



FPPA IDE

Integrated Development Environment

User Manual

Version 1.02 - August 23, 2019

Copyright © 2019 by PADAUK Technology Co., Ltd., all rights reserved

6F-6, No.1, Sec. 3, Gongdao 5th Rd., Hsinchu City 30069, Taiwan, R.O.C.

TEL: 886-3-572-8688  www.padauk.com.tw

IMPORTANT NOTICE

PADAUK Technology reserves the right to make changes to its products or to terminate production of its products at any time without notice. Customers are strongly recommended to contact PADAUK Technology for the latest information and verify whether the information is correct and complete before placing orders.

PADAUK Technology products are not warranted to be suitable for use in life-support applications or other critical applications. PADAUK Technology assumes no liability for such applications. Critical applications include, but are not limited to, those which may involve potential risks of death, personal injury, fire or severe property damage.

PADAUK Technology assumes no responsibility for any issue caused by a customer's product design. Customers should design and verify their products within the ranges guaranteed by PADAUK Technology. In order to minimize the risks in customers' products, customers should design a product with adequate operating safeguards.

Table of content

1. Introduction	5
2. Integrated development system software IDE.....	6
2.1 IDE installation	6
3. The usage of FPPA IDE Integrated development system	7
3.1 The usage of IDE	7
3.2 Building a new ASM project.....	8
3.3. Adding an old ASM file to the new project.....	10
3.4. Program execution and debugging	12
3.5. Building a new Mini-C project	14
4. Introduction of FPPA™ IDE's environment and window	20
4.1. Mini-C's assembly environment.....	20
4.2. The assembly information window of FPPA™ IDE.....	21
4.3. Development tool's workspace management and debugging environment.....	22
4.3.1. FileView	22
4.3.2. LabView	24
4.3.3. Debugging environment of development tool	25
4.3.4. Disassembly code of development tool	30
4.4. Practical example	31

Revision History:

Revision	Date	Author	Description
V1.00	2013/04/10	Kent	Preliminary version
V1.01	2013/05/10	Charley	Update UI figures
V1.02	2019/08/23	Kent	1. Update figures and adjust some contents 2. Split this user manual from original version

1. Introduction

FPPA™ (Field Programming Processor Array) integrated development system provides a system development environment for users to easily develop programs and debug system. This system includes an emulator (ICE), a Writer, and a set of integrated development software (IDE).

Integrated Circuit Emulator (ICE) is used to simulate the functions of PADAUK FPPA™ IC's. Its functions include execution, single step execution, break point, stop run and all the other basics, as well as more advanced functions like processor array content monitoring and memory data monitoring.

Writer is used to write the program file developed by the user into the corresponding IC.

This user manual only describes how to use the IDE software. The usages of ICE and Writer are available on their respective user manuals.

FPPA IDE integrated development software is a Windows-software (as shown in figure 1-1) that includes function modules such as editor, assembler, debugger, and program writer.



Fig. 1-1 FPPA IDE

2. Integrated development system software IDE

2.1 IDE installation

Follow below steps to install the IDE and the USB driver before connecting to the hardware appliance:

Step 1: Get the installer Setup_IDE_0.xx.exe from the website <http://www.padauk.com.tw>. After execution, you will see a message box pop-up as shown in figure 2-1.



Fig.2-1 FPPA™ IDE installation screen

Step 2: Complete the installation.

Note: Before you execute the Setup program, if your PC has both system administrator and visitor modes, you should log in the system administrator mode, and temporarily close the IDE program, otherwise the installation cannot be successful; If you have installed an IDE, you can install it directly . Here are some common mistakes.

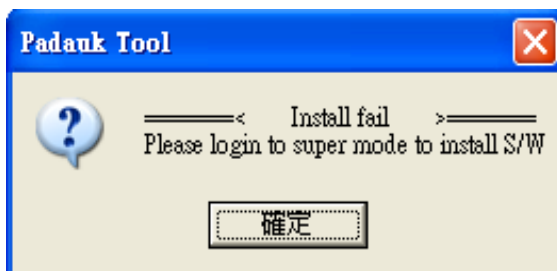


Fig. 2-2. Logging in the system administrator mode

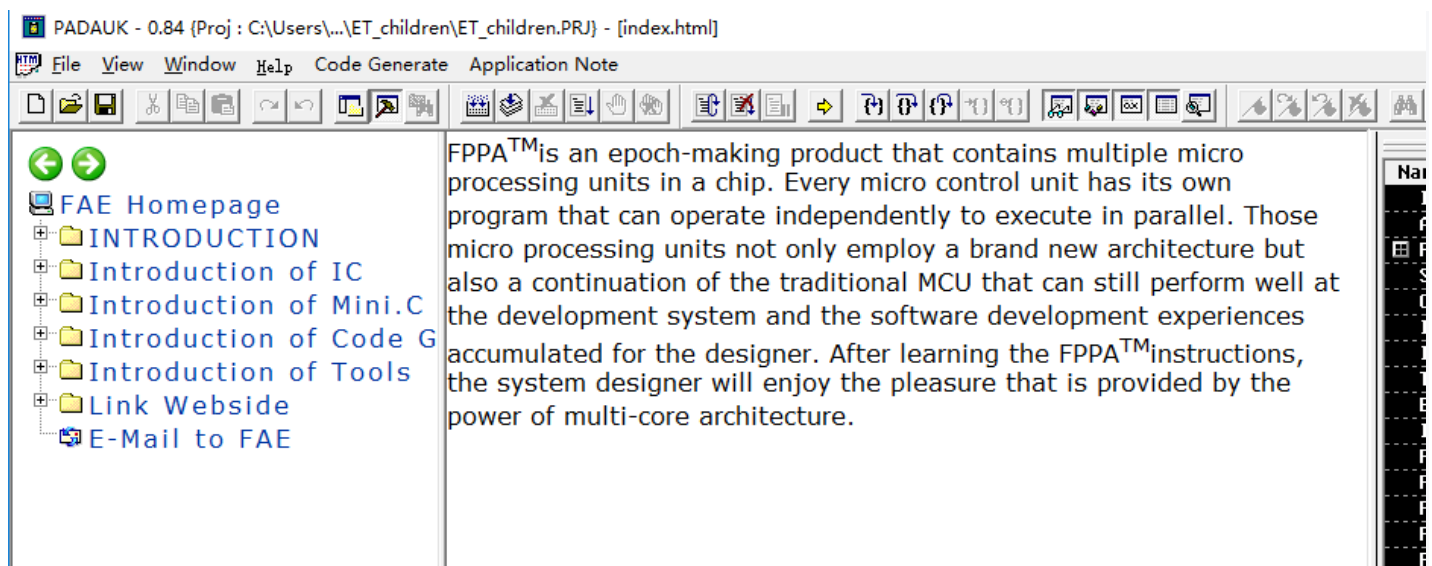


Fig. 2-3 Temporary closing of the IDE program

3. The usage of FPPA IDE Integrated development system

The FPPA IDE's functions include editor, assembler, debugger, and program writer. In the window environment, you can easily complete system programming, debugging and writing.

In case of technical problems in use, you can find solutions to relevant problems from **[Menu]->[Help (H)]->[Application note]** or **[Menu]->[Application note]**.



3.1 The usage of IDE

After correctly installing the software, there will be a "FPPA IDE" shortcut in desktop. As shown in figure 1-1.

Double-click to execute the program.

IDE development system does not provide soft simulation.

IDE programming languages have two types to choose from:

- (1) General ASM type, which is based on the assembly language of Padauk technology and can mix the syntax of C language.
- (2) Mini-C type, which is based on the C language of Padauk technology and supports the syntax of ASM language. It not only provides the convenience of C language but also has the simplicity and efficiency of ASM language.

3.2 Building a new ASM project

After opening the IDE, you can start a new project from menu "File -> New project", or open an old project from menu "File -> Open project".

Note: You must built a new project at the first time to execute the IDE.

Figure 3-1 shows the selection screen when starting a new project.

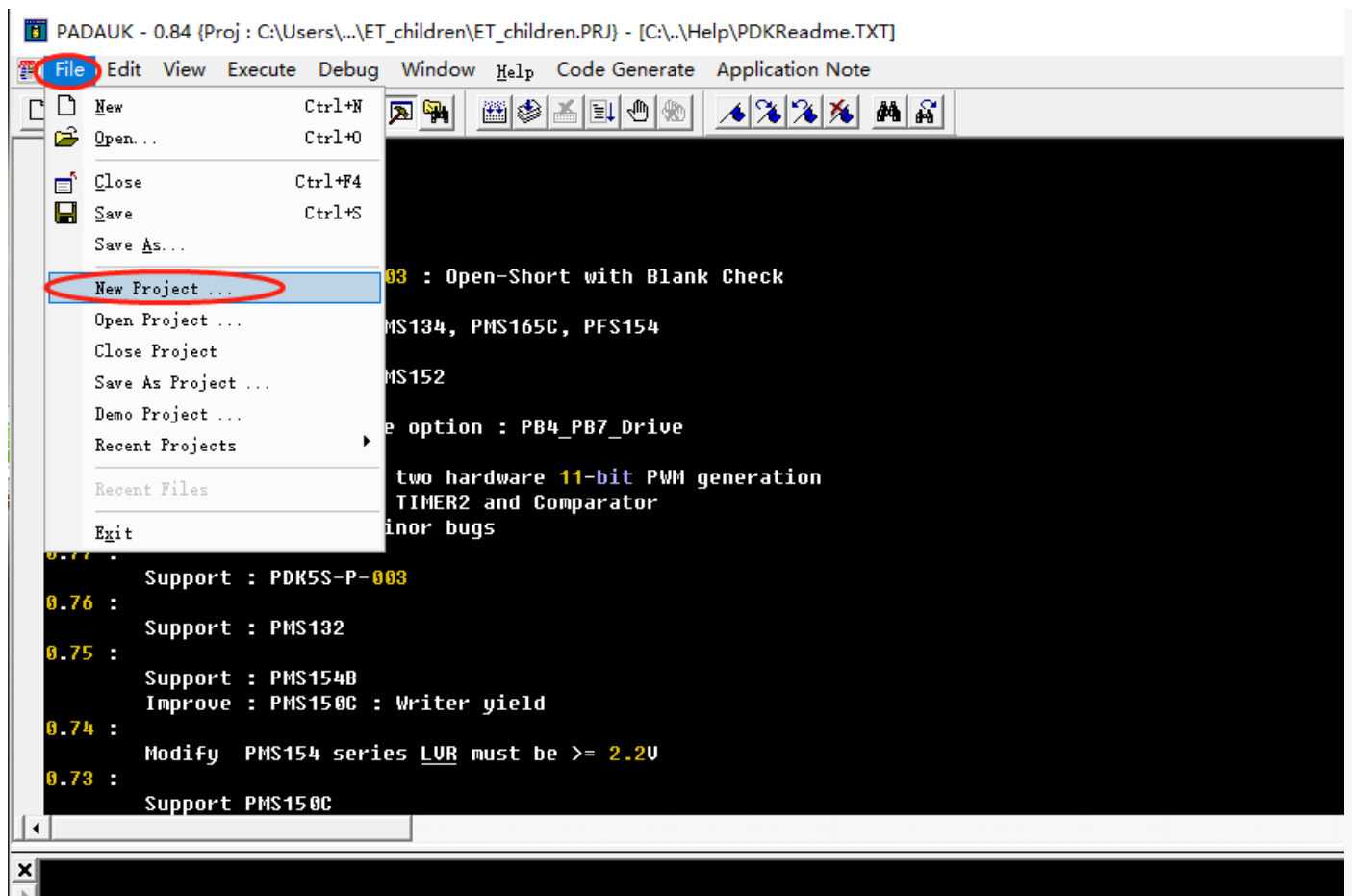


Fig. 3-1 Start a new project

After starting a new project, you will see the dialog box shown in figure 3-2 .

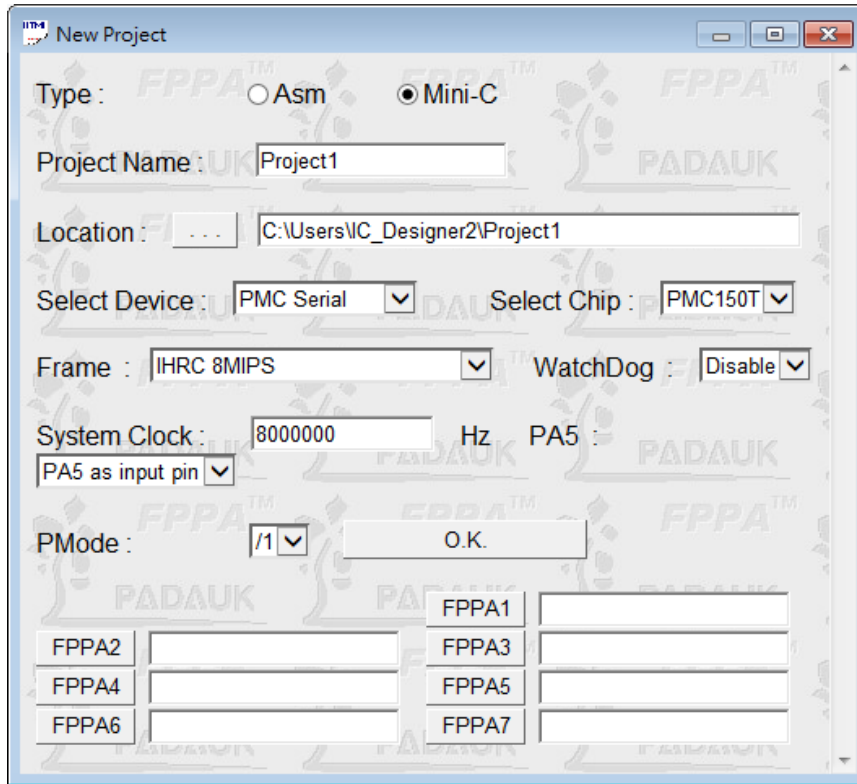


Fig. 3-2 Start a new project/Choose Mini-C type new project

There are two types to choose: ASM type and Mini-C type. You can click **ASM** and **Mini-C** to switch the type. Figure 3-3 shows the selection screen when choosing ASM type.

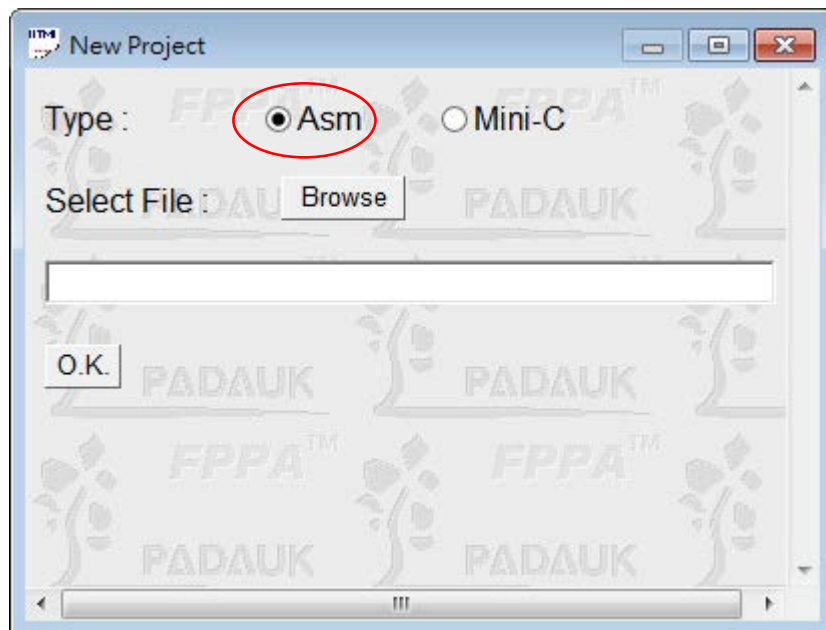


Fig.3-3 Choosing ASM type new project

3.3. Adding an old ASM file to the new project

If an old ASM file exists, select the **ASM** type and click **Browse** to select it, as shown in figure3-4.

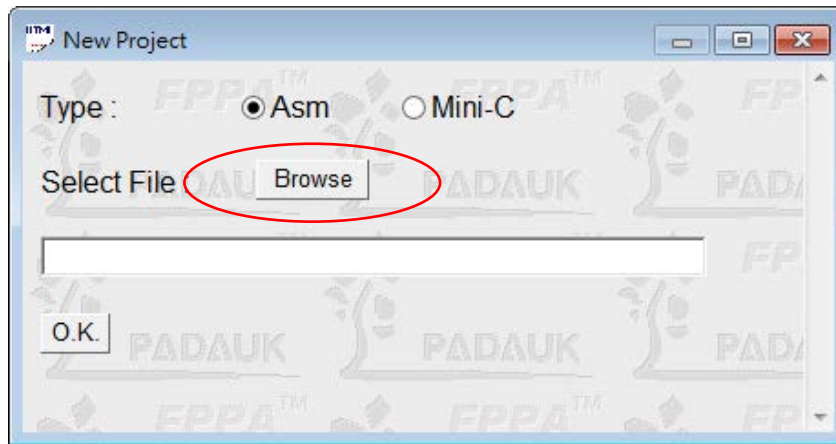


Fig.3-4 Adding an old ASM file to the new project

Then the file selection window appears, as shown in figure 3-5. Select your ASM file.

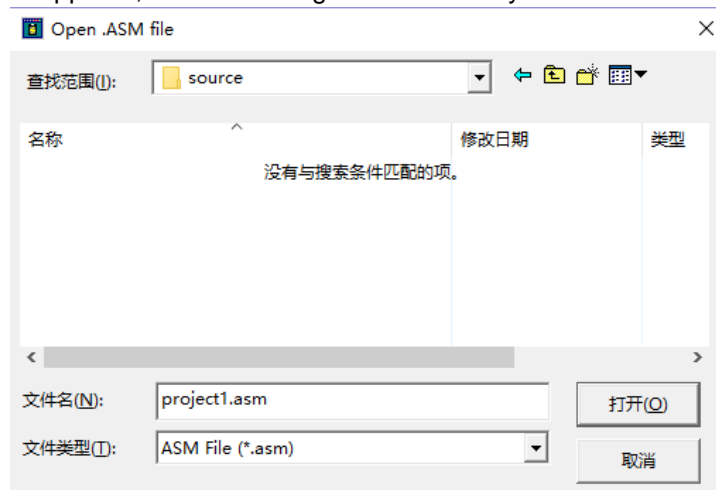


Fig.3-5. ASM source file selection

Once selected, click **OK**. As shown in figure 3-6.

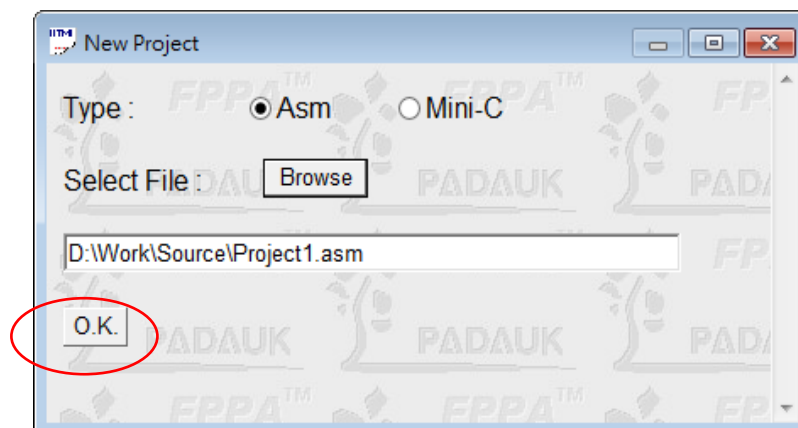


Fig.3-6. The window screen after file selection

If the file you selected does not exist, the system will ask whether to use a simple Demo Code for reference when writing. The window is shown in figure 3-7.

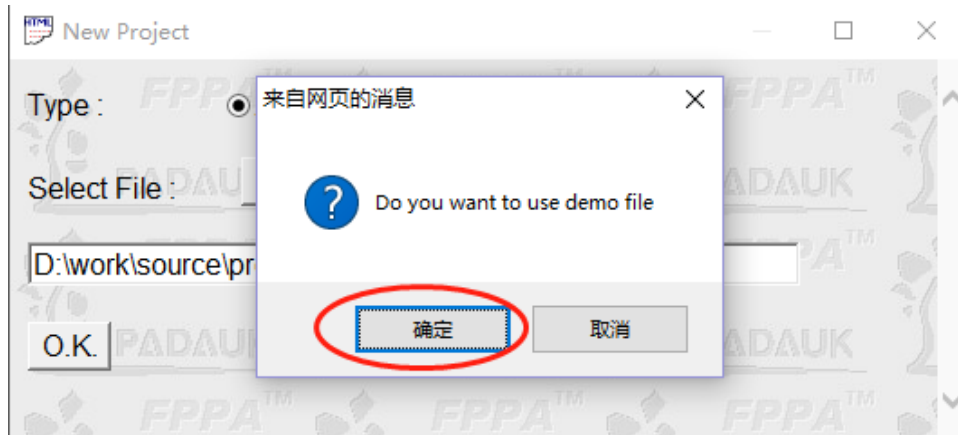


Fig.3-7. Demo code choosing window

Finally, a new project is generated. The window screen is shown in figure 3-8:

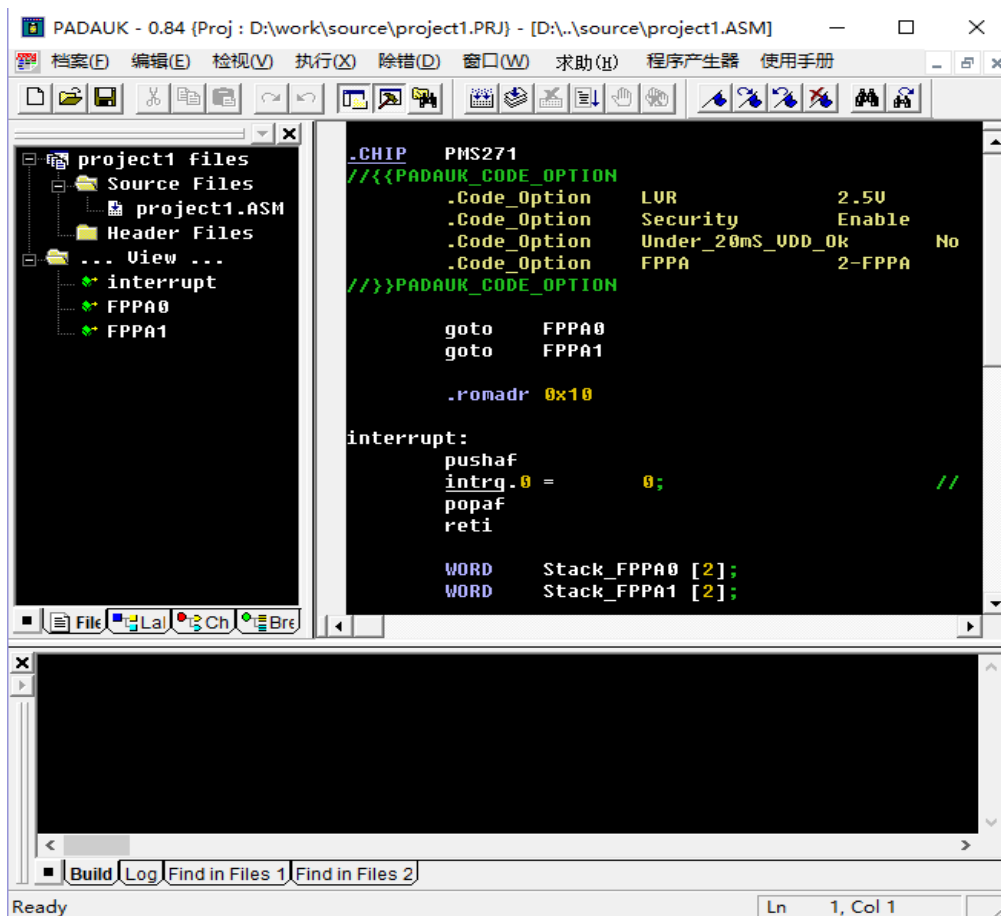



Fig.3-8. A new ASM project

3.4. Program execution and debugging

Once a program is finished, you can start debugging it. Click  to start building the program. As shown in figure 3-9:

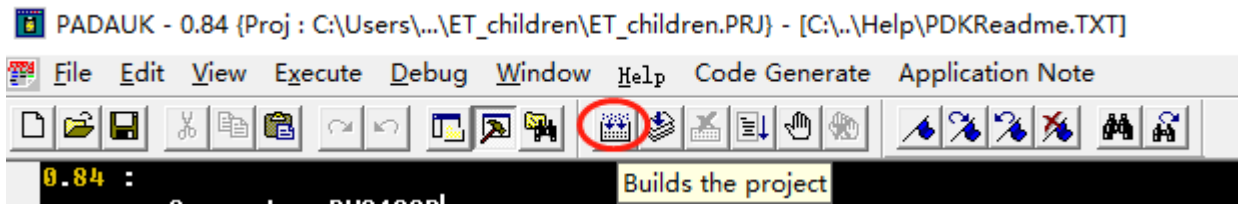


Fig.3-9. Building a program

Program option **Code_Option** is used to select or modify system settings, which can be opened from **Execute** ->**Code Options**, as shown in figure 3-10:

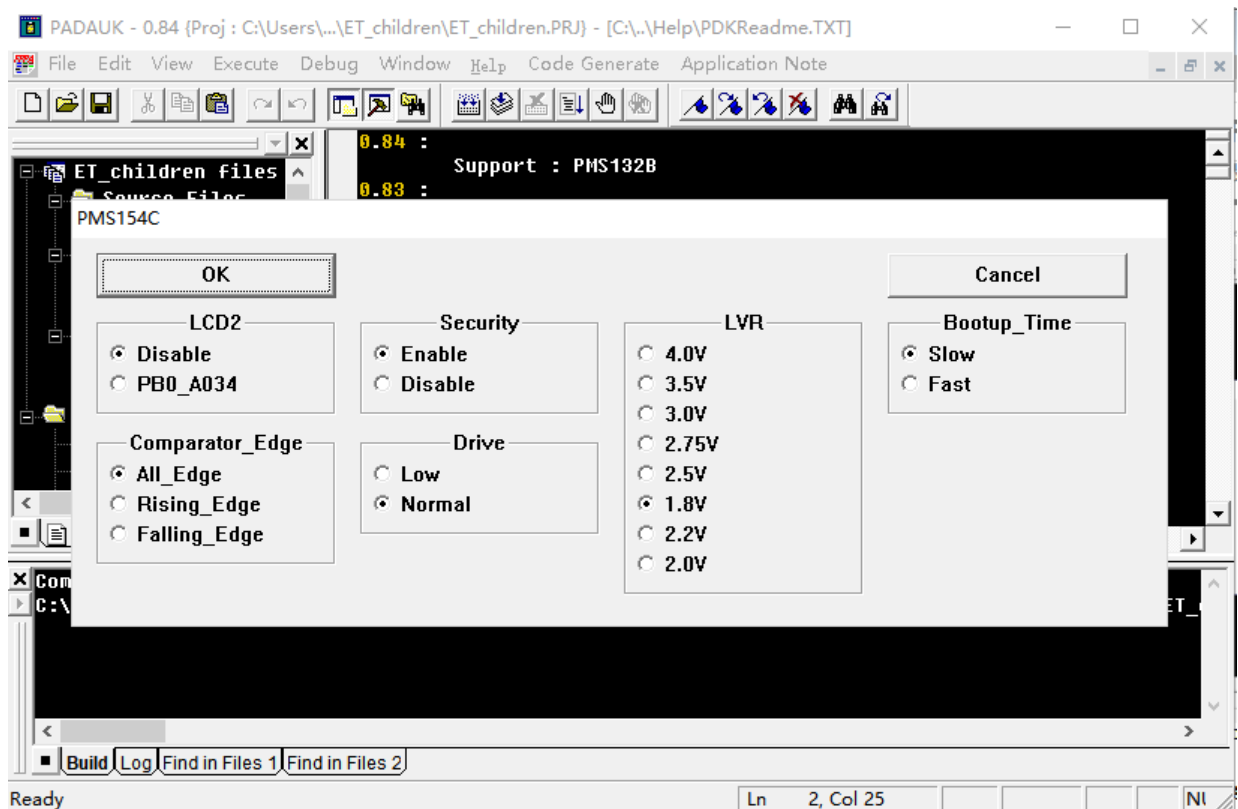



Fig.3-10 Code_Option

Once the program is built, you can start debugging. You can use the single-step key F11 to step through the program, or you can click  in the window to keep the program running. As shown in figure 3-11.

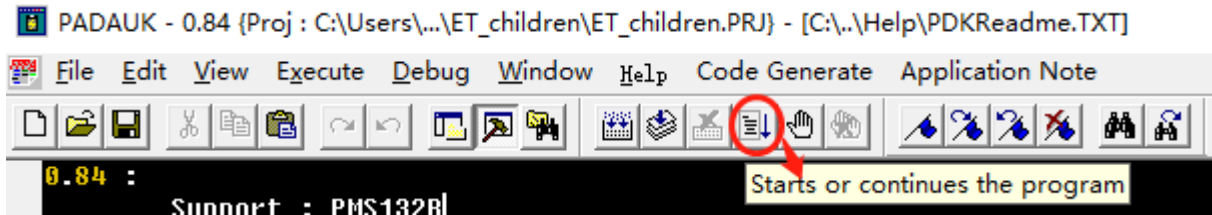


Fig. 3-11. Continuing the program

Note: Before starting debugging, the PC should be connected to ICE for simulation; otherwise, the system will prompt that ICE is not connected, as shown in figure 3-12:

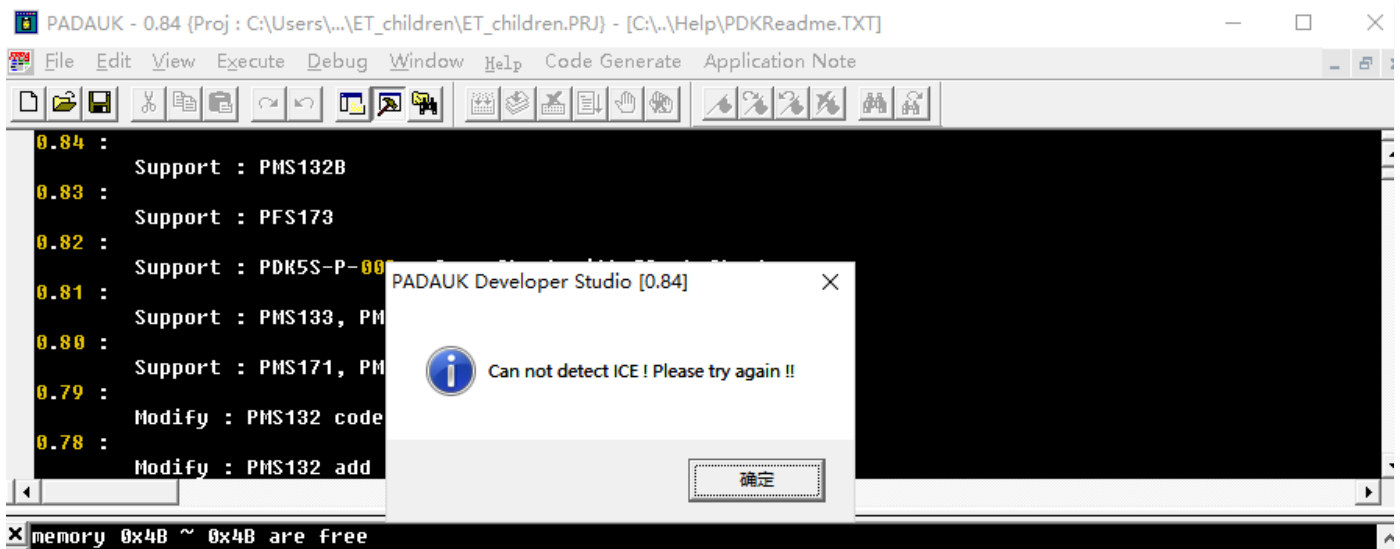


Fig. 3-12 ICE is not connected

After connecting to ICE, you can use the single-step key F11 to go into debug mode and start executing the program. The window screen is shown in figure 3-13.

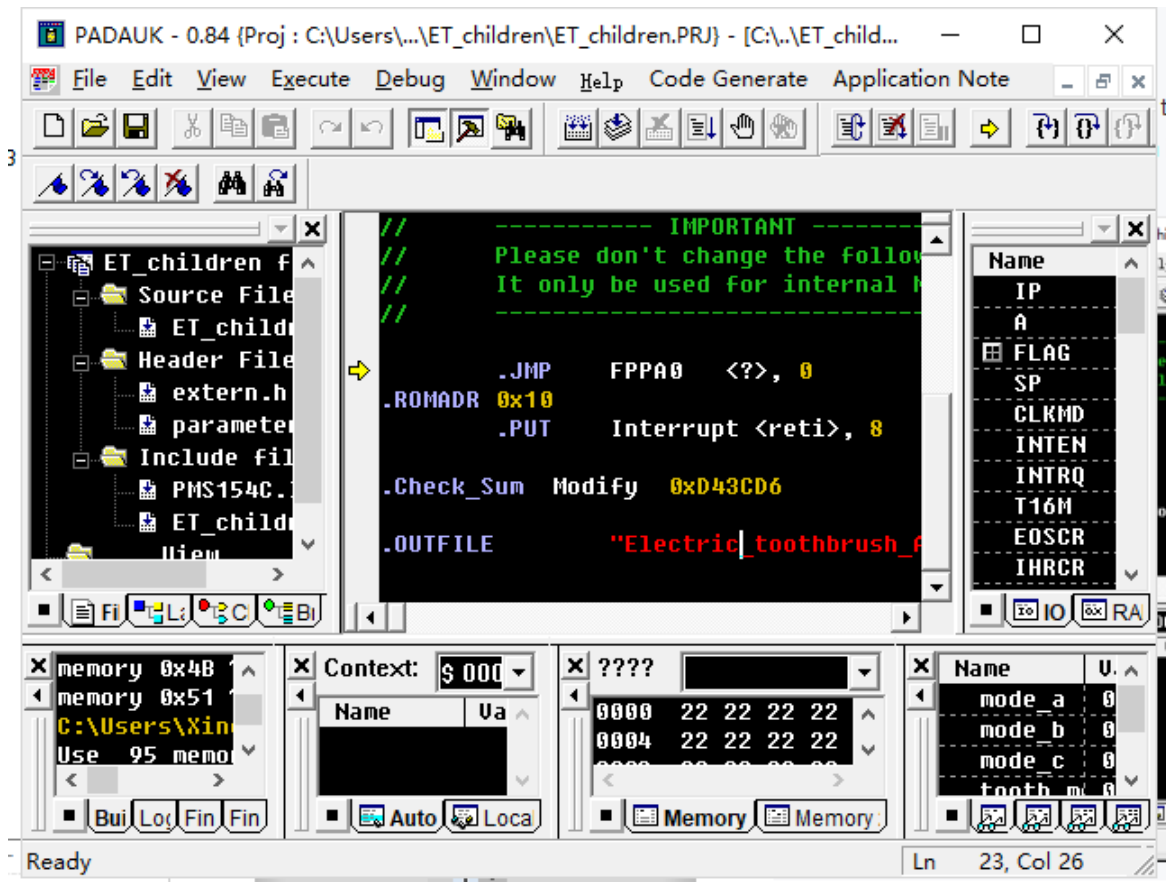


Fig. 3-13 In debug mode

3.5. Building a new Mini-C project

The steps to build a Mini-C type project are as follows:

1. Build a new project from the menu. As shown in figure 3-14.

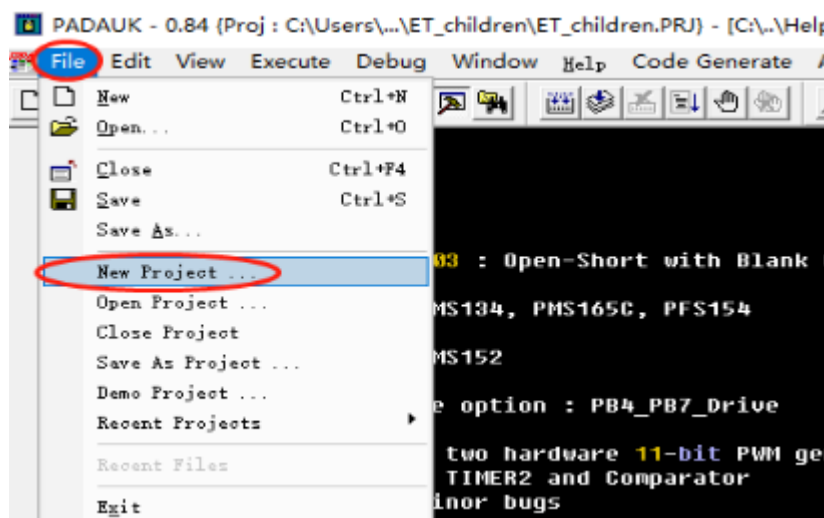


Fig. 3-14 Build a new project

2. The dialog box shown in figure 3-15 will appear. Select Mini-C type.



Fig. 3-15 Selecting language type

3. Then the dialog box shown in figure 3-16 will appear. Type in the project name (default is Project1), select the project location, select IC and parameters (you can use default values) and then click **OK**.

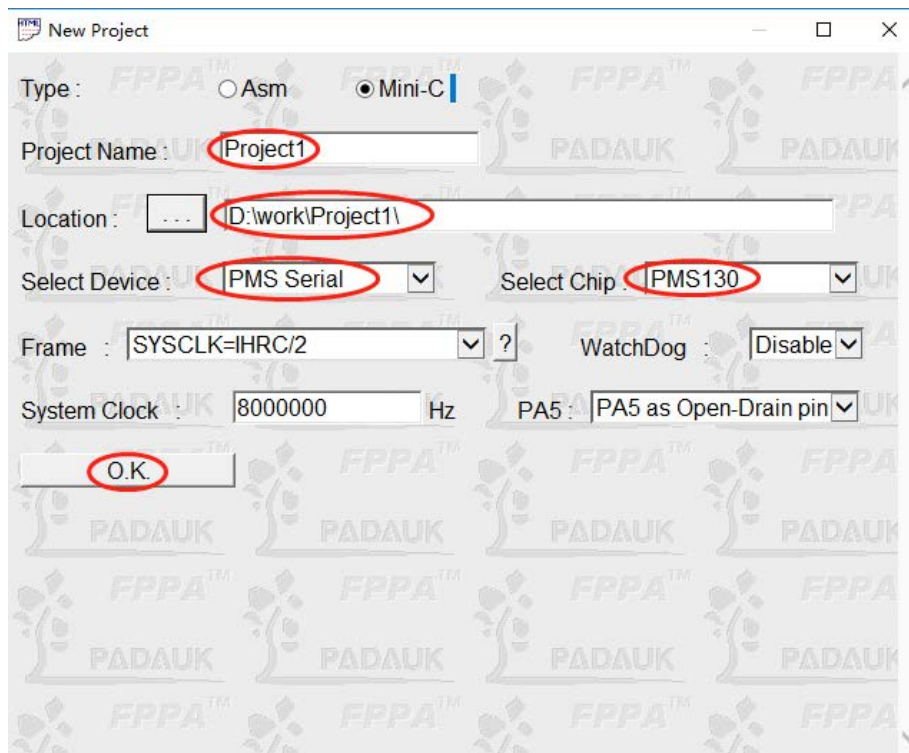


Fig. 3-16 Mini-C parameter selection window

4. After the establishment of the Mini-C project, the window screen of the IDE is shown in figure 3-17.

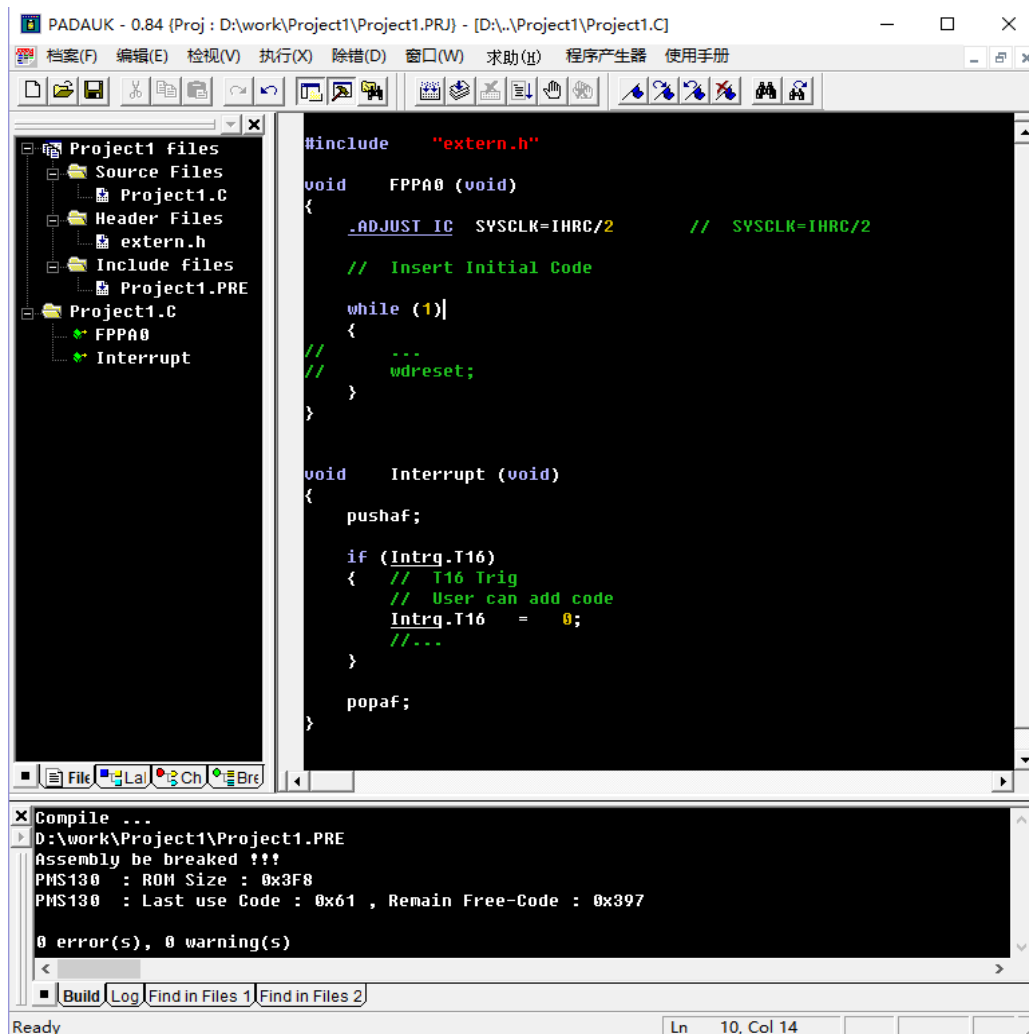



Fig. 3-17 The window screen after the establishment of Mini-C project

5. Click  or press the single-step key F11 to open the **Code_Option** and set program parameters, as shown in figure 3-18.

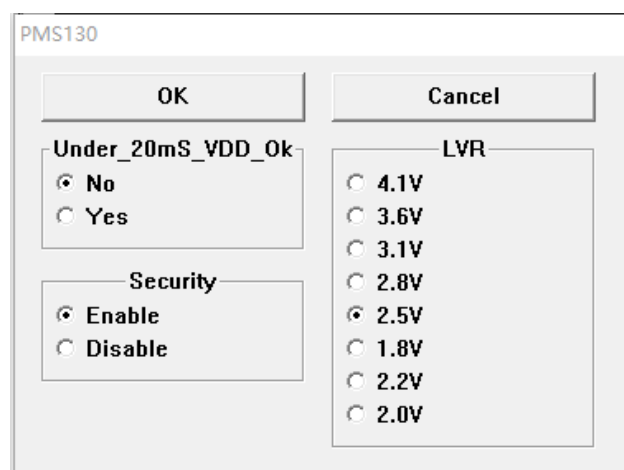


Fig.3-18 Code_Option window

The voltage value of LVR should be coordinated with the system frequency (the default frequency is 8MHz) and IC working voltage to make the system work stably. If you do not select the appropriate voltage, the IDE will prompt error. As shown in figure 3-19:

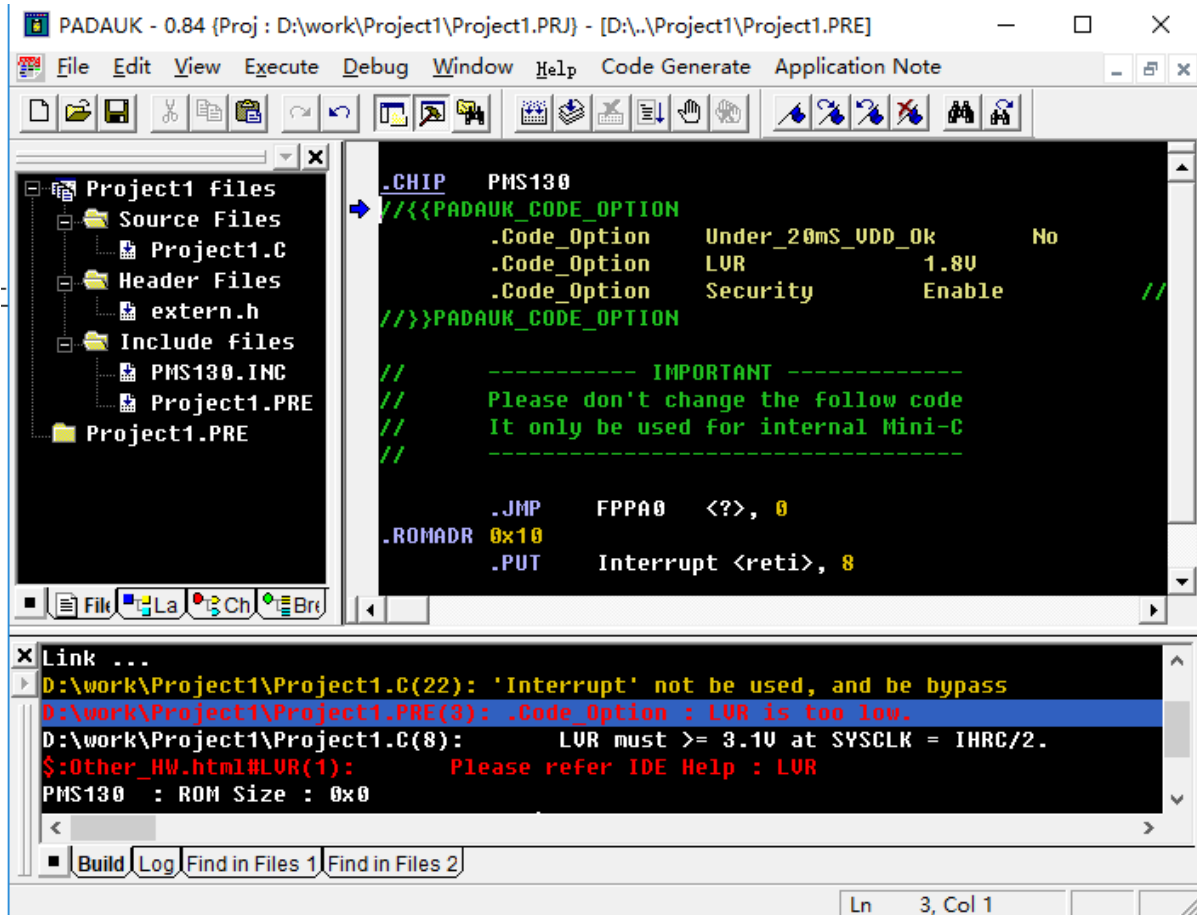


Fig. 3-19 Adjusting LVR

If this occurs, you can resolve the error by reducing the system frequency (the default is 8MHz) or increasing the LVR value.

In the .C file, you can delete or comment out ".ADJUST_IC SYSCLK=IHRC/2 "(as shown in Fig. 3-20) and click **Build** or **Rebuild All** to calibrate IC and select system frequency(as shown in Fig. 3-21).

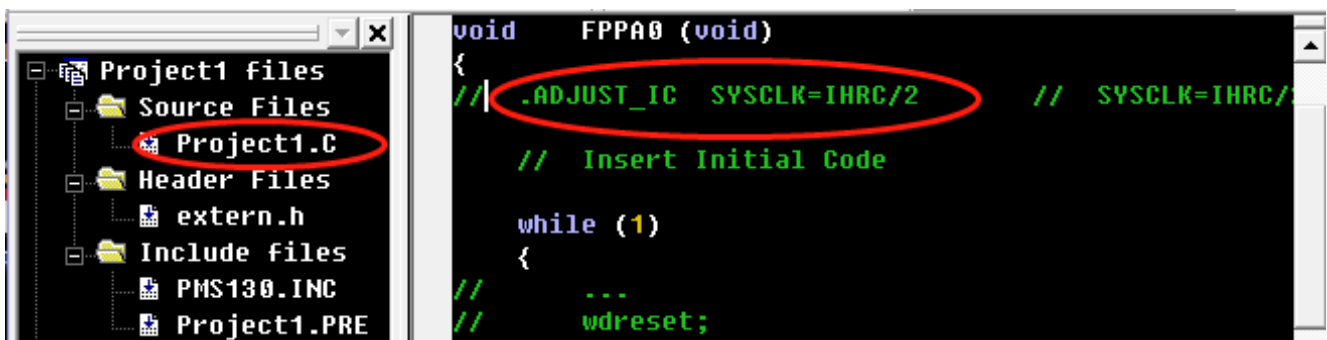


Fig.3-20 System frequency calibration statement

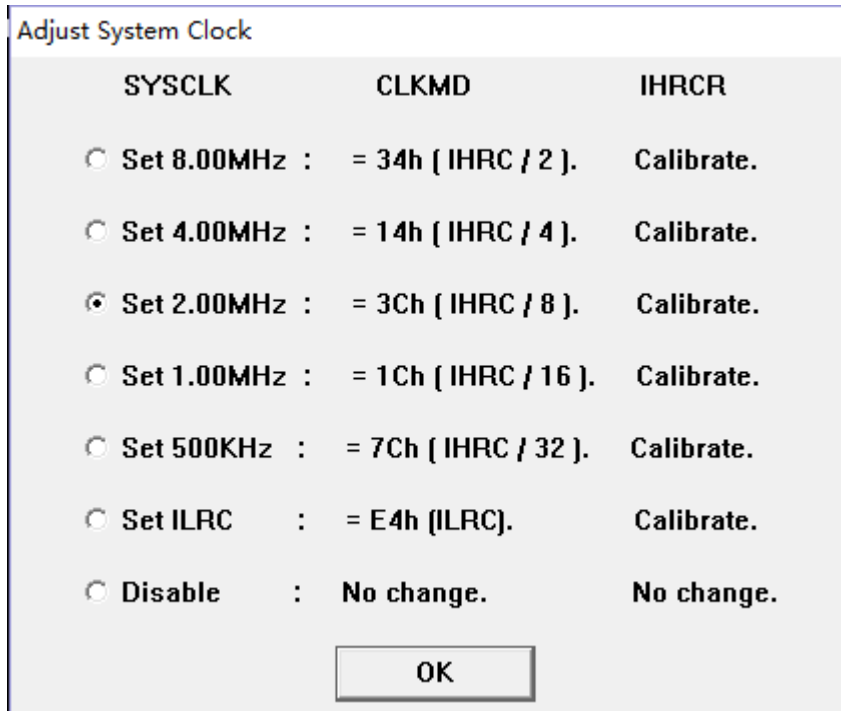


Fig.3-21 System frequency selection, IHRC calibration options

IDE can calibrate band-gap when building some IC (such as PMS130). Therefore, after you select the system frequency, you can see a pop-up dialog box. As shown in figure 3-22.

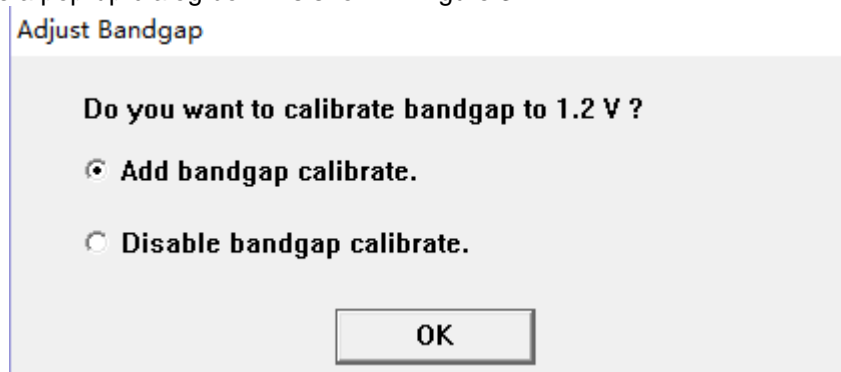


Fig. 3-22 Band-gap calibration option

After the selection, you can see the system frequency, working voltage and band-gap calibration of the IC in the program, as shown in figure 3-23:

```
.ADJUST IC SYSCLK=IHRC/8, IHRC=16MHz, UDD=5U, Bandgap=0n;
// WatchDog Disable, RAM 0 ~ 0xF temporary be used
// You can add the follow code :
//     CLKMD.En_WatchDog = 1; // WatchDog Enable
```

Fig.3-23 Band-gap calibration option

The working voltage VDD refers to the working voltage during ICE simulation. You can adjust the working voltage based on the system frequency and LVR setting to make the system work stably.

Note: PMS130 is selected for this example. In figure 3-19, the command ".CHIP XXXXXX "is used to select the IC model. You can use this instruction to change the IC model.

After setting the program parameters, you can write your program functions. Click **Build (B)** or **Rebuild** to enter the debugging environment and start running the program. The window screen is shown in figure 3-24:

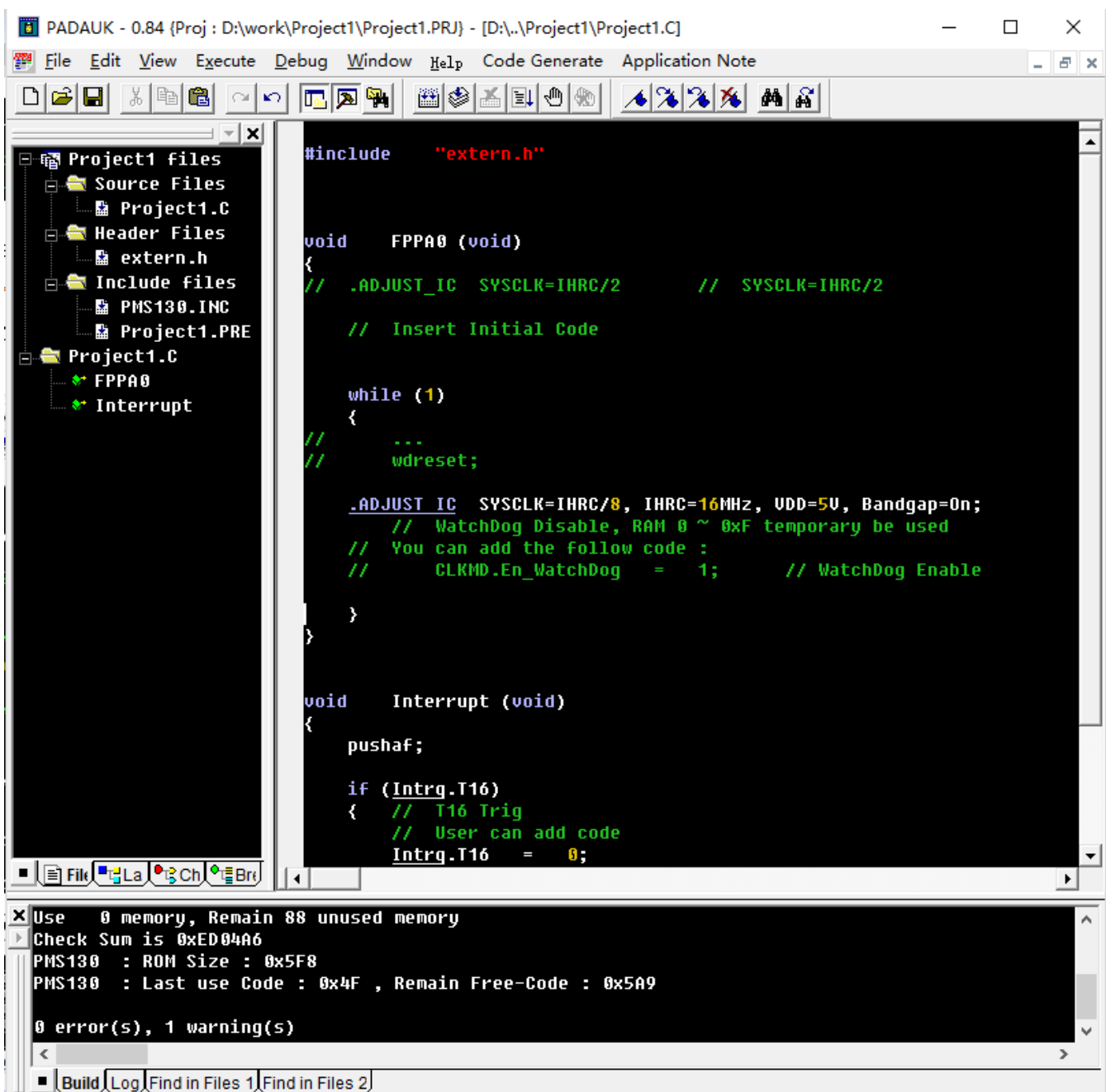


Fig.3-24 Mini-C type program debug window screen

4. Introduction of FPPA™ IDE's environment and window

4.1. Mini-C's assembly environment

The options for executing Mini-C program are shown in figure 4-1.

You can click **Build(F7)** in the menu to assemble program code. Besides, you can start debugging and assembling by clicking **Single (F11)**, **Step over (F10)**, **Go (F5)**, and so on.

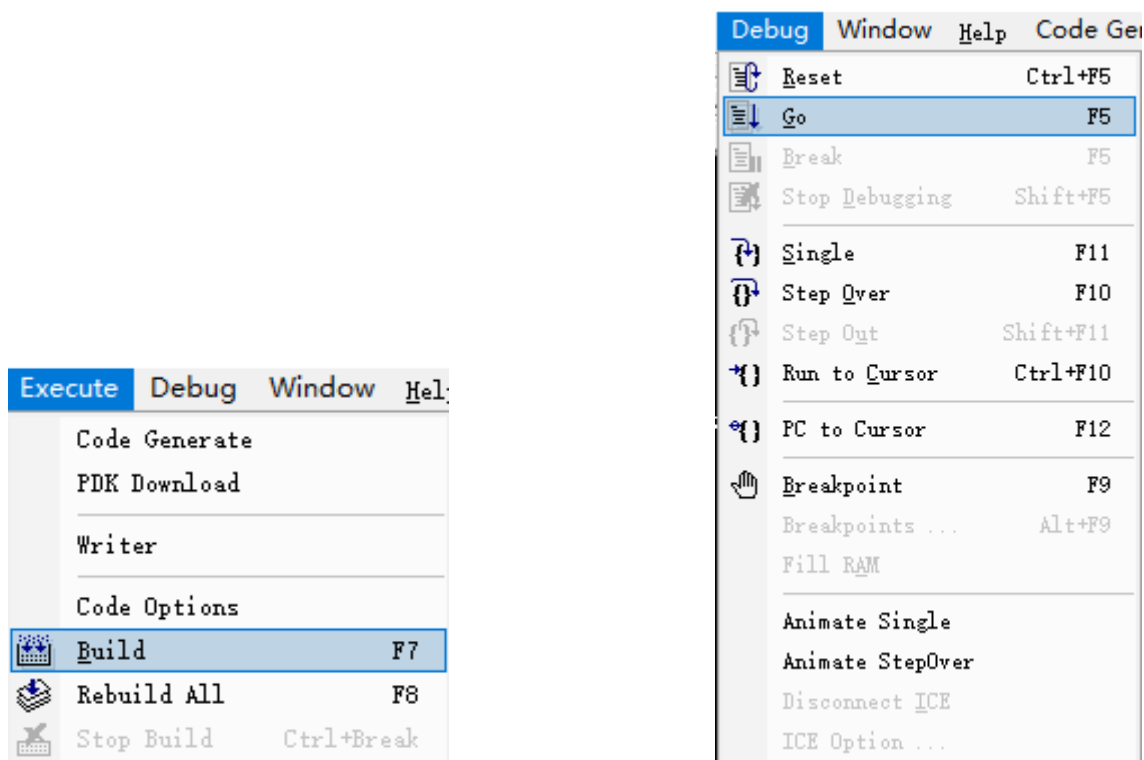
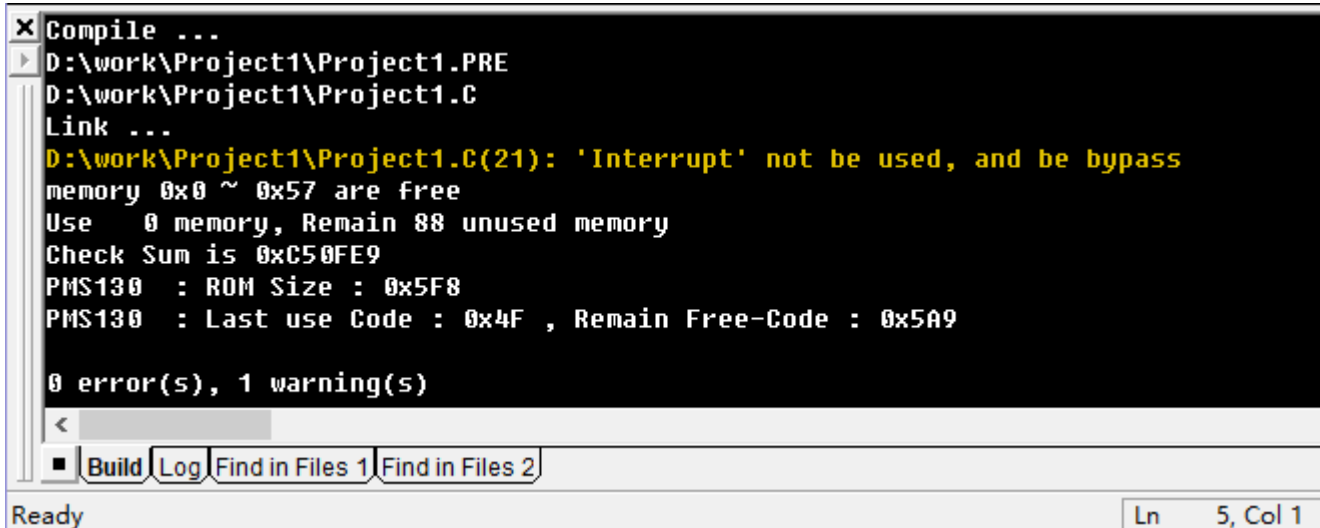


Fig.4-1 Mini-C program execution options

- ◆ **Code Options** in the **Execute** drop-down menu: It enables you to change some program parameters, such as Security, LVR, etc., after which these parameters will be automatically pasted in the program code.
- ◆ **Rebuild All(F8)** in the **Execute** drop-down menu: Reassemble all program code.
- ◆ **Build(F7)** in the **Execute** drop-down menu: To speed up program execution, only assemble the changed parts of the program. The OBJ files created during assembly will exist in the OBJ subdirectory of the project directory. The OBJ subdirectory is automatically created the first time you assemble the program.
- ◆ **Writer** in the **Execute** drop-down menu: Write IC.
- ◆ **Code Generate** in the **Execute** drop-down menu: Generate the needed sample code. The generated code will be pasted to the clipboard, and you can use "Ctrl + V" to put it in the specified place.

4.2. The assembly information window of FPPA™ IDE

After assembling the program, you can see the Mini-C program assembly information at the bottom of the window. As shown in figure 4-2.



```

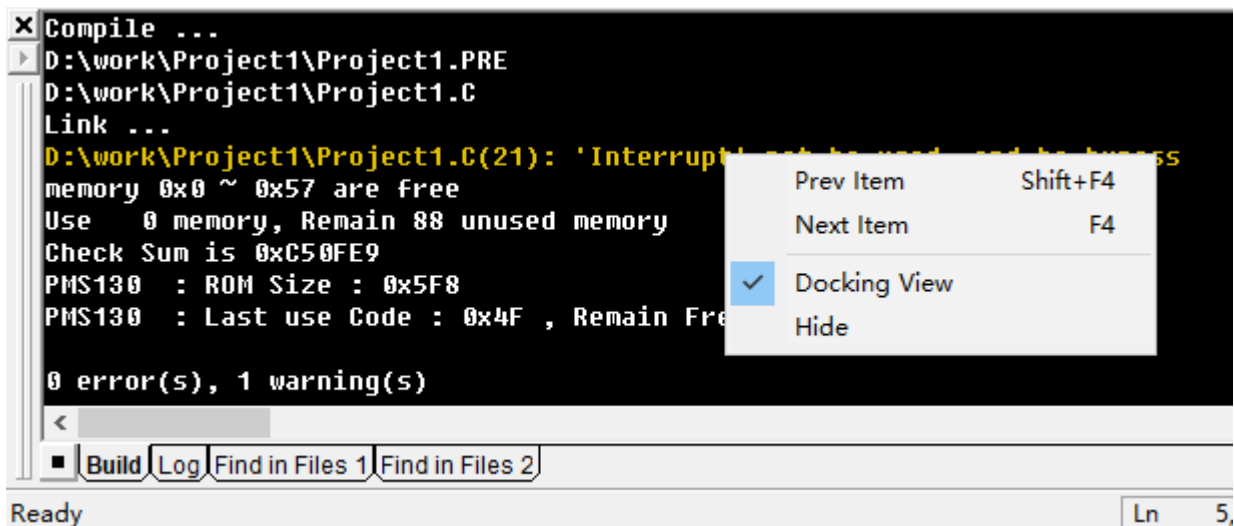
x Compile ...
D:\work\Project1\Project1.PRE
D:\work\Project1\Project1.C
Link ...
D:\work\Project1\Project1.C(21): 'Interrupt' not be used, and be bypass
memory 0x0 ~ 0x57 are free
Use 0 memory, Remain 88 unused memory
Check Sum is 0xC50FE9
PMS130 : ROM Size : 0x5F8
PMS130 : Last use Code : 0x4F , Remain Free-Code : 0x5A9

0 error(s), 1 warning(s)

```

Fig.4-2. The assembly information window of Mini-C program

- ◆ If there is no red error message in the window, that means the program is well assembled.
- ◆ A Warning message in yellow is usually an unused variable or code detected by development tool to alert the user.
- ◆ Double-click the Error/Warning message in the window to specify the corresponding statement location.
- ◆ Right click the window to open a function list. By using the list, you can quickly reach the previous or subsequent error, as shown in figure 4-3.



```

x Compile ...
D:\work\Project1\Project1.PRE
D:\work\Project1\Project1.C
Link ...
D:\work\Project1\Project1.C(21): 'Interrupt' not be used, and be bypass
memory 0x0 ~ 0x57 are free
Use 0 memory, Remain 88 unused memory
Check Sum is 0xC50FE9
PMS130 : ROM Size : 0x5F8
PMS130 : Last use Code : 0x4F , Remain Free-Code : 0x5A9

0 error(s), 1 warning(s)

```

Fig.4-3. Quick debugging method based on Mini-C assembling messages

4.3. Development tool's workspace management and debugging environment

In the left workspace of FPPA™ IDE, you can see contents shown in figure 4-4.

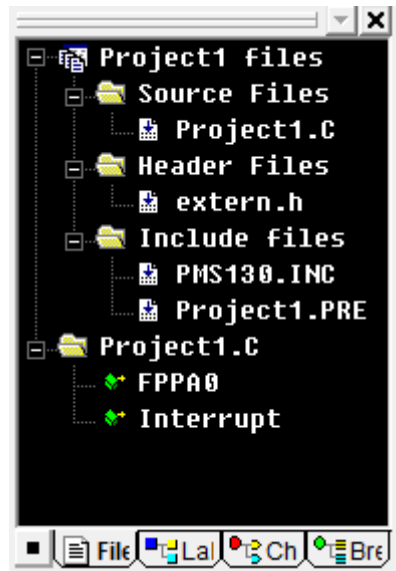


Fig.4-4. Workspace management window

You can use the following four functions by switching windows in the workspace:

- (A).FileView: Show files that the current project consists of.
- (B).LabView: Show labels that are defined by the current open file.
- (C).ChipView: Show the particular information of current FPPA.
- (D).BreakView: Enable or disable various Breaks of ICE.

You can double click a file for opening , or right click the file and click Open.

Example:	Double-click + file	->	To open a file
	Double-click + Label	->	The cursor moves to the Label definition

4.3.1. FileView

- (1) Changing the preposing file of IC setting

Right click the Source Files or the Project File, and then select **Change .PRE to Project** in the open menu to change the current preposing file, as shown in figure 4-5.

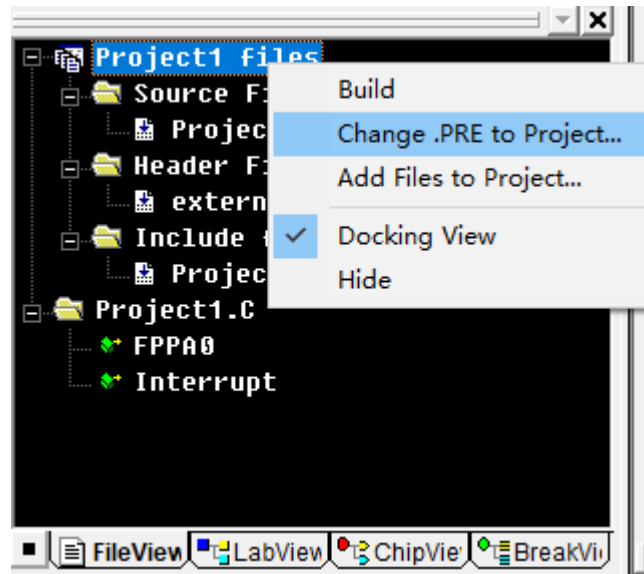


Fig.4-5. Changing the preposing file of IC setting

(2) Linking the file

Right click the **Source Files** or the **Project File**, then select **Add Files to Project** in the open menu to link the file that you want, as shown in figure 4-6.

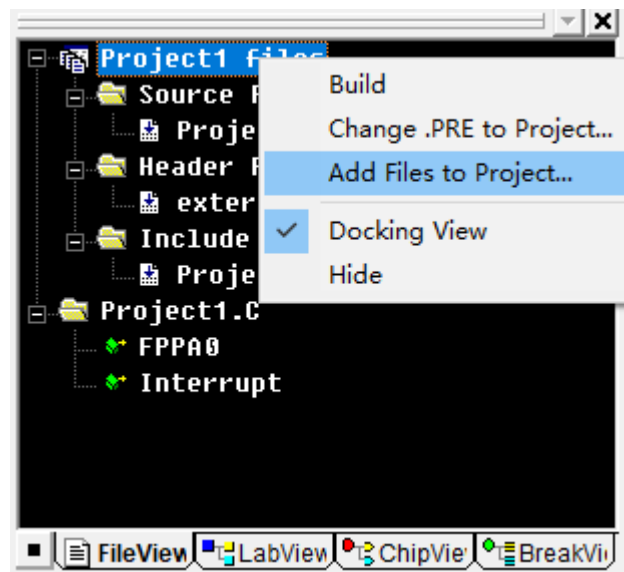


Fig.4-6. Linking the file

(3) Removing the linking file

Right click the **.C** file under the Source Files, and then select **Delete** in the open menu to remove the unneeded file from the project, as shown in figure 4-7.

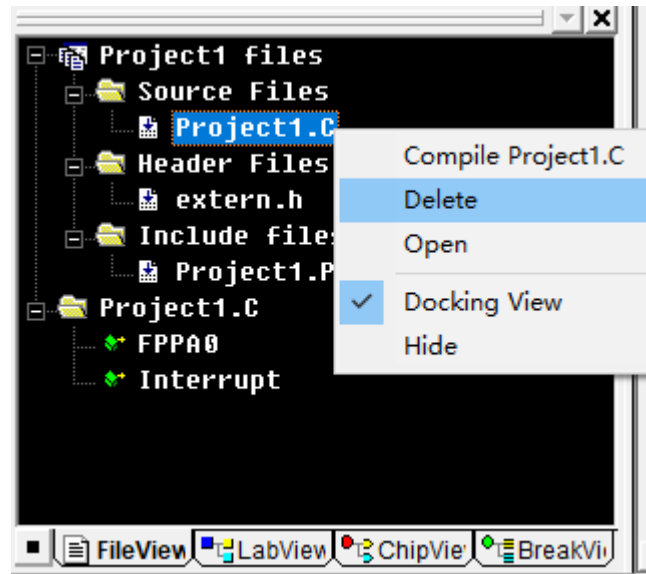


Fig.4-7. Removing the linking file

(4) Reference files of Head File

Right click the Head Files, then select **Add Files to Folder** in the open menu to link the needed file. However, this function is for quick reference only and will not affect the assembly results.

(5) In the ASM type project, only one master file is allowed in the Source Files project; In the Mini-C type project, the link files can be assembled separately (" Compiler XXX.C ").

4.3.2. LabView

(1) There is no more information in this window when the assembly is not okay. As shown in figure4-8.



Fig.4-8. The information in LabView when the assembly is not okay

(2) After the assembly is okay, the information of this window is shown in figure 4-9:

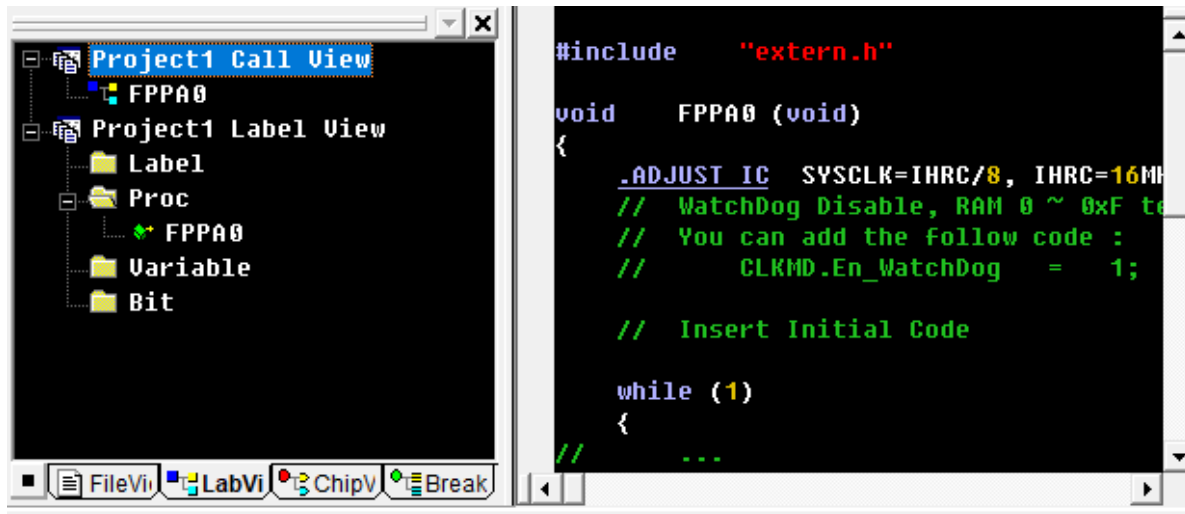


Fig.4-9. The information in LabView when the assembly is okay

LabelView project stores a list of known symbols, which can be divided into four categories: Label/Proc/Variable/Bit, described as follows:

- Label: In the ASM project, any statement whose syntax is "XXX:" .
- Proc: In the Mini-C project, any statement whose syntax is "void XXX (void) {} " .
- Variable: Any variable whose syntax is "BYTE /WORD/EWORD/DWORD XXX" .
- Bit: Any parameter whose syntax is 'Bit XXX [: yyy.n]' .

In the example shown in figure 4-9, you can double-click an item to quickly switch to the target location.

4.3.3. Debugging environment of development tool

Click **Reset**(Ctrl+F5) , **Go** (F5), **Single** (F11), **Step Over**(F10), **Run to Cursor** (Ctrl+F10), **PC to Cursor**(F12), **Animate Single** and **Animate StepOver** and any other operations of the **Debug** menu to enable the system to enter the debugging environment. The function selection is shown in figure 4-10.

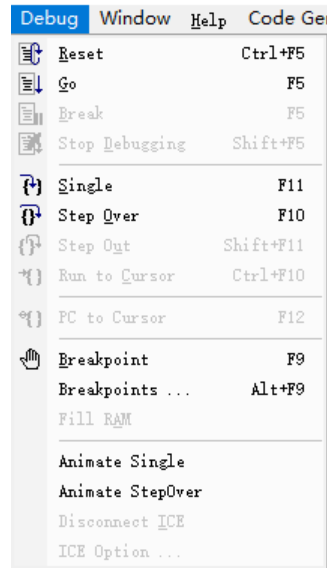


Fig.4-10. Debugging function selection

When you enter the debug environment, the window you see is shown in figure 4-11.

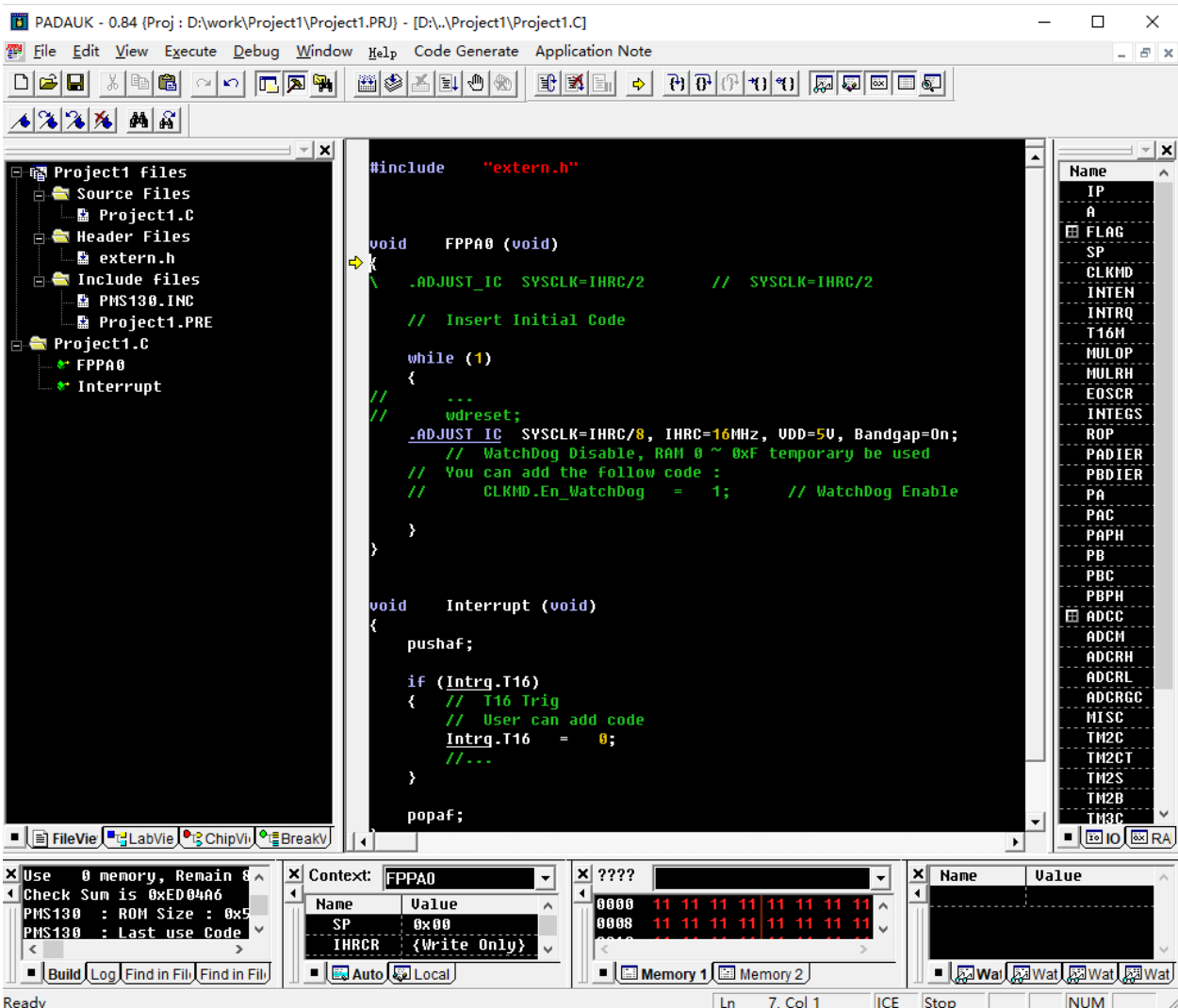


Fig.4-11. Debug window

- (1) **The upper right register window:** Record the information of each relevant register of IC.
- (2) **The bottom right Watch window:** Display the variable input by the user.
- (3) **The lower middle and right Memory window:** Display RAM and LCD data in Hex format.
- (4) **The lower middle and left Variable window:** Record three kinds of information:
 - 1) Context: Stack information of the current program.
 - 2) Auto: Display the Variables that may be used in the next execution.
 - 3) Local: Display the local variables of a program

(5) The Chip View window of upper left Workspace:

The Chip View window records FPPA's A, FLAG, SP, and so on. As shown in figure 4-12, users can quickly view and modify the values of A, FLAG and SP of FPPA in this window.

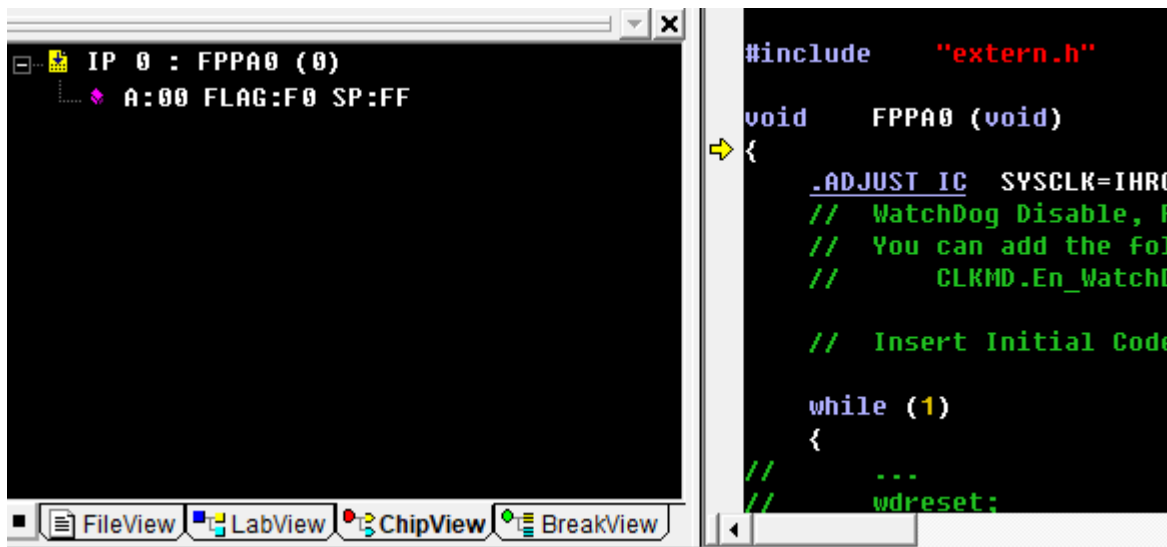


Fig.4-12. ChipView window information

For multi-core series IC, the Chip View window displays the information of each FPPA. And there are two modes (Mult. Traps and Single Trap) for you to choose. The default mode of IDE system is Single Trap mode, as shown in figure 4-13.

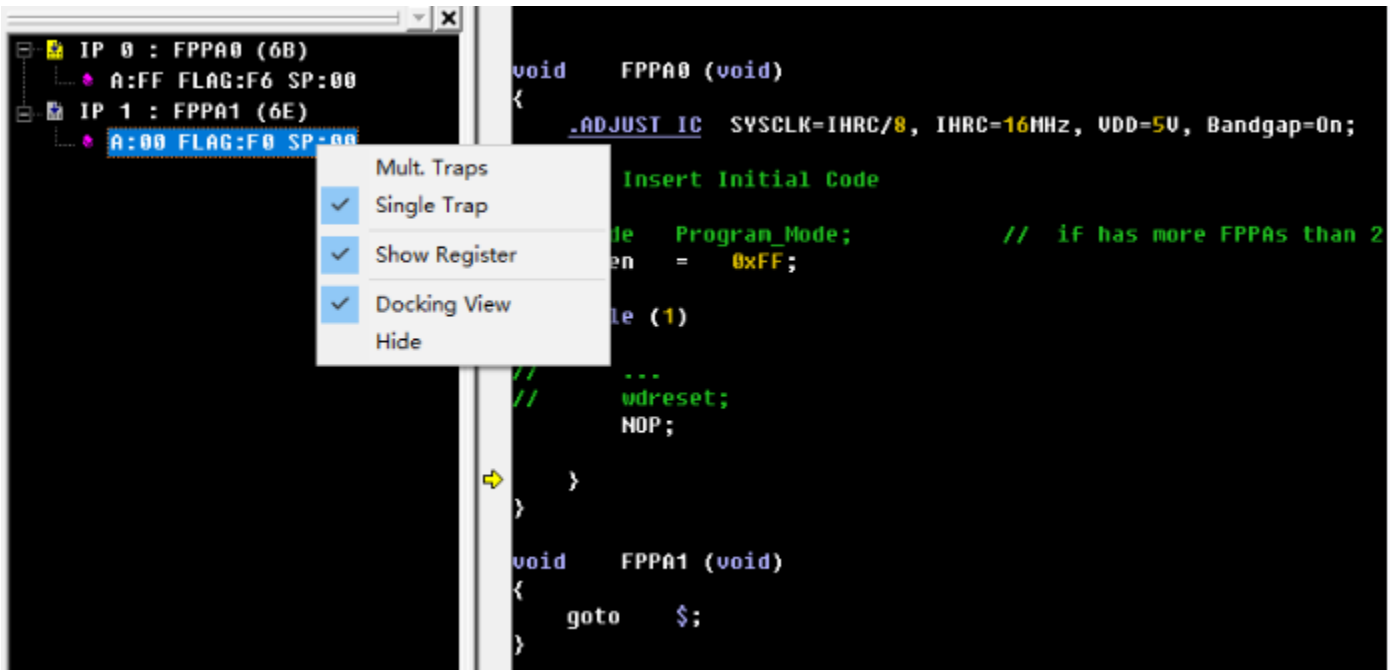


Fig.4-13. Multi-core IC ChipView window information

- In Single Trap mode, you can only monitor one FPPA at a time. If you want to switch the FPPA to the other, just double click it, as shown in figure 4-14.

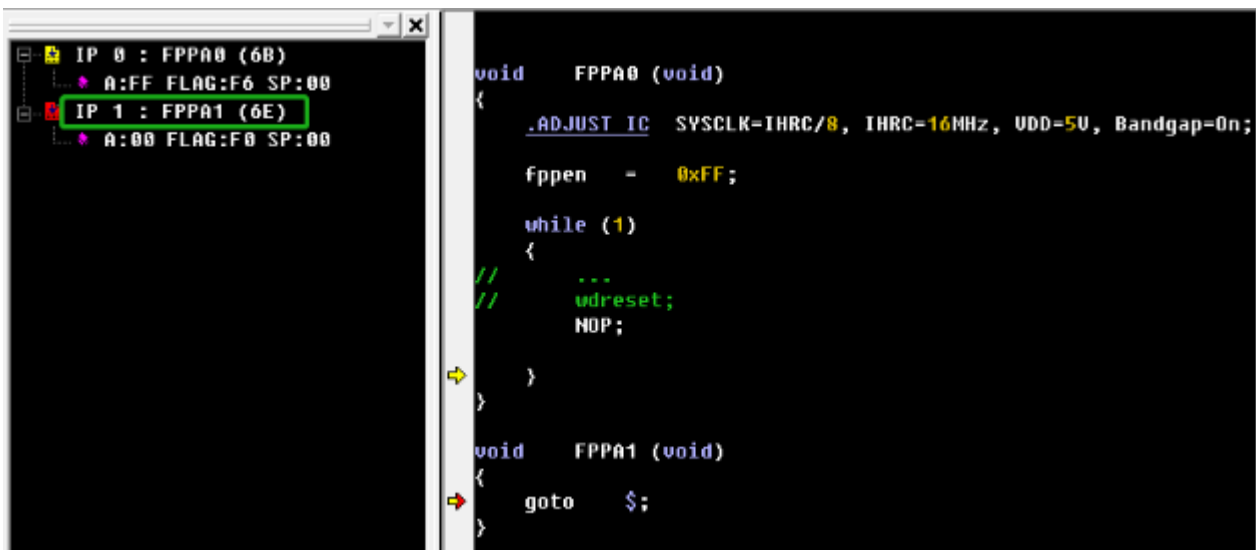




Fig.4-14. The information of switching FPPA in Single Trap mode

The explanation of figure 4-14: IP 0 is the FPPA that be executed (the position where  stopped in), IP 1 is the other FPPA that will be executed next time (the position where  stopped in). Click **Single**(F11) based on the current state to monitor IP 1. As shown in figure 4-15.

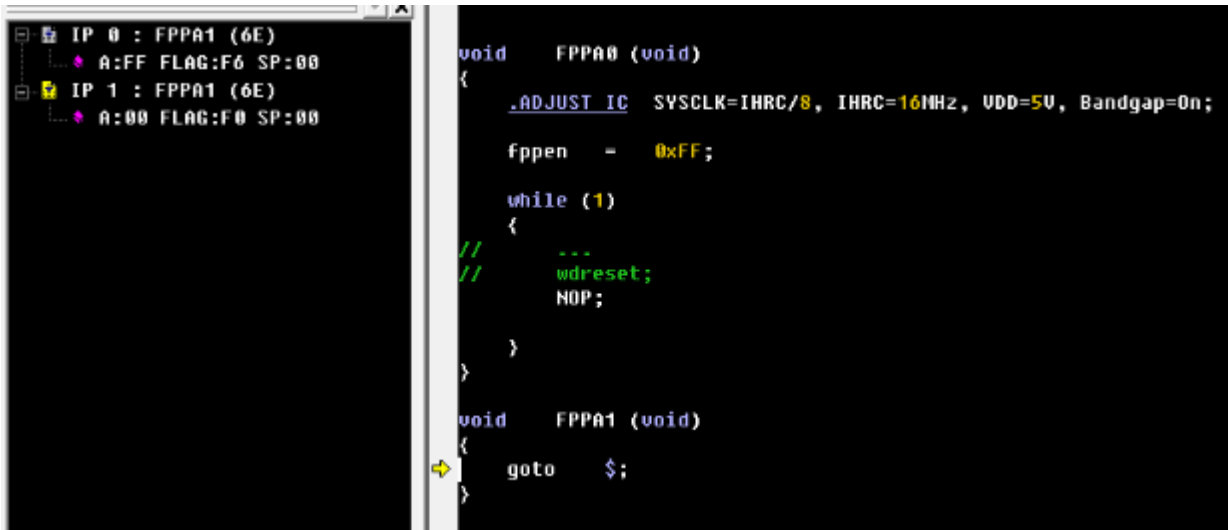



Fig.4-15. Information after switching FPPA

- In the Mult Traps mode, you can monitor two FPPA at the same time. The Mult Traps window is shown in figure 4-16.



Fig.4-16. Mult. Traps window

In the mult. Traps mode, you can click  to enable Trap to monitor each FPPA at the same time. Of course, you can also modify the related contents of FPPA, including A, FLAG, SP. As shown in figure 4-17

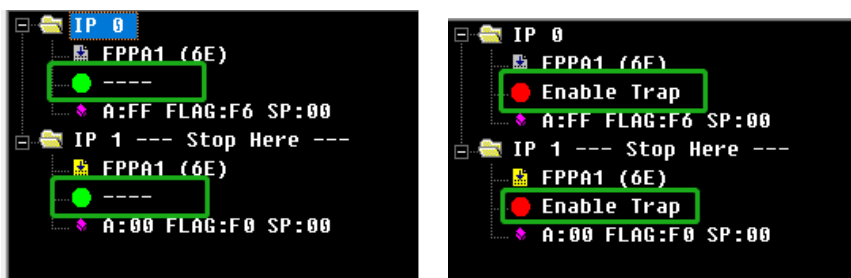


Fig.4-17. ChipView window information

(6) The Break View window of upper left Workspace:

1) In the Break View window, you can enable or disable various Breaks of ICE. As shown in figure 4-18.

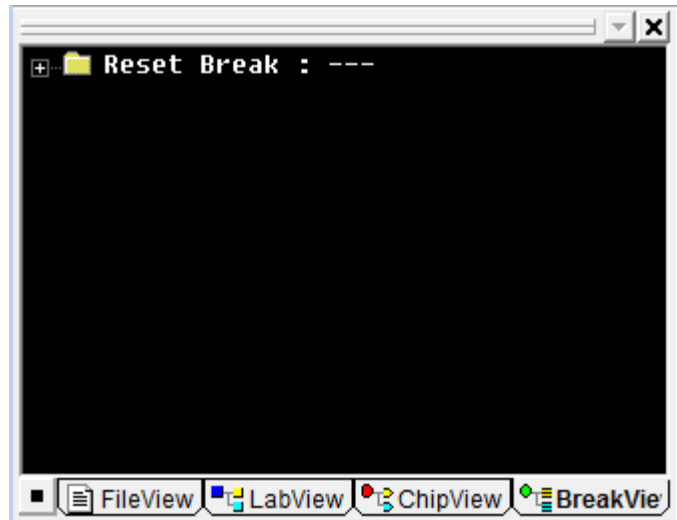


Fig.4-18. Break View window information

2) Reset Break: Detects whether Reset occurs.

4.3.4. Disassembly code of development tool

In a pure assembly language environment, a single step may be interpreted as executing an assembly language instruction; But in the environment of Source Level Debug, "single step" means executing a single line of instructions, which may contains many assembly language instructions, such as macros, or C.

If you want to step through assembly instructions, you should select **Disassembly**(as shown in figure 4-19) and click **Single** (F11) after you entered the debugging environment.

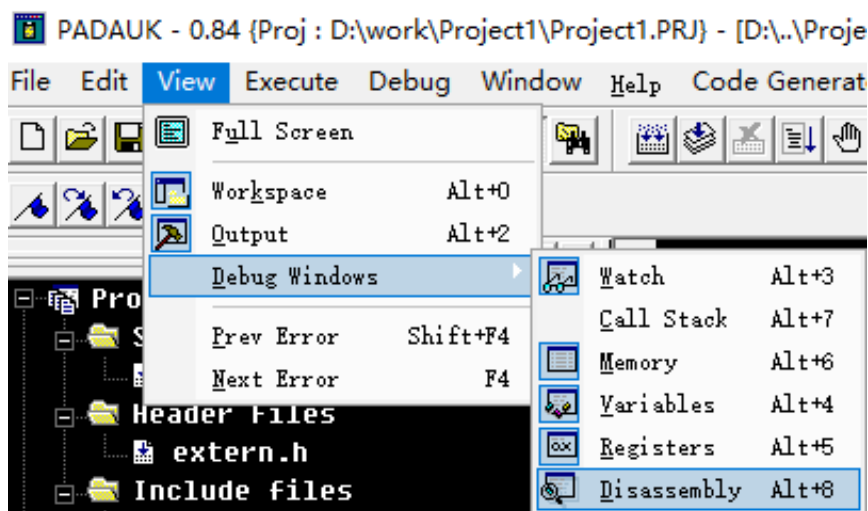


Fig.4-19. Disassembly selection window

In addition, you can right click the assembly code window and select **Save List File** in the pop-up window.

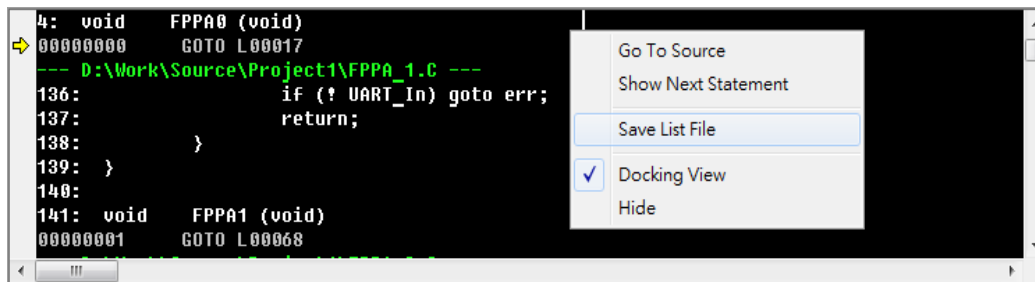


Fig.4-20. Disassembly window

4.4. Practical example

Here is an incorrect example:

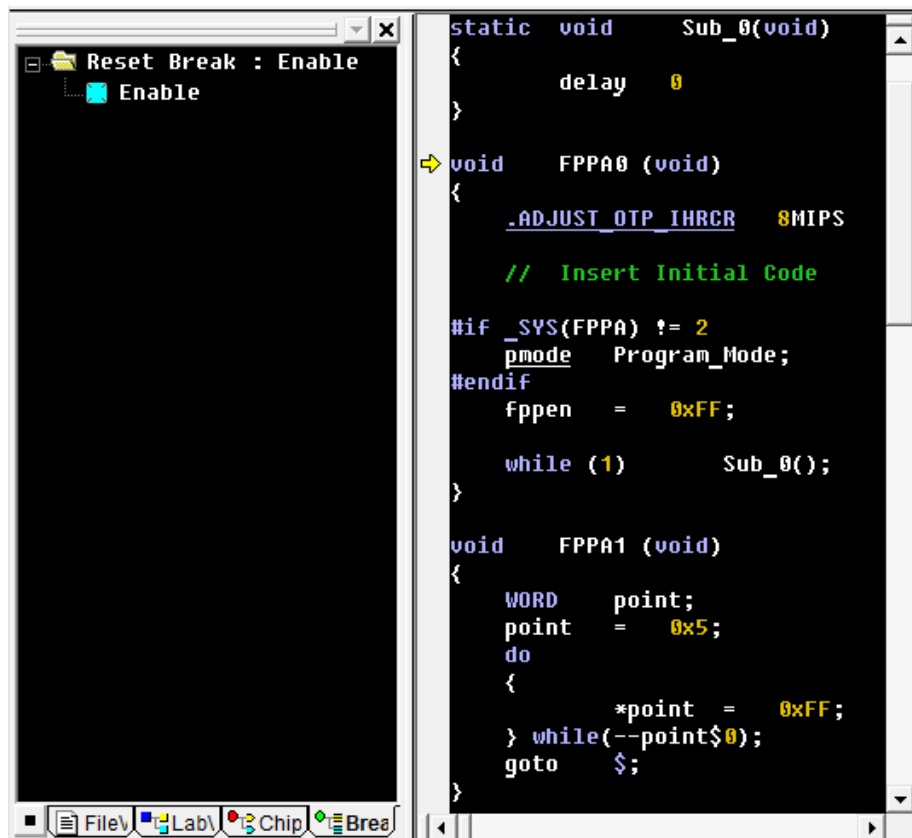


Fig.4-21. The wrong example window

After calibrating IHRC, FPPA0 enters an infinite loop and calls Sub_0. FPPA1 cleans RAM. After ICE executed Free Run, the ROM Break will be triggered as look-up instruction is used during “.ADJUST_OTP_IHRCR(or .ADJUST_IC SYSClk=IHRC/x,IHRC=16MHz)” calibrating IHRC. This is a normal behavior of the program.

Execute Free Run again, and the following information appears on ICE, indicating that the program was jump from 0x18 to illegal 0x1F3B. As shown in figure 4-22.

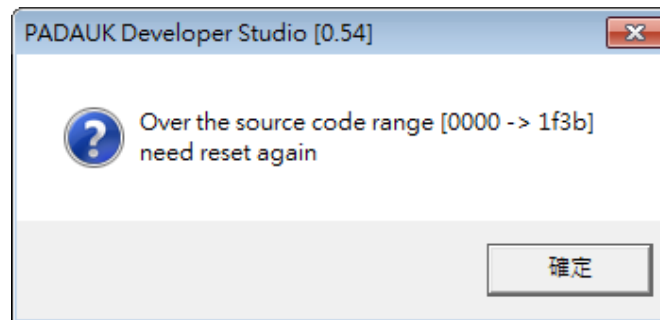


Fig.4-22. The illegal information window of IDE

Look through the disassembly window, address 0x18 is the ret instruction of FPPA0, the value of SP is 0x00; Then look through the Memory window, SP(its name in system is ?_Stk_0) is also in the stack range of FPPA0, but RAM 0/1 was broken to 0x3b/0x1f. That explains why IP is 0x1f3b after RET(in this example, ICE's IP valid range is 0x1FFF, so 0xFF3B --> 0x1F3B).

The other distinct possibility is that FPPA1's Point accidentally crosses the boundary and modifies RAM 0/1 (such as IDXN Point, A, * Point = xx, etc.). On this condition to check FPPA1, we find that the Point of FPPA1 really crossed the boundary. This method is also suitable to more complex situations.

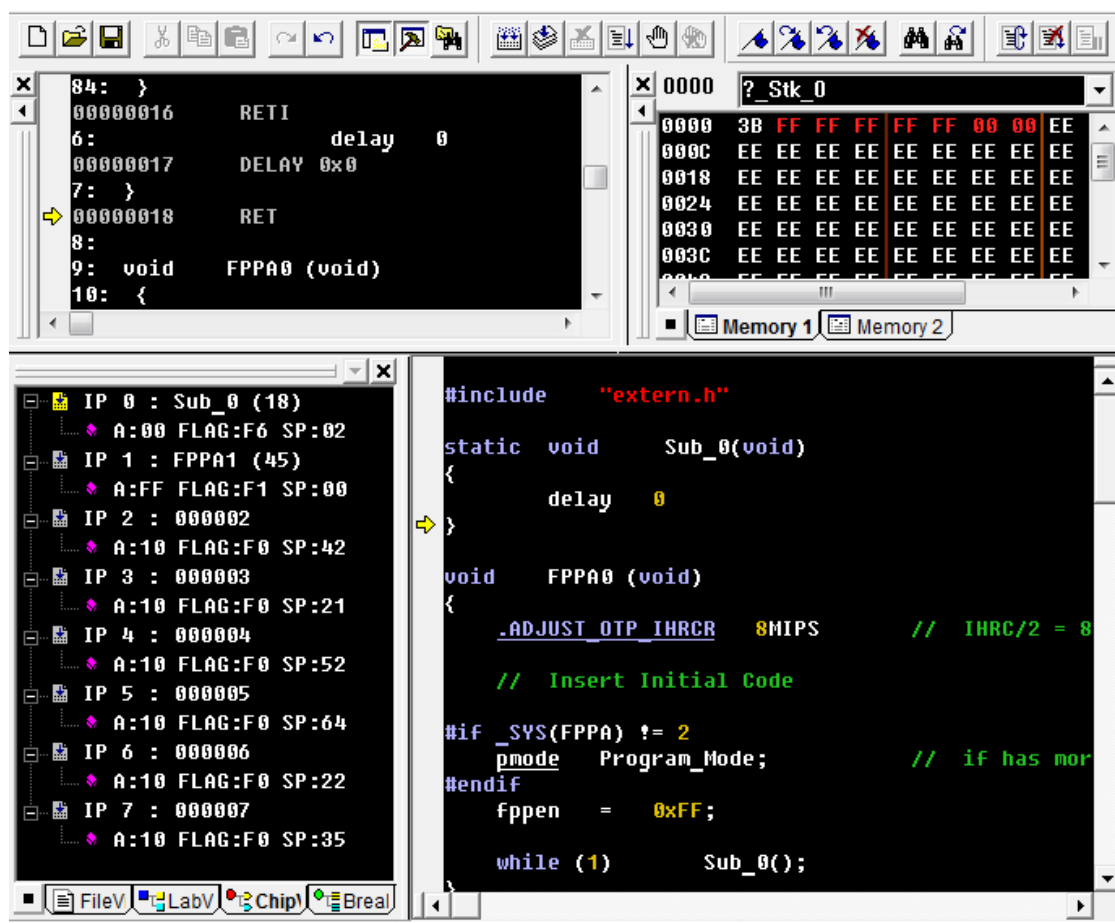


Fig.4-23. The window when checking FPPA1