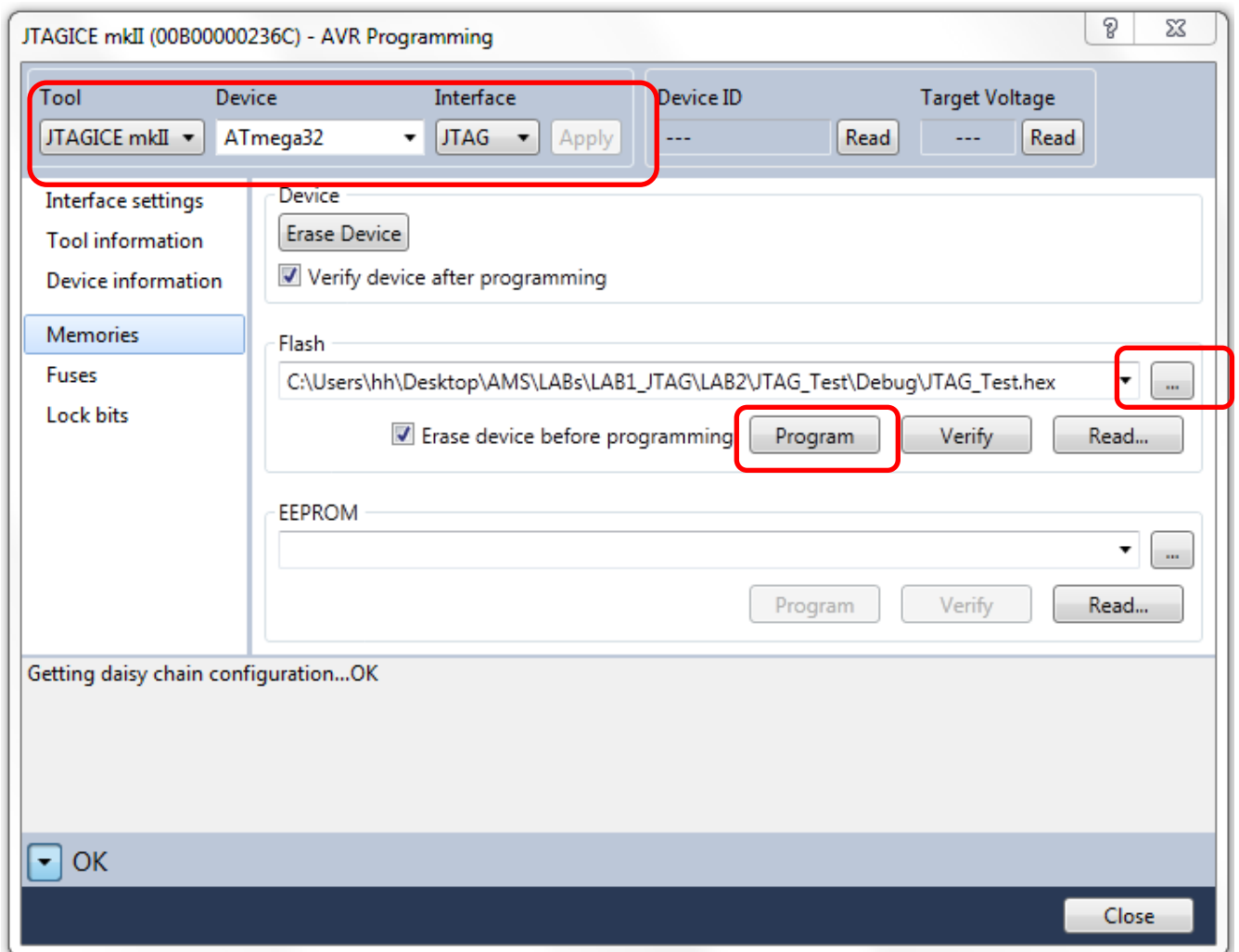
## AMS LAB exercise 2

### Simulator and JTAG debugging

HH, January 9, 2012  
Page 16 of 17

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

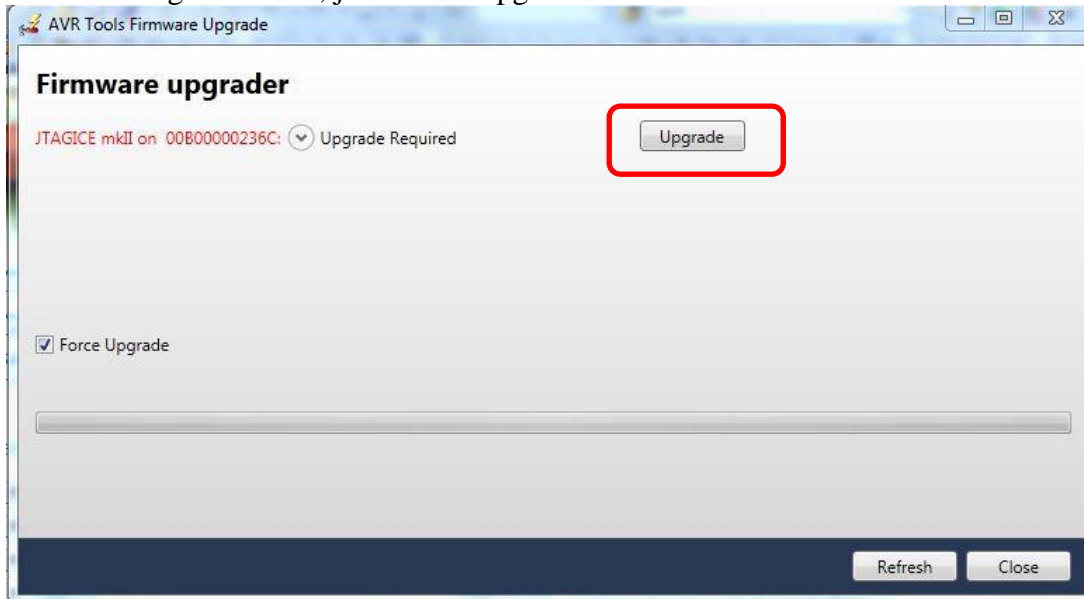
Simply select the JTAG ICE (and “JTAG” as “Interface”) instead of STK500:





When connecting to the JTAG ICE, AVR Studio checks for correct firmware version.

If this message is shown, just click “Upgrade”:



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 executing at the target hardware (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;
    }
    return 0;
}
```

Eventually study the JTAG ICE more detailed in the documentation and create your own experiments.