Debugging Embedded Cores in Xilinx FPGAs [Zynq]

Release 09.2024

Debugging Embedded Cores in Xilinx FPGAs [Zynq]

Debugging Embedded Cores in Xilinx FPGAs [Zynq]

Version 05-Oct-2024

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.

Physical Connection Requirements

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.

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

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

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**

Mictor-38 Pinout

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
GND	5	6	TRACECLK
DBGRQ	7	8	DBGACK
RESET-	9	10	EXTRIG
TDO|-|SWO	11	12	VREF-TRACE
RTCK	13	14	VREF-DEBUG
TCK|TCKC|SWCLK	15	16	TRACEDATA[7]
TMS|TMSC|SWDIO	17	18	TRACEDATA[6]
TDI	19	20	TRACEDATA[5]
TRST-	21	22	TRACEDATA[4]
TRACEDATA[15]	23	24	TRACEDATA[3]
TRACEDATA[14]	25	26	TRACEDATA[2]
TRACEDATA[13]	27	28	TRACEDATA[1]
TRACEDATA[12]	29	30	GND
TRACEDATA[11]	31	32	GND
TRACEDATA[10]	33	34	VCC
TRACEDATA[9]	35	36	TRACECTL
TRACEDATA[8]	37	38	TRACEDATA[0]

MIPI-60 Pinout

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	TMS|TMSC|SWDIO
TCK|TCKC|SWCLK	3	4	TDO|-|SWO
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

Requirements for Serial HSSTP Trace

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.

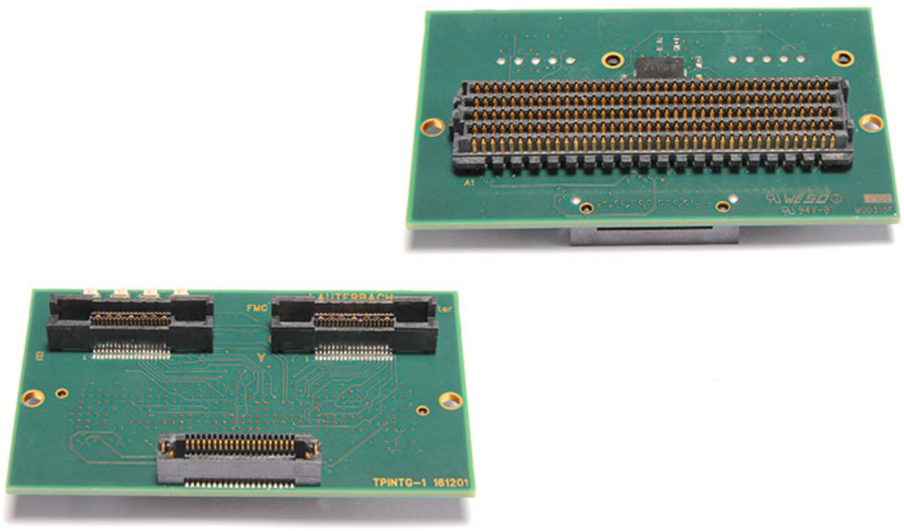
For more information about the target connector, visit **www.lauterbach.com/adetmhsstp.html**

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	Pin	Pin	Signal
TXP[4]	1	2	VREF-DEBUG
TXN[4]	3	4	TCK|TCKC|SWCLK
GND	5	6	GND
TXP[2]	7	8	TMS|TMSC|SWDIO
TXN[2]	9	10	TRST-
GND	11	12	GND
TXP[0]	13	14	TDI
TXN[0]	15	16	TDO|-|SWO
GND	17	18	GND
CLKP	19	20	RESET-
CLKN	21	22	DBGRQ
GND	23	24	GND
TXP[1]	25	26	DBGACK
TXN[1]	27	28	RTCK
GND	29	30	GND
TXP[3]	31	32	TRIGIN
TXN[3]	33	34	TRIGOUT
GND	35	36	RESERVED
TXP[5]	37	38	RESERVED
TXN[5]	39	40	RESERVED

Trace-Adapter for FMC-featured Target Boards

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.



Zynq-7000 Devices

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.

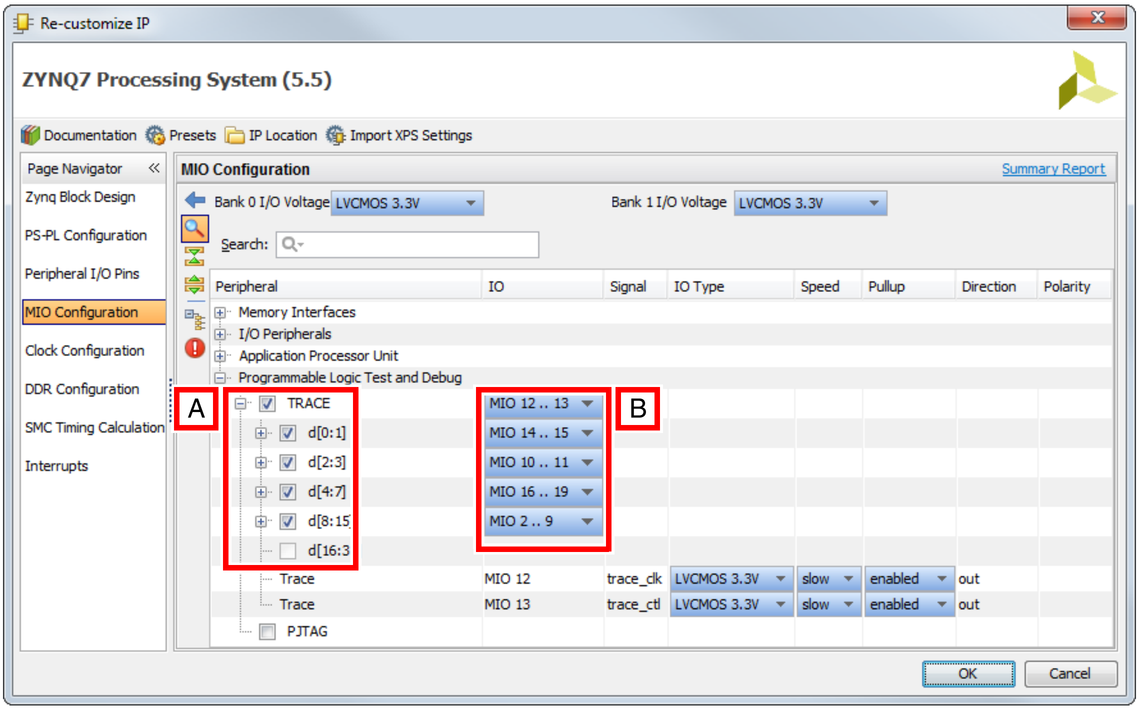
Method 3 is similar to method 2, but instead of dividing the trace clock by two, the width of the data path is reduced internally. This allows using an external 16-bit trace port by internally utilizing the full 32-bit PS-PL TPIU interface. Alternatively, an external 8-bit port can also be created. The 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.



Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

Page Navigator: Zynq Block Design, PS-PL Configuration, Peripheral I/O Pins, MIO Configuration, Clock Configuration, DDR Configuration, SMC Timing Calculation, Interrupts

MIO Configuration — Summary Report

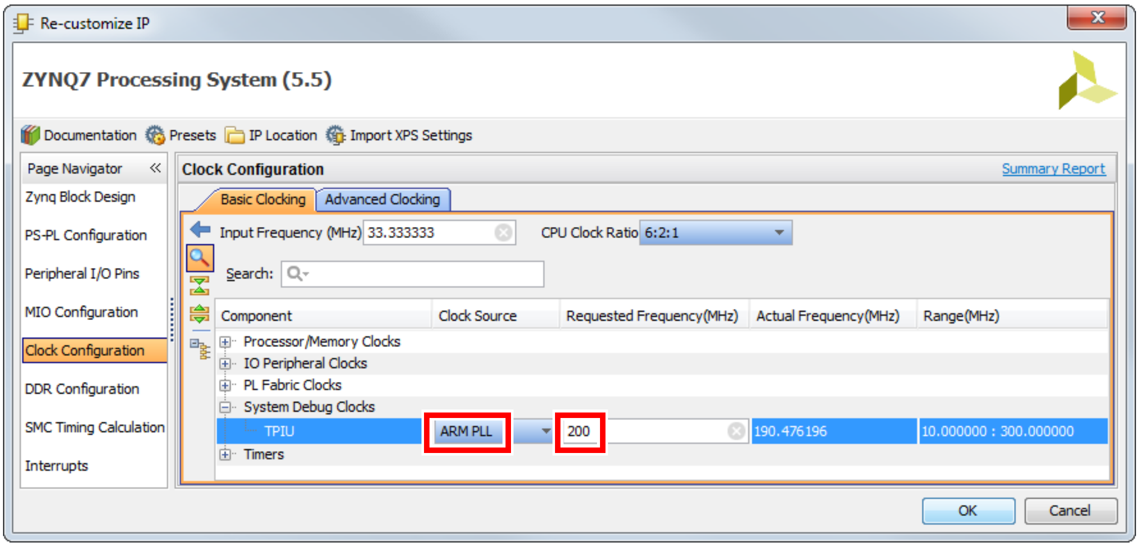
Bank 0 I/O Voltage LVCMOS 3.3V   Bank 1 I/O Voltage LVCMOS 3.3V

Search:

Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction	Polarity
Memory Interfaces							
I/O Peripherals							
Application Processor Unit							
Programmable Logic Test and Debug							
✓ TRACE	MIO 12 .. 13						
✓ d[0:1]	MIO 14 .. 15						
✓ d[2:3]	MIO 10 .. 11						
✓ d[4:7]	MIO 16 .. 19						
✓ d[8:15]	MIO 2 .. 9						
☐ d[16:3							
Trace	MIO 12	trace_clk	LVCMOS 3.3V	slow	enabled	out	
Trace	MIO 13	trace_ctl	LVCMOS 3.3V	slow	enabled	out	
☐ PJTAG							

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 IP Location Import XPS Settings

Page Navigator: Zynq Block Design, PS-PL Configuration, Peripheral I/O Pins, MIO Configuration, Clock Configuration, DDR Configuration, SMC Timing Calculation, Interrupts

Clock Configuration — Summary Report

Basic Clocking | Advanced Clocking

Input Frequency (MHz) 33.333333   CPU Clock Ratio 6:2:1

Search:

Component	Clock Source	Requested Frequency(MHz)	Actual Frequency(MHz)	Range(MHz)
Processor/Memory Clocks				
IO Peripheral Clocks				
PL Fabric Clocks				
System Debug Clocks				
TPIU	ARM PLL	200	190.476196	10.000000 : 300.000000
Timers				

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**).

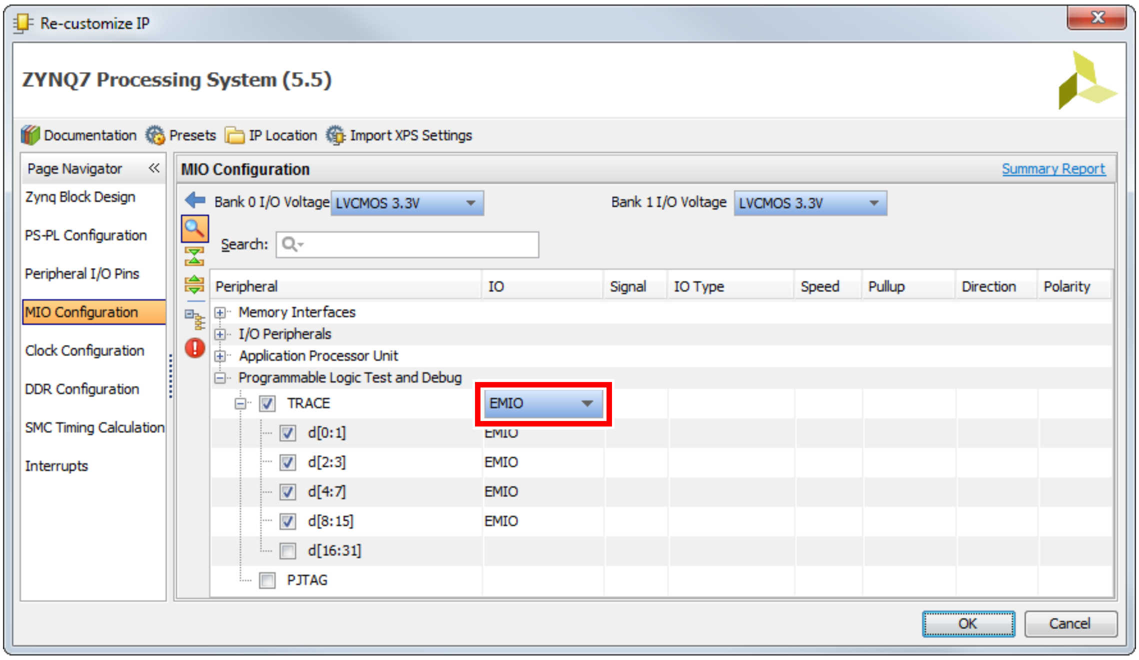
You are now ready to debug and trace your target with TRACE32.

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.

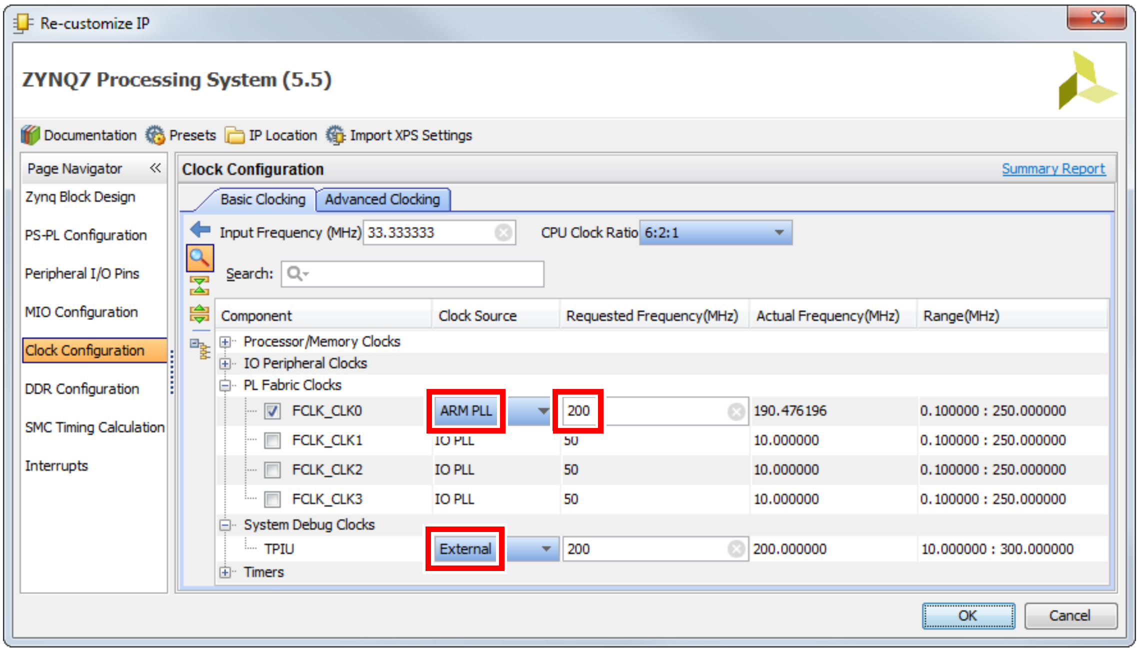


MIO Configuration

Bank 0 I/O Voltage: LVCMOS 3.3V   Bank 1 I/O Voltage: LVCMOS 3.3V

Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction	Polarity
Memory Interfaces							
I/O Peripherals							
Application Processor Unit							
Programmable Logic Test and Debug							
✓ TRACE	EMIO						
✓ d[0:1]	EMIO						
✓ d[2:3]	EMIO						
✓ d[4:7]	EMIO						
✓ d[8:15]	EMIO						
☐ d[16:31]							
☐ PJTAG							

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.

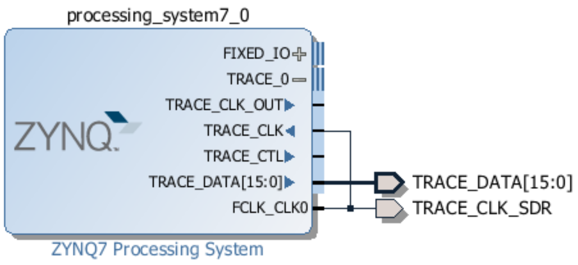


Clock Configuration

Input Frequency (MHz) 33.333333   CPU Clock Ratio 6:2:1

Component	Clock Source	Requested Frequency(MHz)	Actual Frequency(MHz)	Range(MHz)
Processor/Memory Clocks				
IO Peripheral Clocks				
PL Fabric Clocks				
✓ FCLK_CLK0	ARM PLL	200	190.476196	0.100000 : 250.000000
☐ FCLK_CLK1	IO PLL	50	10.000000	0.100000 : 250.000000
☐ 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	200.000000	10.000000 : 300.000000
Timers				

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;
            rTraceData_q <= wTraceData;
        end if;
    end process;

    oTraceData   <= rTraceData_qq;
    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).

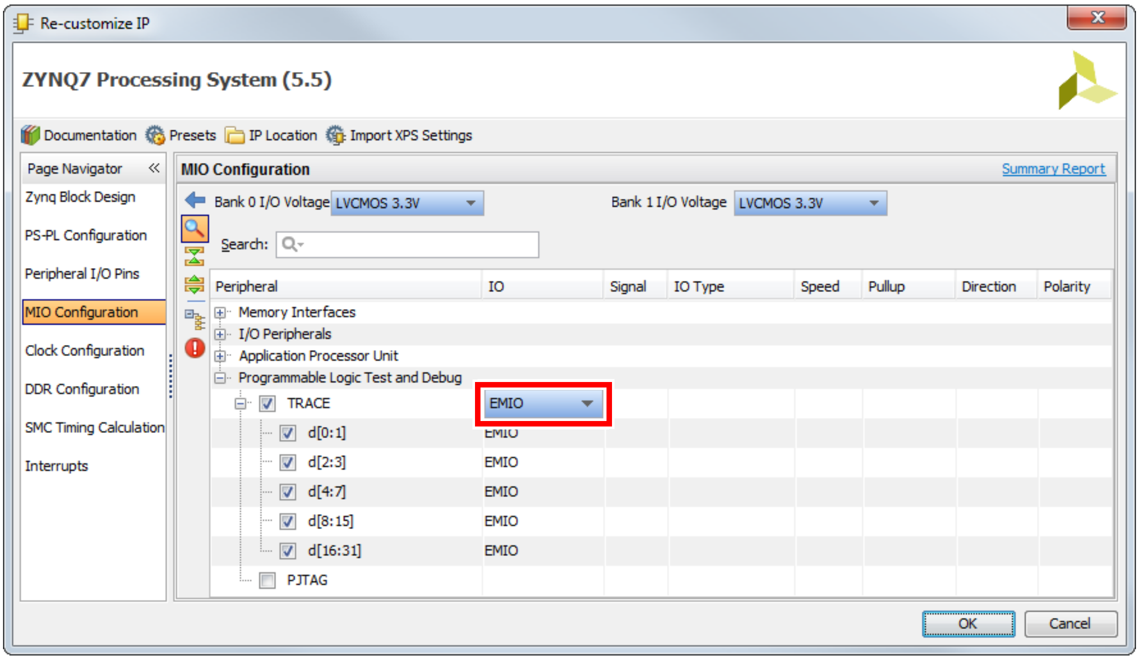
You are now ready to debug and trace your target with TRACE32.

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

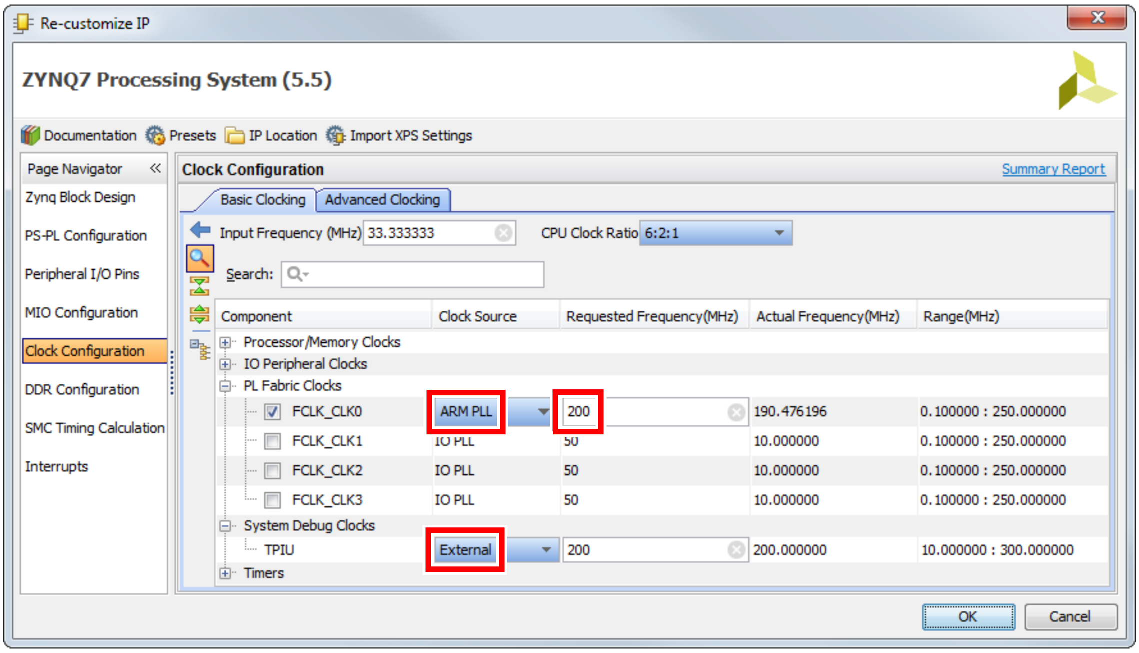
ZYNQ7 Processing System (5.5)

MIO Configuration

Bank 0 I/O Voltage: LVCMOS 3.3V   Bank 1 I/O Voltage: LVCMOS 3.3V

Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction	Polarity
Memory Interfaces							
I/O Peripherals							
Application Processor Unit							
Programmable Logic Test and Debug							
✓ TRACE	EMIO						
✓ d[0:1]	EMIO						
✓ d[2:3]	EMIO						
✓ d[4:7]	EMIO						
✓ d[8:15]	EMIO						
✓ d[16:31]	EMIO						
PJTAG							

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 IP Location Import XPS Settings

Page Navigator
- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

Clock Configuration Summary Report

Basic Clocking | Advanced Clocking

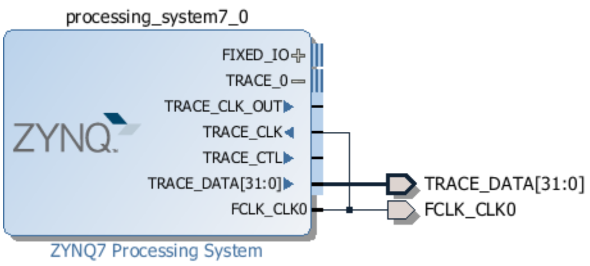
Input Frequency (MHz) 33.333333 CPU Clock Ratio 6:2:1

Search:

Component	Clock Source	Requested Frequency(MHz)	Actual Frequency(MHz)	Range(MHz)
Processor/Memory Clocks				
IO Peripheral Clocks				
PL Fabric Clocks				
☑ FCLK_CLK0	ARM PLL	200	190.476196	0.100000 : 250.000000
☐ FCLK_CLK1	IO PLL	50	10.000000	0.100000 : 250.000000
☐ 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	200.000000	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 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 (
      iClk      => wTraceClkSdr,
      iData     => wTraceData,
      oClk      => oTraceClkDdr,
      oData     => oTraceData
   );
end architecture;
```

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

You are now ready to debug and trace your target with TRACE32.

Performing a Debugger-Based Boot on the Zynq-7000

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

UltraScale+ Devices

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 PowerTrace II / PowerTrace III / PowerTrace PX with AutoFocus preprocessor
2 (16 bit @ 250 MHz DDR)	17 PL I/Os	yes	1000 MB/s	
2 (8 bit @ 250 MHz DDR)	9 PL I/Os	yes	500 MB/s	
3 (2 Lanes @ 6.25 Gbps)	2 GTH	yes	1000 MB/s	PowerTrace II / PowerTrace III / PowerTrace PX with serial preprocessor or PowerTrace Serial
3 (1 Lane @ 6.25 Gbps)	1 GTH	yes	625 MB/s	
3 (1 Lane @ 10Gbps)	1 GTH	yes	1000 MB/s	PowerTrace Serial
4 (PCIe v2 ×2)	2 GTR	no	1000 MB/s	
4 (PCIe v2 ×1)	1 GTR	no	500 MB/s	

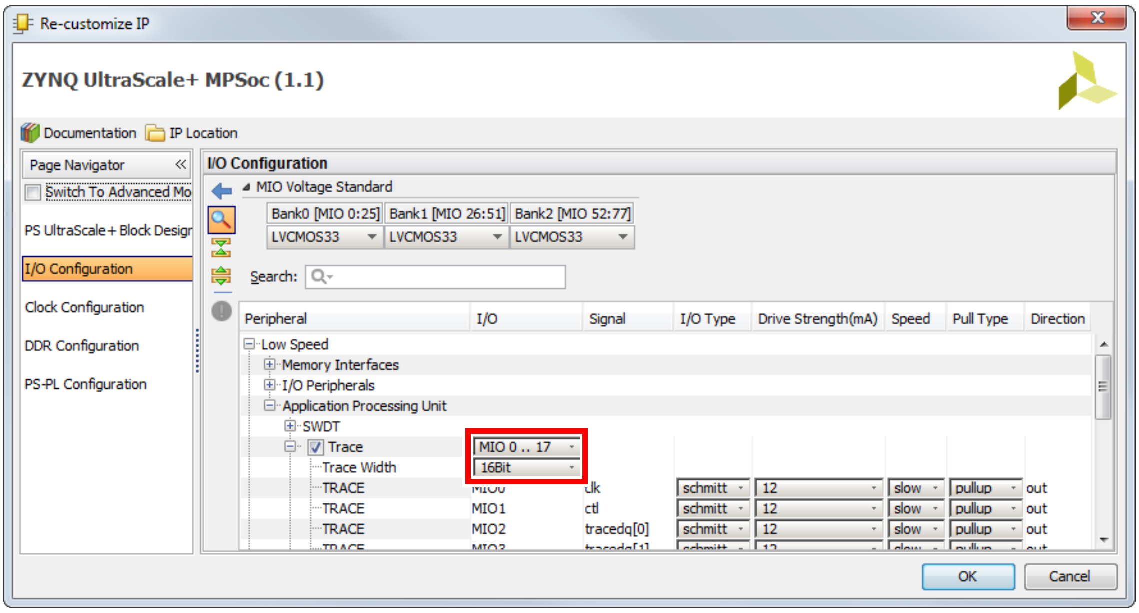
NOTE:

- 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.



I/O Configuration

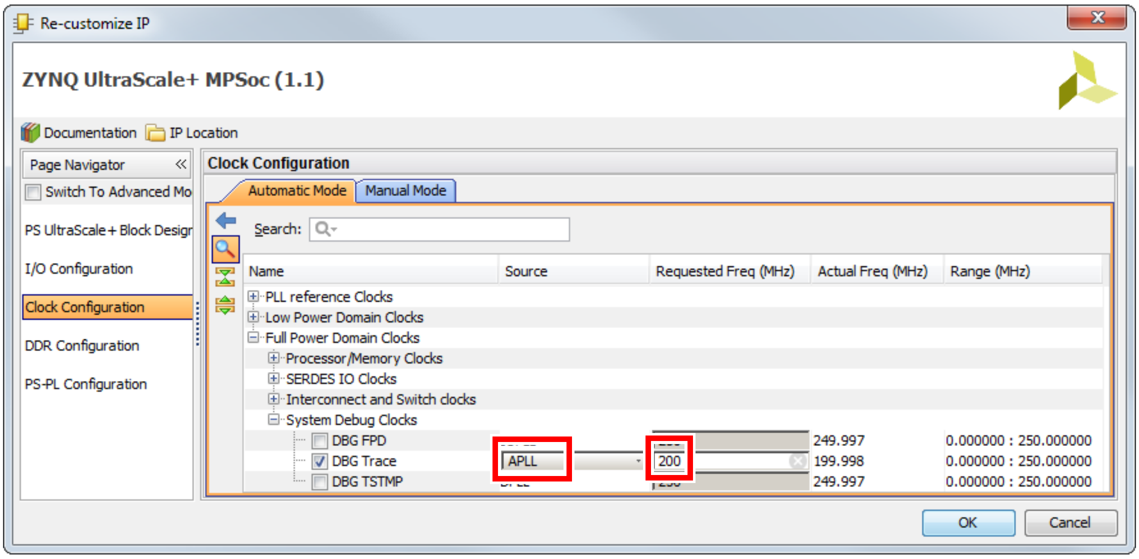
MIO Voltage Standard

Bank0 [MIO 0:25]	Bank1 [MIO 26:51]	Bank2 [MIO 52:77]
LVCMOS33	LVCMOS33	LVCMOS33

Search:

Peripheral	I/O	Signal	I/O Type	Drive Strength(mA)	Speed	Pull Type	Direction
Low Speed							
Memory Interfaces							
I/O Peripherals							
Application Processing Unit							
SWDT							
✓ Trace	MIO 0 .. 17						
Trace Width	16Bit						
TRACE	MIO0	clk	schmitt	12	slow	pullup	out
TRACE	MIO1	ctl	schmitt	12	slow	pullup	out
TRACE	MIO2	tracedq[0]	schmitt	12	slow	pullup	out
TRACE	MIO3	tracedq[1]	schmitt	12	slow	pullup	out

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.



Clock Configuration

Search:

Name	Source	Requested Freq (MHz)	Actual Freq (MHz)	Range (MHz)
PLL reference Clocks				
Low Power Domain Clocks				
Full Power Domain Clocks				
Processor/Memory Clocks				
SERDES IO Clocks				
Interconnect and Switch clocks				
System Debug Clocks				
DBG FPD			249.997	0.000000 : 250.000000
✓ DBG Trace	APLL	200	199.998	0.000000 : 250.000000
DBG TSTMP			249.997	0.000000 : 250.000000

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).

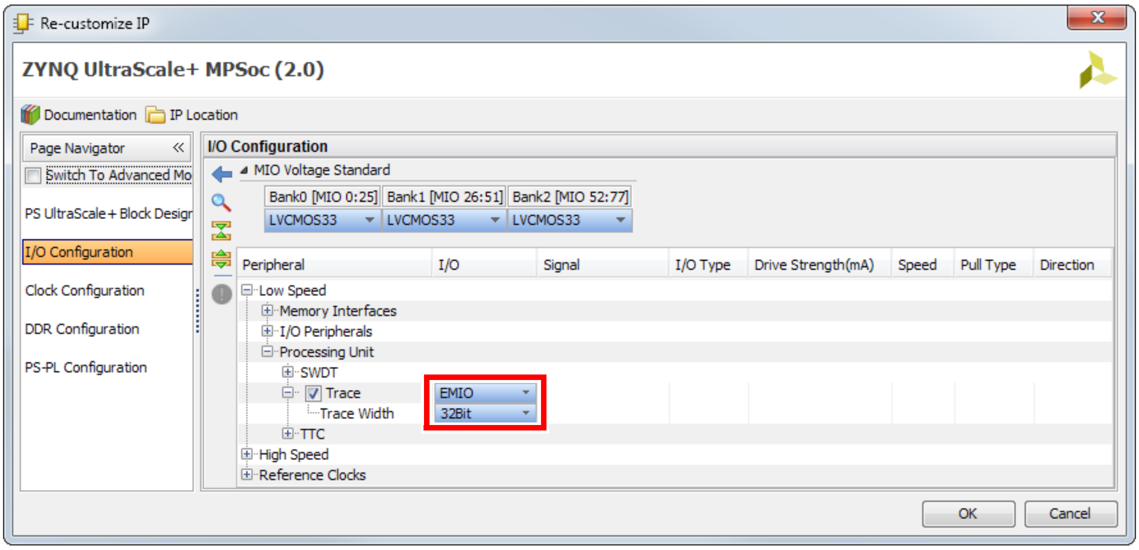
You are now ready to debug and trace your target with TRACE32.

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.



ZYNQ UltraScale+ MPSoc (2.0)

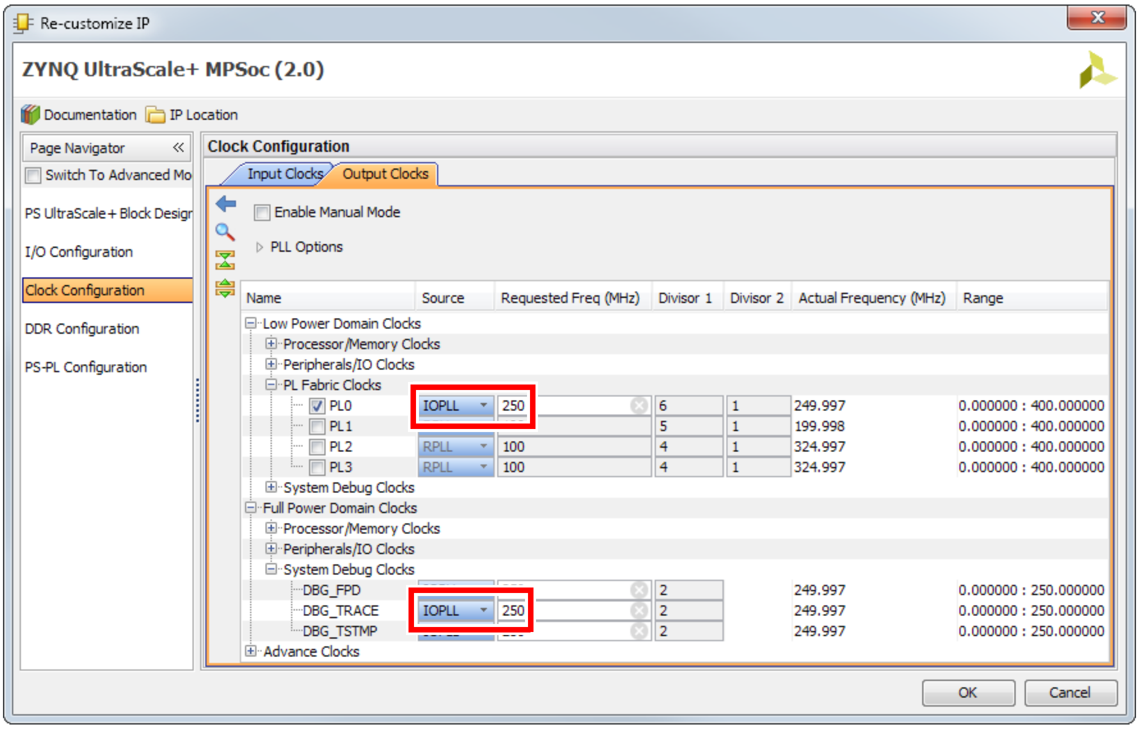
I/O Configuration

MIO Voltage Standard

Bank0 [MIO 0:25]	Bank1 [MIO 26:51]	Bank2 [MIO 52:77]
LVCMOS33	LVCMOS33	LVCMOS33

Peripheral	I/O	Signal	I/O Type	Drive Strength(mA)	Speed	Pull Type	Direction
Low Speed							
Memory Interfaces							
I/O Peripherals							
Processing Unit							
SWDT							
✓ Trace	EMIO						
Trace Width	32Bit						
TTC							
High Speed							
Reference Clocks							

4. 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.



ZYNQ UltraScale+ MPSoc (2.0)

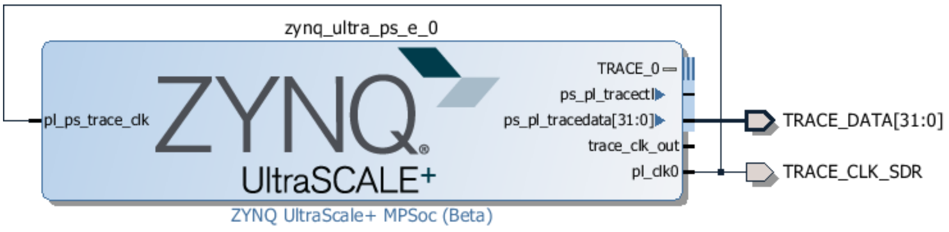
Clock Configuration

Enable Manual Mode

PLL Options

Name	Source	Requested Freq (MHz)	Divisor 1	Divisor 2	Actual Frequency (MHz)	Range
Low Power Domain Clocks						
Processor/Memory Clocks						
Peripherals/IO Clocks						
PL Fabric Clocks						
✓ PL0	IOPLL	250	6	1	249.997	0.000000 : 400.000000
PL1			5	1	199.998	0.000000 : 400.000000
PL2	RPLL	100	4	1	324.997	0.000000 : 400.000000
PL3	RPLL	100	4	1	324.997	0.000000 : 400.000000
System Debug Clocks						
Full Power Domain Clocks						
Processor/Memory Clocks						
Peripherals/IO Clocks						
System Debug Clocks						
DBG_FPD			2		249.997	0.000000 : 250.000000
DBG_TRACE	IOPLL	250	2		249.997	0.000000 : 250.000000
DBG_TSTMP			2		249.997	0.000000 : 250.000000
Advance Clocks						

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.

7. 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 zynq_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,
        gBitsOut      => 16
    ) 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 (*.cmd) 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
```

You are now ready to debug and trace your target with TRACE32.

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.

1. 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:

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

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

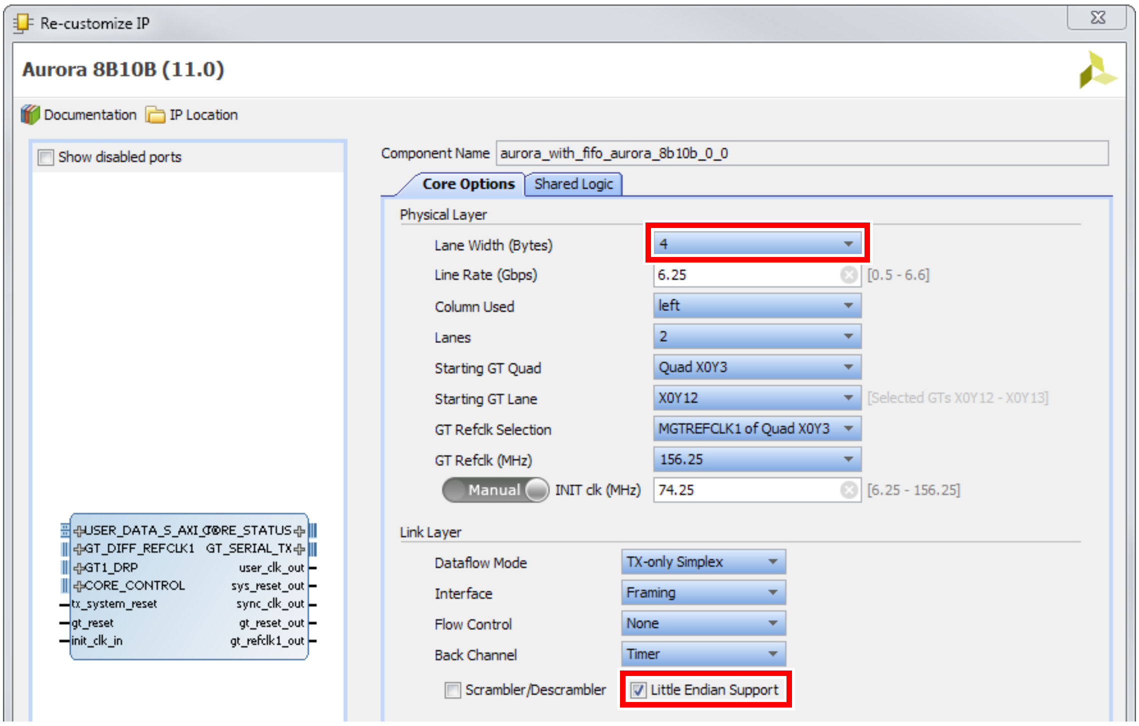
2. Create a second block design to hold the Xilinx Aurora encoder and a FIFO. In this example, it is called **aurora_and_fifo**.

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

Aurora 8B10B (11.0)

Documentation IP Location

Show disabled ports

Component Name aurora_with_fifo_aurora_8b10b_0_0

Core Options | Shared Logic

Physical Layer

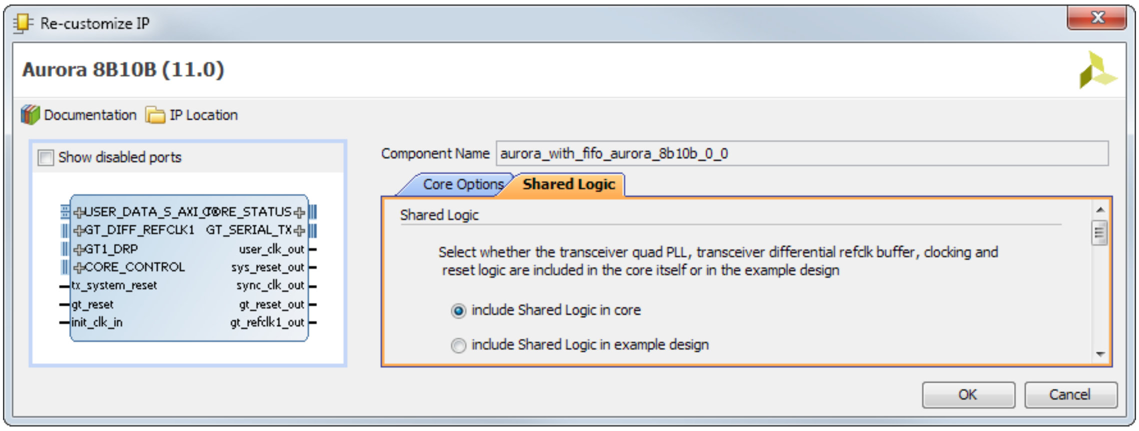
Lane Width (Bytes)	4	
Line Rate (Gbps)	6.25	[0.5 - 6.6]
Column Used	left	
Lanes	2	
Starting GT Quad	Quad X0Y3	
Starting GT Lane	X0Y12	[Selected GTs X0Y12 - X0Y13]
GT Refclk Selection	MGTREFCLK1 of Quad X0Y3	
GT Refclk (MHz)	156.25	
Manual INIT clk (MHz)	74.25	[6.25 - 156.25]

Link Layer

Dataflow Mode	TX-only Simplex
Interface	Framing
Flow Control	None
Back Channel	Timer

[ ] Scrambler/Descrambler [✓ Little Endian Support]

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

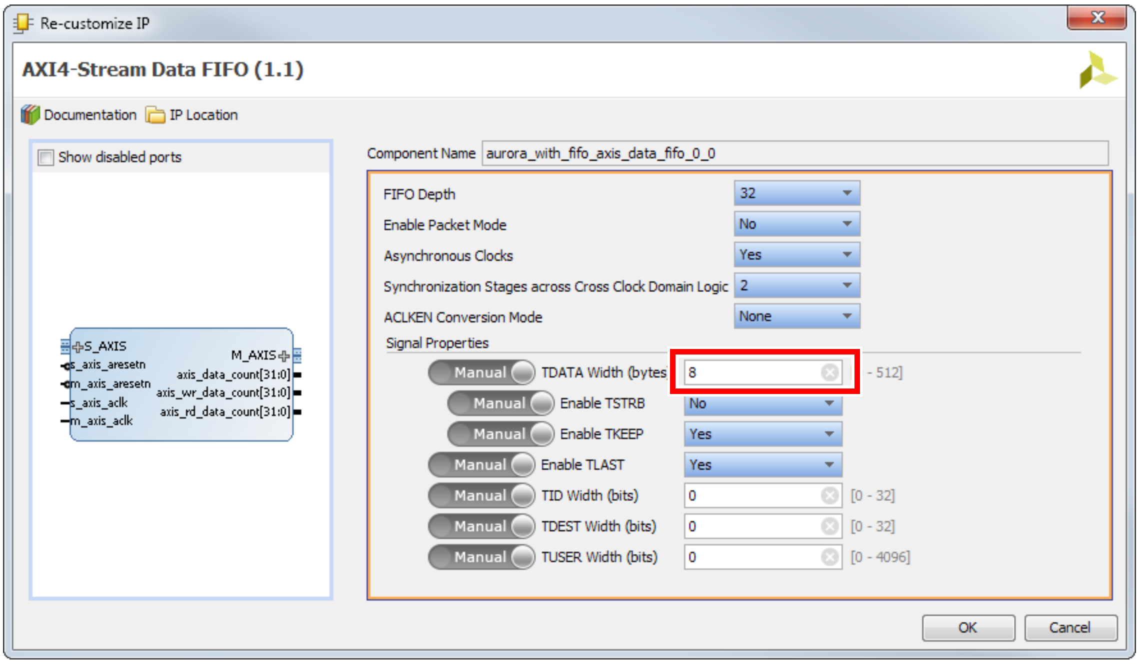
Shared Logic

Select whether the transceiver quad PLL, transceiver differential refclk buffer, clocking and reset logic are included in the core itself or in the example design

[● include Shared Logic in core]

○ include Shared Logic in example design

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.



Re-customize IP

AXI4-Stream Data FIFO (1.1)

Documentation IP Location

Show disabled ports

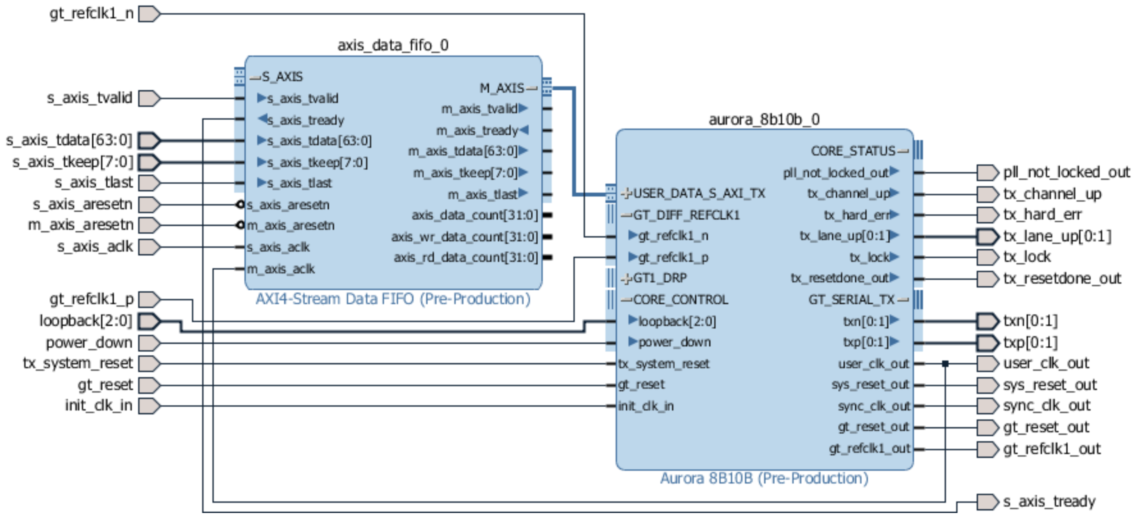
Component Name: aurora_with_fifo_axis_data_fifo_0_0

FIFO Depth	32
Enable Packet Mode	No
Asynchronous Clocks	Yes
Synchronization Stages across Cross Clock Domain Logic	2
ACLKEN Conversion Mode	None

Signal Properties

Manual	TDATA Width (bytes)	8	- 512]
Manual	Enable TSTRB	No	
Manual	Enable TKEEP	Yes	
Manual	Enable TLAST	Yes	
Manual	TID Width (bits)	0	[0 - 32]
Manual	TDEST Width (bits)	0	[0 - 32]
Manual	TUSER Width (bits)	0	[0 - 4096]

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
        gBytes:             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          => '0',
        gt_refclk1_n        => iAuroraRefN,
        gt_refclk1_p        => iAuroraRefP,
        txn                 => oAuroraTxN,
        txp                 => oAuroraTxP,
        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,
        gt_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.

You are now ready to debug and trace your target with TRACE32.

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

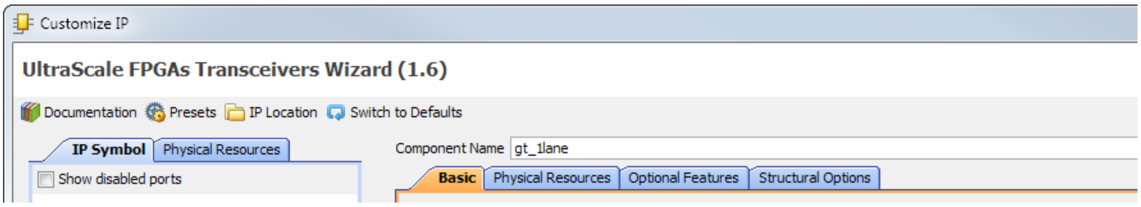
Use the following step-by-step procedure to produce an FPGA design that supports tracing the Arm cores via HSSTP at a lane rate of 10 Gbps. Because the Xilinx Aurora 8b10b core does not support this lane rate, this step-by-step procedure uses a custom Aurora encoder coupled with the UltraScale FPGAs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

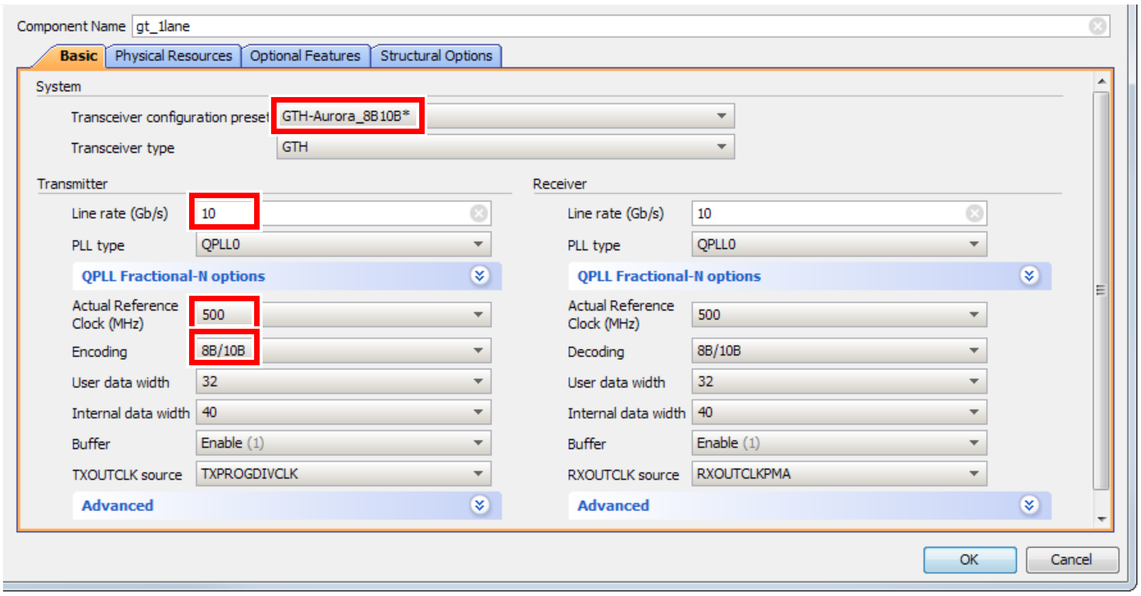

Customize IP
UltraScale FPGAs Transceivers Wizard (1.6)

Component Name gt_1lane

- 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.



Component Name: gt_1lane

Basic | Physical Resources | Optional Features | Structural Options

System

Transceiver configuration preset	GTH-Aurora_8B10B*
Transceiver type	GTH

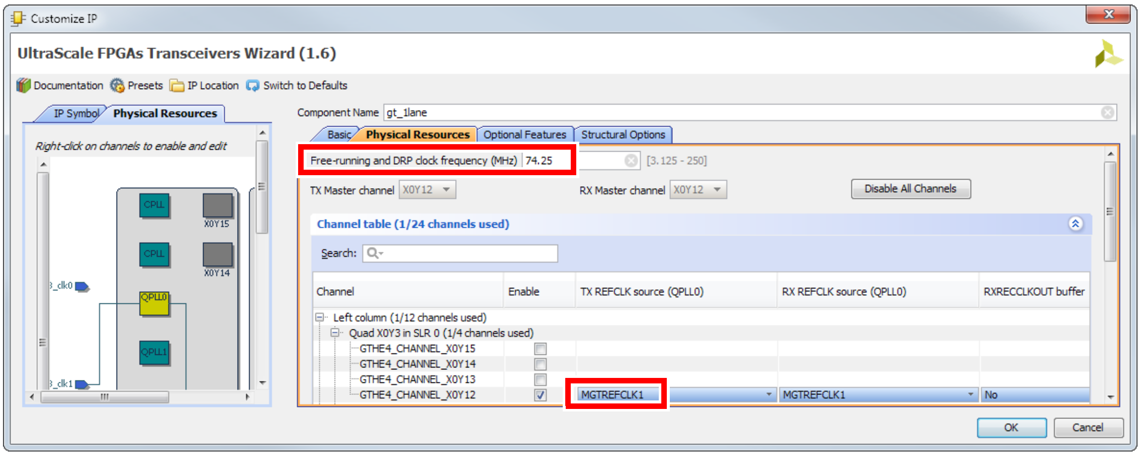
Transmitter

Line rate (Gb/s)	10
PLL type	QPLL0
QPLL Fractional-N options	
Actual Reference Clock (MHz)	500
Encoding	8B/10B
User data width	32
Internal data width	40
Buffer	Enable (1)
TXOUTCLK source	TXPROGDIVCLK
Advanced	

Receiver

Line rate (Gb/s)	10
PLL type	QPLL0
QPLL Fractional-N options	
Actual Reference Clock (MHz)	500
Decoding	8B/10B
User data width	32
Internal data width	40
Buffer	Enable (1)
RXOUTCLK source	RXOUTCLKPMA
Advanced	

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

UltraScale FPGAs Transceivers Wizard (1.6)

Component Name: gt_1lane

Basic | Physical Resources | Optional Features | Structural Options

Free-running and DRP clock frequency (MHz): 74.25 [3.125 - 250]

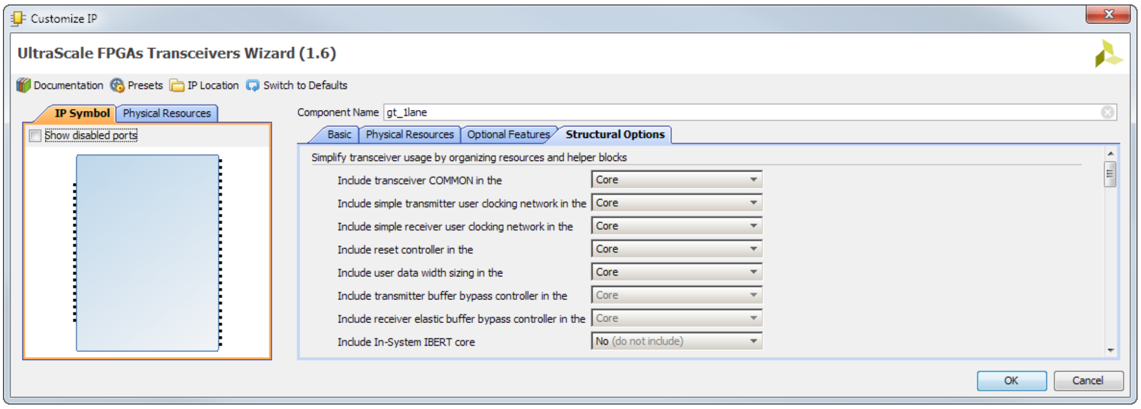
TX Master channel: X0Y12   RX Master channel: X0Y12   Disable All Channels

Channel table (1/24 channels used)

Search:

Channel	Enable	TX REFCLK source (QPLL0)	RX REFCLK source (QPLL0)	RXRECCLKOUT buffer
Left column (1/12 channels used)				
Quad X0Y3 in SLR 0 (1/4 channels used)				
GTHE4_CHANNEL_X0Y15	☐			
GTHE4_CHANNEL_X0Y14	☐			
GTHE4_CHANNEL_X0Y13	☐			
GTHE4_CHANNEL_X0Y12	✓	MGTREFCLK1	MGTREFCLK1	No

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



Customize IP

UltraScale FPGAs Transceivers Wizard (1.6)

Component Name: gt_1lane

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)

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;
        iRst:        in  std_logic;
        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 zynq_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,
        oData           => wTxData,
        oDataK          => wTxDataK
    );
    -- continued on next page
    -- continued from previous page
    yGt: entity work.gt_1lane port map (
        gtwiz_reset_clk_freerun_in(0)         => iInitClk,
        gtwiz_reset_all_in(0)                 => iRst,
        gtrefclk00_in(0)                      => wAuroraRef,

        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,
        gthtxn_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'),
        rxmcommaalignen_in                    => (others => '0'),
        rxpcommaalignen_in                    => (others => '0')
    );

    yBufAurora: IBUFDS_GTE4 port map (
        O          => wAuroraRef,
        I          => iAuroraRefP,
        IB         => 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.

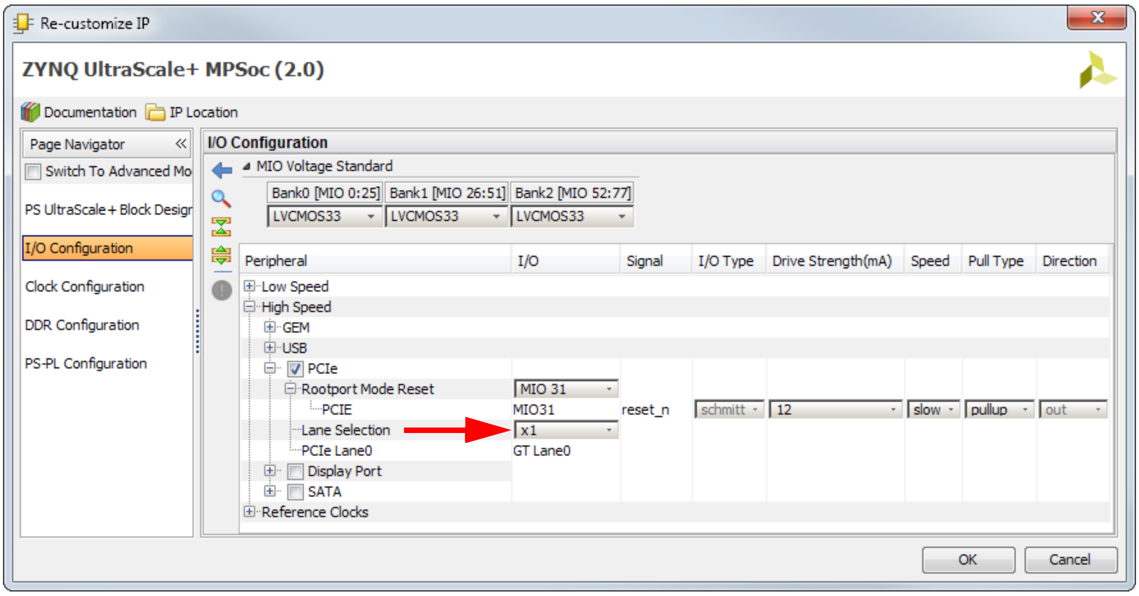
You are now ready to debug and trace your target with TRACE32.

Exporting the UltraScale+ Trace Interface via PCIe

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.



I/O Configuration

MIO Voltage Standard

Bank0 [MIO 0:25]	Bank1 [MIO 26:51]	Bank2 [MIO 52:77]
LVCMOS33	LVCMOS33	LVCMOS33

Peripheral	I/O	Signal	I/O Type	Drive Strength(mA)	Speed	Pull Type	Direction
Low Speed							
High Speed							
GEM							
USB							
✓ PCIe							
Rootport Mode Reset	MIO 31						
PCIE	MIO31	reset_n	schmitt	12	slow	pullup	out
Lane Selection	x1						
PCIe Lane0	GT Lane0						
Display Port							
SATA							
Reference Clocks							

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).

You are now ready to debug and trace your target with TRACE32.

Using the Example Design for the ZCU102

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**.