

Ashling Product Brief APB211 v.1.0.6, 15th January 2013

PathFinder-XD for MIPS[™] Powered Devices Simulator

Contents

| 1. | Introd | luction | | 2 | | | | |
|---|--------|-----------------------------|---|--------|--|--|--|--|
| 2. In 2. 3. Us 4. Er 4. 4. | Instal | lation | | 2 | | | | |
| | 2.1 | Windov | vs™ Installation | 2 | | | | |
| | 2.2 | Linux Ir | nstallation | 2 2 | | | | |
| 3. | Using | PathFind | er-XD with the QEMU simulator | 3 | | | | |
| Instal 2.1 2.2 Using | Embe | dded Linu | ix Debugging with PathFinder-XD and the QEMU Simulator | 10 | | | | |
| | 4.1 | Prepari | ng for debugging | 10 | | | | |
| | 4.1.1 | Building with debug symbols | 10 | | | | | |
| | | 4.1.2 | Compiler optimisations | 10 | | | | |
| 2. li 2 3. L 4. E 4 4 | | 4.1.3 | On-demand paging | | | | | |
| | 4.2 | Stop-m | ode Debugging | 11 | | | | |
| | | 4.2.1 | Sample Stop-mode Linux Debugging Session | 11 | | | | |
| | | | 4.2.1.1 Invoking PathFinder-XD and connecting to the QEMU simulator | 11 | | | | |
| | | | 4.2.1.2 Loading kernel symbol information to PathFinder-XD and executing | | | | | |
| | | | the kernel image | 12 | | | | |
| Ins 2.1 2.2 Us Us En 4.1 4.2 | | | 4.2.1.3 Debug a module from init module() | 17 | | | | |
| | | | 4.2.1.4 Debugging a process from main() | 19 | | | | |
| | | | 4.2.1.5 Debugging a running process | 20 | | | | |
| 3. 4. | | | 4.2.1.6 Library debugging | 21 | | | | |
| | 4.3 | Run-mode Debugging | | | | | | |
| | | 4.3.1 | Preparing for Run-mode Debugging | 21 | | | | |
| | | | 4.3.1.1 Setting up networking between host and QEMU simulator | 21 | | | | |
| | | | 4.3.1.2 Modify QEMU invocation parameters | 22 | | | | |
| | | | Copy and Paste Warning! Do not copy and paste the above into PathFinder- | | | | | |
| | | | XD; it will not work due to PDF issues. The above invocation line can be copied | | | | | |
| | | | and pasted from the README.TXT supplied with PathFinder-XD. | 23 | | | | |
| | | 4.3.2 | Sample Run-mode Linux Debugging Session | 23 | | | | |
| | | | 4.3.2.1 Debugging the Module and Process | 24 | | | | |
| | | | 4322 Debugging multi-threaded applications | 31 | | | | |

4.3.2.2 Debugging multi-threaded applications4.3.2.3 Debugging more than one application at the same time

35

1. Introduction

PathFinder-XD is a C/C++/Assembly debugger based on the Eclipse framework and supports debugging using the QEMU software simulator (<u>www.qemu.org</u>) or the Ashling Opella-XD Debug Probe connected to MIPS[™] powered target hardware.

| 💆 Debug - C:\Program Files\Ashling\PathFinder-XDforMIPS\examples\LE\UnCached\src | \main.c - PathFinde | r-XD for MIPS | |
|--|---------------------|-------------------|-------------------|
| <u>F</u> ile <u>E</u> dit <u>L</u> inux <u>T</u> arget <u>R</u> un <u>W</u> indow <u>H</u> elp | | | |
| 🔲 🖻 🖕 🖇 💕 👹 😂 | | | |
| 🏇 Debug 🛛 🦄 🖗 🕪 💷 🔳 🔍 🗫 ⊥৫ 🔜 🖬 🛒 🏟 🏹 👘 | 🗣 Breakpoints 🚺 |] Memory 🔐 Regist | ers 🖾 🙀 Expressio |
| QEMU Simulator [PathFinder-XD Embedded Debugging] | Name | | Value |
| ⊿ 🎲 UnCached_LE.elf | 👬 General Re | gisters | |
| Thread/Core [0] (Suspended : Breakpoint) | 1010 zero | | 0x0 |
| ■ main() at main.c:18 0xa00005a0 QEMUConsole | 000 at | | 0x1 |
| j gdb | < | | |
| S Gup | 4 | | |
| 🔁 main.c 🛛 | - 8 | 🕪= Variables 🖾 🔛 | Disassembly |
| *************************************** | * | Name | Туре |
| int main(void) | | (×)= iError | unsigned int |
| <pre>i unsigned int iError = NO ERROR;</pre> | | (×)= cCommand | char |
| char cCommand = NO_COMMAND; | = | 🥭 sData | data_struct |
| data_struct sData; | | | |
| ClearDisplay (); | | | |
| <pre>while (TRUE) /* Repeatedly handle commands */</pre> | | | |
| { /* 11=24 for and */ | | | |

Figure 1. PathFinder-XD for MIPS™

PathFinder-XD supports both "bare-metal" (no target operating system) and Embedded Linux based debugging. This application note introduces PathFinder-XD Simulator version and covers using PathFinder-XD with the QEMU Simulator.

2. Installation

PathFinder-XD can be hosted under Windows™ or x86 based Linux and installation requires full administration privileges.

Please note: If you have PathFinder-XD already installed then please uninstall it before proceeding.

2.1 Windows[™] Installation

Run the SETUP.EXE program from the Windows directory on the supplied CD (or download) and follow the on-screen instructions.

2.2 Linux Installation

Run the ./SETUP32 (32-bit Linux) or ./SETUP64 (64-bit Linux) program from the supplied CD (or download) and follow the on-screen instructions. PathFinder-XD for MIPS is tested on the following Linux platforms:

- Fedora 13/Ubuntu 10.04 LTS 32-bit/64-bit versions
- Ubuntu 08.04 LTS 32-bit/64-bit (requires a specific version of PathFinder-XD)

Please note that the 64-bit Linux version of PathFinder-XD for MIPS requires the 32-bit library package ia32libs library, hence, make sure this is installed in your system. For example, to install on Ubuntu/Debian, issue the following command:

> \$sudo apt-get install ia32-libs

3. Using PathFinder-XD with the QEMU simulator

In this section we will look at using PathFinder-XD with the QEMU simulator. Ashling provide a binary (executable) version of the QEMU simulator built for the MIPS architecture installed and ready to run. To use QEMU with PathFinder-XD complete the follow the following steps:



- 1. Run PathFinder-XD
- 2. PathFinder-XD will then load as follows:



Figure 2. PathFinder-XD for MIPS™ loading

If this is your first-time running then you will be prompted to specify your Workspace (default directory for projects etc). Accept the default which is located in PathFinder-XD's installation directory.

3. In PathFinder-XD, create a New Target Configuration via the Target menu



Figure 3. Target Configuration

and select the **Debug using Simulator (QEMU)** option as shown below

| 2 New Connection | |
|--|--------------------|
| Select Remote System Type Please select the system type of the remote | system to connect. |
| System type: | |
| type filter text | |
| Ashling Debugging Debug using Debug Probe Debug using Simulator (QEMU) | |
| <u>4 Sack</u> | lext >EinishCancel |

Figure 4. Selecting Remote System Type

4. Click **Next** and we can now configure our QEMU settings as shown below:

| Initial target byte order Port(prefix -p if using older versions of QEMU) Other QEMU invocation parameters | | |
|--|--|--------|
| Simulator | | |
| | | |
| QEMU configuration | | |
| Simulator QEMU configuration QEMU executable Initial target byte order Port(prefix -p if using older versions of QEMU) Other QEMU invocation parameters | use default | Browse |
| Simulator QEMU configuration QEMU executable Initial target byte order Port(prefix -p if using older versions of QEMU Other QEMU invocation parameters | Little Endian | • |
| Simulator QEMU configuration QEMU executable Initial target byte order Port(prefix -p if using older versions of QEMU) Other QEMU invocation parameters | 1234 | |
| | -M malta -S | * |
| | | - |
| | • | 4 |
| | | |
| | | |
| | | |
| | on use default <u>Browse</u> rder Ing older versions of QEMU) 1234 -M malta -S | |
| | | |
| | | |
| | | |
| | | |
| (?) < <u>B</u> ack Next | > <u>F</u> inish | Cancel |
| | | |

Figure 5. QEMU Simulator Configuration

Settings include:

- **QEMU executable:** The actual binary file executed by PathFinder-XD when invoking QEMU. Usually not changed
- Initial target byte order: Required endianness
- Port: The TCP/IP port used by PathFinder-XD to communicate with QEMU. Usually not changed

 QEMU Invocation parameters: Allows you to configure QEMU. For more details on QEMU configuration, see the following links: http://wiki.gemu.org/download/gemu-doc.html#sec 005finvocation

Click Finish when done.

5. PathFinder-XD will now create a new QEMU Simulator setting in it's **Remote Systems** Window as shown below:

http://wiki.gemu.org/download/gemu-doc.html#MIPS-System-emulator

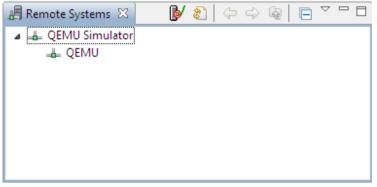


Figure 6. Remote Systems Window

Right-click on **QEMU** and click **Connect to** invoke QEMU. Once invoked, the **Remote Systems** window will update as follows:

| 📕 Remote Systems 🛛 | D 🗗 | \$ 3 | \Rightarrow | 6 | $\overline{}$ | |
|---------------------|-----|--------|---------------|---|---------------|--|
| | | | | | | |
| | | | | | | |

Figure 7. Remote Systems Window after QEMU connection

6. We can now download a program to QEMU by right-clicking over **QEMUSim** and selecting **Download and** Launch

| 📲 Remote Systems | × | 🕑 🖗 🗢 🕼 🖪 | ~ - 8 |
|---|---|---------------------|--------|
| ▲ ↓ QEMU Simulai ▲ ↓ QEMU ▲ ↓ QEMU ØEMU | | | |
| Contraction of the second | | Go To | + |
| | 8 | Refresh | |
| | Þ | Rename | F2 |
| | х | Delete | Delete |
| | V | Download And Launch | |
| | _ | Reset | + |

Figure 8. Download and Launch

PathFinder-XD includes a suitable program (C:\Program Files\Ashling\PathFinder-XDforMIPS\examples\LE\UnCached\Debug-LE\UnCached LE.elf) for running with QEMU.

| 2 QEMU Simulator |
|---|
| Modify attributes and launch |
| Name: QEMU Simulator |
| 📄 Main 🏇 Debugger 🦉 Source |
| Download |
| ELF (binary) path C:\Program Files\Ashling\PathFinder-XDforMIPS\examples\LE\UnCached\Debug-LE\UnCached_LE.elf Browse Load Options Symbols only Program and symbols Load (program and symbols) and verify (program) Use fast download (requires 512 bytes of RAM) RAM address OS Awareness Enable OS debugging Linux |
| Apply Reyert |

Figure 9. Specifying Target Program to Download

In the **Debugger** tab you can tell PathFinder-XD to stop at main() as shown below

|) Main 🕸 Debugger | 🤣 Source | |
|---------------------|---|-----------------|
| Stop on startup at: | main | |
| Debugger Options | | |
| Main | | |
| GDB debugger: | C:\Program Files\Ashling\PathFinder-XDforMIPS\GNUDebugger\mips-linux-gnu-gdb-qemu.exe | Browse |
| GDB command file: | .gdbinit | B <u>r</u> owse |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

Figure 10. Stopping at main()

When finished, press **Debug** to load your program.

7. PathFinder-XD will now download the program to QEMU and update it's Windows as follows allowing you to start your Debug session:

| 💆 Debug - C:\Program Files\Ashling\PathFinder-XDforMIPS\examples | \LE\UnCached\src\startup.S - PathFinder-XD for | MIPS | | | |
|---|--|------------------|----------------------|----------------------|-----------------------------|
| <u>File Edit Linux Target R</u> un <u>W</u> indow <u>H</u> elp | | | | | |
| u ra 🗠 📝 🚏 💕 🗧 | | | | | 😭 🐂 Debug |
| 🕸 Debug 🛛 🥻 🕪 💷 | 🛯 🤽 👁 🗽 🍽 🛒 🏟 🏹 🗖 🗖 | °₀ Breakpoints ⊠ | 🔋 Memory 🚻 Regist | ers 🙀 Expressions | 🗶 💥 🔗 🖓 🔪 🔲 🕀 |
| QEMU Simulator [PathFinder-XD Embedded Debugging] UnCached_LE.elf If Thread/Core [0] (Suspended : User Request) If thread/Core [0] (Suspended : User Request) QEMUConsole Jeg QEMUConsole Jeg gdb | | | | | |
| | | | | | - |
| | | 4 | | | Þ |
| 📓 startup.S 🛛 | | - E | 🛛 🗱 Variables 🖾 🎬 | Disassembly | 1 🗤 🕸 🖇 🗶 🦌 🖬 🔽 🖓 🗖 |
| //// Routine: start | | * | Name | Туре | Value |
| // Initialises core | | | | | |
| // | | | | | |
| LEAF(_start) | | | | | |
| <pre>// 1. Initialise CPU registers</pre> | | | | | |
| or v0,zero, zero or v1,zero, zero | | | | | |
| or a0,zero, zero | | | | | |
| or a1,zero, zero | | | | | |
| or a2,zero, zero or a3,zero, zero | | | | | |
| or t0,zero, zero | | | | | |
| or t1,zero, zero | | | | | |
| or t2,zero, zero | | | | | |
| or t3,zero, zero or t4,zero, zero | | | | | • |
| or t5,zero, zero | | | | | |
| or t6,zero, zero | | | | | |
| or t7,zero, zero or s0,zero, zero | | | | | |
| or s1,zero, zero | | | | | |
| or s2.7ero. 7ero.∢ | | | 4 | | Ŧ |
| | | | | (m) | |
| | 📮 Console 🛛 🔳 🗮 🙀 📑 🚡 | | <u>* 💷 • 📬 • " 🗖</u> | Section File Browser | <u>∲</u> ⊞ ⊟ [□] □ |
| | QEMU Simulator [PathFinder-XD Embedded De The target is assumed to be little of | | | type filter text | |
| Z QEMUSIM | The carge is assumed to be fitter (| | | the United LT of | |
| Be demosini | | | | 🏇 UnCached_LE.e | 217 |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | 4 | | | | |
| | | | , | | |
| | | | | | |

Figure 11. PathFinder-XD after program download

8. You can now control execution (start, stop, step etc.) using the Debug bar:

| 莎 Debug | 8 | × | 8 | | | | 3 | P | .P | -0 | i⇒ | V | ¢ | | |
|----------------------------------|------|---|---|--|--|--|---|---|----|-----|------|-------|-----|-----|---------|
| | | | | | | | | | | | | | | | |
| Go | | | | | | | | | | | | | | | |
| Stop/H | lalt | | | | | | | | | | | | | | |
| Step Into, Over and Return (Out) | | | | | | | | | | | | | | | |
| Termir | | | | | | | | | | deb | ua s | essio | n m | ear | nina we |

Terminate (this button actually terminates the debug session meaning we have to **Download and Launch** again)

Figure 12. Execution Control

When setting/toggling breakpoints in the Source and Disassembly Windows, make sure the mouse pointer is hovering over the left-most column (known as the ruler) of the Window as shown below:

| .S | startup.S 🔂 r | main.c 🛛 | |
|----|-----------------------------|------------|---------------|
| | <pre>int main(void) {</pre> | | |
| ~ | unsigned | int iError | = NO_ERROR; |
| | char | cCommand | = NO_COMMAND; |
| | data_struct | sData; | |
| | ClearDispla | | |

Figure 12. Setting a breakpoint

Ashling Product Brief APB211 Page 7 of 35 9. To watch a variable or expression, select it using the mouse and Add Watch Expression via the right-mouse button menu as show below:

| strcpy (sz | zDis 1 | | | | | | | |
|--|------------|----------------------|---------------|------|----|-----|---|---|
| | | Revert File | | | | | | |
| ľ | | Save | Ctrl+S | | | | | |
| void ClearDisp { | olay | Show In | Alt+Shift+W ► | | | | | |
| int iIndex | G | Source | Alt+Shift+S ► | | | | | |
| < | _ | Surround With | Alt+Shift+Z ► | | | | | |
| Remote Systems | | Refactor | • | | | | | |
| 建 QEMU Simulat 교환 QEMU | tor | Declarations | + | | | | | |
| | Sim | References | + | | | | | |
| | ⇒] | Run to Line | Ctrl+R | | | | | |
| | 3 | Move To Line | | | | | | |
| | ъ | Resume At Line | | | | | | |
| | ¥+¥ ₹? | Add Watch Expression | | | | | | |
| | ₽. | PC at CurrentLine | | | | | | |
| | | Preferences | | | | | | |
| Figur | ~e 13. A | dding a Watch Exp | oression | | | | | _ |
| ି⊚ Breakpoints 📋 Memory 🐰 Registers ରିଙ୍କି Expre | essions & | 3 | 🏝 📲 | + 3 | ۰. | 🔁 I | 2 | ę |

Name : szDisplay

. .

 $\overline{\nabla}$

| | туре | value | Details: "Writing to A | completed \\000" |
|---|--------------------|----------------|------------------------|--------------------|
| | char [50] | 0xa0004050 | Default:0xa0004050 | compileceu.(\000 , |
| F | igure 14. Expressi | on window show | ing watched expression | |

Value

Expression

b 🥭 szDisplay

Туре

You can also quickly watch an expression by hovering the mouse pointer over it as shown below: strcpy (szDisplay, szLocalString);

| | зстеру (| SZUISPINY, SZLOCAISCI ING | /3 | | |
|----|----------------------------------|---------------------------|-----------------------|--------------------------------------|---------|
| 20 | 3 | Expression | Туре | Value | |
| | | 🔺 🥭 szDisplay | char [50] | 0xa0004050 | |
| | void ClearDi | (x)= szDisplay[0] | char | 87 'W' | |
| | { int iInd | (x)= szDisplay[1] | char | 114 'r' | |
| | for (iIn | (A) an Disular (2) | char. III | 105.0 | · · · · |
| .8 | szDi } « Remote Systems | Default:0xa0004050 | A completed.\\000", ' | ' <repeats 25="" times=""></repeats> | E |
| | Remote Systems | HEX:0X00004050 | 000010000001010000 | | - |
| | 📲 QEMU | + | | | - F |
| | 👼 QEM | USim | | | |

Figure 15. Quick watch via mouse hover

10. PathFinder-XD supports a Console Windows (or Views) which can be opened from the Window menu:

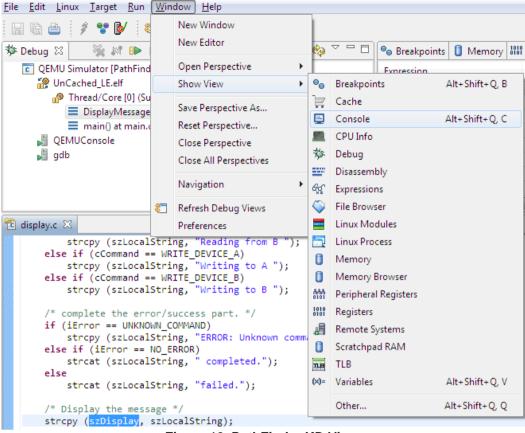


Figure 16. PathFinder-XD Views

Console. Allows you to enter debug commands and view their output. The GNU GDB syntax is fully supported. See here for details: <u>http://sourceware.org/gdb/current/onlinedocs/gdb/index.html</u> or for a handy quick-reference card see here: <u>http://users.ece.utexas.edu/~adnan/gdb-refcard.pdf</u>

| DEMU | Simulator [P | athFinder-XI | D Embedder | Debugging | il adb | | |
|------|--------------|--------------|-------------|-----------|----------|----------|----------|
| info | | oen neer se | o ennocence | | 11 9 0 0 | | _ |
| lino | zero | at | v0 | v1 | a0 | a1 | air |
| RØ | 00000000 | 00000001 | a0004050 | 00000000 | a0004068 | a0004228 | |
| | t0 | t1 | t2 | t3 | t4 | t5 | ti |
| R8 | 80808080 | 01010101 | 00000000 | 2e646574 | 00000000 | 00000010 | 00000800 |
| | s0 | s1 | s2 | s3 | s4 | s5 | si |
| R16 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 0000000 |
| | t8 | t9 | kØ | k1 | gp | sp | si |
| R24 | 80000800 | 9e190c8f | 00000000 | 00000000 | a0004110 | a0004200 | a000420(|
| | sn | 10 | hi | bad | cause | pc | |
| | 0040ff00 | 00000000 | 00000000 | 00000000 | 00800400 | a00003fc | |
| • | • | | | • | | - | |

Figure 17. PathFinder-XD Console showing the output of the info reg command

For example, to dump **16 w**ords of memory in hex format from 0xA0004200 enter the examine command as follows:

| x /16wx 0xA0 | 004200 | | | |
|--------------|------------|------------|------------|------------|
| 0xa0004200: | 0x00000000 | 0x00000000 | 0x74697257 | 0x20676e69 |
| 0xa0004210: | 0x41206f74 | 0x6f632020 | 0x656c706d | 0x2e646574 |
| 0xa0004220: | 0x00000000 | 0xa0004228 | 0x0000004 | 0xa0004238 |
| 0xa0004230: | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| | | | | |

Console commands can also be stored in a text file (GDB script file) and executed from PathFinder-XD's **Run** menu.

4. Embedded Linux Debugging with PathFinder-XD and the QEMU Simulator

PathFinder-XD supports Embedded Linux Debugging for kernels based on v2.6 or later. Support works in two modes:

- Stop-mode: In stop-mode debugging, the whole system is halted (e.g. kernel and applications) whenever a breakpoint is taken.
- Run-mode: Run-mode debugging requires an application (GDB server) running on the QEMU simulator. In run-mode, the kernel continues to run when an application breakpoint is taken.

Stop-mode debugging is useful for bringing up the kernel as it allows debugging from reset. Stop-mode can also be used for process debugging, however, the kernel/interrupts etc. will not continue to run when halted (unlike run-mode). When stop-mode debugging a process, PathFinder-XD automatically scans the kernel MMU mapping for that process and sets up the MIPS core TLB to allow debug access to the process's memory area.

Run-mode debugging requires that the kernel is up and running and allows non-intrusive debug of a process (i.e. the kernel will continue to run even when a process is halted). Run-mode also supports thread-aware breakpoints and simultaneous debug of multiple processes.

This section demonstrates Linux Kernel Debugging using PathFinder-XD and the QEMU simulator running v2.6.32 of the Linux Kernel on a Windows host machine. A suitable, pre-built version of the v2.6.32 Linux Kernel and root file-system is also available for download from Ashling (http://www.ashling.com/support/MIPS/QEMU_LINUX/MIPS_QEMU_LINUX_v2.6.32.ZIP); this should be installed by unzipping on to your local hard-disk (ensure you preserve the directory structure as present in the ZIP file).

Ashling recommend a high-spec PC (e.g. dual-core with 4GB memory) for use with this example given the processing-power required to run Linux on the QEMU simulator.

4.1 Preparing for debugging

This section is only necessary if you are building/using your own Kernel; the version supplied by Ashling includes all of the following requirements.

4.1.1 Building with debug symbols

Your kernel, modules, processes, libraries, drivers etc. must be built with debug symbols as PathFinder-XD needs to access global structures and variables etc. to support Linux debugging. *Please note* that debug symbols for Linux kernel (vmlinux) are required to debug user-mode applications in stop-mode (to allow PathFinder-XD to handle memory mapping which requires kernel symbols). Kernel symbols are not required for run-mode debugging.

- For the kernel, run make menuconfig, select Kernel hacking, enable Kernel debugging and Compile the kernel with debug and run make to rebuild the kernel with debug symbols.
- For non-kernel items, add the compiler gcc switch -g (which will generate debug symbols) to your makefile and rebuild.

4.1.2 Compiler optimisations

Compiler optimisations should not be used as they can cause misalignment between the generated symbolic information and the actual generated machine code thus causing problems with debugging. In particular, the flag --ffunction-sections should not be used as it will create .text sections for every function causing problems for PathFinder-XD. See here for more details: <u>http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html</u>. To remove these optimizations, change your makefile and rebuild e.g.:

Change arch/mips/Makefile

from:

```
cflags-y := --ffunction-sections (Line number 51 in Linux kernel 2.6.27)
```

to:

#cflags-y := --ffunction-sections

4.1.3 On-demand paging

Linux uses "on-demand paging" meaning that a process's (and its dependant libraries) code, data and stack are not actually paged into memory until they are first used. This can cause problems when you wish to "stop-mode" debug a process from its initialisation as it may not yet be present in memory. For example, you cannot set software breakpoints which require patching of the software breakpoint instruction into the appropriate process's memory location until the actual associated process code page is in memory. Depending on the size of your target's memory space and your memory management unit (MMU configuration), you may or may not have this issue. If you do then Ashling provide a kernel patch that will force all of a process's code, data and stack pages into memory. This file is installed with PathFinder and is called ash_load_process_pages.c. Installing the patch requires that you modify some existing kernel files and rebuild; please refer to the file for full details. *Note that this patch is required only for stop-mode debugging*.

4.2 Stop-mode Debugging

The following features are supported:

- Linux Kernel debugging:
 - Debug modules built as part of the Kernel
- Linux dynamically loadable Modules/Driver debugging:
 - List all inserted modules
 - o Debug an already inserted module
 - Debug a module from init_module()
 - Linux process (application) and library debugging:
 - List all running processes and threads
 - Debug a running process
 - Debug a process from main ()
 - Debug shared libraries

4.2.1 Sample Stop-mode Linux Debugging Session

This section demonstrates Linux Kernel Debugging using PathFinder-XD and the QEMU simulator running v2.6.32 of the Linux Kernel.

4.2.1.1 Invoking PathFinder-XD and connecting to the QEMU simulator

Invoke PathFinder and configure for the QEMU simulator as shown in section **3**. The **QEMU Invocation parameters** need to be modified to load the Linux kernel as follows:

-M malta -kernel <drive>:\<path>\vmlinux -hda <drive>:\<path>\rootfs.ext2 -append "root=/dev/hda rw"

Where $<drive>:<path>\$ is the location of your Linux kernel. In our walk-through, we assume this is c:\MIPS_QEMU_LINUX_v2.6.32\.

For example:

| QEMU configuration | | |
|---|--|-----|
| QEMU executable | use default Brows | e |
| Initial target byte order | Little Endian | • |
| Port(prefix -p if using older versions of QEMU) | 1234 | |
| | -M malta -kernel c:\MIPS_QEMU_LINUX_v2.6.32\vmlinux -hda c:\MIPS_QEMU_LINUX_v2.6.32\rootfs.ext2 -append "root=/dev/hda rw" | * |
| Other QEMU invocation parameters | | - |
| | 4 | - F |

Figure 18. QEMU invocation parameters for Linux debugging

Please note that if you need to debug the kernel from start up, the "-s" option should be appended with the invocation parameters as follows:

-M malta -kernel <drive>:\<path>\vmlinux -hda <drive>:\<path>\rootfs.ext2 -append "root=/dev/hda rw" -S

These invocation parameters ensure the Linux kernel (vmlinux) is loaded to the simulator and supplied root file system (rootfs) is used. Invocation parameters include:

- -kernel This option loads the Linux kernel (vmlinux) file to the QEMU simulator
- -hda This option mounts a hard disk image (rootfs.ext2) for the simulator. This
 - hard disk contains the root file system (rootfs) required for the kernel to load.
- -append This option passes the kernel arguments.

4.2.1.2 Loading kernel symbol information to PathFinder-XD and executing the kernel image

Connect to the simulator (right-click **QEMU** and select **Connect**), right-click over **QEMUSim** and select **Download and Launch** as follows:

| 📕 Remote Systems 🛛 | 💕 🗞 (-> 🖗 1 | |
|--------------------|---------------------|--------|
| 🔺 📲 QEMU Simulator | | |
| ⊿ _⊉ QEMU | | |
| QEMU ¹⁵ | Go To | ۰ |
| 2 | Refresh | |
| Þ | Rename | F2 |
| × | Delete | Delete |
| ₿¢ | Download And Launch | |
| | Reset | • |
| | | |

Figure 19. Download and launch

Now, enable Linux debugging via the **Enable OS debugging** check box (this ensures that PathFinder-XD will add the **Linux** specific menu allowing you to perform Module and Process debugging).

In this example, our kernel binary image is already downloaded to the simulator (via the QEMU invocation), hence, we only need to select **Load symbols only** (for the kernel image), specify the **ELF(binary) path** (c:\MIPS_QEMU_LINUX_v2.6.32\vmlinux) and press **Debug.** This will load the kernel symbols into PathFinder-XD to allow symbolic kernel debug.

| 2 QEMU Simulator | × |
|--|--------------|
| Modify attributes and launch | Ť. |
| Name: QEMU Simulator | |
| 📄 Main 🕸 Debugger 🧤 Source | |
| Download | |
| ELF (binary) path C:\MIPS_QEMU_LINUX_v2.6.32\vmlinux | Browse |
| Load Options | |
| Symbols only | |
| Program and symbols | |
| Load (program and symbols) and verify (program) | |
| RAM address | |
| OS Awareness Image: Stress | |
| | Apply Revert |
| ? | Debug Close |

Figure 20. Loading the kernel symbols

Run the simulator using button and the kernel will boot in a seperate QEMU shell as shown below (this may take a few minutes depending on your PC speed):

OEMU - - pcnet32: 1 cards_found. serio: i8042 KBD port at 0x60,0x64 irq 1 serio: i8042 AUX port at 0x60,0x64 irq 12 mice: PS/2 mouse device common for all mice rtc_cmos rtc_cmos: rtc core: registered rtc_cmos as rtc0 rtc0: alarms up to one day, 242 bytes nuram rtc-test rtc-test.0: rtc core: registered test as rtc1 rtc-test rtc-test.1: rtc core: registered test as rtc2 TCP cubic registered NET: Registered protocol family 17 NET: Registered protocol family 15 rtc_cmos rtc_cmos: setting system clock to 2013-01-14 14:27:30 UTC (1358173650) input: AT Raw Set 2 keyboard as /class/input/input0 input: ImExPS/2 Generic Explorer Mouse as /class/input/input1 EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended VFS: Mounted root (ext2 filesystem) on device 3:0. Freeing prom memory: 956k freed Freeing unused kernel memory: 168k freed Initializing random number generator... EXT2-fs error (device hda): ext2_lookup: deleted inode referenced: 598 Remounting filesystem read-only EXT2-fs error (device hda): ext2_lookup: deleted inode referenced: 598 Remounting filesystem read-only read-only file system detected...done Insert the floppy module... Floppy drive(s): fd0 is 1.44M FDC 0 is a S82078B Welcome to Ashling QEMU Linux ashqemu login:

Figure 21. QEMU Linux shell

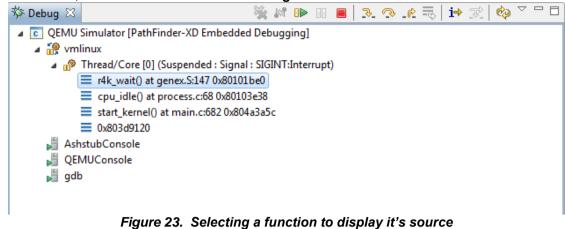
You can login in to the kernel using root (no password required). Once the kernel is booted, we can halt it within

```
PathFinder-XD (by pressing Used Stop/Halt). PathFinder-XD then updates as follows:
```

| Debug - C:\MIPS_QEMU_LINUX_v2.6.32\src\linux-2.6.32.14\arch\mip | s\kernel\genex.S - PathFinder-XD for MIPS | | | | |
|---|--|--|--|------------------|--------------------------------|
| <u>File Edit Linux Target Run Window H</u> elp | | | | | |
| 🔛 🖻 🗁 🕴 🖋 💕 😂 | | | | | 😭 🎦 Debug |
| 🕸 Debug 🛛 🧏 👘 🗈 🛙 | I 🔍 👁 Le 🔜 🖬 😒 🏟 🏹 🗖 | Sreakpoints ⋈ | Memory III Registers | Sec Expressions | 🗶 💥 🤐 🖓 🔪 🕞 😫 🗸 |
| ▲ CEVU Simulator (PathFinder-XD Embedded Debugging) ▲ Montonia ▲ Thread/Core [0] (Suspended : Signal : SIGINT:Interrupt) ■ rik_wait() at genes.SI47 0.800101e8 ■ cpu_idle() at process.c58 0.80010288 ■ start_kernel() at main.c682 0.804a3a5c ■ 0.80308102 ■ AthstubConsole ■ QEMUConsole ■ gdb | | | | | |
| 🔁 calibrate.c 🛛 🔒 genex.S 🕱 | | | 😡= Variables 🙁 🚟 Dis | assembly | £; +i • 0 # × % □ e' ▼ □ □ |
| <pre>nop nop .set mips3 wait /* end of rollback region (the region size must .set pop 1: END(r4k_wait) .macro BUIL_ROLLBACK_PROLOGUE handler FEXPORT(rollback_Vhandler) .set push .set noat NFC0 k0, CP0_EPC PTR_LA k1, r4k_wait ori k0, 0xif /* 32 byte rollback region */ xori k0, 0xif /* 32 byte rollback region */ xori k0, 0xif /* 32 byte rollback region */ NFC0 k0, CP0_EPC 9: 4</pre> | be power of two) */ | | Name | Type | Value |
| 📕 Remote Systems 🛛 🛛 🕑 🐑 🗘 🗇 🎝 👰 📄 🍸 🗖 🗖 | Console 🛛 | | • | File Browser 🖂 | 🔗 🖬 🗖 🖓 |
| ag tellinde systems so | <pre>GENUSing coll GENUSing and the stark of the stark of the stark of the stark stark of the stark of the stark of the stark of the stark stark of the stark of the stark of the stark of the stark stark of the stark of the stark of the stark of the stark stark of the stark of the stark of the stark of the stark stark of the stark of the stark of the stark of the stark stark of the stark of th</pre> | bugging] gdb traces ad 1 ="0",addr="0x80103 addr="0x80103238", ="0x804a3a5c",func func="??"}] ad 1 0 3 ="0",addr="0x80101 addr="0x80103238", ="0x804a3a5c",func | be0",func="r4k func="cpu_idle ="start_kernel be0",func="r4k func="cpu_idle | type filter text | |
| | | | , | | |

Figure 22. PathFinder-XD after halting the kernel

Ashling Product Brief APB211 Page 14 of 35 If no source is shown, then select a function in the **Debug** window



which will open a new tab in the source window allowing you to select Edit Source Lookup Path... button.

| 📧 r4k_wait() at arch/mips/kernel/genex.S:147 0x80101be0 🛛 | |
|---|--|
| Can't find a source file at "arch/mips/kernel/genex.S" Locate the file or edit the source lookup path to include its location. | |
| View Disassembly | |
| Locate File | |
| Edit Source Lookup Path | |
| | |
| | |
| | |

Figure 24. Edit Source Lookup Path

Specify your source-directory as shown below (make sure Search subfolders is selected)

| 💆 Add Source | X |
|---|--|
| Add a container to the source lookup path A directory in the local file system | *8 |
| 💆 Add File System Directory | × |
| File system folder Specify folder and whether subfolders should be sear | ched to the design of the desi |
| Directory: C:\MIPS_QEMU_LINUX_v2.6.32\src Search subfolders | <u>B</u> rowse |
| ? | OK Cancel |
| ? | OK Cancel |

Figure 25. Edit File System Directory

Ashling Product Brief APB211 Page 15 of 35 Notice the following additional windows in PathFinder-XD:

| SFile Browser 🛛 | 🔗 🕀 🕞 🗖 |
|-------------------|----------|
| type filter text | |
| 🔺 🕸 vmlinux | ^ |
| b lighuid.h | |
| b is module.c | |
| b is malta-time.c | |
| b imewait_sock.h | |
| b c genhd.h | |
| 🖻 🖻 mii.h | |
| ⊳ 💼 ide.h | |
| keyboard.c | |
| pgtable-32.h | |
| N 🖪 do scalh c | • |

Figure 26. File Browser view showing all kernel source-files

| 🍑 File Browser 📘 | Linux Modules | 🛛 🔗 🍫 🗆 🗆 |
|------------------|---------------|-----------|
| Address | Name | Size |
| 0xC005EC60 | floppy | 56159 |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

Figure 27. Linux Module view showing all currently loaded kernel modules (enabled via Linux menu)

| 🍑 File Browse | er 🧮 l | Linux Mod | lules 🔂 Linux Pr | ocess 🛛 | & & | |
|---------------|--------|-----------|------------------|---------|----------|---|
| Address | PID | Core Id | Command | ASID | Status | _ |
| 0x8047C600 | 0 | 0 | [swapper] | 0x0 | Running | |
| 0x8781BB68 | 1 | 0 | init | 0x5a | Sleeping | = |
| 0x8781B6E0 | 2 | 0 | [kthreadd] | 0x0 | Sleeping | |
| 0x8781B258 | 3 | 0 | [ksoftirqd/0] | 0x0 | Sleeping | |
| 0x8781ADD0 | 4 | 0 | [events/0] | 0x0 | Sleeping | |
| 0x8781A948 | 5 | 0 | [khelper] | 0x0 | Sleeping | |
| 0x8783FB68 | 8 | 0 | [async/mgr] | 0x0 | Sleeping | |
| 0x8783F6E0 | 9 | 0 | [pm] | 0x0 | Sleeping | |
| 0x878ED6E0 | 72 | 0 | [sync_supers] | 0x0 | Sleeping | |
| 0x878FCDD0 | 74 | 0 | [bdi-default] | 0x0 | Sleeping | • |

Figure 28. Linux Process view showing all processes (enabled via Linux menu)

Full kernel source-level debug is now possible.

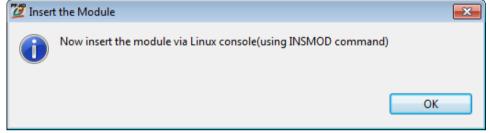
4.2.1.3 Debug a module from init module()

Use the Linux|Modules|Debug A Module From Initialisation menu to debug a module from its init module() entry point as follows:

| PF. | 🖉 Debug A Mod | ule From Initialisation | | — ——————————————————————————————————— |
|-----|---------------|--|-------|--|
| | | | | |
| | Module name | C:\MIPS_QEMU_LINUX_v2.6.32\src\moduletest\chardev.ko | | Browse |
| | | | | |
| | | | | |
| | | | Debug | Cancel |
| | | | | |

Figure 29. Specifying the module to debug

Once specified, you now need to insert the module via the shell as follows:



Press OK and PathFinder will run Linux allowing you to enter a shell command as follows:

| # insmod chardev.ko chardev: module license 'unspecified' taints kernel. Disabling lock debugging due to kernel taint | | |
|---|--------------------------|------------------------------|
| L | # insmod chardev.ko | |
| Disabling lock debugging due to kernel taint | chardev: module license | 'unspecified' taints kernel. |
| producting fock acougging and to kerner carne | Disabling lock debugging | due to kernel taint |

Figure 30. Inserting (running) the module

PathFinder-XD then halts the module at init_module() allowing module debug. You may have to specify the location of your source-file as before (C:\MIPS QEMU LINUX v2.6.32\src\moduletest)

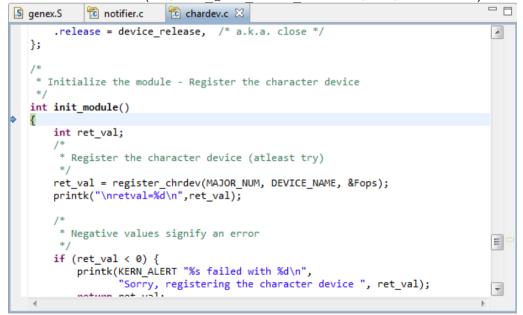


Figure 31. Module source

PathFinder-XD's File Browser will also update to show the source files associated with the module:

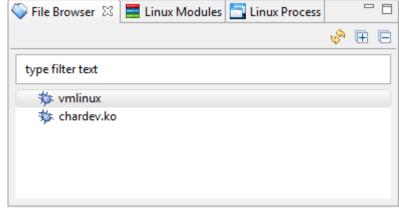


Figure 32. File Browser view showing modules sources

And the **Linux Modules** window will now list the new module:

| Sile Browser | 🧮 Linux Mod 🛛 | 🔄 🛅 Linux Proc 🖓 🗖 |
|--------------|---------------|--------------------|
| | | S 4 |
| Address | Name | Size |
| 0xC007C870 | chardev | 2781 |
| 0xC005EC60 | floppy | 56159 |
| | | |
| | | |
| | | |
| | | |
| | | |

Figure 33. Linux Modules window listing the new module

You can also view the internal module kernel structures via the right-mouse button menu as follows:

| Address | Name | Size | | |
|--------------------------|-----------|---------------------------------------|--|--|
| 0xC007C870 0xC005EC60 | ch flo | Module Structure Aodule Symbol | | |
| | | | | |

| [⊜] ⊜ Breakpoints | Memory | 1010 Registers | ର୍ଟ୍ଟ Expre | essions 🛛 | | | | | | | E |
|--------------------------------------|---------------|-------------------|-------------|---------------------|-----------------|-----------|-----|---|---|-----|---|
| | | | | <i>R</i> |) ⇒t i | \bullet | × 🖗 | 🖸 | 1 | ¢ | Þ |
| Expression | Ту | pe | | Value | | | | | | | 4 |
| ▲ ◆ (struct module*)0x (×)= state | | uct module * | | 0xc007c870 |) | | | | | | |
| | | enum module_state | | MODULE_STATE_COMING | |]_ | | | | | |
| 👂 🥭 list | str | uct list_head | | {} | | | | = | | | |
| 👂 🥭 name | ch | ar [60] | | 0xc007c87c | : | | | | | | |
| 👂 🥭 mkob | j str | uct module_k | object | {} | | | | | | | |
| 👂 🔶 modir | nfo_attrs str | uct module_at | tribute * | 0x87a6c480 |) | | | | | | |
| > + versio | n co | nst char * | | 0x0 | | | | | | | |
| b srcver | rsion co | const char * | | 0x8799b880 | 0 "FB2A6DBDC7D5 | | | | | | |
| b holde | rs_dir str | uct kobject * | | 0x879b8480 |) | | | | | | |
| b syms | со | nst struct kern | el_sym | 0x0 | | | | | | | |
| b crcs | co | nst long unsig | ned int * | 0x0 | | | | | | | |
| (×)= num_ | syms un | signed int | | 0 | | | | | | | |
| ⊳ 🔶 kp | str | uct kernel_par | am * | 0x0 | | | | | | | |
| (×)⊧ num_ | kp un | signed int | | 0 | | | | | | | |
| (×)= num_ | gpl_syms_un | signed int | | 0 | | | | | | | 1 |
| s 🛋 anl o | /mc _ co | net etruct kern | el cum | 0~0 | | | | Ŧ | • | - P | |

Figure 34. Viewing the internal kernel module structures

In addition, you can load module symbols for a module that is already loaded (Load Module Symbol menu option in the right-mouse button menu) as follows:

| | | | | \$ ~ |
|--------------|-----|-----------------------|---------------|------|
| Address | Nar | ne | Size | |
| 0xC007C870 | حطي | rdav | 2701 | |
| 0xC005EC60 f | | Show Module Structure | | |
| | | Load M | lodule Symbol | |
| | _ | | | |
| | | | | |
| | | | | |

Figure 35. Viewing the internal kernel module structures

4.2.1.4 Debugging a process from main()

Use the Linux|Processes|Debug A Process From main() to debug a process from it's entry point (this feature is only available in stop-mode i.e. when the kernel is halted) as follows:

| 💆 Debug A Proce | ess From main() | 3 | | | | |
|--|--|---|--|--|--|--|
| Process name rootfs Directory | C:\MIPS_QEMU_LINUX_v2.6.32\src\ashtestapp\ashtestapp Browse C:\MIPS_QEMU_LINUX_v2.6.32\rootfs Browse | | | | | |
| Shared library Shared Library Path(s) Add Remove | | | | | | |
| Debug Cancel | | | | | | |
| | Figure 36. Debugging a process from main() | | | | | |

Ashling Product Brief APB211 Page 19 of 35 **rootfs Directory** specifies where the root file-system (rootfs) resides in your host machine. This location is needed for loading shared library symbols in PathFinder-XD. Once specified, you now need to run the process from the shell as follows:



Press OK and PathFinder will run Linux allowing you to enter a shell command as follows:

| Ħ | ./ashtestapp |
|---|--------------------------------|
| | Figure 37. Running the process |

PathFinder-XD then halts the process at main() function as shown below:

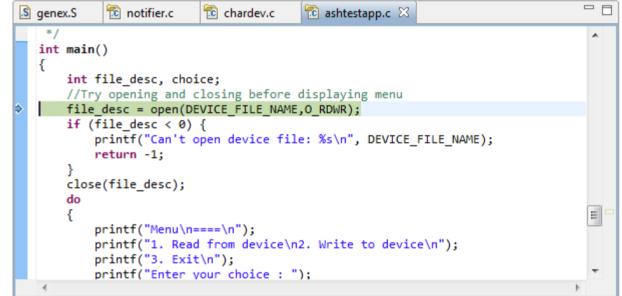


Figure 38. PathFinder-XD halted at the process's main() function

The File Browser window will update to show the process's source-code.

Please note: When using the QEMU simulator it is not possible to list processes whilst debugging a process in stop-mode (e.g. in PathFinder's Linux Process Window); this restriction is not present when debugging a process in run-mode or when using hardware based target debugging (e.g. Opella-XD).

4.2.1.5 Debugging a running process

You can load the symbols for a running process via the Linux Process window. Right-click on the process and select Load Process Symbol:

| 💯 Load Symbols for p | process ashtestapp | × |
|---|---|--------|
| Process symbol file rootfs Directory | C:\MIPS_QEMU_LINUX_v2.6.32\src\ashtestapp\ashtestapp C:\MIPS_QEMU_LINUX_v2.6.32\rootfs | Browse |
| Shared library Shared Library Path() | | Add |
| | | Remove |

Figure 39. Loading a process's symbols

4.2.1.6 Library debugging

Debugging of libraries is handled seamlessly without any extra requirements/setup.

4.3 Run-mode Debugging

Run-mode debugging is done via a "simulated" Serial/Ethernet interface and requires an application (GDB server) running on the simulator. In run-mode, the kernel continues to run when a process (application) breakpoint is taken. Run-mode debugging requires that the kernel is up and running and allows non-intrusive debug of process (i.e. the kernel will continue to run even when a process is halted).

4.3.1 Preparing for Run-mode Debugging

For run-mode debugging, a simulated Ethernet connection needs to be established between the host machine and the simulator (QEMU). This section explains how to configure a networking connection between the host OS and simulator so that run-mode debugging can be done through Ethernet.

4.3.1.1 Setting up networking between host and QEMU simulator

QEMU uses TAP (<u>http://en.wikipedia.org/wiki/TUN/TAP</u>) interfaces to provide full networking capability with the host OS. TAP simulates an Ethernet device entirely in software.

The Windows host OS does not provide a TAP adapter, hence, Windows users will need to download and install a TAP adapter. In our example we will use OpenVPN which can be downloaded from http://openvpn.net/index.php/open-source/downloads.html. Running as administrator, download and install OpenVPN (selecting all options). Once installed, you will have a new Network Adapter (under Network Connections). Configure its IP address via the right-mouse Properties menu as follows:

| 📮 openvpn Properties | | | | | | | | |
|---|--|--|--|--|--|--|--|--|
| Networking Sharing | | | | | | | | |
| Connect using: | | | | | | | | |
| TAP-Win32 Adapter V9 | | | | | | | | |
| Configure | | | | | | | | |
| This connection uses the following items: | | | | | | | | |
| ✓ Client for Microsoft Networks ✓ □ □ QoS Packet Scheduler | | | | | | | | |
| File and Printer Sharing for Microsoft Networks | | | | | | | | |
| ✓ - Internet Protocol Version 6 (TCP/IPv6) | | | | | | | | |
| ✓ Internet Protocol Version 4 (TCP/IPv4) ✓ Ink-Laver Topology Discovery Mapper I/O Driver | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| Install Uninstall Properties | | | | | | | | |
| Description | | | | | | | | |
| Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks. | | | | | | | | |
| | | | | | | | | |
| OK Cancel | | | | | | | | |

Figure 40. TAP Adapter

| Internet Protocol Version 4 (TCP/IPv4) Properties | | | | | | | | |
|---|---------------------|--|--|--|--|--|--|--|
| General | | | | | | | | |
| You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings. | | | | | | | | |
| Obtain an IP address automatically | | | | | | | | |
| • Use the following IP address: | | | | | | | | |
| IP address: | 192.168.10.1 | | | | | | | |
| Subnet mask: | 255 . 255 . 255 . 0 | | | | | | | |
| Default gateway: | | | | | | | | |
| Obtain DNS server address autom | natically | | | | | | | |
| • Use the following DNS server add | resses: | | | | | | | |
| Preferred DNS server: | | | | | | | | |
| Alternate DNS server: | · · · | | | | | | | |
| Validate settings upon exit | Ad <u>v</u> anced | | | | | | | |
| | OK Cancel | | | | | | | |

Figure 41. TAP adapter properties

In our example, we will assign the address 192.168.10.1 as shown above. In **Network Connections**, rename (via right-click mouse menu) the TAP network adapter **openvpn** (note this is case-sensitive, hence, make sure you specify "openvpn")

4.3.1.2 Modify QEMU invocation parameters

We now need to modify QEMU invocation parameters to ensure QEMU creates a new virtual Network Interface Card (NIC) and that it connects to the TAP adapter running on our Windows host.

The **QEMU Invocation parameters** need to be modified to as follows:

-M malta -kernel C:\MIPS_QEMU_LINUX_v2.6.32\vmlinux -hda C:\MIPS_QEMU_LINUX_v2.6.32\rootfs.ext2 -append "root=/dev/hda rw ip=192.168.10.2" net nic -net tap,ifname=openvpn

Where:

-net nic will create a virtual NIC

-net tap, ifname=openvpn will ensure that TAP interface is connected to the openvpn TAP adapter installed on the host

Note in our example above we have assigned 192.168.10.2 as the IP address of the virtual Linux machine running on the QEMU simulator. The above invocation will also work for Stop-mode debugging as outlined in the previous section.

Copy and Paste Warning! Do not copy and paste the above into PathFinder-XD; it will not work due to PDF issues. The above invocation line can be copied and pasted from the README.TXT supplied with PathFinder-XD.

4.3.2 Sample Run-mode Linux Debugging Session

This section demonstrates Linux Run-mode Process Debugging using PathFinder-XD and the QEMU simulator running v2.6.32 of the Linux Kernel. The example will demonstrate debugging of a Process and a Module (that contains functions called from the Process). As before we have to prepare our kernel for debug, download it to the simulator, execute it and load the kernel symbols into PathFinder. See previous sections **4.1**, **4.2.1.1** and **4.2.1.2**. Once these steps are complete we are ready to begin debugging our Module and Process as follows:

4.3.2.1 Debugging the Module and Process

1. First we load the Module (using insmod) from our Linux shell as follows:

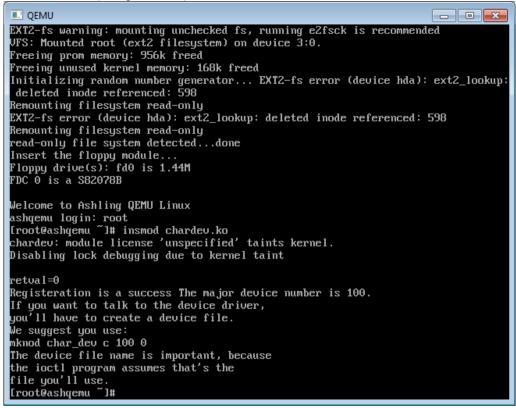


Figure 42. Loading the Module to be debugged

Note: do not attempt to load a module twice or debugging will not work correctly (use rmmod chardev.ko if you need to remove or unload the module)

2. Now, we halt the kernel in PathFinder-XD and load the Module symbols from within the PathFinder-XD Linux Modules window:

| Address | Name | Size | |
|------------|---------------------|-------------|-----------|
| 0xC007C870 | charde ₁ | 2701 | |
| 0xC005EC60 | floppy | Show Module | Structure |
| | | Load Module | Symbol |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

| 💆 Load Symbols Of A Module | | | × |
|----------------------------|--|----|--------|
| | | | |
| Module name (without .ko) | chardev | |] |
| Module symbol file | C:\MIPS_QEMU_LINUX_v2.6.32\src\moduletest\chardev.ko | | Browse |
| | | | |
| | | ОК | Cancel |
| | | | |

Figure 43. Loading the Module symbols

Ashling Product Brief APB211 Page 24 of 35

Ashling Product Brief APB211 Page 25 of 35 3. Notice how the File Browser now shows the Module and Kernel symbols:

| Serie Browser | Linux Modules | ę. | Ŧ | |
|------------------|---------------|----|---|--|
| type filter text | | | | |
| ⊳ 🕸 vmlinux | | | | |
| b 🏇 chardev.ko | | | | |
| | | | | |

Figure 44. File Browser showing Kernel and Module symbols

We can double-click on the Module to list the files and double-click on a source-file to show it in the Source Window. In the below screen-shot we have opened the Module source-file (chardev.c) and set a breakpoint at the function device_write which we wish to debug (i.e. this function located in the Module is called from the the Process)

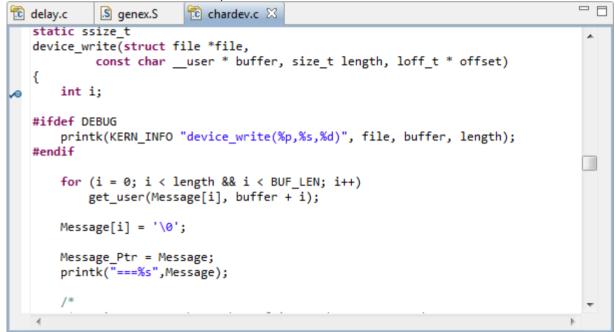


Figure 45. Setting a Breakpoint in the Module

4. Next, we resume execution of the Kernel in PathFinder-XD

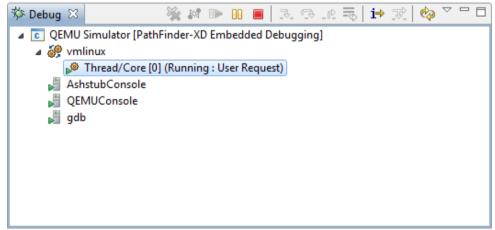


Figure 46. Running the Kernel

and launch gdbserver on the simulator (i.e. in the Linux shell) specifying the Process we wish to debug (ashtestapp). Notice how we tell ./gdbserver which port to listen on (1234)



Figure 47. Launching the Process

Ashling Product Brief APB211 Page 26 of 35 5. Now we need to **Debug A Process in Run-mode** using PathFinder-XD (the Kernel is now running) as follows:

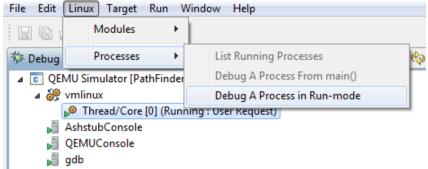


Figure 48. Debugging a Process in Run-mode

| | Ne need to specify the Process: | |
|---------|---|-----------------|
| 📄 Run-ı | ode Debugging Options 💱 Source | |
| Process | :\MIPS_QEMU_LINUX_v2.6.32\src\ashtestapp\ashtestapp | B <u>r</u> owse |

Figure 49. Specifying the Process

The location of the shared libraries:

| r Options | _ | | | | | |
|-----------------------------------|------------------|--|--|--|--|--|
| Shared Libraries | Connection | | | | | |
| ries: | | | | | | |
| C:\MIPS_QEMU_LINUX_v2.6.32\rootfs | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | Shared Libraries | | | | | |

Figure 50. Specifying the Share Library location

And finally, the connection mechanism (TCP in our example) and IP address of the simulator system (which is running gdbserver):

| Debugger Options | |
|--------------------------|--------------|
| Main Shared Libraries | Connection |
| Туре: ТСР 🔻 | |
| Host name or IP address: | 192.168.10.2 |
| Port number: | 1234 |
| | |

Figure 51. Specifying the Connection mechanism

Now press **Debug** to start debugging the Process

| 💆 Ashling TCP_Serial connection (QEMU) |
|--|
| Specify Attributes For Debugging A Process In Run-mode |
| Name: Ashling TCP_Serial connection (QEMU) |
| Run-mode Debugging Options 5 Source |
| Process C:\MIPS_QEMU_LINUX_v2.6.32\src\ashtestapp\ashtestapp Browse |
| Stop on startup at: main |
| Debugger Options |
| Main Shared Libraries Connection |
| GDB debugger: C:\Program Files\Ashling\PathFinder-XDforMIPS\GNUDebugger\mips-linux-gnu-gdb-qemu.exe |
| GDB command file: .gdbinit Browse |
| (Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".) |
| Non-stop mode (Note: Requires non-stop GDB) |
| Force thread list update on suspend |
| Automatically debug forked processes (Note: Requires Multi Process GDB) |
| |
| |
| |
| |
| Apply Re <u>v</u> ert |
| Debug Close |

Figure 52. Debugging a Process in Run-mode dialog

| Debug - C:\MIPS_QEMU_LINUX_v2.6.32\src\ashtestapp\ashtestapp | .c - PathFinder-XD for MIPS | | | | | |
|--|--|-----------------|------------------------------|--|---------------|---------------------------------------|
| ile <u>E</u> dit <u>L</u> inux <u>T</u> arget <u>R</u> un <u>W</u> indow <u>H</u> elp | | | | | | |
| 🖫 🗟 🔌 🛠 💕 🗧 | | | | | E | 🎦 Debug |
| 🌣 Debug 🛿 🧏 🕪 🔢 | 🔳 3. 👁 🕼 🔜 i+ 🛒 🏟 🍸 🗖 🗖 | 🗣 Breakpoints 🖾 | Memory 👭 Registers 🔅 | Expressions | 🗙 💥 🎯 🖓 1 | 🔌 🗄 🔄 🖓 🗖 |
| QEMU Simulator (PathFinder-XD Embedded Debugging) Wmlinux Thread/Core [0] (Running : User Request) AshstubConsole QEMUConsole gdb Ashhing TCP_Selal connection (QEMU) (PathFinder-XD Running) esthetapp [cores: 0] esthetapp [cores: 0] esthetapp [cores: 0] (Suspended : Breakpoint) main() at ashtestapp.c:127 0x400dfc gdb | rode Debugging] | ✓ A chardev.c | [ine 135] | | | |
| 🖻 delay.c 🛛 🕄 genex.S 🔀 chardev.c 🔂 ashtestapp.c | 8 | - 0 | 🗱 Variables 🖾 🎫 Disas: | embly | (in 🙀 🏟 🖉 🗶 🕅 | ≫ • • • • ▼ − |
| /* | | | Name | Туре | Value | |
| * Main - Call the ioctl functions | | | (x)= file_desc | int | 0 | |
| <pre>*/ int main()</pre> | | | (x)= the_desc (x)= choice | int | 2141647 | 7208 |
| <pre>int file_desc, choice; //Try opening and closing before displaying mu file_desc = open(DEVICE_FILE_NAME,O_RDWR); if (file_desc < 0 { printf("Can't open device file: %s\n", DET return -1; } close(file_desc); do { printf("Menu\n====\n"); printf("S. Read from device\n2. Write to openintf("S. Exit\n"); printf("S. Exit\n"); printf("Enter your choice : "); scanf("Ad",&folice); switch(choice) Remote Systems 23</pre> | /ICE_FILE_NAME); device\n"); | | | le Browser 23 | Linux Modules | · · · · · · · · · · · · · · · · · · · |
| Regence 2 Construction | Ashling TCP_Serial connection (QEMU) [PathFi | | bugging] gdb | | | • |
| _at: QEMU ∰ QEMUSim | | | | e filter text ☆ vmlinux ☆ chardev.ko | | |
| | | | - | | | |
| D° | · | 1 | | | | |

6. PathFinder-XD will now update as follows:

Figure 53. PathFinder-XD in Run-mode

If no source is shown then select the Edit Source Lookup Path... button

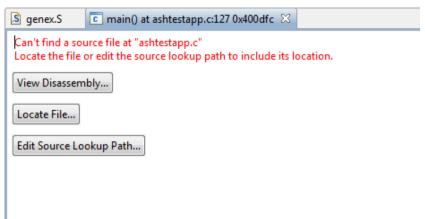


Figure 54. Edit Source Lookup Path

and specify your source-directory as shown below:

| 💆 Add File System Directory | — |
|--|--|
| File system folder Specify folder and whether subfolders should be search | ned to the second secon |
| Directory: C:\MIPS_QEMU_LINUX_v2.6.32\src | Browse |
| ✓ Search sub <u>f</u> olders | |
| ? | OK Cancel |

Figure 55. Edit File System Directory

Notice how:

• The **Debug** window show both the Kernel (Embedded Debugging) and Process (Run-mode Debugging) status:

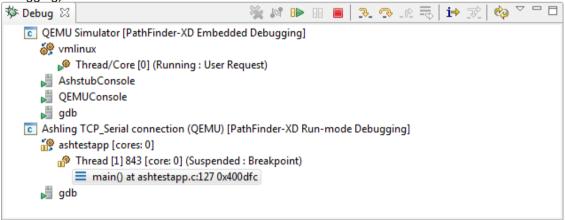


Figure 56. PathFinder-XD Debug Window showing Kernel and Process (Kernel Run-mode) status

• The File Browser shows the Module, Process and Kernel sources:

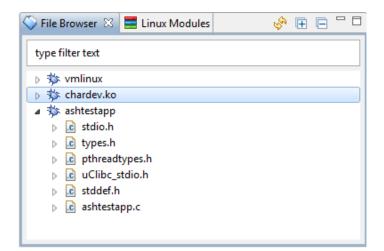


Figure 57. PathFinder-XD File Browser showing Module, Process and Kernel sources

• The Source window shows the source code for our Process from main()

```
💼 delay.c
             S genex.S
                          Chardev.c
                                         🛅 ashtestapp.c 🖾
    * Main - Call the ioctl functions
    */
   int main()
   {
       int file_desc, choice;
       //Try opening and closing before displaying menu
Ð
       file_desc = open(DEVICE_FILE_NAME,O_RDWR);
       if (file_desc < 0) {</pre>
           printf("Can't open device file: %s\n", DEVICE_FILE_NAME);
           return -1;
       }
       close(file_desc);
       do
       {
           printf("Menu\n====\n");
           printf("1. Read from device\n2. Write to device\n");
           printf("3. Exit\n");
```

Figure 58. Process Source

7. We can now debug our Process as normal with the Kernel running in the background. After running the process, choose the options as seen in the figure below:

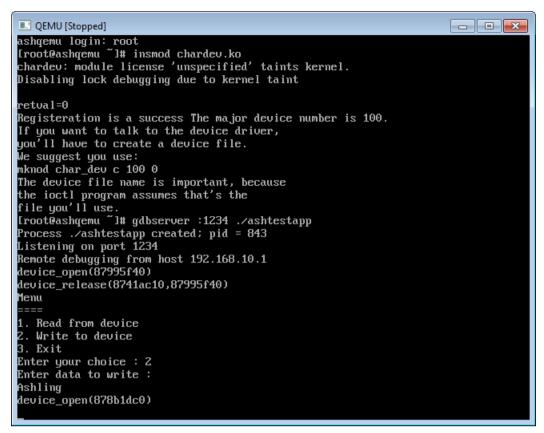


Figure 59. Running the process

This will invoke the functions with a breakpoint set and PathFinder-XD's Debug window will update as follows:

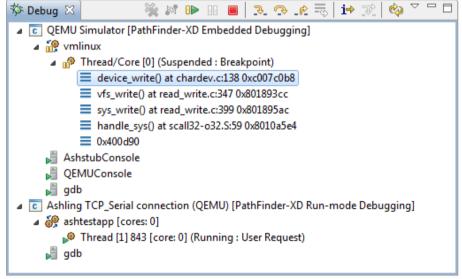


Figure 60. PathFinder-XD Debug Window showing the Kernel halted

Notice how the Kernel is now shown as halted (i.e. PathFinder-XD has automatically switched from run-mode to stop-mode as the kernel is halted due to the breakpoint in the Module). This demonstrates how PathFinder-XD easily switches between stop-mode and run-mode within the same debug session.

4.3.2.2 Debugging multi-threaded applications

Multi-threaded applications are supported in run-mode debugging only. All the application threads and the associated Call Stack for each thread are listed. In addition, it is possible to set thread specific breakpoints.

To debug a multi-threaded application in run-mode, complete the following steps:

1. Launch gdbserver from the simulator (i.e. in the Linux shell) specifying the application we wish to debug (threadtestapp).



Figure 61. Launching multi-threaded Process

 Now we need to Debug A Process in Run-mode using PathFinder-XD (the Kernel is now running) as follows:

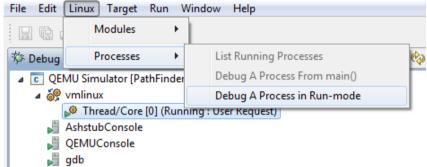


Figure 62. Debugging a Process in Run-mode

3. Select the application (Process) to debug

| Process | C:\MIPS_QEMU_LINUX_ | v2.6.32\src\threadtestapp\threadtestapp Browse Browse |
|---------|---------------------|---|
| Choos | e the location of | shared libraries: Debugger Options Main Shared Libraries Connection |
| | | Directories: C:\MIPS_QEMU_LINUX_v2.6.32\rootfs |
| | | |
| | | |

Figure 63. Specifying the Share Library location

5. And finally, the connection mechanism (TCP in our example) and IP address of the simulator (which is running gdbserver):

| Debugge | r Options | | | |
|---------|--------------------|--------------|---|--|
| | | | | |
| Main | Shared Libraries | Connection | | |
| Туре: | TCP - | | | |
| Host n | ame or IP address: | 192.168.10.2 | | |
| Port nu | umber: | 1234 |] | |
| | | | | |

Figure 64. Specifying the Connection mechanism

- 6. Now press Debug to start debugging the Process
- 7. The program will now run to the main() function.

| Debug - C:\MIPS_QEMU_LINUX_v2.6.32\src\threadtestapp\threadtest | tapp.c - PathFinder-XD for MIPS | | | | |
|--|--|------------------|--|--|--|
| <u>File Edit Linux Target Run W</u> indow <u>H</u> elp | | | | | |
| 🗐 🕼 🖆 🖸 📽 💕 🗧 | | | | | 😭 🎦 Debug |
| 🏇 Debug 🛛 🦓 🎶 🕪 💷 🛢 | I 🔍 👁 🕼 🔜 🖬 🕱 🔞 🏹 🗖 🗖 | ®₀ Breakpoints ⊠ | Memory 👭 Registers | ିଙ୍କ Expressions | 🗶 💥 🔗 🖓 🔪 🔳 🕀 🤤 🗸 🐂 |
| CEMU Simulator [PathFinder-XD Embedded Debugging] wininux Forread/Core [0] (Running : User Request) AchstubConsole gdb CEMUCOnsole gdb Ashling TCP. Serial connection (QEMU) [PathFinder-XD Run-moc thread(Estapp [core: 0] m Thread [1843] [core 0] (Suspended : Breakpoint) main() at threadtestapp.c:20 0x40083c gdb | de Debugging] | | | | |
| 🔞 delay.c 📓 genex.S 🛍 threadtestapp.c 🛛 | | - 8 | 🕪= Variables 🔀 🎫 Dis | assembly | , 🐗 🏟 🖇 💥 📑 😁 🖓 🗖 |
| <pre>#define _REENTRANT #include <pthickline <pre="">% for the state of t</pthickline></pre> | | | Name (0) th/1 (0) th/2 ⇒ msg1 ⇒ msg2 | Type pthread_t pthread_t const char* const char* | Value 716480640 2141666008 0x7fa78da8 ** 0x0 |
| | | 4 | • | | 4 |
| | | | <u> </u> | File Browser 🛛 | 🔗 🕀 🗖 🗖 |
| & QEMU Simulator & QEMU ₽ QEMUSim | QEMU Simulator [PathFinder-XD Embedded Dei | bugging] gdb | | type filter text ★ vmlinux ☆ threadtestapp | |
| □◆ | | | | | |

Figure 65. Program runs to main()

8. Set a breakpoint in threadfn () and run to that point, the Debug view updates as follows:

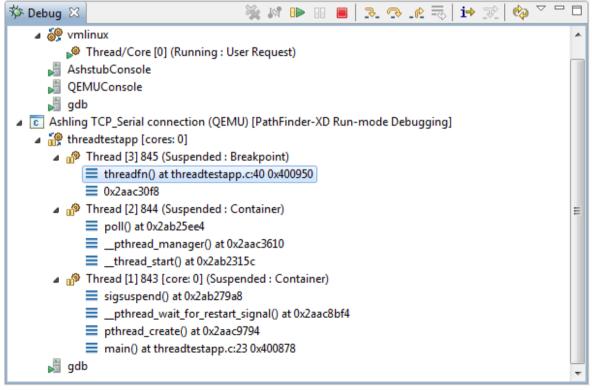


Figure 66. Multi-threaded Debug View

Notice how the threads are listed and the call stack for each thread is shown in the Debug view. While clicking on each thread context, all PathFinder-XD windows will update accordingly (i.e. thread specific).

To set a thread specific breakpoint:

- Set a breakpoint in a location by double clicking on the ruler.
- Right click the on the breakpoint in the ruler and choose Breakpoint Properties

| 1 | | | adtestapp.c 🛛 | | |
|---|----|---|---------------|--|--------------------------------------|
| Γ | } | return 0; | | | |
| | vo | oid* threadfn(void* msg) | | | |
| <pre>{ int exit =0; int i; </pre> | | | | | |
| 2 | | <pre>for (i = 0; i< 20; i++){ if(i%2 == 0)</pre> | | | |
| Y | | Toggle Breakpoint | | | |
| | | Disable Breakpoint | | | |
| | | Breakpoint Properties | | | |
| | 0 | Breakpoint Types | • | | |
| Γ | | Toggle Hardware Breakpoint | | | |
| | | Add Bookmark | | | |
| L | | Add Task | | | |
| đ | ✓ | Show Quick Diff | Ctrl+Shift+Q | | 📮 Console 🛛 🔳 🗮 💥 |
| | | Show Line Numbers | | | QEMU Simulator [PathFinder-XD Embedo |
| | | Preferences | | | |

Figure 67. Selecting Breakpoint Properties

| 2 Properties for | |
|---|--|
| type filter text Actions Common Filter | Filter Restrict to Selected Processes and Threads: |
| ? | OK Cancel |

• And in the filtering section, check the threads you wish to associate with the breakpoint.

Figure 68. Breakpoint Properties Dialog

• Click OK to set the breakpoint

4.3.2.3 Debugging more than one application at the same time

To debug more than one application or process at a time, you must launch a separate gdbserver for each process. Use separate port number for each GDB server and use the "δ" symbol at the end of the command. Each gdbserver process will start in the background as shown in screenshot.

| [root@ashqemu ~]# gdbserver :1234 ./threadtestapp | å |
|--|-----------|
| [1] 859 [root@ashqemu ~]# Process ./threadtestapp created; Listening on port 1234 | pid = 862 |
| [root@ashqemu ~]# gdbserver :2345 ./ashtestapp & | |
| [2] 864 Process ./ashtestapp created; pid = 867 [root@ashqemu ~]# Listening on port 2345 | |

Figure 69. Launching Multiple Processes

You can now connect to each process via the **Debug A Process in Run-mode** menu. The Debug View will show each process as follows:

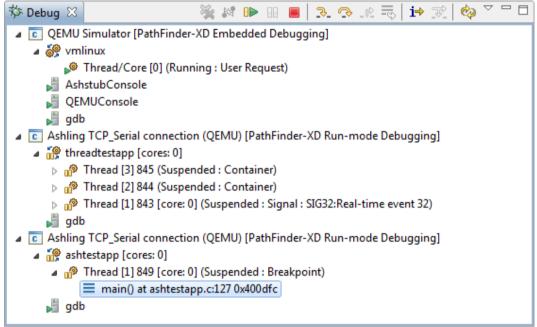


Figure 70. Debugging Multiple Processes

These examples show the power of PathFinder-XD's Embedded Linux support, in particular, the ability to debug Processes whilst the Kernel is running (Run-mode) and to debug the interaction between Processes and the Kernel (including Kernel modules). We hope you like it! Please send your feedback to hugh.okeeffe@nestgroup.net

Doc: APB211-PF-XD_MIPS_SIM, Hugh O'Keeffe and Suresh PC, Ashling Microsystems