

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

© Cypress Semiconductor Corporation, 2003-2016. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Trademarks

All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Source Code

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer

CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.

Contents



1.	Introduction				
2.	Sam	Sample Program			
	2.1	Start new project with Softune Workbench	7		
	2.2	"Main.c"	7		
	2.3	Compiling "Main.c"	8		
3.	Deb	ugging, First Steps	9		
	3.1	Setup Hardware	9		
	3.2	Entering Debugger Mode	9		
	3.3	Using Bookmarks	11		
	3.4	Start Execution	12		
	3.5	Stop Execution	12		
	3.6	Reset MCU	12		
4.	Mon	nitoring and Manipulating	13		
	4.1	Monitoring and Manipulating Processor Status	13		
	4.2	Monitoring and Manipulating CPU Registers	13		
	4.3	Monitoring and Manipulating Assembly Variables	15		
	4.4	Monitoring and Manipulating C Variables	16		
	4.5	Monitoring and Manipulating Memory	17		
	4.6	Symbol view	19		
	4.7	Local variables	20		
5.	Brea	akpoints	21		
	5.1	Setting Break points	21		
		5.1.1 Setting code break point through editor window	21		
		5.1.2 Setting code break point using Dialog box	23		
	5.2	Position of Break point	25		
6.	Evei	nts	28		
	6.1	How to Set Code and Data Events	28		
		6.1.1 Code Event	29		
		6.1.2 Data Event			



7.	Trace	38
	7.1 Trace Window	
	7.2 Trace View	40
	7.3 Trace Jump	41
	7.4 Back Trace	43
	7.5 Search Trace	44
	7.6 Trace Setup	45
	7.7 Saving Trace Data	47
8.	Time Measurement	48
9.	Call Stack	50
10.	Function Call	51
11.	Vector	53
	11.1 Display and setting vectors	
	11.1.1 Display	
	11.1.2 Setting an address	
	11.2 Jump	
12.	Debug Environment Setup Procedure	
	12.1 Execution	
	12.2 Watch	
	12.3 Radix	
	12.4 Emulation	
	12.5 Breakpoint	
	12.6 Monitoring	
	12.7 Directory	61
	12.8 Tab	61
	12.9 Error output	62
	12.10 Access Size	63
	12.11 Load	64
	12.12 External Memory Emulation	65
	12.13 Frequency	66
	12.14 Inaccessible area	66
13.	Trigger-Input and Emulator-Output	68
	13.1 The BNC Connectors	68
	13.2 Trigger-Input	68
	13.3 Emulator-Output	69
14.	Installing LAN	70
	14.1 Overview	70
	14.2 Configuring the LAN Adapter	70
	14.3 Configuring Operating System "Windows [™] "	
	14.4 Checking the network-connection	74
	14.5 Troubleshooting	74
	14.6 Softune Workbench	



15.	Miscellaneous	.77
	15.1 View Mode of the Editor	.77
16.	Appendix	.78
	16.1 Related Documents	.78
17.	Additional Information	.79
Revi	sion History	.80
	Document Revision History	.80

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 "mb91467d.h"
#include "vectors.h"
void wait (short int cnt)
{
   int i;
PDR16 = 0xFF;
for(i=0;i<cnt;i++);</pre>
   PDR16 =0 \times 00;
3
void main(void)
   __EI();
__set_il(31);
InitIrqLevels();
                          /* enable interrupts */
                          /* allow all levels */
                           /* 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.

Compiling "Main.c" 2.3

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 selecting Project→Build, or pressing "Ctrl-F8".

Build all source files regardless of data

or

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 "mb91467d.h"
         #include "vectors.h"
         void wait (short int cnt)
0000402A0: 1704
                        ST
                               R4,0-R15
  000402A2: 1781
                        ST
                               RP,@-R15
  000402A4: 0F02
                        ENTER
                               #008
         {
               int i;
               PDR16 = 0xFF;
  000402A6: CFF0
                        LDI:8
                               #FF,RO
  000402A8: 8B0D
                        MOV
                               R0,R13
  000402AA: 1A10
                        DMOVB
                               R13,0010
               for(i=0;i<cnt;i++);</pre>
  000402AC: C000
                        LDI:8
                               #00,R0
  000402AE: 3FF0
                               R0,0(R14,-4)
                        ST
  000402B0: 4050
                        LDUH
                               @(R14,10),R0
 0000402B2: 97A0
                       EXTSH
                               RO
 0000402B4: 2FF1
                       LD
                               0(R14,-4),R1
 0000402B6: AA01
                       CMP
                               R0,R1
 0000402B8: EB04
                               000402C2
                       BGE
 0000402BA: 2FF0
                        LD
                               @(R14,-4),R0
 ○000402BC: A410
                        ADD
                               #1,R0
  000402BE: 3FF0
                        ST
                               R0,0(R14,-4)
  000402C0: E0F7
                               000402B0
                        BRA
               PDR16 =0x00;
  00040202: 0000
                               #00,R0
                       LDI:8
  000402C4: 8B0D
                        MOV
                               R0,R13
-X 0000402C6: 1A10
                        DMOVB
                               R13,0010
         }
  00040208: 9F90
                        LEAVE
  000402CA: 0781
                        LD
                               @R15+,RP
  000402CC: A301
                        ADDSP
                               #4
  000402CE: 9720
                        RET
         void main(void)
→ 0000402D0: 1781
                        ST
                               RP,0-R15
```



≓X)000402D2: 0F01 ENTER #004 ł EI(); /* enable interrupts */ 000402D4: 9310 ORCCR #10 /* allow all levels */ set_i1(31); 000402D6: 871F STILM #1F InitIrqLevels(); /* init interrupts */ 000402D8: 9F8C00040000 #00040000,R12 LDI:32 000402DE: 971C CALL @R12 PORTEN = 0x3;/* enable I/O Ports */)000402E0: C030 LDI:8 #03,R0)000402E2: 9F8C00000498 LDI:32 #00000498,R12 000402E8: 16C0 R0,0R12 STB /* This feature is not supported by MB91V460A */ /* For all other devices the I/O Ports must be enabled*/ PFR16 = 0x00:000402EA: COOO #00,R0 LDI:8 000402EC: 9F8C00000D90 #00000D90,R12 LDI:32 000402F2: 16C0 STB R0,0R12 DDR16 = OxFF; 000402F4: CFF0 LDI:8 #FF,RO #00000D50,R12 000402F6: 9F8C00000D50 LDI:32 000402FC: 16C0 STB R0,0R12 while(1) /* endless loop */ ł $HWWD_CL = 0;$ 000402FE: 9F80000004C7 LDI:32 #000004C7,R0 00040304: 8070 BANDL #7,0R0 asm(" NOP"); 00040306: 9FA0 NOP wait (5000); 00040308: 9B041388 #01388,R4 LDI:20 0004030C: D7C9 CALL \wait asm(" NOP");)0004030E: 9FA0 NOP -X)00040310: E0F6 BRA 000402FE } } 00040312: 9F90 LEAVE 00040314: 0081 LD @(R13,R8),R1 00040316: 9720 RET

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 $\underline{D}ebug \rightarrow \underline{R}un \rightarrow \underline{G}o$, 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



Ē↓

Stop execution or se

or select <u>D</u>ebug→<u>A</u>bort.

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 <u>Debug</u> \rightarrow Rese<u>t</u> of MCU.

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

Ð

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.

	Z S	ГΤ	ΠN	Πz	\Box V	ΠC
--	-----	----	----	----	----------	----

The flags are:

Abbr.	Flag Names			
1	nterrupt enable flag (1 = enable)			
S	Stack flag (0 = User stack; 1 = System stack)			
Т	Sticky bit flag (1 = shift right instruction executed)			
Ν	legative flag (MSB = 1 in last operation)			
Z	Zero flag (Last operation resulted in "0")			
V	Overflow flag (Overflow at last operation)			
С	Carry flag (Last operation caused carry)			

The value of the flags can be easily changed by clicking into the white square. A "check mark" ($\sqrt{}$) 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:

ĺ	Register _ 🗆 🗙
I	R0:06719720 R1:000F02CC R2:000F82CC R3:00000000
I	R4:80E5F877 R5:00000017 R6:00000005 R7:7CD128C0
I	R8:000000FF R9:FA334825 R10:2A58FE93 R11:CB7E9D93
I	R12:00000B0A R13:000F0000 R14:BA997F4D R15:FFFFFFF8
I	MDH:E8FACF47 MDL:A747AB7F RP:0000B0F4 PS:000F0000
I	PC:0000BFF8 USP:EF47BF7F SSP:FFFFFF8 CCR:
I	S:0 I:0 N:0 Z:0 V:0 C:0 SCR: D1:0
l	D0:0 T:0 ILM:OF TBR:000FFC00



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:

Edit register		×
<u>R</u> egister name:	RO 💌	ОК
Register <u>v</u> alue:	H'06719720	Cancel

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...

🚾 Registe	r			
R0:06719	720 R1:	000F02CC	R2:000F82CC	R3:00000000
R4:80E5F	877 R5:	00000017	R6:00000005	R7:7CD128C0
R8:00000	0FF R9:	FA334825	R10:2A58FE93	R11:CB7E9D93
R12:00000			betup	R15:FFFFFFF8
MDH:E8FAC PC:0000E		A747AB7F EF47BF7F	Edit	PS:000F0000 CCR:
S:0 I	:0 N:0	Z:0	Close	R: D1:0
D0:0 1	:0 ILM:0	F TBR:00		

Setup display registe	er	×
<u>R</u> egister list		ОК
R1 R8 R15 S R2 R9 MDH C R3 R10 MDL S R4 R11 RP I R5 R12 PS N R6 R13 PC Z		Cancel
•	▶ <u>Append</u>	
Current display registe		
R0 R7 R14 L	JSP V Delete	
	SSP C	
R3 R10 MDL S B4 B11 BP L	5 D1 <u>Reset</u> D0	
R5 R12 PS N	I T	
R6 R13 PC Z		





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.

Watch1			
	Radix 🕨		
	Set Element Edit		
	Delete All delete	Setup watch	×
	Monitoring Individual Setting	Variable <u>n</u> ame: PDR00	ОК
	Memory window		Cancel
	Property	Watch: 1 ▼	[
	Close		

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.

Kara Kara Kara Kara Kara Kara Kara Kara	Variable set with Mode
	option - Assembler
PDR00 = H'00000000	Variable set with Mode option – C language/Automatic

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".

Edit variable		×
PDR00:		ОК
101100.		Cancel



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.

Watch1	Radix Fet Element Edit Delete		
	All delete Monitoring Individual Setting Memory window Property Close	Setup watch Variable name: Mode: C language Watch:	Cancel

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".

Edit variable		×
j.	H10003	OK
		Cancel

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



Monitoring and Manipulating Memory 4.5

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								
Address	+0	+2	+4	+6	+8	+A	Ascii	
00030000	7701	DFFC	3DF4	213B	BE3D	AFAE	$\varpi \ldots = \ldots ! ; \ldots = \ldots$	-
0003000C	F1A4	76DB	E67E	FE22	F737	5B71	v~.".7[q	
00030018	AFF5	EB9E	94F8	7D4B	B96D	E2FA	}K.m	
00030024	79A7	775E	ADFF	4EE7	5FFE	57F4	y.w^N₩.	
00030030	EBFE	BAC7	1779	73EE	57DF	ECA3	ys.W	
0003003C	5A6B	EB7F	F9ED	26D8	D69E	651D	Zk&e.	
	587F	BOAF	6FD5	8A7F	BF81	67F1	Xog.	
00030054	5BFE	9F3E	3D7F	FDFF	DA2D	5A3E	[>=Z>	
00030060	53D6	B62E	7EF2	1AD7	FAB7	7FCF	S~	
0003006C	6D5E	EBA5	BFFF	7EE8	719D	D757	m^vW	
00030078	0F27	66B7	FDB7	2DEB	7369	65FB	.'fsie.	
00030084	9816	727F	EBD3	BFD7	EF5F	52D5	rR.	
00030090							zD	
0003009C	B4FD	79DA	F969	F6ED	7FDF	8DF9	yi	
000300A8	CF15	8D4D	FFBD	3706	14F3	DEDB	M7	_

Memory window can be split. Move pointer near to top right corner, pointer will change to $\frac{1}{2}$; drag it down with right mouse button pressed to split.

Memory							_		- 🗆 🗵	🔟 Memory								- U ×
Address	+0	+2	+4	+6	+8	+À	Ascii		-	Address	+0	+2	+4	+6	+8	+A	Ascii	
00030000	7701	DFFC	3DF4	213B	BE3D	AFAE	w=.	!:.=		00030000	7701	DFFC	3DF4	213B	BE3D	AFAE	w=.!;.=	
0003000C	F1A4	76DB	E67E	FE22	F737	5B71	v	•.".7[q		0003000C	F1A4	76DB	E67E	FE22	F737	5B71	v~.".7[q	
00030018	AFF5	EB9E	94F8	7D4B	B96D	E2FA		}K.m		00030018	AFF5	EB9E	94F8	7D 4B	B96D	E2FA	}K.m	
00030024	79A7	775E	ADFF	4EE7	5FFE	57F4	y.⊎^	NW.		00030024	79A7	775E	ADFF	4EE7	5FFE	57F4	y.w^NW.	
00030030	EBFE	BAC7	1779	73EE	57DF	ECA3	3	73.W		00030030	EBFE	BAC7	1779	73EE	57DF	ECA3	ys.W	
0003003C	5A6B	EB7F	F9ED	26D8	D69E	651D	Zk	ае.		0003003C	5A6B	EB7F	F9ED	26D8	D69E	651D	Zke.	_ 1
00030048	587F	BOAF	6FD5	8A7F	BF81	67F1	Xo.	g.										
00030054	5BFE	9F3E	3D7F	FDFF	DA2D	5A3E	[>=.	Z>		Address	+0	+2	+4	+6	+8	+A	Ascii	•
00030060	53D6	B62E	7EF2	1AD7	FAB7	7FCF	s~.			00030000	7701	DFFC	3DF4	213B	BE3D	AFAE	w=.!;.=	
0003006C	6D5E	EBA5	BFFF	7EE8	719D	D757	m^	~.qW		00030000	F1A4	76DB	E67E	FE22	F737	5B71	v~.".7[q	
00030078	0F27	66B7	FDB7	2DEB	7369	65FB	.'£	sie.		00030018	AFF5	EB9E	94F8	7D4B	B96D	E2FA	}K.m	
00030084	9816	727F	EBD3	BFD7	EF5F	52D5	r	R.		00030024	79A7	775E	ADFF	4EE7	5FFE	57F4	y.w^N₩.	
00030090	EBAF	7844	B513	F27F	EFE2	8697	zD			00030030	EBFE	BAC7	1779	73EE	57DF	ECA3	ys.⊎	
00030090	B4FD	79DA	F969	F6ED	7FDF	8DF9	yi			0003003C	5A6B	EB7F	F9ED	26D8	D69E	651D	Zk&e.	
8400E000								7	- 1	00030048	587F	BOAF	6FD5	8A7F	BF81	67F1	Xg.	- 1
J									<u> </u>	00020054		0505	20.75	-	DA 2D	ENOF	r ~~ 7~	



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.

Memory									×				
Address 00030000 00030002 00030024 00030030 00030048 00030048 00030048 00030060 00030064 00030060 00030060 00030078 00030084 00030090 00030090	F1A4 AFF5 79A7 EBFE 5A6B 587F 5BFE 53D6 6D5E 0F27 9816 EBAF B4FD	76DB EB9E 775E BAC7 EB7F B0AF 9F3E B62E EBA5 66B7 727F 7A44	E67E 94F8 ADFF 1779 F9ED 6FD5 3D7F 7EF2 BFFF FDB7 EBD3 B513 F969	+6 +8 2138 BE3D FE22 F737 7D4B B96D 4EE7 5FFE 73EE 57DF 26D8 D69E Compare. Find Special Inline Ass Jump Add book Edit	5B71 E2FA 57F4 ECA3 651D	y.w^. Zk	~.". .}K. .N ys.W 	7[q m .W. J .e. .g.		falfword iyte alfword Vord ooubleWo	rd V	OK Canc	
,				- Setup Inaccessi Break Event Watch Close		a							

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.

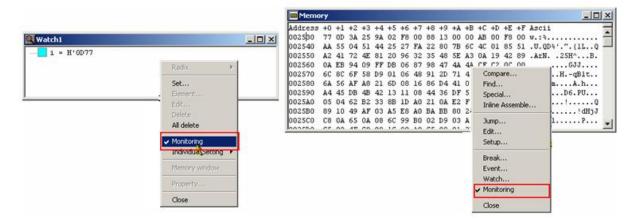
Edit memor	×	
<u>A</u> ddress:	H'00030000	<u>U</u> pdate
<u>D</u> ata:	H'7701	Close
<u>T</u> ype:	Halfword	

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*

Setup debug er	vironment				×
Tab	Error output	Access		Load	ļ
	nory emulation	Frequency			
Execution W	atch Radix Emul	ation Break	Monitori	ng Director	וצ
Control windo		atch windo	w		
Control samp	ling				
Sampling <u>T</u>	ime: D'1000	ms			
			DK I	Cancel	



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...*.

Symbol Symbo X 🖃 🖺 91460_template_91467d.abs **_** General 🙆 Class Information 🚊 💼 Global Symbol Symbol lyoid wait(🔋 DefaultIRQHandler Scope X. 🔋 InitIrqLevels 🖪 mair Storage class: global 📔 wait Address: H'000402A0 U cnt 🕛 i Close 🙎 __start 📓 __systemstack 📓 __systemstack_top 📓 __userstack 📓 __userstack_top U_acr0 Symbol × 💟 _acrl U_acr2 General 💟 _acr3 🛄 _acr4 Symbol llong 💟 _acr5 Scone \wait\ U_acr6 Storage class: auto U_acr7 U_acsr0 Address: +H'00000004 U_adcr U_adcr0 Close U _adcrl

The Symbol sub window shows then the address:



Icon Reference

The following icons are used:

Icons	Flag Names
	Function
- D	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

	12: V	oid wa:	it (short i	nt cnt)	
ΗЮ	000402A0:	1704		ST	R4,0-R15
ΗXΟ	000402A2:	1781		ST	RP,0-R15
ΗЮ	000402A4:	0F02		ENTER	#008
	13: {				
	14:		int i;		
	15:		PDR16 = 0x	FF;	
ΗXΟ	000402A6:	CFFO		LDI:8	#FF,RO
ΗXΟ	000402A8:	8B0D		MOV	R0,R13
Ю	000402AA:	1A10		DMOVB	R13,0010
	16:		<pre>for(i=0;i<</pre>	cnt;i++);	
ΗХО	000402AC:	C000		LDI:8	#00,R0
HXO	000402AE:	3FF0		ST	R0,0(R14,-4)
Ю	000402B0:	4050		LDUH	@(R14,10),R0
ŧΣ	000402B2:	97A0		EXTSH	R0

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

12: V	oid wa	it (short i	nt cnt)	
X)000402A0:	1704		ST	R4,0-R15
000402A2:	1781		ST	RP,0-R15
000402A4:	0F02		ENTER	#008
13: {				
14:		int i;		
15:		PDR16 = 0x	FF;	
X 000402A6:	CFFO		LDI:8	#FF,RO
000402A8:	8B0D		MOV	R0,R13
X 000402AA:	1A10		DMOVB	R13,0010
16:		<pre>for(i=0;i<)</pre>	cnt;i++);	
2000402AC:	C000		LDI:8	#00,R0
000402AE:	3FF0		ST	R0,0(R14,-4)
О00040280:	4050		LDUH	@(R14,10),R0
О000402В2:	97A0		EXTSH	RO

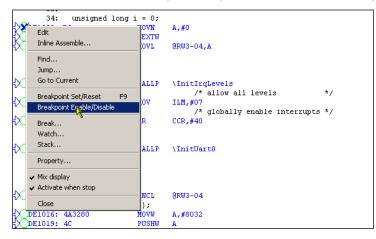


Depending on the selected break point type, a differently colored cross is shown in the circle. For the software break point (default), the cross is blue, while there is a red cross in the circle for the hardware break point

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

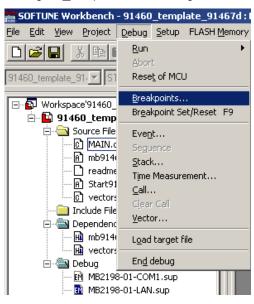
12: V	oid wa	it (short i	nt cnt)	
X)000402A0:	1704		ST	R4,0-R15
X 000402A2:	1781		ST	RP,0-R15
X)000402A4:	0F02		ENTER	#008
13: {				
14:		int i;		
15:		PDR16 = 0x	FF;	
X)000402A6:	CFFO		LDI:8	#FF,RO
X 000402A8:	8B0D		MOV	R0,R13
X 000402AA:	1A10		DMOVB	R13,0010
16:		<pre>for(i=0;i<</pre>	cnt;i++)	;
X0000402AC:	C000		LDI:8	#00,R0
000402AE:	3FF0		ST	R0,0(R14,-4)
X 000402B0:	4050		LDUH	0(R14,10),R0
<pre>\()000402B2:</pre>	97A0		EXTSH	RO



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.

Break				×
Code				
<u>B</u> reak address:	H'000	402AC		<u>S</u> et
Туре:	Hardy	ware	•	Detajl
Remain:	5			
Break <u>l</u> ist				
Status Add	dress	Condition	Symbol	
4				
Enable	Disable	Delete	<u>A</u> ll delete	Change
			Jump	Close



Type desired <u>Break address</u>

Break				X
Code				
<u>B</u> reak address: Type:	H'000 Hardv	402AC vare		<u>S</u> et Deta <u>i</u> l
Remain:	5			
Break list	ldress Disable	Condition	Symbol	►
			Jump	Close

Click on 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.

12: VC	oid wa	it (short i	nt cnt)	
X)000402A0:	1704	·	ST	R4,0-R15
-X 000402A2:	1781		ST	RP,0-R15
-X 000402A4:	0F02		ENTER	#008
13: {				
14:		int i;		
15:		PDR16 = 0x1	FF;	
→ 000402A6:	CFFO		LDI:8	#FF,RO
<pre></pre>	8B0D		MOV	R0,R13
-X 000402AA:	1A10		DMOVB	R13,0010
16:		<pre>for(i=0;i<)</pre>	cnt;i++);	
->X000402AC:	C000		LDI:8	#00,R0
-X 000402AE:	3FF0		ST	R0,0(R14,-4)
-{Х)000402В0:	4050		LDUH	@(R14,10),R0
<₩000402B2:	97A0		EXTSH	RO





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

Break			×
Code			
<u>B</u> reak address:	H'000402AC		Set
Туре:	Software	•	Detaji
Remain:	4095		
Break Jist			
<u>E</u> nable <u>D</u>	sable Dele <u>t</u> e	<u>A</u> ll delete	Change
		Jump	Close

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,

Break	Break	×
Code	Code	
Break address: H'000402BE	Break address: H'000402BE	
Type: Hardware Detail	Type: Software Detail	
Remain: 3	Remain: 4094	
Break list Status Address Condition Symbol enable 000402C4 MAIN.c\$24	Break jist Status Address Symbol enable 000402BC MAIN.c\$20	
enable 000402CA MAIN.c\$27		
Close	jump Close	



RXC -		void wait (short int cnt)
	13:	{
HXO –	14:	int i;
ΗXO –	15:	PDR16 = 0xFF;
ΗXO –	16:	for(i=0;i <cnt;i++)< th=""></cnt;i++)<>
ΗXO –	17:	{
	18:	
εXO –	19:	asm(" NOP");
-> X 1	20:	asm(" NOP");
εXO –	21:	asm(" NOP");
-> % 2	22:	asm(" NOP");
RX -	23:	asm(" NOP");
- >X 3	24:	asm(" NOP");
RC -	25:	asm(" NOP");
HXX -	26:	asm(" NOP");
- >%	27:	asm(" NOP");
RXC -	28:	asm(" NOP");
	29:	
	30:	
	31:	}
εXO –	32:	PDR16 =0x00;
	33:	
ΗXΟ –	34:	}
	35:	

Let us change the code, replace couple of $_asm(``NOP'')$; instruction with some other instruction in *wait* function.

And let us assume that Breakpoint position changes as shown below

Break X	Break	×
Code	Code	
Break address: H'00000000	Break address: H100000000 Set	
Type: Hardware Detajl	Type: Software Detajl	
Remain: 3	Remain: 4094	
Break jist Status Address Condition Symbol enable 000402C4 enable 000402CA	Break list Status Address Symbol enable 000402C6 MAIN.c\$20 enable 000402D2 MAIN.c\$22	
Enable Delete All delete Change	Enable Disable Delete All delete Change	
Jump Close	Jump	





-			
12: V	oid wa	it (short int cnt)	
₩ 0000402A0:	1704	ST	R4,0-R15
₩¥X0000402A2:	1781	ST	RP,0-R15
X 000402A4:	0F02	ENTER	#008
13: {			
14:		int i;	
15:		PDR16 = 0xFF;	
₩¥X0000402A6:	CFFO	LDI:8	#FF,RO
₩¥X0000402A8:	8B0D	MOV	R0,R13
₩X0000402AA:	1A10	DMOVB	R13,0010
16:		<pre>for(i=0;i<cnt;i++)< pre=""></cnt;i++)<></pre>	
₩ 000402AC:	C000	LDI:8	#00,R0
₩ 000402AE:	3FF0	ST	R0,0(R14,-4)
Ң∕О000402В0:	4050	LDUH	@(R14,10),R0
₩¥X0000402B2:	97A0	EXTSH	R0
₩Q000402B4:		LD	0(R14,-4),R1
₩Q000402B6:		CMP	R0,R1
≒ X)000402B8:	EB13	BGE	000402E0
17:		{	
18:		i = i+l;	
₩X_000402BA:	2FF0	LD	0(R14,-4),R0
₩X_0000402BC:	A410	ADD	#1,R0
₩¥X)000402BE:	3FF0	ST	R0,0(R14,-4)
19:		i = i+2;	
HXQ000402C0:	2 FF 0	LD	0(R14,-4),R0
HXQ000402C2:		ADD	#2,R0
HXX000402C4:	3FF0	ST	R0,@(R14,-4)
20:		i = i+3;	
HXX000402C6:		LD	0(R14,-4),R0
KQ000402C8:		ADD	#3,R0
HXX000402CA:	3FF0	ST	R0,0(R14,-4)
21:		i = i+4;	
KQ000402CC:		LD	0(R14,-4),R0
XQ000402CE:		ADD	#4,R0
₹X_000402D0:	3FF0	ST	R0,0(R14,-4)
22:		i = i+5;	
0000402D2:		LD	0(R14,-4),R0
K 000402D4:		ADD	#5,R0
X 2000402D6:		ST	R0,0(R14,-4)
K 000402D8:		LD	0(R14,-4),R0
K 000402DA:		ADD	#1,R0
K 000402DC:		ST	R0,0(R14,-4)
₩X_000402DE:	EOE8	BRA	000402B0
23:		,	
24:		} DDD16 -0:000	
25:		PDR16 =0x00;	#00 D0
0000402E0:		LDI:8	#00,R0
000402E2:		MOV	R0,R13
₩X_000402E4:	TATO	DMOVB	R13,0010
26:			
27: }			

For Software breakpoint 1, this was at source code line no. 20, since after build there is valid instruction at that line; breakpoint is maintained at same source code line main.c\$20

For Software breakpoint 2, this was at source code line no. 22, since after build there is valid instruction at that line; breakpoint is maintained at same source code line main.c\$22

For Hardware breakpoint 3, this was at source code line no. 24, since after build there is no valid instruction at that line; breakpoint is maintained at same memory location H' 000402C4

For Hardware breakpoint 4, this was at source code line no. 27, since after build there is no valid instruction at that line; breakpoint is maintained at same memory location H' 000402CA

6. Events



6.1 How to Set Code and Data Events

Two types of Event can be set. Those are Code event and Data event.

Code Event:

Two code events can be set.

Code event can be set with pass count from 1 to 65535 and address mask.

Pass count: It is the count of times PC needs to pass the set <u>A</u>ddress before program execution stops at the address.

Address Mask: A pattern of characters, 32 bits long, used to select some of the bits from the *address*. For each bit set to one in *Address <u>mask</u>* field, corresponding bit from <u>Address</u> is compared for exact match with the corresponding bit hold by PC and for each bit set to zero, corresponding bit from <u>Address</u> is ignored. Detail explanation with example is given in coming chapters.

Data Event:

When Data event is set, program execution stops when the data at the address is accessed.

As explained in code event it is possible to set data event point with Address mask

It is also possible to specify the <u>Data</u> content with <u>Data mask</u>, which is when read from or written to the <u>Address</u>, program execution will halt.

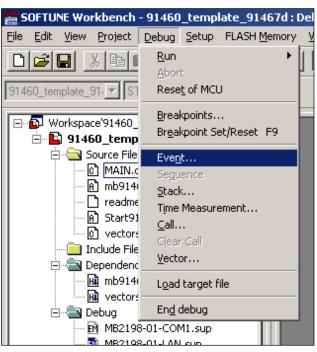
Further data can be specified with size option i.e. byte, half word and word.

Two data event can be set.



6.1.1 Code Event

As shown below, Event dialog box is opened by clicking on Event... option on Debug pull down menu.



In the below dialog box, *Event Address* - Specifies the address at which the event occurrence condition is to be set. *Address Mask* specifies address mask. Under *Mode*, two possible modes can be selected. In <u>OR Mode</u>, events are triggered when the event 1 condition or the event 2 condition is established. In *Sequential mode* Events are triggered when the event 1 condition and event 2 condition are established in sequence.

Event				×
Code Data				
Event <u>n</u> umber:	1		1	Append
<u>A</u> ddress:	H'00040302	No symbol		
Address <u>m</u> ask:	H'FFFFFFF0	Pa <u>s</u> s count:	D'10	
Mode				
⊙ <u>O</u> R mode	C Seguential mod	e		
Event list				
no.en/dis ad	dr mask pas	s cond	symbol	
	Enal	ole Disat	ble	Delete
			ок	Cancel



When *address mask* is selected, for each bit set in *address <u>mask</u>* field, corresponding bit from <u>Address</u> is compared for the exact match with the corresponding bit of address hold by Program Counter i.e. Program will stop executing due to event break whenever following condition is met...

Event			2
Code Data			
Event <u>n</u> umber:	1		Append
<u>A</u> ddress:	H'00040302	No symbol	
Address <u>m</u> ask:	H'FFFFFF0	Pa <u>s</u> s count:	D'1
Mode	Seguential mod	de	
	ldr mask pas	ss condis	ymb o l
l enable 0	0040302 FFFFFF0	1(0)	
	<u>Ena</u>	able <u>D</u> isable	2 Delete

(Address hold by PC) & (Address Mask) = (address) & (Address Mask)

For example, if we set Code Breakpoint at address H' 00040302 and address Mask H'FFFFFF0 and when program is executed it will stop at all the address (where there is valid instruction) from H' 00040300 to H' 0004030F

Similarly when we select *Pass count* (1 to 255), program will halt executing at <u>Address</u> when program counter executes instruction at the address selected in the <u>Address</u> field, as many numbers of times as that is selected by *Pass count*.



For example, if we set code break point at <u>Address H'000402AC</u> and select <u>Pass</u> count equal to D'5. When program is run, it halts due to hardware code break point when value of variable 'i' equal to D'5

12: vo	id wai	t (short.	int cnt)	
(X)000402A0:	1704		ST	R4,0-R15
X)000402A2:	1781		ST	RP,0-R15
X 000402A4:	0F02		ENTER	#008
13: {				
14:		int i;		
15:		PDR16 = 0	xFF;	
X)000402A6:	CFFO		LDI:8	#FF,RO
X 000402A8:	8B0D		MOV	R0,R13
X)000402AA:	1A10		DMOVB	R13,0010
16:		<pre>for(i=0;i</pre>	<cnt;i++);< td=""><td>;</td></cnt;i++);<>	;
X)000402AC:	C000		LDI:8	#00,R0
(X)000402AE:	3FF0		ST	R0,0(R14,-4)
↓ 000402B0:	4050		LDUH	@(R14,10),R0
ҢХ 000402В2:	97A0		EXTSH	R0
Ң∕О000402В4:	2FF1		LD	0(R14,-4),R1
ҢХ)000402В6:	AA01		CMP	R0,R1
Ң∕)000402В8:	EB04		BGE	00040202
↓ 000402BA:	2FF0		LD	0(R14,-4),R0
ҢХ)000402ВС:	A410		ADD	#1,R0
₩¥X0000402BE:	3FF0		ST	R0,0(R14,-4)
Ң∕∕000402C0:	EOF7		BRA	000402B0

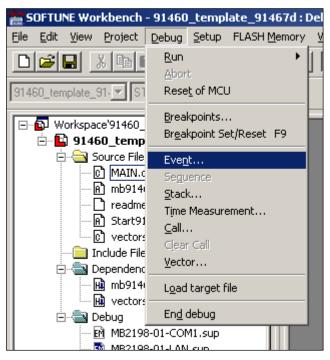
Event can be released by selecting it from *Event list,* and clicking on *Delete* button, alternately it can also be disabled by clicking *Disable* button

Event			×
Code Data			(
Event <u>n</u> umber:	1		Append
<u>A</u> ddress:	H'000402AC	No symbol	
Address <u>m</u> ask:	H'FFFFFFE	Pa <u>s</u> s count:	D'5
_ Mode			
⊙ <u>O</u> R mode	C Seguential mod	le	
- Event list			
	ddr mask pas 00402AC FFFFFFFE	s cond 5(5)	l symbol
T chapte o	00402AC 11111112	0(0)	
	<u>E</u> na	ble Disa	ble Delete
	<u></u>		
			OK Cancel



6.1.2 Data Event

As shown below, *Break* dialog box is opened by clicking on *breakpoints…* option on *Debug* pull down menu.



Event			×
Code Data			
E⊻ent number:			Append
<u>A</u> ddress:	H'00000000	No symbol	
Address <u>m</u> ask:	H'FFFFFFF		
<u>D</u> ata:		Sjze:	Byte 💌
Data mas <u>k</u> :	H'FF		🗖 Data <u>n</u> ot
Mode		Attribute	
⊙ <u>O</u> R mode	O Sequential n	node 🔽 <u>R</u> ea	ad 🔽 🔟rite
Event list	atus addr add	r msk data	data msk size cond
no.en/ dis st	acus addr add	I_MSR data	alor max size cond
	E	nable Disa	ble Dele <u>t</u> e
			OK Cancel

Click on Data tab,



Address mask and Pass count feature work same as that of code event. <u>Data</u> specifies the data to be set as the event occurrence condition. <u>Data mask</u> specifies data mask. <u>Data not</u> specifies the condition when the data values do not match. <u>Size</u> specifies a data access size. <u>Attribute</u> specifies a data access attribute.

In *Event* dialog box, under *attribute* option it can be specified if *read* or *write* access at <u>Address</u> should cause a break.

Event		×
Code Data		
E <u>v</u> ent number:		Append
<u>A</u> ddress:	H'00000000 No symbol	
Address <u>m</u> ask:	H'FFFFFFF	
<u>D</u> ata:	Sjze: By	te 💌
Data mas <u>k</u> :	H'FF 🗖	Data <u>n</u> ot
Mode	Attribute	
⊙ <u>O</u> R mode	○ Sequential mode ☑ Read	☑ Write
Event jist	atus addr addr_msk data data_	msk size cond
	Enable Disa <u>b</u> le	Deleţe
	OK	Cancel

Under size option it is possible to select either byte, halfword or word type data.

Event			×
Code Data			
Event number:			Agpend
<u>A</u> ddress:	H'00000000	No symbol	
Address <u>m</u> ask:	H'FFFFFFF		
<u>D</u> ata:		Sjze:	Byte 💌
Data mas <u>k</u> :	H'FF		🗖 Data <u>n</u> ot
Mode		Attribute	
⊙ <u>O</u> R mode	C Sequential n	node 🛛 🗹 <u>R</u> e	ad 🗹 🗹 rite
Event list	atus addr add	lr_msk data	data_msk size cond
	Ē	inable Dis	able Delețe
			OK Cancel

Further, comparison condition can be selected among Data equal to or Data not.



Note, that the Data and Data mask text boxes are only available if either Data agreement or Data not is selected.

When <u>Data</u> field is left empty, for all data access at <u>Address</u> for read or write access as selected by attribute, program execution will halt.

Event		X
Code Data		
E <u>v</u> ent number:	1	Agpend
<u>A</u> ddress:	H'000283E0	No symbol
Address <u>m</u> ask:	H'FFFFFFF	
<u>D</u> ata:		Size: Word 💌
Data mas <u>k</u> :	H'FFFFFFF	🗖 Data <u>n</u> ot
_ Mode		Attribute
	C Sequential mod	e 🔽 <u>R</u> ead 🔽 <u>W</u> rite
Event jist no.en/dis st	-	msk data data_msk size cond
l enable R/	W 000283E0 FFFFF	FFF word
	<u>E</u> nat	ble Disa <u>b</u> le Delete
		OK Cancel

For example, if we set breakpoint as shown in above figure, program execution will halt every time when there is a read or write access to location H'000283E0

When *Data <u>not</u>* check box is unselected, for all data access at <u>Address</u>, the content *read* from or *written* to, is equal to that specified in <u>Data</u> field, program execution will halt.

Event			×
Code Data			
E <u>v</u> ent number:	1 •		Append
Address:	H'000283E0	No symbol	
Address <u>m</u> ask:	H'FFFFFFF		
<u>D</u> ata:	H'0000005	Sjze:	Word
Data mas <u>k</u> :	H'FFFFFFF		🗖 Data <u>n</u> ot
Mode		Attribut	te
⊙ <u>O</u> R mode	C Sequential n	node 🔽 <u>R</u>	ead 🔽 🔟rite
Event list no.en/dis st 1 enable R/	W 000283E0		data_msk size cond 15 FFFFFFFF word
			OK Cancel
		_	



For example, if we set breakpoint as shown in above figure, program execution will halt every time when data H' 05 is read from or written to location H'000283E0.

Similarly if we select *Data <u>not</u>* check box, for all data access at <u>Address</u>, the content *read* from or *written* to, is not equal to that specified in <u>Data</u> field, program execution will halt.

Event			×
Code Data			
E <u>v</u> ent number:	1 •		Append
Address:	H'000283E0	No symbol	
Address <u>m</u> ask:	H'FFFFFFF		
<u>D</u> ata:	H'0000005	Sjze:	Word
Data mas <u>k</u> :	H'FFFFFFF		🔽 Data <u>n</u> ot
- Mode		Attribute	
● <u>O</u> R mode	○ <u>S</u> equential m	ode 🔽 🗹 <u>R</u> ea	ad 🔽 <u>W</u> rite
Event list			
no.en/dis st.		r_msk data	data_msk size cond
l enable R/	W 000283E0	!00000005	FFFFFFFF word
1			
	<u></u>	nable Disa	ble Delete
			OK Cancel

For example, if we set breakpoint as shown in above figure, program execution will halt every time when data other than H' 05 is read from or written to location H'000283E0.

When *Data mask* is selected, for each bit set in *Data mask* field, that bit from *Data* field is compared for the exact match with the corresponding bit of Data which is to be read or written at the break address by the MCU i.e. Program will stop executing due to break point whenever following condition is met...

(Data read by MCU at Break Address Selected) & (Data Mask) = (Data) & (Data Mask)

For example, if we set Data Breakpoint at address H' 000283E0, which is the address of a int type variable and if the attribute is set to read, comparison condition is set to Data agreement, data is set to H'000000F and Data Mask is set to H'FFFFFF3.





vent Code Data			2
Event number:	1		Agpend
<u>A</u> ddress:	H'000283E0	No symbol	
Address <u>m</u> ask:	H'FFFFFFF		
<u>D</u> ata:	H'0000000F	Sjze:	Word
Data mas <u>k</u> :	H'FFFFFF3		🔲 Data <u>n</u> ot
Mode © <u>O</u> R mode	C <u>S</u> equential m	ode	e ead 🔽 <u>W</u> rite
Event jist	tatus addr add	r msk data	data msk size cond
l enable B			F FFFFFFF3 word
	F	nable Di:	sa <u>b</u> le
	<u></u>		
			OK Cancel

When program is executed it will stop when data, either H'00000C or H'000000D or H'000000E or H'000000F is read from break address H' 000283E0.

Now let us consider that we want in a following code, we want program to stop executing when count = H'05 and i = H'FF. We can do this using sequential mode, while setting an event

```
void wait (short int cnt)
{
    int i;
PDR16 = 0xFF;
    for(i=0;i<cnt;i++);</pre>
    PDR16 = 0 \times 00;
}
void main(void)
{
    int count = 0;
    _EI();
                                  /* enable interrupts */
      _set_i1(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 = 0 \times 00;
    DDR16 = 0xFF;
    while(1)
                                  /* endless loop */
    {
       HWWD_CL = 0;
       count++;
__asm(" NOP");
       wait (5000);
__asm(" NOP");
    }
3
```



For this we need to set event 1 and 2 as described below, assuming *count* is located at H'000283F0 and *i* is located at H'000283E0

Event X	Event
Code Data	Code Data
Event number: 1	Event number: Append
Address: H'000283F0 No symbol	Address: H'000283E0 No symbol
Address mask:	Address mask: H'FFFFFFF
<u>D</u> ata: H'00000005 Sjze: Word ▼	Data: H'000000FF Size: Word 💌
Data mask: HFFFFFFF Data not	Data mas <u>k</u> : H'FFFFFFF Data <u>n</u> ot
Mode C □R mode Sequential mode	Mode C OR mode C OR mode Image: Sequential mode
Event jist no.en/dis status addr addr_msk data data_msk sise cond	Event list no.en/dis status addr addr_msk data data_msk sise cond
1 enable R/W 000283F0 00000005 FFFFFFF word 2 enable R/W 000283E0 0000000FF FFFFFFFF word	1 enable R/W 000283F0 00000005 FFFFFFF word 2 enable R/W 000283E0 0000000FF FFFFFFFF word
<u>Enable</u> Disa <u>b</u> le	
OK Cancel	OK Cancel

7. Trace



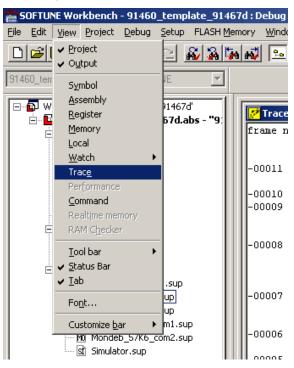
How To Use The Trace Buffer

Trace function displays addresses and instructions executed so far. While execution of a program, the address, data and status information can be sampled and stored in the trace buffer. This function is called real-time trace.

In-depth analysis of a program execution history can be performed using the data recorded by real-time trace.

7.1 Trace Window

Trace window can be opened by clicking Trace, on View pull down menu.





🦉 Trace					_ 🗆 ×
frame no.	address	mnemo	nio	c (-65535 00000)	
:	000402BC	ADD		#1,R0	_
:	000402BE	ST		R0,0(R14,-4)	
-00011 :	write 000	00748	at	000283E0	
:	000402C0	BRA		000402B0	
-00010 :	000402B0	LDUH		@(R14,10),R0	
-00009 :	read	1388	at	000283EE	
:	000402B2	EXTSH	I	RO	
:	000402B4	LD		0(R14,-4),R1	
-00008 :	read 000	00748	at	000283E0	
:	000402B6	CMP		RO,R1	
:	000402B8	BGE		000402C2	
:	000402BA	LD		0(R14,-4),R0	
-00007 :	read 000	00748	at	000283E0	
:	000402BC	ADD		#1,RO	
:	000402BE	ST		R0,0(R14,-4)	
-00006 :	write 000	00749	at	000283E0	
:	000402C0	BRA		000402B0	
-00005 :	000402B0	LDUH		0(R14,10),R0	
-00004 :	read	1388	at	000283EE	

In a Trace result display, the first column shows the frame number.

The second column shows the internal bus address.

The third column shows either the disassembled machine code (mnemonic) or a data transfer (inclusive data value).

Note that the last executed frame has the number 0 and all previous frames negative numbers.

Every time, after program run, to refresh a trace window right click on Trace Window, on a popup menu click on *Refresh*.

To clear previous data, right click on Trace Window, on a popup menu click on Clear.

🧭 Trace		
frame no.	address mnemonic (-65535 00000)	
:	000402BA LD @(R14,-4),R0	-
-04467 :	read 000003CC at 000283E0	
:	000402BC ADD #1,R0	
:	000402BE ST R0,0(R14,-4)	
-04466 :	write 000003CD at 000283E0	
:	000402C0 BRA 000402B0	
-04465 :	000402B0 LDUH 0(R14,10),R0	
-04464 :	read 1388 at 000283EE Refresh	
:	UUU4U2BZ EXTSH RU	
:	000402B4 LD @(R14,-4), Jump	
-04463 :	read 000003CD at 000283E0 Back trace	
:	000402B6 CMP R0,R1	
:	000402B8 BGE 000402C2	
:	000402BA LD @(R14,-4), Cycle	
-04462 :	read 000003CD at 000283E0 Source	
:	000402BC ADD #1,R0 Task	
:	000402BE ST R0,0(R14,-	
-04461 :	write 000003CE at 000283E0 Setup	
:	000402C0 BRA 000402B0 Detail	
-04460 :	000402B0 LDUH 0(R14,10), Find	
-04459 :	read 1388 at 000283EE Save	
:	000402B2 EXTSH R0	
:	000402B4 LD @(R14,-4 <mark>)</mark> , ^{Clear}	
-04458 :	read 000003CE at 000283E0 Close	
:	000402B6 CMP R0,R1	
:	000402B8 BGE 000402C2	Î
:	000402BA LD @(R14,-4),R0	
-04457 :	read 000003CE at 000283E0	
:	000402BC ADD #1,R0	
•		•

Trace



7.2 Trace View

Trace view can be changed among Instruction view, raw data view or source view.

For Instruction view, right click on Trace window, on a popup menu click on Instruction.

Instruction view show instruction cycles using assembler mnemonics

🚰 Trace		_ 🗆 🗵
frame no.		•
:	000402BA LD @(R14,-4),R0	_
-04467 :	read 000003CC at 000283E0	
:	000402BC ADD #1,R0	
	000402BE ST R0,0(R14,-4)	
-04466 :	write 000003CD at 000283E0	
:	000402C0 BRA 000402B0	
-04465 :	000402B0 LDUH @(R14,10),R0	
-04464 :	read 1388 at 000283EE Refresh	
	000402B2 EXTSH R0	
:	000402B4 LD 0(R14,-4), Jump	
-04463 :	read 000003CD at 000283E0 Back trace	
	000402B6 CMP R0,R1	
:	000402B8 BGE 000402C2	
:	000402DA LD ((K14,-4))	
-04462 :	read 000003CD at 000283E0 Source	
	000402BC ADD #1,R0 Task	
:	000402BE ST R0,0(R14,-	
-04461 :	WILE DUDUGCE at DUDZ63ED	
:	000402C0 BRA 000402B0 Detail	
-04460 :	000402B0 LDUH @(R14,10), Find	
-04459 :	read 1388 at 000283EE Save	
:	000402B2 EXTSH RO	
:	000402B4 LD 0(R14,-4), Clear	
-04458 :	read 000003CE at 000283E0 Close	
:	000402B6 CMP R0,R1	
:	000402B8 BGE 000402C2	
:	000402BA LD @(R14,-4),R0	
-04457 :	read 000003CE at 000283E0	
	000402BC ADD #1,R0	
•		

For raw data view, right click on Trace window, on a popup menu click on Cycle.

frame no.	address data (-65535 00000)		
00028 :	read 00000744 at 000283E0		
-00027 :	read 00000744 at 000283E0		
-00026 :	write 00000745 at 000283E0		
-00025 :	000402B0 LDUH @(R14,10),R0		
-00024 :	read 1388 at 000283EE		
-00023 :	read 00000745 at 000283E0		
-00022 :	read 00000745 at 000283E0		
-00021 :	write 00000746 at 000283E0		
-00020 :	000402B0 LDUH @(R14,10),R0		
-00019 :	read 1388 at 000283EE		
-00018 :	read 00000746 at 000283E0		
-00017 :	read 00000746 at 000283E0		
-00016 :	write 00000747 at 000283E0	Refresh	
-00015 :	000402B0 LDUH @(R14,10),R0		
-00014 :	read 1388 at 000283EE	Jump	
-00013 :	read 00000747 at 000283E0	Back trace	
-00012 :	read 00000747 at 000283E0	Instruction	
-00011 :	write 00000748 at 000283E0	✓ Cycle	
-00010 :	000402B0 LDUH 0(R14,10),R0		
-00009 :	read 1388 at 000283EE	Source	
-00008 :	read 00000748 at 000283E0	Task	
-00007 :	read 00000748 at 000283E0	Setup	
-00006 :	write 00000749 at 000283E0		
-00005 :	000402B0 LDUH @(R14,10),R0	Detail	
-00004 :	read 1388 at 000283EE	Find	
-00003 :	read 00000749 at 000283E0	Save	
-00002 :	read 00000749 at 000283E0	Clear	
-00001 :	write 0000074A at 000283E0		
00000 :	000402B0 LDUH @(R14,10),R0	Close	

For Source view, right click on Trace window, on a popup menu click on *Source*. Source view shows program source code (e.g. C - language)

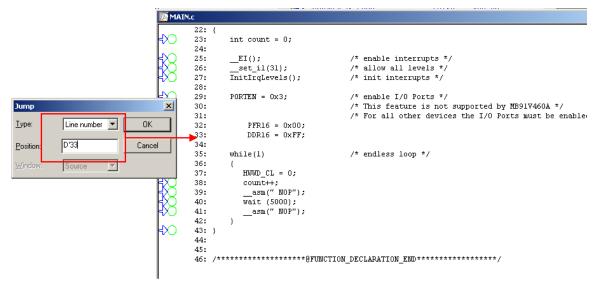


7.3 Trace Jump

To move cursor to particular frame in trace window, right click on *trace* window, click on *Jump...*, on *Jump* dialog box, select *frame* under *Type* option and desired frame number under *Position* option and click *OK*

			🚰 Trace			
			frame no.	address	mnemonic (-65535 00000)
			-01234 :	read	1388 at 000283EE	
		×	• •	000402B2	EXTSH RO	
			:	000402B4	LD @(R14,-4),R1	
			-01233 :	read 000	005DF at 000283E0	
			:	000402B6	CMP RO,R1	
Jump		×	:	000402B8	BGE 000402C2	
-			:	000402BA	LD @(R14,-4),RO	
<u>T</u> ype:	Frame 💌	ОК	-01232 :	read 000	005DF at 000283E0	
			:	000402BC	ADD #1,R0	
Position:	-D'1234	Cancel	:	000402BE	ST R0,0(R14,-4)	
			-01231 :	write 000	005E0 at 000283E0	
<u>₩</u> indow:	Source 🔻		:	000402C0	BRA 000402B0	
	· — —		-01230 :	000402B0	LDUH @(R14,10),R0	
			-01229 :	read	1388 at 000283EE	
			:	000402B2	EXTSH RO	
			:	000402B4	LD @(R14,-4),R1	
			-01228 :	read 000	005E0 at 000283E0	
			:	000402B6	CMP R0,R1	
			:	000402B8	BGE 000402C2	
			:	000402BA	LD @(R14,-4),RO	-
			•			

To view source code at particular line number, right click on trace window, click on *Jump...*, on *Jump* dialog box, select *Line number* under *Type* option and source line number under *Position* option, click *OK*, in editor window one can view source code at selected line number





To view assembly instruction at particular memory location, right click on trace window, click on *Jump...*, on *Jump* dialog box select *Address* under *Type* option and memory location under *Position* option and than select either Source or Assembly under <u>Window</u> option.

MATH

			MAIN.c				
		ſ	27:	InitIrqLevels	();	<pre>/* init interrupt:</pre>	s */
				9F8C00040000	LDI:32	#00040000,R12	
Jump		×	₹X)000402E2:	9710	CALL	@R12	
			28:				
<u>Type:</u>	Address 🔽 🛛 🔿	к 📔 🛛	29:	PORTEN = 0x3;		/* enable I/O Port	ts *
			-XQ000402E4:		LDI:8	#03,R0	
Position:	H'000402EE Car	ncel 📔 🥤		9F8C00000498	LDI:32	#00000498,R12	
-		P		16C0	STB	R0,0R12	
Window:	Source 🔻		30:			/* This feature i:	
	,		31:			/* For all other (devi
			32: -X_000402EE:	PFR16 = 0>		#00 B0	
				9F8C00000D90	LDI:8	#00,R0	
			0000402F0:		LDI:32 STB	#00000D90,R12 R0,@R12	
				1000	210	RU, BRIZ	
		r					
		r	Assembly				
		r I		C000	LDI:8		
			Assembly	C000 9 F 8C00000D90	LDI:8 LDI:32	#00,R0 #0000D90,R12	
		1 	Assembly	9F8C00000D90		#00,R0	
Jump			Assembly 0000402EE: 0000402F0:	9F8C00000D90 16C0	LDI:32	#00,R0 #00000D90,R12	
			Assembly 0000402EE: 000402F0: 000402F6: 000402F6:	9F8C00000D90 16C0	LDI:32 STB	#00,R0 #00000D90,R12 R0,@R12	
Jump Type:	Address 💌 🖸		Assembly 0000402EE: 0000402F0: 0000402F6: 0000402F8: 0000402FA: 000040300:	9F8C00000D90 16C0 CFF0 9F8C00000D50 16C0	LDI:32 STB LDI:8 LDI:32 STB	#00,R0 #00000D90,R12 R0,@R12 #FF,R0 #00000D50,R12 R0,@R12	
<u>T</u> ype:		IK c	Assembly 0000402EE: 0000402F0: 0000402F6: 0000402F8: 0000402F8: 000040300: 000040300:	9F8C00000D90 16C0 CFF0 9F8C00000D50 16C0 9F80000004C7	LDI:32 STB LDI:8 LDI:32 STB LDI:32	#00,R0 #00000D90,R12 R0,@R12 #FF,R0 #00000D50,R12 R0,@R12 #000004C7,R0	
			Assembly 0000402EE: 000402F6: 000402F8: 000402F8: 000402F8: 000040300: 00040300: 00040302: 00040302:	9F8C00000D90 16C0 CFF0 9F8C00000D50 16C0 9F80000004C7 8070	LDI:32 STB LDI:8 LDI:32 STB LDI:32 BANDL	#00,R0 #0000090,R12 R0,@R12 #FF,R0 #00000D50,R12 R0,@R12 #000004C7,R0 #7,@R0	
<u>I</u> ype: <u>P</u> osition:	H'000402EE Car	IK c	Assembly 000402EE: 000402F0: 000402F6: 000402F8: 000402FA: 00040300: 00040302: 00040308: 00040308:	9F8C00000D90 16C0 CFF0 9F8C00000D50 16C0 9F80000004C7 8070 2FF0	LDI:32 STB LDI:8 LDI:32 STB LDI:32 BANDL LD	#00,R0 #0000090,R12 R0,@R12 #FF,R0 #00000050,R12 R0,@R12 #000004C7,R0 #7,@R0 @(R14,-4),R0	
<u>T</u> ype:		IK c	Assembly 000402EE: 000402F6: 000402F8: 000402F8: 000402F4: 00040300: 00040302: 00040308: 00040308: 00040308:	9F8C00000D90 16C0 CFF0 9F8C00000D50 16C0 9F80000004C7 8070 2FF0 A410	LDI:32 STB LDI:8 LDI:32 STB LDI:32 BANDL LD ADD	#00,R0 #0000D90,R12 R0,@R12 #FF,R0 #00000D50,R12 R0,@R12 #000004C7,R0 #7,@R0 @(R14,-4),R0 #1,R0	
<u>I</u> ype: <u>P</u> osition:	H'000402EE Car	IK c	Assembly 000402EE: 000402F0: 000402F6: 000402F8: 00040300: 00040300: 00040302: 00040308: 00040302: 00040308: 00040306:	9F8C00000D90 16C0 CFF0 9F8C00000D50 16C0 9F80000004C7 8070 2FF0 A410 3FF0	LDI: 32 STB LDI: 8 LDI: 32 STB LDI: 32 BANDL LD ADD ST	#00,R0 #0000090,R12 R0,@R12 #FF,R0 #00000050,R12 R0,@R12 #000004C7,R0 #7,@R0 @(R14,-4),R0	
<u>I</u> ype: <u>P</u> osition:	H'000402EE Car	IK c	Assembly 000402EE: 000402F6: 000402F6: 000402F8: 00040300: 00040300: 00040308: 00040308: 00040308: 00040306: 000400306: 00040306: 00040306: 00040306: 00040306: 00040306: 00040306: 00040306: 00040306: 00040306: 00040306: 00040306: 00040306: 00040306: 00040306: 00040306: 000405: 0005: 0005: 0005: 0005: 0005: 0005: 0005: 0005: 0005:	9F8C00000D90 16C0 CFF0 9F8C00000D50 16C0 9F80000004C7 8070 2FF0 3FF0 9FA0	LDI: 32 STB LDI: 8 LDI: 32 STB LDI: 32 BANDL LD ADD ST NOP	#00,R0 #00000D90,R12 R0,@R12 #FF,R0 #00000D50,R12 R0,@R12 #000004C7,R0 #7,@R0 @(R14,-4),R0 #1,R0 R0,@(R14,-4)	
<u>I</u> ype: <u>P</u> osition:	H'000402EE Car	IK c	Assembly 000402EE: 000402F0: 000402F6: 000402F8: 00040300: 00040300: 00040302: 00040308: 00040302: 00040308: 00040306:	9F8C00000D90 16C0 CFF0 9F8C00000D50 16C0 9F80000004C7 8070 2FF0 3FF0 9FA0	LDI: 32 STB LDI: 8 LDI: 32 STB LDI: 32 BANDL LD ADD ST	#00,R0 #0000D90,R12 R0,@R12 #FF,R0 #00000D50,R12 R0,@R12 #000004C7,R0 #7,@R0 @(R14,-4),R0 #1,R0	

To view data at particular memory location, right click on trace window, click on *Jump*..., on *Jump* dialog box select *Address* under <u>Type</u> option and memory location under <u>Position</u> option and than select memory under <u>window</u> option.

Jump		×	
<u>T</u> ype:	Address 💌	OK	
Position:	H'000402B2	Cancel	
Window:	Memory		

Memory								
Address	+0	+2	+4	+6	+8	+A	Ascii	
000402B2	97A0	2FF1	AA01	EB04	2FF0	A410	//	-
000402BE	3FF0	E0F7	C000	8B0D	1A10	9F90	?	
000402CA	0781	A301	9720	1781	0F02	C000		
000402D6	3FF0	9310	871F	9F8C	0004	0000	?	
000402E2	971C	C030	9 F 8C	0000	0498	16CO	0	
000402EE	C000	9 F 8C	0000	0D90	16CO	CFFO		
000402FA	9 F 8C	0000	0D50	16CO	9F80	0000	P	
00040306	0407	8070	2FF0	A410	3FF0	9FA0	p/?	
00040312	9B04	1388	D7C4	9FAO	E0F3	9F90		
0004031E	0781	9720	A410	3FF0	9 F 80	0000	?	
0004032A	0407	8070	9FA0	9B04	1388	D7B5	p	
00040336	9FAO	EOF3	9 F 90	0781	9720	4480	D.	
00040342	5109	001A	29E8	1CE1	0700	5284	Q)R.	
0004034E	4A04	0082	2800	0075	0C3B	0008	J(u.;	
0004035A	9141	42D8	0601	0083	2430	2A41	.AB\$0*A	· _



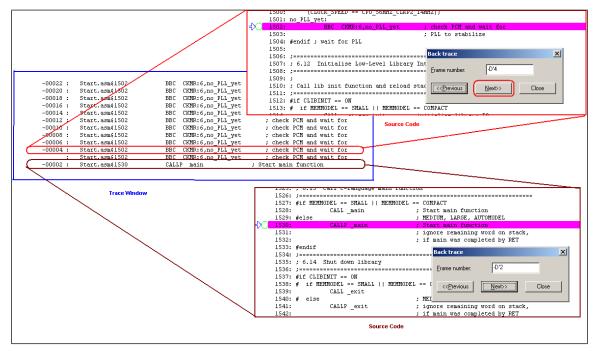
7.4 Back Trace

When selected *Source view*, one can use *Back trace*... functionality. Right click on editor window, click on *Back trace*..., *Back trace* dialog box will appear



With this functionality, one can traverse back and forth in source window in a same sequence, in which source code is executed. Source code at particular frame number will be highlighted by pink background while traversing.

For example, in figure below, trace window shows address and instruction at frame number -D'2 and -D'4 executed in sequence. In a *back trace* dialog box, if we select *frame number* -D'4, corresponding line at D'1502 in source code will be highlighted, if we click next, source code line number D'1530 will be highlighted which corresponds to *frame number* -D'2.





7.5 Search Trace

To find particular memory location in trace window, one can use *find* feature. Right click on Trace window, on a popup menu click on *Find*. Search trace dialog box will appear.

Search trace			×
Address			
Search <u>a</u> ddress:	H'000403	Find <u>n</u> ext	
Address <u>m</u> ask:	H'FFFFFF	Close	
Searching start number:	-D'2274		

The line having information about searched memory location is highlighted with black background.

-01012		WIICE	SOAA BI	, UUZSSA "	T
-01811	:	write	55AA at	: 00253C *	1
-01810	:	write	55AA at	00253E *	1
-01809	:	read	BO at	: 000403 *	9
-01808	:	F800A2	BBC	0403:6,STARTUP\no_PLL_yet*	1
-01807	:	read	BO at	000403 *	9
-01806	:	F800A2	BBC	0403:6,STARTUP\no_PLL_yet*	1
-01805	:	read	BO at	* 000403 *	10
-01804	:	F800A2	BBC	0403:6,STARTUP\no_PLL_yet*	1
-01803	:	read	BO at	* 000403 *	9
-01802	:	F800A2	BBC	0403:6,STARTUP\no_PLL_yet*	1
-01801	:	read	BO at	* 000403 *	10
-01800	:	F800A2	BBC	0403:6,STARTUP\no_PLL_yet*	1
-01799	:	read	BO at	000403 *	9
-01798	:	F800A2	BBC	0403:6,STARTUP\no_PLL_yet*	1
-01797	:	read	B0 at	* 000403 *	10

The required memory location is entered in Search address,

Search trace			×
Address			
Search <u>a</u> ddress:	H'000403	Find <u>n</u> ext	
Address <u>m</u> ask:	H'FFFFF	Close	
Searching start number:	-D'2274		

The frame number from where to start the trace search is entered in Searching start number

iearch trace			×
Address			_,
Search <u>a</u> ddress:	H'000403	Find <u>n</u> ext	
Address <u>m</u> ask:	H'FFFFFF	Close	
Searching start number:	-D'2274)	



When Address <u>mask</u> is selected, for each bit set in Address <u>mask</u> field, that bit from Search <u>a</u>ddress field is compared for the exact match with the corresponding bit of the memory location read by the debugger from trace data buffer. I.e. The line having the frame number for which following condition is met, will be highlighted in Trace window

(Memory location read by debugger) & (Address Mask) =

(Search address) & (Address Mask)

For the trace search, configured as shown in the figure below,

Search trace			×
Address			
Search <u>a</u> ddress:	H'F800A2	Find <u>n</u> ext	
Address <u>m</u> ask:	H'FFFFF0	Close	
Searching start number:	-D'1783		

When every time *Find next* button is clicked, trace frame, containing memory address in the range of H'F800A0 to H'F800AF is highlighted. When search reaches the first frame number (-D'1) it rollbacks and search continues from last frame (-D'65535).

7.6 Trace Setup

Trace functionality can be configured by using Setup Trace dialog box.

Right click on trace window, on pop up menu, click on Setup...

Setup tr	ace		×
Trace	Data trace range		
_ Stati	us		
•	Enable	⊂ <u>D</u> isable	
⊢ Buffe	er full break		
0	<u>B</u> reak	● <u>N</u> ot break	
	e sampling mode		
•	Trace <u>f</u> ull mode	C Trace trigger mode	
		ОКС	ancel

Trace data buffering can be enabled or disabled by selecting *Enable* or *Disable* radio button, under *Status* option.

C Status		
Enable	O <u>D</u> isable	



Trace data is buffered in the trace buffer. The trace buffer becomes full some time during debugging because its size is finite. When the trace buffer becomes full, the program being executed can be stopped. Trace buffer full break is set by selecting *Break* radio button under *Buffer full break* option.

MB2198-01 Emulator has 64K frames internal trace buffer and 256M frames External trace buffer.

Buffer full break	
C <u>B</u> reak	Not break

When program is halted by trace data buffer full break, corresponding line will be highlighted with yellow back ground.

50FTUNE Workbench - 91460_template_914		X
Ele Edit Yew Project Debug Setup FLA:		×
	AN LEE BOOOS SOON SOON SOON SOON	
91460_template_91.		
Image: Signal of the	<pre></pre>	- -
4	V MAINLE Memory D StarS1450 am & Watch1 & Local	۲ ۲
, Break at 000402BA by unknown break factor		DEBUG MB91F467D EML Break IP=000402BA



7.7 Saving Trace Data

Since Softune version V30L34R05 it is possible to save the trace data beginning with a starting frame and ending with an end frame. Please enter negative numbers due to negative trace frames.

Save As	×
Save jn: 🗀 10Port 💽 🖛 🗈 📸 🎫	
ABS 🔋 test.log	
C LST	
OBJ	
C OPT	
Prc Control Co	
C Src	
File name: test.log Save	
Save as type: Logging(*.log) Cancel	
Start frame: -100	
End frame: -1	

The save dialog is available by right-clicking in the trace window and choosing Save file

8. Time Measurement



How to Use Time Measurement Function

The Time Measurement is the simplest method of measurement. A 64-bit counter is used for this measurement resulting maximum cycle count of 18446744073709551615.

In this method two breakpoints have to be set. After the program has run from breakpoint 1 to 2 the Time Measurement will show the difference between the breakpoints.

Measurement time	X	Time from Reset to 1 st Breakpoint
From Initialize: 0h00m00s452ms986us725ns[Time]	<u>C</u> lear	Time from 1 st Breakpoint to
From Last Executed: 0h00m00s152ms198us075ns[Time] From Initialize: 929982[Cycle]	Close	2 nd Breakpoint
(929981 - 929999) From Last Executed: 328302[Cycle] (328301 - 328319)	Comment	Number of cycles from Reset to 1 st Breakpoint
(320301 - 320313)		Number of cycles from 1 st Breakpoint to 2 nd Breakpoint

Now, take a look at example as shown below, in a *main()* function, two breakpoints are set. In *a wait()* function *PORT00* is set to high when entered in function and then set to low when left the function.

<pre>23: void main(void) 24: { 25: 26: DDR00 = 0xFF; 27:asm(" NOP"); 28: wait (5000); 29:asm(" NOP"); 30: 31: } 32: 33: 34: 35: void wait (int cnt) 36: { 37: int i ; 38: PDR00 = 0xFF; 39: for (i = 0; i < cnt ; i++);</pre>		
25: 26: DDR00 = 0xFF; 27:am(" NOP"); 28: wait (5000); 29:asm(" NOP"); 30: 31: } 32: 33: 34: 35: void wait (int cnt) 36: { 37: int i ; 38: PDR00 = 0xFF;	23:	
<pre>26: DDR00 = 0xFF; 27:asm(" NOP"); 28: wait (5000); 29:asm(" NOP"); 30: 31: } 32: 33: 34: 35: void wait (int cnt) 36: { 37: int i ; 38: PDR00 = 0xFF;</pre>	24:	∞
<pre>27:asm(" NOP"); 28: wait (5000); 29:asm(" NOP"); 30: 31: } 32: 33: 34: 35: void wait (int cnt) 36: (37: int i ; 38: PDR00 = 0xFF;</pre>	25:	_
<pre>28: wait (5000); 29:asm(" NOP"); 30: 31: } 32: 33: 34: 35: void wait (int cnt) 36: { 37: int i ; 38: PDR00 = 0xFF;</pre>	26:	\sim
<pre>28: wait (5000); 29:asm(" NOP"); 30: 31: } 32: 33: 34: 35: void wait (int cnt) 36: { 37: int i ; 38: PDR00 = 0xFF;</pre>	27:	×io –
<pre> 29:asm(" NOP"); 30: 31:) 32: 33: 34: 35: void wait (int cnt) 36: (37: int i ; 38: PDR00 = 0xFF; </pre>	28:	87 -
30: 30: 31:) 32: 33: 34: 35: void wait (int cnt) 36: (37: int i ; 38: PDR00 = 0xFF;		× 1
<pre>> 31: } 32: 32: 33: 34: 35: void wait (int cnt) 36: { 37: int i ; 38: PDR00 = 0xFF;</pre>		
32: 33: 34: 35: void wait (int cnt) 36: { 37: int i ; 38: PDR00 = 0xFF;		\sim
33: 34: 35: void wait (int cnt) → 36: { 37: int i ; → 38: PDR00 = 0xFF;		\sim
34: 35: void wait (int cnt) 36: { 37: int i ; 38: PDR00 = 0xFF;		
35: void wait (int cnt) 36: (37: int i ; 38: PDR00 = 0xFF;		
X 36: { 37: int i ; X 38: PDR00 = 0xFF;		
37: int i ; X 38: PDR00 = 0xFF;		\sim
XQ 38: PDR00 = 0xFF;		~~
		Σ
Y		87
X 40: PDR00 = 0x00;		87 -
41:		~
42:		
X 43: }		\sim
···· ,		\sim



On Debug pull down menu click on *Time Measurement...*, Click on Clear if there are any time entries not equal to zero, Click on Close then.

Measurement time	×
From Initialize: 0h00m00s452ms986us725ns[Time] From Last Executed: 0h00m00s152ms198us075ns[Time]	<u>C</u> lear
From Last Executed, undomous sens sousor ons (ime) From Initialize: 929982[Cycle] (929981 - 929999)	Close
From Last Executed: 328302[Cycle] (328301 - 328319)	Comment

Run the program. The MCU will stop at the beginning of the main function (1st breakpoint). This first stop initializes the time measurement counter. Next, run the program till the second breakpoint.

Now on Debug pull down menu click on Time Measurement..., one can find the following information

Measurement time		×
	m00s002ms345us250ns[Time] 0h00m00s002ms344us925ns[Time]	<u>Clear</u>
From Initialize: (150121 - 150139)	150122[Cycle]	Close
From Last Executed: (150101 - 150119)	150102[Cycle]	Comment

9. Call Stack



How to Use Call Stack Feature

Usually, a program is a set of several subroutines. For this reason, as debugging advances, function calls of several stages occur. For example, one routine calls another and the called routine further calls another.

Call stack window can be opened by clicking on Stack under Debug menu

Call stack	×
Eunction name:	Jump
(wait(1388) (main()	Cancel
	Parameter

The call stack retains the relationship between function calls. Clicking a function name from the function name list immediately displays information for the function in the Source Window.

The function written in the lowermost line of the function name list is the main function. This main function calls the function above it. The called function further calls a function above it. In this way, the function written in the uppermost line is the function in which the current PC exists. When return is executed, functions are deleted in turn from the function name list, starting from the uppermost line.

When a check mark is set to *Parameter*, an argument value is displayed after each function name, as shown in above figure. When no check mark is set, only parentheses "()" are displayed after each function name.

10. Function Call



How to Use Function Call Feature

The specified function can be started during debugging without reference to the flow of the program. This function is known as function call. This feature can be used when execution is stopped. After MCU reset one should at least execute one instruction before using this feature. The *Function call* dialog box is opened by right clicking on *Call*... on Debug pull down menu

Function call	×
Eunction	ОК
Display return value	Cancel

When the function call dialog box opens, specify the function you want to call with a correct argument. If a breakpoint is set in the called function, the program stops at this breakpoint. When processing of the called function is terminated and control is returned, the function call result dialog box opens, if *Display return value check box* is checked. The PC then returns to the value before the function was called.

X
OK

To understand the function, let us consider following example code

	23:	void main(void)
Đ –	24:	{
	25:	unsigned char a = 5;
Đ –	26:	unsigned char b = 6;
	27:	unsigned int result;
	28:	
Đ –	29:	result = multiply(a,b);
	30:	
Đ –	31:	}
	32:	
	33:	
	34:	
	35:	int multiply (char varl, char var2)
÷۲O	36:	{
	37:	inti ;
∞	38:	i = varl * var2;
÷۲O	39:	return i;
	40:	
×X	41:	}



Let us set break point at line no. 29 and line no. 38. Start executing program, at first it halts at first break point. Now open *Function Call* dialog box (Debug -> Call...)

When the function definition is, int multiply (char var1, char var2), specify the function call as follow

- multiply (5, 6); Where a constant value is directly specified
- multiply (a, b); Where variable 'a' & 'b' is directly specified

	23: vo 24: { 25: 26: 27: 28:	old maln(vold) unsigned char a = 5; unsigned char b = 6; unsigned int result;				
->8	29:	result = multiply(a,b);				
÷∞ +∞	36: { 37:	nt multiply (char varl, char var2) int i ;	Functio	_	iltiply(a,b) ralue	Cancel
IX.	38: 39:	i = varl * var2;	F	unction ca	all	X
 ‡88 ≠∞	39: 40: 41: }	return i;	- 1	<u>F</u> unction	multiply(5,6)	ОК
L				🔽 Display	<u>r</u> etum value	Cancel

Since break point is set in the called function, the program stops at this breakpoint. When run again at the end of processing of called function, control is returned, the *Result of call function* dialog box as shown below opens. The PC then returns to the value before the function was called.

🔯 call.c		
**** ****	<pre>19: 20: int multiply (char varl, char v 21: 22: 23: void main(void) 24: (25: unsigned char a = 5; 26: unsigned char b = 6; 27: unsigned int result;</pre>	
	28:	*
-}8	<pre>29: result = multiply(a,b);</pre>	
€>O €>O €>O €>O	<pre>30: 31: 32: 33: 34: 35: int multiply (char var1, char v 36: (37: int i ; 38: i = var1 * var2; 39: return i; 40: 41:)</pre>	Result of call function Function: multiply OK

11. Vector



How to Use Vector Feature

An interrupt vector is the memory address of an interrupt handler, or an index into an array called an interrupt vector table or dispatch table. Interrupt vector tables contain the memory addresses of interrupt handlers. When an interrupt is generated, the processor saves its execution state via a context switch, and begins execution of the interrupt handler at the interrupt vector.

11.1 Display and setting vectors

11.1.1 Display

To display Interrupt vector table, click on Vector... at Debug pull down menu

No.	Address	Symbol	Factor 🔺	Start address of
0	0000BFF8	0000BFF8	Ieset	program is usuall
1	05000000	05000000	System reserv	set in reset vecto
2	00000000	00000000	System reserv	L
3	00000000	00000000	System reserv	
4	00000000	00000000	System reserv	
5	00000000	00000000	System reserv	
б	00000000	00000000	System reserv	
7	00000000	00000000	System reserv	
8	00000000	00000000	System reserv	
9	00000000	00000000	System reserv	
10	00000000	00000000	System reserv	
11	00000000	00000000	System reserv 📕	
12	00000000	00000000	Saugets and the same second	



11.1.2 Setting an address

Change the address set in a vector in the following procedure:

Select a vector table number and then click the *Edit* button. The vector edit dialog box shown below opens.

No.	Address	Symbol	Factor 🔺
L2	00000000	00000000	System re:
L3	00000000	00000000	System re
14	00000000	00000000	undefined
L5	00040280	\Default	IMI
Lő	00040280	\Default	external
17	00040280	\Default	external
L8	00040280	\Default	external
19	00040280	\Default	external
20	00040280	\Default	external
1	00040280	\Default	external
22	00040280	\Default	external
23	00040280	\Default	external
í –	00040380	17-41-	Edit vector
<u>C</u> opy	y <u>E</u> dit		Close Vector number: D'16 OK
			Factor : external interrupt #0 Cancel

Set an address and then click the [OK] button.



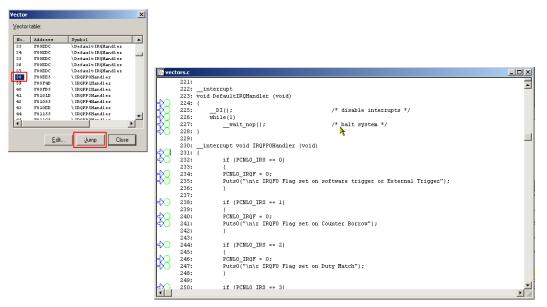
Vector

11.2 Jump

Display the source of the program stored at the address set in the vector table in the following procedure:

1. Select a vector number.

Click the [Jump] button.



If the starting address of the program set in the vector table is incorrect, the source cannot be displayed (disassemble display).

The jump function merely displays the jump destination program; it does not update the program counter to move control to the address set in the vector table.

12. Debug Environment Setup Procedure



12.1 Execution

Setup debug e	nvironment			×
	Error output mory emulation	Frequency	Inacces	
Step mode-	· · ·	ine O <u>h</u>]
Watchdog C <u>E</u> nabl				
_	k point while running- le O Djsable			
			IK	Cancel

Step mode

Automatic

Automatically sets the step unit according to the window display state. If it is a source code view, step will pass through each source line no. If it is a Mix mode code view, step will pass through disassemble source line no.

Source Line

Executes the step in units of source lines.

Machine Language

Executes the step in units of machine languages.

Watchdog

Specify whether to enable or disable the watchdog timer at program execution.

Setting break point while running

If "Setting break point while running" is enabled, it is also possible to break settings even when executing a user program.



12.2 Watch

Setup debug environment			×
Tab Error output External memory emulation Execution Watch Radix E	Frequency	Inaccessi	
- Watch mode	nguage 🔿 /	\ <u>s</u> sembler	
<u>D</u> ata size: Byte	•		
Memory buffering			
● <u>E</u> nable ○ Disa	bļe		
Specified number of array elen	nent		
☑ E <u>n</u> able Elemen <u>t</u> :	D'256		
		IK	Cancel

Watch Mode

Automatic

Sets the watch mode automatically according to the analysis result. Refer 4.4 & 4.3

C Language

Sets the C language mode (interpretation as C language expressions).

- Refer 4.4 & 4.3
- Assembler

Sets the assembler mode (interpretation as assembler expressions).

Refer 4.4 & 4.3

Data Size

Sets the display size in the assembler mode to either byte, word, long, single or double

Memory Buffering

Enable

In case of variables as arrays or structures, memory of whole variables is read.

They are accessed by size of the top variable.

Disable

In case of variables as arrays or structures, the memory of each variable is read.

Specified number of array element

Enable

Debugger displays a warning dialogue in case of bigger array element than the number of array-element that you limited, when you register or expand an array with a watch variable.



element

One can specify number (a default is D'256) of array element. The default of this control is "Enable".

12.3 Radix

Setup debug enviro	onment			×
Tab External memory Execution Watch		Access s Frequency ation Break	Inacces	
	© <u>D</u> ecimal	C <u>O</u> ctal	O <u>B</u> ir	hary
Display		0 <u>N</u> o display		
		0	ĸ	Cancel

Radix

Sets the default base number for numerical value.

Display Source Line

Switches source line display to source line non display or vice versa for Trace- Instruction view.

12.4 Emulation

Setup debug environment			×
Execution Watch Radix Emulation Memory verify operation © Enable © Disable TRIG input	Access s quency n Break Specify ins C Engble	Inacce Monitorir truction ce	ache size
Enable Disable MCU mode Internal trace External trace Enable Enable Enable			
	0	К	Cancel



Memory Verify Operation

Specifies whether to verify memory when data is written to memory.

TRG Input

Specifies whether to enable or disable TRIG pin input. For more information please refer MB2198_01 Hardware manual (MB2198-01-CM71-00413-2E).

MCU Mode

Specifies an MCU operation mode (internal trace, external trace, full trace mode or real-time mode).

Specify instruction cache size

Sets whether to automatically flush instruction cache.

12.5 Breakpoint

During program execution a break point can be set if the following set up is done:

etup debug en	vironment				
Tab	Error output	Access	size	Load	
External memory emulation		Frequency	Inacc	ccessible area	
Execution Wa	tch Radix En	nulation Break	Monitori	ng Directory	
🗖 Default break	point				
• Software	C <u>H</u> ardware				
Alignment erro	or break				
Code	🔲 Da <u>t</u> a				
			ок 📘	Cancel	

Default break point

Specifies the default type of the code breakpoint.

Alignment error break

Specifies whether to suspend MCU execution when an alignment error occurs



12.6 Monitoring

etup	debug envira	nment			
	ectory Tab ep execution			Freque Emulation	ncy Break Monitoring
	ontrol window Memory windo		_ <u>W</u> atch wi	indow	
	ontrol sampling– Sampling <u>T</u> ime:	D'1000	ms		
				ОК	Cancel

Control Window

Memory Window

Specifies whether to monitor the Memory Window.

Watch Window

Specifies whether to monitor the Watch Window.

Object Window

Specifies whether to monitor the Object Window.

Control Sampling

Specifies sampling time for Watch window, Memory window and Object window



12.7 Directory

Setup debug environment	×
Step execution Watch Radix Emulation Directory Tab Error output Load Freque	Monitoring ency Break
Display path information: Source file search path	Append
Append path:	Delete Browse
Directory:	
OK	Cancel

Display path information

Specifies the path information to be displayed.

Append path

Sets the path to be added.

Directory

Displays the currently set directory.

12.8 Tab

Setu	ıp debug e	environment			×
		Vatch Radix Emul			
		mory emulation	Frequency	Inac	cessible area
	Tab	Error output	Access s	ize	Load
I	ab:	D'8			
			0	K	Cancel



Tab

Specifies the Tab. (D'4/D'8)

12.9 Error output

Setup debug e	nvironment		x
External mer Tab In GUI opera Dialog	nory emulation Fr Error output ation	C O <u>n</u> ly error	-
🔲 Djalog	Cutput window	ow C Last occurred error	
In batch ope	eration		
🗖 Di <u>a</u> log	🔽 Ou <u>t</u> put windov	ow	
		OK Cancel	

In GUI Operation

Specifies where to output an error at GUI operation.

In Command Operation

Specifies where to output an error at command operation.

In Batch Operation

Specifies where to output an error at batch operation.

Error Output Level

Sets the output type when several errors occur.



12.10 Access Size

Setup debug er	nvironment			×
External men	· · · · · · · · · · · · · · · · · · ·		Inaco	cessible area
Tab	Error output	Access :	size	Load
<u>S</u> tart Address:	H.0000000	0		Append
End Address:	H.000000	0		
A <u>c</u> cess Size:	Byte	•		Delete
		()K	Cancel

It is a function to set access size when the debugger accesses memory. When this setting is not done, the debugger does memory access by a command qualifiers or the most suitable size. Because it is set automatically about a built-in resource, setting is unnecessary by this function. However, the debugger does memory access by byte size on FILL, MOVE, COMPARE commands.

Start Address

Specifies the start address to be set.

End Address

Specifies the end address to be set.

Access Size

Specifies the access size to be set. (Byte/Halfword/Word)

List

Displays the currently set area.



12.11 Load

Setup debug envi	ronment			×
Step execution Directory Tab	- <u> </u>	ix Emul Load	ation Mo Frequency	onitoring Break
Specification b B <u>e</u> fore:	atch file before/after	load	Bīov	wse
<u>A</u> fter:			Brov	wse
Only <u>d</u> ebug ir	nformation			
		Ok		Cancel

This sets the environment when loading a target file registered in the project.

Specification Batch File before/after load

Before

This specifies the batch file to execute prior to the loading of the target file. This can also be changed using the Debugger's setup wizard.

After

This specifies the batch file to execute after the loading of the target file. This can also be changed using the debugger's setup wizard.

Debug Information Only

This specifies whether or not to load debug information. When checked, only the debug information is loaded.

Consider a system where MCU application program is stored in external flash and MCU is to be started in external vector fetch mode.

For the emulation of such a system one can connect the target with this check box marked. By doing this only debug information is loaded and with the help of Softune one can debug the application program. The restriction in this scenario is one can use only hardware breakpoint for debugging.

Auto Mapping

This specifies whether or not to enable the auto-map setting. When checked, auto-map setting is enabled.

On demand load

Set whether to on demand load debug information. When a checkmark is placed in the checkbox, debug information is on demand loaded.



12.12 External Memory Emulation

Setup debug environment	×
Execution Watch Radix Emulation Break Monitoring Tab Error output Access size Access size Access size Access size External memory emulation Frequency Inaccess Access size Access size	Directory Load sible area
Enable Disable External memory emulation Chip select: CSO Memory type ROM RAM	
ОК	Cancel

Enable/disable

Whether to enable or disable the external memory emulation function is specified.

Chip select

The chip select number that can be output to the external bus is specified.

FR system: CS1 to CS5

FRex system: CS0 to CS7

Memory type

Whether to allow or inhibit write access to external memory is specified.



12.13 Frequency

Setup debug er	nvironment			×
Execution W	atch Radix Emu	lation Break	Monitorin	g Directory
Tab	Error output	Access :	size	Load
External men	nory emulation	Frequency	Inacce	ssible area 💧
CPU freque <u>F</u> requency:		MHz		
		0	ОК	Cancel

CPU Frequency

Set maximum CPU frequency. (Lower value results in higher responsiveness of emulation system.)

12.14 Inaccessible area

Setup debug environment	×
Execution Watch Radix Emulation Break Monit Tab Error output Access size External memory emulation Frequency Ina	toring Directory Load
Start Address: #*0000000	Append
End Address: R '00000000	
List:	Delete
OK	Cancel

Selects default code break point (i.e. Software or Hardware)



This function inhibits access to debugger memory. Up to 16 areas can be set (by increments of one byte).

Start Address

Specifies the start address to be set.

End Address

Specifies the end address to be set.

List

Displays an regions being currently set. When the check mark of the area is removed, that the area is invalidated.

13. Trigger-Input and Emulator-Output



How to Use the Two BNC Connectors

13.1 The BNC Connectors



The MB2198-01 emulator has two BNC connectors. The left one is the "TRIG"-Input and the right one the "EMUL"-Output.

13.2 Trigger-Input

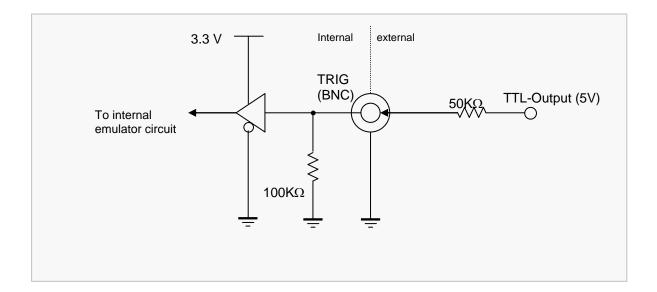
With this input an execution can be stopped. The Trigger-Input is a hardware "break point".

A logical "high" (= 3.3V) on this input stops the execution in the debugging mode. Note, that because of internal latches and different clock speeds of the emulator and the MCU the termination is not immediately. The break slip is in a range of dozens to hundreds machine clock cycles.

The execution can be resumed after a triggered break.

Because of the 3.3V input and an internal 100K pull down resistor, it is recommended to use a serial 50K resistor, if a 5V signal is used:





13.3 Emulator-Output

The BNC-Output "EMUL" goes logical "high" (= 3.3V) if a program is executed and is "low" (= 0V) if the program is stopped or a break point has occurred. This signal can be used for controlling external hardware.

14. Installing LAN



This Chapter Describes How to Install the LAN Interface

14.1 Overview

The emulator MB2198-01 are provided with a local area network adapter that can be used to program and debug a device over a network connection. No additional hardware connection except the LAN connection itself is required for this purpose

Using the Cypress LAN-remote controlled debug facility,

- A Cypress support engineer can easily help solving a concrete problem by debugging your application out of a Cypress support centre.
- You can control the emulator from different locations without having to move your hardware installation from one place to another.
- Program download will be more than 6 times faster than with RS232C

14.2 Configuring the LAN Adapter

- 1. Refer to the LAN Installation Manual coming with the LAN Adapter. This Application Note will give only some additional information.
- 2 Connect emulator and PC by the serial RS232-Port (or USB-Port)
- 3 Start the "LAN Address" Program in the Softune Workbench Folder



- 4 Click on *Communication* to define the serial RS232 Port where the emulator is connected to or use USB connection.
- 5 Read current status from the emulator
- 6 Set the unique IP address given from your network manager. This is an very important point, every IPaddress within a network has to be unique.
- 7 Check the Port address: must be 5001
- 8 Make settings valid by "Set", and reset the emulator when prompted
- 9 Close Program "LAN Address"



🔚 LAN Address	_
<u>File View H</u> elp	
Setting of ICE	
ICE <u>N</u> ame:	MB2198
MAC address	
• Universal	C Local
Universal address:	00:00:0E:1F:13:67
Local address:	
<u>I</u> P address:	192.168.0.1
Port address:	5001
Su <u>b</u> net Mask:	255.255.255.0
<u>Communication</u>	<u>R</u> ead <u>S</u> et

14.3 Configuring Operating System "Windows[™]"

- 1. Within your Windows[™]-directory (e.g. *C:\windows* or *C:\WINNT\system32\drivers\etc*) you should find three files:
- Services
- Hosts
- Imhosts[.sam]

(Note: The files may not have a file-extension!)

2. Make a copy off all three files, like for an example: services to sevices.old, etc.

Edit file services and add the following line:

fjicesv 5001/tcp # Cypress emulator



//////////////////////////////////////	otepad			- 🗆 ×
<u>File</u> Edit Fo	rmat <u>H</u> elp			
# Copyright	(c) 1993-1999 M	icrosoft Corn		
# COPyright	(C) 1995-1999 H.	terosore corp.		
	contains nort n	whers for well-	known services defined by IANA	
#	concarno pore n	Amberb for well	Anown berviceb actined by TANA	
# Format:				
#				
P	name> <nort num<="" td=""><td>er>/<protocol></protocol></td><td>[aliases] [#<comment>]</comment></td><td></td></nort>	er>/ <protocol></protocol>	[aliases] [# <comment>]</comment>	
#		, .p	[] ["	
l'				
echo	7/tcp			
echo	7/udp			
discard	9/tcp	sink null		
discard	9/udp	sink null		
systat	11/tcp	users	#Active users	
systat	11/tcp	users	#Active users	
daytime	13/tcp			
daytime	13/udp			
qotd	17/top	quote	#Quote of the day	
qotd	17/udp	quote	#Quote of the day	
chargen	19/tcp	ttytst source	#Character generator	
chargen	19/udp	ttytst source	#Character generator	
ftp-data	20/tcp		#FTP, data	
ftp	21/tcp		#FTP. control	
telnet	23/tcp			
smtp	25/tcp	mail	#Simple Mail Transfer Protocol	
time	37/tcp	timserver		
time	37/udp	timserver		
rlp	39/udp	resource	#Resource Location Protocol	
fjicesv	5001/tcp			
4				

(note: if 5001/tcp is already contained in the file s*ervices*, use an unused number beginning with 5002 or greater, e.g. fjicesv 5002/tcp. In that case also the emulator address given by the program "LAN address" (see above) has to be changed!

- 3. When saving again the file Services be sure that your editor (e.g. notepad) will not add any extension, e.g. .txt, to your file. To get sure, use quotation marks for the filename: File save as: "services"
- 4. The file *hosts* is used to make a redefinition of the complex IP-number with a simple name within your global network. This may be important if you use a DNS-Server. Of course, you can define different names for the same IP-Address, as shown below. Edit file *hosts* and add the following line:

"The unique IP address (as set by LAN-Address, see above)" "Nickname of emulator"

e.g. 141.187.6.82 Emulator





```
🧖 hosts - Notepad
                                                                        _ 🗆 🗵
<u>File Edit Format H</u>elp
# Copyright (c) 1993-1999 Microsoft Corp.
                                                                            *
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#
       102.54.94.97
                                                 # source server
                        rhino.acme.com
#
        38.25.63.10
                                                 # x client host
                        x.acme.com
127.0.0.1
                localhost
141.187.6.82
                Emulator
```

When saving again the file *Hosts* be sure that your editor (e.g. notepad) will not add any extension, e.g. *.txt*, to your file. To get sure, use quotation marks for the filename: File save as: "*hosts*"

5. The file *Imhosts* (or *Imhosts.sam*) is used to make a redefinition of the complex IP-number with a simple name within your local network. Of course, you can define different names for the same IP-Address, as shown below. Edit file *Imhosts* and add the following line:



🖉 Imhosts.sam - Notepad - 🗆 × File Edit Format Help ٠ # #BEGIN ALTERNATE # #INCLUDE \\localsrv\public\lmhosts # #INCLUDE \\rhino\public\lmhosts # #END_ALTERNATE # # In the above example, the "appname" server contains a special # character in its name, the "popular" and "localsrv" server names are # preloaded, and the "rhino" server name is specified so it can be used # to later #INCLUDE a centrally maintained lmhosts file if the "localsrv" # system is unavailable. # # Note that the whole file is parsed including comments on each lookup, # so keeping the number of comments to a minimum will improve performance. # Therefore it is not advisable to simply add lmhosts file entries onto the # end of this file. 141.187.6.82 fjiecsv 141.187.6.82 Emulator

"the unique IP address (as set by LAN-Address, see above)" "nickname of emulator" e.g.

141.187.6.53 fjicescv 141.187.6.53 Emulator

When saving again the file *Lmhosts* get sure that your editor (e.g. notepad) will not add any extension, e.g. .txt, to your file. To get sure, use quotation marks for the filename: e.g. File save as: "*Imhosts*" (or *Imhosts.sam*)

14.4 Checking the network-connection

Disconnect serial RS232 (or USB) cable from Emulator and try to find the emulator:

- Open DOS-Window (or open RUN (Ausführen) in the Start-Menu)
- The command ping Emulator should acknowledge with some time-values, that means the network is set up right.

The emulator with LAN-adapter is successfully integrated in the network environment and can be used by the Softune Workbench.

14.5 Troubleshooting

If the command ping my_emulator will reply with a timeout-message, try to find the emulator by its IP-address: Type command ping *IP-address*, e.g. ping 141.187.6.53.

If this will work, then check settings (nickname and IP-address) within *Lmhosts* and *Hosts*. If neither nickname (my_emulator) nor IP-address will work, check settings done by the program *LAN-address* and check the file *Services*.

Also check your physical network interconnection cables. Please keep in mind, that the emulator only works with 10Mbit/s. This means in case that for a 100Mbit/s network a 100Mbits/10Mbit - HUB is needed. When using a HUB for 10Base-T then a standard (1:1) network cable has to be used. If the emulator is connected directly to the PC a "crossed network cable" is necessary.



_ 🗆 🗵

_ 🗆 🛛

٠

٠

Good answer: Emulator/LAN-adapter replies with time-values:

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
c:\>ping 141.187.6.82
Pinging 141.187.6.82 with 32 bytes of data:
Reply from 141.187.6.82: bytes=32 time<10ms TTL=64
Reply from 141.187.6.82: bytes=32 time<10ms TTL=64
Reply from 141.187.6.82: bytes=32 time<10ms TTL=64
Ping statistics for 141.187.6.82:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 0ms, Average = 0ms
c:\>ping Emulator
Pinging Emulator
Pinging Emulator [141.187.6.82: bytes=32 time<10ms TTL=64
Reply from 141.187.6.82: bytes=32 time<1
```

"Failed"-answer: Emulator/LAN-adapter replies with timeout message:

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
c:\>ping 141.187.6.82
Pinging 141.187.6.82 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Ping statistics for 141.187.6.82:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
Approximate round trip times in milli-seconds:
    Minimum = Oms, Maximum = Oms, Average = Oms
c:\>
```



14.6 Softune Workbench

Within the Softune Workbench the LAN interface can be used instead of a serial RS232C communication. Detail-Host: can be the "nickname" as defined in the files *hosts* and *Imhosts* or the IP-address of the emulator, set by the program "LAN-Address", can be used.

Setup Wizard	×	
	Please select device type. Type Device Name: AN Detail Host: 141.187.6.82 Nickname of IP address	
< <u>B</u> ack	Next > Finish Cancel	

15. Miscellaneous



Remarks and Hints

15.1 View Mode of the Editor

When starting the Softune Workbench Software for the first time, the text editor has some default viewing

21	unsigned char LED;↓		
22	-		
23	/* Sub Routines */↓		
24	•		
25	void init_timer(void)↓		
26	{↓		
27	^ TMRLR0 = 0x61A7; ^	<u>^</u>	/* relo
28	<pre>^ TMCSR0 = 0x81B;^^</pre>	<u>^</u>	/* pres
29	}↓		

settings, which can be switched off. Source codes will look like in the left picture.

You can disable the viewing of the tabulators, Return signs and the End-of-File delimiter by clicking on the right mouse button just over the text window. Then a large pop up window will open.

Choose Customize. Then the following new window will occur:

Standard Editor	x
View2	
✓ Ret ✓ Tab ✓ EOF N Line Number ✓ Ruler ✓ Auto Indent ✓ C Keyword ✓ C++ Keyword	View Iag: Error:
	Emphasis Characters
Tab <u>S</u> ize: 4	Assembly:
	OK Cancel

The first three entries in the item list are for selecting and deselecting the view of these non-printing characters.

16. Appendix



Related Documents

16.1 Related Documents

Please find further information in the following documents.

- MB2198-01 Hardware Manual (Emulator)
- MB2198-01 Getting Started Application Note
- MB2198-01 Installation Guide Application Note

17. Additional Information



Information about Cypress Microcontrollers can be found on the following Internet page: http://www.cypress.com/cypress-microcontrollers

Revision History



Document Revision History

Document Title: FR Family MB2198-01 Emulator System Getting Started Guide							
Document Number: 002-05222							
Revision	Issue Date	Origin of Change	Description of Change				
	11/13/2003		Initial release				
** 01/25/2008 NO		NOFL	Installing LAN chapter added				
	07/09/2008		Complete reorganisation and based on MB91460 Series				
*A	03/22/2016	NOFL	Migrated Spansion Guide from MCU-AN-391027-E-V20 to Cypress format				