AtmanAVR Quick Start Tips

Overview

AtmanAVR is a high-performance C/C++ integrated development environment, which adopts visual and modular programming. Provide rich humanized tools to make development more convenient.

Visual

Whether creating a project or developing a project, the wizards can help you configure the options of MCU and its peripherals at any time, and automatically generate the code for you.

Modular

AtmanAVR wizard manages every peripheral of AVR (such as timer, ADC, UART, etc.) as a module. When you use the wizard to enable and set up a peripheral, the wizard will automatically generate a module code file, a source file and a header file in the project, and its code includes the initialization function and interrupts function of the peripheral. At the same time, the header file of the module will be automatically included in the main file, and the initialization function of the module will be automatically called in the main function.

Interface

AtmanAVR has many toolbars and windows, and not all of them will be displayed during the initial installation. Users can show or hide some toolbars and windows as needed.
There are two kinds of AtmanAVR interfaces, namely, the **Build** state and the **Debug** state. Each state will remember the toolbars and windows set by users respectively, and when switching states, the toolbars and windows set by users will be displayed.

1. **Build** state – That is, the usual source code editing state.

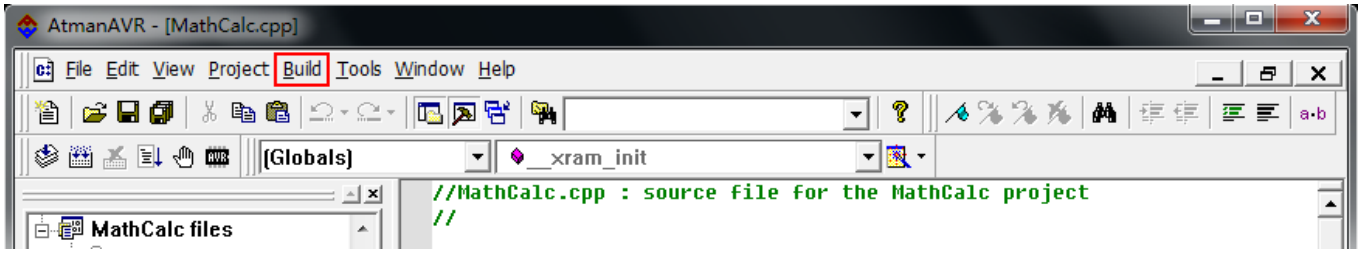

Fig1. **Build** state

2. **Debug** state – That is, the code debugging state.

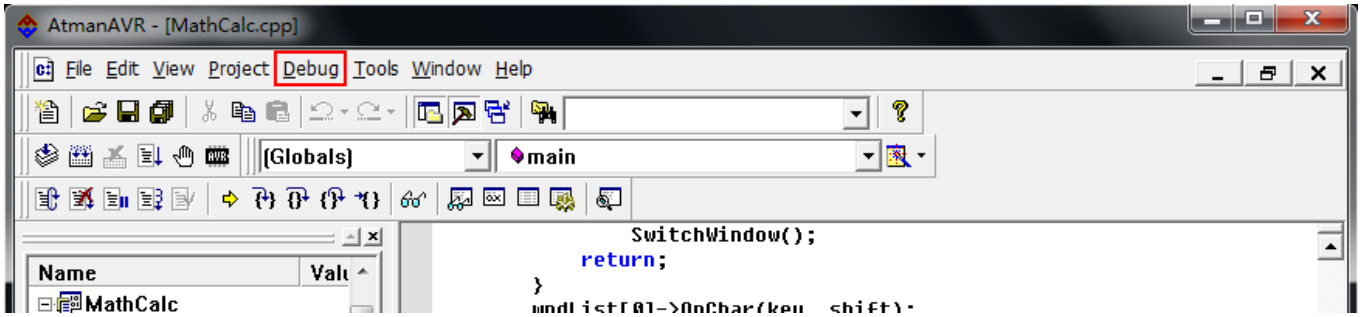

Fig2. **Debug** state

When debugging for the first time, the **Debug** toolbar may not be displayed. According to the following method of showing and hiding the toolbar, the **Debug** toolbar can be displayed. When debugging again, the **Debug** toolbar will be automatically displayed according to the last memory.

Show / Hide Toolbars

1. Right-click in the toolbar area
2. In the pop-up shortcut menu, check or uncheck the toolbar.

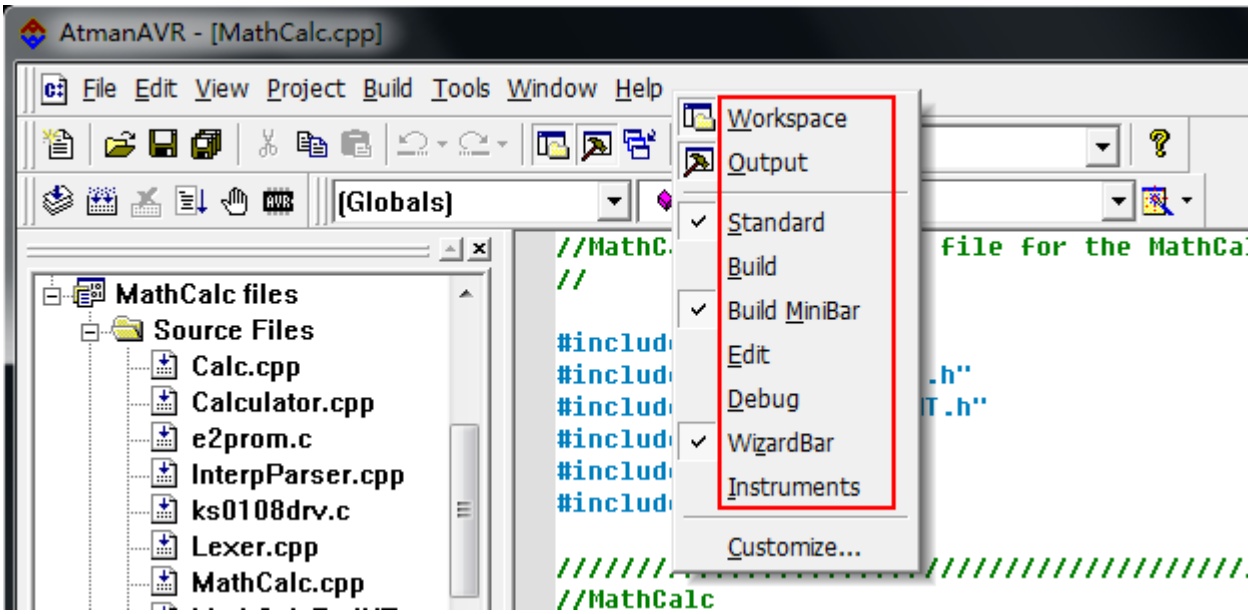

Fig3. Show/Hide Toolbars

Show / Hide Debug Windows

All Debug windows are available only in **Debug** state.

1. From the menu **View** -> **Debug Windows**
2. Click the debugging window to show or hide.

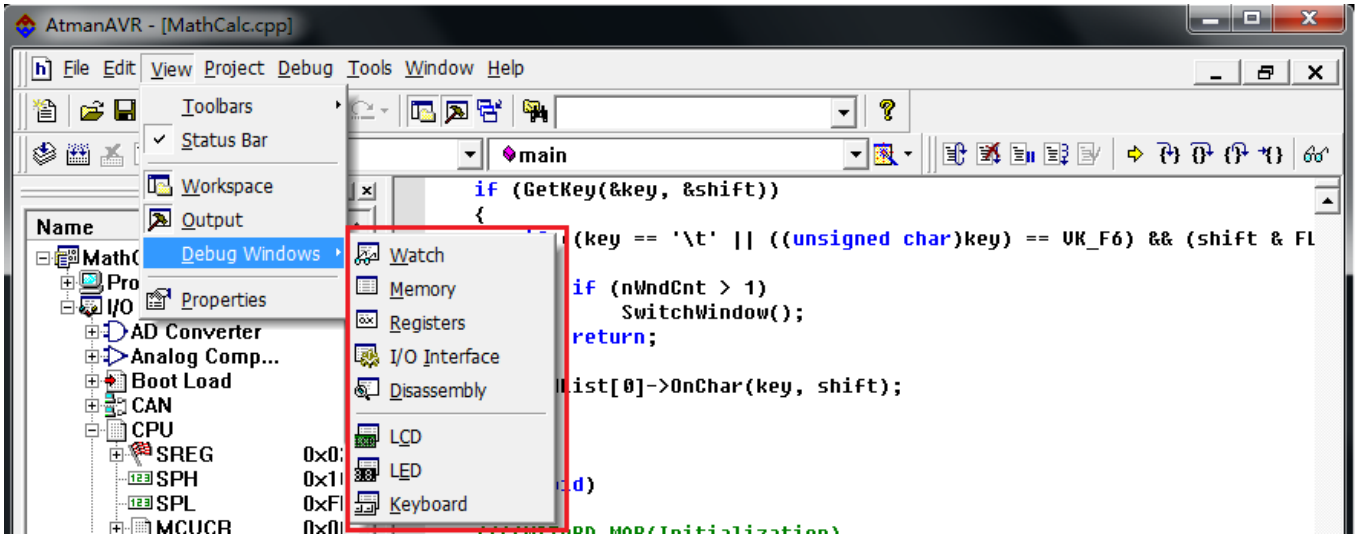

Fig4. Show/Hide **Debug Windows**

Projects Management

In AtmanAVR, the Project Workspace is the container of your development project. When you create a new project, a workspace is also created. You can use the Project Workspace window to view and access various elements of the project.
After you create a project workspace, you can add/delete projects in it.
The operation of managing a project can be Undo and Redo just like text editing.

Project Workspace

The Project Workspace contains three views: File view, Class view and IO view.

File View

The management of project files can be operated in **FileView** or in the menu **Project**.

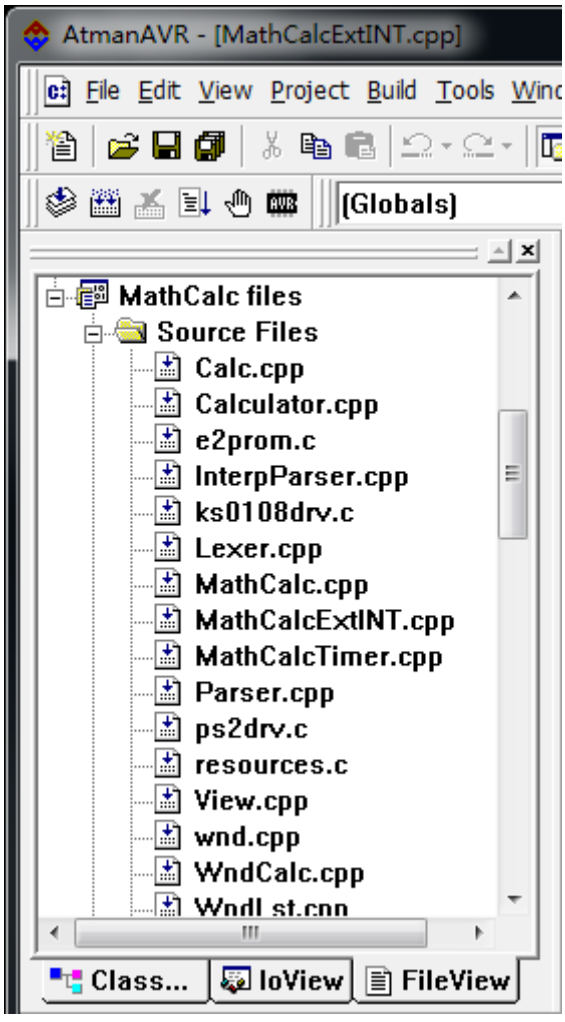

Fig5. FileView

Add a new project to the Workspace

Right-click the workspace in the **FileView**, calls the context menu, and selects the command **Add New Project to Workspace.**

Add an existing project to the workspace

Right-click the workspace in the **FileView**, calls the context menu, and selects the command **Insert Project into Workspace**.

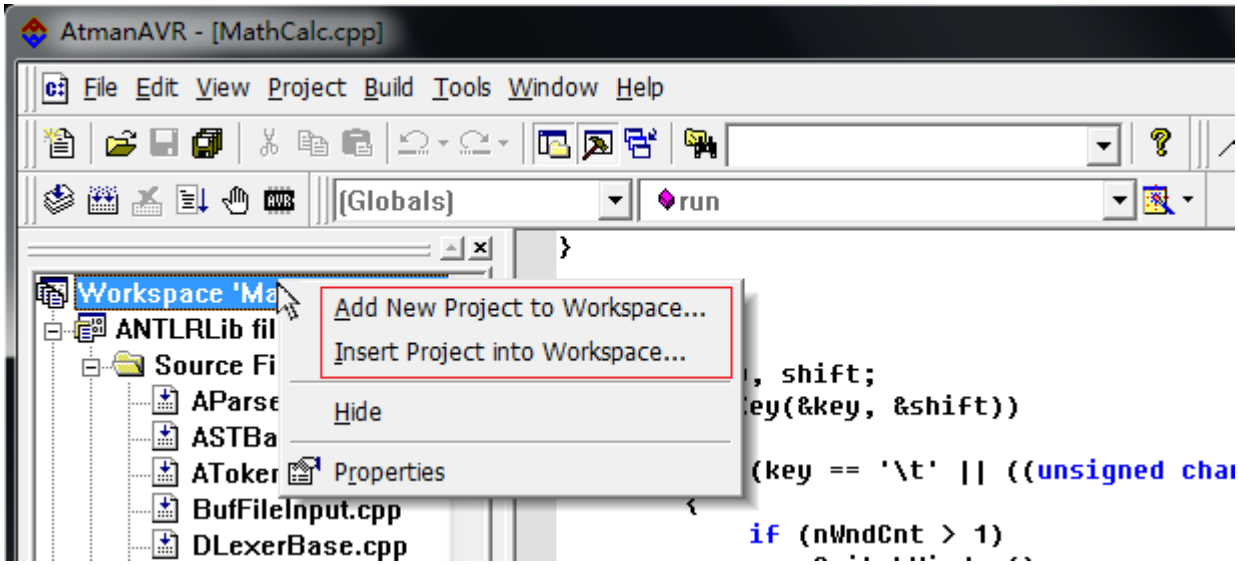

Fig6. Add project to workspace

Delete a project from the workspace

Right-click the item to be removed in **FileView**, calls the context menu, and selects **Delete**.

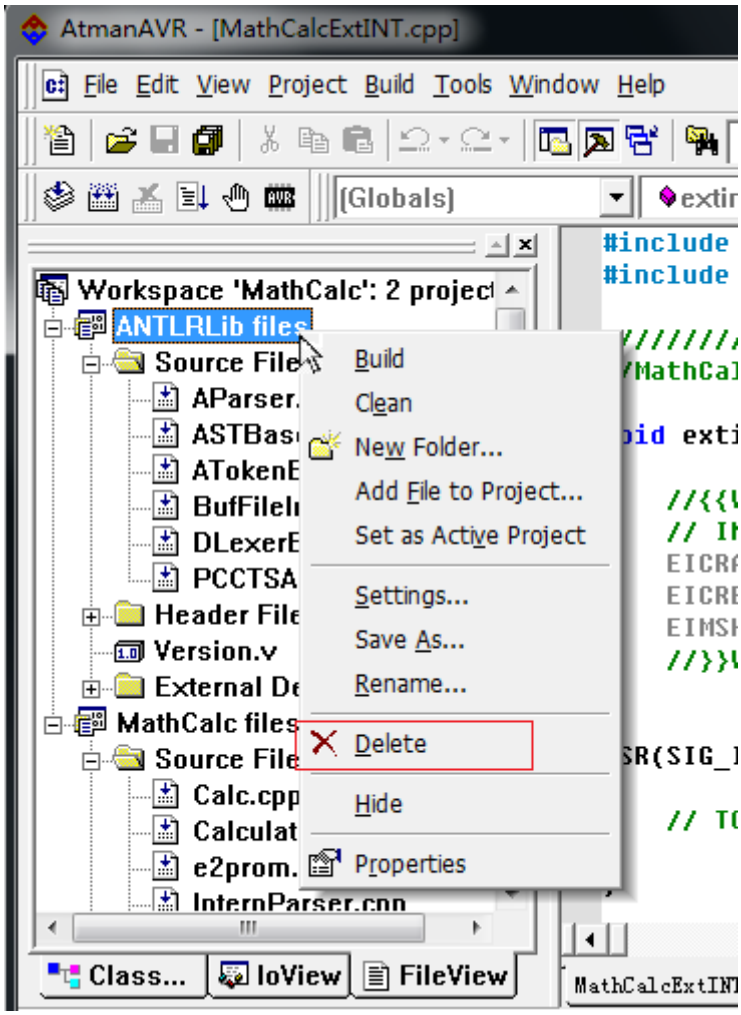

Fig7. Delete a project from the workspace

Add existing files to project

Right-click the project to which you want to add files in **FileView**, call the context menu, and select the command **Add File to Project**.

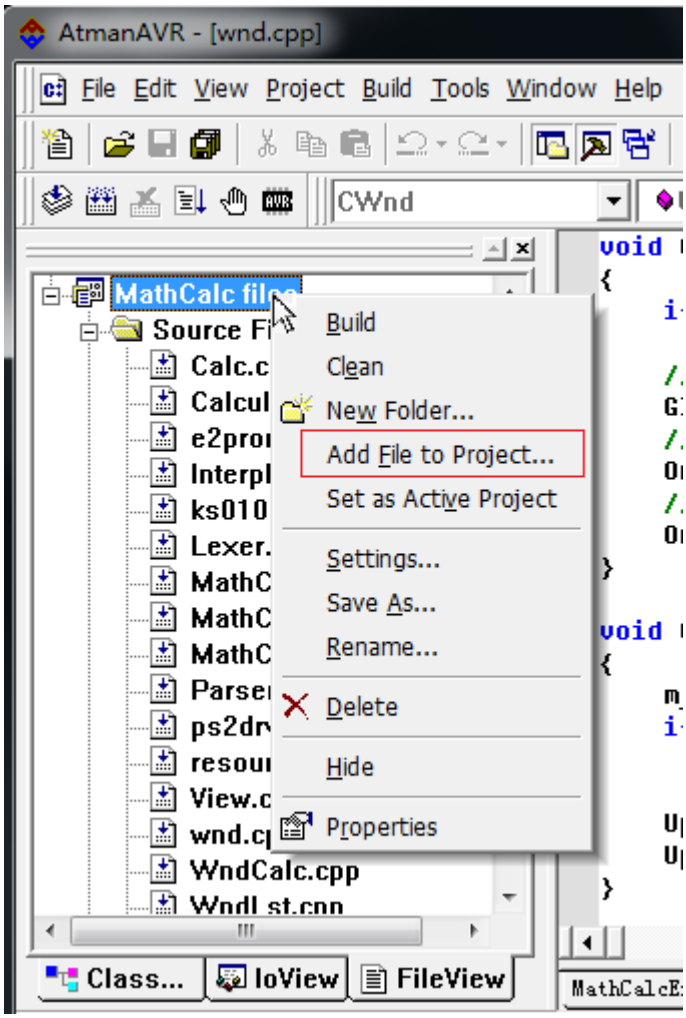

Fig8. Add File to Project

Remove files from a project

Right-click the file to be removed in **FileView**, calls the context menu, and selects **Delete**.

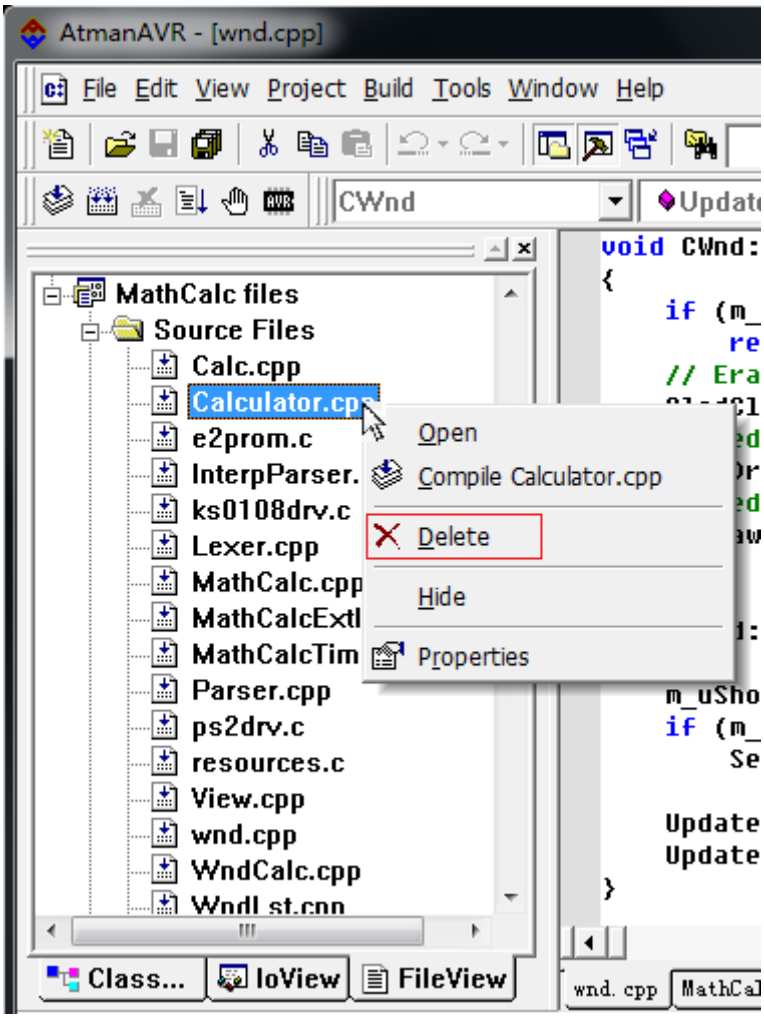

Fig9. Remove file from project

Class View

AtmanAVR uses a dynamic parser to scan the source code files in the project workspace and get the display content of the ClassView panel. When you type a new class, variable, or member function, ClassView and WizardBar will immediately update the display content without saving it in advance.

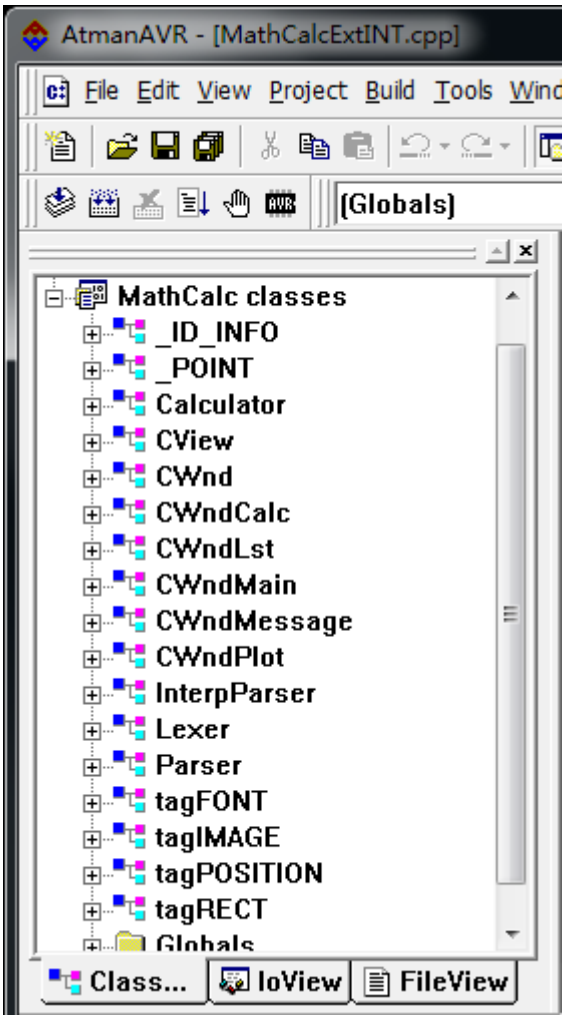

Fig10. Class View

In ClassView, you can:

Locate the definition or declaration of a class or member

Right-click the symbol (class, function, or variable) to find the definition or declaration in ClassView, call the context menu, and select the relevant command.

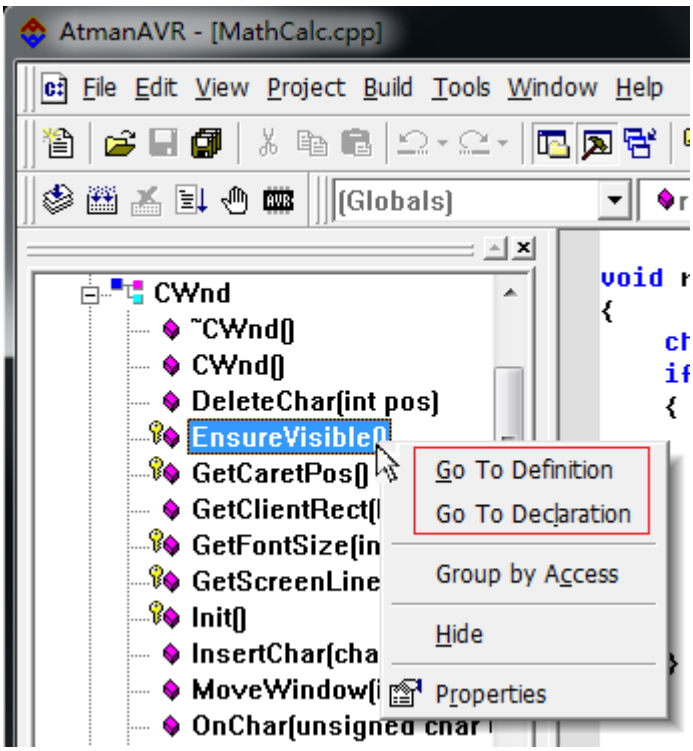

Fig11.. Locating symbol commands

– OR –

Double-click the symbol name directly.

Group members in a class

You can group members of a class alphabetically or by access rights-that is, private, protected, or public.

Select a class node, right-click to call the context menu, and click **Group By Access** to reverse the grouping order.

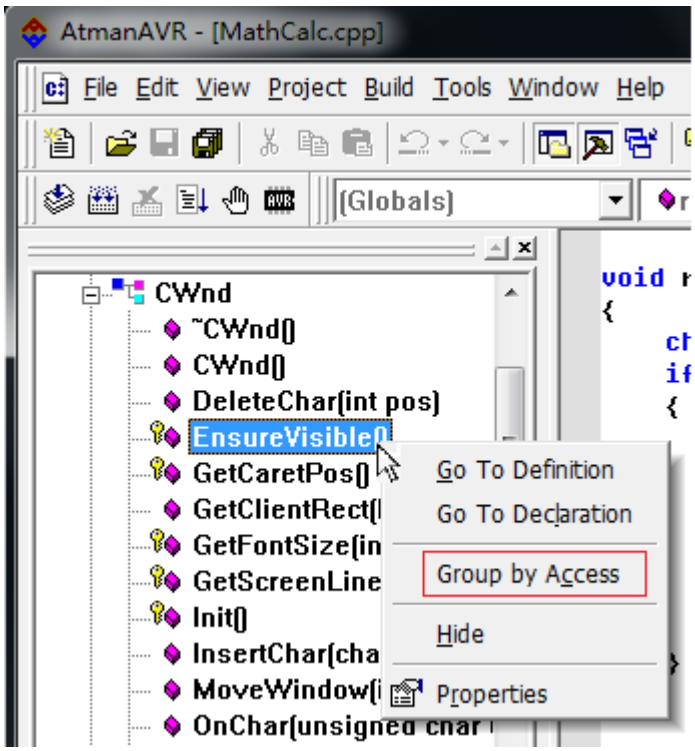

Fig12. Group members of a class command

IOView

IO View allows you to view or modify the values of IO registers and processor information.

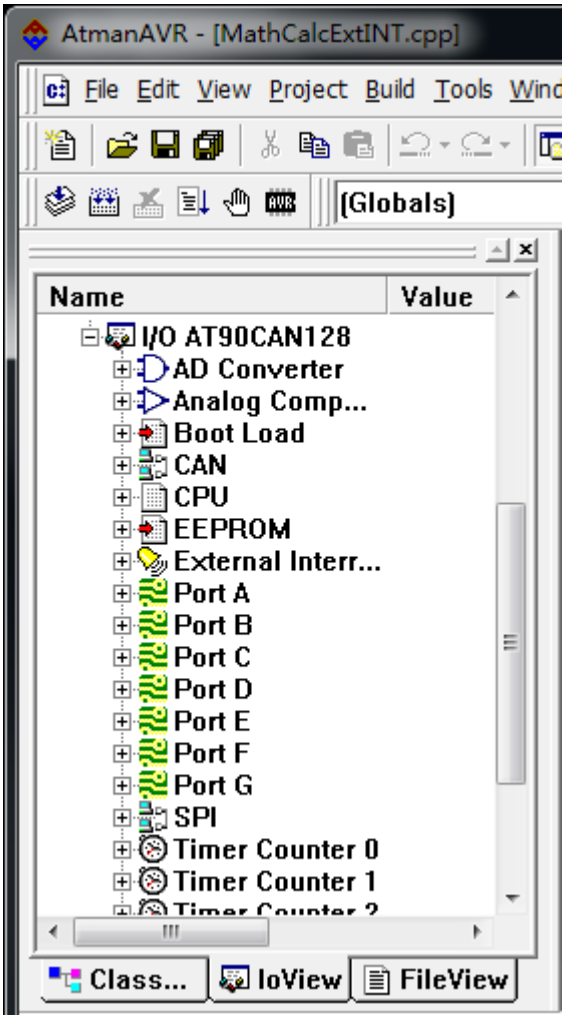

Fig13.. IOView

View information about the processor

Select the **IoView** tab and expand the **Processor** entry.

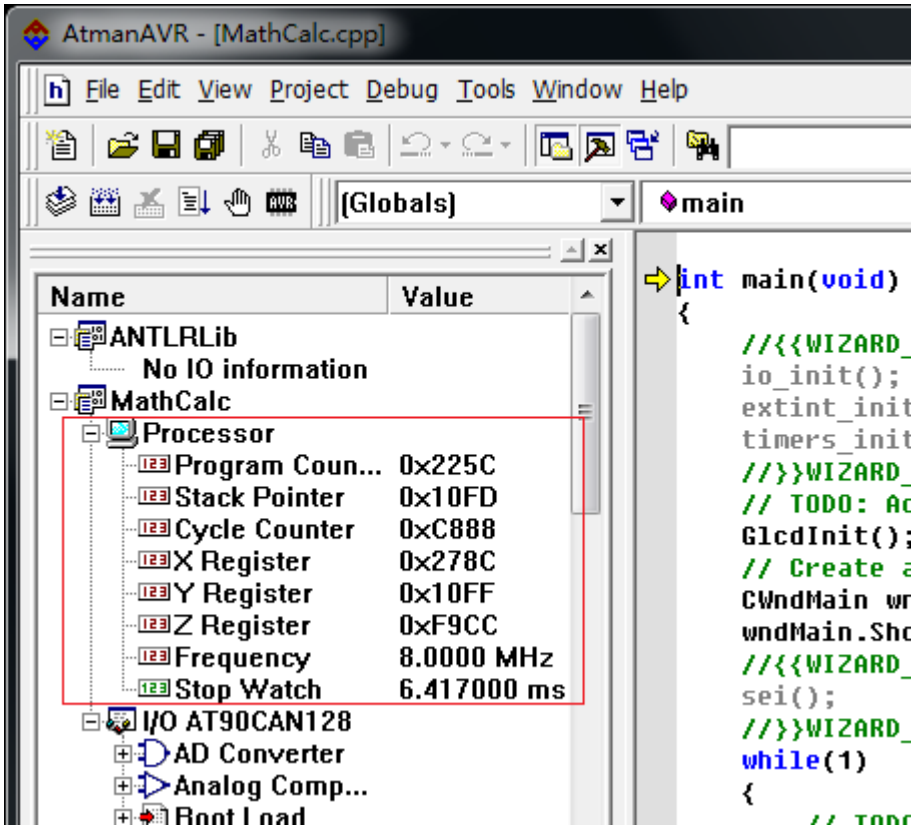

Fig14.. Processor information

Determine the running time of the code

StopWatch of the Processor entry in **IoView** records the running time of the code. Double-click it to clear it and start recording again, which can be used to test the running time of a certain code.

1. Place breakpoints at the beginning and end of the code segment to be tested.
2. Click **Go** or press **F5** to start the debugger.
3. After running to the breakpoint at the beginning of the code, double-click **StopWatch** to clear it.

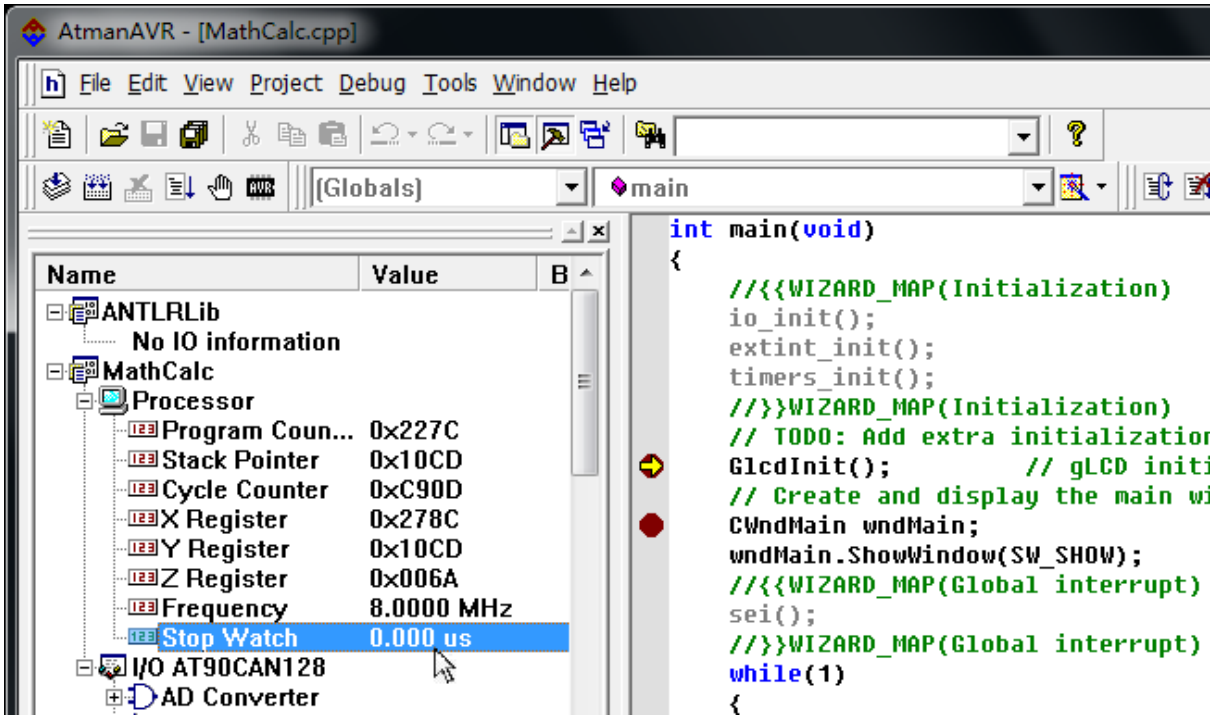

Fig15. Clear StopWatch

4. Click **Go** or press **F5** to continue running the debugger until it reaches the breakpoint at the end of the code.
5. At this point, the value of **StopWatch** is the running time of the code.

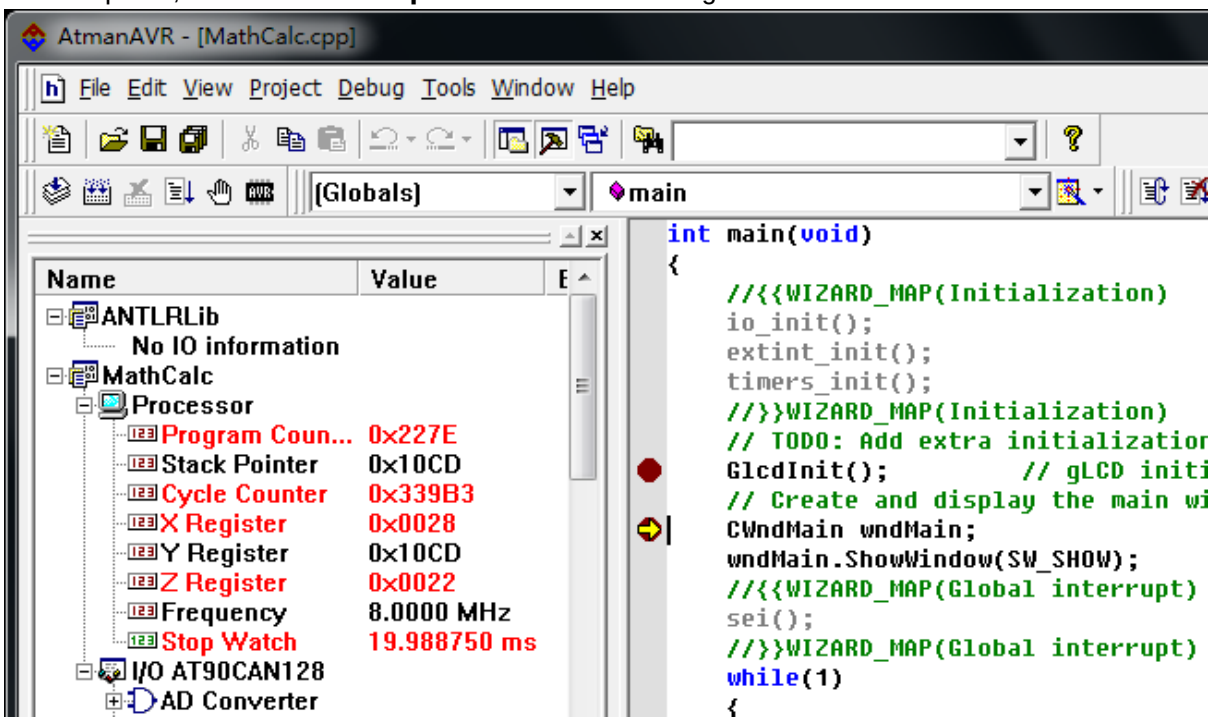

Fig16. StopWatch displays the running time

Modify the value of IO register

1. Expand the peripheral entry containing the I/O register to be modified in **IoView**.
2. Double-click the value of the I/O register, type a new value and press **Enter**

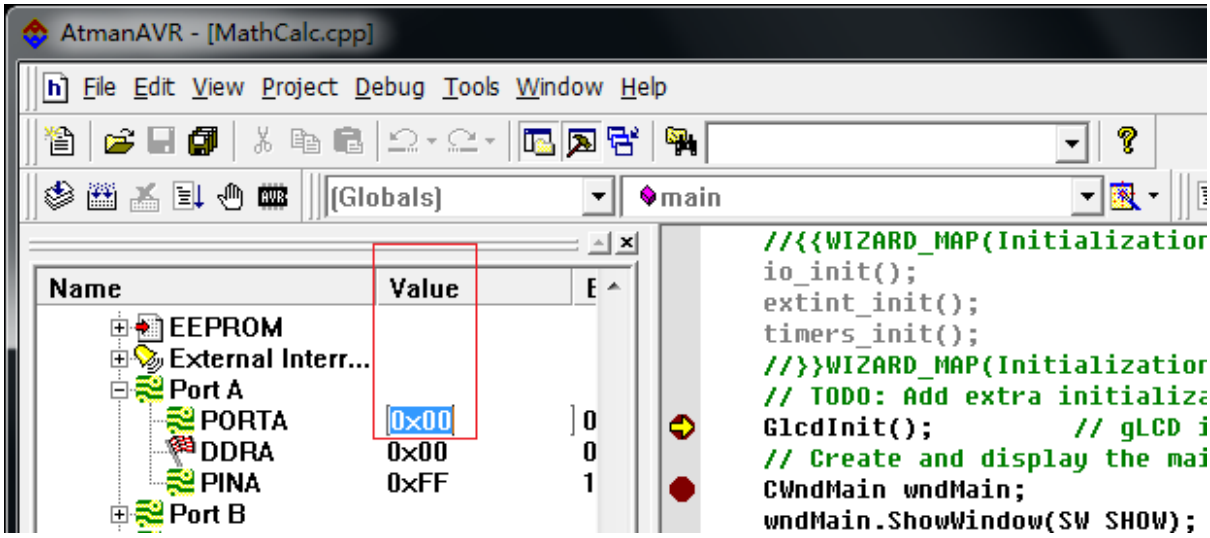

Fig17. Modify IO register

Invert a bit in the IO register

1. Expand the peripheral entry containing the I/O register to be modified in **IoView**
2. Click a bit in the **Bits** column of the I/O register

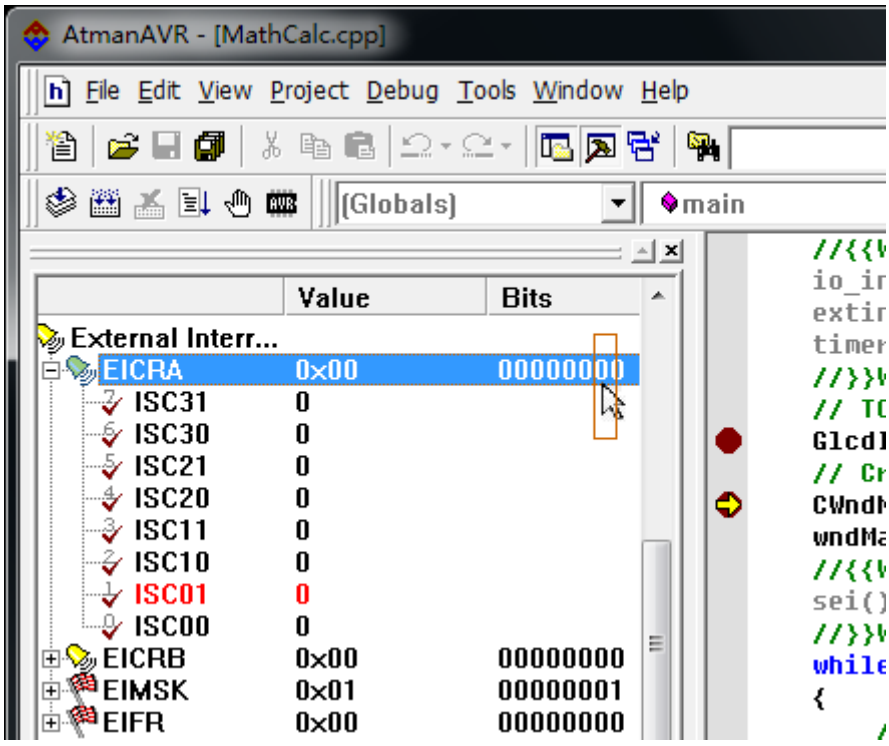

Fig18. Click one of **Bits**

- OR –

1. Click the (+) sign to expand the I/O register.
2. Click the value of the bit you want to invert

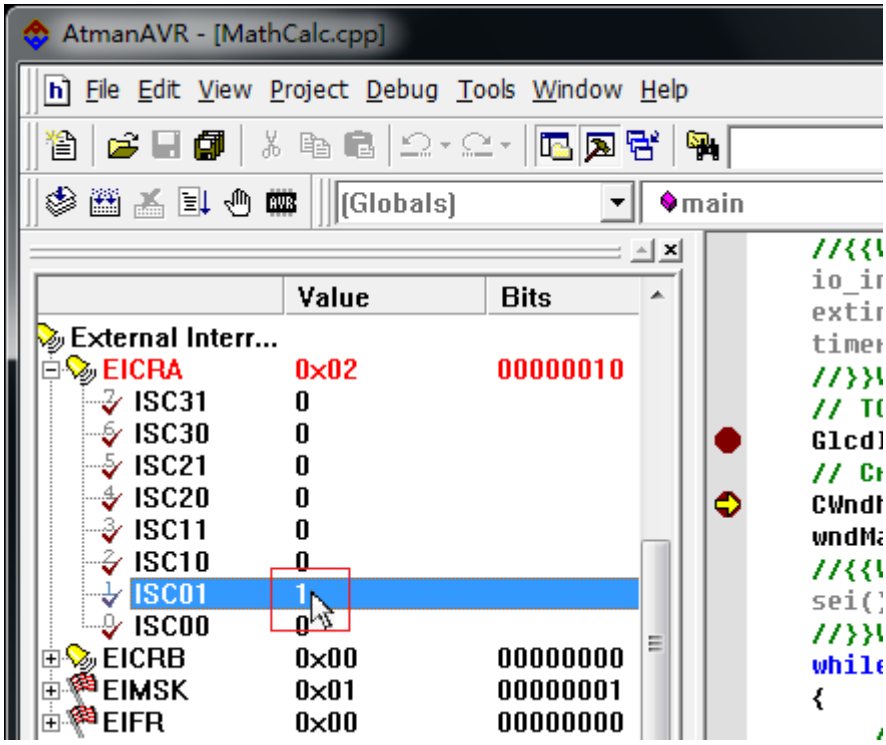

Fig19. Click **Value** of bit

New Project

When creating a new project, AtmanAVR's **Project Wizard** provides a series of dialog boxes for users to configure processor hardware and software resources. Different processors have different options, and the number of setting steps depends on how many peripheral modules the processor supports. However, every step in the **Project Wizard** is not necessary, and you can skip some peripheral settings. When a peripheral needs to be enabled in project development, the peripheral module can be added through the **Code Wizard**.

1. From the **File** menu, choose **New** command

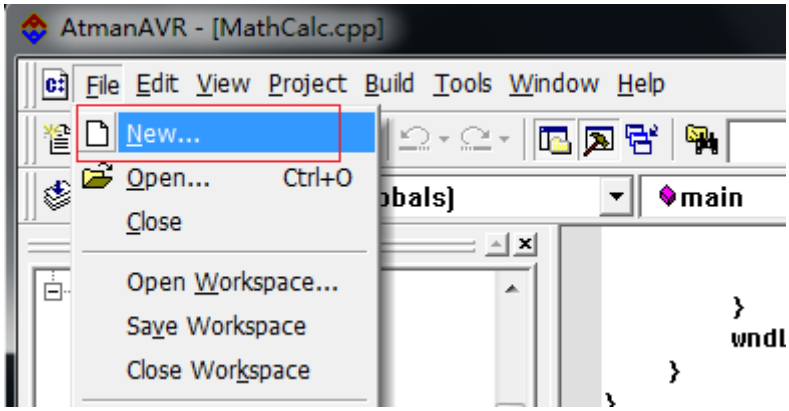

Fig20. **New** command

2. In the **New** dialog box, select the **Projects** page
3. In the project type list on the left, select the project type you want to create.
4. Specify the **Project Name** and storage **Location** on the right, then click **OK** button

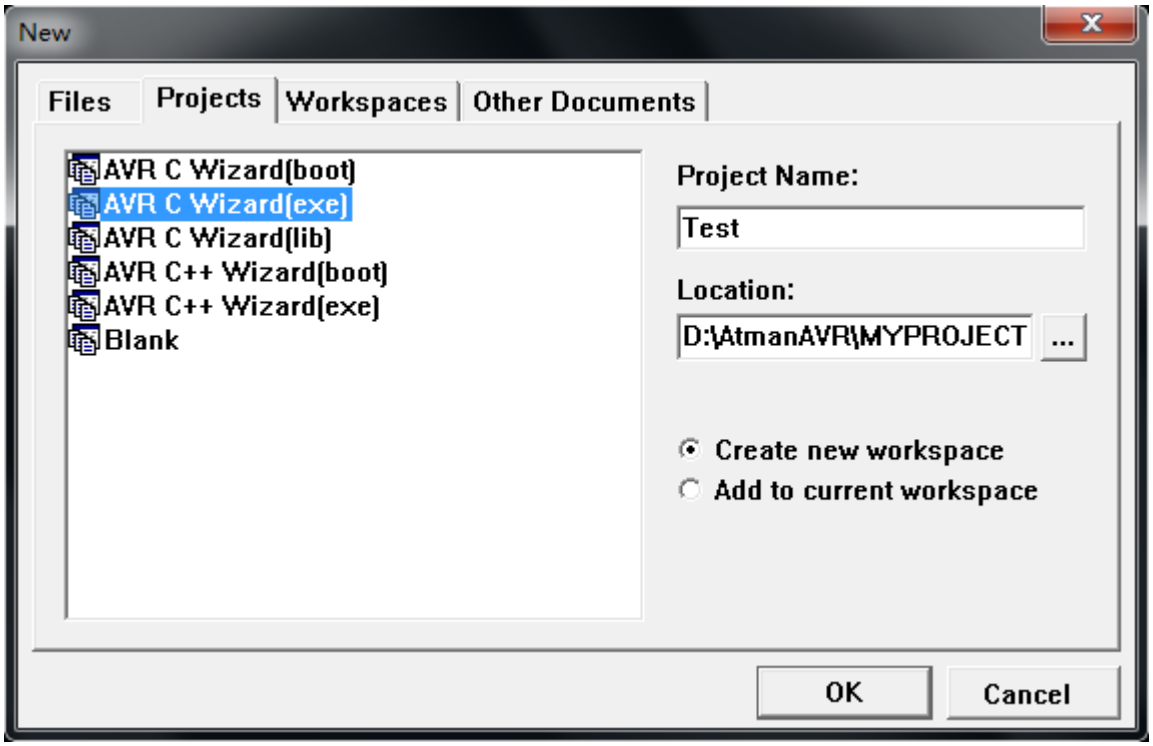

Fig21. **New** window

5. In the pop-up ProjectWizard window, select the **Chip** model, **Clock** frequency and other settings related to the chip.
6. Don't set other peripherals for the time being, and then use the **Code Wizard** to add them as needed. Click the **Finish** button directly

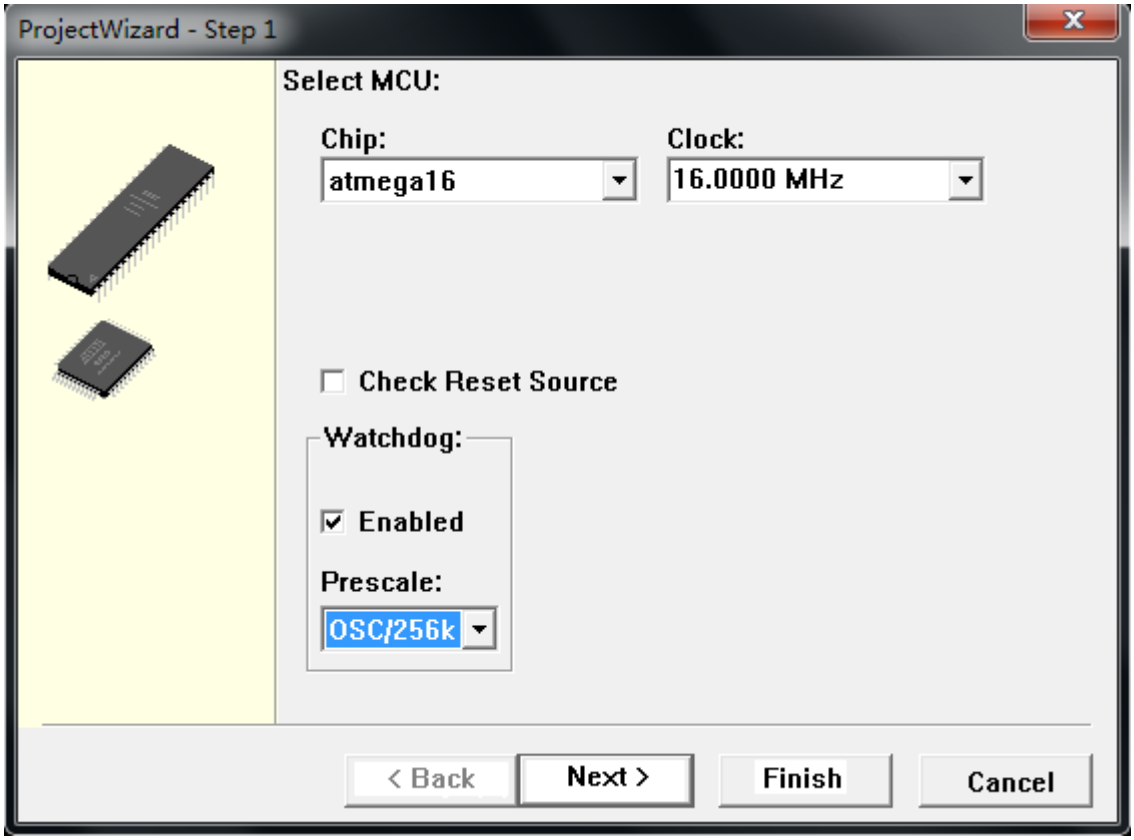

Fig22. Project Wizard

7. AtmanAVR automatically generates the project default files.

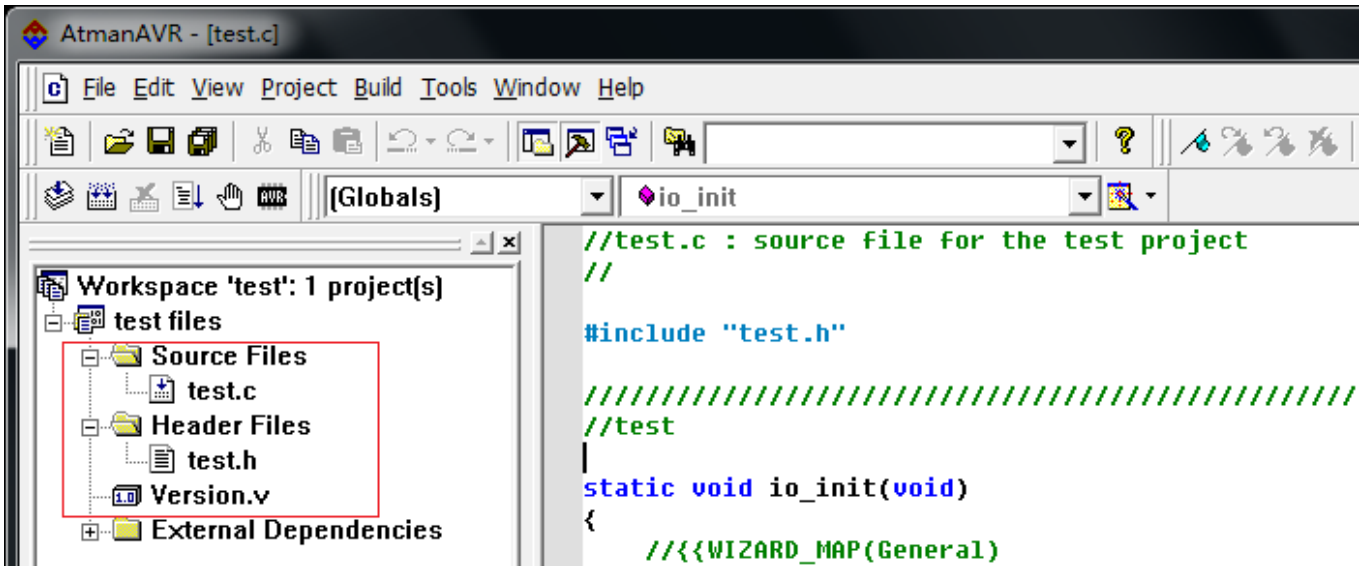

Fig23. Project Wizard generates files

Transplant an existing project

Transplanting an existing project to AtmanAVR is similar to creating a new project.

1. From the **File** menu, choose **New** command
2. In the **New** dialog box, select the **Projects** page
3. In the project type list on the left, select the **Blank** project type
8. Specify the **Project Name** and storage **Location** on the right
4. Click **OK** button

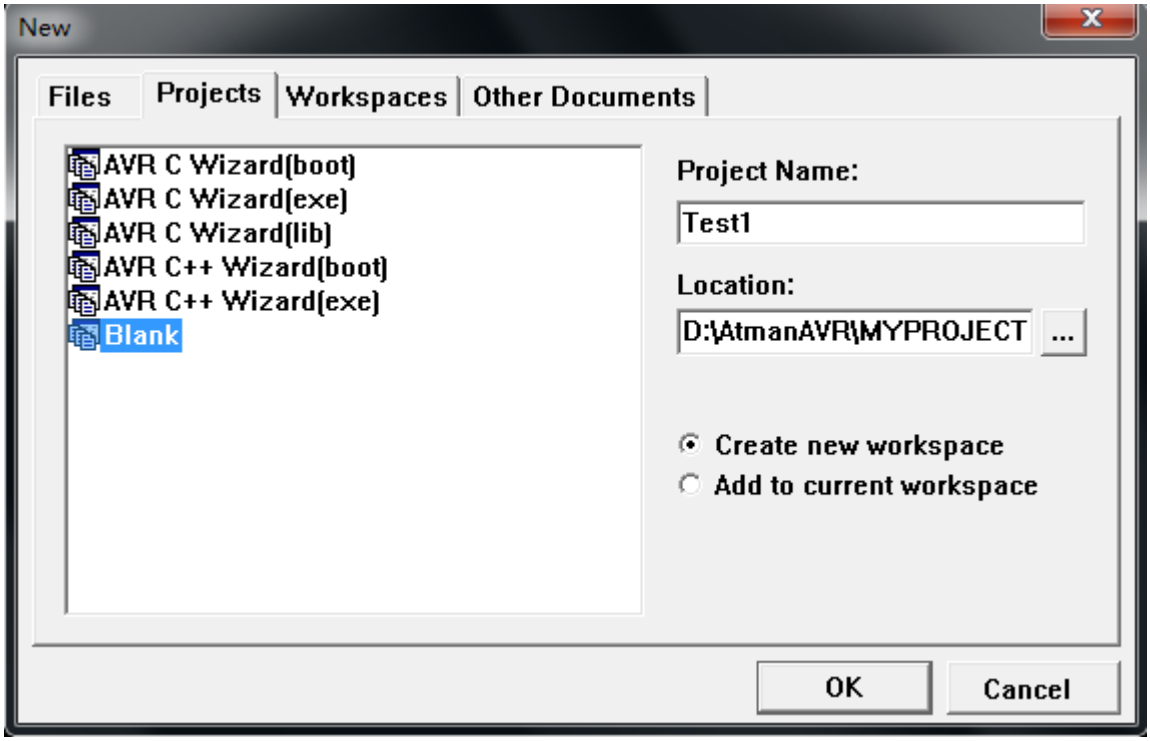

Fig24. **New** window

5. Right-click the newly generated project in **File View**, call the context menu, and select the **Add File to Project** command
6. Select all the source files and header files of the existing project in the open dialog box, and click the **Open** button
7. From the **Project** menu, select the **Settings** command

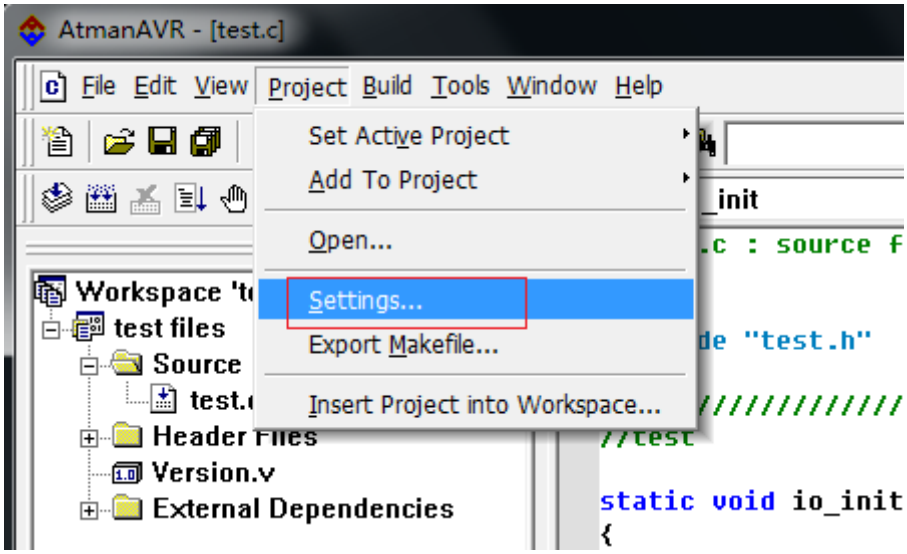

Fig25. Project Settings command

8. Select the **Device** model and **Clock frequency** in the **Settings** dialog box
9. Click **OK** button

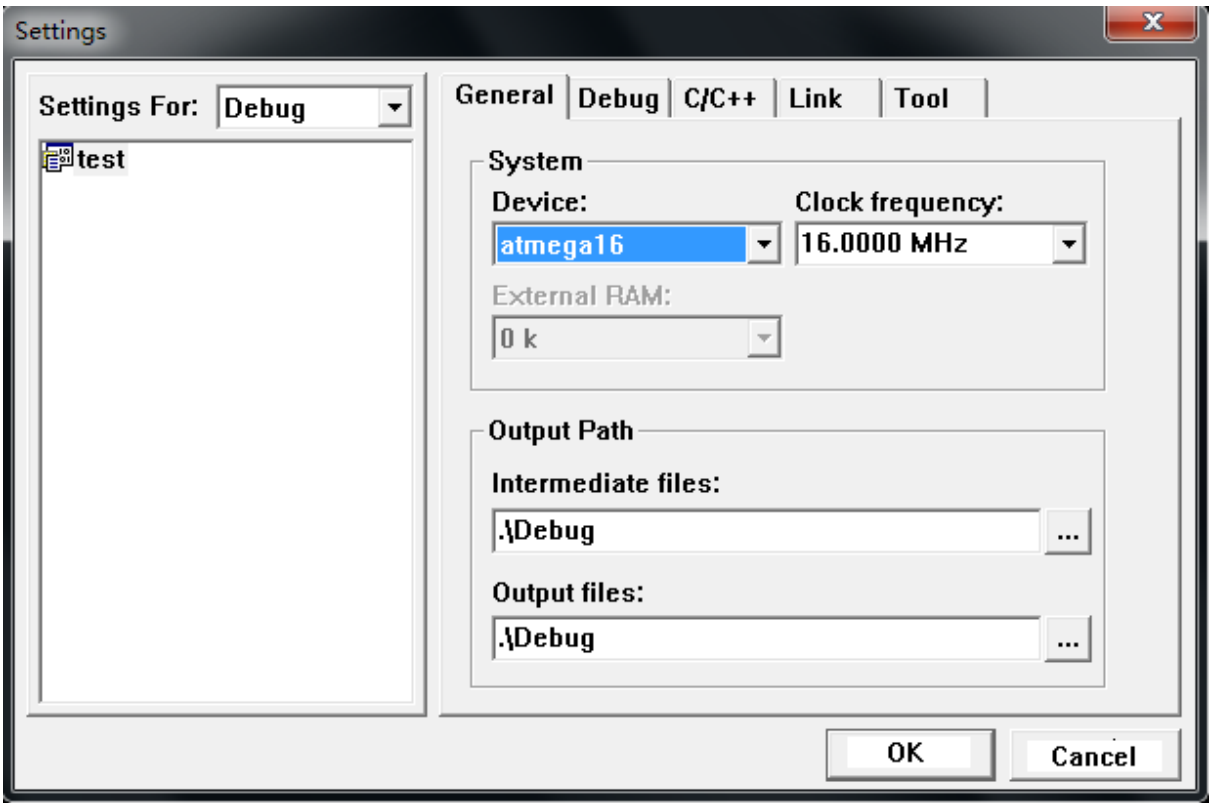

Fig26. Project Settings window

The purpose of the Version.v file

The Version file contains some version information of the project, which you can modify, add and delete.

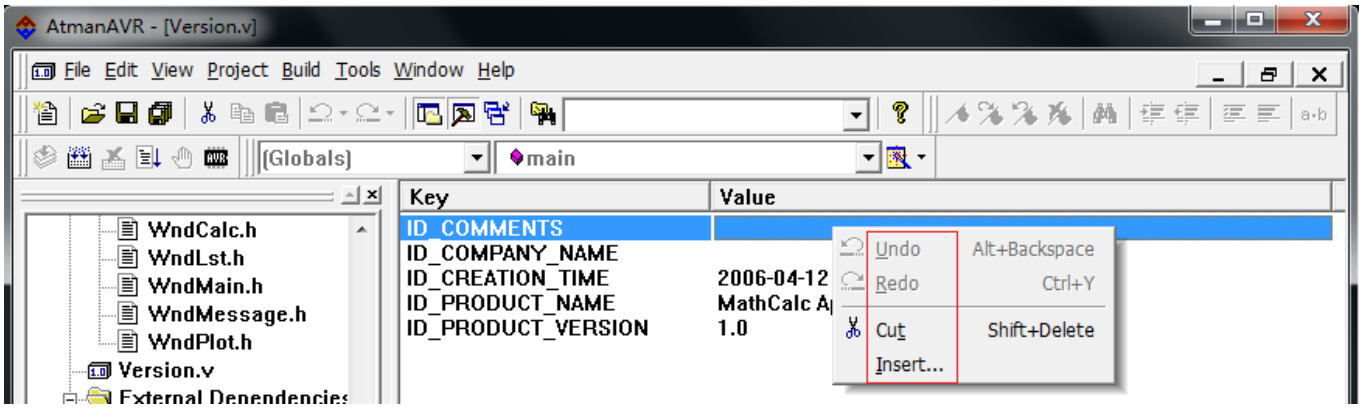

Fig27. Version file context menu

Modify Key value

Double-click this Key to enter the editing state and press **Enter** to finish editing.

Insert Key

Right-click to call the context menu and select **Insert**.

Delete Key

Right-click to call the context menu and select **Cut**.

Use of keys

You can directly refer to these key values in the code, for example:

```
char ver[] = ID_PRODUCT_VERSION;
```

Therefore, the content of the string **ver** is "1.0".

Code Wizard

The Code Wizard can help you set and code the peripherals of AVR conveniently, even if you are not familiar with it, and you don't even have to memorize the functional definitions of its IO registers.

Add peripheral module

1. Select **CodeWizard** from the menu **Tools** or from the context menu of the source window

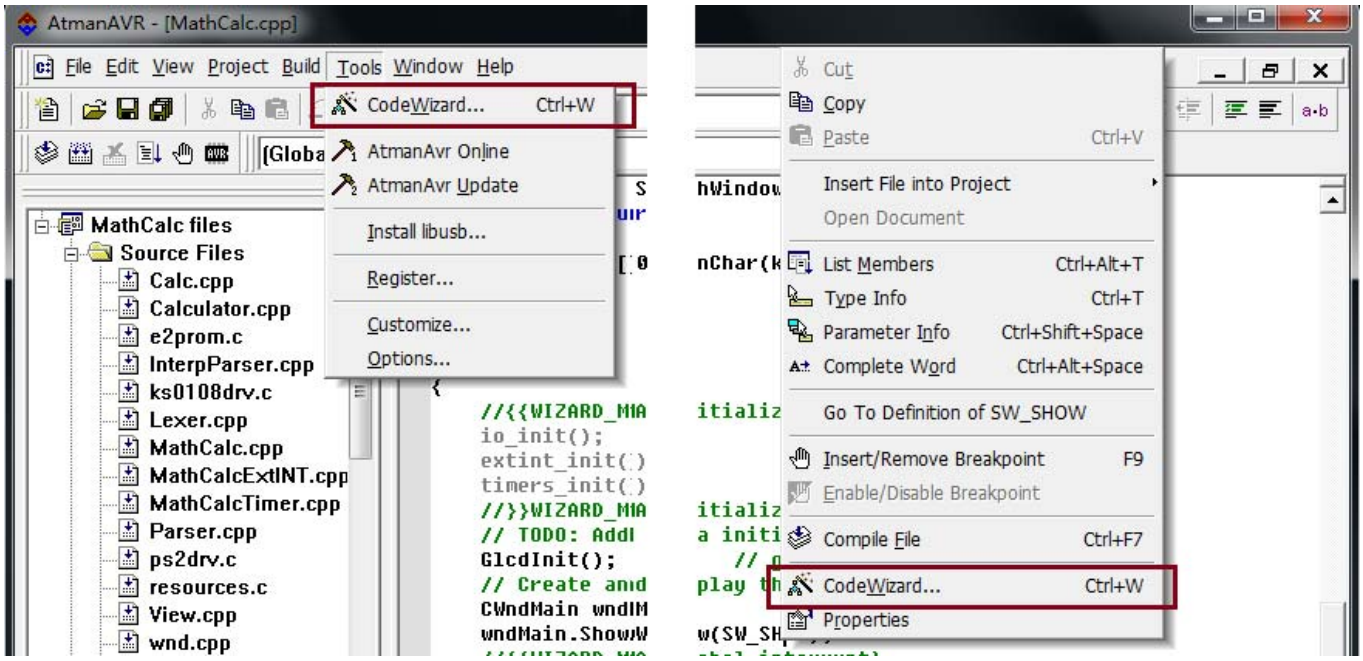

Fig28. Invoke Code Wizard（Tools & Edirtor）

2. Select the peripheral to be added from the **Modules** list in the Code Wizard window, such as ADC
3. Click the button **Add Module** on the right

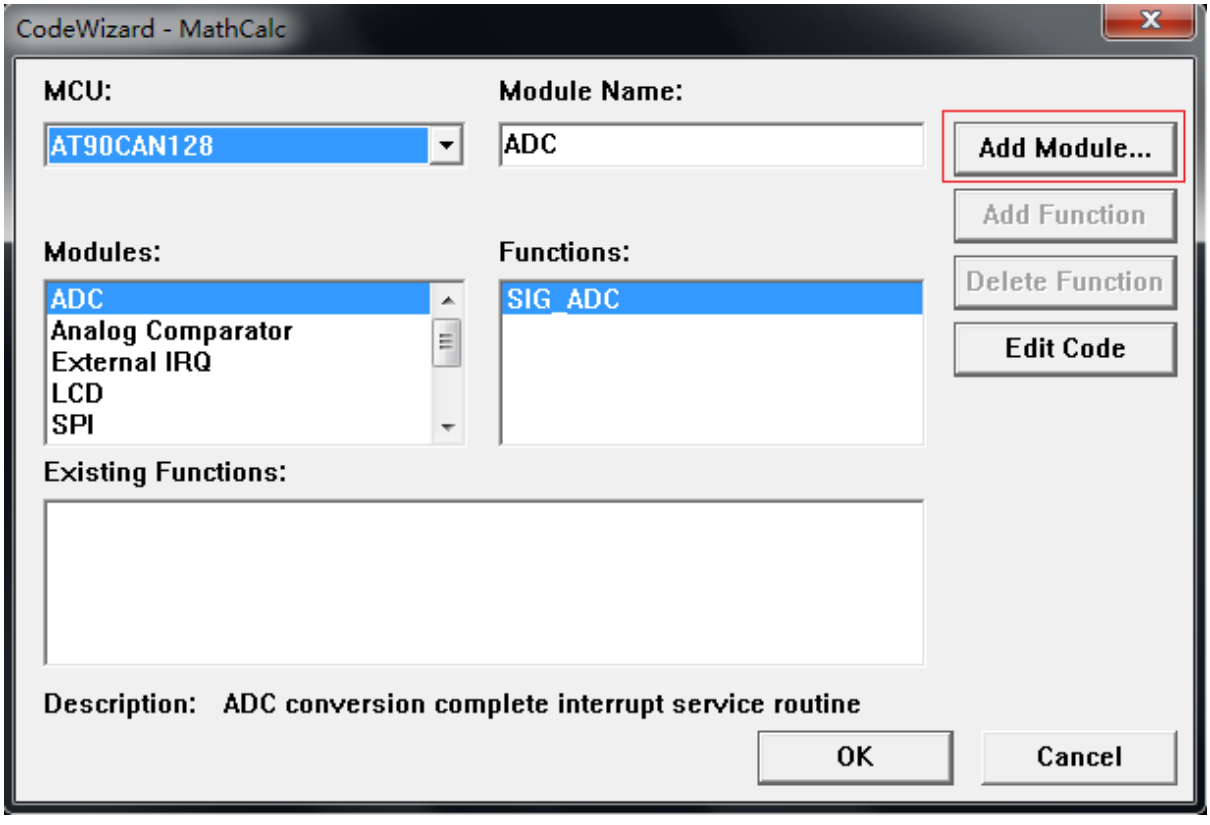

Fig29. Code Wizard

4. Set the options of peripherals in the Project Wizard that pops up
5. Click **Finish** button

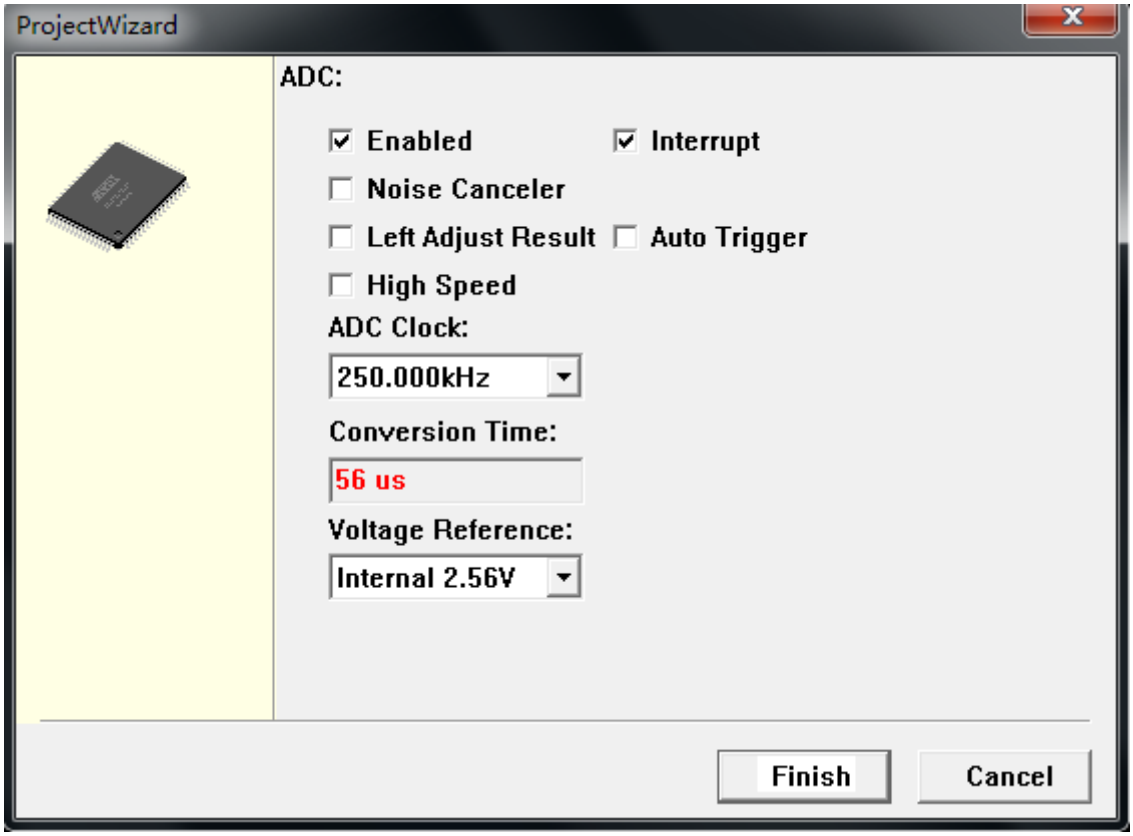

Fig30. Peripheral configuration

6. At this point, the peripheral module files has been automatically generated, and the module header file is included in the main source file and the initialization function of the module is called in the main function

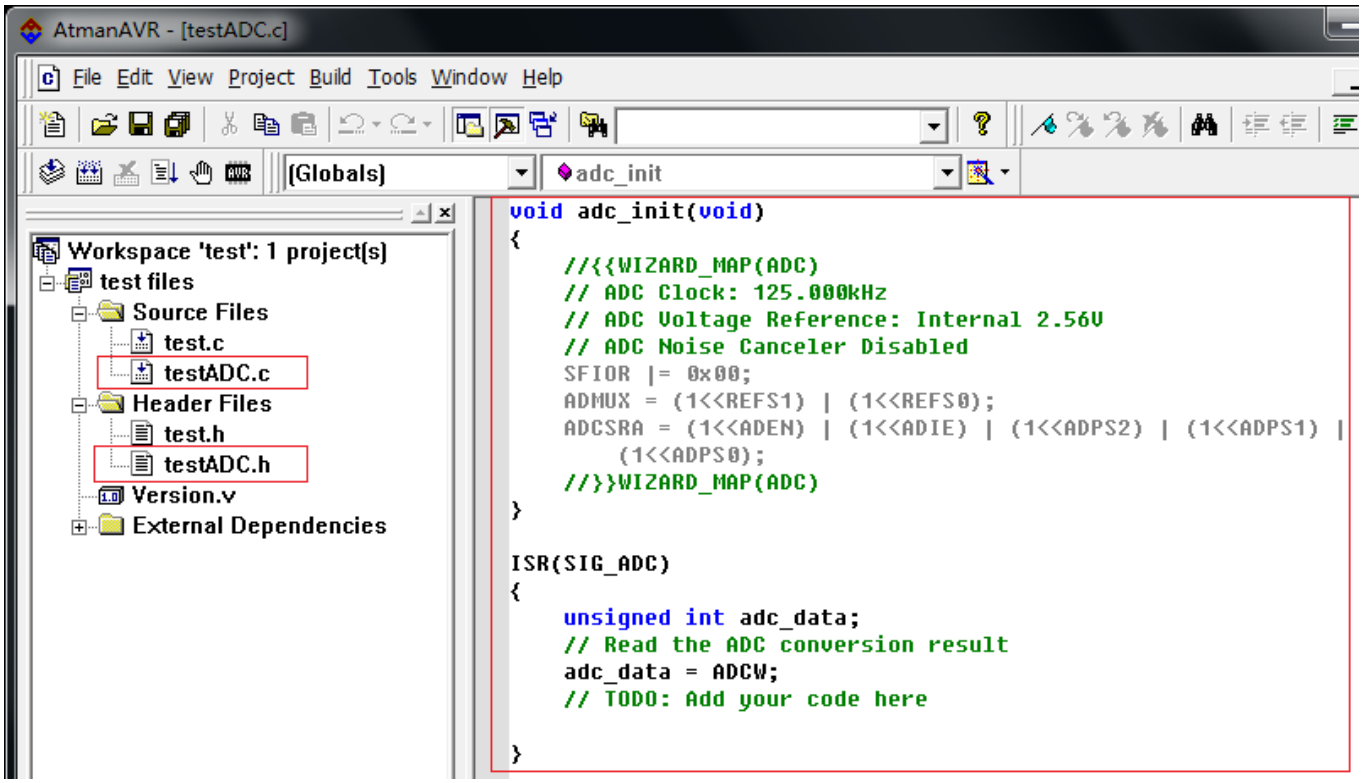

Fig31. Automatic generation of peripheral module files

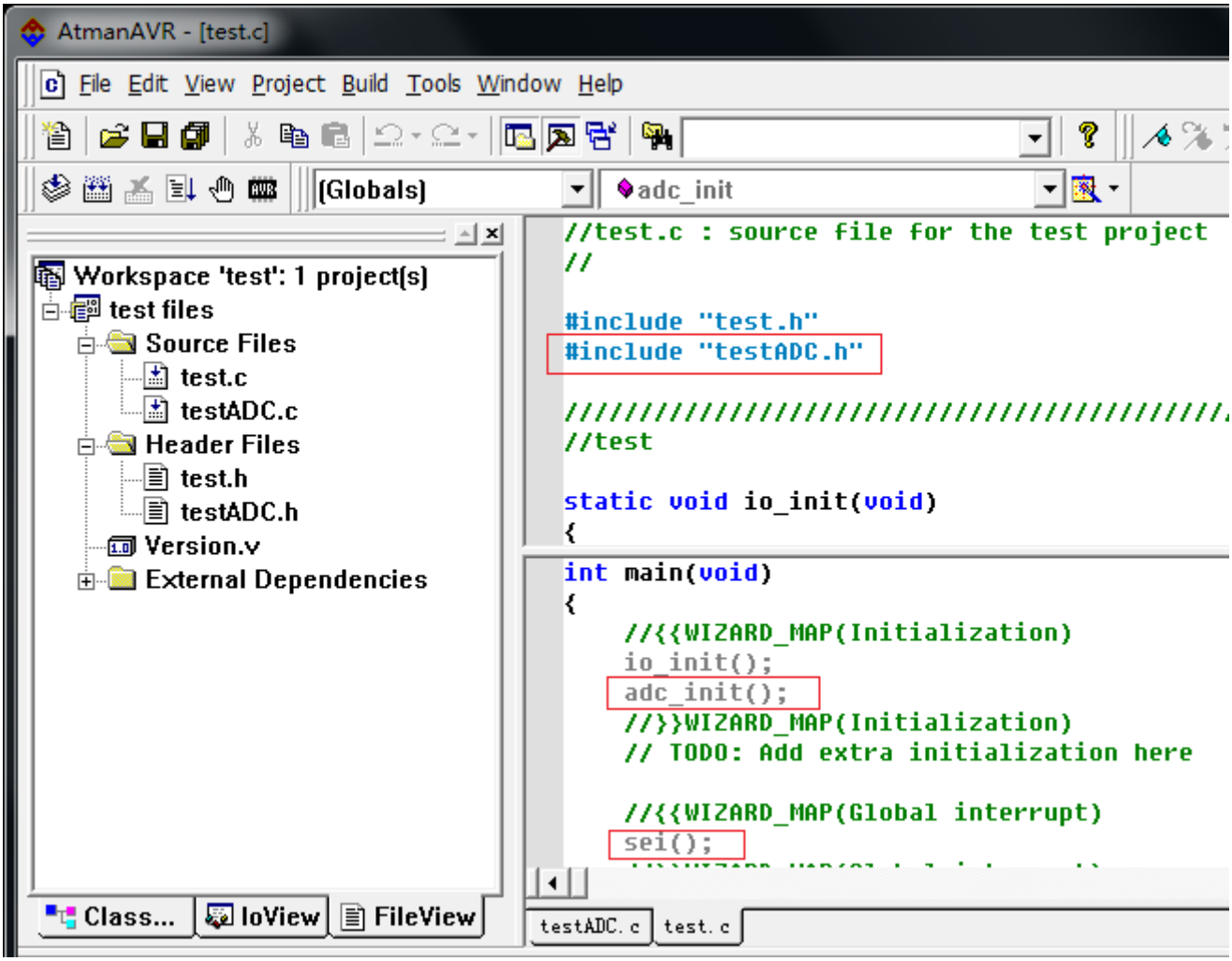

Fig32. Automatically include the module header file, and call the initialization function of the module

7. The window returns to the Code Wizard, and you can repeat steps 2 to 5 to add other peripheral modules

Change peripheral module configuration

In the current version, to change the configuration of existing peripherals in the project, you can:

1. Save the current peripheral module source file as another file name, if the file has your extra code
2. Delete the source file of the peripheral module from the **File View** in the workspace
3. Invoke the Code Wizard to reconfigure the peripheral module

Source Editor

AtmanAVR's source editor is a comprehensive text editor, which supports automatic filling of code syntax by selecting from generated class members, parameters or value lists. Support automatic prompt function parameter information; Support automatic intelligent indentation of C/C++ syntax code format. etc.. The following only introduces practical but inconspicuous shortcut functions.
You can enter some practical functions of the source editor through the menu **Edit** -> **Advanced**.

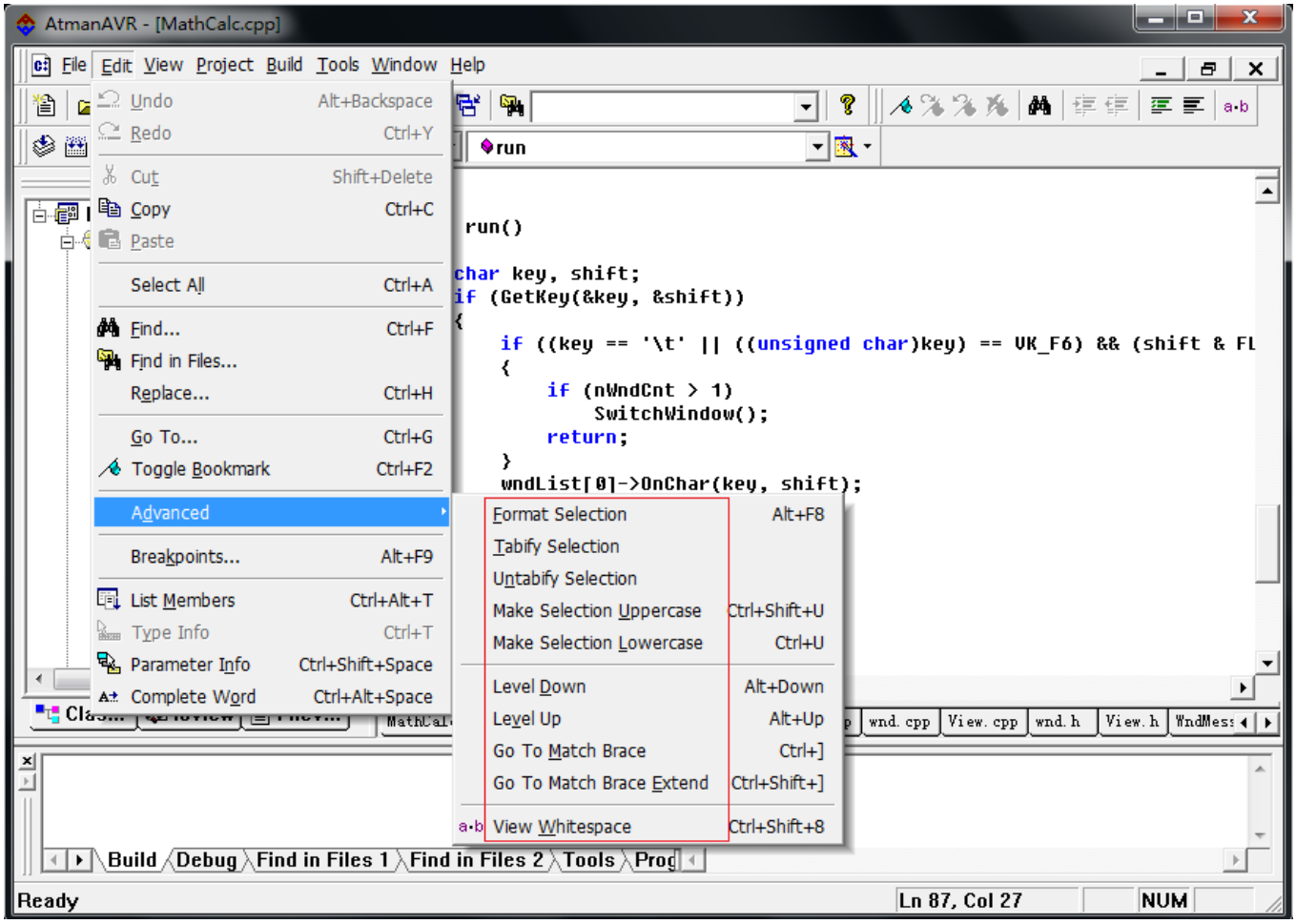

Fig33. Advanced function commands of the source editor

Toggles the case of the selected text

1. Select the text to switch case
2. Press the shortcut key **Ctrl+Shift+U** to change to all uppercase. or
   Press the shortcut key **Ctrl+U** to change to all lowercase.

Increase/decrease indent of the selected text

1. Select the text lines whose indent you want to change
2. Click the  or  button in the **Edit** toolbar

Comment/uncomment the selected text

1. Select the text lines to comment or uncomment
2. Click the  or  button in the **Edit** toolbar

Line number tips

Method 1. Place the mouse on the left margin of the editing window, and the line number of the line where the mouse is located will be displayed.

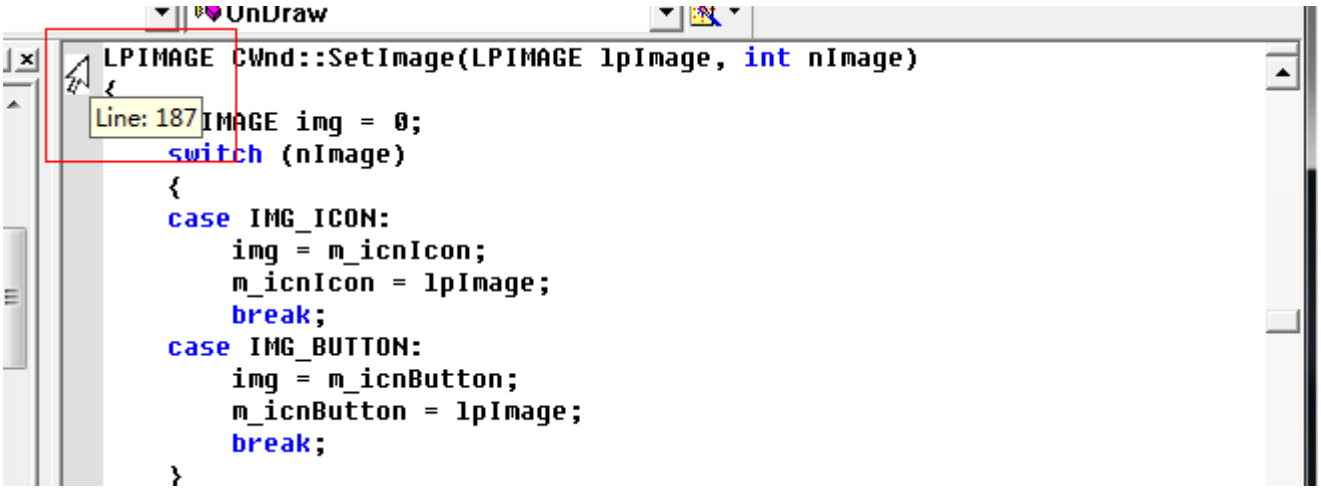

Fig34. Line number display

Method 2. Press the left mouse button on the scroll bar on the right side of the editing window, and the line number of the top line of the current window will be displayed, which will be updated with the drag of the mouse.

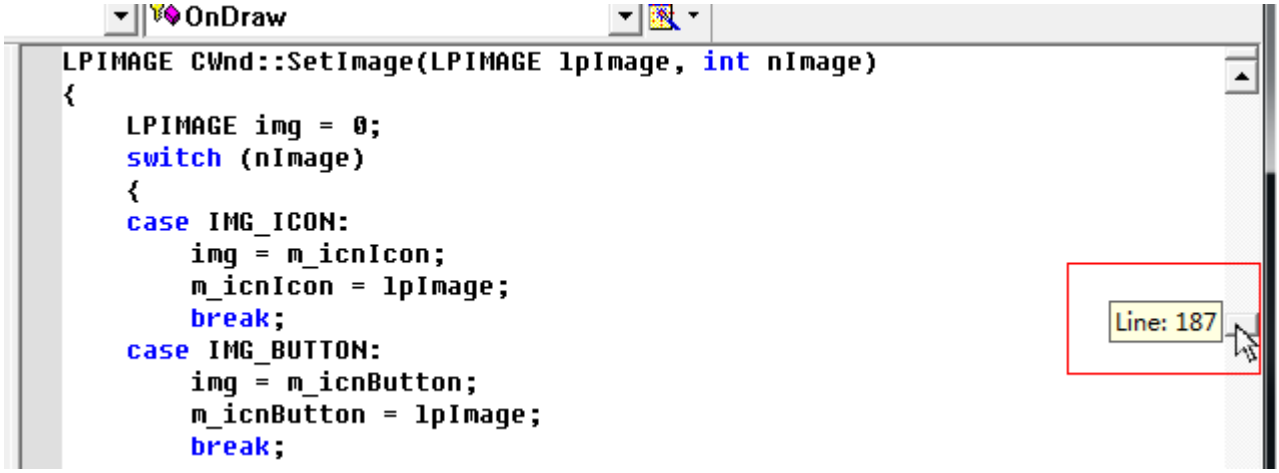

Fig35. Top line number display

Locate error or warning code lines

When compiling files or building projects, error and warning messages are output in the **Build** view of the **Output** window, indicating the file and line number. To locate a line of code, you can:

Method 1. Double-click the prompt message in the **Build** view.

Method 2. Press **F4** to locate the next information line and press **Shift+F4** to locate the previous information line.

Locate the Find in Files result

After executing the **Find in Files** command, the search results are output in the **Find in Files N** view. To locate the result row, you can:

Method 1. Double-click the prompt message in the **Find in Files N** view.

Method 2. Press **F4** to locate the next information line and press **Shift+F4** to locate the previous information line.

Debug

AtmanAVR provides a comprehensive debugger to help debug and simulate executable programs. When debugging, put the mouse pointer on the variable or the selected expression, and the value of the variable or expression will be displayed.
Note: The values of all memory, registers and variables are refreshed only when the debugger is paused. Every step will be refreshed in **Auto Step** mode.

Display the value of an expression in Watch window

To monitor the expression in the Watch window, you can type the expression in the Watch window, or drag the expression directly to the Watch window.

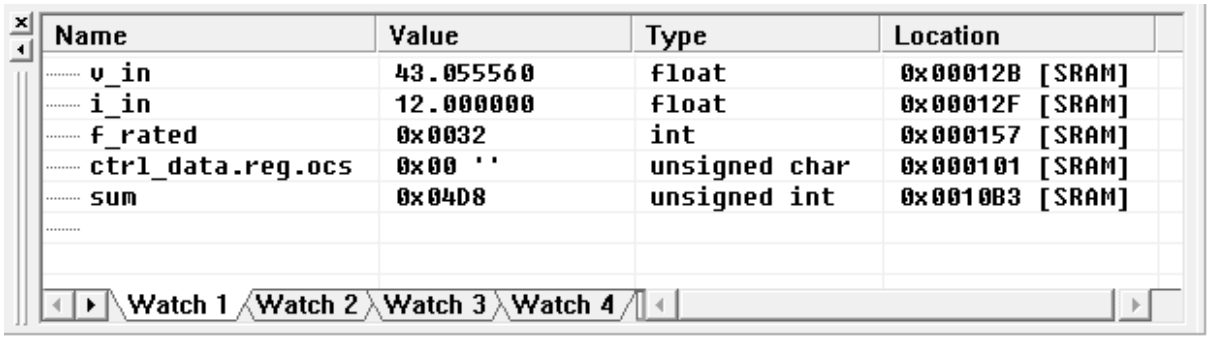

Fig36. Watch window

Display variables in the Memory window

1. Type or paste the hexadecimal address of the variable in the **Address** box of the Memory window. Or, more intuitively, Type or paste the variable name directly in the **Address** box
2. Press **Enter**
3. The Memory window will display the memory content of the variable at the beginning of the first byte in the first line.

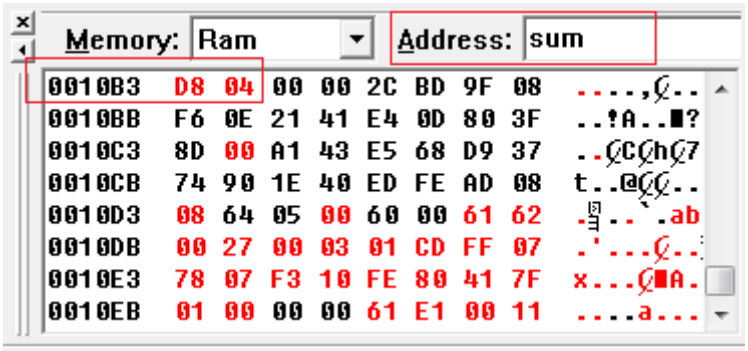

Fig37. Memory window

Test Timer timing length

1. Place a breakpoint at the beginning of timer interrupt function
2. Run the debugger to the breakpoint
3. Double-click the value of **Stop Watch** in **IoView** to clear it

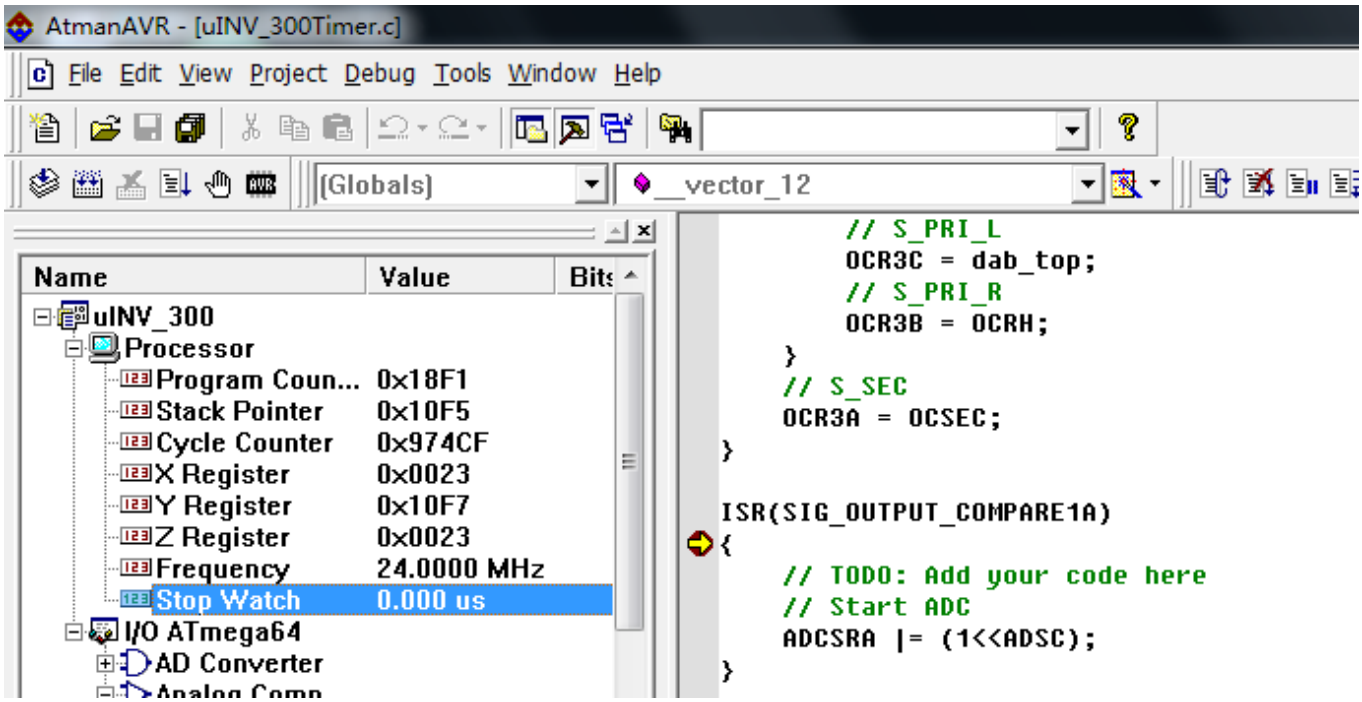

Fig38. Clear Stop Watch

4. Run the debugger to the breakpoint again
5. Now, the value of **Stop Watch** is the timer timing duration

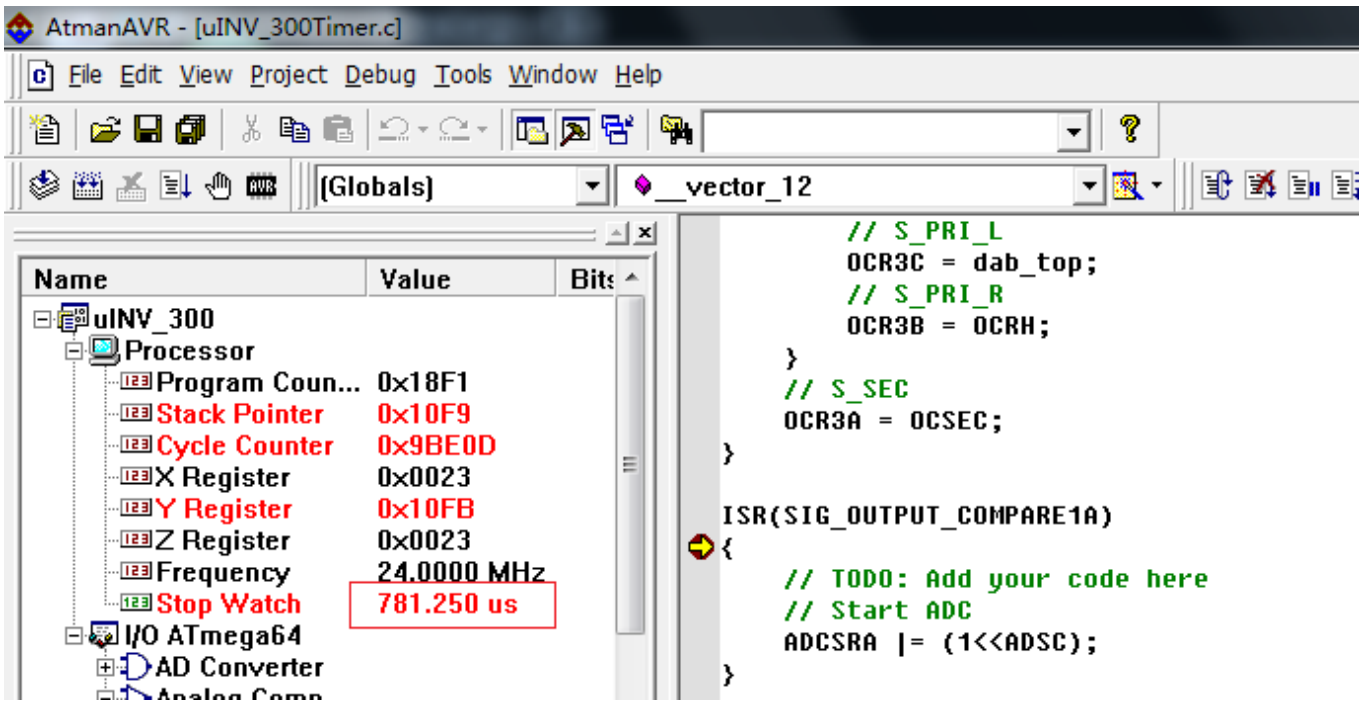

Fig39. StopWatch display duration

Simulate ADC

When debugging ADC code, you can use the I/O Interface window provided by the debugger to simulate and test the performance of the algorithm.
The following settings are automatically remembered when the project exits and automatically restored when the project is reloaded.

1. Start the debugger
2. From the menu **View** -> **Debug Windows**, select **I/O Interface** to display this window.
3. Select the **Analog** tab and slide down to ADC
4. Double-click the Level value of **AREF(V)** to enter the editing state, and type the ADC reference voltage value selected in the project
5. Similarly, type the analog voltage value of the ADC channel used
6. Continue to run the debugger

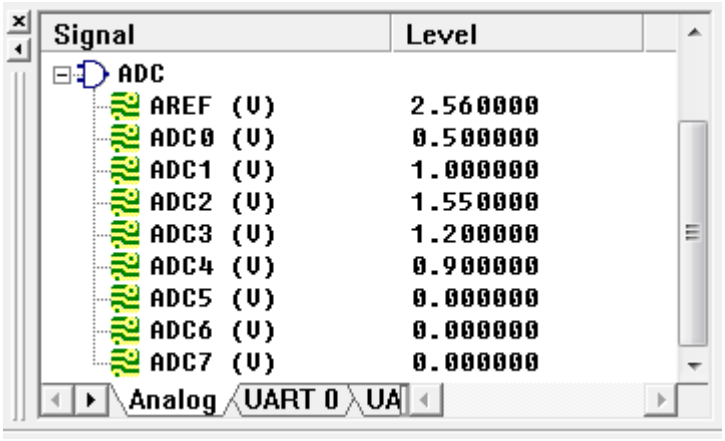

Fig40. ADC Interface

Simulate Analog Comparator

The I/O Interface window provided by the debugger can also simulate an analog comparator.

1. Start the debugger
2. From the menu **View** -> **Debug Windows**, select **I/O Interface** to display this window
3. Select the **Analog** tab and slide up to AC
4. Double-click the Level value of AIN0(V) to edit, and type in the analog voltage value
5. Similarly, type the analog voltage value of the input of AIN1(V) used.

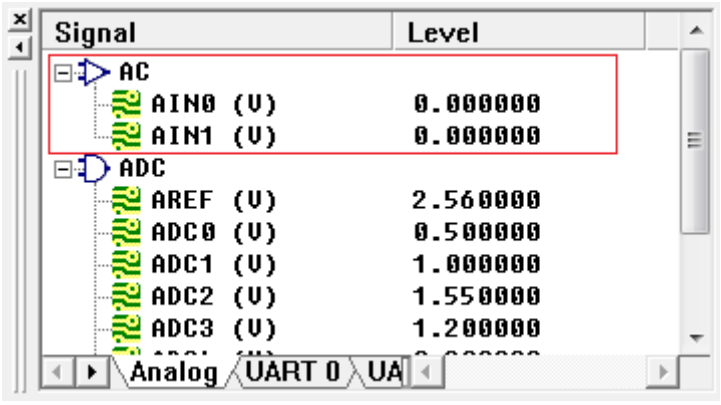

Fig41. AC Interface

Simulate UART

The I/O Interface window provided by the debugger supports simulation of UART peripherals. The sampling control logic of UART simulation interface is the same as that of UART peripheral of AVR. It can display the serial port output of the project code, or input it to the serial port of the project code. It can be regarded as a terminal where you can communicate with the serial port of the debugger. The terminal displays the received data in gray and the transmitted data in black.

Simulate universal serial communication

1. Start the debugger
2. From the menu **View** -> **Debug Windows**, select **I/O Interface** to display this window
3. Select the corresponding UART(n) tab
4. Select the display format of input and output
5. Type or paste the data to be sent in the window, and press **Enter** to start sending
6. The response output of your UART code will be displayed in gray data

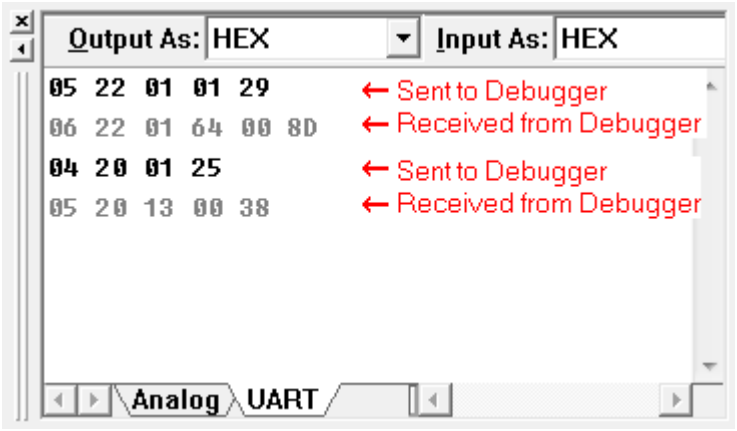

Fig42. UART simulation interface

Simulate multi-processor serial communication

1. Start the debugger
2. From the menu **View** -> **Debug Windows**, select **I/O Interface** to display this window
3. Select the corresponding UART(n) tab
4. Select the display format of input and output
5. Type or paste the address frame (0x100+address) and data frame to be sent in the window, and press **Enter** to start sending
6. The response output of your UART code will be displayed in gray data

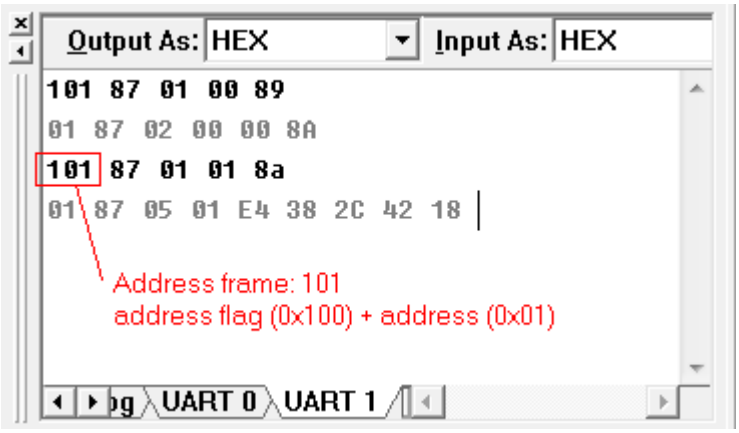

Fig43. Multi-processor Communication simulation

Simulate LCD

The LCD window can be used to simulate the character LCD based on HD44780.

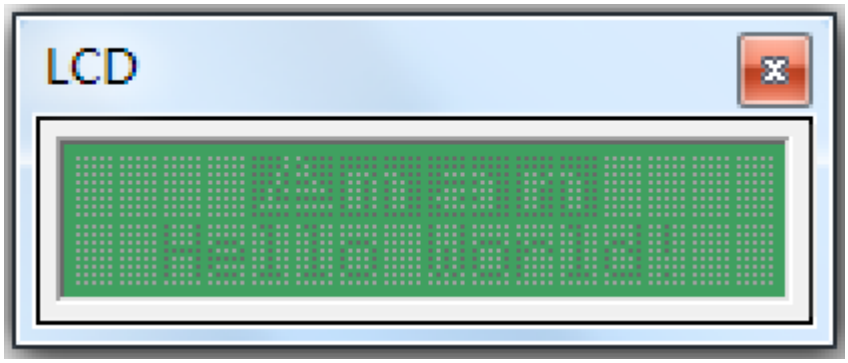

Fig44. LCD simulation example

Connect LCD to MCU

1. Start the debugger
2. From the menu **View** -> **Debug Windows**, select **LCD** to display this window
3. Right-click the LCD window, invoke the context menu, and select the **Properties** command
4. Specify the CPU port and pin for each pin of LCD in the **CPU Port** combo box and **Pin** combo box respectively.
5. Click **OK**

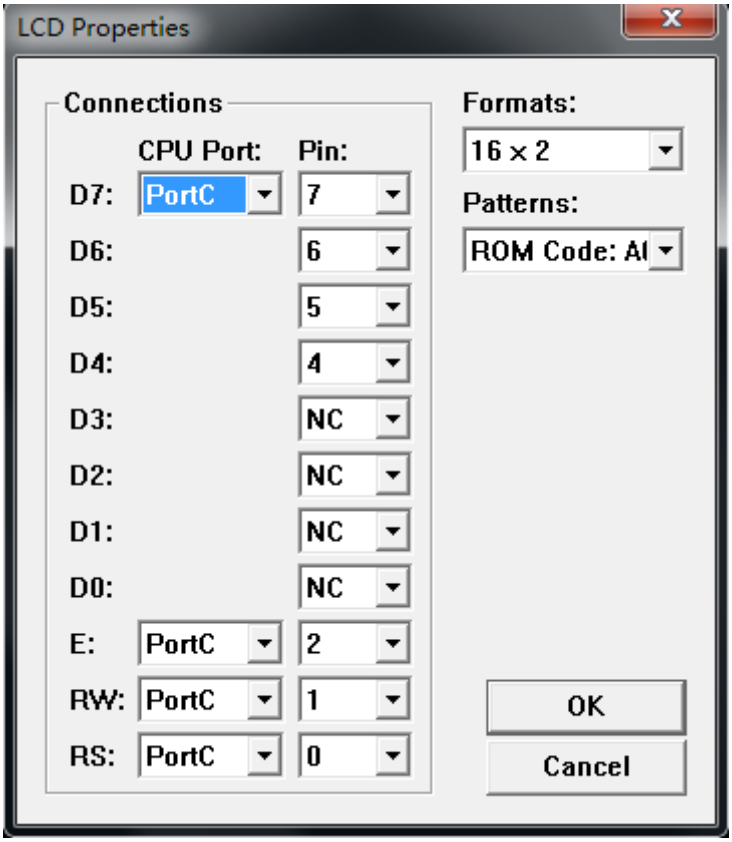

Fig45. Connect LCD

Set LCD type and character generator pattern

1. Start the debugger
2. From the menu **View** -> **Debug Windows**, select **LCD** to display this window
3. Right-click the LCD window, invoke the context menu, and select the **Properties** command
4. Specify the LCD type in the Formats combo box and the character generator mode in the Patterns combo box.
5. Click **OK**

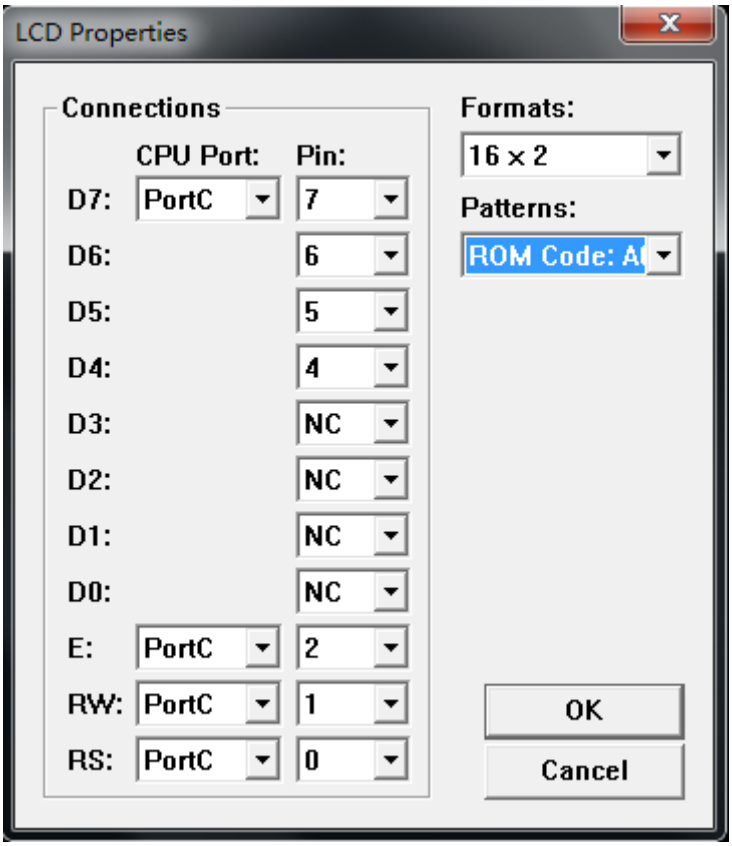

Fig46. Select LCD

Simulate LED

You can simulate 8-segment LEDs by using the LED window. You can specify the LED type (common cathode or common anode) and its debugging behavior.

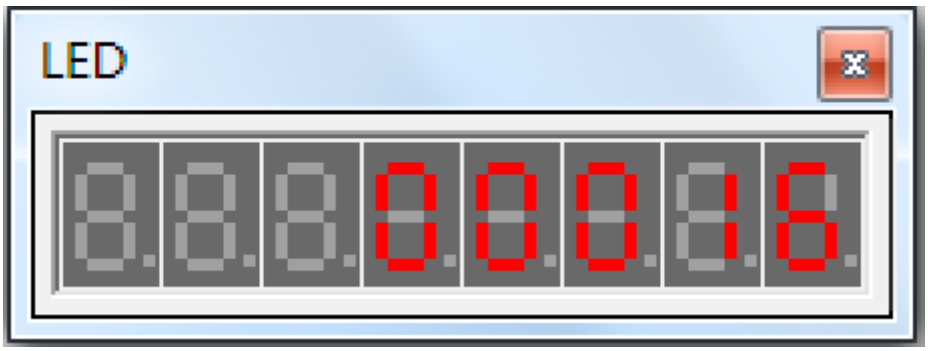

Fig47. LED simulation example

Connect LED to MCU

1. Start the debugger
2. From the menu **View** -> **Debug Windows**, select **LED** to display this window
3. Right-click the LED window, invoke the context menu, and select the **Properties** command
4. Select the LED to be used in the LED Index combo box
5. Specify the CPU Port and Pin for each pin of LED in the **CPU Port** combo box and **Pin** combo box respectively.
6. Repeat steps 4 and 5
7. Click **OK**

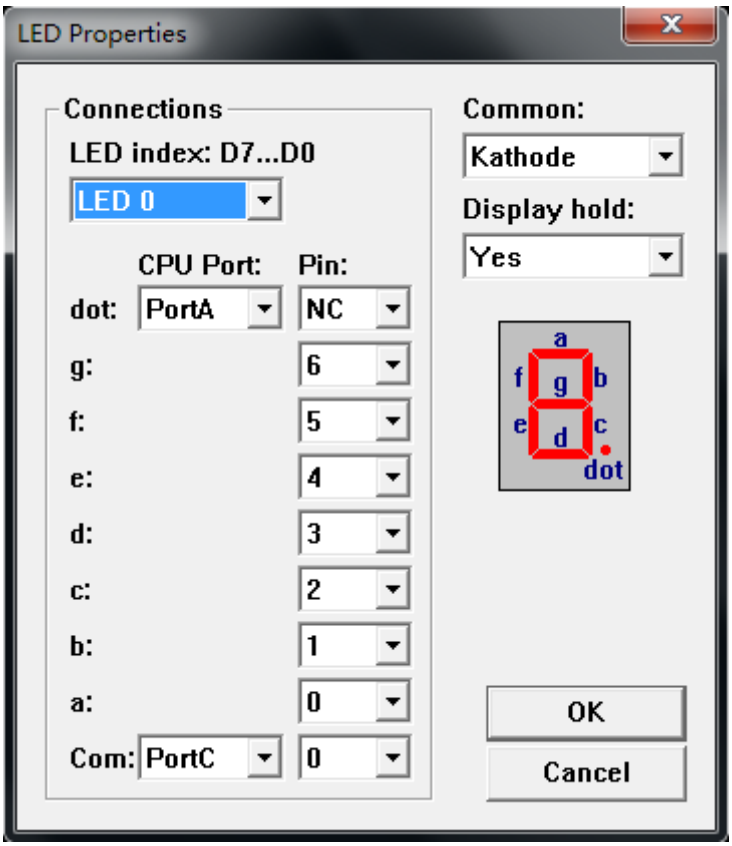

Fig48. Connect LED

Tip You can use the context menu (right-click dialog box) to copy the connection of the current LED and paste it into another LED.

Specify the type of LED and its debugging behavior

1. Start the debugger
2. From the menu **View** -> **Debug Windows**, select **LED** to display this window
3. Right-click the LED window, invoke the context menu, and select the **Properties** command
4. Select the type of LED in the **Common** combo box
5. Specify the debugging behavior in the **Display hold** combo box, and specify whether to keep the display
6. Click **OK**

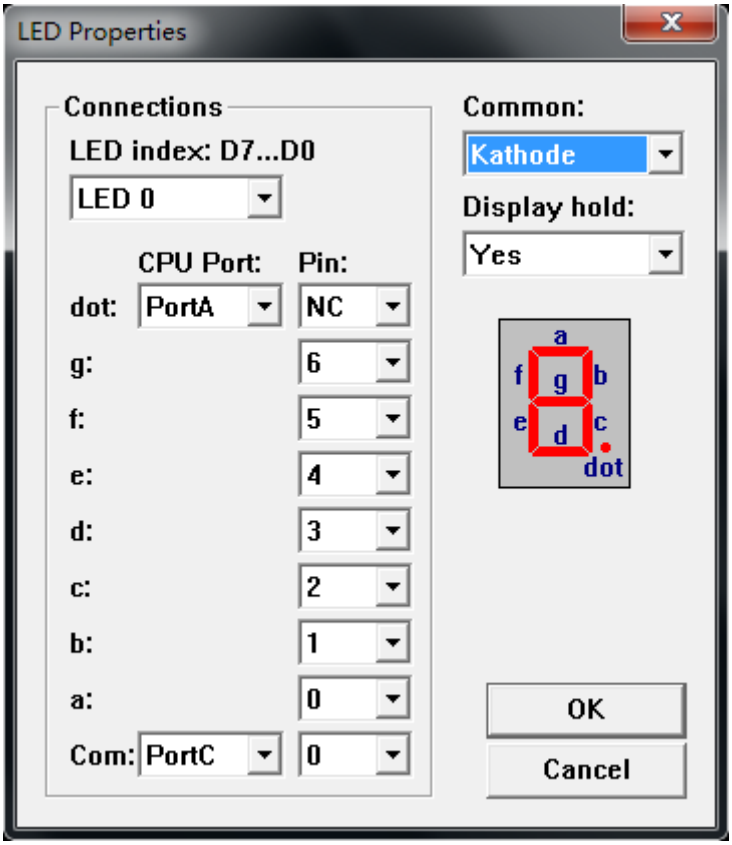

Fig49. Select LED

Tip When using dynamic display multi-bit LED, after setting the display hold, the phenomenon of display flashing bit by bit can be avoided during debugging.

Simulate Keyboard

Keyboard windows can be used to simulate keys. You can set the delay time for pressing the key. The running speed of the debugger is generally much slower than the real speed of the code. After clicking a key, the simulator has to keep the key pressed for a period of time. The delay time set by the Keyboard window refers to the running time of CPU.

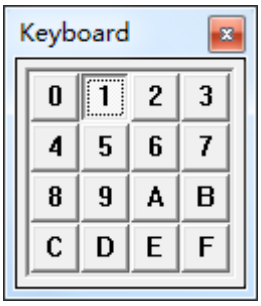

Fig50. Keyboard simulation

Connect Keyboard to MCU

1. Start the debugger
2. From the menu **View** -> **Debug Windows**, select **Keyboard** to display this window
3. Right-click the Keyboard window, invoke the context menu, and select the **Properties** command
4. Select the button to use in the **Button** combo box
5. Specify the CPU Port and Pin for each pin of the button in the **CPU Port** combo box and **Pin** combo box respectively
6. Repeat steps 4 and 5
7. Click **OK**

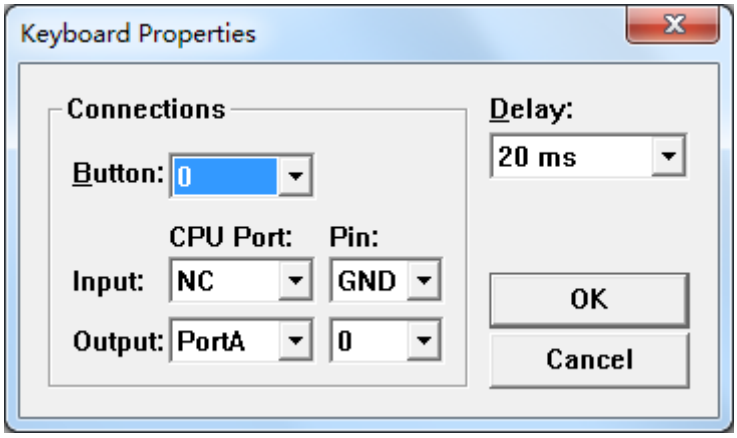

Fig51. Connect Keyboard

Set the holding time when the key is pressed

1. Start the debugger
2. From the menu **View** -> **Debug Windows**, select **Keyboard** to display this window
3. Right-click the Keyboard window, invoke the context menu, and select the **Properties** command
4. In the **Delay** box, select the holding time when the key is pressed
5. Click **OK**

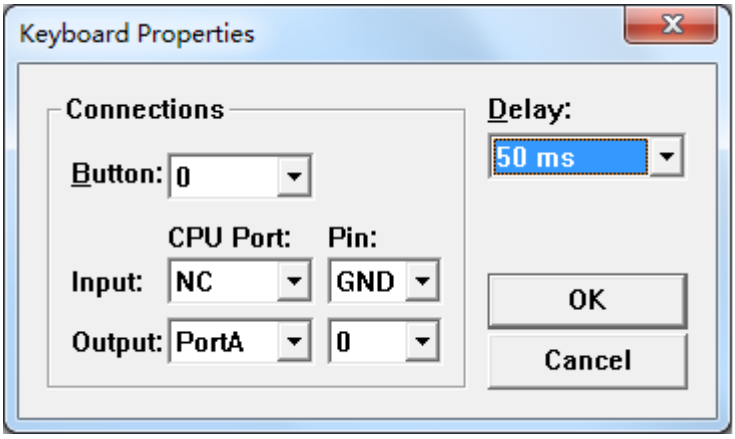

Fig52. Set Keyboard