

Ashling Product Brief APB200

Advanced Debugging using the Ashling MPC5500 tools

1. Contents

1.	Contents	1
2.	Introduction	1
3.	Break on Data Access	1
4.	Tracing Accesses to a Variable	3
	4.1 Cycle accurate mode Data Tracing	5
5.	Tracing Code Execution between two Events	7
6.	Tracing Code Execution up to an Event	10
7.	Tracing Code Execution after an Event	12
8.	Tracing Code Execution up to Program Halt	14
9.	Vitra Trace Diagnostics	15
10.	For more information	16

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)

Breakpoint Confi	guration				
View Breakpoints 3	Set Breakpoints	e200 Watchpo	ints Set Hardware B	reakpoir	nts eTPU Watchpoints
e200 Watchpoint	•	/atchpoint Resou	irces		· · · ·
e200 Instruction \ Watchpoint 0	Watchpoints				
Address	Or	perating Mode	Address Space		
0	Browse A	lways	 Always 	-	Combination
-Watchpoint 1-					Equal
Address	Or	perating Mode	Address Space		
0	Browse	lways	 Always 	•	
-Watchpoint 2-					
Address	Or	perating Mode	Address Space		
0	Browse	lways	 Always 	-	Combination
-Watchpoint 3-					Equal 💌
Address	Op	perating Mode	Address Space		le dan
0	Browse A	lways	 Always 	•	
e200 Data Watc	hpoints				
Watchpoint 0-					
Address		perating Mode	Address Space		
40004000	Browse	lways	 Always 	-	
🗌 🗌 Linked to li	nstruction Watch	npoint 0 Event			Combination
Watchpoint 1 —					Equal 💌
Address		perating Mode	Address Space		E dage
0	Browse	lways	Always	-	
Linked to J	nstruction Watch	noint 2 Event			

Figure 1. e200 Watchpoints Dialog

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.

Breakpoint Configuration	
View Breakpoints Set Breakpoints e200 Watchpoin	ts Set Hardware Breakpoints eTPU Watchpoints
Halt execution on	
🔲 e200 Inst Watchpoint 0 Debug Event	🔽 e200 Data Watchpoint 0 Load Debug Event
🔲 e200 Inst Watchpoint 1 Debug Event	🔲 e200 Data Watchpoint 0 Store Debug Event
🔲 e200 Inst Watchpoint 2 Debug Event	📄 e200 Data Watchpoint 1 Load Debug Event
🔲 e200 Inst Watchpoint 3 Debug Event	📃 e200 Data Watchpoint 1 Store Debug Event
✓ Interrupt Taken Debug Event	🔽 Critical Interrupt Taken Debug Event
Interrupt Return Debug Event	Critical Interrupt Return Debug Event
External Debug Event 1	External Debug Event 2

Figure 2. Set Hardware Breakpoints 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).

40000534	123	•		pl[1] = pZ[1];
40000540	124	٠		<pre>if (p2[i] == '\0')</pre>
	125			break;
40000548	126	٠)
40000550	127	٠		}
	128			
	129			int rand(void)
	130			{
40000554	131	•	⇒	iLastRandValue *= 13;
40000560	132	٠		return (unsigned char)iLastRandValue;
	133			}
	134			·
	135			
	100			
4				
🕨 💼 MPC5	554 Iol∰ a dal	ta memory	/ rea	ad breakpoint 🕒 < 1s 🕘 🏙 eTPU A 🕘 🖑

Figure 3. PathFinder Source Window

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

Load the program via PathFinder's File|Load menu

Trigger Configuration

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.:

33 3		
Trace Options e200 Code e200 Data eTPU C	Code 🛛 eTPU Data 🗍 eDMA Data 🗍 e200 Watchpoints 🗍	eTPU Watchpoints
Trace Emit Options	Generate EVTO Pulse on	
🔽 Emit e200 Ownership Trace	eTPU A Watchpoint 0	🔽 e200 Inst Watchpoint 0
🔽 Emit eTPU A Ownership Trace	🔽 eTPU A Watchpoint 1	🔽 e200 Inst Watchpoint 1
🔽 Emit eTPU B Ownership Trace	🔽 eTPU A Channel Register Write Watchpoint	🔽 e200 Inst Watchpoint 2
🔽 Emit e200 Watchpoint Trace	eTPU A Host Service Request Watchpoint	🔽 e200 Inst Watchpoint 3
🔽 Emit eDMA Watchpoint Trace	eTPU A Link Service Request Watchpoint	🔽 e200 Data Watchpoint 0
💌 Emit eTPU A Watchpoint Trace	eTPU A Match Recognition Watchpoint	🔽 e200 Data Watchpoint 1
🔽 Emit eTPU B Watchpoint Trace	🔽 eTPU A Transition Detect RequestWatchpoint	
Processor Stall Options		
Stall e200 prior to Overflow	eTPU B Watchpoint 0	🔽 eDMA Data Watchpoint 0
🔲 Stall eTPU A prior to Overflow	✓ eTPU B Watchpoint 1	🔽 eDMA Data Watchpoint 1
Stall eTPU B prior to Overflow	eTPU B Channel Register Write Watchpoint	
Misc Options	eTPU B Host Service Request Watchpoint	
Stop Tracing When Buffer Full	eTPU B Link Service Request Watchpoint	
MCKO Frequency Divide Ratio	eTPU B Match Recognition Watchpoint	
1/2 of the System Clock	Image: Figure Provide a manual strength and the stren	
Vitra Trace Buffer Size		
1K Trace Racks		

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.: Trigger Configuration

Start Emitting Code Trace	e eTPU Data eDMA Data e200 Watchpoints eTPU Watchpoints Stop Emitting Code Trace					
Never (Code Trace Disabled) On Program Execution On e200 Inst Watchpoint 0	 On Program Halt On e200 Inst Watchpoint 0 					
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 collison will occur internally within the processor, causing loss of Code Trace information.						

Figure 5. e200 Code Dialog

- 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)

```
Trigger Configuration
```

Trace Options e200 Code e200 Data eTPU Code	eTPU Data eDMA Data e200 Watchpoints eTPU Watchpoints							
Start Emitting Data Trace	Stop Emitting Data Trace							
Never (Data Trace Disabled)								
 On Program Execution 	On Program Halt							
🔿 On e200 Inst Watchpoint 0	C On e200 Inst Watchpoint 0							
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 collison will occur internally within the processor, causing loss of Data Trace information.								
Data Trace Region 0	Data Trace Region 1							
Trace Type Write Trace	Trace Type No Trace							
Access Type Data Accesses 👤	Access Type Data Accesses							
Range Type Inclusive Range 💌	Range Type Inclusive Range 💌							
Start Address 40004000 Browse	Start Address 0 Browse							
End Address 40004000 Browse	End Address FF Browse							

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:

PData Trace (Data Frames: 52035)								
Frame	Time	Processor	Address	Symbol	Size	Value	Access	
00000187	84.7728ms	e200	40004000		32	03987 F 15	Write	
00000188	84.7732ms	e200	40004000		32	2EBE7411	Write	
00000189	84.8432ms	e200	40004000		32	5FABE4DD	Write	
00000190	84.8464ms	e200	40004000		32	DBBA9F39	Write	
00000191	84.8476ms	e200	40004000		32	287A15E5	Write	
00000192	84.8480ms	e200	40004000		32	0E331CA1	Write	
00000193	84.9208ms	e200	40004000		32	B898742D	Write	
00000194	84.9220ms	e200	40004000		32	5FBDE649	Write	
00000195	84.9908ms	e200	40004000		32	DCA4B1B5	Write	
00000196	84.9912ms	e200	40004000		32	345D0631	Write	
00000197	84.9956ms	e200	40004000		32	A8B9507D	Write	
00000198	85.0644ms	e200	40004000		32	91691659	Write	
00000199	85.0676ms	e200	40004000		32	62562285	Write	
00000200	85.0680ms	e200	40004000		32	FE5FC0C1	Write	
00000201	85.0732ms	e200	40004000		32	EADCC9CD	Write	
00000202	85.0764ms	e200	40004000		32	E D363 F 69	Write	
00000203	85.0776ms	e200	40004000		32	OBC13855	Write	
00000204	05 0700-	~200	40004000		22	000000000	Write	

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.

			de eTPU Data e		watchpoints [en o watchpo
Start Emitting D			Stop Emitting D	ata Trace]
O Never (Da	ta Trace Disabled)					
On Progra	m Execution		On Program	m Halt		
O On e200) Inst Watchpoint (v	C 0n e200	Inst Watchpoint 0	Ψ.	
		N	OTE]
could also cau	use a Data Trace n son will occur interr	nessage to occu	ce, do not set Watch r (i.e. LDW, STW, et rocessor, causing los	c). Otherwise a tra s of Data Trace in	ce	
	Write Trace	-	Trace Type	No Trace	-	
Trace Type	Data Accesses		Access Type	Data Accesses		
Trace Type Access Type	100000000			Inclusive Range	-	
	Inclusive Range	-	Range Type	Interest of the starting of		
Access Type		▼ Browse	Start Address	40013008	Browse	
Access Type Range Type	Inclusive Range	▼ Browse Browse			Browse Browse	

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)

Frame	Time	Processor	Address	Symbol	Size	Value	Access
0000013	5.0484ms	e200	40013000	u32bit	32	00000005	Write
00000015	6.0488ms	e200	40013000	u32bit	32	0000006	Write
00000017	7.0484ms	e200	40013000	u32bit	32	00000007	Write
00000019	8.0488ms	e200	40013000	u32bit	32	0000008	Write
00000021	9.0484ms	e200	40013000	u32bit	32	0000009	Write
00000023	10.0488ms	e200	40013000	u32bit	32	000000A	Write
00000025	11.0484ms	e200	40013000	u32bit	32	000000B	Write
00000027	12.0488ms	e200	40013000	u32bit	32	000000C	Write
0000029	13.0484ms	e200	40013000	u32bit	32	000000D	Write
00000031	14.0488ms	e200	40013000	u32bit	32	000000E	Write
00000033	15.0484ms	e200	40013000	u32bit	32	000000F	Write
00000035	16.0488ms	e200	40013000	u32bit	32	0000010	Write
00000037	17.0484ms	e200	40013000	u32bit	32	00000011	Write
00000039	18.0488ms	e200	40013000	u32bit	32	00000012	Write
00000041	19.0484ms	e200	40013000	u32bit	32	0000013	Write
00000043	20.0488ms	e200	40013000	u32bit	32	00000014	Write
00000045	21.0484ms	e200	40013000	u32bit	32	00000015	Write
00000047	22.0488ms	e200	40013000	u32bit	32	00000016	Write
00000049	23.0484ms	e200	40013000	u32bit	32	00000017	Write
00000051	24.0488ms	e200	40013000	u32bit	32	0000018	Write
00000053	25.0484ms	e200	40013000	u32bit	32	00000019	Write
00000055	26.0488ms	e200	40013000	u32bit	32	000001A	Write
00000057	27.0484ms	e200	40013000	u32bit	32	000001B	Write
00000059	28.0488ms	e200	40013000	u32bit	32	0000001C	Write
00000061	29.0484ms	e200	40013000	u32bit	32	000001D	Write
00000063	30.0488ms	e200	40013000	u32bit	32	000001E	Write
00000065	31.0484ms	e200	40013000	u32bit	32	000001F	Write
00000067	32.0488ms	e200	40013000	u32bit	32	00000020	Write
0000069	33.0484ms	e200	40013000	u32bit	32	00000021	Write
00000071	34.0488ms	e200	40013000	u32bit	32	00000022	Write
00000073	35.0484ms	e200	40013000	u32bit	32	00000023	Write
00000075	36.0488ms	e200	40013000	u32bit	32	00000024	Write

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 screenshot (this will prevent the programming halting at each interrupt).

ew Breakpoints Set Breakpoints e200 Watchp	oints Set Hardware Breakpoints eTPU Watchpoints
Halt execution on	
🔲 e200 Inst Watchpoint 0 Debug Event	🔲 e200 Data Watchpoint 0 Load (read) Debug Event
🔲 e200 Inst Watchpoint 1 Debug Event	🔲 e200 Data Watchpoint 0 Store (write) Debug Event
🔲 e200 Inst Watchpoint 2 Debug Event	🔲 e200 Data Watchpoint 1 Load (read) Debug Event
🔲 e200 Inst Watchpoint 3 Debug Event	🔲 e200 Data Watchpoint 1 Store (write) Debug Event
Interrupt Taken Debug Event	Critical Interrupt Taken Debug Event
Interrupt Return Debug Event	🖵 Critical Interrupt Return Debug Event
External Debug Event 1	External Debug Event 2

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:

- **Trigger Configuration** Trace Options e200 Code e200 Data eTPU Code eTPU Data eDMA Data e200 Watchpoints eTPU Watchpoints Trace Emit Options Generate EVTO Pulse on-▼ e200 Inst Watchpoint 0 Emit e200 Ownership Trace eTPU A Watchpoint 0 🔽 Emit eTPU A Ownership Trace eTPU A Watchpoint 1 e200 Inst Watchpoint 1 🔽 Emit eTPU B Ownership Trace ▼ eTPU A Channel Register Write Watchpoint ▼ e200 Inst Watchpoint 2 M Emit e200 Watchpoint Trace eTPU A Host Service Request Watchpoint ▼ e200 Inst Watchpoint 3 M Emit eDMA Watchpoint Trace eTPU A Link Service Request Watchpoint e200 Data Watchpoint 0 eTPU A Match Recognition Watchpoint M Emit eTPU A Watchpoint Trace e200 Data Watchpoint 1 🔽 Emit eTPU B Watchpoint Trace eTPU A Transition Detect RequestWatchpoint Processor Stall Options ✓ eDMA Data Watchpoint 0 🔽 eTPU B Watchpoint 0 Stall e200 prior to Overflow ▼ eTPU B Watchpoint 1 eDMA Data Watchpoint 1 Stall eTPU A prior to Overflow ▼ eTPU B Channel Register Write Watchpoint Stall eTPU B prior to Overflow eTPU B Host Service Request Watchpoint Misc Options eTPU B Link Service Request Watchpoint Stop Tracing When Buffer Full 🔽 eTPU B Match Recognition Watchpoint MCKO Frequency Divide Ratio eTPU B Transition Detect RequestWatchpoint 1/2 of the System Clock $|\mathbf{T}|$ Vitra Trace Buffer Size 1K Trace Racks -
- 1. The **Trace Options** tab can be left at default settings i.e.:

Figure 11. Trace Options 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:

rrigger configuration							
Trace Options e200 Code e200 Data eTPU Cod	le [eTPU Data eDMA Data e200 Watchpoints eTPU Watchpoints						
Start Emitting Code Trace	Stop Emitting Code Trace						
C Never (Code Trace Disabled)							
On Program Execution	C On Program Halt						
© 0n e200 Inst Watchpoint 0 ▼	On e200 Inst Watchpoint 1						
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 collison will occur internally within the pro	ocessor, causing loss of Code Trace information.						

Figure 12. e200 Code 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

Trigger Configuration

• This can be done symbolically using the **Browse**... dialog (invoke via the **Browse**... button and make sure that **Display Line Number Symbols** is checked)

	· · · · · · · · · · · · · · · · · · ·
Trace Options e200 Cod	le e200 Data eTPU Code eTPU Data eDMA Data e200 Watchpoints eTPU Watchpoints
e200 Watchpoint Option	
Allow User Control	pf.a200.Watchpoint Resources
e200 Instruction Watch	Browse 🗵
Watchpoint 0 Address	Code Symbols (*)
400004B8 Bro	
-Watchpoint 1	I ⊕start ⊡- CONTROLR Cancel
Address	FindDeviceADDB
400004E4 Bro	ReadFromDevice Help
Watchpoint 2	WriteToDevice
Address	95
0 Bro	
-Watchpoint 3	100
Address	ter main
0 Bro	Address: A0000AEA
e200 Data Watchpoint	+ strcpy
Watchpoint 0	Symbol Search
Address 0 Bro	Function name with wildcards Expand All
	Search
Linked to Instruct	Collapse All
Watchpoint 1 Address	🗖 Expand Search Result 🔲 eTPU Modules
0 Bro	✓ Display Line Number Symbols
Linked to Instruct	tion Watchpoint 2 Event

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:

🖗 Code T	race (Code	e Frames: 5	589)				
Frame	Time	Processor	Address	Symbol .	Instruction		Source Line
00000000	0.0us	e200	400004B8	\CONTROLR\WriteToDevice	stwu	R1, -0x30(R1)	(
00000001	< 1.6us	e200	400004BC		mflr	RO	
00000001	< 1.6us	e200	400004C0		stw	RO, 0x34(R1)	
00000001	< 1.6us	e200	400004C4		stw	R31, 0x2C(R1)	
00000001	< 1.6us	e200	400004C8		mr	R31, R4	
00000001	< 1.6us	e200	400004CC	\CONTROLR_95	addi	R4, R1, 0x10	FindDeviceADDR (ucCommand, &uiDevice
00000001	1.6us	e200	400004D0		bl	FindDeviceADDR	
00000002	< 1.6us	e200	400004FC	\CONTROLR\FindDeviceADDR	clrlwi	RO, R3, 0x18	if (ucCommand < LAST_COMMAND)
00000002	< 1.6us	e200	40000500		cmplwi	RO, 0x5	
00000002	< 1.6us	e200	40000504		bge-	_109	
00000002	< 1.6us	e200	40000508	\CONTROLR_107	rlwinm	RO, RO, 0x2, 0x0,	*puiDeviceAdr = puiDeviceAddrLookup
00000002	< 1.6us	e200	4000050C		lis	R3, 0x4000	
00000002	< 1.6us	e200	40000510		addi	R3, R3, 0x4008	
00000002	< 1.6us	e200	40000514		lwzx	RO, R3, RO	
00000002	< 1.6us	e200	40000518		stw	RO, OxO(R4)	
00000002	1.6us	e200	4000051C		ь	_110	
00000003	2.8us	e200	40000528	\CONTROLR_110	blr)
00000004	< 2.8us	e200	400004D4	\CONTROLR_97	li	RO, 0x0	ptyCommandData->iDeviceNum = 0;
00000004	< 2.8us	e200	400004D8		stw	RO, 0x4(R31)	
00000004	< 2.8us	e200	400004DC	\CONTROLR_98	li	RO, 0x58	ptyCommandData->ucControlChar = 'X'
00000004	< 2.8us	e200	400004 E 0		stb	RO, 0x0(R31)	
00000004	2.8us	e200	400004 E 4	\CONTROLR_100	1i	R3, 0x0	return NO_ERROR;
00000005	90.8us	e200	400004B8	\CONTROLR\WriteToDevice	stwu	R1, -0x30(R1)	(
00000006	< 92.4us	e200	400004BC		mflr	RO	
00000006	< 92.4us	e200	400004C0		stw	RO, 0x34(R1)	
00000006	< 92.4us	e200	400004C4		stw	R31, 0x2C(R1)	
00000006	< 92.4us	e200	400004C8		mr	R31, R4	

Figure 14. Code Trace Window

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.

ዎ Code T	race (Code	e Frames: 5	589)				
Frame	Time	Processor	Address	Symbol	Instruction		Source Line
00000000	-90.8u	e200	400004B8	\CONTROLR\WriteToDevice	stwu	R1, -0x30(R1)	(
00000001	< -89.2u	e200	400004BC		mflr	RO	
00000001	< -89.2u	e200	400004C0		stw	RO, 0x34(R1)	
00000001	< -89.2u	e200	400004C4		stw	R31, 0x2C(R1)	
00000001	< -89.2u	e200	400004C8		mr	R31, R4	
00000001	< -89.2u	e200	400004CC	\CONTROLR_95	addi	R4, R1, 0x10	FindDeviceADDR (ucCommand,
00000001	-89.2u	e200	400004D0		bl	FindDeviceADDR	
00000002	< -89.2u	e200	400004FC	\CONTROLR\FindDeviceADDF	clrlwi	RO, R3, 0x18	if (ucCommand < LAST_COMMAN
00000002	< -89.2u	e200	40000500		cmplwi	RO, 0x5	
00000002	< -89.2u	e200	40000504		bge-	_109	
00000002	< -89.2u	e200	40000508	\CONTROLR_107	rlwinm	RO, RO, 0x2, 0x0,	*puiDeviceAdr = puiDeviceAd
00000002	< -89.2u	e200	4000050C		lis	R3, 0x4000	
00000002	< -89.2u	e200	40000510		addi	R3, R3, 0x4008	
00000002	< -89.2u	e200	40000514		lwzx	RO, R3, RO	
00000002	< -89.2u	e200	40000518		stw	RO, OxO(R4)	
00000002	-89.2u	e200	4000051C		ь	_110	
00000003	-88.0u	e200	40000528	\CONTROLR_110	blr		}
00000004	< -88.0u	e200	400004D4	\CONTROLR_97	li	R0, 0x0	ptyCommandData->iDeviceNum
00000004	< -88.0u	e200	400004D8		stw	RO, Ox4(R31)	
00000004	< -88.0u	e200	400004DC	\CONTROLR_98	1i	RO, 0x58	ptyCommandData->ucControlCh
00000004	< -88.0u	e200	400004E0		stb	RO, 0x0(R31)	
00000004	-88.0u	e200	400004 E 4	\CONTROLR_100	1i	R3, 0x0	return NO_ERROR;
00000005	0.0us	e200	400004B8	\CONTROLR\WriteToDevice	stwu	R1, -0x30(R1)	C
00000006	< 1.6us	e200	400004BC		mflr	RO	
00000006	< 1.6us	e200	400004C0		stw	RO, 0x34(R1)	
00000006	< 1.6us	e200	400004C4		stw	R31, 0x2C(R1)	

Figure 15. Triggering between Two Events. Relative Time Display.

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:\PFMPC\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 **Vitra Trace Buffer Size** to the maximum supported size

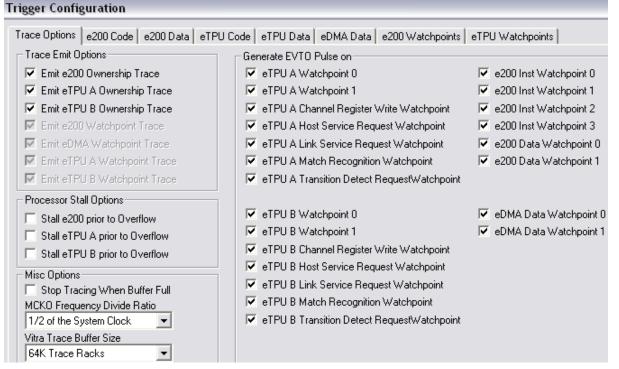


Figure 16. Trace Options Dialog

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:

Trigger Configuration	
Trace Options e200 Code e200 Data eTPU Code	e eTPU Data eDMA Data e200 Watchpoints eTPU Watchpoints
Start Emitting Code Trace	Stop Emitting Code Trace
Never (Code Trace Disabled) On Program Execution	C On Program Halt
On e200 Inst Watchpoint 0	 On e200 Inst Watchpoint 1 ▼
NO If you are using Code Trace with Watchpoint Trace could also cause a Code Trace message to occur message collison will occur internally within the pro-	e, do not set Watchpoints on any instruction that (i.e. BL, BNE, BLR, etc). Otherwise a trace

Figure 17. e200 Code Dialog

- 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)

Trigger Configuration		
e200 Watchpoint Options	e200 Data eTPU Code eTPU Data eDMA Data	e200 '
e200 Instruction Watch Watchpoint 0 Address 40000384 Bro Watchpoint 1 Address 0 Bro Watchpoint 2 Address 0 Bro Watchpoint 3	Code Symbols (*) Code Symbols (*)	

Figure 18. e200 Watchpoints Dialog (setting Watchpoint 0)

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

Trigger Configuration		
e200 Watchpoint Option	e e200 Data eTPU Code eTPU Data eDMA Data s of e200 Watchpoint Resources	e2l
e200 Instruction Watch Watchpoint 0 Address	Browse Code Symbols (*)	
40000384 Bro Watchpoint 1 Address 400004B8 Bro Watchpoint 2 Address 0 Bro Watchpoint 3 Address 0 Bro	mem start start CONTROLR FindDeviceADDR FindDevice WriteToDevice WriteToDevice 95 97 98 100 for main for rand	
e200 Data Watchpoint Watchpoint 0	Symbol Search ────────────────────────────────────	יינ ר

Figure 19. e200 Watchpoints Dialog (setting Watchpoint 1)

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:

Frame	Instruction		Source Line
00000560	lbz	R3, 0x10(R1)	GetNextCommand(&ucCommand,&tyCommandData);
00000560	clrlwi	RO, R3, Ox18	}
00000560	cmplwi	RO, OxO	
00000560	beq+	_48	
00000560	cmplwi	RO, Oxl	if ((ucCommand == READ_DEVICE_A)
00000560	beq-	_55	
00000560	cmplwi	RO, 0x2	
00000560	bne-	_57	
00000561	addi	R4, R1, Ox18	uiError = WriteToDevice (ucCommand, &tyCommandDat
00000561	bl	WriteToDevice	
00000562	stwu	R1, -0x30(R1)	(

Figure 20. Code Trace Window

7. Tracing Code Execution after an Event

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.:

Trigger Configuration		
Trace Options e200 Code e200 Data eTP	JCode eTPU Data eDMA Data e200 Watchpoints	eTPU Watchpoints
Trace Emit Options	Generate EVTO Pulse on	
🔽 Emit e200 Ownership Trace	🔽 eTPU A Watchpoint 0	🔽 e200 Inst Watchpoint 0
🔽 Emit eTPU A Ownership Trace	🔽 eTPU A Watchpoint 1	🔽 e200 Inst Watchpoint 1
🔽 Emit eTPU B Ownership Trace	🔽 eTPU A Channel Register Write Watchpoint	🔽 e200 Inst Watchpoint 2
🔽 Emit e200 Watchpoint Trace	eTPU A Host Service Request Watchpoint	🔽 e200 Inst Watchpoint 3
🔽 Emit eDMA Watchpoint Trace	eTPU A Link Service Request Watchpoint	🔽 e200 Data Watchpoint 0
🔽 Emit eTPU A Watchpoint Trace	eTPU A Match Recognition Watchpoint	💌 e200 Data Watchpoint 1
🔽 Emit eTPU B Watchpoint Trace	eTPU A Transition Detect RequestWatchpoint	
Processor Stall Options		
Stall e200 prior to Overflow	eTPU B Watchpoint 0	🔽 eDMA Data Watchpoint 0
☐ Stall eTPU A prior to Overflow	eTPU B Watchpoint 1	🔽 eDMA Data Watchpoint 1
Stall eTPU B prior to Overflow	eTPU B Channel Register Write Watchpoint	
Misc Options	eTPU B Host Service Request Watchpoint	
Stop Tracing When Buffer Full	eTPU B Link Service Request Watchpoint	
MCKO Frequency Divide Ratio	eTPU B Match Recognition Watchpoint	
1/2 of the System Clock	eTPU B Transition Detect RequestWatchpoint	
Vitra Trace Buffer Size		
1K Trace Racks		

Figure 21. Trace Options Dialog

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:

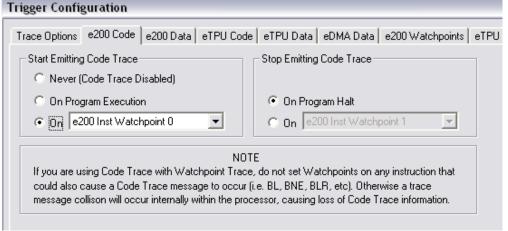


Figure 22. e200 Code Dialog

- 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)

rigger Configuration	
Trace Options 📔 e200 Cod	de e200 Data eTPU Code eTPU Data eDMA Data e200 Watchpoints eTPU Wa
e200 Watchpoint Optic	p£e200 Watchpoint Becources
e200 Instruction Watch Watchpoint 0 Address 40000488 Bro	Code Symbols (*)
Watchpoint 1 Address	Start Control Cancel Cancel
0 Brown	P ReadFromDevice Help Help
Watchpoint 3	97 98 100
Address 0 Bro	Address: 400004B8
e200 Data Watchpoin Watchpoint 0 Address	Symbol Search
0 Bro	Search Expand All
Watchpoint 1 Address	Case Sensitive
0 Bro	tion Watchpoint 2 Event

Figure 23. e200 Watchpoints Dialog (setting Watchpoint 0)

 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:

P	Time	Processor	7.1.1	Symbol	Instruction		Source Line
Frame				-			
00000000	-4.8us				stwu	R1, -0x30(R1)	C
	< -3.2us		400004BC		mflr	RO	
	< -3.2us		400004C0		stw	RO, 0x34(R1)	
	< -3.2us		400004C4		stw	R31, 0x2C(R1)	
	< -3.2us		400004C8		mr	R31, R4	
	< -3.2us			\CONTROLR_95	addi	R4, R1, 0x10	FindDeviceADDR (ucCommand, &uiDe
00000001			400004D0		bl	FindDeviceADDR	
00000002	< -3.2us	e200	400004FC	\CONTROLR\FindDeviceADDR	clrlwi	RO, R3, 0x18	if (ucCommand < LAST_COMMAND)
	< -3.2us		40000500		cmplwi	RO, 0x5	
00000002	< -3.2us	e200	40000504		bge-	_109	
00000002	< -3.2us	e200	40000508	\CONTROLR_107	rlwinm	RO, RO, 0x2, 0x0,	*puiDeviceAdr = puiDeviceAddrLoo
00000002	< -3.2us	e200	4000050C		lis	R3, 0x4000	
00000002	< -3.2us	e200	40000510		addi	R3, R3, 0x4008	
00000002	< -3.2us	e200	40000514		lwzx	RO, R3, RO	
00000002	< -3.2us	e200	40000518		stw	RO, OxO(R4)	
00000002	-3.2us	e200	4000051C		b	_110	
0000003	-1.6us	e200	40000528	\CONTROLR_110	blr		}
00000004	< -1.6us	e200	400004D4	\CONTROLR_97	li	RO, 0x0	ptyCommandData->iDeviceNum =
00000004	< -1.6us	e200	400004D8		stw	RO, 0x4(R31)	
00000004	< -1.6us	e200	400004DC	\CONTROLR_98	li	RO, 0x58	ptyCommandData->ucControlChar =
00000004	< -1.6us	e200	400004E0		stb	RO, 0x0(R31)	
00000004	< -1.6us	e200	400004E4	\CONTROLR_100	1i	R3, 0x0	return NO_ERROR;
00000004	< -1.6us	e200	400004E8		lwz	R31, 0x2C(R1)	
00000004	< -1.6us	e200	400004EC		lwz	RO, 0x34(R1)	
00000004	< -1.6us	e200	400004F0		mtlr	RO	
00000004	< -1.6us	e200	400004F4		addi	R1, R1, 0x30	
00000004	-1.6us	e200	400004F8		blr		
10000005	< 0 0115	e200	40000410		mr	R4 R3	uiRrror = WriteToDevice (ucComms

Figure 24. Code Trace Window

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

Trigger Configuration		
Trace Options e200 Code e200 Data eTPU C	ode eTPU Data eDMA Data e200 Watchpoints	eTPU Watchpoints
Trace Emit Options	Generate EVTO Pulse on	
🔽 Emit e200 Ownership Trace	🔽 eTPU A Watchpoint 0	🔽 e200 Inst Watchpoint 0
🔽 Emit eTPU A Ownership Trace	🔽 eTPU A Watchpoint 1	🔽 e200 Inst Watchpoint 1
🔽 Emit eTPU B Ownership Trace	🔽 eTPU A Channel Register Write Watchpoint	🔽 e200 Inst Watchpoint 2
🔽 Emit e200 Watchpoint Trace	🔽 eTPU A Host Service Request Watchpoint	🔽 e200 Inst Watchpoint 3
🔽 Emit eDMA Watchpoint Trace	🔽 eTPU A Link Service Request Watchpoint	🔽 e200 Data Watchpoint 0
🔽 Emit eTPU A Watchpoint Trace	🔽 eTPU A Match Recognition Watchpoint	🔽 e200 Data Watchpoint 1
🔽 Emit eTPU B Watchpoint Trace	🔽 eTPU A Transition Detect RequestWatchpoint	
Processor Stall Options		
🔲 Stall e200 prior to Overflow	🔽 eTPU B Watchpoint 0	🔽 eDMA Data Watchpoint 0
🔲 Stall eTPU A prior to Overflow	🔽 eTPU B Watchpoint 1	🔽 eDMA Data Watchpoint 1
🔲 Stall eTPU B prior to Overflow	eTPU B Channel Register Write Watchpoint	
Misc Options	eTPU B Host Service Request Watchpoint	
Stop Tracing When Buffer Full	eTPU B Link Service Request Watchpoint	
MCKO Frequency Divide Ratio	eTPU B Match Recognition Watchpoint	
1/2 of the System Clock	eTPU B Transition Detect RequestWatchpoint	
Vitra Trace Buffer Size		
64K Trace Racks		

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:

race Options e200 Code e200 Data eTPU Cod	de [eTPU Data eDMA Data e200 Watchpoints eTPU Watchpoint
Start Emitting Code Trace	Stop Emitting Code Trace
S Never (Code Trace Disabled)	
On Program Execution	On Program Halt
🕐 On 🛛 e200 Inst Watchpoint 0	O On e200 Inst Watchpoint 1
NO If you are using Code Trace with Watchpoint Trac could also cause a Code Trace message to occur message collison will occur internally within the pro	r (i.e. BL, BNE, BLR, etc). Otherwise a trace

Figure 25. Tracing up to a program halt

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:

, -	MPC5566	- ← 🗈 💣 💷		
Name		Date modified	Туре	Size
🐌 Code_T	race	12/02/2013 17:03	File folder	
퉬 Data_Tr	ace	12/02/2013 17:03	File folder	
mpc5566_trace_probe_test.grp		07/02/2013 18:17	GRP File	5 KB
Dia anna anna anna anna anna anna anna a	mpc5566_trace_probe_test.grp Open			
File name:			-	1 Cancel

Browse to the group-file and select **Open**. The test will run and PathFinder will display the results in the **Command** window as follows:

PathFinder for MPC5xxx v1.1.1					
<u>File Run View Trigger Trace Watch Configure Group Files Code Coverage</u>					
🔤 🗀 🎟 🐟 🗢 쑪 🕭 ?" ?" 💺 🏙 🗱 ⊘ 🦻 👻 🖻 ⊅ ⊅ ₮ ₮					
Command					
G >ECHO OFF					
1) - [Disable Modules]					
2) - [Code Trace Test]					
[Code Trace Test: Passed]					
3) - [Data Trace Test]					
[Data Trace Test: Passed]					

******* Trace Diagnostics PASSED ***********************************					
>					

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 <u>ashling.support@nestgroup.net</u>

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 <u>www.ashling.com/support/mpc5500</u> 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