

Debugging Embedded Cores in Xilinx FPGAs [Zynq]

Release 02.2025



TRACE32 Online Help	
TRACE32 Directory	
TRACE32 Index	
TRACE32 Documents	
ICD In-Circuit Debugger	þ
Processor Architecture Manuals	
Arm/CORTEX/XSCALE	Þ
Arm Application Notes	
Debugging Embedded Cores in Xilinx FPGAs [Zynq]	1
Introduction	3
Physical Connection Requirements	4
Requirements for Parallel Trace	4
Requirements for Serial HSSTP Trace	7
Trace-Adapter for FMC-featured Target Boards	8
Zynq-7000 Devices	9
Exporting the Zynq-7000 Trace Interface via FixedIO/MIO	10
Exporting the Zynq-7000 Trace Interface via FPGA Fabric/PL: Using a clock divider	12
Exporting the Zynq-7000 Trace Interface via FPGA Fabric/PL: Using DDR I/O registers	15
Performing a Debugger-Based Boot on the Zynq-7000	19
UltraScale+ Devices	20
Exporting the UltraScale+ Trace Interface via FixedIO/MIO	21
Exporting the UltraScale+ Trace Interface via FPGA Fabric/PL	23
Exporting the UltraScale+ Trace Interface via HSSTP (up to 6.25 Gbps)	26
Exporting the UltraScale+ Trace Interface via HSSTP (10 Gbps)	32
Exporting the UltraScale+ Trace Interface via PCIe	38
Using the Example Design for the ZCU102	39
Performing a Debugger-Based Boot on the Zynq UltraScale+	39

Debugging Embedded Cores in Xilinx FPGAs [Zynq]

Version 13-Feb-2025

Introduction

Some Xilinx FPGAs contain hard processor cores. This document describes how to debug and trace these cores.

The Xilinx **Zynq-7000** and Xilinx **UltraScale+** series contain embedded processor systems that include multiple Arm cores.

This document covers several topics for working with TRACE32 and Xilinx-MPSoC-type SoCs such as **Zynq-7000** or **Zynq Ultrascale+**.

In This Document:

- Physical connection requirements
- How to export the off-chip trace on Zynq-7000
- How to perform a debugger-based boot sequence on the Zynq-7000
- How to export the off-chip trace on Zynq UltraScale+
- How to perform a debugger-based boot sequence on the Zynq UltraScale+

Overview of TRACE32 Commands used in this Application Note:

Analyzer.PortSize	Set port size of physical trace interface, if it differs from TPIU.PortSize
TPIU.PortClock	Set the lane rate for HSSTP serial trace
TPIU.PortMode	Set the HSSTP protocol variant
TPIU.PortSize	Set the number of data pins driven by the Arm CoreSight hardware; Set the number of HSSTP lanes, if applicable
TPIU.RefClock	Configure a reference clock provided by the debugger to the target

Related Documents:

- "Integration for Xilinx Vivado" (int_vivado.pdf) describes how to use Lauterbach PowerDebug hardware tools with Xilinx Vivado.
- "Arm Debugger" (debugger_arm.pdf) describes the processor-specific settings and features for the Cortex-A/R (Armv7, 32-bit) debugger.
- "Armv8 and Armv9 Debugger" (debugger_armv8v9.pdf) describes the processor-specific settings and features for the Cortex-A/R (Armv8, 32/64-bit) debugger.

MPSoC devices use a parallel **TPIU** trace interface to export trace data. This interface can either be exported though PS pins (MIO) or PL pins via the EMIO interface.

For Zynq UltraScale+ FPGAs, this document also provides instructions on how to use the PL portion of the device to convert the parallel interface into a serial HSSTP interface.

Requirements for Parallel Trace

There are two standard connectors for parallel TPIU trace. The first connector is called **Mictor-38**, while the second connector is called **MIPI-60**.

These connectors also include the standard JTAG debug signals. It is possible to either use the JTAG signals on the trace connector or a separate debug connector. We do not recommend routing the JTAG signals to both connectors for reasons of signal integrity.

Pin Name	Description	
VREF-TRACE	Reference voltage for the TRACEDATA[] , TRACECLK and TRACECTL lines	MANDATORY
TRACECLK	Clock line for the trace interface (DDR)	MANDATORY
TRACECTL	Control line for the trace interface	OPTIONAL
TRACEDATA[x:0]	Data lines for the trace interface	MANDATORY

The required pins for the off-chip trace connection are shown in the table below.

For the MPSoC devices, we recommend that you use 16 data lines (**TRACEDATA**[15:0]). The optional pin **TRACECTL** is only required for use with **Wrapped** mode (also called **Normal** mode in Arm terminology). As all Lauterbach tools work equally well with the **Continuous** mode, we exclusively use this mode in this application note and don't require **TRACECTL**. For more information about the connectors and available adaptors, visit:

- Mictor-38: www.lauterbach.com/adetmmictor.html
- MIPI-60: www.lauterbach.com/adetmmipi60.html

Signal	Pin	Pin	Si
N/C	1	2	N/
N/C	3	4	N/
GND	5	6	TF
DBGRQ	7	8	DI
RESET-	9	10	EX
TDOI-ISWO	11	12	VI
RTCK	13	14	VI
TCKITCKCISWCLK	15	16	TF
TMSITMSCISWDIO	17	18	TF
TDI	19	20	TF
TRST-	21	22	TF
TRACEDATA[15]	23	24	TF
TRACEDATA[14]	25	26	TF
TRACEDATA[13]	27	28	TF
TRACEDATA[12]	29	30	G
TRACEDATA[11]	31	32	G
TRACEDATA[10]	33	34	V
TRACEDATA[9]	35	36	TF
TRACEDATA[8]	37	38	TI

Signal

/C /C RACECLK BGACK XTRIG REF-TRACE REF-DEBUG RACEDATA[7] RACEDATA[6] RACEDATA[5] RACEDATA[4] RACEDATA[3] RACEDATA[2] RACEDATA[1] ND ND СС RACECTL RACEDATA[0]

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	TMSITMSCISWDIO
TCKITCKCISWCLK	3	4	TDOI-ISWO
TDI	5	6	RESET-
RTCK	7	8	TRST- PULLDOWN
TRST-	9	10	DBGRQ TRIGIN
DBGACK TRIGOUT	11	12	VREF-TRACE
TRACECLK	13	14	GND
GND	15	16	GND
TRACECTL	17	18	TRACEDATA[19]
TRACEDATA[0]	19	20	TRACEDATA[20]
TRACEDATA[1]	21	22	TRACEDATA[21]
TRACEDATA[2]	23	24	TRACEDATA[22]
TRACEDATA[3]	25	26	TRACEDATA[23]
TRACEDATA[4]	27	28	TRACEDATA[24]
TRACEDATA[5]	29	30	TRACEDATA[25]
TRACEDATA[6]	31	32	TRACEDATA[26]
TRACEDATA[7]	33	34	TRACEDATA[27]
TRACEDATA[8]	35	36	TRACEDATA[28]
TRACEDATA[9]	37	38	TRACEDATA[29]
TRACEDATA[10]	39	40	TRACEDATA[30]
TRACEDATA[11]	41	42	TRACEDATA[31]
TRACEDATA[12]	43	44	GND
TRACEDATA[13]	45	46	GND
TRACEDATA[14]	47	48	GND
TRACEDATA[15]	49	50	GND
TRACEDATA[16]	51	52	GND
TRACEDATA[17]	53	54	GND
TRACEDATA[18]	55	56	GND
GND	57	58	GND
GND	59	60	GND

When exporting a HSSTP trace interface, a 40-pin SAMTEC connector is commonly used. The connector also includes the standard JTAG debug signals. It is possible to either use the JTAG signals on the trace connector or a separate debug connector. We do not recommend routing the JTAG signals to both connectors for reasons of signal integrity.

The connector includes six TXP/TXN pairs. With the UltraScale+ target, there is no benefit in using more than two lanes.

The TX lanes are terminated by the PowerTrace serial or the serial preprocessor. On the target PCB, they should be routed directly between the connector and the FPGA as a 50 Ω ifferential pair.

For more information about the target connector, visit www.lauterbach.com/adetmhsstp.html

The FPGA requires a reference clock for its gigabit transceivers. A clock whose frequency is 1/20 of the bit rate can be provided by the debugger on the CLKP/CLKN pins. It is also possible to provide a clock generator on the target. Check the Xilinx documentation for the allowable frequencies.

Signal TXP[4] **TXN**[4] GND TXP[2] TXN[2] GND TXP[0] TXN[0] GND CLKP CLKN GND TXP[1] TXN[1] GND TXP[3] TXN[3] GND TXP[5] **TXN**[5]

Pin	Pin
1	2
3	4
5	6
7	8
9	10
11	12
13	14
15	16
17	18
19	20
21	22
23	24
25	26
27	28
29	30
31	32
33	34
35	36
37	38
39	40

Signal **VRFF-DFBUG** TCKITCKCISWCLK GND **TMSITMSCISWDIO** TRST-GND TDI TDOI-ISWO GND RESET-DBGRQ GND DBGACK RTCK GND TRIGIN TRIGOUT RESERVED RESERVED RESERVED

Many common evaluation boards are featured with a FMC connector (FPGA Mezzanine Card). Lauterbach offers a HSSTP/MICTOR-to-FMC converter, which allows to connect tools for parallel and serial trace. The order number is LA-2785.





Lauterbach supports three methods for exporting the off-chip trace interface of Zynq-7000 devices:

- 1. Directly using the FixedIO/MIO pins of the Processing System (PS)
- 2. Routing trace data through the Programmable Logic (PL) portion of the SoC, dividing the clock by two
- 3. Routing trace data through the PL portion of the SoC, dividing the data width by two and using DDR output registers to increase the bandwidth

Method 1 is the only method that works while the PL is not yet programmed. However, it is limited to a trace port width of 16 bits. In addition, the signal quality of PS pins is worse than on the I/Os belonging to the PL.

Method 2 allows trace port widths up to 32 bits and trace clock frequencies up to 125 MHz. It consumes a small amount of PL resources for pipeline registers and routing. A clock divider is used to generate the TRACECLK signal from the internal single data rate trace clock. The maximum operating frequency is limited by the maximum clock frequency of the PS-PL TPIU interface.

Method 3 is similar to method 2, but instead of dividing the trace clock by two, the width of the data path is reduced internally. This allows using an external 16-bit trace port by internally utilizing the full 32-bit PS-PL TPIU interface. Alternatively, an external 8-bit port can also be created. The maximum frequency achievable using this method is 250 MHz, which corresponds to 1 GB/s (decimal) when used with a 16-bit trace port.

NOTE: We recommend using method 2 or 3. Note that these methods are interchangeable for a given PCB design. Designing a PCB to use method 1 limits the achievable trace bandwidth.

The remainder of this section contains a step-by-step procedure to each of these methods, followed by instructions related to board bring-up using a debugger.

Exporting the Zynq-7000 Trace Interface via FixedIO/MIO

- 1. Create a new Vivado project with an instance of the Zynq processing system.
- 2. Enter the configuration of the Zynq processing system.
- 3. Enable the trace pin export via MIO by selecting the desired port size, see [A] in the figure below.
- 4. Select the external pins that are connected to the trace port, see [**B**] in the figure below.

E Re-customize IP	Re-customize IP								
ZYNQ7 Process	(YNQ7 Processing System (5.5)								
<i> Documentation</i>	Presets 📄 IP Location 🍈 Import XPS Settin	gs							
Page Navigator 🛛 ≪	MIO Configuration						Sum	mary Report	
Zynq Block Design	Bank 0 I/O Voltage LVCMOS 3.3V	•	Bank 1 I	O Voltage LVCM	DS 3.3V	-			
PS-PL Configuration	Search: Q-								
Peripheral I/O Pins	Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction	Polarity	
MIO Configuration	■ Memory Interfaces								
Clock Configuration	I/O Peripherals I/O Peripherals Application Processor Unit								
DDR Configuration		MIO 12 13 🔻	в						
SMC Timing Calculation	d[0:1]	MIO 14 15 🔻							
Interrupts	🖶 🛛 📝 d[2:3]	MIO 1011 🔻							
		MIO 16 19 🔻							
	⊕ 💽 d[8:15	MIO 2 9 🛛 🔻							
	d[16:3								
	Trace	MIO 12	trace_clk	LVCMOS 3.3V	slow .	r enabled	▼ out		
	Trace	MIO 13	trace_ctl	LVCMOS 3.3V	slow	r enabled	▼ out		
	PJTAG								
						ĺ	ОК	Cancel	

5. Select an internal clock source (Arm PLL, DDR PLL or IO PLL) and the desired frequency for the TPIU (Trace Port Interface Unit).

Please note that the exported TRACECLK is a DDR clock signal whose actual frequency will be half the frequency selected in this configuration window:

Re-customize IP						×		
ZYNQ7 Processing System (5.5)								
🎁 Documentation 🦓 🛛	Presets	s 🛅 IP Location 🍈 Impo	rt XPS Settings					
Page Navigator 🛛 🐇	Cloc	k Configuration				Summary Report		
Zynq Block Design		Basic Clocking Advanced	d Clocking					
PS-PL Configuration	+	Input Frequency (MHz) 33	.333333 🚫 C	CPU Clock Ratio 6:2:1	•			
Peripheral I/O Pins		<u>S</u> earch: Q _▼						
MIO Configuration	 	Component	Clock Source	Requested Frequency(MHz)	Actual Frequency(MHz)	Range(MHz)		
Clock Configuration	-	Processor/Memory Clo IO Peripheral Clocks	cks					
DDR Configuration		PL Fabric Clocks System Debug Clocks						
SMC Timing Calculation		TPIU	ARM PLL	200 🛞	190.476196	10.000000 : 300.000000		
Interrupts		⊕ Timers						
						OK Cancel		

- 6. Finish your Vivado design and export the project to the SDK.
- 7. Generate or regenerate your FSBL (first-stage boot loader).
- 8. Do one of the following:
 - Either program the resulting FSBL to the boot device,
 - Or perform a debugger-based boot (see **Performing a Debugger-Based Boot on the Zynq-7000**).

Exporting the Zynq-7000 Trace Interface via FPGA Fabric/PL: Using a clock divider

- 1. Create a new Vivado project with an instance of the Zynq processing system.
- 2. Enter the configuration of the Zynq processing system.
- 3. Enable the trace pin export via MIO by selecting the desired port size and the pins that will be connected to the trace connector.

Re-customize IP	Re-customize IP								
ZYNQ7 Process	ZYNQ7 Processing System (5.5)								
<i> Documentation</i>	Presets 🛅 IP Location 🍈 Import XPS Settings								
Page Navigator 🛛 🔍	MIO Configuration						Sum	mary Report	
Zynq Block Design	Bank 0 I/O Voltage LVCMOS 3.3V		Bank 1 I	/O Voltage LVCN	4OS 3.3V	-			
PS-PL Configuration	Search: Q-								
Peripheral I/O Pins	Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction	Polarity	
MIO Configuration	🕞 🕀 Memory Interfaces								
Clock Configuration	I/O Peripherals Application Processor Unit								
DDR Configuration	Programmable Logic Test and Debug	EMIO 💌							
SMC Timing Calculation	📝 d[0:1]	EMIO							
Interrupts	📝 d[2:3]	EMIO							
	📝 d[4:7]	EMIO							
	📝 d[8:15]	EMIO							
	d[16:31]								
	PJTAG								
							ОК	Cancel	

4. Activate at least one of the **FCLK_CLK**<*x*> clock signals, which will later be used as the TPIU clock. The exported TRACECLK will be half this frequency. Select **External** as the TPIU clock source.

Re-customize IP									
ZYNQ7 Processing System (5.5)									
🍘 Documentation 🆚 Presets 늡 IP Location 🏨 Import XPS Settings									
Page Navigator 🛛 🛠	Cloc	k Configuration				Summary Report			
Zynq Block Design		Basic Clocking Advanced Clock	king						
PS-PL Configuration	+	Input Frequency (MHz) 33.3333	33 🙁	CPU Clock Ratio 6:2:1	•				
Peripheral I/O Pins	Q 3	Search: Q-							
MIO Configuration	e	Component	Clock Source	Requested Frequency	(MHz) Actual Frequency(MHz) Range(MHz)			
Clock Configuration		Processor/Memory Clocks IO Peripheral Clocks							
DDR Configuration		PL Fabric Clocks	ADM DU	- 200	0 100 476106	0 100000 - 250 000000			
SMC Timing Calculation			IO PLL	50	10.000000	0.100000 : 250.000000			
Interrupts		- FCLK_CLK2	IO PLL	50	10.000000	0.100000:250.000000			
		FCLK_CLK3	IO PLL	50	10.000000	0.100000:250.000000			
		System Debug Clocks							
		TPIU	External	▼ 200	200.00000	10.000000 : 300.000000			
		±" Timers							
						OK Cancel			

5. Connect and export the signals as follows:



To use continuous mode, we do not need the **TRACE_CTL** signal. The **TRACE_CLK_OUT** signal will be generated by the HDL wrapper. We rename **FCLK_CLK0** to **TRACE_CLK_SDR** and will use this signal to sample **TRACE_DATA**.

- 6. Finish your block design and generate the HDL wrapper.
- 7. Modify the HDL wrapper to include the pipeline registers and DDR clock generation for routing the PS trace interface to PL pins:

```
entity zynq_wrapper is
    port (
        oTraceClkDdr: out std_logic;
        oTraceData: out std_logic_vector(15 downto 0)
     );
end entity;
architecture SDR of zynq_wrapper is
    signal wTraceClkSdr: std_logic;
    signal wTraceData: std_logic_vector(15 downto 0);
    signal rTraceClkDdr: std_logic;
    signal rTraceData_q: std_logic_vector(15 downto 0);
    signal rTraceData_qq: std_logic_vector(15 downto 0);
begin
    zynq_i: entity work.zynq port map (
        TRACE_CLK_SDR => wTraceClkSdr,
        TRACE DATA => wTraceData
    );
    trace_pipeline: process(wTraceClkSdr)
    begin
        if rising_edge(wTraceClkSdr) then
            rTraceClkDdr <= not rTraceClkDdr;
            rTraceData_qq <= rTraceData_q;</pre>
            rTraceData_g <= wTraceData;</pre>
        end if;
    end process;
    oTraceData <= rTraceData gg;
    oTraceClkDdr <= rTraceClkDdr;
end architecture;
```

- 8. Assign the **oTraceData[x:0]** and **oTraceClkDdr** to the appropriate FPGA pins matching your board layout. Select the same I/O standard for all pins and the slew rate appropriate for the desired trace port speed.
- 9. Finish your Vivado design and export the project to the SDK.
- 10. Generate or regenerate your FSBL (first-stage boot loader).
- 11. Do one of the following:
 - Either program the resulting FSBL to the boot device,
 - Or perform a debugger-based boot (see "Performing a Debugger-Based Boot on the Zynq-7000", page 19).

Exporting the Zynq-7000 Trace Interface via FPGA Fabric/PL: Using DDR I/O registers

- 1. Create a new Vivado project with an instance of the Zynq processing system.
- 2. Enter the configuration of the Zynq processing system.
- 3. Enable the trace pin export via MIO by selecting twice the desired port size and the pins that will be connected to the trace connector. To use this method, you must select d[0:1] ... d[16:31] for an external 16-bit trace port or d[0:1] ... d[8:15] for an external 8-bit trace port.

Re-customize IP										
ZYNQ7 Process	ZYNQ7 Processing System (5.5)									
🎁 Documentation 🏀	🍘 Documentation 🔞 Presets 늡 IP Location 🍈 Import XPS Settings									
Page Navigator 《	MIC) Configuration						Sum	nary Report	
Zynq Block Design	-	Bank 0 I/O Voltage LVCMOS 3.3	v –	Bank 1 I	/O Voltage LVCN	10S 3.3V	-			
PS-PL Configuration		Search: Q-					_			
Peripheral I/O Pins		Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction	Polarity	
MIO Configuration	•	Memory Interfaces								
Clock Configuration	0	I/O Peripherals Application Processor Unit								
DDR Configuration		Programmable Logic Test and	d Debug							
		TRACE	EMIO 🔻							
SMC Timing Calculation		📝 d[0:1]	EMIO	_						
Interrupts		📝 d[2:3]	EMIO							
		📝 d[4:7]	EMIO							
		d[8:15]	EMIO							
		d[16:31]	EMIO							
		РЈТАG								
								ОК	Cancel	

4. Activate at least one of the **FCLK_CLK**<*x*> clock signals, which will later be used as the TPIU clock. The exported TRACECLK will be using this frequency. Select **External** as the TPIU clock source.

- Re-customize IP									
ZYNQ7 Processing System (5.5)									
🍘 Documentation 🆚 Presets 늡 IP Location 🎕 Import XPS Settings									
Page Navigator 《	Cloc	ck Configuration				Summary Report			
Zynq Block Design		Basic Clocking Advanced Clock	ting						
PS-PL Configuration	+	Input Frequency (MHz) 33.3333	33 区	CPU Clock Ratio 6:2:1	–				
Peripheral I/O Pins		Search: Q.*]					
MIO Configuration	a	Component	Clock Source	Requested Frequency(M	Hz) Actual Frequency(MHz)	Range(MHz)			
Clock Configuration	-	Processor/Memory Clocks IO Peripheral Clocks							
DDR Configuration		PL Fabric Clocks							
SMC Timing Calculation		FCLK_CLK0	ARM PLL	200	× 190.476196	0.100000:250.000000			
Interrupts			TO PLL	50	10.000000	0.100000:250.000000			
Interrupto			TO PLL	50	10.000000	0.100000:250.000000			
		System Debug Clocks	IOPLL	50	10.00000	0.100000:250.000000			
		TPIU	External	▼ 200	200.000000	10.000000 : 300.000000			
				· · · · · · · · · · · · · · · · · · ·					
						OK Cancel			

5. Connect and export the signals as follows:



To use continuous mode, we do not need the **TRACE_CTL** signal. The **TRACE_CLK_OUT** signal will be generated by the HDL wrapper. We use **FCLK_CLK0** to sample **TRACE_DATA**.

- 6. Finish your block design and generate the HDL wrapper.
- 7. Locate the file ~~/demo/arm/hardware/zynq-7000/parallel_trace_adapter.vhd from your TRACE32 installation and add it to the project.
- 8. Modify the HDL wrapper to include the parallel_trace_adapter for routing the PS trace interface to

PL pins:

```
entity zynq_wrapper is
   port (
      oTraceClkDdr: out std logic;
      oTraceData: out std_logic_vector(15 downto 0)
   );
end entity;
architecture DDR of zynq_wrapper is
   signal wTraceClkSdr: std logic;
   signal wTraceData: std_logic_vector(31 downto 0);
begin
   zynq_i: entity work.zynq port map (
      FCLK_CLK0 => wTraceClkSdr,
TRACE_DATA => wTraceData
   );
   adapter i: entity work.parallel trace adapter generic map (
      gPlatform => "ZYNQ7000",
      gBitsIn => 32,
      gBitsOut => 16
   ) port map (
              => wTraceClkSdr,
      iClk
      iData => wTraceData,
oClk => oTraceClkDd
oData => oTraceData
                => oTraceClkDdr,
   );
end architecture;
```

- Assign the oTraceData[x:0] and oTraceClkDdr to the appropriate FPGA pins matching your board layout. Select the same I/O standard for all pins and the slew rate appropriate for the desired trace port speed.
- 10. Finish your Vivado design and export the project to the SDK.
- 11. Generate or regenerate your FSBL (first-stage boot loader).
- 12. Do one of the following:
 - Either program the resulting FSBL to the boot device,
 - Or perform a debugger-based boot (see "**Performing a Debugger-Based Boot on the Zynq-7000**", page 19).

NOTE:To use the off-chip trace port from TRACE32, you need to apply the following
special settings:TPIU.PortSize32 ; internal port size (PS -> PL),
; same as gBitsIn
Analyzer.PortSize 16 ; external port size (PL -> TRACE32),
; same as gBitsOut

This section focuses on the **JTAG-BOOT** mode of the Zynq-7000. In contrast to all other boot modes, this mode is only intended for development. The basic idea is that the CPUs will wait in an endless loop after executing the boot ROM, allowing the JTAG probe to perform all further initialization.

To perform a debugger-based boot:

- 1. Set the boot mode to **JTAG-BOOT** using the MIO lines.
- 2. Reset the SoC, for example by asserting the RESET line. Not all boards have a RESET line connected to the SoC, thus a power cycle or similar might be required.
- 3. Execute the boot ROM.
- 4. Load the FSBL boot code using the debugger.
- 5. Execute the FSBL boot code.
- 6. Optionally load the FPGA fabric using the debugger.

You are now ready to load the next stage boot loader, OS, etc., and to use the optional off-chip trace.

Example files following the above sequence are included in the TRACE32 installation directory under ~~/demo/arm/hardware/zynq-7000 On the Zynq UltraScale+ series, Lauterbach currently supports four ways to export trace data:

- 1. Parallel trace exported using the FixedIO/MIO pins of the Processing System (PS)
- 2. Parallel trace routed through the EMIO pins and the Programmable Logic (PL) portion of the SoC
- 3. HSSTP (serial) trace via PL gigabit transceivers (GTH)
- 4. PCIe (serial) trace via PS gigabit transceivers (GTR)

Each method uses a different kind of SoC resource that is not available to the application while the application is being traced. The following table provides an overview of different configurations, their resource usage, achievable data rate and required Lauterbach hardware.

Method	SoC Resources	PL Used	Data Rate	Lauterbach Trace Hardware
1 (16 bit @125 MHz DDR)	17 PS I/Os	no	500 MB/s	PowerTrace or
2 (16 bit @ 250 MHz DDR)	17 PL I/Os	yes	1000 MB/s	Power Irace II / PowerTrace III /
2 (8 bit @ 250 MHz DDR)	9 PL I/Os	yes	500 MB/s	PowerTrace PX with AutoFocus preprocessor
3 (2 Lanes @ 6.25 Gbps)	2 GTH	yes	1000 MB/s	PowerTrace II /
3 (1 Lane @ 6.25 Gbps)	1 GTH	yes	625 MB/s	Power Irace III / PowerTrace PX with serial preprocessor or PowerTrace Serial
3 (1 Lane @ 10Gbps)	1 GTH	yes	1000 MB/s	PowerTrace
4 (PCle v2 ×2)	2 GTR	no	1000 MB/s	Serial
4 (PCIe v2 ×1)	1 GTR	no	500 MB/s	

NOTE:	 Lauterbach recommends using one of the methods with a data rate of 1000 MB/s, especially if a multi-core program is to be traced. Due to the complexity involved in setting up PCIe trace, we only recommend this method if a PCIe expansion slot is already present on the target board.
-------	--

This section includes a step-by-step introduction for each method as well as a step-by-step introduction for performing a debugger-based boot sequence.

Exporting the UltraScale+ Trace Interface via FixedIO/MIO

- 1. Create a new Vivado project with an instance of the Zynq processing system.
- 2. Enter the configuration of the Zynq processing system.
- 3. Enable the trace pin export via MIO by selecting the desired port size and the pins that will be connected to the trace connector.

📑 Re-customize IP		x						
ZYNQ UltraScale+ MPSoc (1.1)								
🎁 Documentation 🛅 IP Lo	ocation							
Page Navigator 《	I/O Configuration							
Switch To Advanced Mo	🖕 🔺 MIO Voltage Standard							
PS UltraScale + Block Design	Bank0 [MIO 0:25] Bank1 [MIO 26:51] Bank2 [MIO 52:77] LVCMOS33 * LVCMOS33 * LVCMOS33 *							
I/O Configuration	Search: Q							
Clock Configuration	Peripheral I/O Signal I/O Type Drive Strength(mA) Speed Pull Type Direction	1						
DDR Configuration	E-Low Speed	-						
PS-PL Configuration		E						
	TRACE MILOU dk schmitt 12 slow pullup out							
	TRACE MIO1 cti schmitt 12 slow pullup out							
	TDACE MTO2 tracede[1] admitt 12 Slow Dullup out	-						
	OK Cance	el						

4. Select an internal clock source (IOPLL, DPLL, or APLL) and the desired frequency for DBG Trace. Please note that the exported **TRACECLK** is a DDR clock signal whose actual frequency will be half the frequency selected in this configuration window.

FRe-customize IP						×			
ZYNQ UltraScale+ MPSoc (1.1)									
🎁 Documentation 📄 IP Lo	ocation	1							
Page Navigator 🛛 🐇	Cloc	ck Configuration							
Switch To Advanced Mo		Automatic Mode Manual Mode							
PS UltraScale + Block Desigr	4	Search: Q-							
I/O Configuration		Name	Source	Requested Freq (MHz)	Actual Freq (MHz)	Range (MHz)			
Clock Configuration		PLL reference Clocks Dower Domain Clocks							
DDR Configuration									
PS-PL Configuration		System Debug Clocks							
		DBG FPD DBG Trace DBG TSTMP	APLL	200	249.997 199.998 249.997	0.000000 : 250.000000 0.000000 : 250.000000 0.000000 : 250.000000			
						OK Cancel			

- 5. Finish your Vivado design and export the project to the SDK.
- 6. Generate or regenerate your FSBL (first-stage boot loader).
- 7. Do one of the following:
 - Either program the resulting FSBL to the boot device,

- Or perform a debugger-based boot (see "**Performing a Debugger-Based Boot on the Zynq UltraScale+**", page 39).

Exporting the UltraScale+ Trace Interface via FPGA Fabric/PL

- 1. Create a new Vivado project with an instance of the Zynq processing system.
- 2. Enter the configuration of the Zynq processing system.
- 3. Enable the trace pin export via **EMIO**. Select **32Bit** as the trace width. This setting will later be overridden by the debugger.

📭 Re-customize IP	F Re-customize IP												
ZYNQ UltraScale+	ZYNQ UltraScale+ MPSoc (2.0)												
🎁 Documentation 🛅 IP Lo	ocatio	on											
Page Navigator 《	I/O	Configurat	ion										
Switch To Advanced Mo	•	▲ MIO Volt	age Star	ndard									
	0	Bank0	[MIO 0:	25] Bar	nk1 [MIO 26:5	1] Bar	nk2 [MIO 52:77]						
PS UltraScale + Block Desigr		LVCMC	0533	▼ LVC	CMOS33	▼ LVC	CMOS33 🔹 🔻						
I/O Configuration		Peripheral			I/O		Signal		I/O Type	Drive Strength(mA)	Speed	Pull Type	Direction
Clock Configuration	0	-Low Spe	ed										
200 G G U		⊕ Mem	ory Inte	rfaces									
DDR Configuration		⊞ I/OF	Periphera	als									
PS-PL Configuration		⊡-Proc	essing U WDT	nit			_						
			Trace		EMIO	Ŧ							
			Trace	Width	32Bit	Ŧ							
		±∙⊤	TC				-						
		High Spe	eed										
		teren	ice Clock	S									
												ОК	Cancel

 Activate at least one of the PL Fabric Clocks (for example PL0), which will later be used as the TPIU clock. The exported TRACECLK will be using this frequency. Select the same frequency for the DBG_TRACE clock.

🕂 Re-customize IP		×							
ZYNQ UltraScale+	ZYNQ UltraScale+ MPSoc (2.0)								
🍘 Documentation 💼 IP Location									
Page Navigator 《	Clock Configuration								
Switch To Advanced Mo	Input Clocks Output Clocks								
PS UltraScale + Block Desigr	Enable Manual Mode								
I/O Configuration	PLL Options								
Clock Configuration	Requested Freq (MHz) Divisor 1 Divisor 2 Actual Frequency (MHz) R	lange							
DDR Configuration	⊟-Low Power Domain Clocks ⊕-Processor/Memory Clocks								
PS-PL Configuration	Peripherals/IO Clocks								
		000000 + 400 000000							
		000000 : 400.000000							
	PI2 RPI 100 4 1 324.997 0.	.000000 : 400.000000							
	PL3 RPLL 100 4 1 324,997 0.1	.000000 : 400.000000							
	± System Debug Clocks								
	🖃 Full Power Domain Clocks								
	Processor/Memory Clocks								
	🗄 Peripherals/IO Clocks								
	⊡-System Debug Clocks								
	DBG_FPD <u>8</u> 2 249.997 0.	.000000 : 250.000000							
	DBG_TRACE IOPLL 250 2 249.997 0.1	.000000 : 250.000000							
	DBG_TSTMP 2 249.997 0.	000000 : 250.000000							
	the Advance Clocks								
	Of	K Cancel							

5. Connect and export the signals as follows



To use continuous mode, we do not need the **TRACE_CTL** signal. The **trace_clk_out** signal will be generated by the HDL wrapper. We rename **pl_clk0** to **TRACE_CLK_SDR** and will use this signal to sample **TRACE_DATA**.

- 6. Finish your block design and generate the HDL wrapper.
- Locate the file ~~/demo/arm/hardware/zynq_ultrascale/hdl/parallel_trace_adapter.vhd from your TRACE32 installation and add it to the project.
- 8. Modify the HDL wrapper to include the parallel trace adapter that exports the trace port without a clock divider:

```
library IEEE:
use IEEE.std logic 1164.all;
entity zynq_wrapper is
    port (
        oTraceClk: out std logic;
        oTraceData: out std logic vector(15 downto 0)
    );
end entity;
architecture behavioral of zyng wrapper is
    signal wTraceClkSdr: std logic;
    signal wTraceDataSdr: std logic vector(31 downto 0);
begin
    yZynq: entity work.zynq port map (
        TRACE CLK SDR => wTraceClkSdr,
        TRACE DATA => wTraceDataSdr
    );
    yAdapter: entity work.parallel_trace_adapter generic map (
        gPlatform
                     => "ULTRASCALE",
        gBitsIn
                     => 32,
                     => 16
        aBitsOut
    ) port map (
        iClk
                     => wTraceClkSdr,
        iData
                     => wTraceDataSdr,
        oClk
                     => oTraceClk,
        oData
                     => oTraceData
    );
end architecture;
```

9. Assign the **oTraceData[x:0]** and **oTraceClk** to the appropriate FPGA pins matching your board layout. Select the same I/O standard for all pins and the slew rate appropriate for the desired

trace port speed.

- 10. Finish your Vivado design and export the project to the SDK.
- 11. Generate or regenerate your FSBL (first-stage boot loader).
- 12. Do one of the following:
 - Either program the resulting FSBL to the boot device,
 - Or perform a debugger-based boot (see "Performing a Debugger-Based Boot on the Zynq UltraScale+", page 39).
- 13. Use the following **TPIU** and **Analyzer** commands in your PRACTICE start-up script (*.cmm) to configure the trace:

```
TPIU.PortSize 32 ; internal port size (PS -> PL),
        ; same as gBitsIn
Analyzer.PortSize 16 ; external port size (PL -> TRACE32),
        ; same as gBitsOut
```

Exporting the UltraScale+ Trace Interface via HSSTP (up to 6.25 Gbps)

Use the following step-by-step procedure to produce an FPGA design that supports tracing the Arm cores via HSSTP at a bit rate of 6.25 Gbps or lower. This is the maximum supported data rate of the serial preprocessor LA-7988. If you use the PowerTrace Serial, LA-3520, you may want to consider using a higher bit rate, as described in the next section, "Exporting the UltraScale+ Trace Interface via HSSTP (10 Gbps)", page 32.

Note that the bandwidth of the trace infrastructure of the PS is limited to 1000 MB/s. Therefore, using more than two lanes is only beneficial if the lane rate is limited by other factors.

- Perform steps 1 through 6 of the previous section, "Exporting the UltraScale+ Trace Interface via FPGA Fabric/PL", page 23. Depending on the desired number of lanes, select an appropriate trace port frequency to avoid FIFO overflows:
 - For one lane at 6.25 Gbps, set the frequency to 150 MHz.
 - For two lanes at 6.25 Gbps, set the frequency to 250 MHz.
 - For any other setup, calculate the maximum frequency as:

 $f_{\text{max}} = (\text{number of lanes}) \cdot (\text{lane rate}) \cdot \frac{1 \text{ B}}{10 \text{ bit}} \cdot \frac{1 \text{ Hz}}{4 \text{ B/s}}$

The selected frequency must be lower than this frequency and at most 250 MHz.

2. Create a second block design to hold the Xilinx Aurora encoder and a FIFO. In this example, it is called **aurora_and_fifo**.

- 3. In the block design, take the following steps to instantiate an Aurora 8B10B core:
 - Select the desired line rate and set the Lane width to 4.
 - Choose the number of lanes and the location of the GT lane(s) that are connected to the trace

port.

- Select appropriate clock sources for the reference and initialization clocks.
- It is also important to select Little Endian Support.

💶 Re-customize IP		23
Aurora 8B10B (11.0)	1	2
鎻 Documentation 📄 IP Location		
Show disabled ports	Component Name aurora_with_fifo_aurora_8b 10b_0_0	
	Core Options Shared Logic	
	Physical Layer	
	Starting GT Quad Quad X0Y3	
	Starting GT Lane X0Y12 VII [Selected GTs X0Y12 - X0Y13]	
	GT Refclk Selection MGTREFCLK1 of Quad X0Y3 👻	
	GT Refclk (MHz) 156.25	
	Manual INIT dk (MHz) 74.25 [6.25 - 156.25]	
HOUSER_DATA_S_AXI_CORE_STATUS	Link Layer	
⊕GT_DIFF_REFCLK1 GT_SERIAL_TX ⊕GT1 DRP user clk out =	Dataflow Mode TX-only Simplex 🔻	
CORE_CONTROL sys_reset_out	Interface Framing	
gt_resetgt_reset	Flow Control None 🔻	
	Back Channel Timer 💌	
	Scrambler/Descrambler	

4. On the second page of the configuration window, select **include Shared Logic in core**.

Re-customize IP		×
Aurora 8B10B (11.0)		
Documentation 📄 IP Location		
Show disabled ports	Component Name aurora_with_fifo_aurora_8b10b_0_0	
	Core Options Shared Logic	
# @USER_DATA_S_AXI_00RE_STATUS ⊕	Shared Logic	
	Select whether the transceiver quad PLL, transceiver differential refclk buffer, docking and reset logic are included in the core itself or in the example design	
-gt_reset gt_reset_out -	include Shared Logic in core	
	include Shared Logic in example design	-
	ОК	Cancel

5. In the same block design, instantiate an AXI4-Stream Data FIFO. Set **TDATA Width** to four times the number of lanes. The FIFO is used for synchronisation between the Aurora user clock and the trace clock and to buffer data while the Aurora core is busy sending clock compensation sequences and frame delimiters. It is sufficient to make it 32 entries deep.

FRe-customize IP		x
AXI4-Stream Data FIFO (1.1)		
f Documentation 🛅 IP Location		
Show disabled ports	Component Name aurora_with_fifo_axis_data_fifo_0_0	
	FIFO Depth 32 🔻	
	Enable Packet Mode	
	Asynchronous Clocks Yes 💌	
	Synchronization Stages across Clock Domain Logic	
	ACLKEN Conversion Mode	
₩ + S_AXIS M_AXIS + #	Signal Properties	
-cm_axis_aresetn axis_data_count[31:0]=	Manual TDATA Width (bytes 8 S - 512)	
s_axis_aclk axis_rd_data_count[31:0] =	Manual Enable TSTRB No	
	Manual Enable TKEEP Yes V	
	OK Cancel	

6. Connect the two components and export the pins as follows:



- 7. Locate the directory ~~/demo/arm/hardware/zynq_ultrascale/hdl/ from your TRACE32 installation and add the following files to the project:
 - serial_trace_adapter_axis.vhd
 - stream_width_expander.vhd

- tpiu_to_stream.vhd
- 8. Modify the generated wrapper for the Zynq UltaScale+ MPSoC to include both block designs and an adapter from the HDL files imported in step 7:

```
library IEEE;
use IEEE.std logic 1164.all;
entity zynq_wrapper is
    generic (
        -- adapt this to match the FIFO configuration
        qBytes:
                        positive := 8;
        -- adapt this to match the desired number of HSSTP lanes
        gLanes:
                         positive := 2
    );
    port (
        iInitClk:
                    in std_logic;
        iRst:
                    in std_logic;
        iAuroraRefN: in std logic;
        iAuroraRefP: in std logic;
        oAuroraTxN: out std_logic_vector(0 to gLanes - 1);
        oAuroraTxP: out std_logic_vector(0 to gLanes - 1)
    ):
end entity:
architecture behavioral of zynq_wrapper is
    signal wTraceClkSdr: std_logic;
    signal wTraceDataSdr: std_logic_vector(31 downto 0);
    signal wFifoData: std_logic_vector(8*gBytes - 1 downto 0);
    signal wFifoValid:
                        std_logic;
    signal wFifoLast: std_logic;
signal wFifoKeep: std_logic_vector(gBytes - 1 downto 0);
    signal wRstN:
                         std_logic;
begin
    yDesign: entity work.mpsoc port map (
        TRACE CLK SDR => wTraceClkSdr,
        TRACE DATA
                        => wTraceDataSdr
    );
    yAdapter: entity work.serial_trace_adapter_axis generic map (
        gBytes
                          => gBytes
    ) port map (
        iClk
                         => wTraceClkSdr,
        iRst
                         => iRst,
        iData
                         => wTraceDataSdr,
        oData
                         => wFifoData,
        oValid
                         => wFifoValid,
        oLast
                         => wFifoLast,
       oKeep
                         => wFifoKeep
    );
    -- continued on next page
```

```
-- continued from previous page
      yAurora: entity work.aurora and fifo port map (
             loopback
                                          => "000",
             power_down
gt_refclk1_n
gt_refclk1_p
                                        => '0',
                                         => iAuroraRefN,
                                         => iAuroraRefP,
=> oAuroraTxN,
             txn
            txn => oAuroraTxN,
txp => oAuroraTxN,
m_axis_aresetn => wRstN,
s_axis_aresetn => wRstN,
s_axis_aclk => wTraceClkSdr,
s_axis_tdata => wFifoData,
s_axis_tvalid => wFifoValid,
s_axis_tlast => wFifoLast,
s_axis_tkeep => wFifoKeep,
at react => iPat
             qt reset
                                          => iRst,
             init_clk_in => iInitClk,
             tx system reset => wRstN,
      );
      wRstN <= not iRst;
end architecture;
```

- 9. Finish your Vivado design and export the project to the SDK.
- 10. Generate or regenerate your FSBL (first-stage boot loader).
- 11. Do one of the following:
 - Either program the resulting FSBL to the boot device,
 - Or perform a debugger-based boot (see "Performing a Debugger-Based Boot on the Zynq UltraScale+", page 39).
- 12. Use the following commands in your PRACTICE start-up script (*.cmm) to configure the trace:

```
TPIU.PortSize 32

TPIU.PortSize 1Lane ; or 2Lane

TPIU.PortMode Continuous

TPIU.PortClock 6250Mbps : Vivado uses the term 'lane rate' here

TPIU.RefClock 1/20 ; to enable the PortClock/20

; reference clock
```

If your design requires the reference clock from the debugger, you must make sure that the FPGA is reset or programmed after the debugger has been connected and configured.

Exporting the UltraScale+ Trace Interface via HSSTP (10 Gbps)

Use the following step-by-step procedure to produce an FPGA design that supports tracing the Arm cores via HSSTP at a lane rate of 10 Gbps. Because the Xilinx Aurora 8b10b core does not support this lane rate, this step-by-step procedure uses a custom Aurora encoder coupled with the UltraScale FPGAs transceivers wizard.

Note that the bandwidth of the trace infrastructure of the PS is limited to 1000 MB/s. Therefore, using a higher lane rate than 10 Gbps or multiple lanes does not provide any benefit.

We recommend that users first verify their hardware using the previous section, "Exporting the UltraScale+ Trace Interface via HSSTP (up to 6.25 Gbps)", page 26, before following these steps.

- 1. Perform steps 1 through 6 as described in "Exporting the UltraScale+ Trace Interface via FPGA Fabric/PL", page 23. Use a TPIU clock frequency of 250 MHz.
- 2. From the IP catalog, choose the UltraScale FPGAs Transceivers Wizard.

🖵 Customize IP								
UltraScale FPGAs Transceivers Wiza	ard (1.6)							
🞁 Documentation 🐯 Presets 🛅 IP Location 🗔 Sw	🍘 Documentation 🍓 Presets 늡 IP Location 🧔 Switch to Defaults							
IP Symbol Physical Resources	Component Name gt_llane Basic Physical Resources Optional Features Structural Options							

- Select the preset GTH-Aurora_8B10B.
- Set the line rate to 10 GB/s.
- Choose a suitable reference clock. The debugger can provide a 500 MHz reference clock.

- Choose 8B/10B as the encoding.

Since the wizard does not have a simplex option, we also need to configure the receiver, even though it is not used. It is simplest to use the same settings as for the transmitter.

nponent Name gt_1lane					6
Basic Physical Reso	ources Optional Features	Structural Options			
System					
Transceiver configur	ation preset GTH-Aurora_8B	10B*		•	
Transceiver type	GTH			•	
Transmitter		R	leceiver		
Line rate (Gb/s)	10	8	Line rate (Gb/s)	10	8
PLL type	QPLL0	-	PLL type	QPLL0	-
QPLL Fractional-	N options	*	QPLL Fractional-	N options	8
Actual Reference Clock (MHz)	500	•	Actual Reference Clock (MHz)	500	
Encoding	8B/10B	*	Decoding	8B/10B	•
User data width	32	*	User data width	32	•
Internal data width	40	~	Internal data width	40	•
Buffer	Enable (1)	*	Buffer	Enable (1)	*
TXOUTCLK source	TXPROGDIVCLK	*	RXOUTCLK source	RXOUTCLKPMA	•
Advanced		8	Advanced		۲
					OK Cancel

3. On the **Physical Resources** tab, select the channel and reference clock to be used. You also need to provide a free-running clock for initialization and specify its frequency.

🖵 Customize IP						x			
UltraScale FPGAs Transceivers Wizard	UltraScale FPGAs Transceivers Wizard (1.6)								
🍘 Documentation 🊳 Presets 🚞 IP Location 🧔 Switch	h to Defaults								
IP Symbol Physical Resources	Component Name gt_1lane					8			
Bight-click on channels to enable and edit	Basic Physical Resources	Optional Feature	Structural Options						
	Free-running and DRP clock freque	ncy (MHz) 74.25	[3.125 - 250]			Â			
	TX Master channel X0Y12 👻		RX Master channel X0Y12 👻	Disable All Channels					
X0Y15	Channel table (1/24 channel	s used)			۲				
	Search: Q-								
3_dk0 - OPU0	Channel	Enable	TX REFCLK source (QPLL0)	RX REFCLK source (QPLL0)	RXRECCLKOUT buffer				
	- Left column (1/12 channels us	ed)							
	Quad X0Y3 in SLR 0 (1/4 d	hannels used)							
QPIL1	GTHE4_CHANNEL_X011	4							
3 dk1	-GTHE4_CHANNEL_X0Y1	3							
< III >	GTHE4_CHANNEL_X0Y1	2 🗸	MGTREFCLK1	* MGTREFCLK1	No	Ŧ			
					OK Can	icel			

4. On the **Structural Options** tab, set all auxiliary components to be included in the core.

🖵 Customize IP		×
UltraScale FPGAs Transceivers Wiza	rd (1.6)	4
🍘 Documentation 🍓 Presets 🚞 IP Location 🧔 Swit	ch to Defaults	
IP Symbol Physical Resources	Component Name gt_llane	
Show disabled ports	Basic Physical Resources Optional Features Structural Options	
	Simplify transceiver usage by organizing resources and helper blocks	
	Include transceiver COMMON in the Core 💌	-
	Include simple transmitter user clocking network in the Core 🔻	
	Include simple receiver user clocking network in the Core 💌	
	Include reset controller in the Core	
	Include user data width sizing in the Core 🔻	
	Include transmitter buffer bypass controller in the Core 💌	
	Include receiver elastic buffer bypass controller in the Core 💌	
	Include In-System IBERT core No (do not include)	-
		OK Cancel
		Cancer

- 5. Locate the directory ~~/demo/arm/hardware/zynq_ultrascale/hdl/ from your TRACE32 installation and add the following files to the project:
 - serial_trace_adapter_gt.vhd
 - tpiu_to_stream.vhd
 - fifo_inferred.vhd
 - synchronizer.vhd
 - util_pkg.vhd
 - aurora_encoder.vhd
 - aurora_idle_generator.vhd

- lfsr_65_1034.vhd
- 6. Modify the generated wrapper for the Zynq UltaScale+ MPSoC to include the Processing System block design, the transceiver IP generated in steps 2 to 4 and an adapter contained in the HDL files imported in step 5:

```
library IEEE;
use IEEE.std logic 1164.all;
library UNISIM;
use UNISIM.Vcomponents.all;
entity zynq_wrapper is
   port (
       iInitClk: in std_logic;
                   in std logic;
       iRst:
       iAuroraRefN: in std_logic;
       iAuroraRefP: in std_logic;
       oAuroraTxN: out std logic vector(0 to 0);
       oAuroraTxP: out std_logic_vector(0 to 0)
   );
end entity;
architecture behavioral of zyng_wrapper is
    signal wTraceClkSdr: std_logic;
    signal wTraceDataSdr: std_logic_vector(31 downto 0);
    signal wAuroraRef: std_logic;
    signal wUserClk: std_logic;
    signal wTxData: std_logic_vector(31 downto 0);
    signal wTxDataK: std_logic_vector( 3 downto 0);
begin
   yDesign: entity work.mpsoc port map (
       TRACE CLK SDR => wTraceClkSdr,
       TRACE_DATA => wTraceDataSdr
   );
   yAdapter: entity work.serial_trace_adapter_gt port map (
       iTraceClk => wTraceClkSdr,
       iTraceRst
                     => iRst,
       iUserClk
                     => wUserClk,
       iUserRst
                     => iRst,
       iData
                     => wTraceDataSdr,
                     => wTxData,
       oData
       oDataK
                     => wTxDataK
    );
    -- continued on next page
```

```
-- continued from previous page
   yGt: entity work.gt 11ane port map (
       gtwiz reset clk freerun in(0)
                                            => iInitClk,
       gtwiz reset all in(0)
                                             => iRst.
                                            => wAuroraRef,
       gtrefclk00 in(0)
       gtwiz userclk tx reset in
                                            => (others => '0'),
       gtwiz_userclk_tx_usrclk2_out(0)
                                            => wUserClk,
        gtwiz reset tx pll and datapath in => (others => '0'),
       gtwiz reset tx datapath in
                                            => (others => '0'),
       gtwiz userdata tx in
                                             => wTxData,
        tx8b10ben in
                                             => (others => '1'),
        txctrl0 in
                                             => (others => '0'),
        txctrl1 in
                                            => (others => '0'),
        txctrl2 in(7 downto 4)
                                            => (others => '0'),
        txctrl2 in(3 downto 0)
                                           => wTxDataK,
       athtxn out
                                             => oAuroraHpc1TxN,
       gthtxp out
                                             => oAuroraHpc1TxP,
       gtwiz userclk rx reset in
                                           => (others => '0'),
       gtwiz_reset_rx_pll_and_datapath_in => (others => '0'),
       gtwiz_reset_rx_datapath_in
                                            => (others => '0'),
       gthrxn in
                                             => (others => '0'),
       gthrxp in
                                             => (others => '0'),
       rx8b10ben in
                                             => (others => '0'),
       rxbufreset in
                                             => (others => '0'),
       rxcommadeten in
                                            => (others => '0'),
                                            => (others => '0'),
        rxmcommaalignen in
       rxpcommaalignen in
                                           => (others => '0')
    );
   yBufAurora: IBUFDS_GTE4 port map (
            => wAuroraRef,
       0
                => iAuroraRefP,
        Ι
       TB
                => iAuroraRefN,
       CEB
                => '0'
    );
end architecture;
```

- 7. Finish your Vivado design and export the project to the SDK.
- 8. Generate or regenerate your FSBL (first-stage boot loader).
- 9. Do one of the following:
 - Either program the resulting FSBL to the boot device,
 - Or perform a debugger-based boot (see "Performing a Debugger-Based Boot on the Zynq

UltraScale+", page 39).

10. Use the following **TPIU** commands in your PRACTICE start-up script (*.cmm) to configure the trace:

TPIU.PortSize 32 TPIU.PortSize 1Lane TPIU.PortMode Continuous TPIU.PortClock 10000Mbps TPIU.RefClock 1/20 ; if you require the 500 MHz reference clock

If your design requires the reference clock from the debugger, you must make sure that the FPGA is reset or programmed after the debugger has been connected and configured.

Off-chip trace via PCIe differs from the other trace methods because it requires assistance from the target code. The debugger acts as a slave device to the PCIe root complex in the PS. The operating system running on the target must enumerate the debugger and assign an address to it.

This means that the debugger must already be ready and configured while the target is booting. Trace capture is only possible after the device has booted and the debugger must not be disconnected from the target while it is running.

- 1. Create a new Vivado project with an instance of the Zynq processing system.
- 2. Enter the configuration of the Zynq processing system.
- 3. Enable the PCIe root port and select the correct number of lanes. Using four lanes is possible, but will not increase the bandwidth available for trace.

IF Re-customize IP											
ZYNQ UltraScale+ MPSoc (2.0)											
🛍 Documentation 🛅 IP Location											
Page Navigator											
Switch To Advanced Mo	+	🔶 🔺 MIO Voltage Standard									
PS UltraScale + Block Design		Bank0 [MIO 0:25] Bank1 [MIO LVCMOS33 + LVCMOS33	26:51] Bank2 [MIO 5 + LVCMOS33	2:77]							
I/O Configuration	\\$	Peripheral	I/O	Signal	I/O Type	Drive Strength(mA)	Speed	Pull Type	Direction		
Clock Configuration	0										
DDR Configuration		High Speed GEM H-USB									
PS-PL Configuration		PCIe Rootport Mode Reset	MIO 31	-	a de avitat			-			
		····PCIE	MI031	reset_n	schmitt *	112 *	I SIOW *	j puliup *	out		
		PCIe Lane0	GT Lane0								
		Reference Clocks									
								ок	Cancel		

- 4. Finish your Vivado design and export the project to the SDK.
- 5. Generate or regenerate your FSBL (first-stage boot loader).
- 6. When setting up the target's operating system, make sure that the appropriate drivers are enabled for PCIe support.
- 7. Verify that PCIe is operational using an expansion card such as a PCIe-to-USB converter.
- 8. Do one of the following:
 - Either program the resulting FSBL to the boot device,
 - Or perform a debugger-based boot (see "Performing a Debugger-Based Boot on the Zynq UltraScale+", page 39).

For the ZCU102 evaluation board, Lauterbach provides an example Vivado project for download at https://www.lauterbach.com/support/static/armdemos/20170313ZCU102_trace_example.xpr.zip.

The design contains five top-level entities for different trace setups:

- top_parallel_8bit
- top_parallel_16bit
- top_hsstp_1lane_6250mbps
- top_hsstp_2lane_6250mbps
- top_hsstp_1lane_10000mbps

Use "Set as Top" on one of these entities in the project manager before generating a bitstream.

Parallel trace is exported on the P6 connector on the ZCU102, while serial trace requires an adapter to be plugged into the J4 port (HPC1). Contact Lauterbach to obtain this adapter.

Performing a Debugger-Based Boot on the Zynq UltraScale+

This section focuses on the **JTAG-BOOT** mode of the ZynqUltraScale+. In contrast to all other boot modes, this mode is only intended for development. The basic idea is that the CPUs will wait in an endless loop after executing the boot ROM, allowing the JTAG probe to perform all further initialization.

To perform a debugger-based boot:

- 1. Set the boot mode to **JTAG-BOOT** using the MIO lines.
- 2. Reset the SoC, for example by asserting the RESET line. Not all boards have a RESET line connected to the SoC, thus a power cycle or similar might be required.
- 3. Execute the boot ROM.
- 4. Load the FSBL boot code using the debugger.
- 5. Execute the FSBL boot code.
- 6. Optionally load the FPGA fabric using the debugger.

You are now ready to load the next stage boot loader, OS, etc., and to use the optional off-chip trace.

Example files following the above sequence are included in the TRACE32 installation directory under ~~/demo/arm/hardware/zynq_ultrascale.