



P&E GDB Server Plug-In for E200 Devices

Debug Configuration User Guide, v.1.03

1 Introduction

This user guide describes the basic process for setting up your debug connection within the P&E GDB Server Plug-In for NXP® E200 devices under your Eclipse-based IDE. A quick start guide is followed by a section that describes the various settings that can be used to configure your setup. The P&E GDB Server and Eclipse plug-in are fully tested and supported on the following operating systems:

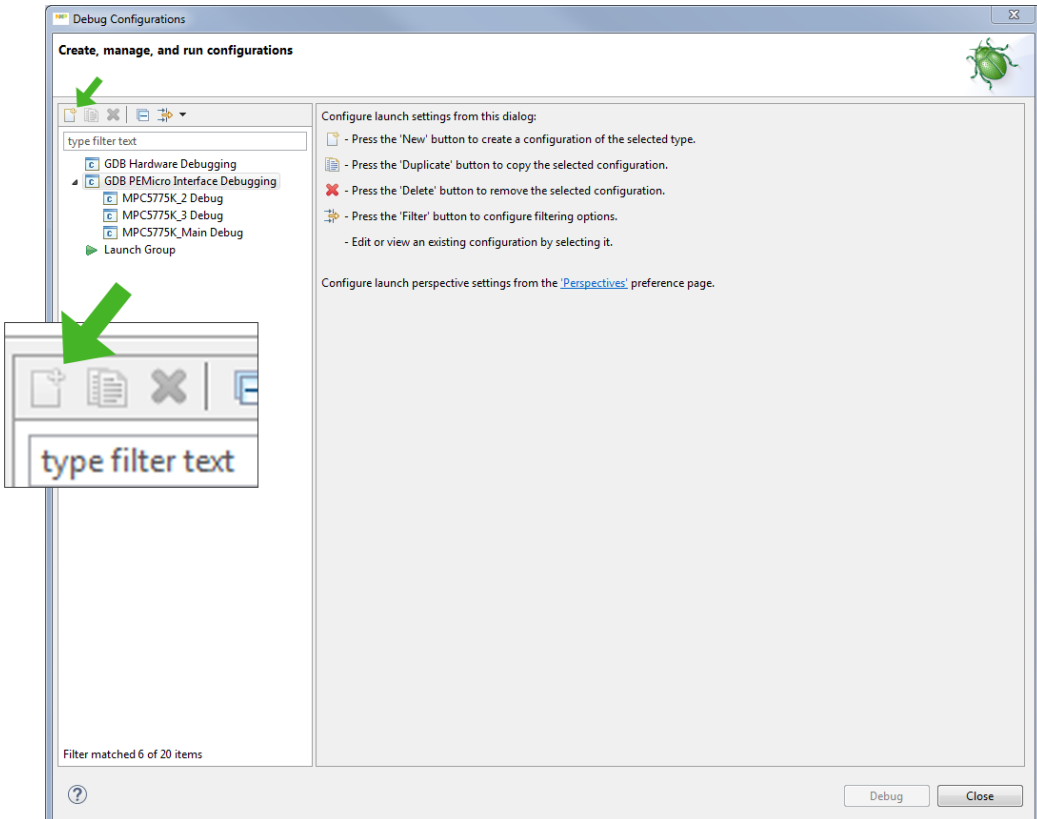
- Windows 7
- Windows 8
- Windows 10
- Debian 8.2
- Ubuntu 14.04
- CentOS 7

2 P&E GDB Server For E200 Devices- Quick Start Guide Using P&E Hardware Interfaces

Use the following steps to get started setting up your debug connection within the P&E GDB Server For E200 Devices.

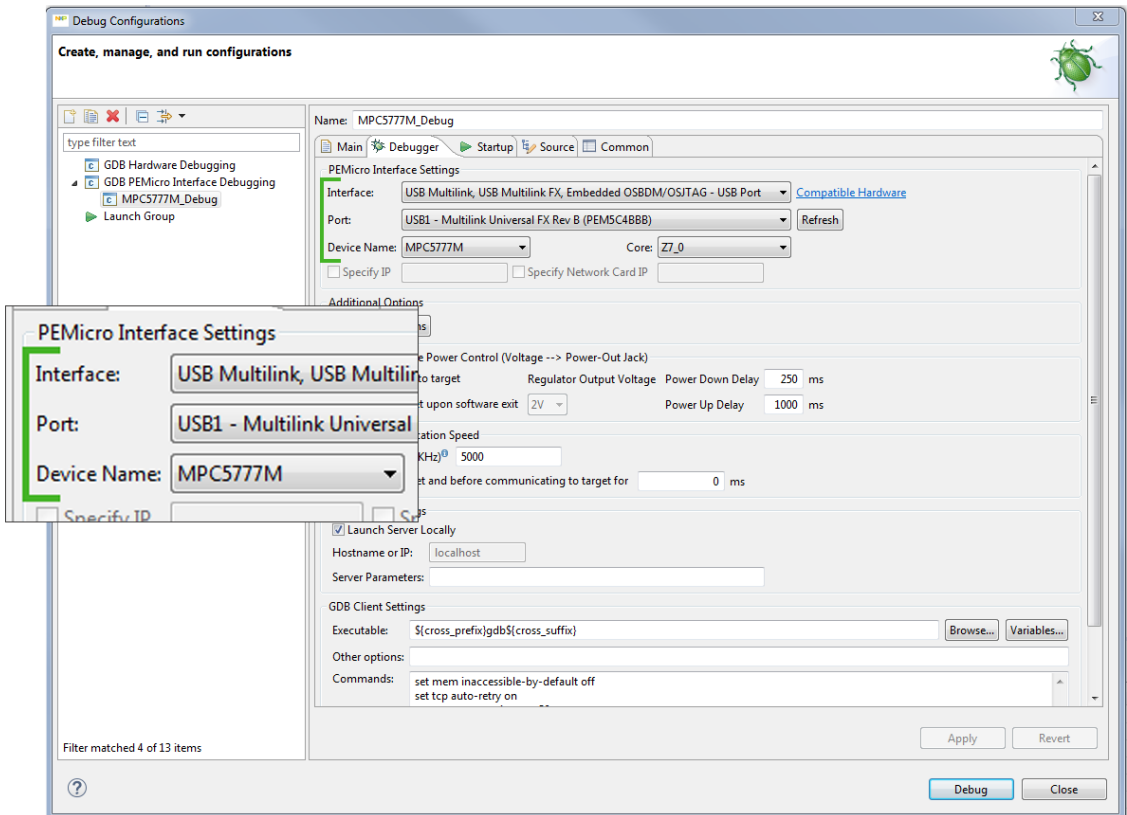
1. After creating and building your project, navigate to Run -> Debug Configurations from the menu bar.
2. Select “GDB PEMicro Interface Debugging” from the left panel. Click on the New Launch Configuration button to create a debug configuration.

Figure 2-1: New Launch Configuration Button



3. Click on the “Debugger” tab. A new panel will appear allowing the user to specify P&E hardware and settings. The user is required to specify the Interface, Port, Device Name, and Core. For most setups, the remaining settings may be left as default. Refer to **Section 3 - Changing P&E Connection Settings** for more details about each option.

Figure 2-2: Specify Interface, Port, Device Name, and Core



4. Click Apply and Debug.

3 Changing P&E Connection Settings

Connection settings for P&E hardware interfaces are configured in the Debug Configurations dialog box.

Figure 3-1: P&E E200 Launch Configuration Dialog Box

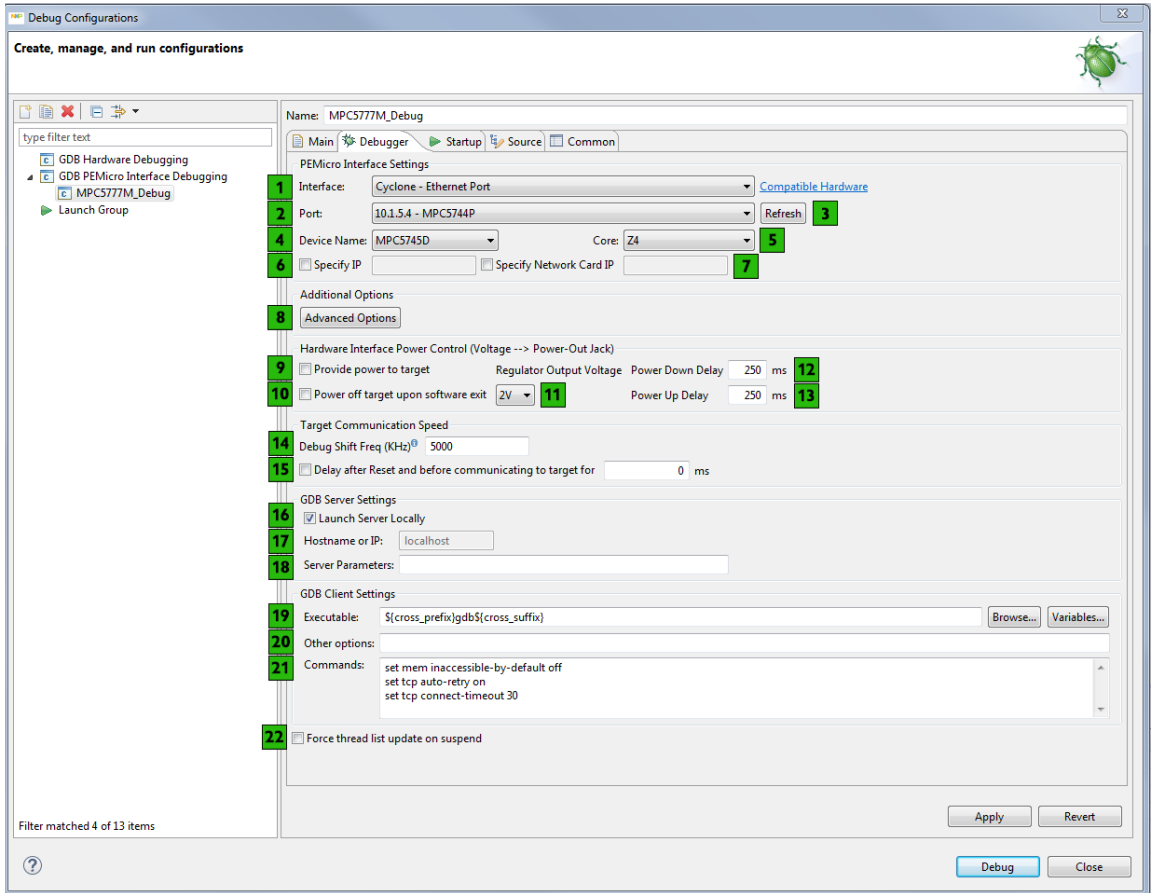


Table 1.1 describes the options for this view.

Table 1.1 Connection Parameter Options

Option	Description
[1] Interface	<p>Use this option to select the interface type. Select a supported interface from the list box. The options are:</p> <ul style="list-style-type: none"> • USB Multilink, USB Multilink FX, Embedded OSBDM/OSJTAG - USB Port • Cyclone - Serial Port • Cyclone - USB Port • Cyclone - Ethernet Port <p>NOTE: Click on the “Compatible Hardware” link to help you determine which P&E hardware is most suitable for your project.</p>
[2] Port	<p>This option selects the port over which debug communications is conducted. Select an available port from the list box.</p>
[3] Refresh	<p>Click this button to have the workstation scan for a valid interface and port. Valid interfaces and ports appear in the <i>Interface</i> and <i>Port</i> list boxes.</p>
[4] Device Name	<p>Selects the E200 processor being debugged..</p>
[5] Core	<p>Use this option to specify which core to debug. For multicore debugging, create a separate debug configuration for each core.</p>
[6] Specify IP (Cyclone Ethernet only)	<p>Use this option to specify the IP address of a hardware interface outside of the local network. Click on the checkbox to enable the textbox. This will also override the port dropdown box. Currently supports IPv4 only.</p>

Option	Description
[7] Specify Network Card IP (Cyclone Ethernet only)	Use this option to specify the local network card IP address if there are multiple cards on your computer.
[8] Advanced Options	Opens a menu option to configure the Flash algorithm or to preserve non-volatile memory. Please see Section 4 - Advanced Debug & Programming Options .
[9] Provide power to target (USB Multilink Universal FX and Cyclone Universal [FX] only)	<p>Check this option to have the Cyclone Universal [FX] or USB Multilink Universal FX (circuitry) supply power to the hardware target.</p> <p>Uncheck this option to not provide power.</p> <p>Note: For USB Multilink Universal FX, use the jumper settings located at JP10 to provide either 3.3V or 5V.</p>
[10] Power off target upon software exit (USB Multilink Universal FX and Cyclone Universal [FX] only)	<p>Check this option to turn off the power when the program terminates.</p> <p>Uncheck this option to leave the hardware target powered continuously.</p>
[11] Regulator Output Voltage (Cyclone Universal [FX] only)	<p>This option adjusts the output voltage that powers the hardware target.</p> <p>Select a voltage value from this option's list box.</p> <p>CAUTION An improper voltage setting can damage the board.</p>
[12] Power down delay (USB Multilink Universal FX and Cyclone Universal [FX] only)	<p>This option specifies amount of time for which the target will be turned off during a RESET power cycling sequence.</p> <p>Enter the delay interval (in milliseconds) in this option's text box.</p>

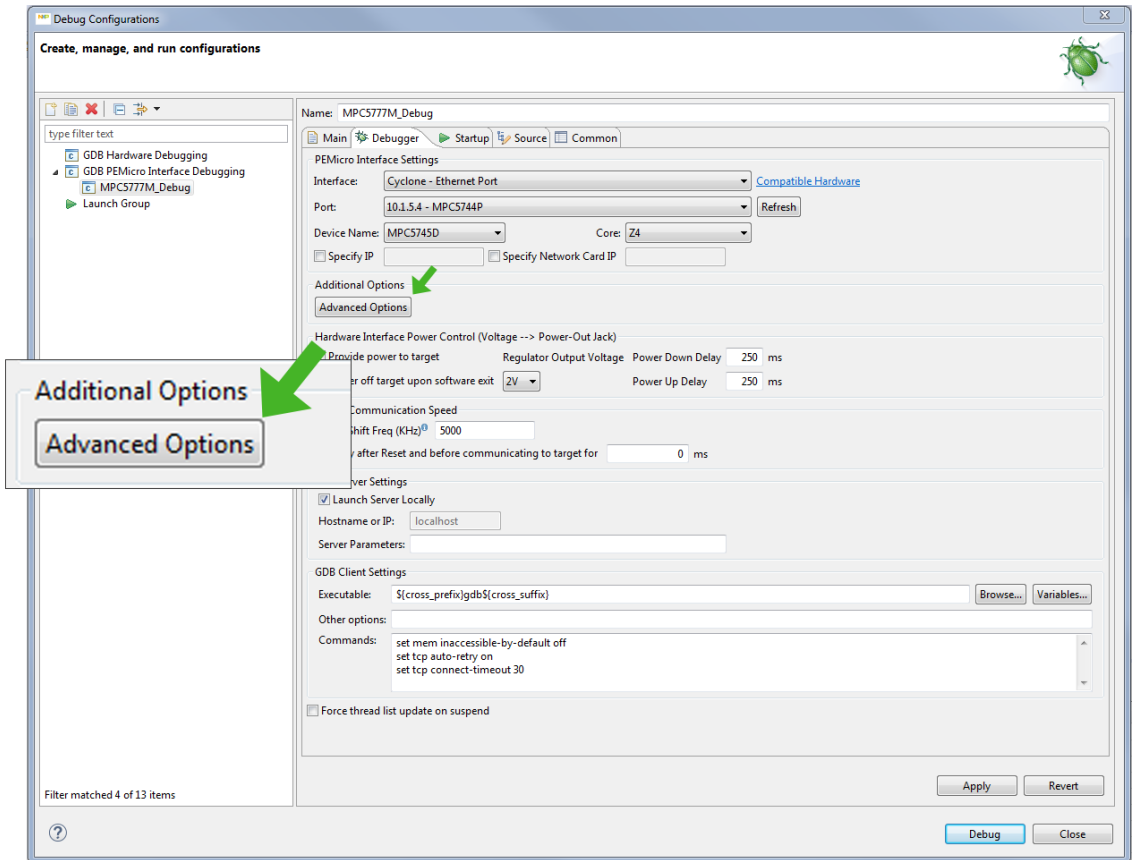
Option	Description
[13] Power up delay (USB Multilink Universal FX and Cyclone Universal [FX] only)	<p>This option specifies amount of time for which the target will remain powered prior to a RESET power cycling sequence.</p> <p>Enter the delay interval (in milliseconds) in this option's text box.</p>
[14] Debug Shift Freq.	<p>Specifies the debug shift frequency (in KHz) of P&E's hardware debug interfaces. Faster shift frequencies result in faster debug operations such as memory reads and flash programming. However, this frequency cannot exceed 1/6 of the target processor's bus frequency.</p>
[15] Delay After Reset	<p>Specifies a delay after the programmer resets the target that we check to see if the part has properly gone into background debug mode. This is useful if the target has a reset driver which hold the MCU in reset after the programmer releases the reset line. The n value is a delay in milliseconds.</p>
[16] Launch Server Locally	<p>Automatically starts and stops the P&E GDB Server locally for each debug session. Otherwise, the P&E GDB Server needs to be started manually. In this case, the GDB Server must be running before launching a debug session.</p>
[17] GDB IP Address	<p>Specifies the IP Address of the P&E GDB Server. This parameter is not used if the server is launched locally.</p>
[18] GDB Server Parameters	<p>Specifies additional command-line parameters when the P&E GDB Server is launched locally.</p>

Option	Description
[19] Executable	Specifies the path to the GDB client executable.
[20] Other Options	Specifies additional command-line parameters to the GDB client.
[21] Commands	Specifies additional commands to be executed by the GDB client at startup.
[22] Force Thread List Update On Command	Specifies whether CDT should ask the GDB client for updated thread information each time the target processor is suspended.

4 Advanced Debug & Programming Options

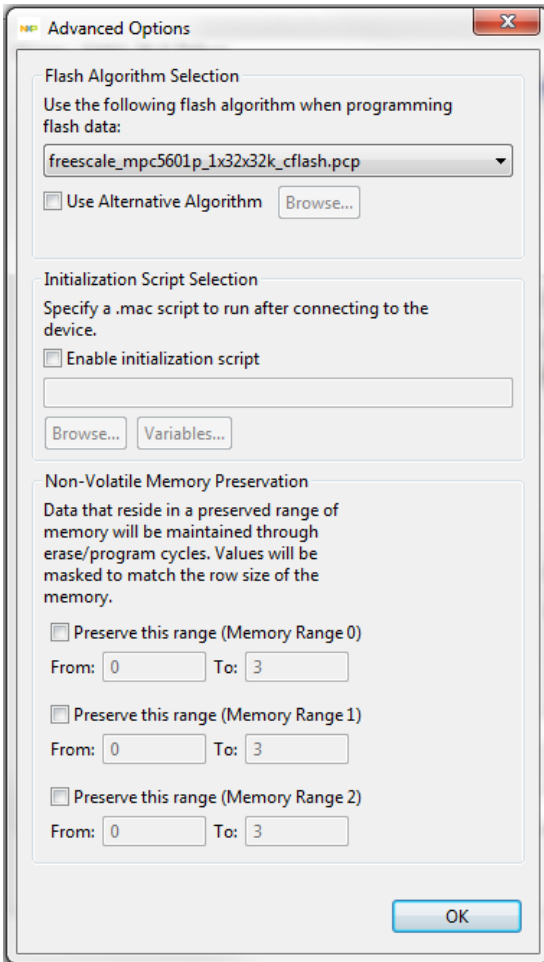
P&E's E200 Eclipse GDB Server Plug-In supports a collection of advanced debug and FLASH programming features. The Advanced Options dialog can be opened from the Launch Configuration Dialog of any P&E GDB Server Plug-In For E200 Devices-based debug configuration.

Figure 4-1: Advanced Options Button



Advanced options include: Preserving Memory Ranges and Alternative Algorithm Selection.

Figure 4-2: Advanced Options Dialog



4.1 Preserve Range

You have the option of preserving up to three independent ranges of non-volatile memory.

Ranges that are designated as "preserved" are read before an erase and re-programmed immediately afterwards, thereby preserving the data in these ranges.



Any attempt to program data into a preserved range is ignored. When entering an address into the preserved range field (hexadecimal input is expected) the values are masked according to the row size of the device. This ensures that the re-programming of preserved data does not cause any conditions that disturb programming.

4.2 Alternative Algorithm

Once you create a project for a specific E200 microprocessor, the debugger specifies a default algorithm to use during all Flash programming operations. The debugger uses this algorithm for nearly all programming requirements. The default algorithm can be found in <Eclipse_IDE_Installation_Folder>/eclipse/plugins/com.pemicro.debug.gdbjtag.pne_xxx/win32/gdi/P&E/. However, you can override the default algorithm via the Alternative Algorithm function. This feature can be used to select a custom programming algorithm, or to select another one of P&E's many programming algorithms for use with a specific project.

Alternative algorithms can be useful when one is trying to program internal or external FLASH memories with custom FLASH configurations.

Warning: Selecting the incorrect programming algorithm may damage your device, lead to under/over programming situations, or result in failure to program portions of the project file. Therefore it is recommended to use the default algorithm unless there is a compelling reason to do otherwise.

4.3 Initialization Script Selection

Initialization Script Selection functionality allows the user to specify a custom initialization script to run at the beginning of a debug session. This functionality can be helpful in initializing onboard resources such as external memory or a watchdog timer outside of an actual debug project. In order to enable the Initialization Script Selection feature, the user needs to check the Enable Initialization Script checkbox and point to an initialization macro file with the .mac extension. Commands in the following format should further be used to initialize desired registers with specific initialization values:

```
meminit.l $register_location $new_register_value
execute_opcode $command_value
spr spr_register_numberT $new_register_value
```



Example:

REM This script is compatible with MPC567xK devices in VLE.

REM Setup MMU for entire 4GB address space

REM Base address = \$0000_0000

REM TLB0, 4 GByte Memory Space, Not Guarded, Cacheable, All Access

spr 624t \$10000000 ; MAS0

spr 625t \$C0000B00 ; MAS1

spr 626t \$00000020 ; MAS2

spr 627t \$0000003F ; MAS3

execute_opcode \$7C0007A4 ; tlbwe

REM Initialize all of the Main SRAM - 512KB

meminit.l \$40000000 \$4007FFFF

5 Troubleshooting PE micro Debug Connection

5.1 Target Communication Speed

PE micro's debug configuration allows the user to modify the JTAG communication shift frequency between the debug interface and the target. By default, this frequency is set to a maximum value of 5000KHz, to take advantage of the fastest run control and FLASH programming experience:

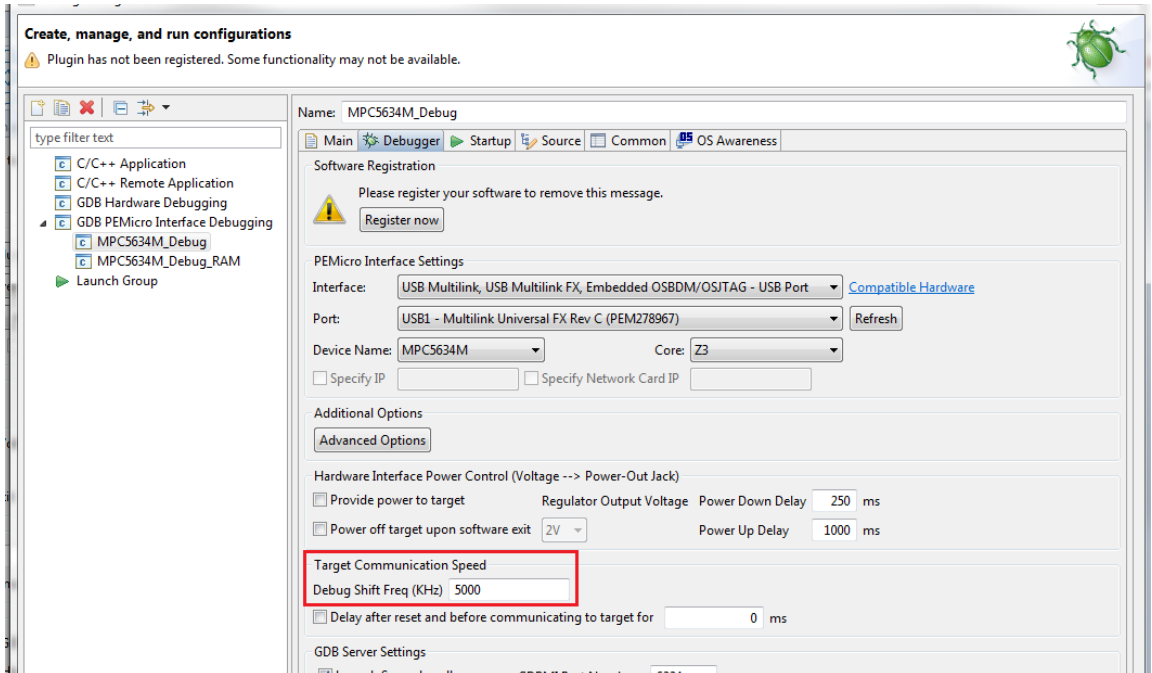


Figure 5-1: JTAG Communication Shift Frequency

At the same time, not every PowerPC processor might be able to support the highest debug shift frequencies with all PE micro debug interfaces. For example, the debug frequency for MPC5634M board used in conjunction with the Multilink Universal FX Rev C. needs to be lowered to 4000Khz in order to succeed. Hence, if the debug session fails to successfully start up or fails on FLASH programming, lowering the debug shift frequency by a factor of 2 or 4 is the first troubleshooting recommendation.

To get to PE micro’s debug configuration window, one should select “Debug Configuration” from the menu next to the debug icon, and switch over to the Debug tab, within the “Debug Configuration” menu pop- up:

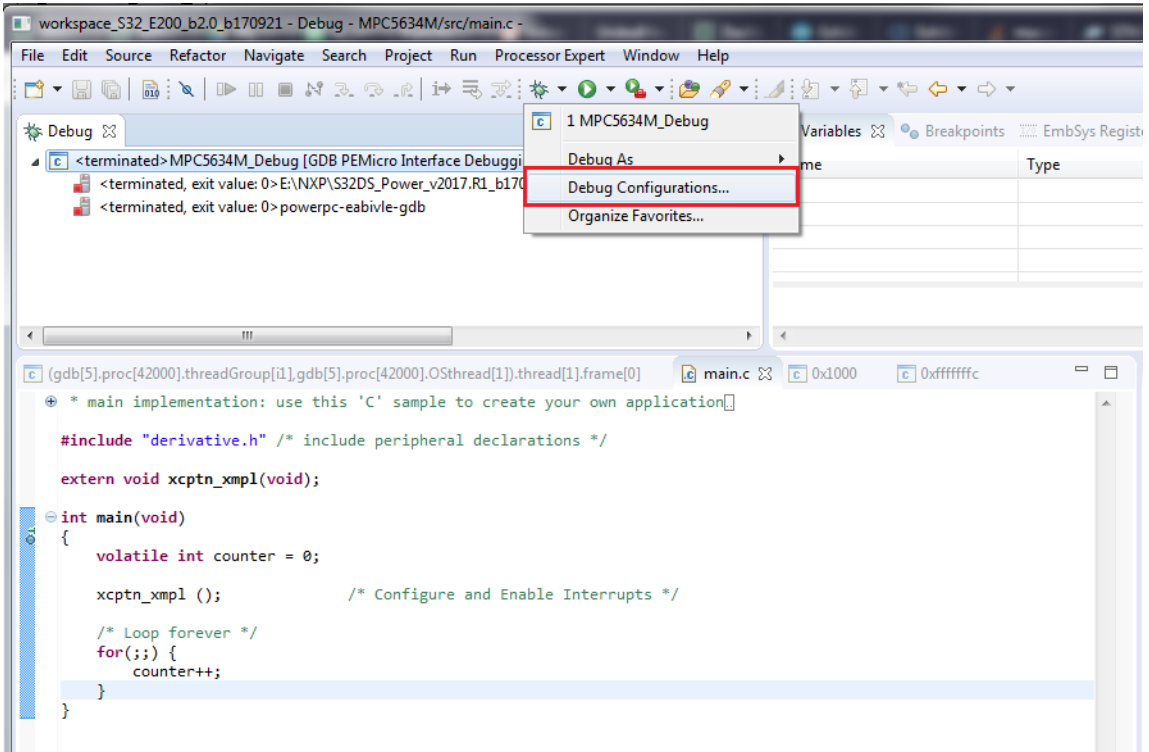


Figure 5-2: Navigate to PEmicro’s Debug Configuration Window

6 Attach Debug Session

P&E's Eclipse plugin supports the "attach" type of debug session. During an "attach" the GDB client is launched in a way that does not disturb the state of a target device, i.e. it skips the FLASH programming and reset steps which are a part of a standard P&E debug session.

Below is a detailed description of how detach/kill buttons within the Eclipse IDE are implemented:

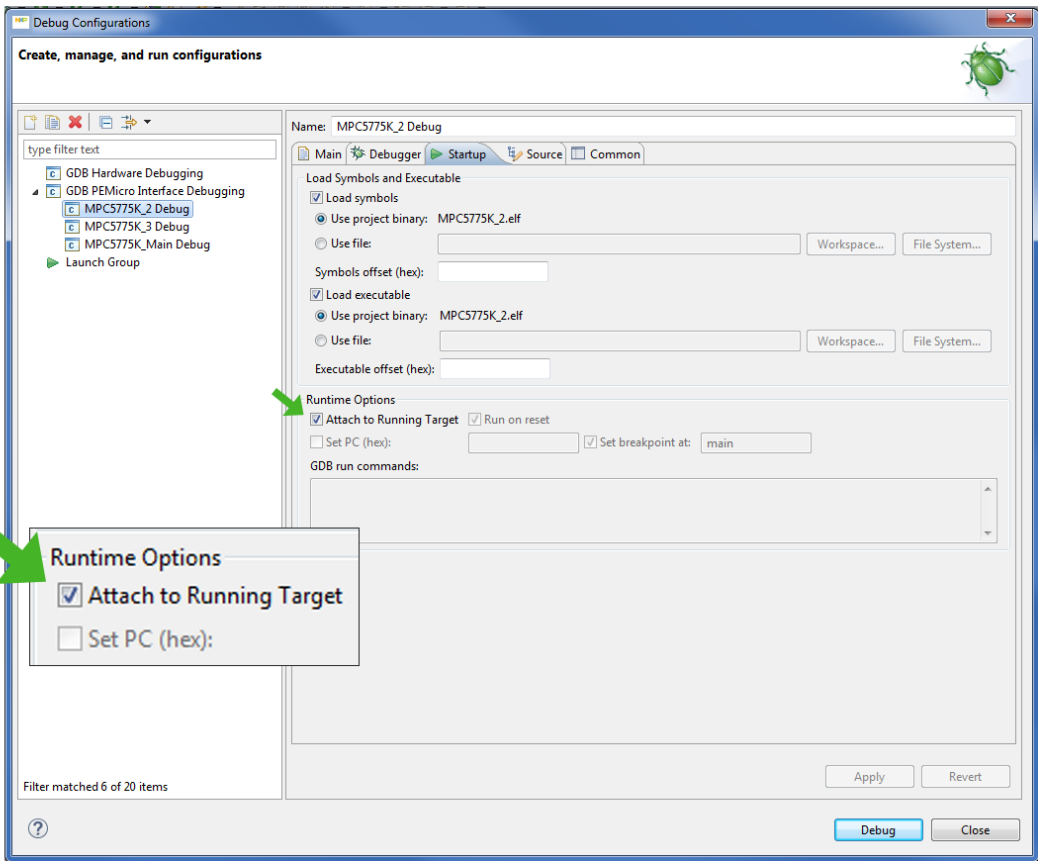
Detach/Disconnect - Debug session exits with the device in running state

Kill/Terminate - Debug session exits with the device in the debug (not running) state

Attach - Connection does not disturb the device. If it is running, P&E leaves it running and shows it to the user as such. If it is stopped, P&E shows it as stopped.

Attach Debug Session is enabled from within the Startup tab in P&E's debug configuration dialog (see **Figure 5-2**).

Figure 6-1: Enable Attach Debug Session



7 Multi-Core Project Debug Configuration Settings

The project wizard will generate multiple projects, one for each core, when a multi-

core project is created. To be able to debug multiple cores, the main core's debug session needs to flash program the ELF files for all cores. The P&E plug-in aggregates all binary information for programming, however the debug session will only display debug information for the specific core being debugged. Then an "attach debug session" is used to debug the other cores, otherwise starting multiple debug sessions on different device cores without attaching will reprogram the flash.

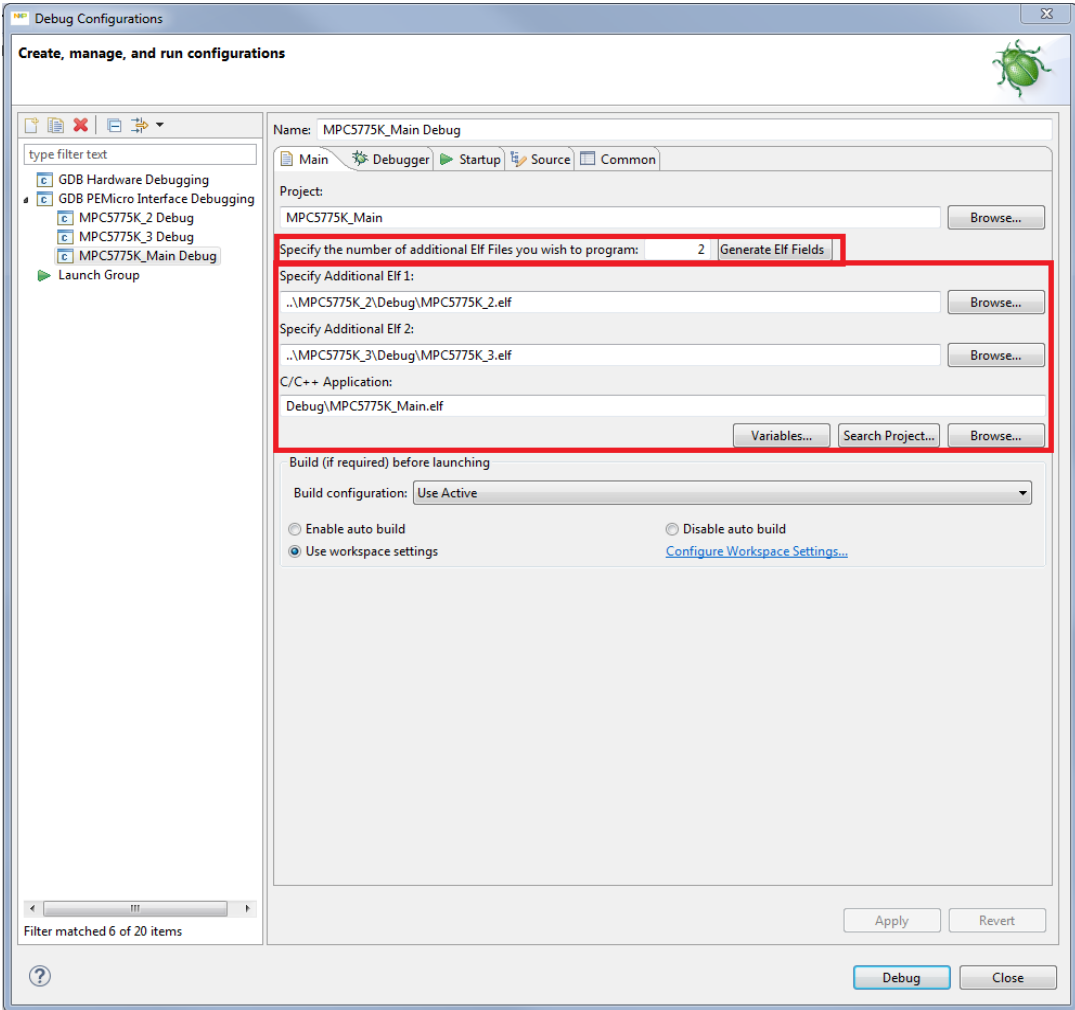
Note: Ensure the main core's project sets the reset vector and turns on the secondary cores. Otherwise the debug session for the other cores will not work. Within the reference manual of the chip, check for the MC_ME chapter.

To properly debug all of the cores, follow these steps in order:

7.1 Main Core

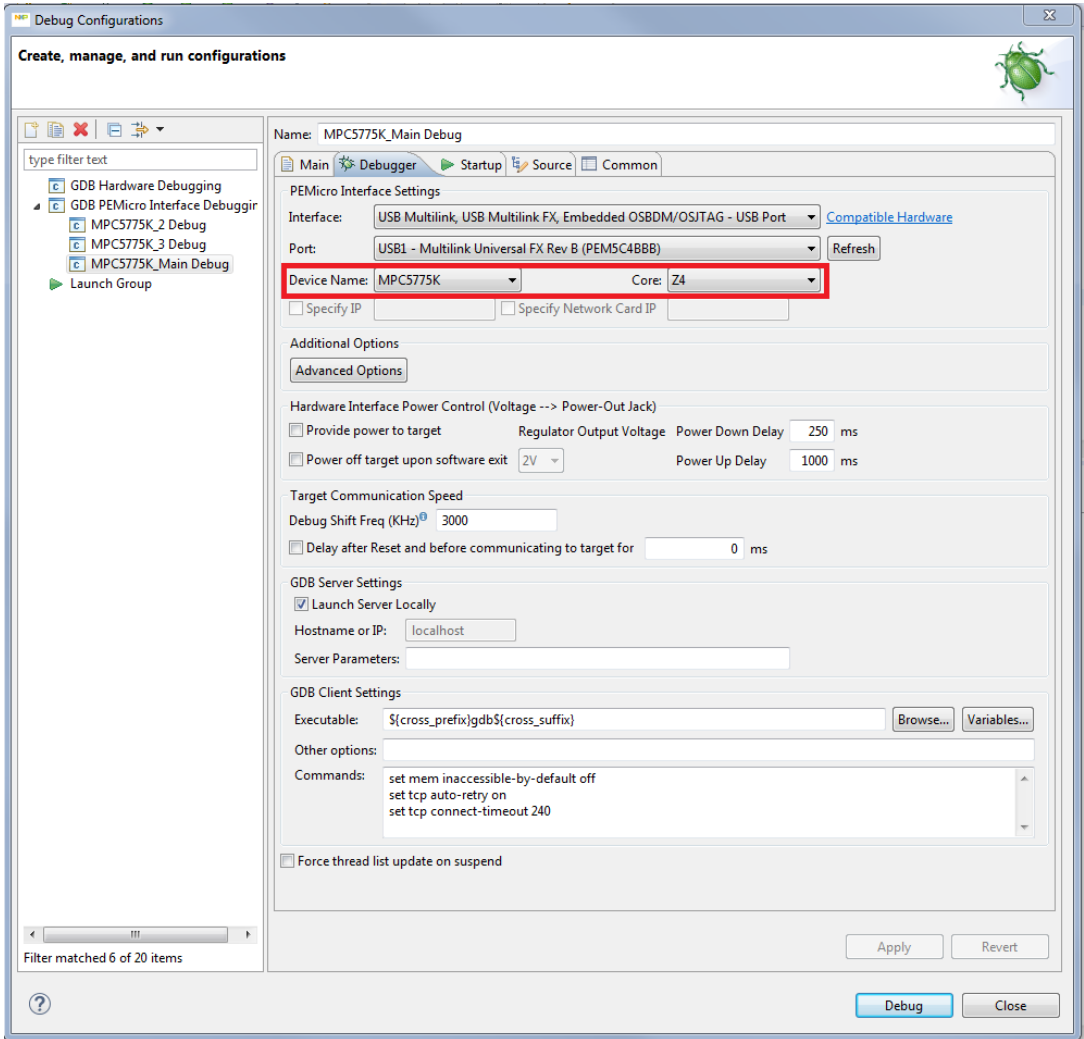
1. Set up the debug configuration for the main core, if not already done by the project wizard. The main core is responsible for programming the binary files for all device cores. In the **Main tab**, the settings for the main core must specify all of the ELF files to flash program. Use the **Generate ELF Fields** button to populate multiple lines and **Browse** to specify the ELF files for the other core. The main core's ELF file is already being loaded as the **C/C++ Application** setting which allows the debugger to load debug information, in addition to binary files for source-level run control debugging.

Figure 7-1: Main Tab of Main Core Debug Configurations



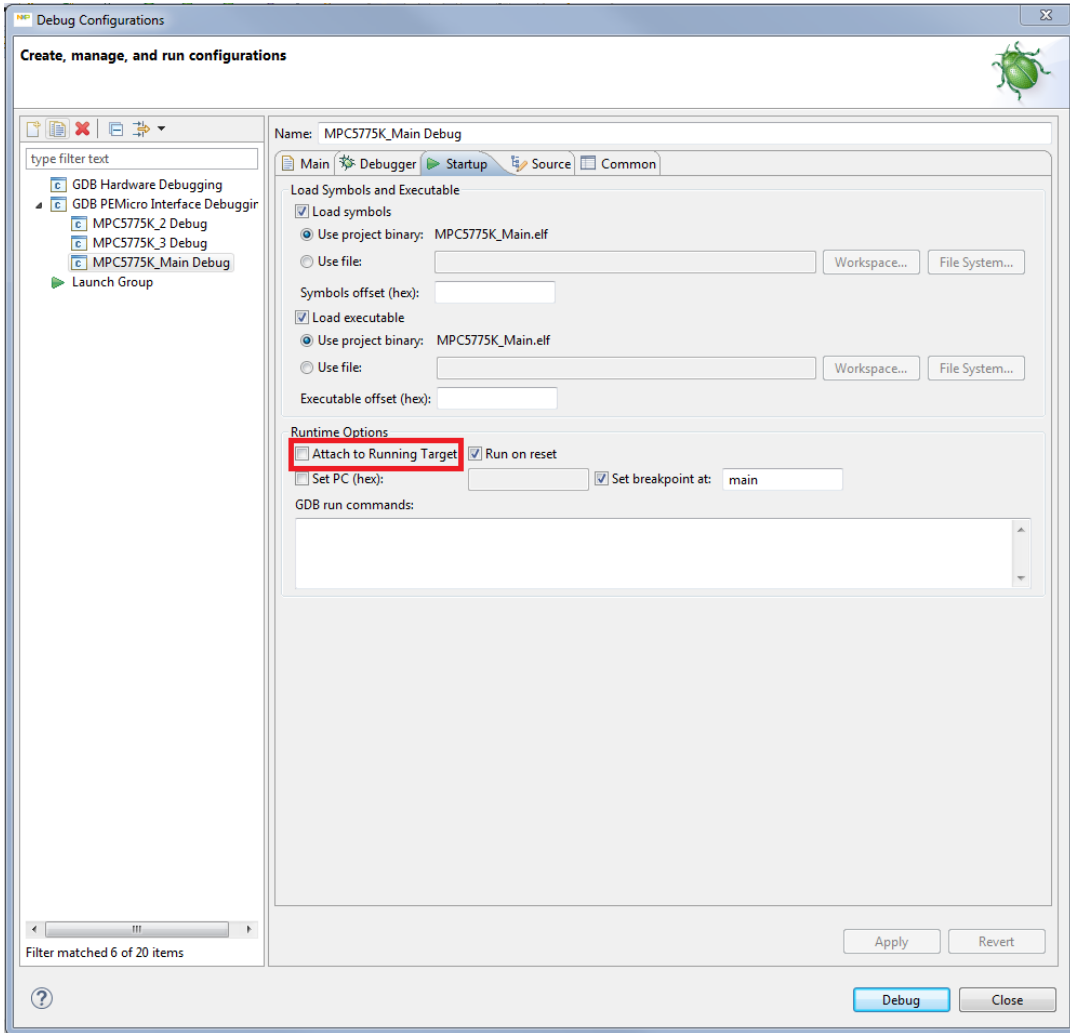
2. Setup the P&E hardware interface on the main core. Within the **Debugger** tab, select the P&E hardware interface and ensure that correct **Device Name** is selected and that the main core is selected under the **Core** option. Please read **CHAPTER 2 - P&E GDB Server For E200 Devices- Quick Start Guide Using P&E Hardware Interfaces** to learn about all of the settings.

Figure 7-2: Debugger Tab of Main Core Debug Configurations



3. Check the **Startup** tab. For the main core, the checkbox for **Attach to Running Target** should not be checked.

Figure 7-3: Startup Tab of Main Core Debug Configurations

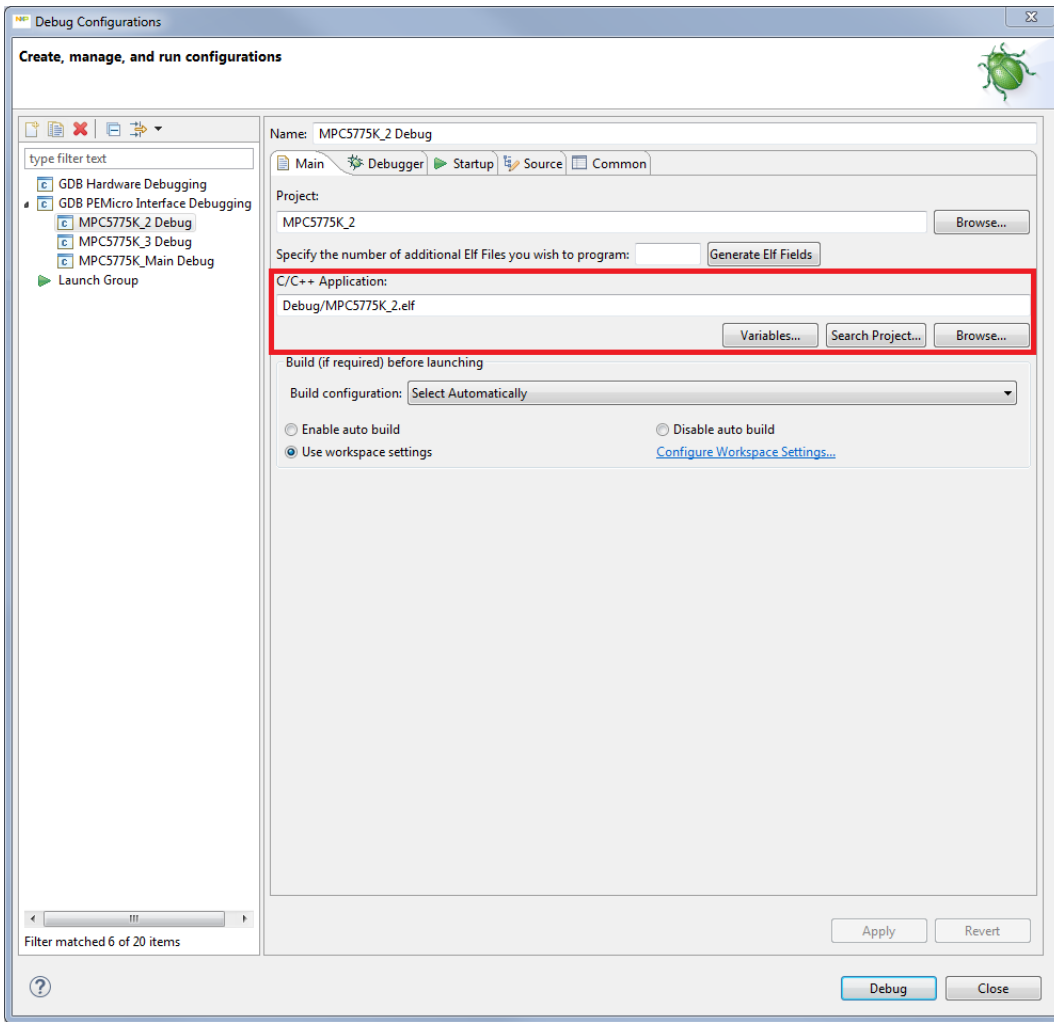


4. Click on the **Debug** button to start the main core's debug session. The P&E plug-in will erase and flash program all of the ELF files. The debug session will then load the object information from an ELF file for the main core. Run or step past the initialization of the other cores so that they are enabled.

7.2 Secondary/Other Cores

5. Set up the debug configuration for the other cores. Each core needs to load its own debugging information. Check the **Main tab** and check that the **C/C++ Application** setting is set to the correct ELF file.

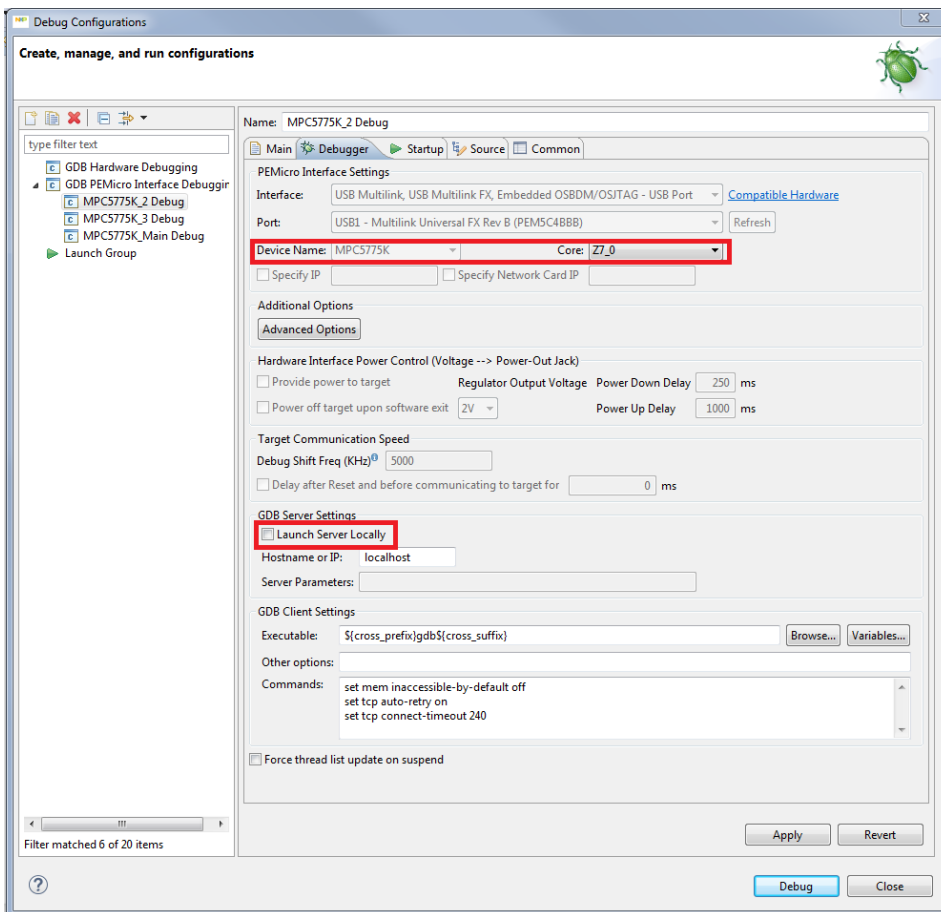
Figure 7-4: Main Tab of Secondary Core Debug Configurations



6. Setup the **Debugger Tab** for the other cores. If a flash programming and debug session has already been launched on the main core, the other cores can be attached to the same P&E hardware interface for a multi-core user experience. Ensure that the **Device Name** and **Core** settings are properly set up for the correct core for each debug session.

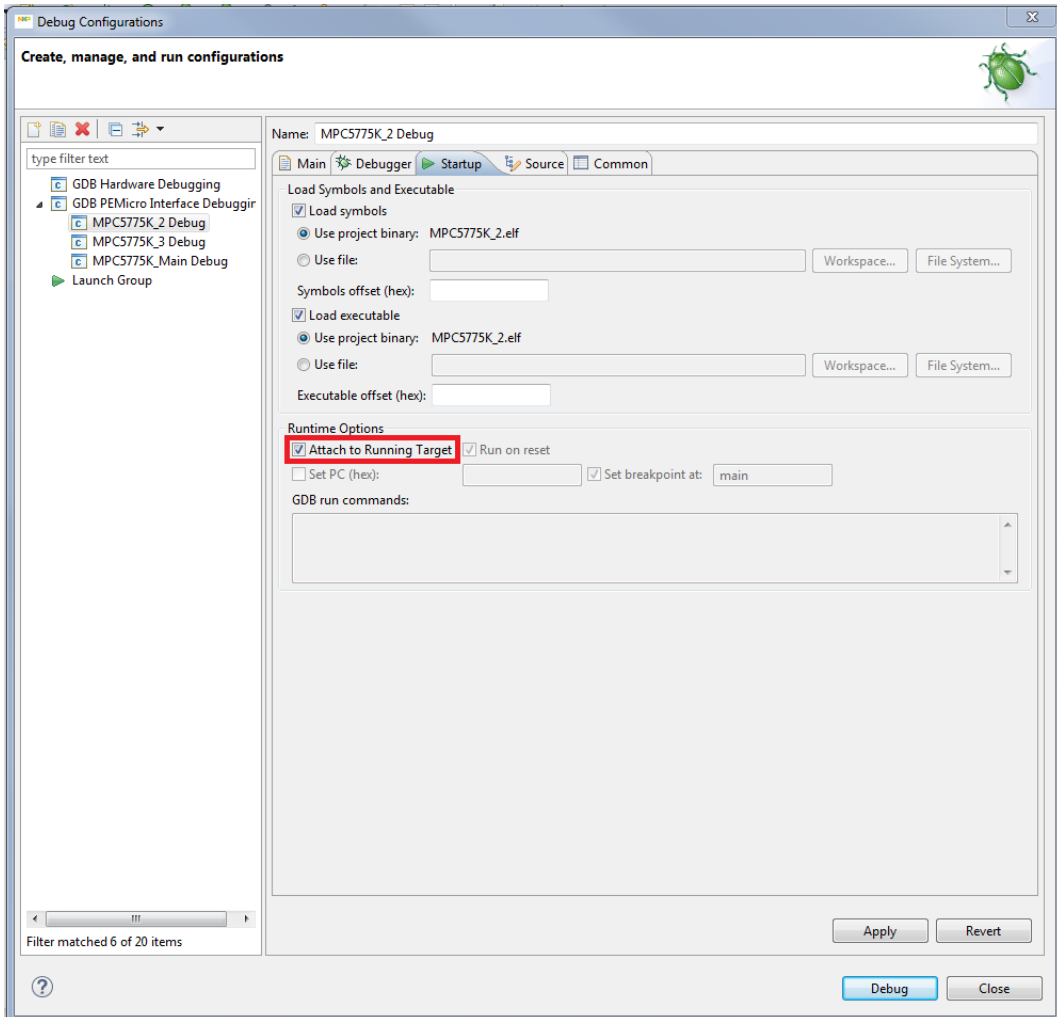
Note: Multiple core debug sessions can be launched as both local or remote sessions. Our recommendation is for the user to launch both debug sessions locally on the same IP address.

Figure 7-5: Debugger Tab of Secondary Core Debug Configurations



7. Set up the **Startup** tab for the other cores. The checkbox for **Attach to Running Target** needs to be checked, otherwise the debug session will erase and program the flash again, thereby disrupting an ongoing debug session running on the main core.

Figure 7-6: Startup Tab of Secondary Core Debug Configurations



8. Click on the **Debug** button to start the other core's debug session. The attach

session will only load the debug information. If the other cores are set up to already be running when enabled, then the attach session will start up as running.

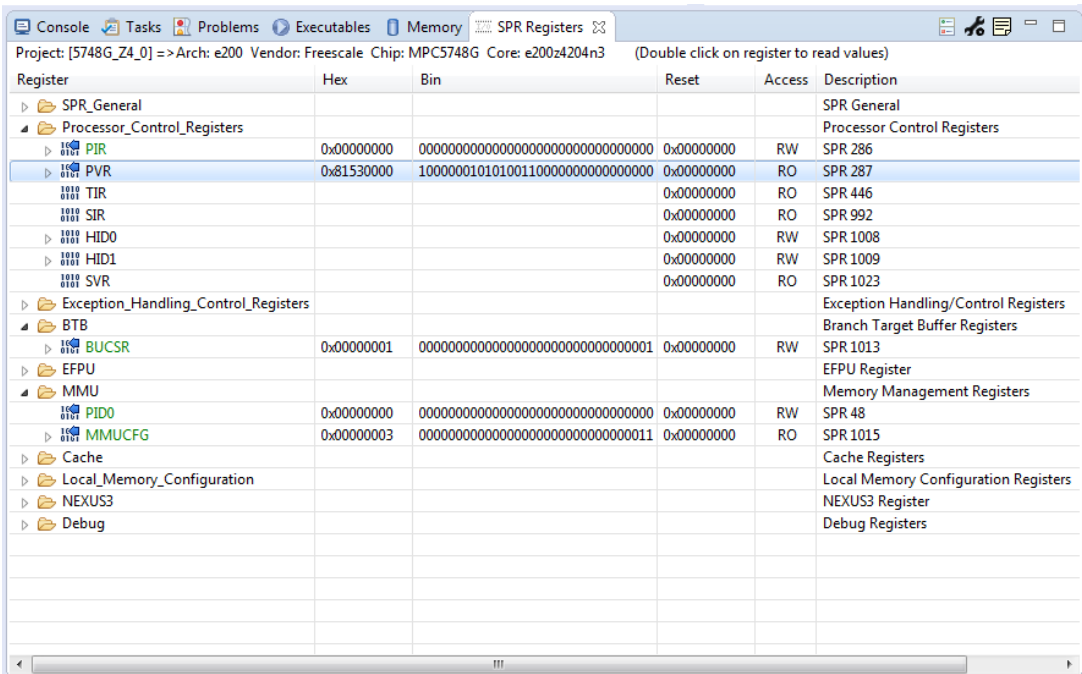
Note: Disconnecting, terminating, or resetting any of the debug sessions of the main core or secondary cores will affect all of the debug session running on the same P&E hardware interface. Running, stepping, setting breakpoints, and inspecting registers and variables will not affect the other cores and will only execute on the core in the currently selected project perspective.

8 Viewing and Changing the Special Purpose Registers (SPR)

S32 Power IDE v1.2 and higher added a special SPR register window to allow users to easily view and modify the values of SPR registers.

To open and view the SPR register dialog, please navigate to Window -> Show View -> Other -> Debug -> SPR Registers

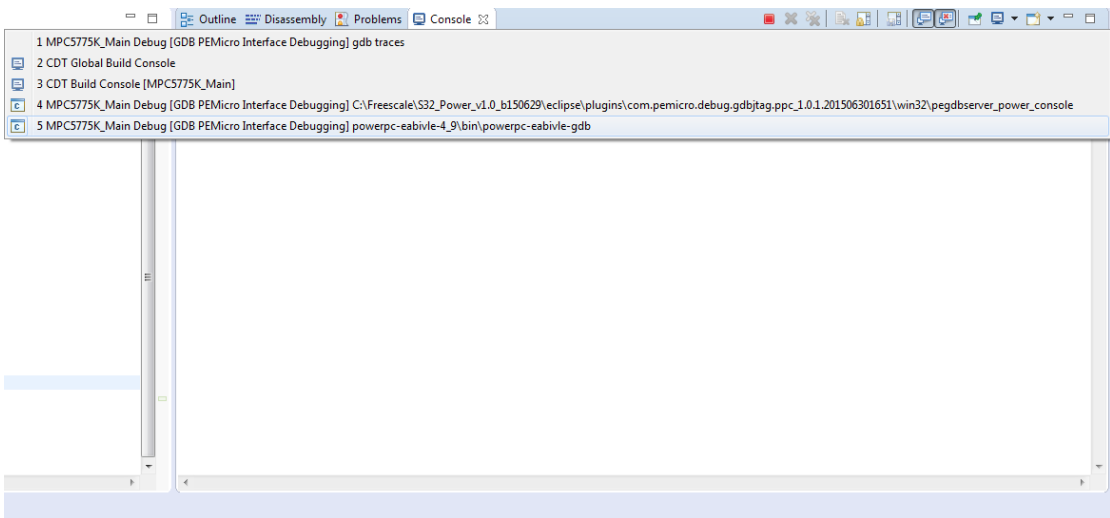
Figure 8-1: SPR Register Dialog



The user can also modify SPR registers in command line fashion with direct access via `powerpc_eabivle_gdb` console. During a debug session, the P&E GDB server console (`pegdbserver_power_console`) can display the Special Purpose Registers (SPR) by typing commands into the GNU GDB client console (`powerpc_eabivle_gdb`). To view the SPR value, follow these steps:

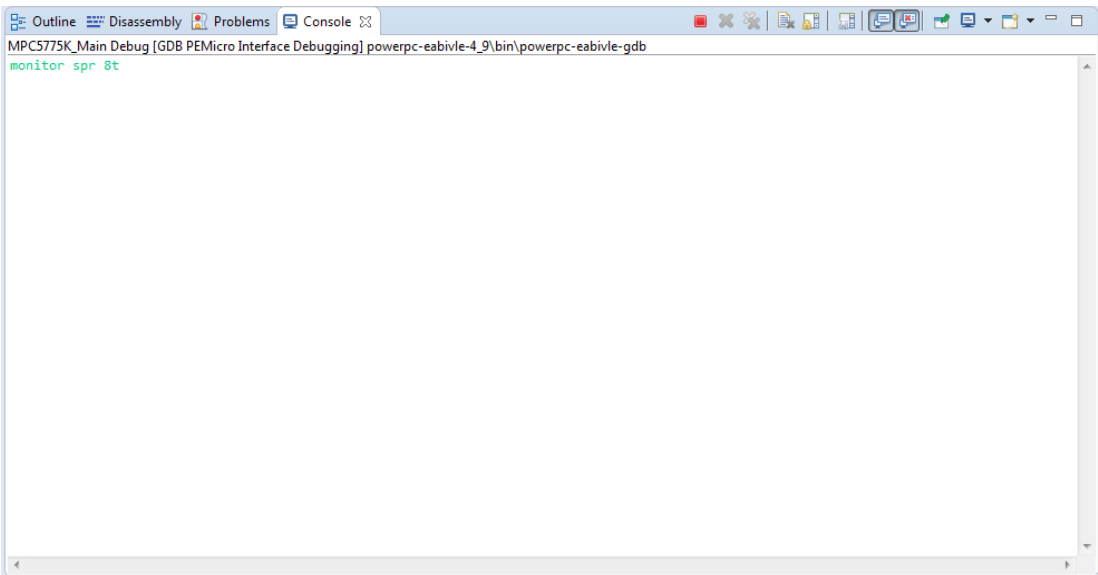
1. Open the console window and select the **powerpc_eabivle_gdb** console.

Figure 8-2: Select Console View



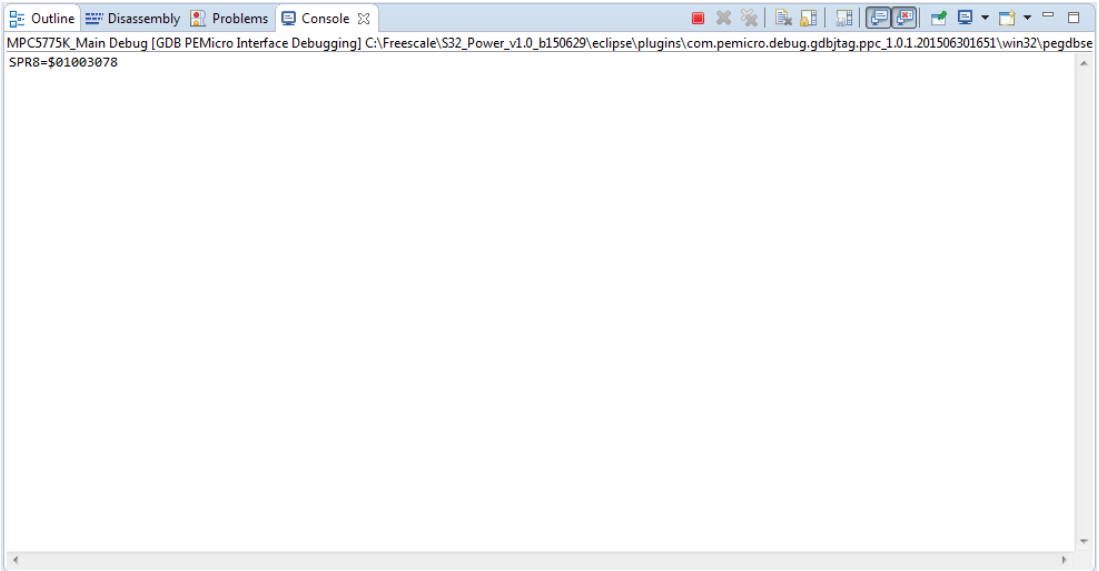
2. Type into the console the command “monitor spr x”, where ‘x’ is the register number. Decimal values should be specified with the letter ‘t’, and hex values with the letter ‘h’. For example, use “monitor spr 8t” to view the value of SPR 8.

Figure 8-3: Monitor Command Typed into GNU GDB Client Console



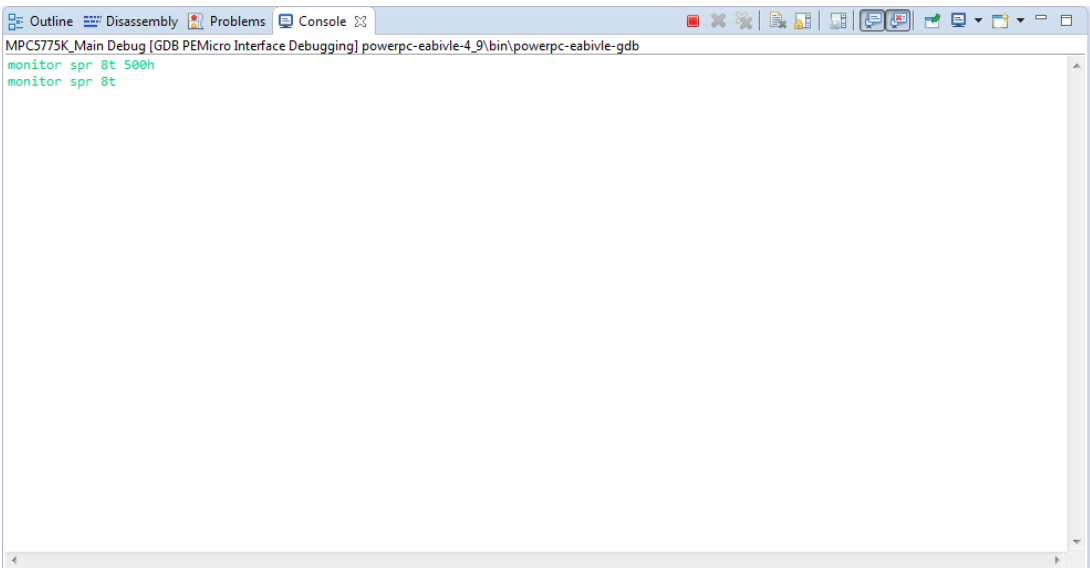
3. Hit the Enter key and the console will switch over to the **pegdbserver_power_console**. The console will display the value in hexadecimal.

Figure 8-4: SPR Value Displayed in P&E GDB Server Console



4. To modify the SPR value, switch back to the **powerpc_eabivle_gdb** console. The command to modify the value is “monitor spr x n” where ‘x’ is the register number and ‘n’ is the new value. Again, use the letter ‘t’ to specify decimal and letter ‘h’ for hexadecimal. For example, “monitor spr 8t 500h”.

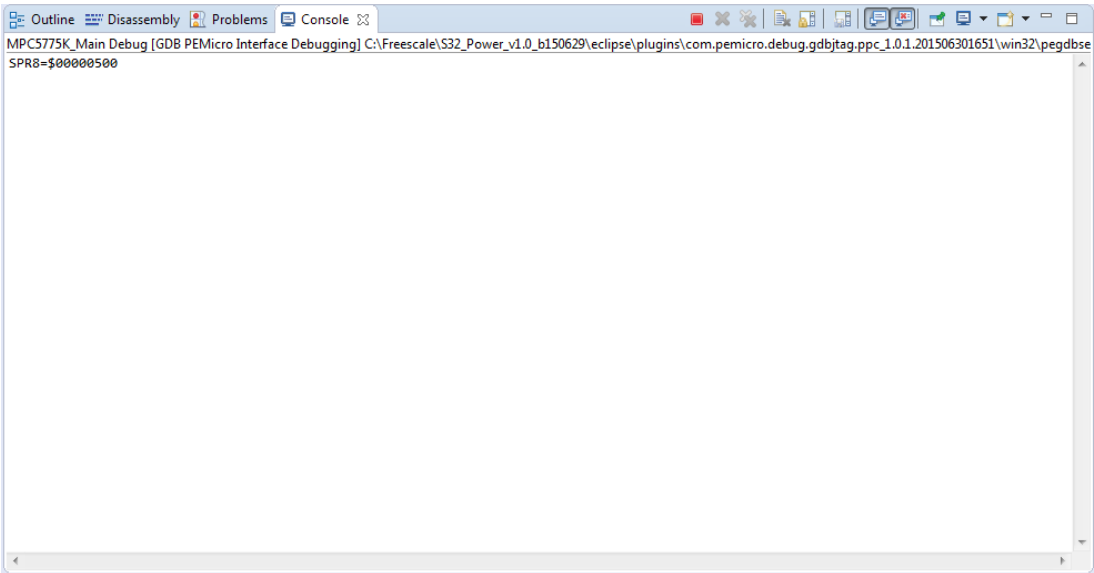
Figure 8-5: Input SPR value



```
MPC5775K_Main Debug [GDB PEMicro Interface Debugging] powerpc-eabivle-4_9\bin/powerpc-eabivle-gdb
monitor spr 8t 500h
monitor spr 8t
```

5. To confirm that the value has changed, call the monitor command again to view it in the **pegdbserver_power_console**.

Figure 8-6: Display new value of SPR

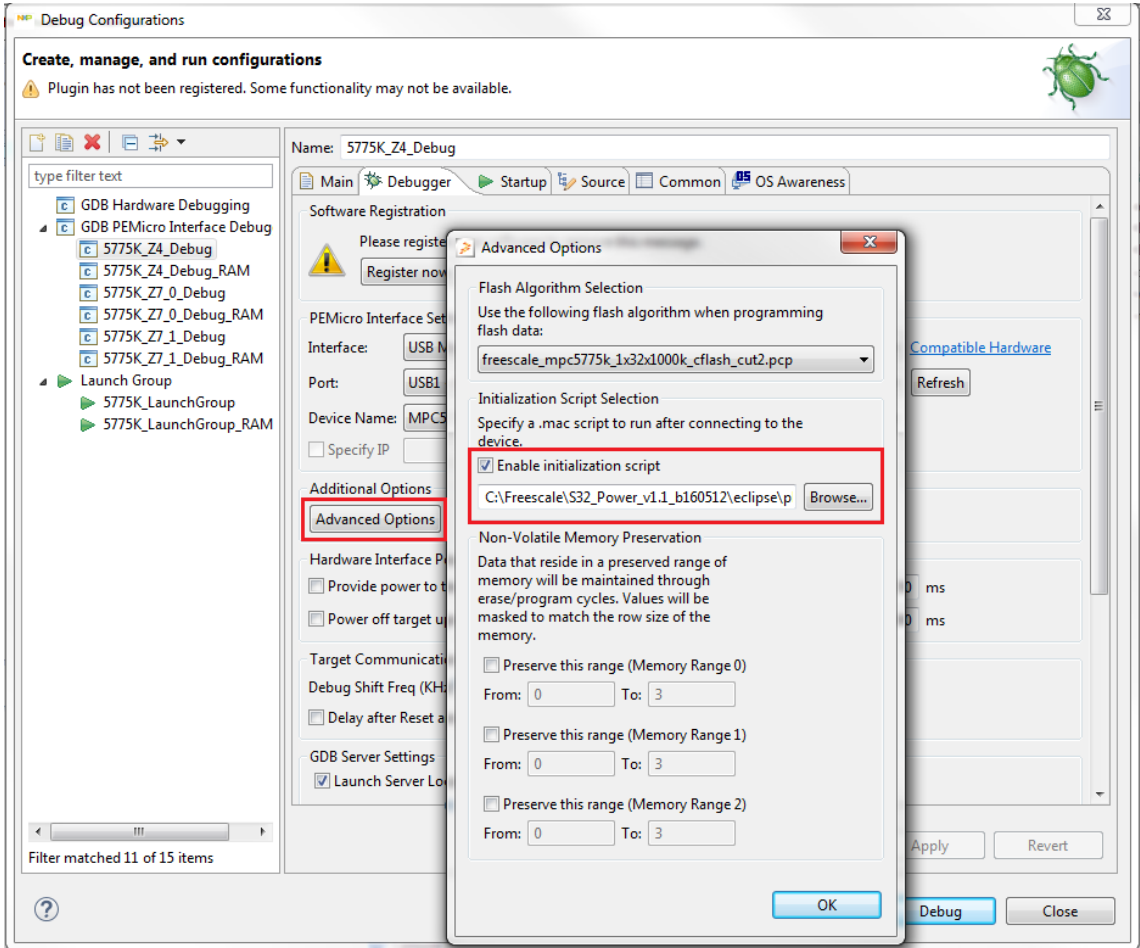


9 Configuration Macro Scripts

To address a need to configure devices at the beginning of a debug session, P&E added support for a user to add a custom Macro file within Advanced Options Dialog.

To manually add a configuration Macro(.mac) file to an S32 E200 project, please navigate to P&E Debug Configuration -> Debug Tab -> Advanced Options and browse to an initialization script after checking Enable initialization script check box.

Figure 9-1: Browse To Initialization Script In Advanced Options Dialog



To add a custom initialization file via a Project Wizard when a project for a specific device is being created, please add the following information to P&E's launch configuration file:

```
<stringAttribute key="com.pemicro.debug.gdbjtag.ppc.macScript"
value="C:\Path_to_Macro_File\Name_of_Init.mac"/>
```

```
<booleanAttribute key="com.pemicro.debug.gdbjtag.ppc.macScriptEnable"
value="true"/>
```

Below is a list of macro script commands that can be used to initialize specific device peripherals at the beginning of a debug session:

9.1 BLOCK FILL or BF Command:

The BF or FILL command fills a block of memory with a specified byte, word or long. The optional variant specifies whether to fill the block in bytes (.B, the default), in words (.W) or in longs (.L). Word and long must have even addresses.

Syntax:

```
BF[.B | .W | .L] <startrange> <endrange> <n>  
FILL[.B | .W | .L] <startrange> <endrange> <n>
```

Where:

<startrange> Beginning address of the memory block (range).
<endrange> Ending address of the memory block (range).
<n> Byte or word value to be stored in the specified block.

The variant can either be .B, .W, .L, where:

- .B Each byte of the block receives the value.
- .W Each word of the block receives the value.
- .L Each word of the block receives the value.

Examples:

BF C0 CF FF	Store hex value FF in bytes at addresses C0-CF.
FILL C0 CF FF	Store hex value FF in bytes at addresses C0-CF
BF.B C0 CF AA	Store hex value AA in bytes at addresses C0-CF.
FILL.B C0 CF AA	Store hex value AA in bytes at addresses C0-CF.
BF.W 400 41F 4143	Store word hex value 4143 at addresses 400-41F.
FILL.W 400 41F 4143	Store word hex value 4143 at addresses 400-41F.
BF.L 1000 2000 8F86D143	Store long hex value 8F86D143 at address 1000-2000
FILL.L 1000 2000 8F86D143	Store long hex value 8F86D143 at address 1000-

9.2 BR Command

Sets or clears a breakpoint at the indicated address. Break happens if an attempt is made to execute code from the given address. There are at most 7 breakpoints. They cannot be set at a misaligned address. Typing BR by itself will show all the breakpoints that are set and the current values for n.

Syntax:

BR [add] [n]

Where:

add Address at which a break point will be set.

n If [n] is specified, the break will not occur unless that location has been executed n times. After the break occurs, n will be reset to its initial value. The default for n is 1.

Examples:

BR 100 ; Set break point at hex address 100.

9.3 CR Command

The CR command sets the condition register (CR) to the specified hexadecimal value.

Syntax:

CR <n>

Where:

<n> The new hexadecimal value for the CR.

Example:

CR \$C4 Assign the value C4 to the CR.

9.4 CTR Counter Register Command

The CTR command sets the counter register (CTR) to the specified hexadecimal value.

Note:

The counter register is used by the CPU for looping purposes. This register is also a special purpose register.

Syntax:

CTR <n>

Where:

<n> The new hexadecimal value for the CTR.

Example:

CTR \$100 Assign the value \$100 to the CTR.

9.5 EXECUTE_OPCODE Command

Treats a numeric parameter as an opcode and executes it.

Syntax:

EXECUTE_OPCODE <n>

Where:

<n> Numeric opcode.

Examples:

EXECUTE_OPCODE \$7C0007A4 ; tlbwe

9.6 MM Memory Modify Command

The MM command directly modifies the contents of memory beginning at the specified address. The optional variant specifies whether to fill the block in bytes (.B, the default), in words (.W), or in longs (.L).

If the MM command includes optional data value(s), the software assigns the value(s) to the specified address(es) (sequentially), then the command ends. No window will appear in this case.

Syntax:

```
MM [.B|.W|.L] <address>[<n> ...]
```

Where:

<address> The address of the first memory location to be modified.

<n> The value(s) to be stored (optional).

Examples:

```
MM 400 00 Assign value 00 to hex address 400.
```

```
MM.L 200 123456 Place long hex value 123456 at hex address 200.
```

9.7 MSR Command

The MSR command sets the machine status register (MSR) to the specified hexadecimal value.

Syntax:

```
MSR <n>
```

Where:

<n> The new hexadecimal value for the MSR.

Example:

```
MSR $C4 Assign the value C4 to the MSR.
```

9.8 NOBR Command

Clears all break points. Please note that issued NOBR commands will clear all breakpoints on run control layer, which might not properly be reflected by breakpoint viewer within Eclipse CDT layer.

Syntax:

NOBR

Example:

NOBR Clears all break points.

9.9 PC Program Counter Command

The PC command assigns the specified value to the 32-bit program counter (PC). As the PC always points to the address of the next instruction to be executed, assigning a new PC value changes the flow of code execution.

If source code is showing in a code window, an alternative way for setting the Program Counter is to position the cursor on a line of code, then press the right mouse button and select the Set PC at Cursor menu item. This assigns the address of that line to the PC.

Syntax:

PC <address>

Where:

<address> The new PC value.

Example:

PC \$0500 Sets the PC value to 0500

9.10 R(x) Generate Purpose Register Command

The R(x) command sets the value of the 32-bit General Purpose Register R(x), where (x) is a value from 0 to 31. For targets that support the 64-bit SPE registers, use the H

and L suffixes.

Syntax:

R(x)[H | L] [n]

Where:

(x) Value from 0 to 31, corresponding to which register the user intends to write.

[H | L] Indicates register with most significant bits H or least significant bits L. Default is L.

[n] Value to be written to register.

Example:

R3 \$CF03D4AA Writes value of \$CF03D4AA to General Purpose Register R3.

R3H \$CF03D4AA Writes value of \$CF03D4AA to General Purpose Register R3H.

9.11 REM Command

The REM command allows a user to display comments in a macro file. When the macro file is executing, the comment appears in the status window. The text parameter does not need to be enclosed in quotes.

Syntax:

REM <text>

Where:

<text> A comment to be displayed when a macro file is executing.

Example:

REM Program executing Display message "Program executing" during macro file

execution.

9.12 SPR Command

The SPR Command displays the value of the Special Purpose Register (x). The user can then enter a new value for the register or a simple carriage return to keep the same value. The addresses used are the same as for the MTSPR or MFSPR instructions. This is used to setup the LR, CTR, IMMR, and other special purpose registers.

Syntax:

SPR (x) [n]

Where:

(x) Value from 0 to 1023 corresponding to which register the user intends to write.

Note: The default debugger base is hexadecimal, so to force the register number to be base 10, add the character T as a suffix.

[n] Optional Value to be written to register.

Example:

SPR 638T \$1234Write \$1234 to the IMMR special purpose register.

9.13 XER Command

The XER command sets the integer exception register (XER) to the specified hexadecimal value.

Syntax:

XER <n>

Where:

<n> The new hexadecimal value for the XER.



Example:

XER \$C4 Assign the value C4 to the XER.