German-Russian Institute of Advanced Technologies

TU-Ilmenau (Germany) and KNTRU-KAI (Kazan, Russia)

Guidelines for laboratory work №3 of the subject

«Computer systems»

«AVR Simulation with the ATMEL AVR Studio 4»

Kazan 2014

AVR Simulation with the ATMEL AVR Studio 4

Getting Started

The AVR Studio uses a COF object file for simulation. This file is created with through the C compiler by selecting COF as the output file type. For more information on creating this file, see the C compiler documentation. Launch the AVR Studio by either selecting it through the Start Menu or by selecting the program icon (if available). Either method will produce the IDE shown below in figure 2. Once the IDE is running, select File Open through either the File Pull-down Menu or by clicking on the File Open Button.
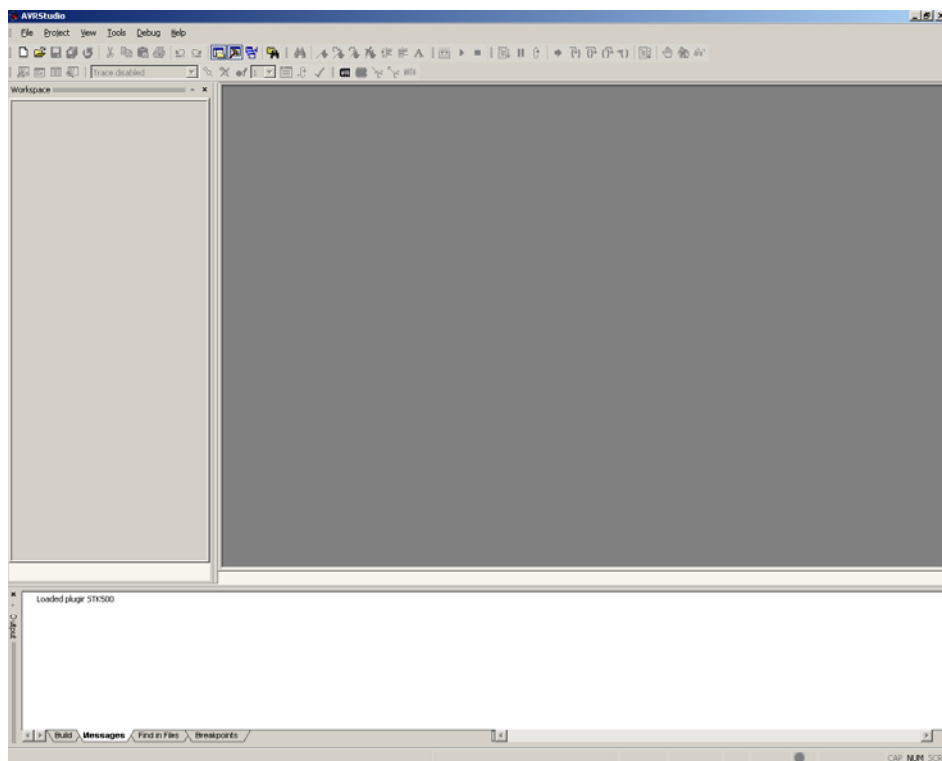
Select the desired COF file for simulation through the File Open window. Note this window uses standard Windows navigation.

Either double clicking on the file or by clicking on the file and then selecting the Open Button can open the file.

1. Select the proper *.cof file
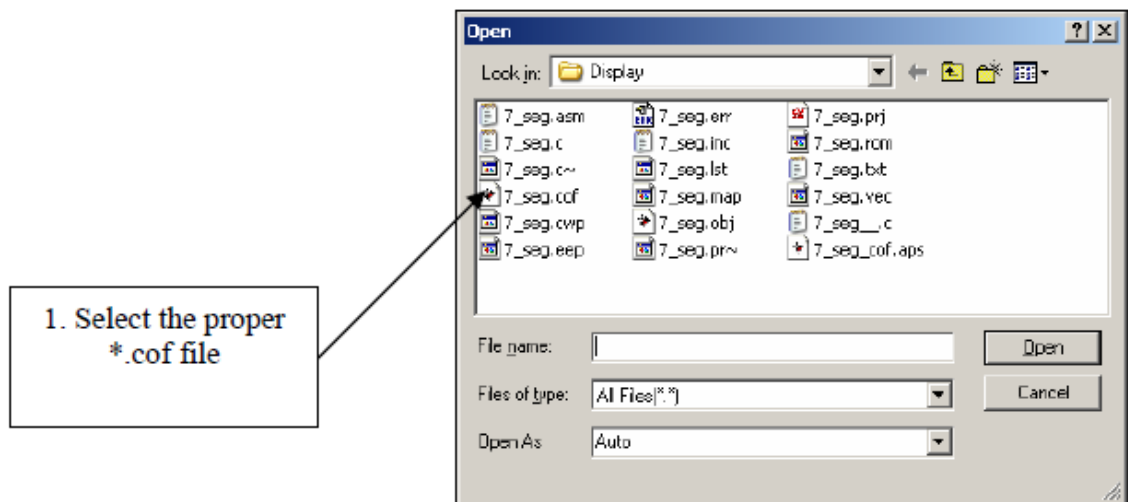
Device Selection

After the source file has been opened, the device and debugging platform must be specified. When doing simulation, select the AVR Simulator option and ensure that the proper AVR target device is selected. Once the correct target AVR microcontroller and platform have been selected, click on the Finish Button.
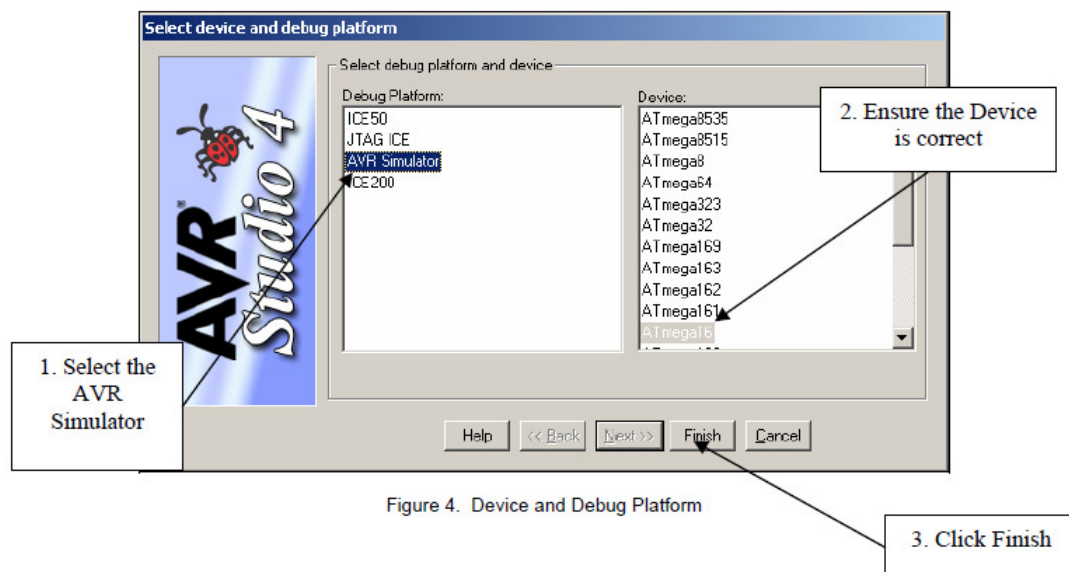

1. Select the AVR Simulator

2. Ensure the Device is correct

3. Click Finish

Figure 4. Device and Debug Platform

IDE Windows

The IDE has several windows that provide important information to the user. These windows may be opened automatically by the software or may need to be activated by the user.

Regardless of how the windows are activated, they can be moved and resized to fit the taste of the user. The main windows of interest are the Workspace, Source Code, Output, and Watch windows.
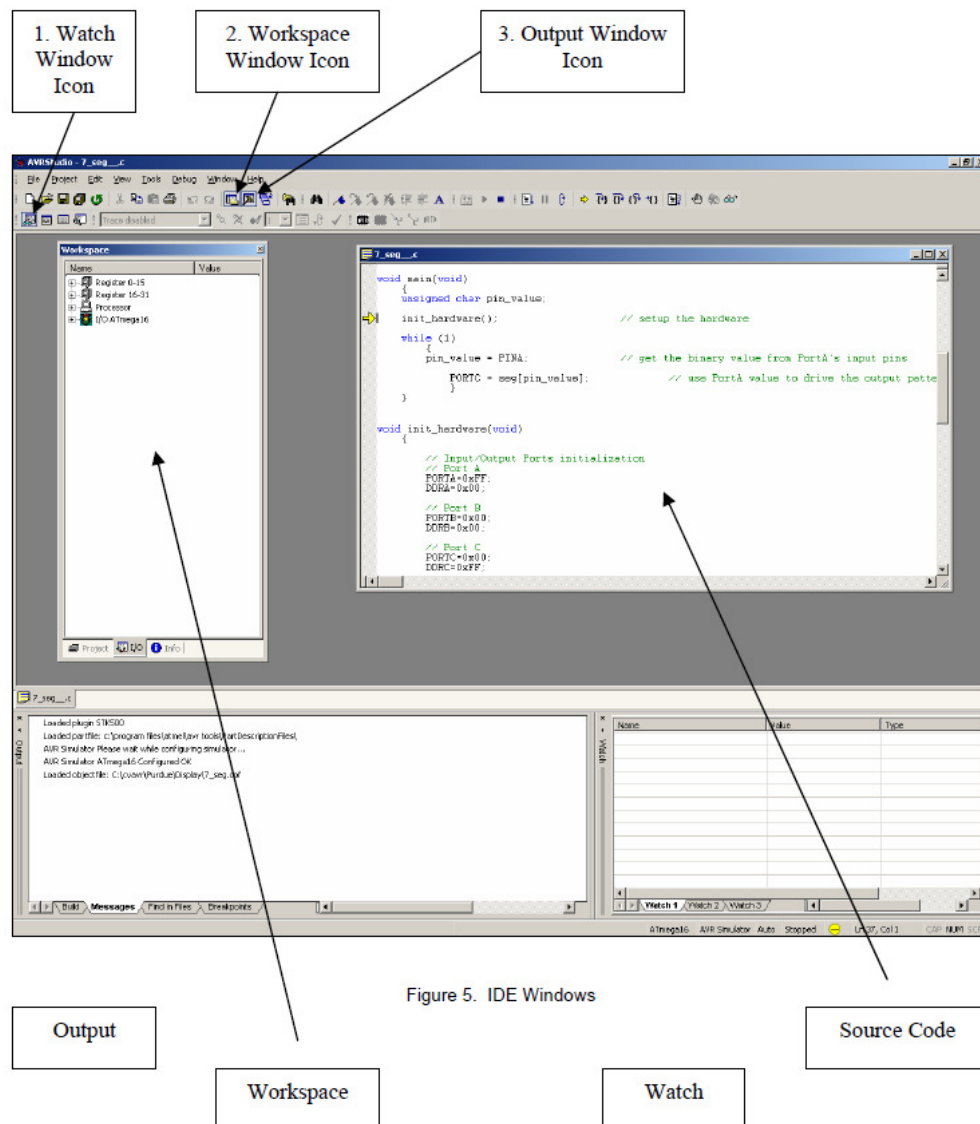
Figure 5. IDE Windows

Workspace Window

The Workspace window shown at the right holds important information about the microcontroller. Clicking on the expand symbols will provide detailed information about the selected item. For instance, expanding the I/O selection produces detailed information about the microcontroller ports, timers, USART, etc. These views are vital for simulating microcontroller software. They allow the user to monitor the values as well as introduce new information as inputs into the system. Additional information may be hidden from the user if the window is too narrow. Click and drag the right side of the window to re-size it and reveal possible hidden information. Double-clicking on the value of one of the PIN registers allows the user to enter or set the input value for that particular port. Likewise, the user may click on one of the boxes under the bit position to set an input.

1. Click to Expand

1. Expand Additional Layers

2. Expand to See Additional Registers
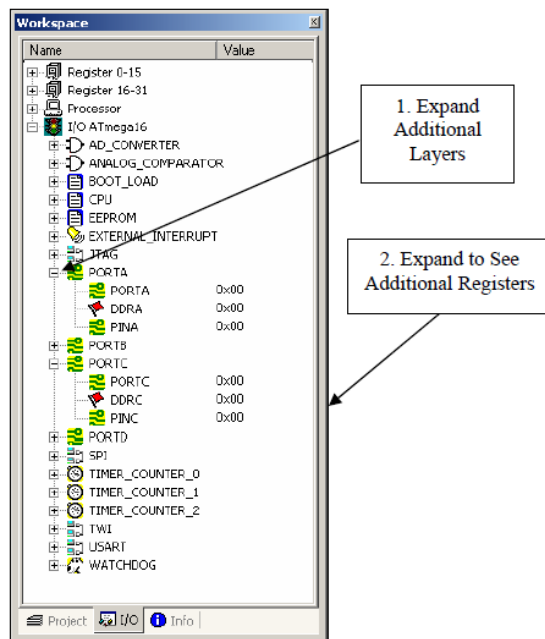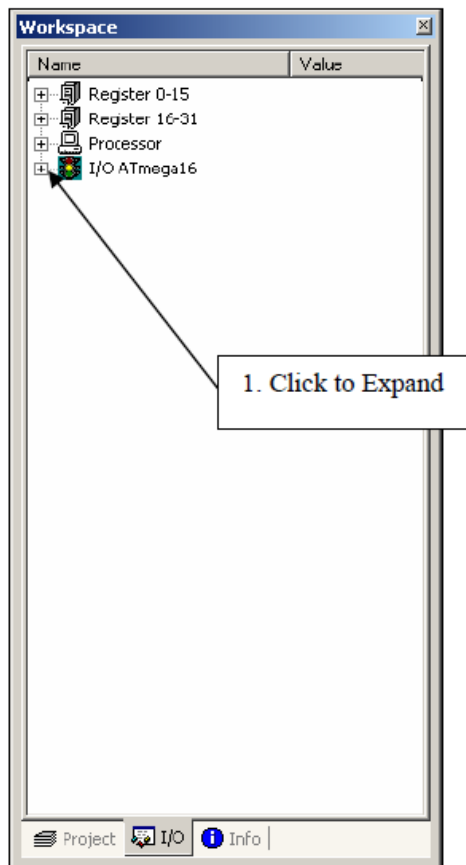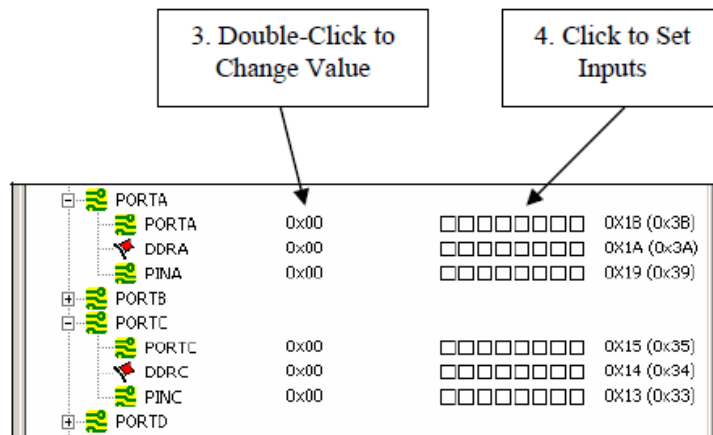
Watch Window

The Watch Window allows the user to monitor the variables used in the software. To Add a variable to be monitored, right-click in the window and select Add from the menu. The next line or box in the Watch window is bounded by a box and has a flashing cursor present inside of it. Type the name of the variable EXACTLY as it appears in the source code. It should be noted that local variables are only valid when inside the function where the reside.

2. Select Add
   From
   the Menu

3. Type
Variable Name
Here
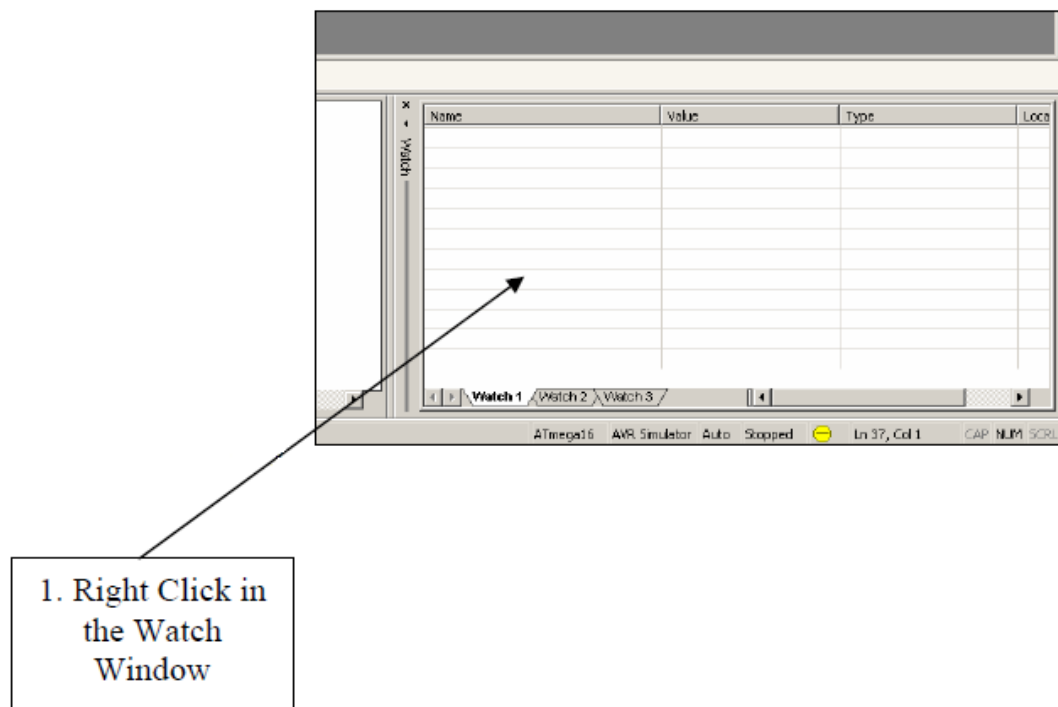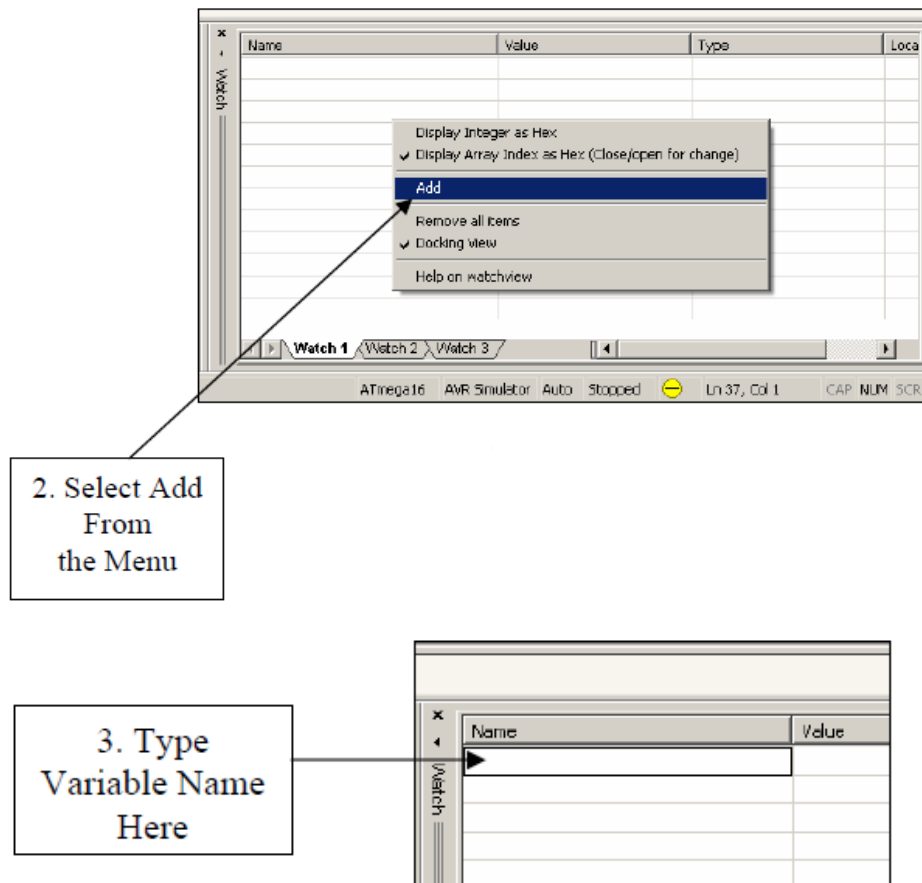
Output Window

The Output window provides feedback to the user. This includes messages about the microcontroller, object file, etc.
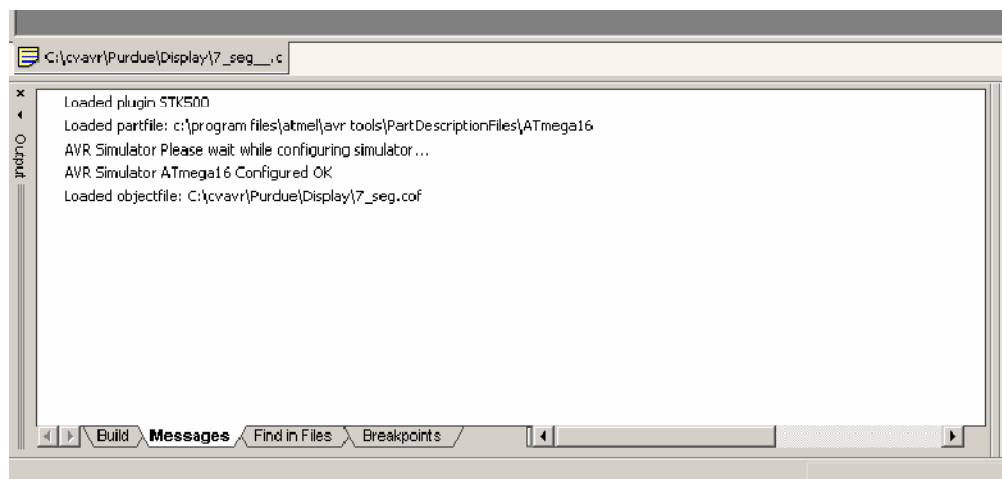
Simulator Options

Before actually starting a simulation, the Frequency of the target AVR should be set. The MegaAVR Development board from PRLLC operates at a clock

frequency of 6.0MHz. This frequency happens NOT to be one of the selections from the pull-down menu. Therefore, the user must manually type this value into the program.
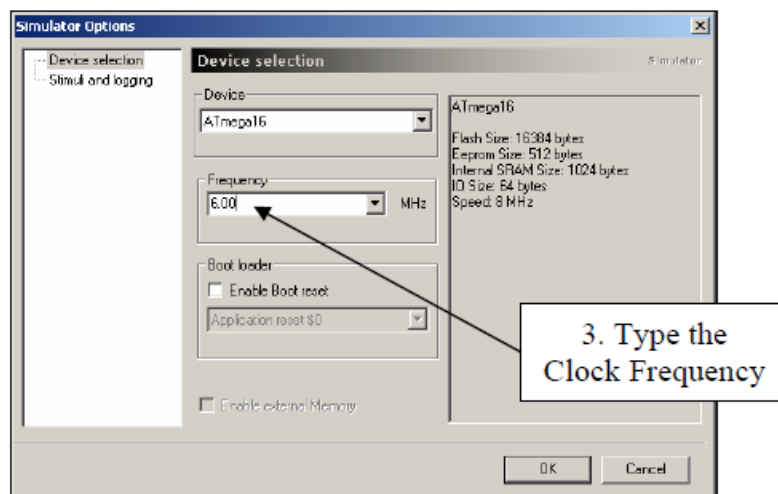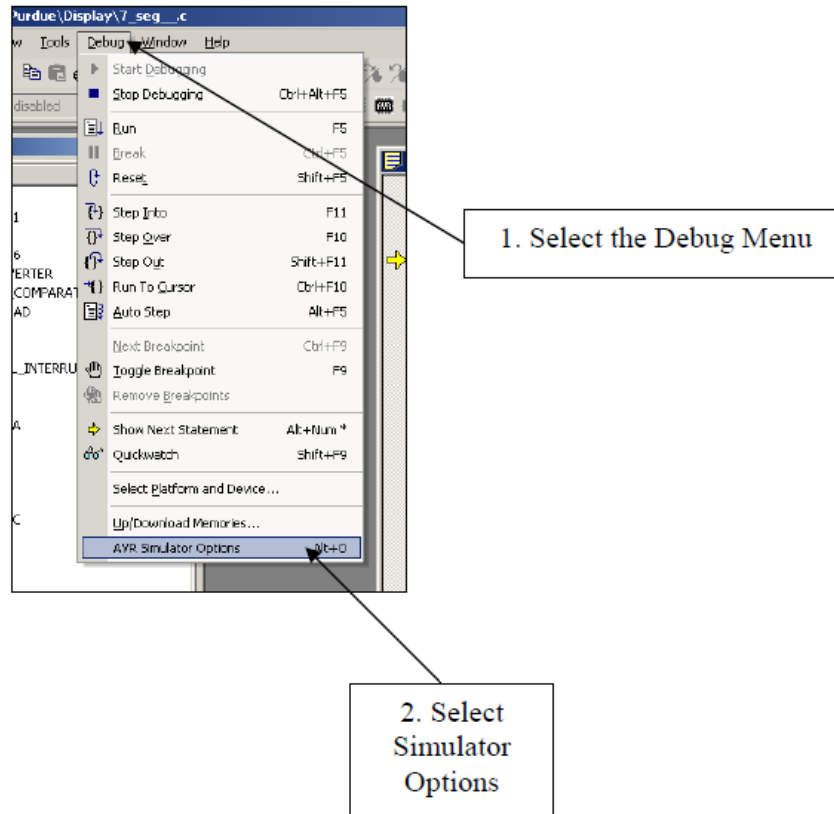

## Source Code Simulation Controls

IDE Toolbar

Once the IDE has been configured and the windows are positioned to the satisfaction of the user, the actual simulation of the source code can begin. The

yellow arrow indicates the next *statement* to be executed. The toolbar located in figure 14 shows a list of possible options to execute the source code. These options include Step Into (Single Step), Step Over, Step Out, Run to Cursor, Auto Step, and Run. In addition to the previous functions, the user can also Set reakpoints, Reset, and Stop Debugging.
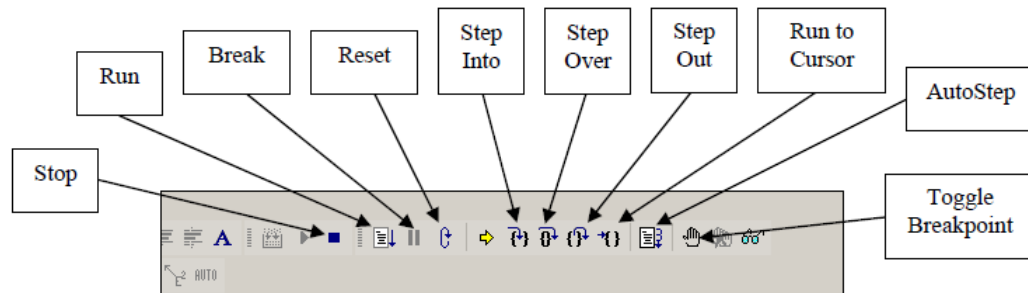
Next Statement

The yellow arrow shown below in Figure 15 *points* to the next instruction to be executed. This provides the user a visual indication of the future instruction or function to be performed.
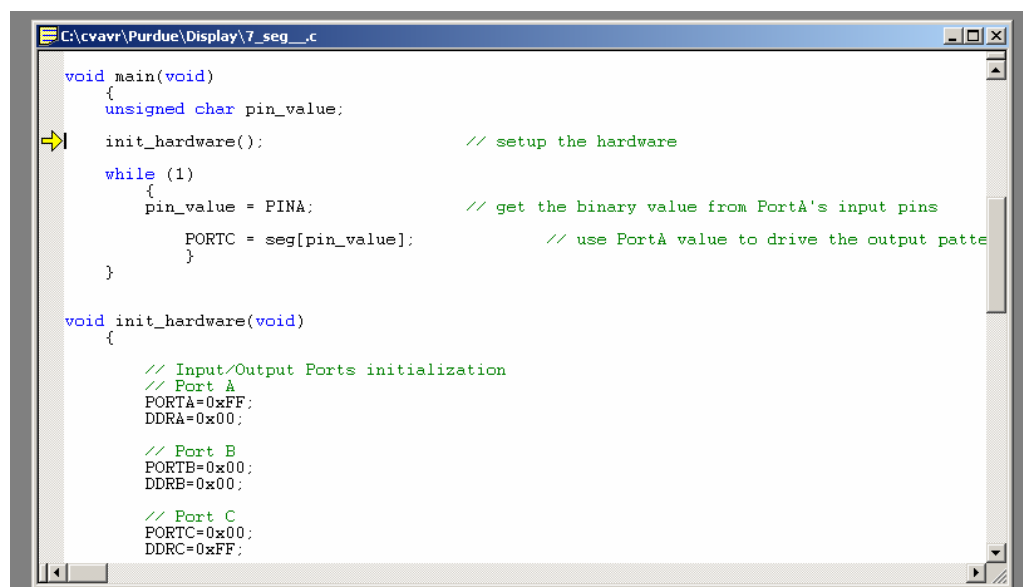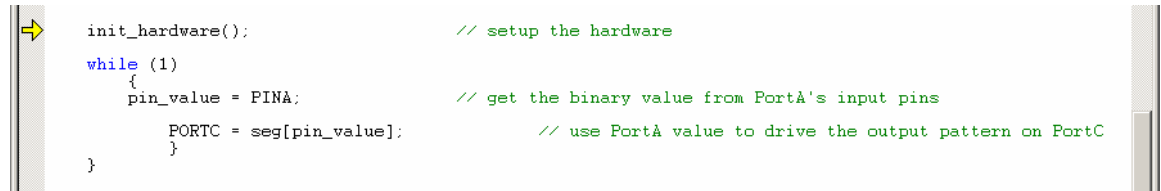

Step Into

Perhaps the most commonly used operation when simulating software is the Step Into operation. The operation can also be referred to as Single Stepping. This command allows a single instruction to be executed at a time. This includes

instructions that are located inside of a function call and thus the name Step Into. Executing a Step Into command is accomplished by either clicking on the icon or by pressing the F11 key. Performing this command on the sample program shown in Figure 17 causes the next instruction to be performed.
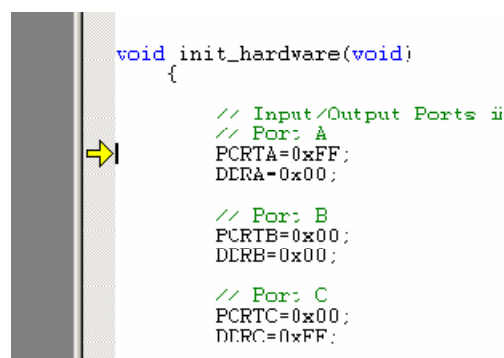
```
init_hardware();                    // setup the hardware

while (1)
    {
    pin_value = PINA;               // get the binary value from PortA's input pins

        PORTC = seg[pin_value];         // use PortA value to drive the output pattern on PortC
        }
}
```

Step Into Function

In this example, the next instruction is a function call to a function named *init_hardware( )* that will initialize the hardware for the microcontroller. Figure 18 shows the results of executing the Step Into command. The yellow cursor is now pointing to the first statement of the function.

Performing an additional Step Into executes the current statement and moves to the next statement in the function. This process can be repeated throughout the software.
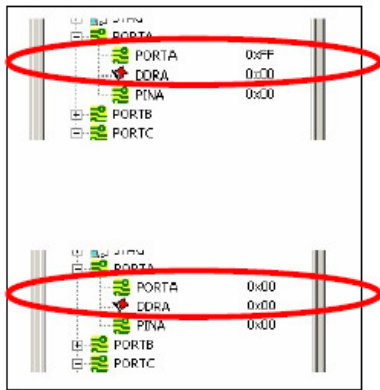
```
void init_hardware(void)
    {

        // Input/Output Ports in
        // Port A
        PCRTA=0xFF;
        DIRA=0x00;

        // Port B
        PCRTB=0x00;
        DIRB=0x00;

        // Port C
        PCRTC=0x00;
        DIRC=0xFF;
```
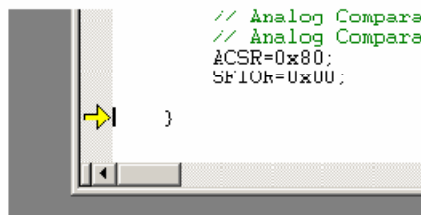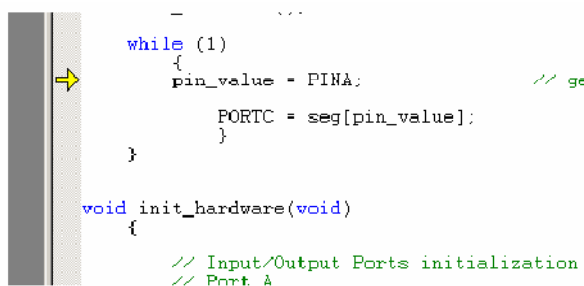
First Line in Function

PORTA Values

Last Line of Function
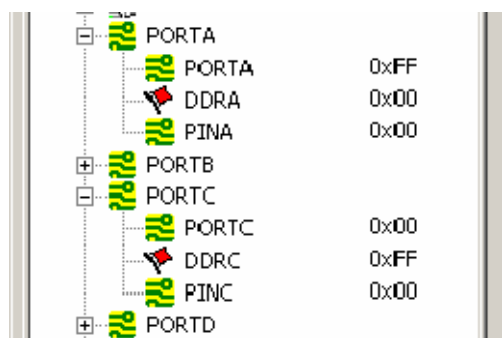

Return to Main Function

Step Over

 The Step Over command causes the simulator to give the illusion that it skipped over the function without executing the individual statements located inside of it. For example, revisiting the same sequence from the Step Into discussion, this time, instead of using the Step Into command, a Step Over command will be used.
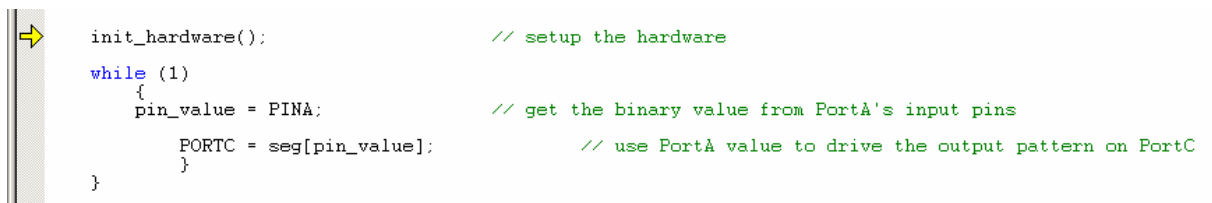
Main Prior to Function Call
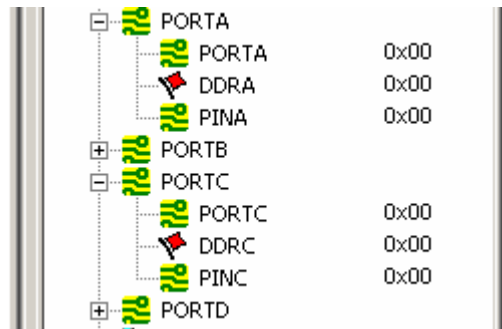
I/O Status Prior

I/O Status After

Main After Function Call
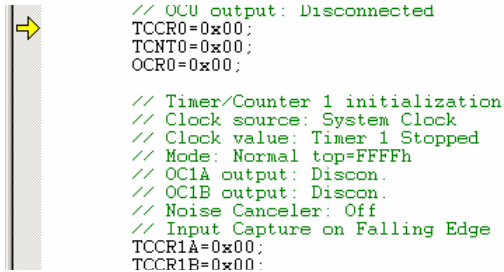
This technique is extremely useful when the user isn't interested or concerned with *watching* each and every instruction in a function being executed. This command can also be used as a time saving technique while performing a simulation. The Step Over command can be executed by clicking on the Step Over icon or by pressing the F10 key.

Step Out

The Step Out command is executed inside a function when the user wants to return back to the calling function without having to execute each individual

step of the function. For instance, the user may be interested in single stepping through the first several instructions inside a function. Once through the area of interest, the user wants to return to the function from which the current function was called in a single command.
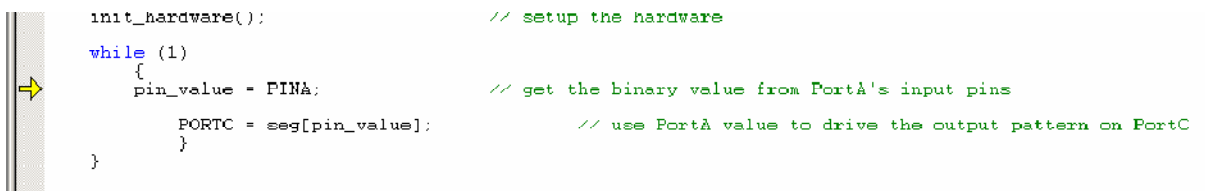

Inside the Function

instructions of interest, executing a Step Out command finishes the remainder of the instructions inside the function and places the cursor at the next instruction in the calling function. This command can also be used to save time while simulating software. This command is executed by clicking on the Step Out icon or by pressing the F11 key while holding down the SHIFT key.

Returning to the Calling Function

Auto Step

The Auto Step function can be viewed as the PC executing a series of Step Into commands automatically. This command can be executed by either clicking on the Auto Step icon or by pressing the F5 key while holding down the ALT key. Once this command is started, each statement will be executed in order. The cursor and yellow arrow will still indicate the next statement to be executed. However, since the PC is executing single steps continuously, the yellow arrow and cursor are also moving constantly. The views inside of the Workspace window

are also updated after each instruction is executed allowing the user to actually see the changing values in registers. The ability to view the changing value is a very important feature associated with this command. It must be noted that the simulator is capable of changing the values located inside the Workspace but the user is not. The simulation must be stopped prior to the user manipulating these values.

Break

The Break function allows the user to stop a simulation that is under the control of the PC. This command allows a user a stop or pause a simulation that is using the Auto Step function, for example, to enter new data into the Workspace or to modify a variable located in the Watch window. Once stopped, a simulation can be re-started by choosing any of the methods previously described.
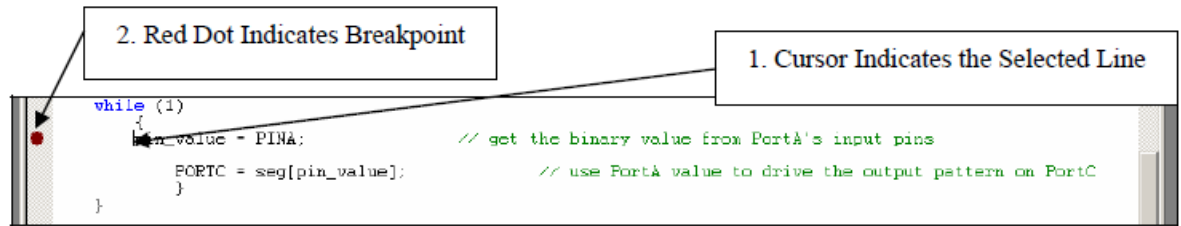
Reset

The Reset command will perform the same function as a *Real* reset on the actual hardware would produce. The cursor and thus the next statement to be executed will be set as the very first instruction as if the file was just opened. In addition, the values of the registers will also be set to the reset state. The values of variables used in the software will not be affected by this command. The variables will retain their values until a process overwrites them. A Reset can be executed by either clicking on the Reset icon or by pressing the F5 key while holding down the SHIFT key.

Breakpoints

Breakpoints allow the user to start a simulation *free running* and have the simulation stop when a certain instruction or place in the program is reached. Breakpoints can be *set* by placing the cursor at a specific line in the software and then selecting the breakpoint icon or by pressing the F9 key. The breakpoint can be cleared by pressing the breakpoint icon again or by pressing the F9 key (a second time) while the cursor is located on the line that the breakpoint is located. A Red Dot indicates that the line has a breakpoint.

2. Red Dot Indicates Breakpoint

1. Cursor Indicates the Selected Line

```
while (1)
{
    pin_value = PINA;        // get the binary value from PortA's input pins
    PORTC = seg[pin_value];  // use PortA value to drive the output pattern on PortC
}
}
```

Breakpoints

Run

The Run option allows the software to be simulated as quickly as possible. The downside of this option is that *none* of the registers or variables will be updated (visually to the user) while the simulation is in process. The user must manually stop the simulation by using the break command. Alternatively, the user can set a breakpoint on a particular line of code that will stop the simulation when encountered.

TASKS

1. Write a program, calculating Fibonacci numbers. In a Fibonacci series each number is the sum of the two previous numbers.

2. Use CodeVision AVR to write a program.

3. Create a project, build it and simulate with the AVR Studio simulator.

4. Demonstrate the work of your program to the instructor.

5. Open the Watch window, showing the variables.

6. Explain the results.

CONTROL QUESTIONS

1. Why do we need a Workspace window?

2. Why do we need a Watch window?

3. Why do we need an Output window?

4. Why do we need an IDE Toolbar?

REPORT FORM

German-Russian Institute of Advanced Technologies

TU-Ilmenau (Germany) and KNTRU-KAI (Kazan, Russia)

Laboratory work 3

« **AVR Simulation with the ATMEL AVR Studio 4**»

Student: _____

Teacher: _____

Kazan 2014