Ashling Product Brief APB200

Advanced Debugging using the Ashling MPC5500 tools

1. Contents
1. Contents
2. Introduction
3. Break on Data Access
4. Tracing Accesses to a Variable
   4.1 Cycle accurate mode Data Tracing
5. Tracing Code Execution between two Events
6. Tracing Code Execution up to an Event
7. Tracing Code Execution after an Event
8. Tracing Code Execution up to Program Halt
9. Vitra Trace Diagnostics
10. For more information…

2. Introduction
This document provides some examples of advanced debugging features available when using the Ashling MPC5500 Tools. Examples are based on PathFinder for MPC5500 v1.1.1 using an Ashling Vitra for MPC5500 Emulator and MPC5566 based target board. It is assumed that you have installed/configured PathFinder appropriately for use with the target board. Advanced debugging features described include:

- How to break (halt) when a particular variable in your program is accessed
- How to trace (capture) all write accesses to a particular variable
- How to trace code execution between two events, after an event, up to an event or a program halt

3. Break on Data Access
This example shows how to break (halt execution) when a particular variable in your program is accessed. We will use the example program

C:\PFMPC\Examples\Controlr\MPC5534\BIN\CONTROLR_RAM.CSO

Load the program via PathFinder’s File|Load menu

To halt when ever the variable iLastRandValue is accessed then setup an e200 Data Watchpoint at this variable as follows:

1. Open the Breakpoint Configuration dialog via the Run menu
2. Set the Address field of e200 Data Watchpoints Watchpoint 0 to iLastRandValue (use the Browse button to symbolically pick iLastRandValue)
3. In the **Set Hardware Breakpoints** tab check **e200 Data Watchpoint 0 Load Debug Event Set** to halt when `iLastRandValue` is read and **e200 Data Watchpoint 0 Store Debug Event Set** when `iLastRandValue` is written to.

![e200 Watchpoints Dialog](image)

**Figure 1. e200 Watchpoints Dialog**

4. Click OK and run the program from reset (**Run|Go from Reset**). The program will halt when `iLastRandValue` is either read or written to depending on your selection. Notice how the cause of break is shown in PathFinder’s Status bar (we halted in the example below when `iLastRandValue` was read).

![Breakpoint Configuration](image)

**Figure 2. Set Hardware Breakpoints Dialog**
4. Tracing Accesses to a Variable

This example shows how to trace (capture) all write accesses to specified variable in your program. We will use the example program

*C:\FFMFC\Examples\Controlr\MPC5534\BIN\CONTROLR_RAM.CSO*

Load the program via PathFinder’s File|Load menu

To trace write accesses to the variable iLastRandValue then setup a Trigger using Trigger|Trigger Configuration as follows:

1. The Trace Options tab can be left at default settings i.e.:

   **Figure 4. Trace Options Dialog**
   Depending on your Trace requirements you may want to adjust the Vitra Trace Buffer Size.
2. The **e200 Code** tab can be left at default settings i.e.:

<table>
<thead>
<tr>
<th>Trigger Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Emitting Code Trace</td>
</tr>
<tr>
<td>- Never (Code Trace Disabled)</td>
</tr>
<tr>
<td>- On Program Execution</td>
</tr>
<tr>
<td>- On e200 Inst Watchpoint 0</td>
</tr>
<tr>
<td>Stop Emitting Code Trace</td>
</tr>
<tr>
<td>- On Program Halt</td>
</tr>
<tr>
<td>- On e200 Inst Watchpoint 0</td>
</tr>
</tbody>
</table>

**NOTE**

If you are using Code Trace with Watchpoint Trace, do not set Watchpoints on any instruction that could also cause a Code Trace message to occur (i.e. BL, BNE, BLR, etc). Otherwise a trace message collision will occur internally within the processor, causing loss of Code Trace information.

---

3. The **e200 Data** tab should be set as shown below.

- **Set Start Emitting Data Trace to On Program Execution** and **Stop Emitting Data Trace to On Program Halt**.
- **Set Data Trace Region 0|Trace Type to Write Trace** and the **Start Address and End Address to iLastRandValue** (use the **Browse** button to symbolically pick iLastRandValue)

<table>
<thead>
<tr>
<th>Trigger Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Emitting Data Trace</td>
</tr>
<tr>
<td>- Never (Data Trace Disabled)</td>
</tr>
<tr>
<td>- On Program Execution</td>
</tr>
<tr>
<td>- On e200 Inst Watchpoint 0</td>
</tr>
<tr>
<td>Stop Emitting Data Trace</td>
</tr>
<tr>
<td>- On Program Halt</td>
</tr>
<tr>
<td>- On e200 Inst Watchpoint 0</td>
</tr>
</tbody>
</table>

**NOTE**

If you are using Data Trace with Watchpoint Trace, do not set Watchpoints on any instruction that could also cause a Data Trace message to occur (i.e. LDW, STW, etc). Otherwise a trace message collision will occur internally within the processor, causing loss of Data Trace information.

---

**Figure 5. e200 Code Dialog**

**Figure 6. e200 Data Dialog**
4. Click **Activate**, enable Trace (**Trace|Enable Trace**) and run the program from reset (**Run|Go from Reset**). Halt the program after a few seconds and open the Data Trace window. All write accesses to iLastRandValue will be shown as below:

![Data Trace Window](image)

Figure 7. Data Trace Window

4.1 **Cycle accurate mode Data Tracing**

PathFinder v1.1.1 supports **Cycle accurate mode** data tracing. In this mode, every Nexus data-access trace packet emitted from the chip is given a unique time-stamp. When **Cycle accurate mode** is off, then multiple Nexus packets will be assigned the same time-stamp. **Cycle accurate mode** therefore gives you more accurate time-stamps at the expense of less trace capacity.

![Trigger Configuration](image)

Figure 8. Enabling Cycle accurate mode
The follow screen-shots illustrate tracing with **Cycle accurate mode** on and off. These screen-shots use the example program `C:\PFMPC\Examples\1ms Timer\MPC5566\bin\RAM.CSO` which uses a 1mS interrupt handler to increments a variable. When **Cycle accurate mode** is on we can see the variable writes are accurately measured at 1mS intervals (+/-0.0005 mS)

![Data Trace](image)

**Figure 9. Data-tracing with Cycle accurate mode on**

When running this program, ensure that Critical Interrupt Hardware Breakpoints are off as per the below screen-shot (this will prevent the programming halting at each interrupt).

![Breakpoint Configuration](image)

**Figure 10. Setting Critical Interrupts off**
5. Tracing Code Execution between two Events

In this example we demonstrate using Instruction Watchpoints to trace all e200 code execution between two specific events (the entry and exit of a particular function). We will use the example program:

C:\PFMPC\Examples\Controlr\MPC5534\BIN\CONTROLR_RAM.CSO

Load the program via PathFinder’s File|Load menu

To capture all code execution of the function WriteToDevice then setup a Trigger using Trigger|Trigger Configuration as follows:

1. The Trace Options tab can be left at default settings i.e.:

   - **Generate EVTO Pulse on**:
     - eTPU A Watchpoint 0
     - eTPU A Watchpoint 1
     - eTPU A Channel Register Write Watchpoint
     - eTPU A Host Service Request Watchpoint
     - eTPU A Link Service Request Watchpoint
     - eTPU A Match Recognition Watchpoint
     - eTPU A Transition Detect Request Watchpoint

   - **Start Emitting Code Trace on e200 Inst Watchpoint 0**
   - **Stop Emitting Code Trace on e200 Inst Watchpoint 1**

   **Figure 12. e200 Code Dialog**

2. The e200 Code tab should be set to **Start Emitting Code Trace on e200 Inst Watchpoint 0** and **Stop Emitting Code Trace on e200 Inst Watchpoint 1** as shown below:

   **Figure 11. Trace Options Dialog**
3. The e200 Data, eTPU Code, eTPU Data, eDMA Data and eTPU Watchpoints should be left at their default values and the e200 Watchpoints tab should be set as shown below.
   - Check Allow User Control of e200 Watchpoint Resources
   - Set Watchpoint 0 to the entry of WriteToDevice
   - Set Watchpoint 1 to the exit of WriteToDevice
   - This can be done symbolically using the Browse... dialog (invoke via the Browse... button and make sure that Display Line Number Symbols is checked)

   ![Figure 13. e200 Watchpoints Dialog (setting Watchpoint 1)]

4. Click Activate, enable Trace (Trace|Enable Trace...) and run the program from reset (Run|Go from Reset). Halt the program after a few seconds and open the Code Trace window. All calls to WriteToDevice will be shown as below:
Instruction Watchpoint 0 (i.e. our “Start Trigger”) is shown in green (e.g. Frame 0 and 5 in the above screen-shot) and Instruction Watchpoint 1 (i.e. our “Stop Trigger”) is shown in red (e.g. Frame 4 in the above screen-shot). The Time column shows the time stamp of each captured frame. PathFinder only knows the absolute time for discontinuous instructions (e.g. bl, b, blr) or instructions at which a Watchpoint occurs, hence, the time for other frames is shown relative (< or >) to these frames. To quickly measure the time difference between frames, double-click on the Time column in the ‘reference’ frame. For example, in the below screen-shot we have set frame 5 as the reference frame by double-clicking in the Time column of frame 5. All other frame times are now shown relative to frame 5.
6. Tracing Code Execution up to an Event

In this example we demonstrate using Instruction Watchpoints to trace all e200 code execution up to a specific event (the entry to a particular function). We will use the example program:

C:\PPMPC\Examples\Controlr\MPC5534\BIN\CONTROLR_RAM.CSO

Load the program via PathFinder’s File|Load menu

To capture all code execution up to the call to the function WriteToDevice then setup a Trigger using Trigger|Trigger Configuration as follows:

1. The Trace Options tab should be set as below i.e. uncheck Stop Trace When Buffer Full and set the Virta Trace Buffer Size to the maximum supported size

![Figure 16. Trace Options Dialog](image)

5. The e200 Code tab should be set to Start Emitting Code Trace on e200 Inst Watchpoint 0 and Stop Emitting Code Trace on e200 Inst Watchpoint 1 as shown below:

![Figure 17. e200 Code Dialog](image)
6. The e200 Data, eTPU Code, eTPU Data, eDMA Data and eTPU Watchpoints should be left at their default values and the e200 Watchpoints tab should be set as shown below.
   - Check Allow User Control of e200 Watchpoint Resources
   - Set Watchpoint 0 to the entry of main as follows (i.e. start tracing as soon as program begins executing)

   ![Figure 18. e200 Watchpoints Dialog (setting Watchpoint 0)](image)

   - Set Watchpoint 1 to the entry of WriteToDevice (i.e. the point we want tracing to stop)

   ![Figure 19. e200 Watchpoints Dialog (setting Watchpoint 1)](image)

2. Click Activate, enable Trace (Trace|Enable Trace...) and run the program from reset (Run|Go from Reset). Halt the program after a few seconds and open the Code Trace window. All code execution up to WriteToDevice will be shown as below:
In this example we demonstrate using Instruction Watchpoints to trace all e200 code execution after a specific event (the entry of a particular function) until the trace buffer is full. We will use the example program:

```
C:\PFMPC\Examples\Controlr\MPC5534\BIN\CONTROLR_RAM.CSO
```

Load the program via PathFinder's File|Load menu.

To capture all code execution after the call to the function `WriteToDevice` then setup a Trigger using Trigger|Trigger Configuration as follows:

3. The Trace Options tab can be left at default settings i.e.:

![Figure 21. Trace Options Dialog](image)

Note that the Stop Tracing When Buffer Full ensures we halt tracing once we have a full trace buffer (buffer size can be adjusted using Vitra Trace Buffer Size).
4. The e200 Code tab should be set to Start Emitting Code Trace on e200 Inst Watchpoint 0 and Stop Emitting Code Trace On Program Halt as shown below:

5. The e200 Data, eTPU Code, eTPU Data, eDMA Data and eTPU Watchpoints should be left at their default values and the e200 Watchpoints tab should be set as shown below.
   - Check Allow User Control of e200 Watchpoint Resources
   - Set Watchpoint 0 to the entry of WriteToDevice i.e. the point at which we want tracing to start
   - This can be done symbolically using the Browse... dialog (invoke via the Browse... button and make sure that Display Line Number Symbols is checked)

6. Click Activate, enable Trace (Trace|Enable Trace...) and run the program from reset (Run|Go from Reset). Halt the program after a few seconds and open the Code Trace window. All code execution from WriteToDevice will be shown as below:
8. Tracing Code Execution up to Program Halt

This example shows how to trace all code execution up to program halt (e.g., your program hits a breakpoint or you halt it via PathFinder)

1. The Trace Options tab should be set as below i.e. uncheck Stop Trace When Buffer Full and set the Vitra Trace Buffer Size to the maximum supported size

2. The e200 Code tab should be set to Start Emitting Code Trace On Program Execution and Stop Emitting Code Trace on Program Halt as shown below:
This configuration will capture trace continuously until your program halts (note that no Watchpoints are needed in this configuration).

9. Vitra Trace Diagnostics
This section shows how to verify that your Vitra is correctly capturing trace data. The test will ensure that your Vitra and Target Probe Assembly (cable between your Vitra and your target system) are functional. This test requires PathFinder v1.1.1 or later. The test is included with your PathFinder software and involves running a group-file (script-file). By default, the group-file is stored in:

PFMPC\Ashling Trace Probe Test\MPC5566\mpc5566_trace_probe_test.grp

This can be run via PathFinder’s Run|Run A Group-file menu as follows:

![PathFinder Run A Group-file menu](image)

Browse to the group-file and select Open. The test will run and PathFinder will display the results in the Command window as follows:
If the tests fail, then:

1. Ensure Vitra is properly connected to your MPC5566 based target system and that the target system is powered up.

2. Ensure you are using with an MPC5566 based target system (other devices will not work).

3. If the problems still persist then you may have a faulty Vitra or cable; contact Ashling support on ashling.support@nestgroup.net

10. For more information…
You’ll find full details on all PathFinder operations and commands in the appropriate Ashling User manuals. To keep your Ashling software up-to-date, check regularly for the latest software downloads at www.ashling.com/support/mpc5500 by following the link to Download PathFinder-MPC5500.

www.ashling.com/support/mpc5500

Ashling Microsystems Ltd
National Technology Park
Limerick
Ireland

Phone: +353 61 334466
Fax: +353 61 334477
Email: support@ashling.com

Doc: APB200-MPC5500Trace.doc 13th Feb 2013