



Sophia
systems™

Sophia systems™

Has made professional grade embedded systems tools for over 30 years

More than 70 full time dedicated embedded design tools engineers

Offering a wide range of embedded development tools for:

- **ARM**
- **Intel**
- **Marvell**
- **Texas Instruments**
- **Tensilica**
- **Renesas**
- **Many others**





World-wide Sales & Support

Antycip

Direct Insight

Emdoor

Flash Technology

MDS Technology

Sightsys

Sophia Systems USA

Steffi Smile Agencies

Superlink Technologies

Trident Infosol Pvt. Ltd

France, Italy

United Kingdom

China, Hong Kong

Singapore, Malaysia

South Korea

Israel

The Americas

India

Taiwan

India

Sandgate III-P for Mobile Device

This is a working PXA310 device that is *very close* to a finished commercial product.



Video Accelerator
DDR Memory
Enhanced Intel Wireless MMXTM2
Enhanced Intel SpeedStep®
Graphic Acceleration
USB2.0 Device
NAND Flash
Several OS Supported

WiFi (802.11b/g) W-SIM
Large capacity memory for your applications
USB OTG Functions can be added as USB host
MicroSD
Three axis acceleration sensor
Terrestrial magnetism sensor
Optional: VGA LCD
High-performance 624MHz PXA310

For Software vendors
Application Middleware

For mobile info device-
Development Vendors

For service Content
Providers

For device vendors

For system integrators

For Educational facilities



Speed up delivery - Add our JTAG emulator with Watchpoint debug environment to solve software and hardware problems not addressed by MS WinCE or Intel C++ development tools. It links into your IDE session to add low level hardware debugging .



FLASH Memory Utilities

Flash Memory Setup ✕

Flash memory type: INTEL 16M 28F128K3/K18 16-BIT(write buffer)(block lock release) ▼

Start address: R;0x12000000 ▼

Flash Memory Connections

Parallel Connections

1 2 4 8

Series Connections

1 ▼

Add Range

Delete Range

Delete All

Close

Use work memory for Downloading (work area setting is in different dialog box)

Flash memory ranges:

R;0x00000000 - R;0x01ffffff	P2 S1 32BitBus	INTEL 16M 28F128J3A 16-BIT(write buffer 32byte)(bl
R;0x10000000 - R;0x10ffffff	P1 S1 16BitBus	FUJITSU 16M MBM29BS/FS12DH 16-BIT
R;0x12000000 - R;0x13ffffff	P1 S2 16BitBus	TOSHIBA 16M TC58FVM7T(B)2ATG 16-BIT

- Over 300 FLASH devices supported
- Parallel and series memory configurations are possible
- Clear, fill, and test target FLASH
- High speed direct to FLASH download
- Upload target FLASH
- Macro controlled programming



The screenshot displays the Watchpoint software debugger interface with several windows open:

- Project Window:** Shows a tree view of project files including `_memcpy`, `chk_address`, `config_reg_value`, `count_7seg`, `gpioInit()`, `greet()`, `i2cInit()`, and `init()`.
- Disassembly Window:** Shows assembly code for `LDRB R0,[R13,#0]` at address `0x000006f0`. The instruction is highlighted in yellow.
- Register Window:** Lists CPU registers from `R0` to `R10` with their current values. `R0` is `0x00000013`.
- Source Code Window:** Shows C code for `while(ch[i] != 43){` at address `00327`. The line is highlighted in yellow.
- Watch Window:** A table with columns for Symbol and Value. It lists variables like `i` (0), `config_reg_value` (0x3000), and `count_7seg` (0x0000077c).
- Local Window:** A table with columns for Symbol and Value. It lists local variables `i` (0) and `ch` (0x00000000).
- Memory Dump Window:** Shows a hex dump of memory starting at address `0x000006f0`. The first few bytes are `00 00 dd e5 2b 00 50 e3 0b 00 00 0a 01 21`.
- Command Window:** Contains commands like `// Start flash memory clear...`, `// (ALL Clear : 0x00000000)`, `FHCLEAR ALL 0x00000000`, and `// Normal End`.

- ▶ Source code down to bit level
- ▶ Unlimited software breakpoints
- ▶ Single-step source and assembly
- ▶ View/Modify Memory in multiple formats in multiple windows
- ▶ View & Modify Variables
- ▶ View & Modify Registers
- ▶ Reverse assembly
- ▶ Mixed mode code display
- ▶ Modify assembly code
- ▶ Watch window
- ▶ Local variable window
- ▶ FLASH – Test, Read, Write, Fill Clear - Hundreds supported
- ▶ Compiler support for Intel C, Linux, WinCE5 & WinCE6



View & Modify Special Function Registers

WATCHPOINT [HyperUSB PXA27x I/O port 0000H] - 2p.dbg

File Edit Search View Resource Go Options Window Help

PRJ REG MEM ASM CMD BIN HIST JANE HIST CVG CALL WAT LOC BRO HIST

GO STOP STEP PASS OUT GO ME H BP S BP R SET ARM THU MB AU TO MAP BAT

Internal I/O Registers

Register filename: C:\WATCHPOINT\US2_PXA27x\IPXA27x.csv Browse Load

Group:

- MSC0 - Static Memory Control Register 0
- MSC1 - Static Memory Control Register 1
- MSC2 - Static Memory Control Register 2
- MECR - Expansion Memory Configuration Register
- SXCNFG - Synchronous Static Memory Configuration Register
- MCMEM0 - PC Card interface Common Memory Space Socket 0 Timing Configuration
- MCMEM1 - PC Card interface Common Memory Space Socket 1 Timing Configuration
- MCATT0 - PC Card interface Attribute Space Socket 0 Timing Configuration
- MCATT1 - PC Card interface Attribute Space Socket 1 Timing Configuration
- MCIO0 - PC Card interface I/O Space Socket 0 Timing Configuration
- MCIO1 - PC Card interface I/O Space Socket 1 Timing Configuration
- MDMRS - SDRAM Mode Set Configuration Register
- BOOT_DEF - Boot-Time Default Configuration Register
- BSCNTR0 - System Memory Buffer Strength Control Register 0
- BSCNTR1 - System Memory Buffer Strength Control Register 1
- MDMRSPL - Low-Power SDRAM Mode Register Set Configuration Register
- BSCNTR2 - System Memory Buffer Strength Control Register 2
- BSCNTR3 - System Memory Buffer Strength Control Register 3
- SA1110 - SA-1110 Compatibility Mode for Static Memory Register
- LCD Controller
 - LCCR0 - LCD Controller Control Register 0
 - LCCR1 - LCD Controller Control Register 1
 - LCCR2 - LCD Controller Control Register 2
 - LCCR3 - LCD Controller Control Register 3
 - LCCR4 - LCD Controller Control Register 4
 - LCCR5 - LCD Controller Control Register 5
 - FBR0 - DMA Channel0 Frame Branch Register 0
 - FBR1 - DMA Channel1 Frame Branch Register 1
 - FBR2 - DMA Channel2 Frame Branch Register 2
 - FBR3 - DMA Channel3 Frame Branch Register 3
 - FBR4 - DMA Channel4 Frame Branch Register 4
 - FBR5 - DMA Channel5 Frame Branch Register 5
 - FBR6 - DMA Channel6 Frame Branch Register 6
 - LCSR0 - LCD Controller Status Register 0
 - LCSR1 - LCD Controller Status Register 1
 - OVL1C1 - Overlay 1 Control Register 1
 - OVL1C2 - Overlay 1 Control Register 2
 - OVL2C1 - Overlay 2 Control Register 1

Register Value: 0x5c80400
Address Value: 0x44000000

Bit

- 29:
- 28:
- 27:
- 26: LDDALT - LDD Alternate Mapping Control Bit
- 25: OUC - Overlay Underlay Control Bit
- 24: CMDIM - LCD COMMAND INTERRUPT MASK
- 23: RDSTM - LCD READ STATUS INTERRUPT MASK
- 22: LCDT - LCD PANEL TYPE
- 21: OUM - OUTPUT FIFO UNDERRUN MASK
- 20: BSMO - BRANCH STATUS MASK
- 19: PDD - PALETTE DMA REQUEST DELAY (bits7)
- 18: PDD - PALETTE DMA REQUEST DELAY (bits6)
- 17: PDD - PALETTE DMA REQUEST DELAY (bits5)
- 16: PDD - PALETTE DMA REQUEST DELAY (bits4)
- 15: PDD - PALETTE DMA REQUEST DELAY (bits3)
- 14: PDD - PALETTE DMA REQUEST DELAY (bits2)
- 13: PDD - PALETTE DMA REQUEST DELAY (bits1)
- 12: PDD - PALETTE DMA REQUEST DELAY (bits0)
- 11: DIS - LCD QUICK DISABLE MASK
- 10: DIS - LCD DISABLE
- 9: DPD - DOUBLE-PIXEL DATA PIN MODE
- 8:
- 7: PAS - PASSIVE/ACTIVE DISPLAY SELECT
- 6: EOFM0 - END OF FRAME MASK(Channel 1_2)
- 5: IUM - INPUT FIFO UNDERRUN MASK
- 4: SOFM0 - START OF FRAME MASK(Channel 1_2)
- 3: LDM - LCD DISABLE DONE MASK
- 2: SDS - SINGLE-/DUAL-PANEL DISPLAY SELECT
- 1: CMS - COLOR/MONOCHROME SELECT
- 0: ENB - LCD CONTROLLER ENABLE

Read/Write register: + -

Initial Apply Regisplay

**SFR Bits are labeled
Like having the manual on line**

Click to turn it off

Ready

start WATCHPOINT [Hyper... Microsoft PowerPoint ...

RESET TARGET 9:30 AM



Program Execution

The screenshot displays the Watchpoint debugger interface with several key components:

- Top Menu:** File, Edit, Search, View, Resource, Go, Options, Window, Help.
- Toolbar:** Includes a large 'GO' button, 'PASS', 'OUT', 'ME', 'BP', 'SET', 'ARM', 'MB', 'TO', 'MAP', 'BAL', 'LOC', 'BRC', 'HIST', 'TRC', 'CVG', 'PFM', 'PRF', and 'BTR' buttons.
- Project Window (Left):** Shows a tree view of the project 'sample.axf' with folders like 'Global Variables(222)', 'Array.c', 'Char.c', 'Control.c', 'Double.c', and various function pointers.
- Source Code Window (Right):** Displays the C source code for 'C:\WATCHPOINT\...\ArmC\Char.c'. The current line of execution is highlighted in red (line 00033). A yellow arrow points to this line, labeled 'PC line'. Executable lines are marked with a red 'H' icon.
- Disassembly Window (Bottom):** Shows the assembly code corresponding to the source code. The current instruction is highlighted in red: 'MOV R2, R13'. A blue arrow points from the source code window to this disassembly window, labeled 'Switch between Source & Disassembly windows'.

Executable lines display in red

PC line

Switch between Source & Disassembly windows



Single Stepping

The screenshot shows the Watchpoint debugger interface. At the top, the menu bar includes File, Edit, Search, Window, and Help. Below the menu bar is a toolbar with several buttons. Two buttons, 'STEP' and 'PASS', are highlighted with yellow boxes. A blue callout box with white text points to the 'PASS' button, stating: "Pass runs full speed to the next call function". Another blue callout box with white text points to the 'STEP' button, stating: "Single Step within current function".

The main window is divided into several panes:

- Project Window:** Shows a tree view of the project files. The file 'Double.c' is expanded, and the function 'About_double()' is selected.
- Code Window:** Displays the source code for 'About_double()'. The current line of execution is highlighted in yellow. The code includes:


```

00033 Enumeration kind;
00034 char auto1, auto2;
00035
00036
00037 auto1=auto2=0x10;
00038 for(kind=AssignConst; kind!=CalcVariable; ++kind){
00039     i auto1=val_auto_char(kind, auto1, auto2);
00040     auto1=ref_auto_char(kind,&auto1,&auto2);
00041     auto1=ptr_auto_char(kind,&auto1,&auto2);
00042 }
00043 }
00044
00045 static char val_auto_char(Enumeration kind, char arg1,
00046 {
00047     switch(kind){
00048     case AssignConst:
00049         arg1=(char)1;
00050         arg2=(char)2;
00051         break;
00052     case AssignVariable:
00053         arg1=arg2;
00054         arg2=arg1;
00055     }
00056 }
      
```
- Disassembly Window:** Shows the assembly code corresponding to the source code. The current instruction is highlighted in yellow:


```

0x0000e6e8 E1A0200D MOV R2,R13
      
```

At the bottom of the window, there is a status bar with the text 'Ready' and a 'BREAK' button.



Setting Breakpoints

```

C:\Ygywin\usr\work\Nios2.1_1_port\src\char.c
00015 static char ptr_sta_char(Enumeration);
00016 static void ext_char(void);
00017 static char val_ext_char(Enumeration);
00022 About_char()
$ 00023 {
00024     auto_char();
00025     sta_char();
$ 00026     ext_char();
00027 }
00028
00029 /*-----
00030 /* Function for automatic char variable
00031 /*-----
00032 static void auto_char()
00033 {
00034     char auto1, auto2;
00035
00036
$ 00037     auto1=auto2=0x10;
00038     for(kind=AssignConst; kind!=CalcVariable; ++kind){
00039         auto1=val_auto_char(kind, auto1, auto2);
00040         auto1=ref_auto_char(kind,&auto1.&auto2);

```

Simply click the step to set breakpoints in the source and assembly code windows

Unlimited Breakpoints

Breakpoints Setup

CPU Status: Code Fetch Add

Load module: D:\wp\xscale\sample\XScale\ArmC\sample.axf

Address/File: main Browse Source Files, Symbols...

Line number:

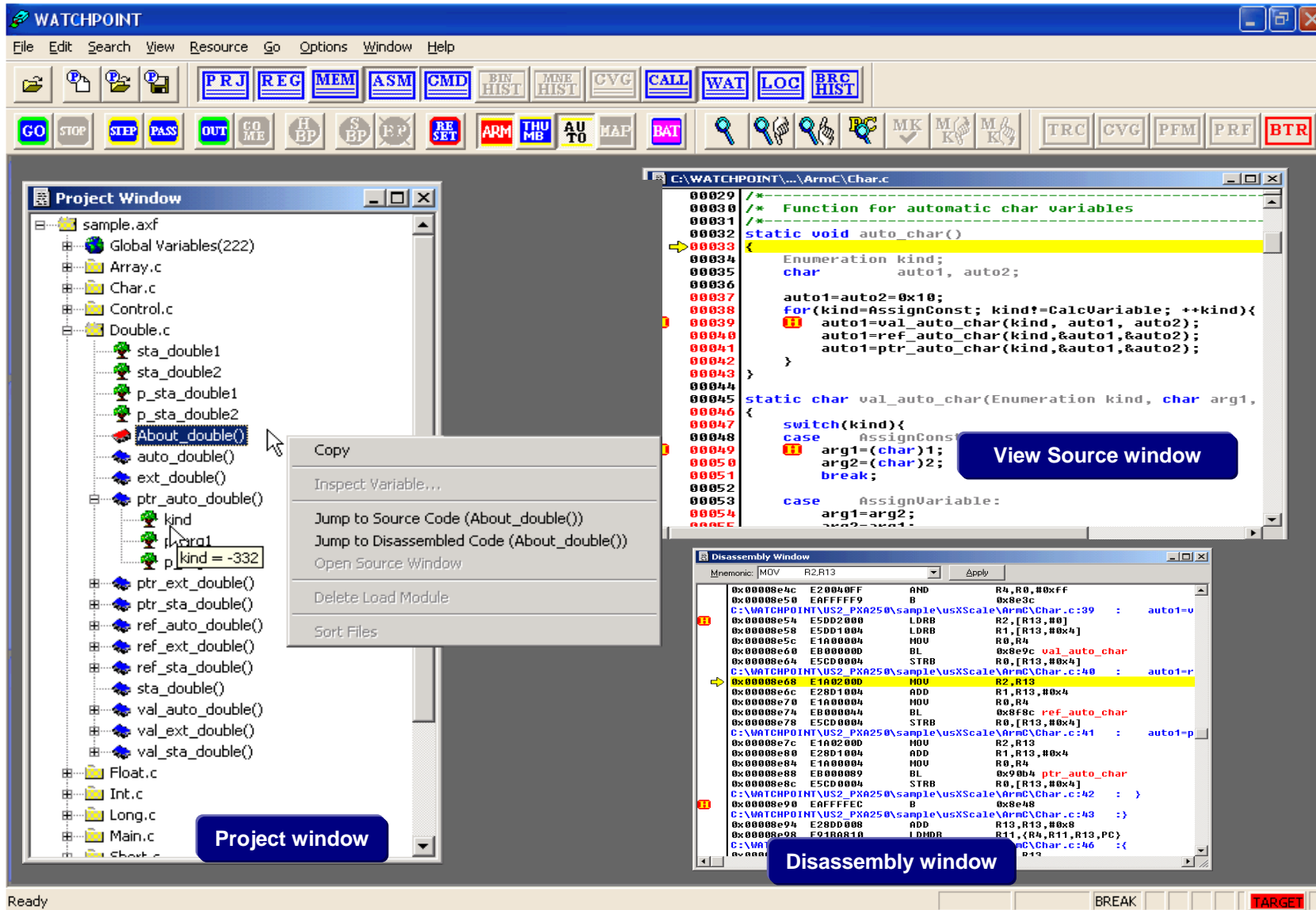
Breakpoints:

[o] D:\wp\xscale\sample\XScale\ArmC\Main.c:L12 <Soft(ARM)>

Enable
Delete
Delete All
OK
Cancel

Or use the breakpoint setup window

Find & Display Code



The screenshot displays the Watchpoint debugger interface with three main windows:

- Project Window:** Shows a tree view of the project structure. The file `About_double()` is selected, and a context menu is open over it. The menu options include:
 - Copy
 - Inspect Variable...
 - Jump to Source Code (About_double())
 - Jump to Disassembled Code (About_double())
 - Open Source Window
 - Delete Load Module
 - Sort Files
- Source Code Window:** Displays the C source code for `ArmCChar.c`. The function `auto_char()` is highlighted in yellow. A blue callout box labeled "View Source window" points to the source code.
- Disassembly Window:** Shows the assembly code corresponding to the selected source code. The instruction `MOV R2, R13` is highlighted in yellow. A blue callout box labeled "Disassembly window" points to the disassembly.

Find Source and Disassemble code fast from Project window



Project Window

The Project Window displays a hierarchical tree view of source files and variables. The tree is expanded to show the 'Double.c' file, which contains several variables and functions. A context menu is open over the 'About_double()' function, showing options such as 'Copy', 'Inspect Variable...', 'Jump to Source Code (About_double())', 'Jump to Disassembled Code (About_double())', 'Open Source Window', 'Delete Load Module', and 'Sort Files'. A tooltip is visible over the 'kind' variable, showing its value as 'kind = -332'.

Load module information

Load module file names that are linked to the source file names

Double click on the source file name to display source code in the source window

Double click on the function name to display the function in the source window

The Source Window displays the source code for the 'auto_char()' function. The code is as follows:

```

00029 /*-----*/
00030 /* Function for automatic char variables */
00031 /*-----*/
00032 static void auto_char()
00033 {
00034     Enumeration kind;
00035     char auto1, auto2;
00036
00037     auto1=auto2=0x10;
00038     for(kind=AssignConst; kind!=CalcVariable; ++kind){
00039         auto1=val_auto_char(kind, auto1, auto2);
00040         auto1=ref_auto_char(kind,&auto1,&auto2);
00041         auto1=ptr_auto_char(kind,&auto1,&auto2);
00042     }
00043 }
00044
00045 static char val_auto_char(Enumeration kind, char arg1,
00046 {
00047     switch(kind){
00048     case AssignConst:
00049         arg1=(char)1;
00050         arg2=(char)2;
00051         break;
00052
00053     case AssignVariable:
00054         arg1=arg2;
00055         arg2=arg1;
    
```

A blue callout box labeled 'Source window' points to the function definition.



View source code

- ▶ Reserved words and comments display in a different color
- ▶ Search forward or backward
- ▶ Set Bookmarks
- ▶ Set unlimited breakpoints in the source window
- ▶ Jump to function
- ▶ Jump to and back from assembly and source view windows

```
C:\WATCHPOINT\...\ArmC\Char.c
00029 /*-----
00030 /* Function for automatic char variables
00031 /*-----
00032 static void auto_char()
00033 {
00034     Enumeration kind;
00035     char auto1, auto2;
00036
00037     auto1=auto2=0x10;
00038     for(kind=AssignConst; kind!=CalcVariable; ++kind){
00039         auto1=val_auto_char(kind, auto1, auto2);
00040         auto1=ref_auto_char(kind,&auto1,&auto2);
00041         auto1=ptr_auto_char(kind,&auto1,&auto2);
00042     }
00043 }
00044
00045 static char val_auto_char(Enumeration kind, char arg1,
00046 {
00047     switch(kind){
00048     case AssignConst:
00049         arg1=(char)1;
00050         arg2=(char)2;
00051         break;
00052
00053     case AssignVariable:
00054         arg1=arg2;
00055         arg2=arg1;
```



Disassembly View

Disassembly Window

Mnemonic: MOV R2,R13 Apply

	0x00008e4c	E20040FF	AND	R4,R0,#0xff	
	0x00008e50	EFFFFFF9	B	0x8e3c	
	C:\WATCHPOINT\US2_PXA250\sample\usXScale\ArmC\Char.c:39 : auto1=v				
H	0x00008e54	E5DD2000	LDRB	R2,[R13,#0]	
	0x00008e58	E5DD1004	LDRB	R1,[R13,#0x4]	
	0x00008e5c	E1A00004	MOV	R0,R4	
	0x00008e60	EB00000D	BL	0x8e9c val_auto_char	
	0x00008e64	E5CD0004	STRB	R0,[R13,#0x4]	
	C:\WATCHPOINT\US2_PXA250\sample\usXScale\ArmC\Char.c:40 : auto1=r				
→	0x00008e68	E1A0200D	MOV	R2,R13	
	0x00008e6c	E28D1004	ADD	R1,R13,#0x4	
	0x00008e70	E1A00004	MOV	R0,R4	
	0x00008e74	EB000044	BL	0x8f8c ref_auto_char	
	0x00008e78	E5CD0004	STRB	R0,[R13,#0x4]	
	C:\WATCHPOINT\US2_PXA250\sample\usXScale\ArmC\Char.c:41 : auto1=p				
	0x00008e7c	E1A0200D	MOV	R2,R13	
	0x00008e80	E28D1004	ADD	R1,R13,#0x4	
	0x00008e84	E1A00004	MOV	R0,R4	
	0x00008e88	EB000089	BL	0x90b4 ptr_auto_char	
	0x00008e8c	E5CD0004	STRB	R0,[R13,#0x4]	
	C:\WATCHPOINT\US2_PXA250\sample\usXScale\ArmC\Char.c:42 : }				
H	0x00008e90	EFFFFFFEC	B	0x8e48	
	C:\WATCHPOINT\US2_PXA250\sample\usXScale\ArmC\Char.c:43 : }				
	0x00008e94	E28DD008	ADD	R13,R13,#0x8	
	0x00008e98	E91BA810	LDMDB	R11,{R4,R11,R13,PC}	
	C:\WATCHPOINT\US2_PXA250\sample\usXScale\ArmC\Char.c:46 : {				
	0x00008e9c	E1A0000D	MOV	R12,R13	

View Assembly Code

Display mnemonic code

Display the source file
and its symbol

In-line assembly

Jump to Source window from
Disassembly View



In-line Assembly

Input new instruction

Before

```

Disassembly Window
Mnemonic: MOV R12,R13 Apply
D:\usr\program\WP\pxa250\sample\usXScale\HighC\char.c:33 :{
R; 0x00000524 E1A0C00D MOV R12,R13
R; 0x00000528 E92DDF00 STMFD R13!,{R8-R12,R14,PC}
R; 0x0000052c E28DB018 ADD R11,R13,#0x18
R; 0x00000530 E24DD008 SUB R13,R13,#0x8
D:\usr\program\WP\pxa250\sample\usXScale\HighC\char.c:37 : auto1=auto2=0x10;
R; 0x00000534 E3A02010 MOV R2,#0x10
R; 0x00000538 E5CD2000 STRB R2,[R13,#0]
R; 0x0000053c EA00019C B 0xbb4
R; 0x00000540 E1A0C00D MOV R12,R13
D:\usr\program\WP\pxa250\sample\usXScale\HighC\char.c:38 : for(kind=AssignConst; kind!=CalcVariable;
  
```

After

```

Disassembly Window
Mnemonic: B 0x524 Apply
D:\usr\program\WP\pxa250\sample\usXScale\HighC\char.c:33 :{
R; 0x00000524 E1A0C00D MOV R12,R13
R; 0x00000528 E92DDF00 STMFD R13!,{R8-R12,R14,PC}
R; 0x0000052c E1A00000 NOP
R; 0x00000530 E1A00000 NOP
D:\usr\program\WP\pxa250\sample\usXScale\HighC\char.c:37 : auto1=auto2=0x10;
R; 0x00000534 E1A00000 NOP
R; 0x00000538 E1A00000 NOP
R; 0x0000053c E1A00000 NOP
R; 0x00000540 EAFFFFF7 B 0x524 auto_char
D:\usr\program\WP\pxa250\sample\usXScale\HighC\char.c:38 : for(kind=AssignConst; kind!=CalcVariable;
  
```

Adjusted instruction set



Memory Window

The screenshot shows the Watchpoint debugger interface. The main window displays C code from `D:\usr\...\HighC\char.c`. A blue callout box labeled "Input New Value" points to the "Start address" field in the "Memory Dump Window" which contains `%auto1`. Another blue callout box labeled "Drag & Drop to display dump address" points to the address `0x00000010` in the memory dump window. A third blue callout box labeled "Memory window" points to the memory dump window itself. The memory dump shows hexadecimal addresses and their corresponding data values.

Code Snippet:

```

00023 {
00024     auto_char();
00025     sta_char();
00026     ext_char();
00027 }
00028
00029 /*-----*/
00030 /* Function for automatic char variables */
00031 /*-----*/
00032 static void auto_char()
00033 {
00034     Enumeration kind;
00035     char auto1, auto2;
00036
00037     auto1=auto2=0x10;
00038     for(kind=AssignConst; kind!=CalcVariable; ++kind){
00039         auto1=val_auto_char(kind,auto1,auto2);
00040         auto1=ref_auto_char(kind,&auto1,&auto2);
00041         auto1=ptr_auto_char(kind,&auto1,&auto2);
00042     }
00043 }
00044
00045 static char val_auto_char(Enumeration kind, char arg1, char arg2)
00046 {
00047     switch(kind){
00048     case AssignConst:
00049         arg1=(char)1;
00050         arg2=(char)2;
00051         break;
00052
00053     case AssignVariable:
00054         arg1=arg2;
00055         arg2=arg1;
00056         break;
00057
00058     case CalcConst:

```

Memory Dump Window:

Address	+0	+1	+2	+3
0x00000010	ff	c3	c3	c3
0x00000020	ff	c3	c3	c3
0x00000030	ff	c3	c3	c3
0x00000040	ff	c3	c3	c3
0x00000050	ff	c3	c3	c3
0x00000060	ff	c3	c3	c3
0x00000070	ff	c3	43	c3
0x00000080	0d	c0	a0	e1
0x00000090	00	a0	e3	03
0x000000a0	0d	20	a0	e1
0x000000b0	5d	00	00	eb
0x000000c0	ff	a0	0c	e2
0x000000d0	80	df	2d	e9
0x000000e0	01	e0	a0	e1
0x000000f0	00	c0	ce	e5
0x00000100	0a	c1	8a	e0
0x00000110	01	a0	8a	e2
0x00000120	ef	ff	ff	ea



Macro & Command Line Input Window

The screenshot shows the 'Command Window' interface. The main area displays assembly code with addresses and instructions:

```
>RASM 0 LENGTH 0x54c  
0x00000000 MOV R13,#0x1a000  
0x00000004 B 0x3F7c  
0x00000008 BICGT R12,R3,#0xdc000003  
0x0000000c STMII B R9,{R0-R2,R4,R5,R7-R9,R12,R14,PC}^
```

Below the code, the command '>pin' is entered, followed by the prompt '[CPU Pin Status]'. A status bar contains several options: BREAK, TRC-OFF, CVG-OFF, PFM-OFF, LOG-OFF, and View MCU pin states. At the bottom, a menu bar includes buttons for navigation and actions: <<, <, >, >>, PIN (highlighted in green), QUERY, RASM, REG, RESET, SEARCH, SHELLEXE, STEP, and SWITCH. A blue callout box with the text 'Prompted command line input' has an arrow pointing to the PIN button.

Command Window

RESET LOG-OFF Go from current PC

>STEP

FROM

Command Window

```

>STEP FROM 0 ASM 1
R0:00defba0 R1:00000000 R2:00000000 R3:00000019
R4:5f4d1c58 R5:00000000 R6:00000002 R7:5f4d1c58
R8:00000000 R9:00000003 R10:00000000 R11:00000000
R12:00defbf0 R13:00000000 R14:00000000 R15:0000001a

CPSR:00000000
-----
0x00000000 B 0x1000

```

RESET LOG-OFF Step count

>STEP FROM 0 ASM 10

COUNT = 1



Watch Window

The screenshot shows the Watchpoint debugger interface. The main window displays the source code for `sample.c` with a yellow highlight on line 00217. The **Watch Window** is open, showing a list of variables and their values. A blue callout box with an arrow points to the value `0x12345678` in the `sInt` row, with the text "Input new value".

Symbol	Value
sta_struct1	0x00017ea0 {...}
PtrNext	0x00017ea0 *0xffffffff
sChar	0x00017ea4 '\xff' 0xff (255)
uChar	0x00017ea5 '\xff' 0xff (255)
sShort	0x00017ea6 0xffff (-1)
uShort	0x00017ea8 0xffff (65535)
sInt	0x00017eac 0x12345678
uInt	0x00017eb0 0xffffffff (4294967295)
sLong	0x00017eb4 0xffffffff (-1)
uLong	0x00017eb8 0xffffffff (4294967295)
sFloat	0x00017ebc -6.805646E+038
sDouble	0x00017ec0 -3.595386269724E+308
ext_long1	0x00017a1c 0x0 (0)
ext_char1	0x000179bc '\x00' 0x00 (0)
ext_array2	0x00017d84 @0x00017d84
[0]	0x00017d84 @0x00017d84 "\xff\xff\xff\xff"
[1]	0x00017d8e @0x00017d8e "\xff\xff\xff\xff"
[2]	0x00017d98 @0x00017d98 "\xff\xff\xff\xff"
[0]	0x00017d98 '\xff' 0xff (255)
[1]	0x00017d99 '\xff' 0xff (255)
[2]	0x00017d9a '\xff' 0xff (255)
[3]	0x00017d9b '\xff' 0xff (255)

At the bottom of the interface, the **Command Window** shows the following commands:

```
// Start Flash memory clear...
// (Block Clear : 0x00000000)
FMCLEAR BLOCK 0x00000000
// Normal End
```

The **Command Window** also includes a **ASSIGN** button and a status bar at the bottom showing "Ready", "Row 218", "Column 1", "BREAK", and "TARGET".



Viewing Variables

```
C:\Cygwin\usr\work\Nios2.1_1\port\src\char.c
00248     }
00249
00250     return((char)0x10);
00251 }
00252
00253 /*-----*/
00254 /* Function for global char variables          */
00255 /*-----*/
00256 extern char      ext_char1, ext_char2;
00257 extern char      *p_ext_char1, *p_ext_char2;
00258
00259 static void ext_char()
00260 {
00261     Enumeration kind;
00262
00263     ext_char1=ext_char2=0x10;
00264     p_ext_char1 = &ext_char1;
00265     p_ext_char2 = &ext_char2;
00266     for(kind=AssignConst; kind!=CalcVariable; ++kind){
00267         ext_char1=val_ext_char(kind);
00268         ext_char1=ref_ext_char(kind);
00269         ext_char1=ptr_ext_char(kind);
00270     }
00271 }
00272
00273 static char val_ext_char(Enumeration kind)
```

Tip inspect value



ext_char1 = '\x10' 0x10 (16)



Modify Variables

The screenshot shows the Watchpoint debugger interface. The main window displays the source code for `sample.c`. A yellow highlight is on the line `if((i&0x10000) == 0) // Display LED`. The `Inspect Variable` dialog box is open, showing the variable `arg1[3]` with a new value of `0x55` and a current value of `'7'` at address `0xc33f`. The dialog also shows the memory contents of the array `arg1`.

Inspect Variable

Variable Format: (unsigned char)arg1[3]

New value: **input new value**

Current value:

```

[-] arg1 = @0xc33c "aaa7a"
... [0] = 'a' 0x61 (97)
... [1] = 'a' 0x61 (97)
... [2] = 'a' 0x61 (97)
... [3] = '7' 0x37 (55)
... [4] = 'a' 0x61 (97)

```

Selected symbol address value: **Address**

Buttons: Redisplay, Range..., Watch, Close

Command Window:

```

// Start Flash memory clear...
// (Block Clear : 0x00000000)
FMCLEAR BLOCK 0x00000000
// Normal End

```

Debugger Status: Row 218 Column 1 BREAK TARGET

View & modify a structure, array, integer, or other variables.



Variable Array View

Inspect Variable

Variable Format: (char)*(p_arg2)

New value:

Apply

Current value:

```

p_arg2 = *R:0x00019fa4 "\x02\xff\x0c5\x01\xff<\x88"
  [0] = '\x02' 0x02 (2)
  [1] = '\xff' 0xff (251)
  [2] = '\x0c' 0x0c (12)
  [3] = '5' 0x35 (53)
  [4] = '\x01' 0x01 (1)
  [5] = '\xff' 0xff (251)
  [6] = '=' 0x3d (61)
  [7] = '<' 0x3c (60)
  [8] = '\x88' 0x88 (136)
    
```

Refresh

Range...

Watch

Close

Selected symbol address: R:0x00019fa4

Display Array Range

Symbol: p_arg2

Type: char *

Display Array Elements:

Starting element: 0

Ending element: 13

OK

Cancel

Inspect Variable

Variable Format: (char)*(p_arg2)

New value:

Apply

Current value:

```

  [2] = '\x0c' 0x0c (12)
  [3] = '5' 0x35 (53)
  [4] = '\x01' 0x01 (1)
  [5] = '\xff' 0xff (251)
  [6] = '=' 0x3d (61)
  [7] = '<' 0x3c (60)
  [8] = '\x88' 0x88 (136)
  [9] = '\x00' 0x00 (0)
  [10] = '\x00' 0x00 (0)
  [11] = '\x00' 0x00 (0)
  [12] = '\xbd' 0xbd (189)
  [13] = '\xef' 0xef (239)
    
```

Refresh

Range...

Watch

Close

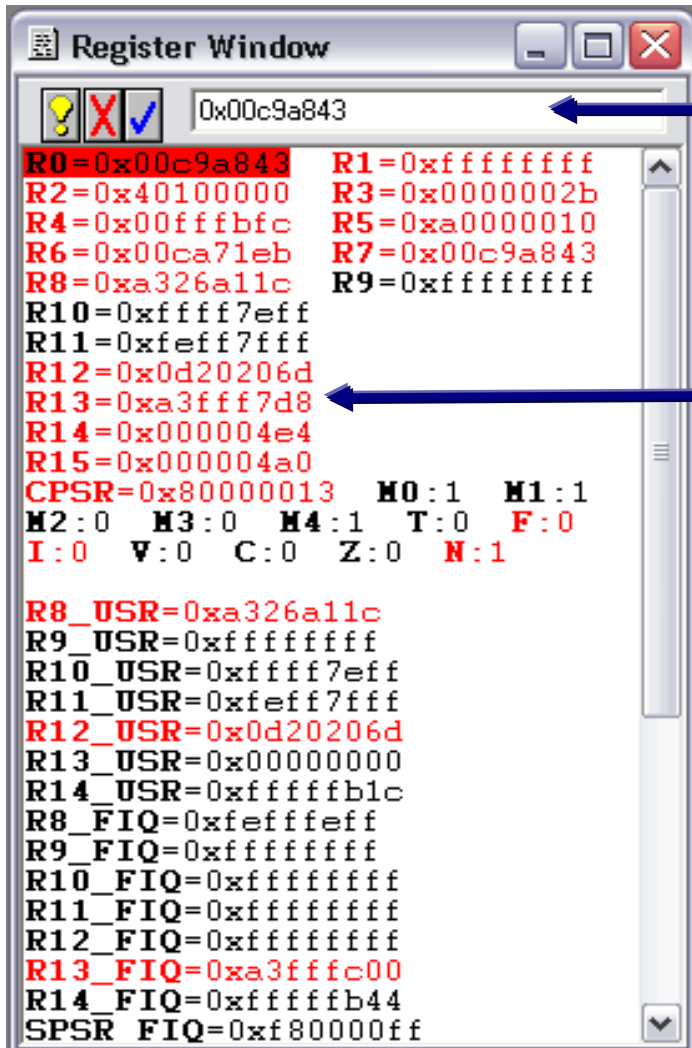
Selected symbol address: R:0x00019fa4

View and modify array elements

Displays the additional range



Viewing and Changing Registers



Click a register and input new value

After go or step all changed register values appear in red

EECO

Enable Engineering Co. Inc.

Call 800.686.4228

email sales@eecosales.com

www.eecosales.com