Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

FR Family MB2198-01 Emulator System Getting Started Guide

Doc. No. 002-05222 Rev. *A

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
http://www.cypress.com

Copyrights

Trademarks

Source Code

Disclaimer

Contents

1. Introduction

This document will help you how to debug an emulation system with the MB2198-01 Emulation with the Softune Workbench V60L06. For in-depth information please refer to the following manuals:

- MB2198-01 Hardware Manual (Emulator)
- MB2198-01 Getting Started Application Note (MCU-AN-391027)
- MB2198-01 Installation Guide Application Note (MCU-AN-391026)

This document describes the debugging methods of a MB91V460 system together with a SK-91F467-Flexray target board. Please note, that the debugging principle is the same for other systems.

2. Sample Program

Sample Program for Debugging

2.1 Start new project with Softune Workbench

At first choose an evaluation MCU (here: MB91467D), copy the template project of the “Softune samples” into an own folder (here: “Emulation_Test”) and start the Softune Workbench Software.

2.2 “Main.c”

The following sample code program, based on the standard template project, is used for demonstrating emulation and debugging. Please change “Main.c” to the following:

```
/************************@INCLUDE_START************************/
#include "mb91467d.h"
#include "vectors.h"
/************************@INCLUDE_END************************/

/******************@FUNCTION_DECLARATION_START*****************/
void wait (short int cnt)
{
    int i;
    PDR16 = 0xFF;
    for(i=0;i<cnt;i++);
    PDR16 =0x00;

}

void main(void)
{
    __EI();                     /* enable interrupts */
    __set_il(31);               /* allow all levels */
    InitIrqLevels();            /* init interrupts */

    PORTEN = 0x3;               /* enable I/O Ports */
                                /* This feature is not supported by MB91V460A */
                                /* For all other devices the I/O Ports must be enabled*/
    PFR16 = 0x00;
    DDR16 = 0xFF;

    while(1)                    /* endless loop */
    {
       HWWD_CL = 0;
       __asm(" NOP");
       wait (5000);
       __asm(" NOP");
    }
}
```

This program is only an example with no “great assignment“. It contains a simple wait-function (`void wait`) , which needs an short integer value for the wait time. The resulting delay time depends on the value itself and the clock speed of the emulation system.

At first the interrupts are enabled (although they are not used in this example), then the Port16 of the MCU is set to “output“ (Port16 is the LED-Port of the SK-91F467-Flexray board). Then the `wait` function is called with the value `5000`.

2.3 Compiling “Main.c”

To compile the project, please use “Setup Project” first. In Project→Setup Project→C Compiler→Category: Optimize has to be selected General-purpose Optimization Level: **None**.

Then compile the project by clicking on  Build all source files regardless of data or selecting *Project→Build*, or pressing “Ctrl-F8”.

Watch for error messages. If all is ok, you will get the following message:

Now building...

----------Configuration: 91460_template_91467D.prj - STANDALONE----------

vectors.c

Start91460.asm

mb91467d.asm

MAIN.c

Now linking...

<your path>91460_template_91467d\STANDALONE\ABS\91460_template_91467d.abs

Now starting load module converter...

<your path>\91460_template_91467d\STANDALONE\ABS\91460_template_91467d.mhx

----

No Error.

----

3. Debugging, First Steps

How to Enter Debugging Mode

3.1 Setup Hardware

For the next steps you have to set up your emulation hardware. Please refer to the application notes “Installation Guide MB2198-01” (MCU-AN-391026) and “Emulator System MB2198-01, Getting started” (MCU-AN-391027) for details.

3.2 Entering Debugger Mode

After successful compilation of the project start the debugging mode via COM1/2, USB or LAN by double clicking on the regarding Debug-“.sup”-entry in the workspace window. After successful connection to the emulator, reset MCU, open “Main.c” (close it first, if it is open) and then click on right mouse button and select “*Mix Display*”. Your generated code should look like the following sample code.

```
          /*************************@INCLUDE_START************************/
          #include "mb91467d.h"
          #include "vectors.h"
          /*************************@INCLUDE_END************************/

          /*******************@FUNCTION_DECLARATION_START*****************/
          void wait (short int cnt)
000402A0: 1704            ST      R4,@-R15
000402A2: 1781            ST      RP,@-R15
000402A4: 0F02            ENTER   #008
          {
                int i;
                PDR16 = 0xFF;
000402A6: CFF0            LDI:8   #FF,R0
000402A8: 8B0D            MOV     R0,R13
000402AA: 1A10            DMOVB   R13,@010
                for(i=0;i<cnt;i++);
000402AC: C000            LDI:8   #00,R0
000402AE: 3FF0            ST      R0,@(R14,-4)
000402B0: 4050            LDUH    @(R14,10),R0
000402B2: 97A0            EXTSH   R0
000402B4: 2FF1            LD      @(R14,-4),R1
000402B6: AA01            CMP     R0,R1
000402B8: EB04            BGE     000402C2
000402BA: 2FF0            LD      @(R14,-4),R0
000402BC: A410            ADD     #1,R0
000402BE: 3FF0            ST      R0,@(R14,-4)
000402C0: E0F7            BRA     000402B0
                PDR16 =0x00;
000402C2: C000            LDI:8   #00,R0
000402C4: 8B0D            MOV     R0,R13
000402C6: 1A10            DMOVB   R13,@010

          }
000402C8: 9F90            LEAVE
000402CA: 0781            LD      @R15+,RP
000402CC: A301            ADDSP   #4
000402CE: 9720            RET

          void main(void)
000402D0: 1781            ST      RP,@-R15
```

```
000402D2: 0F01            ENTER    #004
          {
              __EI();                  /* enable interrupts */
000402D4: 9310            ORCCR    #10
              __set_il(31);            /* allow all levels */
000402D6: 871F            STILM    #1F
              InitIrqLevels();         /* init interrupts */
000402D8: 9F8C00040000    LDI:32   #00040000,R12
000402DE: 971C            CALL     @R12

              PORTEN = 0x3;            /* enable I/O Ports */
000402E0: C030            LDI:8    #03,R0
000402E2: 9F8C00000498    LDI:32   #00000498,R12
000402E8: 16C0            STB      R0,@R12
                                       /* This feature is not supported by MB91V460A */
                                       /* For all other devices the I/O Ports must be enabled*/
                   PFR16 = 0x00;
000402EA: C000            LDI:8    #00,R0
000402EC: 9F8C00000D90    LDI:32   #00000D90,R12
000402F2: 16C0            STB      R0,@R12
                   DDR16 = 0xFF;
000402F4: CFF0            LDI:8    #FF,R0
000402F6: 9F8C00000D50    LDI:32   #00000D50,R12
000402FC: 16C0            STB      R0,@R12

              while(1)                 /* endless loop */
              {
                 HWWD_CL = 0;
000402FE: 9F80000004C7    LDI:32   #000004C7,R0
00040304: 8070            BANDL    #7,@R0
                 __asm(" NOP");
00040306: 9FA0            NOP
                 wait (5000);
00040308: 9B041388        LDI:20   #01388,R4
0004030C: D7C9            CALL     \wait
                 __asm(" NOP");
0004030E: 9FA0            NOP
00040310: E0F6            BRA      000402FE
              }
          }
00040312: 9F90            LEAVE
00040314: 0081            LD       @(R13,R8),R1
00040316: 9720            RET


        /*******************@FUNCTION_DECLARATION_END*****************/
```

3.3 Using Bookmarks

Since Softune version V60L06 it is possible to set bookmarks in source code windows.

Bookmarked lines are marked with completely in green in source code lines. With the bookmark arrow buttons it can be stepped through the code, stopping at bookmarked lines.

3.4 Start Execution

To enter the run mode, click on  Run continuously or select *Debug→Run→Go*, or press “F5”.

Now the program is being executed. If you are using a target system with LEDs on Port16, you will see the LEDs flicker if SK-91F467D-Flexray Target board is used.

3.5 Stop Execution

To stop the MCU, click on  Stop execution or select *Debug→Abort*.

Now the system is halted, but it can be continued again by clicking on “*Run continuously*” or selecting “*Go*”.

3.6 Reset MCU

To reset the MCU, clock on  Reset MCU or select *Debug→Reset of MCU*.

Note: This works only if the application is stopped (e.g. breakpoint, etc.)

4. Monitoring and Manipulating

How to Monitor and Manipulate CPU Registers, Variables and Memory

4.1 Monitoring and Manipulating Processor Status

The Condition Code Register (CCR) is always displayed below the workspace window.

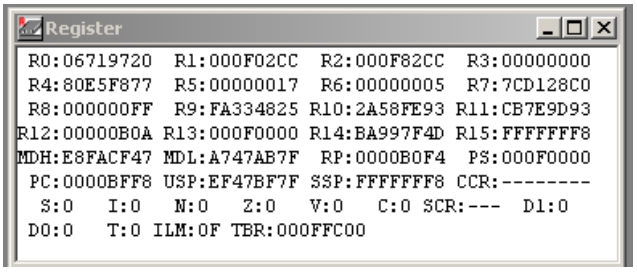
The flags are:

Abbr.	Flag Names
I	Interrupt enable flag (1 = enable)
S	Stack flag (0 = User stack; 1 = System stack)
T	Sticky bit flag (1 = shift right instruction executed)
N	Negative flag (MSB = 1 in last operation)
Z	Zero flag (Last operation resulted in “0”)
V	Overflow flag (Overflow at last operation)
C	Carry flag (Last operation caused carry)

The value of the flags can be easily changed by clicking into the white square. A “check mark” (√) indicates that the flag is set (== 1).

4.2 Monitoring and Manipulating CPU Registers

To display the CPU Registers window choose in the debugging mode: *View -> Register*. A new window will occur and look like this:



The registers are:

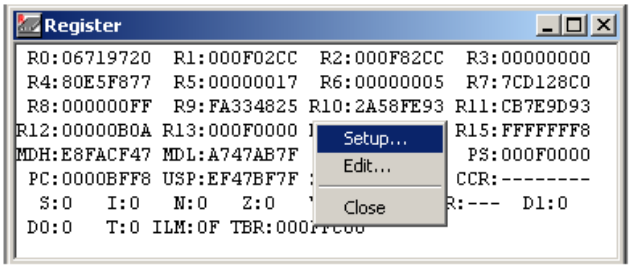
Abbr.	Flag Names
R0-R12	General purpose registers
R13	Virtual Accumulator
R14	Frame pointer
R15	Stack pointer
PC	Program counter
PS	Program status
TBR	Time Base register
RP	Return pointer
SSP	System stack pointer
USP	User stack pointer
MDH, MDL	Multiply-Divide register
TBR	Interrupt vector Table base register

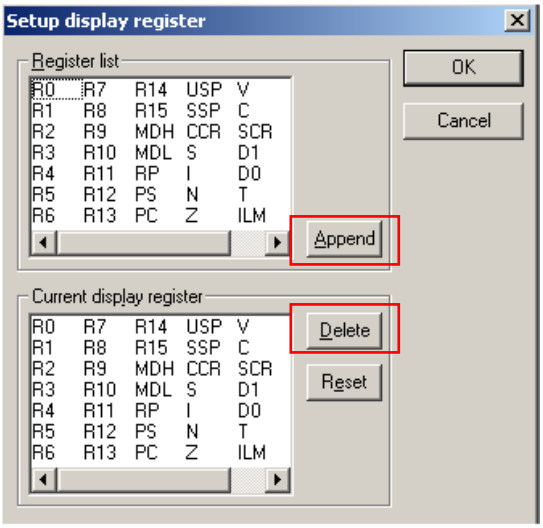
The contents of these registers can be changed by double-clicking them. A pop-up window will occur and look like the following picture:

Under *Register value* option one can enter a new value for the register. Note, that the values always are shown in hexadecimal notation by set-up default, but one can enter even decimal values (beginning with “D’”), binary values (beginning with “B’”), or octal values

(beginning with “O’”).

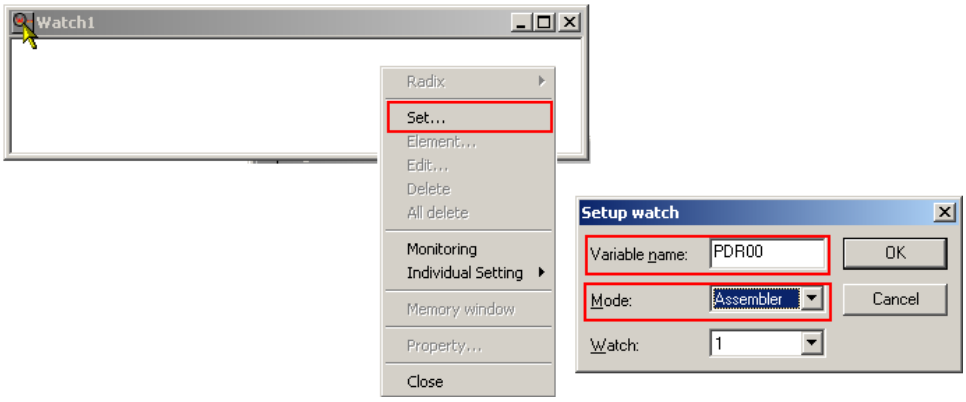
Registers can be added or removed by right clicking on *Register* window and selecting *setup…*



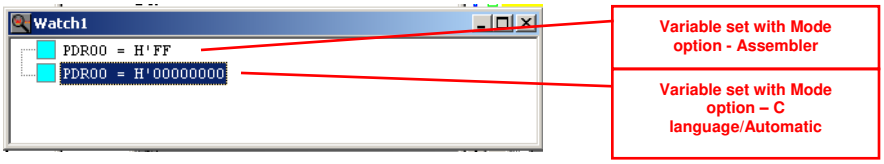


4.3 Monitoring and Manipulating Assembly Variables

To display assembly variables choose in the debugging mode: *View -> Watch -> Watch1*. A new window *Watch* will occur. Click in this window on the right mouse button and select Set… A pop-up window *Setup watch* will appear.



Under *Variable name* option one can enter the variable name of the assembly program. The Mode must be *Assembler* in this case. The Watch window will then contain the variable name and value. If we select other than this then watch window will show variables memory location and not the data contained at that location.

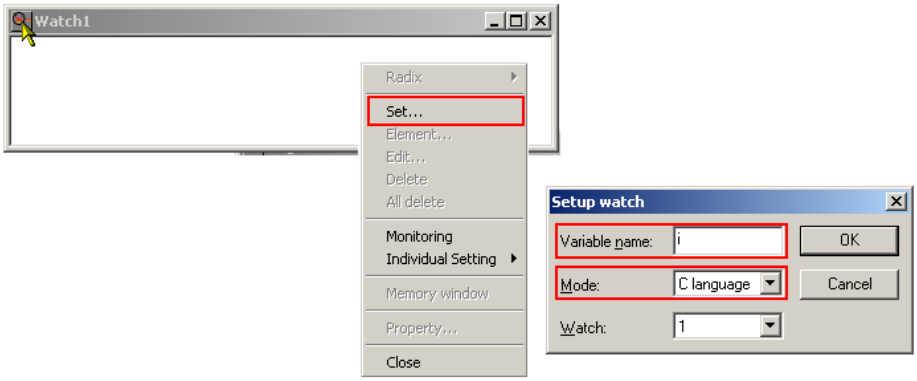


Note: You can change the radix of the value by right-clicking on the variable entry and choose via Radix: Binary, Octal, Decimal, or Hexadecimal.

To manipulate the value just double-click on the entry and enter in the pop-up window *Edit variable* a new value. The radix can be chosen via “D’”, “H’”, “B’”, or “O’”.

4.4 Monitoring and Manipulating C Variables

To display C variables choose in the debugging mode: *View -> Watch -> Watch1*. A new window *Watch* will occur. Click in this window on the right mouse button and select *Set…* A pop-up window *Setup watch* will appear.



Under *Variable name* option one can enter the variable name of the C program. The Mode must be *C language or Automatic* in this case. The Watch window will then contain the variable name and value.

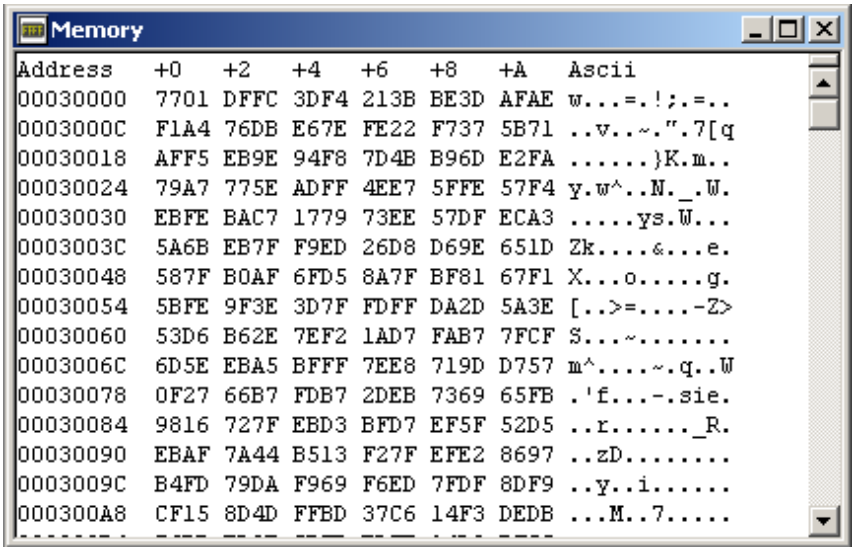
Note: You can change the radix of the value by right-clicking on the variable entry and choose via Radix: Binary, Octal, Decimal, or Hexadecimal.

To manipulate the value just double-click on the entry and enter in the pop-up window *Edit variable* a new value. The radix can be chosen via “D’”, “H’”, “B’”, or “O’”.

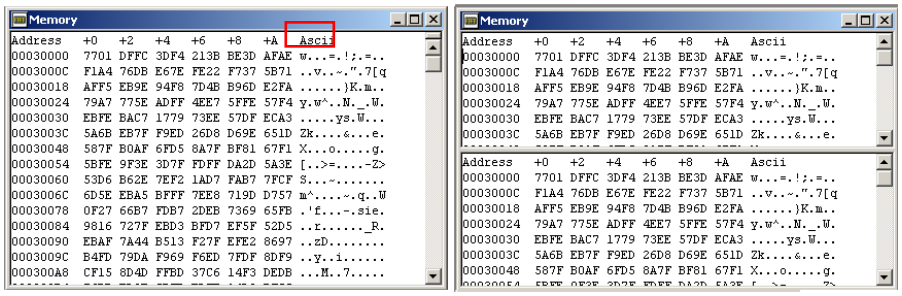
As explained in previous chapter to view memory content of Special Function Register, one should select *Assembler* under *Mode* option.

4.5 Monitoring and Manipulating Memory

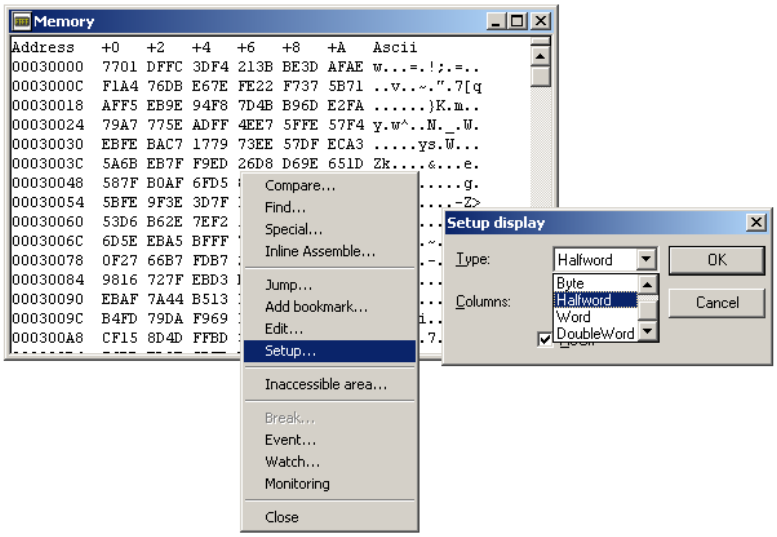
To display the MCU memory choose in the debugging mode: *View -> Memory*. A pop-up window *Memory* will occur and ask for the start address to be displayed. Type for instance H'2530 (or just 2530) for the RAM area. Following window occur which is something like a "Hex-Editor":



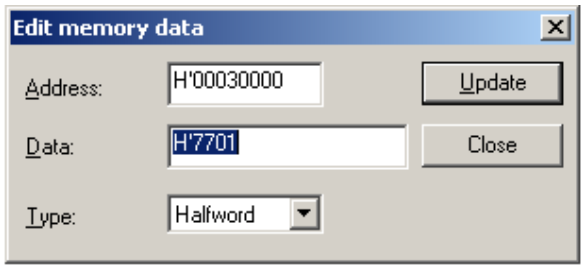
Memory window can be split. Move pointer near to top right corner, pointer will change to  ; drag it down with right mouse button pressed to split.



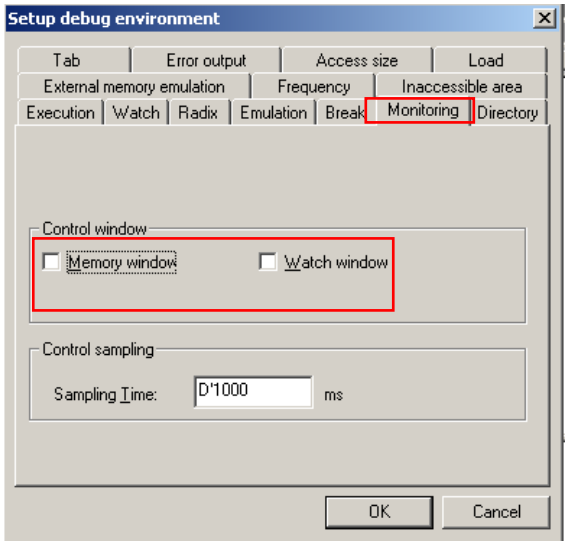
Also view can be set up to see data in bit, byte, word or long. Right click on memory window, click on setup. On a *Setup Display* dialog box select bit, byte, word or long from drop down menu.



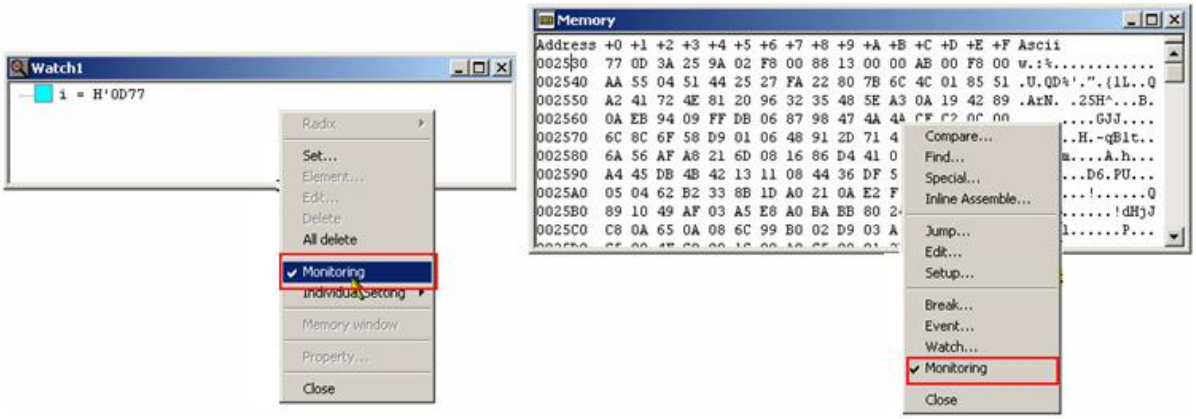
To change a memory content just double click on the respecting byte and *Edit memory data* dialog window pops up. In this window one can specify the address (default is the address of the clicked byte) and the new value. The value can be entered in hexadecimal, decimal, binary or octal format.



To see the memory in “real time” during execution, choose *Setup -> Debug environment -> Debug environment...* Then select the *Monitoring* tab and enter “D’1000” at *Sampling Time* and under *Control window* option click on check box *Memory window* and *Watch window*



Now, when the program is executed, *Watch* and *Memory* window is updated at 1000 ms, and one can see the addresses H'2530 in *Memory* window and variable 'i' in *Watch* window changing its values. Alternately one can select the same functionality by right clicking on *Watch/Memory* window, and on popup menu clicking on *Monitoring*

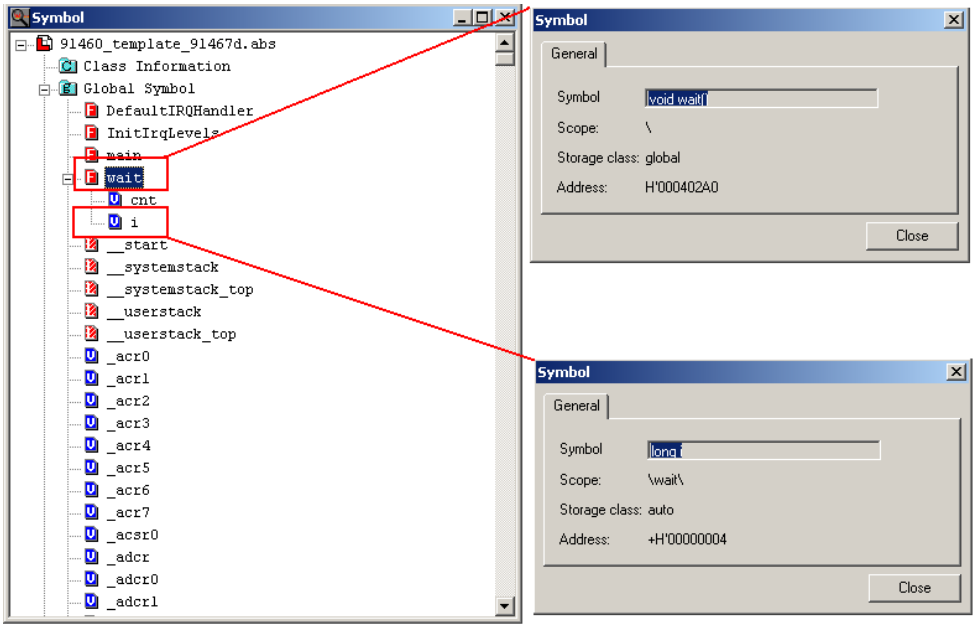


4.6 Symbol view

If one want to know where a variable is located in the memory than one can choose

View -> Symbol. Then unfold the sub list Project_name.abs/Global Symbol. Click with the right mouse button to the variable you want to get information about and select *Property…* .

The Symbol sub window shows then the address:



Icon Reference

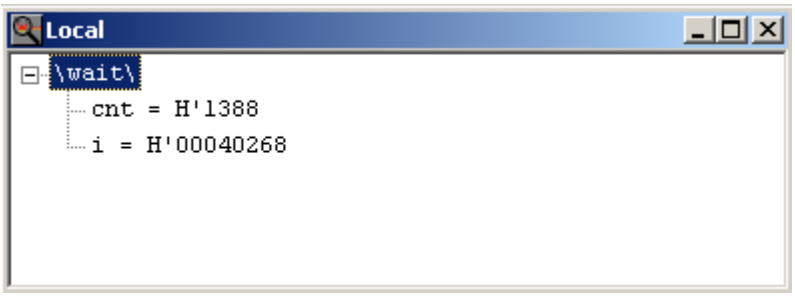
The following icons are used:

Icons	Flag Names
	Function
	Variable
	Label

4.7 Local variables

Local variables of functions can be displayed via View ->Local. A new window will open.

Note, that this window only shows contents if the debugger is in stop mode (e.g. breakpoint reached) and the actual function has local variables.



5. Breakpoints

How to Set Break Points

Code Break Point:

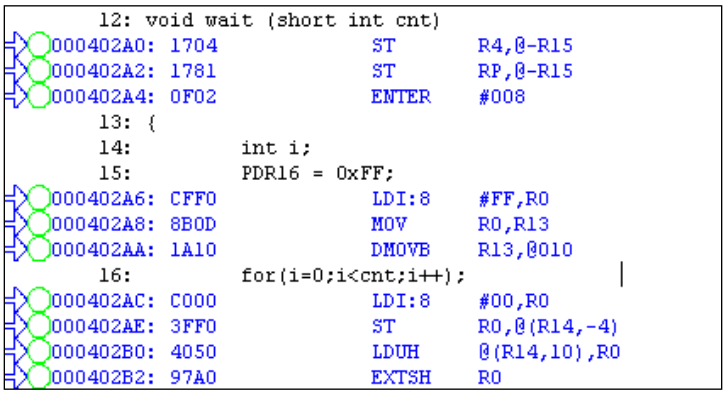
When code break point is set, program execution stops when the PC passes the break address (when instruction at that address is executed).

Four hardware code break points can be set and 4096 software break points can be set. For software break point, program halts every time when PC passes through the set address.

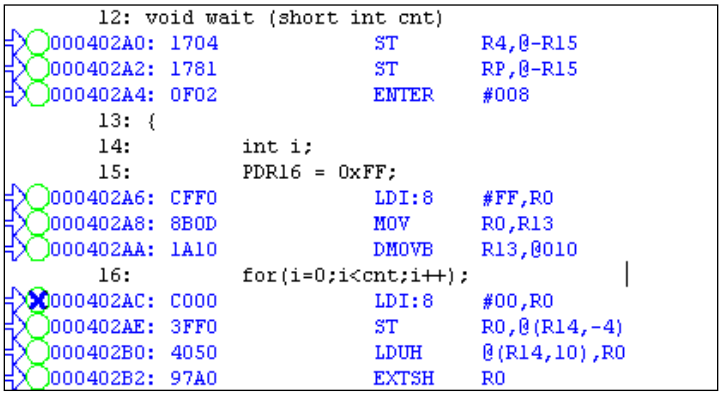
5.1 Setting Break points

5.1.1 Setting code break point through editor window

Each assembler line in the mixed mode display of the source code has a blue arrow and a green circle symbol



In these lines clicking into the circle can set a breakpoint or right click into the circle, click on Break Point Set/Reset. The symbol then turns to

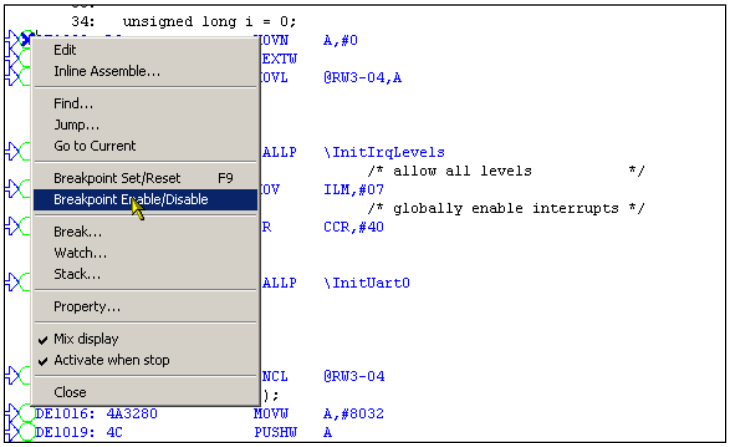


Depending on the selected break point type, a differently colored cross is shown in the circle. For the software break point (default), the cross is blue, while there is a red cross in the circle for the hardware break point

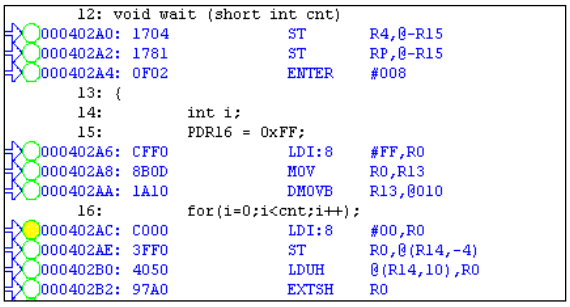
If you now start the execution, the CPU will halt on this break point. The actual line gets a yellow background color.

Clicking to this circle again or right clicking into the circle and clicking on Break Point Set/Reset, releases the break point.

Alternately, break point can be disabled by right clicking and on popup menu clicking Break Point enable/Disable.



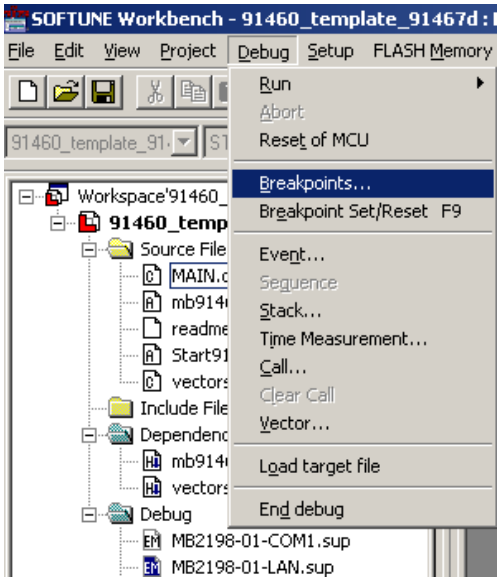
The symbol will change to



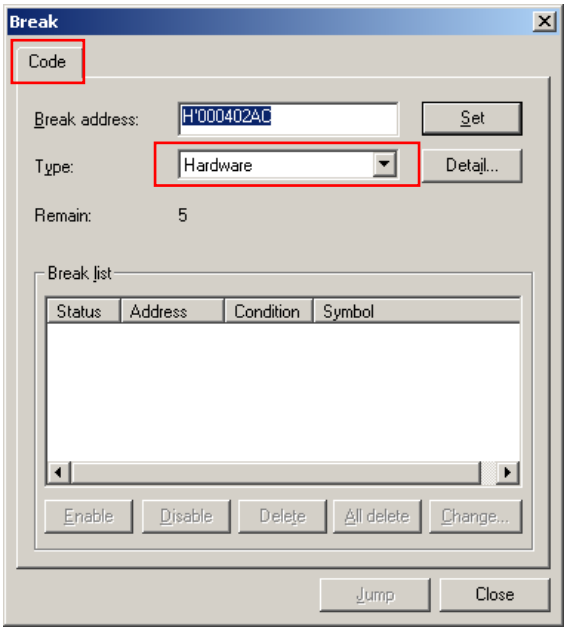
5.1.2 Setting code break point using Dialog box

A code break point can also be set using Dialog box

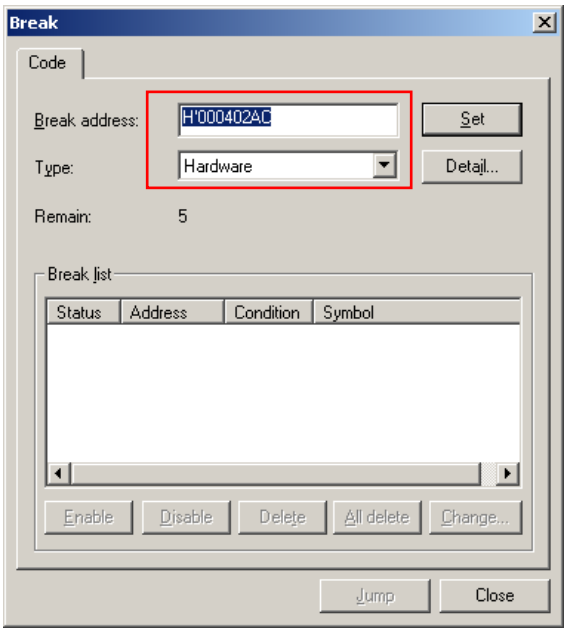
Open break point dialog box by clicking on *Breakpoints…* on Debug menu.



On *code* tab, select *software* or *Hardware* type breakpoint.

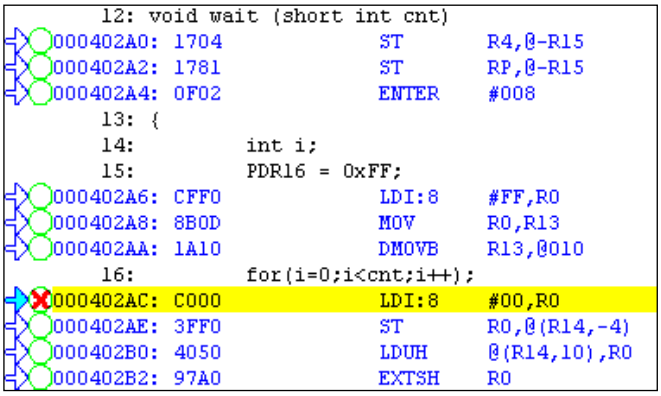


Type desired *Break address*

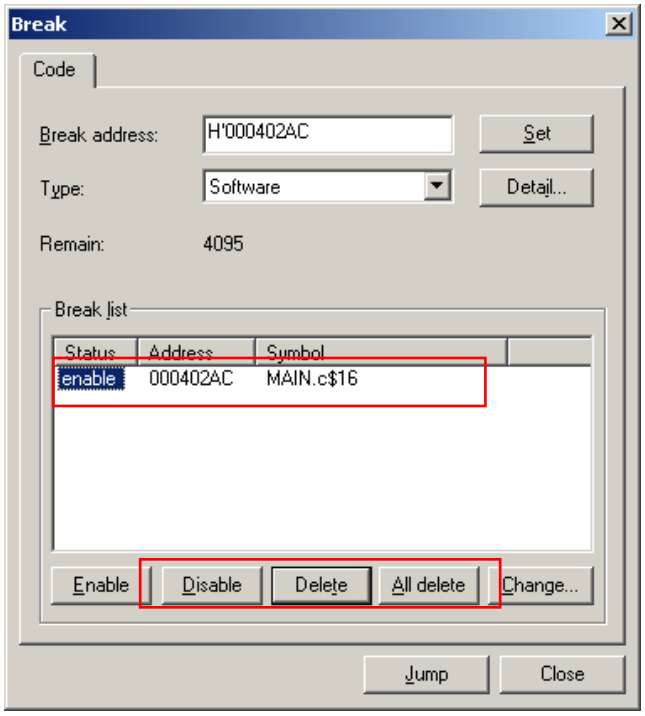


Click on *set,* to set software or Hardware breakpoint at required address

If you now start the execution, the CPU will halt on this break point. The actual line gets a yellow background color.



Breakpoint can be released by selecting breakpoint from *Break list,* and clicking on *Delete* button, alternately it can also be disabled by clicking *Disable* button



5.2 Position of Break point

When break points are set to some location and after that code is modified and build again, position of previously set break point will be decided as per following rule

- Break point will be set to the same source code line number
- If it is not possible to meet above condition, it will be set to the same address location

For example,