

Pathwave FPGA

M8132A FSP Documentation

Notice

© Keysight Technologies, Inc. 2021

1400 Fountaingrove Pkwy., Santa Rosa, CA 95403-1738, United States

All rights reserved.

No part of this documentation may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Keysight Technologies, Inc. as governed by United States and international copyright laws.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause.

Use, duplication or disclosure of Software is subject to Keysight Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Table of Contents

Installation Requirements	7
Overview	8
Scope.....	9
Available FPGA Features and Resources	10
Available Configurations.....	11
Port Mapping	13
FPGA Configurable Region Interfaces	14
ODI - Optical Data Interface.....	14
ODI Resets	18
ODI Startup reset.....	19
ODI Receive Status.....	19
ODI Per Channel Flow Control.....	20
ODI Clock.....	20
10Gbe Ethernet Interface	21
10Gbe Ethernet Resets.....	23
10GbE Ethernet Receive Status	24
DPU2DPU	25
DPU2DPU Resets.....	27
DPU2DPU Receive Status	28
Register Access	29
Register Reset.....	30
Register Clock.....	30
Memory Mapped DMA Access	31
Clock 400M.....	32
400M Reset.....	32
Clock 500M.....	33
500M Reset.....	33
Sync Puse	33
GP_Trigger_IN.....	34
GP_Trigger_OUT	34
HS_TRIGGER_OUT.....	35
Gearbox Clear	36
Boundary Scan	37
BSP IP Repository	38
FPGA A Sandbox 1 Configuration A	38
FPGA A Sandbox 1 Configuration B	40
FPGA B Sandbox 2 Configuration B	42
Common	44
reset_chain	44
Example Designs	45
M8132A Template Example Design Description.....	48
General	48
FPGA A example design for 4 ODI operation	49
FPGA A example design for 3 ODI plus 8 10GbE operation.....	50
FPGA B example design for either modes of operation	51
ODI.....	52
General.....	52
Basic Operation.....	52
A Note on ODI over AXI4S.....	52
A Note on the 400bit bus operation.....	53
The 4 ODI FPGA A Example Design ODI features	53
The 3 ODI FPGA A Example Design features	54
The FPGA B Example Design features	54
Source Code.....	55

Memory Map	56
ODI Test Control & Status	57
ODI Mux Control Ctl/Stat	58
ODI Pass Through Ctl/Stat	58
Capture Memory	59
Register Descriptions	59
Test Packet Generation Registers	59
Interlaken Channel	59
Transmit General	61
RX Status	63
Receive General	64
RX Active Clocks Lower	64
RX Active Clocks Upper	64
RX Total Clocks Lower	64
RX Total Clocks Upper	64
Segment Errors	64
Core Clock Rate	65
TX Clock Crossing fifo depth	65
TX Clock Crossing fifo depth max	65
RX Clock Crossing fifo depth	65
RX Clock Crossing fifo depth max	65
RX Flow Control fifo depth	65
RX Flow Control fifo depth max	65
ODI Pass Through Ctl/Stat	66
General	66
DPU_BB_ODI_CHANNEL	66
DPU_BB_ODI_PACKET_LENGTH	66
DPU_BB_CONTROL	66
DPU_BB_STATUS	66
General	67
DPU_BB_DATA_COUNT	67
DPU_BB_CORE_CLOCK_RATE	67
ODI Mux Control Ctl/Stat	67
DPU_MUX_SELECT	67
Valid Combinations for in 2 and out 4	68
Valid Combinations for in 4 and out 2	68
Valid Combinations for FPGA B or FPGA A in 3 ODI plus 10GbE operation	69
NOTES:	69
DPU_MUX_CONTROL	70
DPU_MUX_STATUS	70
DPU_MUX_DATA_COUNT_LOWER	71
DPU_MUX_DATA_COUNT_UPPER	71
DPU_MUX_TX_FIFO_DEPTH	71
DPU_MUX_RX_FIFO_DEPTH	71
DPU_MUX_STARTUP_DEPTH	71
DPU_MUX_EMPTY_DEPTH	71
DPU_MUX_ODI_PACKET_LENGTH	71
Capture Control	72
General	72
Capture Control	72
Format	72
Capture Data Vectors	72
Read Request Addr	73
StartAddr and Offset	73
Read Request Samples	73
Read Request Control	73
Read Data	74
Abort	74
Capture Memory Status Register	74
DDC Access Control Register	74
ReadData(0 - F) Read only	75
ReadData0	75
ReadData1	75

ReadData2.....	75
ReadData3.....	75
ReadData4.....	75
ReadData5.....	75
ReadData6.....	75
ReadData7.....	75
ReadData8.....	75
ReadData9.....	75
ReadDataA.....	75
ReadDataB.....	75
ReadDataC.....	75
ReadDataD.....	75
ReadDataE.....	75
ReadDataF.....	75
General.....	76
Ethernet.....	78
ET Control.....	79
ET TX Address.....	79
ET TX Length.....	80
ET TX IPG.....	80
ET TX Repeats.....	80
ET RX Address.....	80
ET RX Length.....	80
ET Rx Errored Count.....	80
ET Rx Length Errored Count.....	80
ET Rx Data Errored Count.....	81
ET Tx Ack Count.....	81
ET RX Ack Count.....	81
Ethernet Test Operation and procedures.....	82
Addresses.....	82
Source Address Override.....	83
Examples.....	83
Loopback Dongle.....	83
Breakout cable.....	83
DPU2DPU.....	87
General.....	87
A note on the DPU2DPU AXI4S stream.....	88
Memory Map.....	88
DPU2DPU Test Control & Status.....	89
Register Descriptions.....	89
DPU2DPU Test Packet Generation Registers.....	89
Test Benches.....	93
m8132a_pw_a_sandbox_a_example_4_2_tb.vhd.....	93
m8132a_pw_d2d_b_sandbox_b_example_4_2_tb.vhd.....	93
m8132a_pw_d2d_eth_a_sandbox_b_example_4_2_tb.vhd.....	94
dpu_caa_pw_example_4_2_data_512_capture_tb.vhd.....	94
capture_mem_top_tb.vhd.....	95
dpu_caa_pw_example_4_2_ethernet_tb.vhd.....	95
dpu_caa_pw_example_4_2_data_400_path_tb.vhd.....	96
dpu_caa_pw_b_example_4_2_data_400_loopback_tb.vhd.....	96
dpu_caa_pw_example_4_2_data_400_loopback_tb.vhd.....	97
dpu_caa_pw_a_example_4_2_dpu2dpu_tb.vhd.....	97
dpu_caa_pw_d2d_b_example_4_2_dpu2dpu_tb.vhd.....	97
dpu_caa_pw_example_4_2_dpu2dpu_tb.vhd.....	97
Bus Widths and Sample Ordering.....	98
400Bit to 512Bit Gearbox and ODI Transmission.....	98
512Bit and ODI Transmission.....	100
M8132A Register Example.....	105
M8132A Trigger Out Example.....	108
32GS/s TX/RX ODI Connection.....	114

Installation Requirements

In addition to the PathWave FPGA system requirements, this FSP requires the following software to allow the output products to be used with the M8132A Module

Vendor	Software	Release Supported	May work, but not supported	Release Explicitly not-supported
Keysight	M8131A/M8132A SFP Drivers, programming libraries and Software Front Panel for the following modules: M8132A	4.1.0.0 to 4.2.X.X	NA	Older versions
Xilinx	Vivado	2019.2	2020.X, 2021.X	Versions older than 2019.2

Overview

The M8132A is a high performance DSP module, which incorporates 160Gb/s optical IO using ODI, and as of release 4.1 has the option to replace one of the ODI links with eight 10GbE Optical Links compatible with IEEE 802.3-2012 10GBASE-SR optical interfaces.

The M8132A data sheet outlines the performance specifications for the M8132A and should be available at:

<http://www.keysight.com/find/M8132A>

The M8132A User's Guide provides information on the modules base operation, and information on the example software that operates with the example design templates provided with the FSP

It is available from the drop down menu "Help" → "User Guide" from the soft front panel of the M8132A, or from the start menu folder on the machine with the M8131A/M8132A main install: Keysight M8131 → Keysight M8131 Documentation. Then open and "M8132A User's Guide.pdf"

Scope

This document is a technical reference for the M8132A 640 Gb/s Digital Signal Processor Module, with regard to the configurable regions of the M8132A FPGAs. The FPGA configuration regions utilize Xilinx FPGA partial reconfiguration technology to allow customers to design and configure defined sections of the M8132A FPGAs without the need to power down or reboot the M8132A or host computer, respectively.

Available FPGA Features and Resources

The M8132A is purchased as a 4 channel processing system.

The M8132A supports two customer accessible areas (CAA), each supporting 2 main IO channels, and as of release 4.1, two inter-FPGA bi-directional links, and the option to replace one of the ODI channels with 8 10GbE links.

There is one CAA per FPGA, the FPGA are both xcvu9p-flga2577-2L-e.

There are two different startup options for the M8132A Soft Front Panel (SFP), which load up either one of two possible pairs of static regions with default partial Customer Accessible Area contents.

See the M8132A User's Guide Chapter 3 section dealing with Command Line Arguments for information on how to start the SFP with the eight 10GbE ports instead of the 4th ODI port.

There are three configurations supplied with the FPGA Support Package (FSP) that allow the user to build partial bit-streams suitable for the particular startup static regions in use.

The "A" configuration for FPGA A supports the 4 ODI IO operation, the "B" configuration for FPGA A supports the 3 ODI plus 8 10GbE IO operation.

There is one "B" configuration for FPGA B which is used for both startup options.

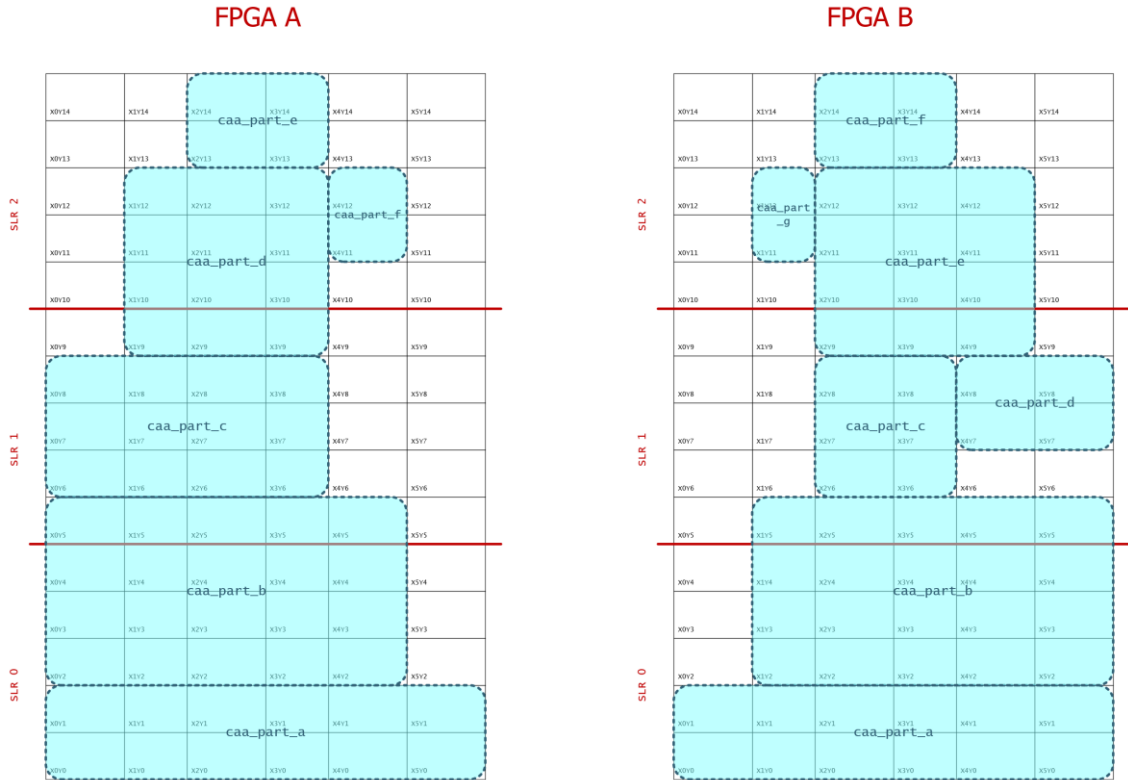
The following table outlines the FPGA resources which are available for custom logic within the sandbox per FPGA.

(Note these values are less than the total FPGA resources within the chip, due to the requirement to provide infrastructure within the FPGA to support the customer accessible area).

The two different configurations for FPGA A have the same amount of resources within the customer accessible area, FPGA B has slightly less available resource due to static area signal requirements for routing to the PCB.

Resource Type	Resource Size for customer region A	Resource Size for customer region B
CLB Flip-FLOPs (k)	1,584	1,530
CLB LUTs (k)	792	765
Max Dist. RAM (Mb)	24.8	23.9
Total Block RAM (Mb)	50.625	48.1
Ultra RAM (Mb)	229.5	225
Clock Mgmt Tiles (CMTs)	62	60
DSP Slices	4,848	4224
Global Buffers	1020	1032

The figure below shows in pale blue shading the customer accessible areas for the two FPGA.



Available Configurations

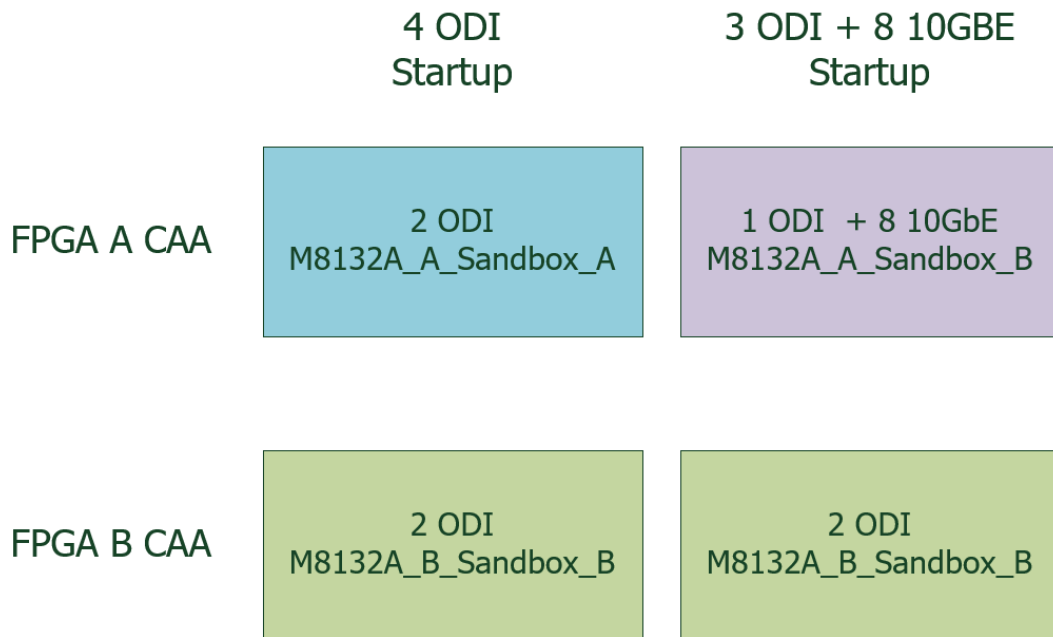
There are two configurable regions per M8132A, one per FPGA.

There is one supported FSP version, with two configurations for FPGA A, and one configuration for FPGA B, M8132A_A_Sandbox_A, M8132A_A_Sandbox_B and M8132A_B_Sandbox_B.

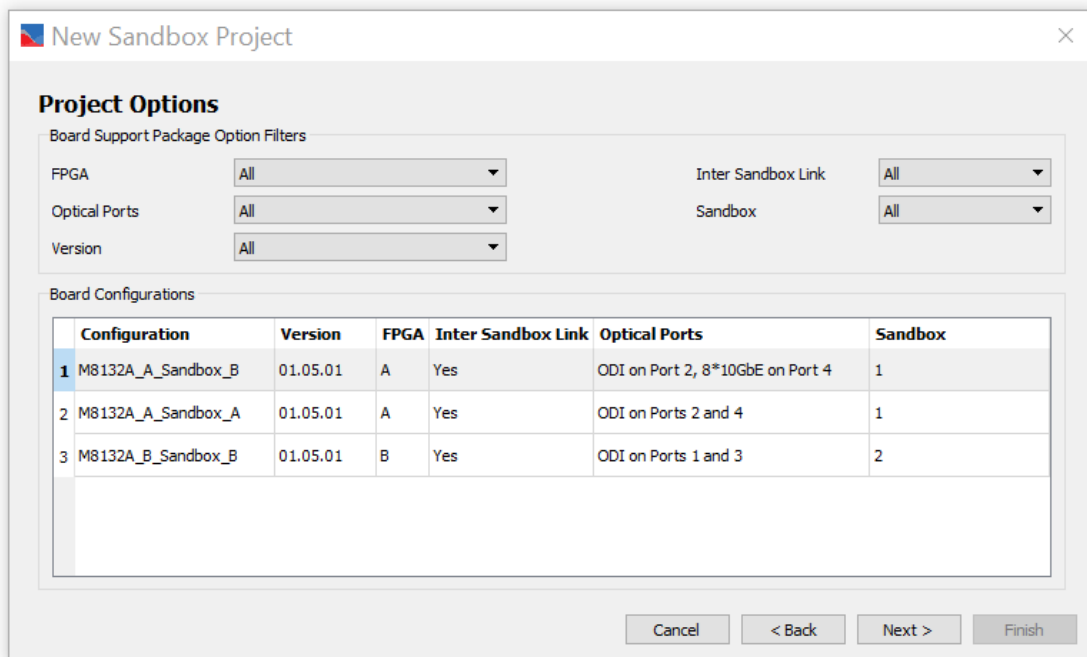
The table below shows the mapping of the static region with the various configurations to allow the user to determine which configuration to use when creating partial designs.

Connectivity	Customer Accessible Area A configurations	Customer Accessible Area A main ports	Customer Accessible Area B configurations	Customer Accessible Area B main ports
3 ODI plus 8 10GbE	M8132A_A_Sandbox_B	1 ODI, 8 10GbE Ethernet, DPU2DPU	M8132A_B_Sandbox_B	2 ODI, DPU2DPU
4 ODI	M8132A_A_Sandbox_A	2 ODI, DPU2DPU	M8132A_B_Sandbox_B	2 ODI, DPU2DPU

The figure below shows in an alternate view to the table above, the main differences between the two startup options.



The screenshot below shows one stage of the New Sandbox Project dialog within the Pathwave FPGA tool, with the 3 different configurations available shown. Various filters can be applied to narrow down to the particular configuration to use depending on the optical interfaces desired in the design.



Port Mapping



The image above shows a CAD representation of the front panel of the M8132A module. The steel spring loaded shutters protect the optical interfaces. See the main M8132A User Guide for more details on the various input and output signals and connectors.

Of interest here is the mapping between the front panel ports and the two FPGA and the Customer Accessible Areas.

FPGA B is connected to the Optical ports 1 and 3, and these ports are always ODI.

FPGA A is connected to the Optical ports 2 and 4, with port 2 always carrying ODI, while port 4 will either have an ODI interface or eight 10GbE interfaces. See the M8132A User Guide for more details on the optical fibers required for connection to the module.

FPGA Configurable Region Interfaces

The following sections describe the physical interfaces which are available within a M8132A FPGA configurable region.

The ODI interfaces are synchronous to Clock_ODI.

The AXI4Lite interface for register write and read is synchronous to Clock_AXI4LITE_REGS

The 10Gb Ethernet interfaces are synchronous to Clock_400M

The DPU2DPU inter-CAA links are synchronous to Clock_400M

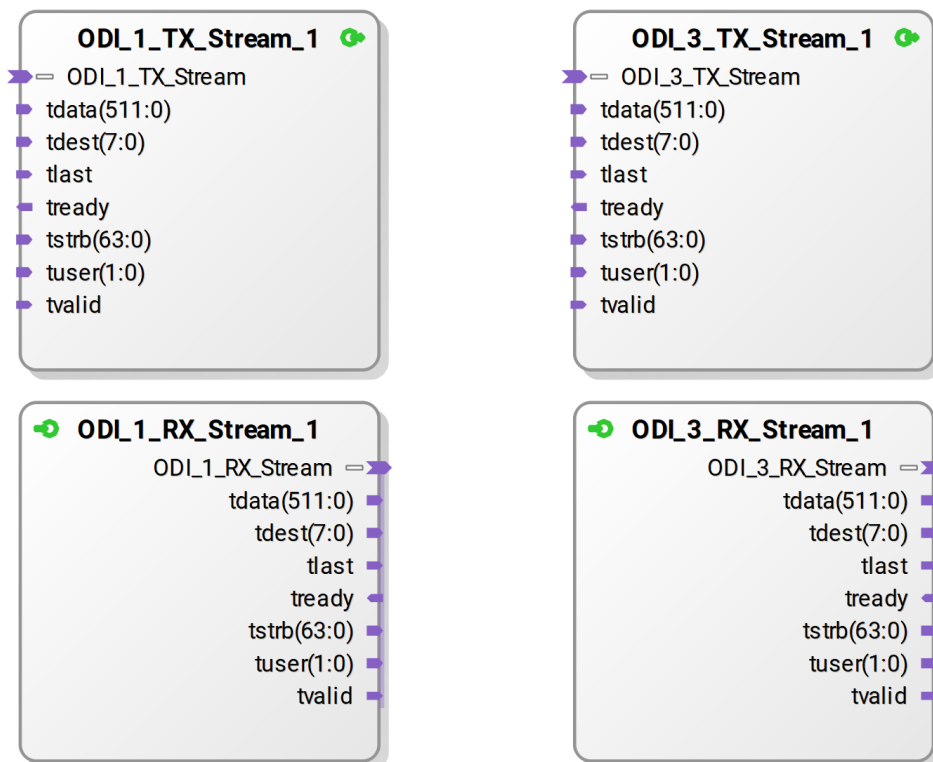
The trigger signals are synchronous to Clock_400M. Please see the module's User's Guide for a more detailed discussion of the trigger system before using these signals.

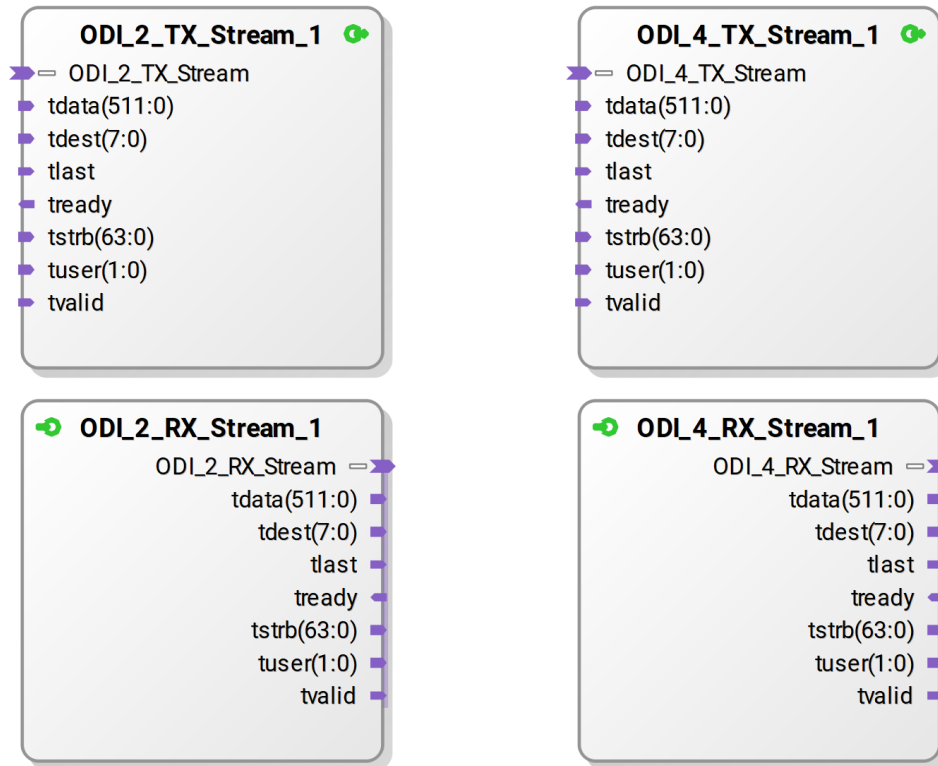
ODI - Optical Data Interface

There are either one or two ODI interface per configurable region, each ODI interface is separated between transmit (TX) and receive (RX)

Customer Region B Sandbox 2 always has 2 ODI interfaces available, connected to Port 1 and Port 3

Customer Region A Sandbox 1 either has 1 ODI interface, connected to Port 2, or two ODI interfaces, connected to Port 2 and Port 4 available, depending on which configuration is used, to match the startup options of the M8132A Soft Front Panel.





The signals are synchronous with Clock_ODI, (332MHz)

ODI_2_RX_Stream and ODI_4_RX_Stream are inputs to partial region 1, within FPGA A

ODI_2_TX_Stream and ODI_4_TX_Stream are outputs from partial region 1, within FPGA A

ODI_1_RX_Stream and ODI_3_RX_Stream are inputs to partial region 2, within FPGA B

ODI_1_TX_Stream and ODI_3_TX_Stream are outputs from partial region 1, within FPGA B

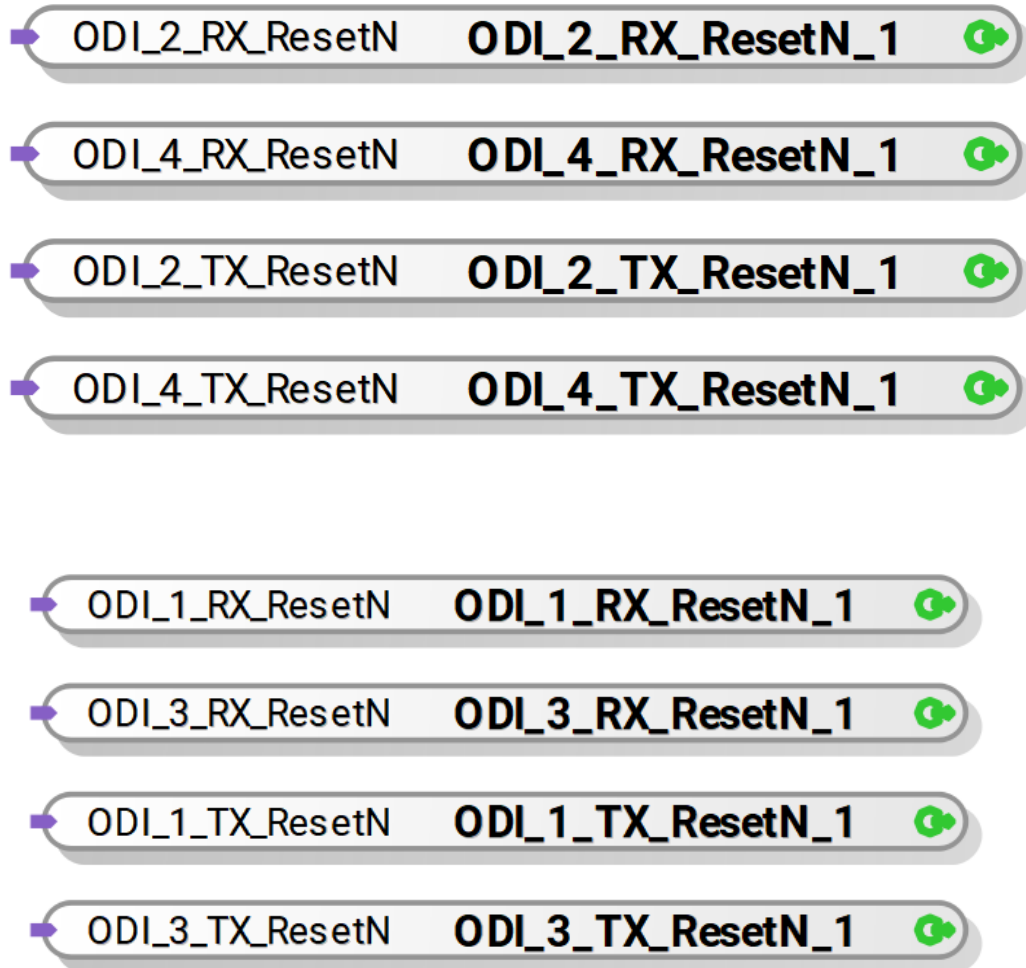
ODI_4_TX_Stream and ODI_4_RX_Stream are only available within the configuration M8132A_A_Sandbox_A, the configuration for use with a 4 ODI startup.

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
ODI_X_TX_Stream : AXI4S	tvalid	1		input to interface, standard AXI4S use
	tready	1		output from interface, standard AXI4S use
	tdata	512		input to interface, 512bit, 64 byte data, must be qualified by the TSTRB signal. Either the full 64 bytes, or the upper 32 bytes are only valid combinations of data. ODI Packet Data can be either odd or even multiple of 32bytes long. Minimum size is 64bytes
	tdest	8		input to interface, maps to the ODI channel number
	tlast	1		input to interface, indicates this current transfer includes the last transfer of a packet

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
	tstrb	64		input to interface, qualifies the data, only two values are valid, either 0xFFFFFFFFFFFFFFFF i.e. ALL 64 bytes are valid, or 0xFFFFFFFF00000000 i.e. only the TOP 32 bytes are valid. Further, 0xFFFFFFFF00000000 is only valid when TLAST is also true, i.e. the end of an odd multiple of 32byte packet.
	tuser	2	1	Input to interface, err_in, allows the user to indicate something is wrong with the current packet being sent into the interface, only valid when TLAST is true. This indicator will be passed out at the far end allowing the receiving device to take action if appropriate.
			0	Input to interface, "midword" indicator. Only valid when TLAST is true. If the current transfer is a full 64byte transfer, and TLAST is true, midword as 0 indicates the bottom 32 bytes is the last portion of the current packet. If midword is a 1, this indicates that the top 32bytes is the last portion of one packet, and the bottom 32bytes is the first portion of the next packet.
ODI_X_RX_Stream : AXI4S				AXI4 Streaming interface for connection to the receive half of an ODI port with a 512Bit data interface, and various sideband channels. Data format is variable, but includes RAW ODI, or Vita-49 wrapped ODI depending on the remote source generating the data
	tvalid	1		output from interface, standard AXI4S use
	tready	1		input to interface, standard AXI4S use
	tdata	512		output from interface, 512bit, 64 byte data, is qualified by the TSTRB signal. Either the full 64 bytes, or the upper 32 bytes are valid combinations of data. ODI Packet Data can be either odd or even multiple of 32bytes long. Minimum size is 64bytes
	tdest	8		output from interface, maps to the ODI channel number
	tlast	1		output from interface, indicates this current transfer includes the last transfer of a packet
	tstrb	64		output from interface, qualifies the data, only two values are valid, either 0xFFFFFFFFFFFFFFFF i.e. ALL 64 bytes are valid, or 0xFFFFFFFF00000000 i.e. only the TOP 32 bytes are valid. Further, 0xFFFFFFFF00000000 is only valid when TLAST is also true, i.e. the end of an odd multiple of 32byte packet.
	tuser	2	1	output from interface, err_out, allows the user to be informed that something is wrong with the current packet being received from the interface, only valid when TLAST is true. This indicator may have been passed in at the far end before transmission, or it was generated by the ODI cores to indicate a transmission

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
				error of some sort, allowing the receiving device to take action if appropriate.
			0	output from interface, "midword" indicator. Only valid when TLAST is true. If the current transfer is a full 64byte transfer, and TLAST is true, midword as 0 indicates the bottom 32 bytes is the last portion of the current packet. If midword is a 1, this indicates that the top 32bytes is the last portion of one packet, and the bottom 32bytes is the first portion of the next packet.

ODI Resets



Each ODI port has a separate reset that should be driven by user logic

This reset acts upon the AXI4S interface of the ODI function within the static area, it does not reset the optical side of the ODI function

The reset should be driven and released before users try to transmit to or receive from an ODI port.

In general, In the case of receiving from an ODI port, the local RX Interface should be reset and the user logic waiting to receive before the far end remote device starts transmitting.

If receive flow control is active on the local ODI interface in the static region, and the far end remote device supports flow control, then once the local RX interface reset is released, the far end remote device could start transmission, before the local logic is ready, with the flow control holding off the far end until the local logic starts receiving

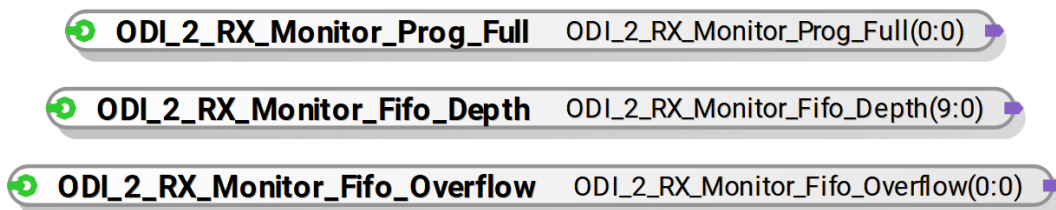
Interface Name	signal name	Width (bits)	Bit Field	Description
ODI_X_TX_ResetN		1		input to interface, resets the user side of the tx ODI Interface

Interface Name	signal name	Width (bits)	Bit Field	Description
ODI_X_RX_ResetN		1		input to interface, resets the user side of the rx ODI Interface

ODI Startup reset

There is no individual ODI startup reset, if one is needed, use a suitable synchronization method to create one from the 400M startup reset signal "Clock_400_ResetN"

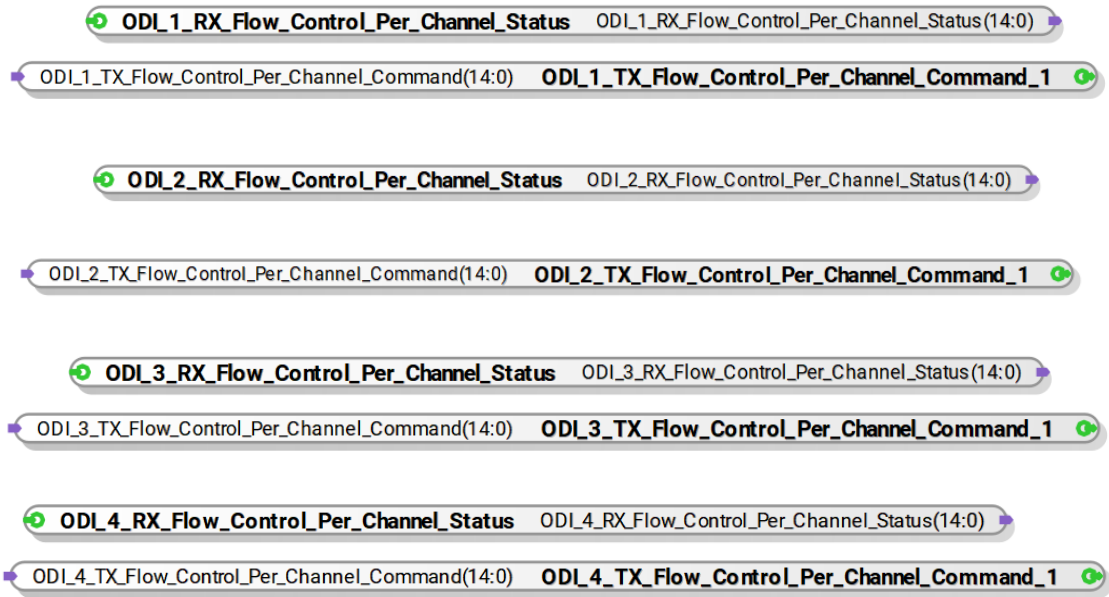
ODI Receive Status



These three signals provide a means for the user logic within the configurable region to monitor the status of the flow control fifo within the static area ODI function.

Interface name	Signal name	Width (bits)	Bit Field	Description
odi_X_rx_prog_full		1		output from interface, the ODI interface in the static area has a programmable depth guide for the internal fifo, this bit represents the current depth as being above if true, or below , if false, the current set level
odi_X_rx_fifo_depth		10		output from interface, the ODI interface in the static area has a fifo for dealing with flow control, this output represents the current depth
odi_X_rx_ovf		1		output from interface, the ODI interface in the static area has a fifo for dealing with flow control, this output represents the fact that the far end ignored local flow control messages send to the far end, and caused the fifo to fill, and have more information arrive that was dropped and not written to the fifo. This is an error condition, the ODI interface needs deactivated and reactivated to clear this condition

ODI Per Channel Flow Control



For Future Use

ODI Clock



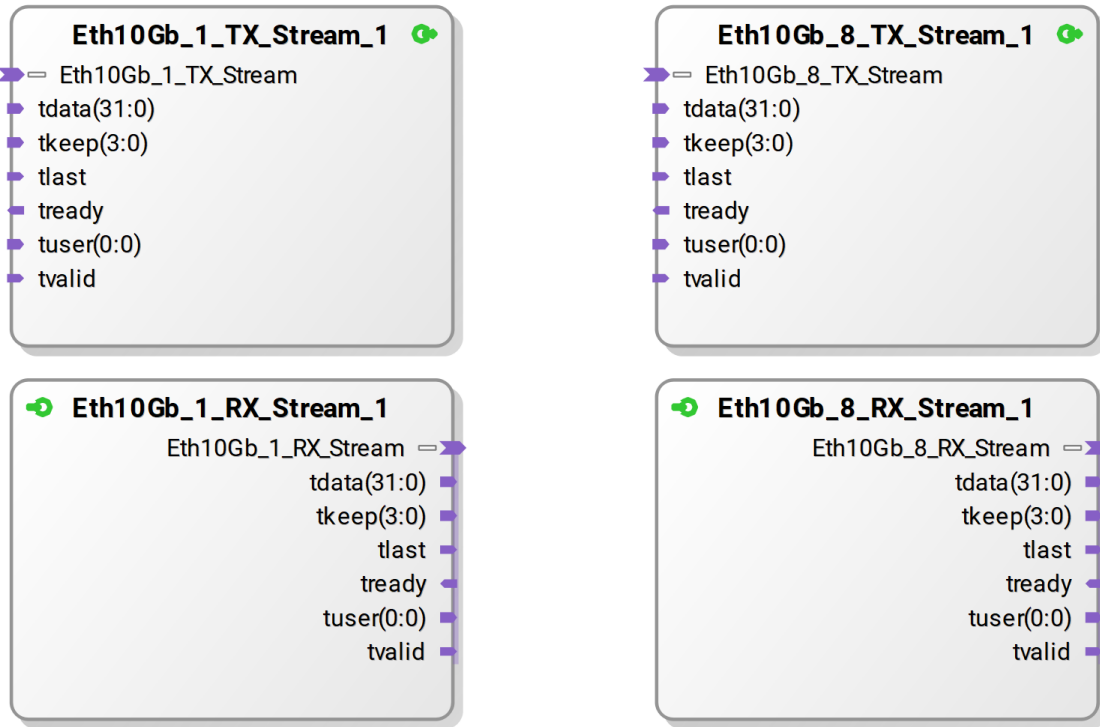
Interface Name and Type	Signal Name	Width (bits)	Description
Clock_ODI		1	ODI interface clock, 332MHz The ODI AXI4S interfaces are synchronous with this clock

10Gbe Ethernet Interface

Configuration M8132A_A_Sandbox_B gives the user the choice to connect up to to eight 10Gb Ethernet interfaces instead of the ODI interface on optical port 4

Each Ethernet interface is separated between transmit (TX) and receive (RX)

The picture here just shows the first pair, Eth10Gb_1_TX_Stream, Eth10Gb_1_RX, and the last pair, Eth10Gb_8_TX_Stream, Eth10Gb_8_RX_Stream.



The other 10GbE Ethernet Interfaces, 2 to 7 have the same connectivity.

The signals are synchronous with Clock_400M, (400MHz) They are transferred to/from 312.5Mhz clocks in the static area.

Eth10Gb_1_RX_Stream to Eth10Gb_8_RX_Stream are inputs to partial region 1, within FPGA A of configuration M8132A_A_Sandbox_B

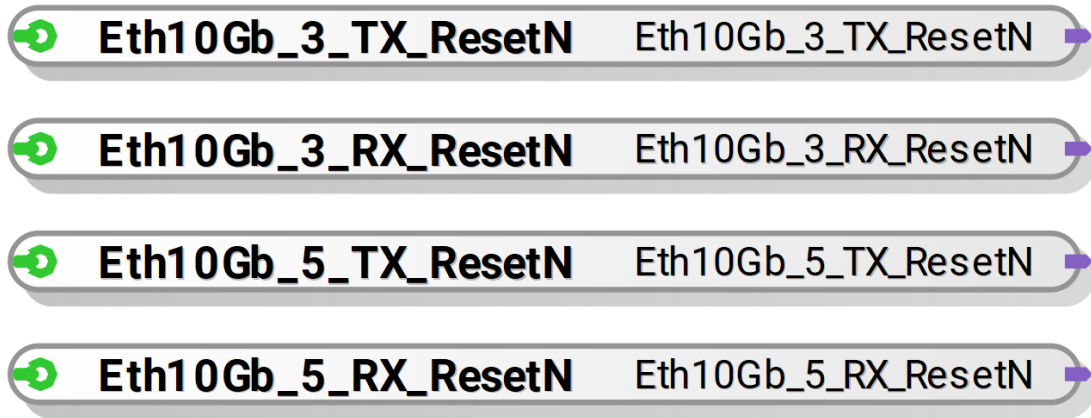
Eth10Gb_1_TX_Stream to Eth10Gb_8_TX_Stream are outputs from partial region 1, within FPGA A of configuration M8132A_A_Sandbox_B

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
Eth10Gb_X_TX_Stream : AXI4S	tvalid	1		input to interface, standard AXI4S use
	tready	1		output from interface, standard AXI4S use
	tdata	32		input to interface, 32bit, 4 byte data. When tlast is low all 4 bytes are valid. When tlast is high 1-4 bytes will be valid according to the settings on tkeep

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
	tlast	1		input to interface, indicates this current transfer includes the last transfer of a packet
	tkeep	4		input to interface, qualifies the last word of data, "1111" ₂ indicates all 4 bytes (bits 31:0) are valid. "0111" ₂ indicates 3 bytes (bits 23:0) are valid. "0011" ₂ indicates 2 bytes (bits 15:0) are valid. "0001" ₂ indicates 1 byte is valid. All other patterns on tkeep are invalid. Only relevant when tlast is set.
	tuser	1		Input to interface, should be set if the customer cannot finish the current packet.
Eth10Gb_X_RX_Stream : AXI4S				AXI4 Streaming interface for connection to the receive half of an Ethernet port with a 32Bit data interface, and various sideband channels.
	tvalid	1		output from interface, standard AXI4S use
	tready	1		input to interface, standard AXI4S use
	tdata	32		output from interface, 32bit, 4 byte data. When tlast is low all 4 bytes are valid. When tlast is high 1-4 bytes will be valid according to the settings on tkeep .
	tlast	1		output from interface, indicates this current transfer includes the last transfer of a packet
	tkeep	4		output from interface, qualifies the last word of data, "1111" ₂ indicates all 4 bytes (bits 31:0) are valid. "0111" ₂ indicates 3 bytes (bits 23:0) are valid. "0011" ₂ indicates 2 bytes (bits 15:0) are valid. "0001" ₂ indicates 1 byte is valid. All other patterns on tkeep are invalid. Only relevant when tlast is set.
	tuser	1	0	output from interface. valid at the same time as tlast. Indicates something wrong with the packet. Error conditions include packet too short, packet too long, FCS error, XGMII code error or bad control frame received.

When a packet is transmitted over the interface there should be no gaps in tvalid for the duration of the packet. In the receive direction rx tready should be high always. The receiver is not allowed to push back.

10Gbe Ethernet Resets



Each Ethernet port has a separate reset input that should be connected to user logic

The diagram above only illustrates the resets for the transmit and receive ports for Ethernet channels 3 and 5, there are transmit and receive resets for all 8 Ethernet transmit and receive channels.

These resets come from the 10Gb Ethernet functionality in the static area, and should be used to reset user logic within the Customer Accessible Area

Interface Name	signal name	Width (bits)	Bit Field	Description
Eth10Gb_X_TX_ResetN		1		output from the interface, should be used to reset the user logic transmit Ethernet circuitry

Interface Name	signal name	Width (bits)	Bit Field	Description
Eth10Gb_X_RX_ResetN		1		output from the interface, should be used to reset the user logic receive Ethernet circuitry

10GbE Ethernet Receive Status

 **Eth10Gb_2_RX_Status** Eth10Gb_2_RX_Status(2:0) 

 **Eth10Gb_7_RX_Status** Eth10Gb_7_RX_Status(2:0) 

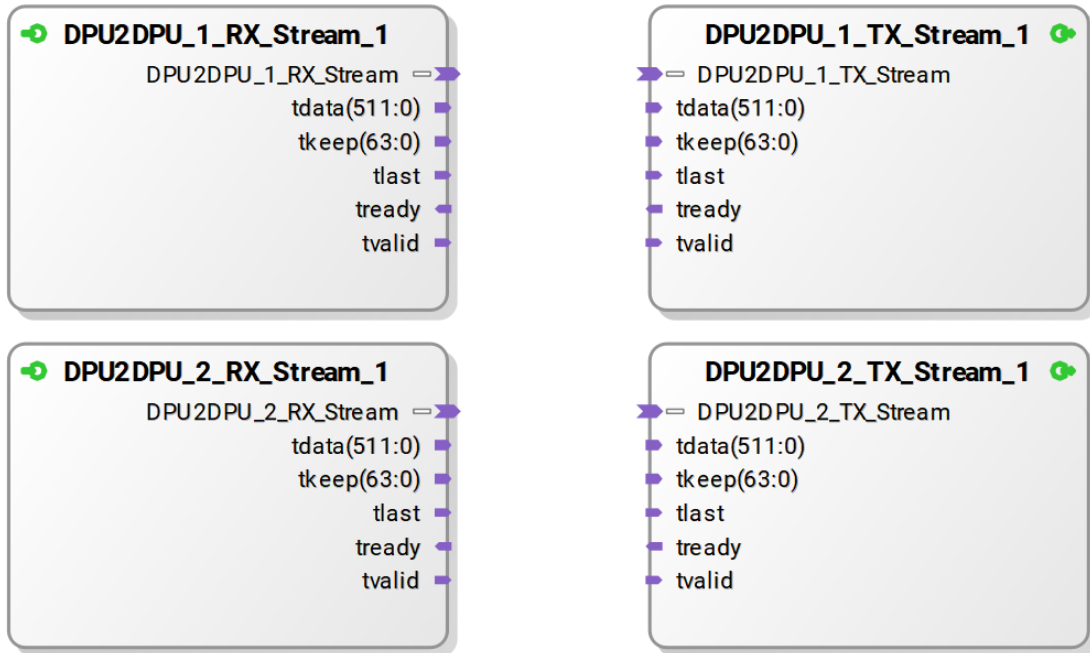
Each Ethernet receive port has a low speed status feed, the signals are synchronous with the 400MHz clock Clock_400M, but are not related to any particular AXI4S transaction on the main Ethernet Channel's streams.

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
Eth10Gb_X_RX_Status: 3 bit Bus		3	2	Block Lock Status: this indicates the Ethernet MAC within the static area for this channel is asserting that the receiver has achieved block lock as defined in Clause 49.2.14 of IEEE 802.3, level sensitive, active high
			1	Remote Fault Alarm: this indicates the Ethernet MAC within the static area for this channel is asserting the remote fault alarm signal, level sensitive, active high
			0	Local Fault Alarm: this indicates the Ethernet MAC within the static area for this channel is asserting the local fault alarm signal, level sensitive, active high

DPU2DPU

There are two DPU2DPU AXI4S interfaces which permit transfer of data between the two Customer Accessible Areas, i.e. between Sandbox 1 and Sandbox 2

The two links are bidirectional, and are provided as separate transmit and receive streams



The signals are synchronous with the Clock_400M

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
DPU2DPU_X_TX_Stream : AXI4S	tvalid	1		input to interface, standard AXI4S use
	tready	1		output from interface, standard AXI4S use
	tdata	512		input to interface, 512bit, 64 byte data. When tlast is low all 64 bytes are valid. When tlast is high 1-64 bytes will be valid according to the settings on tkeep
	tlast	1		input to interface, indicates this current transfer includes the last transfer of a packet
	tkeep	64		input to interface, qualifies the last word of data, "ffffffffffffffff" ₁₆ indicates all 64 bytes (bits 511:0) are valid. "7fffffffffffffff" ₁₆ indicates 63 bytes (bits 503:0) are valid. "0000000000000003" ₁₆ indicates 2 bytes (bits 15:0) are valid. "0000000000000001" ₁₆ indicates 1 byte is valid. Only relevant when tlast is set.

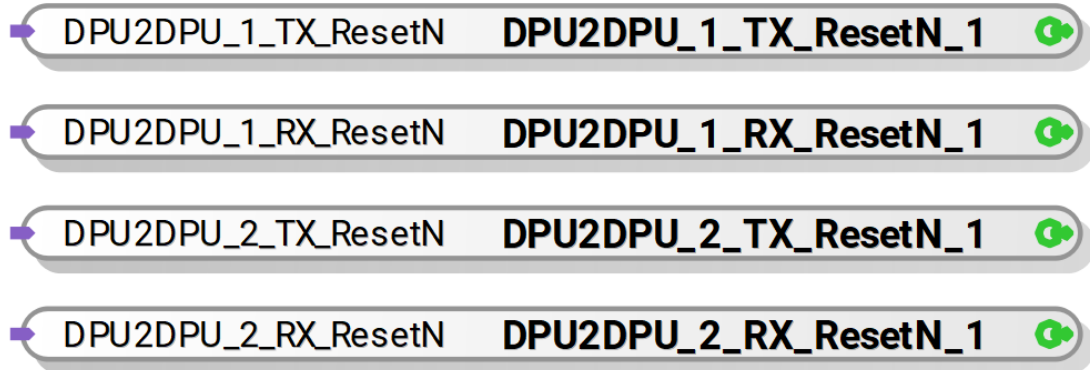
Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
DPU2DPU_X_RX_Stream : AXI4S				AXI4 Streaming interface for connection to the receive half of a DPu2DPU port with a 32Bit data interface, and various sideband channels.
	tvalid	1		output from interface, standard AXI4S use
	tready	1		input to interface, standard AXI4S use
	tdata	512		output from interface, 512bit, 64 byte data. When tlast is low all 64 bytes are valid. When tlast is high 1-64 bytes will be valid according to the settings on tkeep .
	tlast	1		output from interface, indicates this current transfer includes the last transfer of a packet
	tkeep	64		ouput from interface, qualifies the last word of data, "ffffffffffffffff" ₁₆ indicates all 64 bytes (bits 511:0) are valid. "7fffffffffffffff" ₁₆ indicates 63 bytes (bits 503:0) are valid. "0000000000000003" ₁₆ indicates 2 bytes (bits 15:0) are valid. "0000000000000001" ₁₆ indicates 1 byte is valid. Only relevant when tlast is set.

Data sent to one of the TX interface of Sandbox 1 will appear at the appropriate RX interface of Sandbox 2. Similarly, data sent to one of the TX interface of Sandbox 2 will appear at the appropriate RX interface of Sandbox 1.

The Receiver side should be started, and assert tready on the "listening" sandbox before the transmission on the "speaking" sandbox side is started. There are fifos within the static area to handle clock domain crossing, but they are not large enough to handle very much data before they fill and data is dropped, there is no back-pressure on the actual inter chip links, so data will be dropped if the transmitter is started before the receiver is ready.

There is back pressure on the transmit clock domain crossing fifo in the sending side interface of the static area, so user code within the CA must obey the usual AXI4S tready signal on that interface to prevent data loss

DPU2DPU Resets



Each DPU2DPU port has a separate reset that should be driven by user logic





This reset acts upon the AXI4S interface of the DPU2DPU function within the static area, it does not reset the actual links between the FPGA

The reset should be driven and released before users try to transmit to or receive from a DPU2DPU port.

Interface Name	signal name	Width (bits)	Bit Field	Description
DPU2DPU_X_TX_ResetN		1		input to interface, resets the user side of the tx DPU2DPU interface logic in the static region of the FPGA

Interface Name	signal name	Width (bits)	Bit Field	Description
DPU2DPU_X_RX_ResetN		1		input to interface, resets the user side of the rx DPU2DPU interface logic in the static region of the FPGA

DPU2DPU Receive Status

 DPU2DPU_1_RX_Monitor_Overflow	DPU2DPU_1_RX_Monitor_Overflow
 DPU2DPU_1_RX_Monitor_Fifo_Depth	DPU2DPU_1_RX_Monitor_Fifo_Depth(7:0)
 DPU2DPU_2_RX_Monitor_Overflow	DPU2DPU_2_RX_Monitor_Overflow
 DPU2DPU_2_RX_Monitor_Fifo_Depth	DPU2DPU_2_RX_Monitor_Fifo_Depth(7:0)

These two signals provide a means for the user logic within the configurable region to monitor the status of the dpu2dpu link within the static region of the FPGA

Interface name	Signal name	Width (bits)	Bit Field	Description
DPU2DPU_X_RX_Monitor_Overflow		1		output from interface, if the far end transmits before the receiver is ready, the clock domain crossing fifo in the static area can overflow, which this bit records, cleared with the appropriate DPU2DPU_X_RX_ResetN signal
DPU2DPU_X_RX_Monitor_Fifo_Depth		8		output from interface, the DPU2DPU interface in the static area has a fifo for handling clock domain crossing. There is no push back from the local receiver to the far end transmitter, this signal allows a user to monitor how deep the fifo is being filled. Note there will be some delay in this interface updating as data arrives in this FPGA's static region over the DPU2DPU link from the far end. Any receiver logic in this sandbox should be capable of receiving whatever the transmitter in the other FPGA sandbox transmits

Register Access



There is one Register access port per configurable region, the signals are synchronous with clock_AXI4LITE_REGS, which runs at 125MHz

The user can choose between adding "RegisterBank" components, which hide the AXI4Lite bus, and use pre-built register components, or can have the full AXI4Lite bus for connecting to their own register system

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
Register : AXI4LITE_REGS				Register access interface, Up to 1Mbyte space, accessed in 4byte/32bit registers, i.e. 252,144 maximum registers (subject to place and route constraints, resource limitations etc)
	awprot	3	2:0	output from interface, normal axi4Lite operation
	awvalid	1		output from interface, normal axi4Lite operation
	awready	1		input to interface, normal axi4Lite operation
	awaddr	20	19:0	output from interface, normal axi4Lite operation
	bready	1		output from interface, normal axi4Lite operation
	bvalid	1		input to interface, normal axi4Lite operation
	bresp	2	1:0	input to interface, normal axi4Lite operation

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
	wvalid	1		output from the interface, normal axi4Lite operation
	wready	1		input to the interface, normal axi4Lite operation
	wdata	32	31:0	output from the interface, normal axi4Lite operation
	arprot	3	2:0	output from interface, normal axi4Lite operation
	arvalid	1		output from the interface, normal axi4Lite operation
	arready	1		input to the interface, normal axi4Lite operation
	araddr	20	19:0	output from the interface, normal axi4Lite operation
	rvalid	1		input to the interface, normal axi4Lite operation
	rready	1		output from the interface, normal axi4Lite operation
	rresp	2	1:0	input to the interface, normal axi4Lite operation
	rdata	32	31:0	input to the interface, normal axi4Lite operation

Register Reset



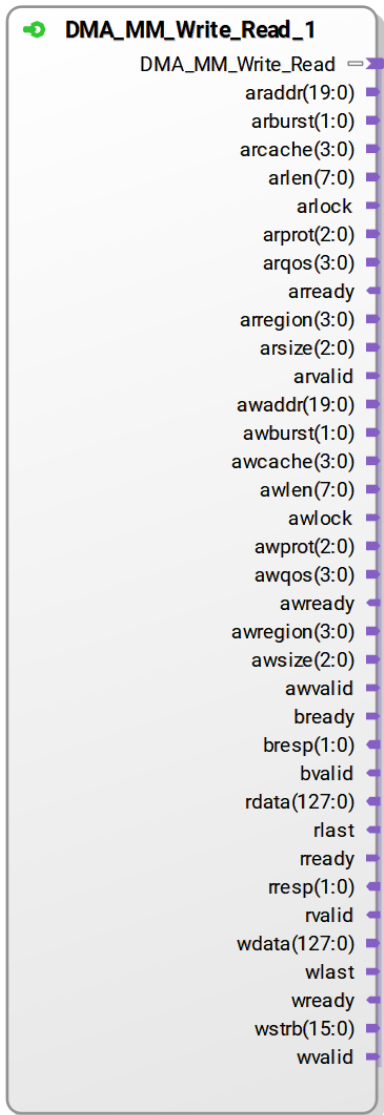
Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
AXI4LITE_REGS_ResetN		1		output from the interface, resets all slave interfaces

Register Clock



Interface Name and Type	Signal Name	Width (bits)	Description
Clock_AXI4LITE_REGS		1	Register access interface clock, 125MHz The AXI4Lite interface for register access is synchronous with this clock

Memory Mapped DMA Access



For Future Use

Memory Mapped DMA Reset



For Future Use

Memory Mapped DMA Clock



For Future Use

Clock 400M



Interface Name and Type	Signal Name	Width (bits)	Description
Clock_400M		1	Main data processing clock for 400bit, 40 samples at 10bits per sample For deterministic latency between an M8131A and an M8132A, this clock and a 40 times super sample 10bit scheme must be employed, further the reference input of the M8132A must be connected to the reference output of the M8132A to lock the 400M clocks together

400M Reset



Interface Name and Type	Signal Name	Width (bits)	Description
Clock_400_ResetN			Startup reset, synchronous to 400M clock, active low, Set active at programming, goes inactive "TBD" 400M clocks later

Clock 500M



Interface Name and Type	Signal Name	Width (bits)	Description
Clock_500M		1	Main data processing clock for 320bit, 32 samples at 10bits per sample Some tools for creating FPGA processing RTL automatically work better with power of two super sampling. This clock in conjunction with fifos and gearboxes can be used to convert a 400bit@400M data path to a 320bit@500M datapath

500M Reset



Interface Name and Type	Signal Name	Width (bits)	Description
Clock_500M_ResetN		1	Startup reset, synchronous to 500M clock, active low, Set active at programming, goes inactive "TBD" 500M clocks later

Sync Puse



See the M132A user guide for details on how to employ the sync pulse mechanism

Interface Name and Type	Signal Name	Width (bits)	Description
Sync_Pulse		1	Signal for co-ordination of operations across multiple FPGA, synchronous with the 400M clock

GP_Trigger_IN

GP_Trigger_IN GP_Trigger_IN(2:0)

Functionality is controlled by software which controls the mapping of various input signals of the module front panel to the connections of the Sandbox, see the M8132A user guide for more details

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
GP_Trigger_IN				General Purpose triggers into the Sandbox, synchronous to the 400M clock
	GP_Trigger_IN	3	2:0	input to the sandbox

GP_Trigger_OUT

GP_Trigger_OUT(2:0) **GP_Trigger_OUT_1**

Functionality is controlled by software which controls the mapping of the connections of the Sandbox to various output signals of the module front panel, see the M8132A user guide for more details

Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
GP_Trigger_OUT				General Purpose triggers out from the Sandbox, synchronous to the 400M clock
	GP_Trigger_OUT	3	2:0	output from the sandbox

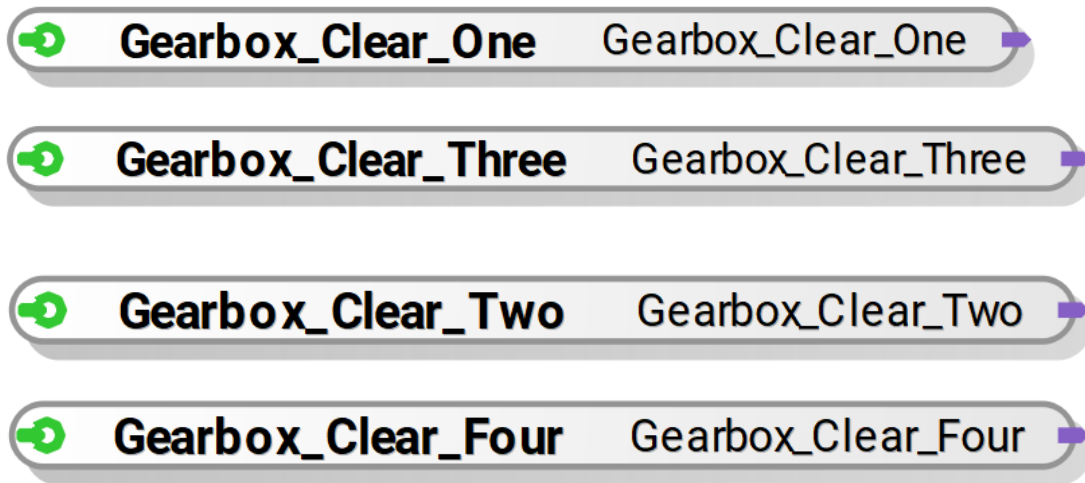
HS_TRIGGER_OUT

HS_Trigger_Out(39:0) **HS_Trigger_Out_1**

The High Speed Trigger out connects to the SMA output on the front panel of the M8132A

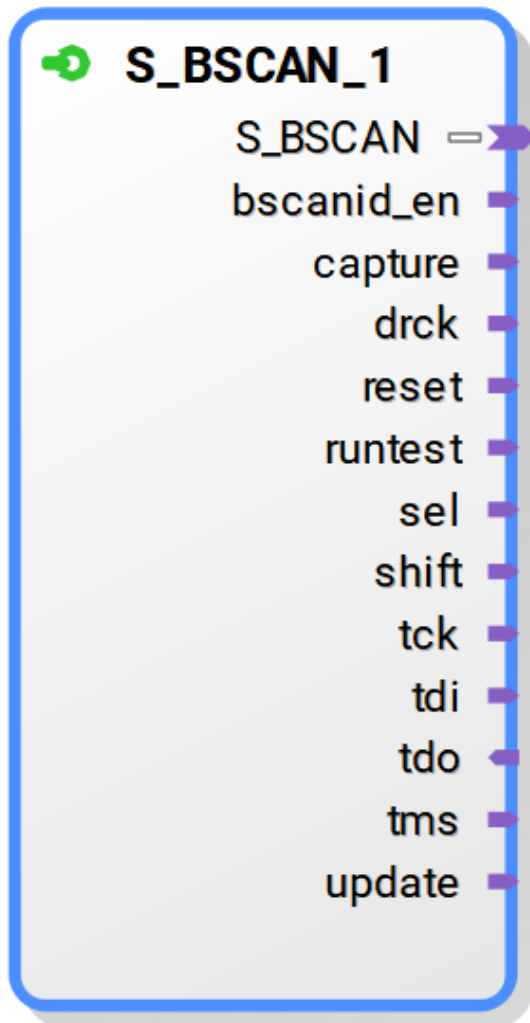
Interface Name and Type	Signal Name	Width (bits)	Bit Field	Description
HS_Trigger_Out				Source of signal to feed to the Front panel SMA connector synchronous to the 400M clock
	HS_Trigger_Out	40	39:0	output from the sandbox For instance, it could be used to feed the MSB of each of the 40 samples of the 400 bit 10 times super-sample bus at 400M within the sandbox when working with 10bit data, to create a waveform on the trigger out SMA, synchronous to the received data over ODI after it has been passed through an appropriate fifo and gearbox.

Gearbox Clear



Interface Name and Type	Signal Name	Width (bits)	Description
Gearbox_Clear_One Gearbox_Clear_Two Gearbox_Clear_Three Gearbox_Clear_Four	Gearbox_Clear_One	1	synchronous with Clock 400M, core_cclk system software sets and clears this bit to prime the 512-400 and 400-512 gearboxes in preparation for receiving ODI connection 1 streaming that originated in a 40 sample by 10bit @400M system
	Gearbox_Clear_Two	1	synchronous with Clock 400M, core_cclk system software sets and clears this bit to prime the 512-400 and 400-512 gearboxes in preparation for receiving ODI connection 2 streaming that originated in a 40 sample by 10bit @400M system
	Gearbox_Clear_Three	1	synchronous with Clock 400M, core_cclk system software sets and clears this bit to prime the 512-400 and 400-512 gearboxes in preparation for receiving ODI connection 3 streaming that originated in a 40 sample by 10bit @400M system
	Gearbox_Clear_Four	1	synchronous with Clock 400M, core_cclk system software sets and clears this bit to prime the 512-400 and 400-512 gearboxes in preparation for receiving ODI connection 4 streaming that originated in a 40 sample by 10bit @400M system

Boundary Scan



The M8132A Sandbox has the necessary signals on the interface to allow debugging with a Xilinx Integrated Logic Analyser (ILA)

In a Xilinx Virtex Ultrascale Plus partial reconfiguration region design, a Debug Bridge has to be employed to allow connection from the debug hub in the static region to any ILA generated as IP cores with appropriate Xilinx tools. At the time of the 2019.2 release, the "MARK_DEBUG" insertion flow is not supported for Reconfigurable modules. See Xilinx User Guide UG909 (v2019.2) January 2020 and Xilinx Product Guide PG245 December 5, 2018 for more details.

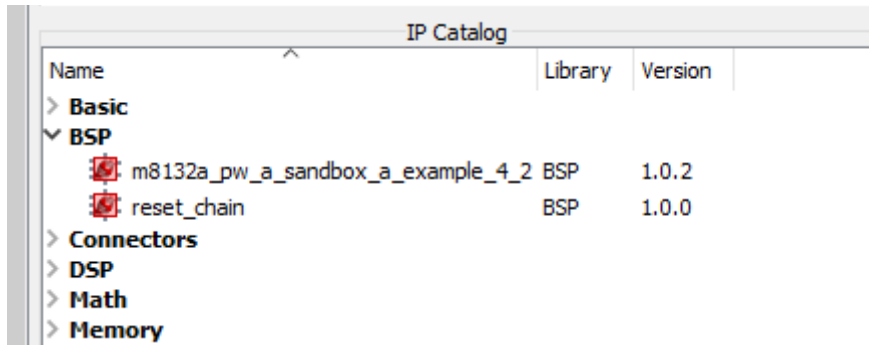
Interface Name and Type	Signal Name	Width (bits)	Description
S_BSCAN	bscanid_en	1	Standard Xilinx BSCAN signal
	capture	1	Standard Xilinx BSCAN signal
	drck	1	Standard Xilinx BSCAN signal
	reset	1	Standard Xilinx BSCAN signal
	runtest	1	Standard Xilinx BSCAN signal
	sel	1	Standard Xilinx BSCAN signal
	shift	1	Standard Xilinx BSCAN signal
	tck	1	Standard Xilinx BSCAN signal
	tdi	1	Standard Xilinx BSCAN signal
	tdo	1	Standard Xilinx BSCAN signal
	tms	1	Standard Xilinx BSCAN signal
	update	1	Standard Xilinx BSCAN signal

BSP IP Repository

FPGA A Sandbox 1 Configuration A

An IP component is provided with the FSP that replicates the default contents of the partial region as programmed at system startup when the static area and default partial area is programmed in FPGA A for 4 ODI operation

This IP component appears within the IP Catalog of the Pathwave FPGA window within the BSP group.



When dragged into the design area it appears so

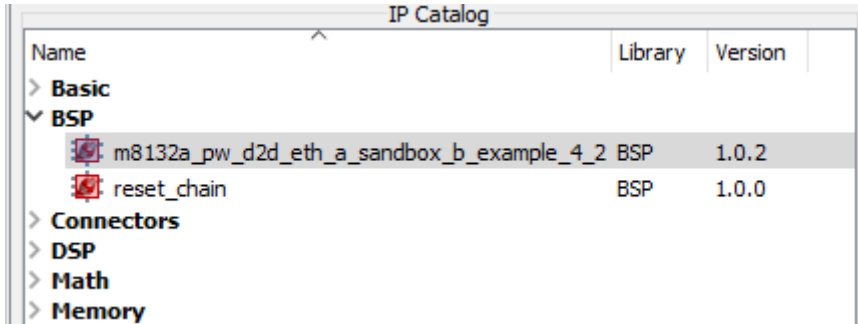
m8132a_pw_a_sandbox_a_example_4_2_1

- ▶ Clock_AXI4LITE_REGS
 - ▶ AXI4LITE_REGS_aresetn
 - ▶ Clock_400M
 - ▶ Clock_400_ResetN
 - ▶ Clock_500M
 - ▶ Clock_500_ResetN
 - ▶ Clock_ODI
 - ▶ Clock_DMA_MM
 - ▶ DMA_MM_Write_Read_ResetN
 - ▶ ⊕ M_AXI_MM
 - ▶ ⊕ AXI4LITE_REGS
 - ▶ ⊕ odi_2_rx
 - ▶ ⊕ odi_4_rx
 - ▶ ⊕ dpu2dpu_1_rx
 - ▶ ⊕ dpu2dpu_2_rx
 - ▶ odi_2_rx_prog_full
 - ▶ odi_2_rx_fifo_depth(9:0)
 - ▶ odi_2_rx_ovf
 - ▶ odi_4_rx_prog_full
 - ▶ odi_4_rx_fifo_depth(9:0)
 - ▶ odi_4_rx_ovf
 - ▶ odi_2_rx_fc_pc_status(14:0)
 - ▶ odi_4_rx_fc_pc_status(14:0)
 - ▶ dpu2dpu_1_rx_overflow
 - ▶ dpu2dpu_1_rx_depth(7:0)
 - ▶ dpu2dpu_2_rx_overflow
 - ▶ dpu2dpu_2_rx_depth(7:0)
 - ▶ sync_pulse
 - ▶ clear_two
 - ▶ clear_four
 - ▶ gp_trig_in(2:0)
 - ▶ S_BSCAN_drck
 - ▶ S_BSCAN_shift
 - ▶ S_BSCAN_tdi
 - ▶ S_BSCAN_update
 - ▶ S_BSCAN_sel
 - ▶ S_BSCAN_tms
 - ▶ S_BSCAN_tck
 - ▶ S_BSCAN_runttest
 - ▶ S_BSCAN_reset
 - ▶ S_BSCAN_capture
 - ▶ S_BSCAN_bscanid_en
- odi_2_tx_ARESETn ▶
 - odi_2_rx_ARESETn ▶
 - odi_4_tx_ARESETn ▶
 - odi_4_rx_ARESETn ▶
 - dpu2dpu_1_tx_ARESETn ▶
 - dpu2dpu_1_rx_ARESETn ▶
 - dpu2dpu_2_tx_ARESETn ▶
 - dpu2dpu_2_rx_ARESETn ▶
 - odi_4_tx ⊕ ▶
 - odi_2_tx ⊕ ▶
 - dpu2dpu_2_tx ⊕ ▶
 - dpu2dpu_1_tx ⊕ ▶
 - odi_2_tx_fc_pc_command(14:0) ▶
 - odi_4_tx_fc_pc_command(14:0) ▶
 - gp_trig_out(2:0) ▶
 - hs_trig_out(39:0) ▶
 - S_BSCAN_tdo ▶

FPGA A Sandbox 1 Configuration B

An IP component is provided with the FSP that replicates the default contents of the partial region as programmed at system startup when the static area and default partial area is programmed in FPGA A for 3 ODI plus 8 10GbE operation.

This IP component appears within the IP Catalog of the Pathwave FPGA window within the BSP group



When dragged into the design area is appears so

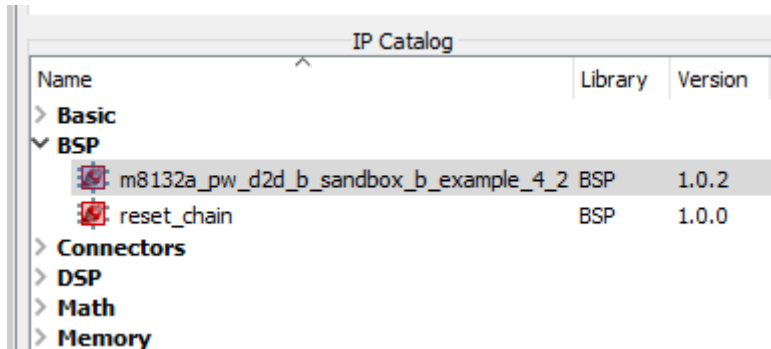
m8132a_pw_d2d_eth_a_sandbox_b_example_4_2_1	
•	Clock_AXI4LITE_REGS
•	AXI4LITE_REGS_aresetn
•	Clock_400M
•	Clock_400_ResetN
•	Clock_500M
•	Clock_500_ResetN
•	Clock_DMA_MM
•	DMA_MM_Write_Read_ResetN
•	Clock_ODI
•	eth_1_tx_ARESETn
•	eth_1_rx_ARESETn
•	eth_2_tx_ARESETn
•	eth_2_rx_ARESETn
•	eth_3_tx_ARESETn
•	eth_3_rx_ARESETn
•	eth_4_tx_ARESETn
•	eth_4_rx_ARESETn
•	eth_5_tx_ARESETn
•	eth_5_rx_ARESETn
•	eth_6_tx_ARESETn
•	eth_6_rx_ARESETn
•	eth_7_tx_ARESETn
•	eth_7_rx_ARESETn
•	eth_8_tx_ARESETn
•	eth_8_rx_ARESETn
•	odi_2_tx_ARESETn
•	odi_2_rx_ARESETn
•	dpu2dpu_1_tx_ARESETn
•	dpu2dpu_1_rx_ARESETn
•	dpu2dpu_2_tx_ARESETn
•	dpu2dpu_2_rx_ARESETn
•	odi_2_tx
•	eth_2_tx
•	eth_3_tx
•	eth_4_tx
•	eth_5_tx
•	eth_6_tx
•	eth_7_tx
•	eth_8_tx
•	eth_1_tx
•	dpu2dpu_2_tx
•	dpu2dpu_1_tx
•	odi_2_tx_fc_pc_command(14:0)
•	gp_trig_out(2:0)
•	hs_trig_out(39:0)
•	S_BSCAN_tdo
•	M_AXI_MM
•	AXI4LITE_REGS
•	odi_2_rx
•	eth_1_rx
•	eth_2_rx
•	eth_3_rx
•	eth_4_rx
•	eth_5_rx
•	eth_6_rx
•	eth_7_rx
•	eth_8_rx
•	dpu2dpu_1_rx
•	dpu2dpu_2_rx
•	odi_2_rx_prog_full
•	odi_2_rx_fifo_depth(9:0)
•	odi_2_rx_ovf
•	odi_2_rx_fc_pc_status(14:0)
•	eth_1_rx_stat(2:0)
•	eth_2_rx_stat(2:0)
•	eth_3_rx_stat(2:0)
•	eth_4_rx_stat(2:0)
•	eth_5_rx_stat(2:0)
•	eth_6_rx_stat(2:0)
•	eth_7_rx_stat(2:0)
•	eth_8_rx_stat(2:0)
•	dpu2dpu_1_rx_overflow
•	dpu2dpu_1_rx_depth(7:0)
•	dpu2dpu_2_rx_overflow
•	dpu2dpu_2_rx_depth(7:0)
•	sync_pulse
•	clear_two
•	gp_trig_in(2:0)
•	S_BSCAN_drck
•	S_BSCAN_shift
•	S_BSCAN_tdi
•	S_BSCAN_update
•	S_BSCAN_sel
•	S_BSCAN_tms
•	S_BSCAN_tck
•	S_BSCAN_runtst
•	S_BSCAN_reset
•	S_BSCAN_capture
•	S_BSCAN_bscanid_en

FPGA B Sandbox 2 Configuration B

An IP component is provided with the FSP that replicates the default contents of the partial region as programmed at system startup when the static area and default partial area is programmed in FPGA B.

The same contents in this FPGA are used for either of the two startup options, 4 ODI or 3 ODI plus eight 10GbE.

This IP component appears within the IP Catalog of the Pathwave FPGA window within the BSP group.



When dragged into the design area it appears so

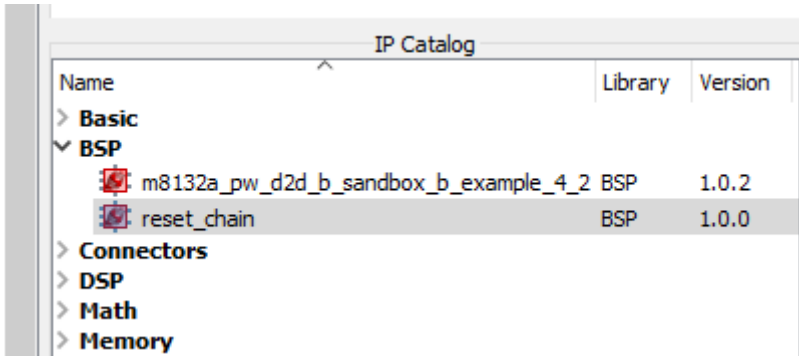
m8132a_pw_d2d_b_sandbox_b_example_4_2_1

- ▶ Clock_AXI4LITE_REGS
- ▶ AXI4LITE_REGS_aresetn
- ▶ Clock_400M
- ▶ Clock_400_ResetN
- ▶ Clock_500M
- ▶ Clock_500_ResetN
- ▶ Clock_ODI
- ▶ Clock_DMA_MM
- ▶ DMA_MM_Write_Read_ResetN
- ▶ + M_AXI_MM
- ▶ + AXI4LITE_REGS
- ▶ + odi_1_rx
- ▶ + odi_3_rx
- ▶ + dpu2dpu_1_rx
- ▶ + dpu2dpu_2_rx
- ▶ odi_1_rx_prog_full
- ▶ odi_1_rx_fifo_depth(9:0)
- ▶ odi_1_rx_ovf
- ▶ odi_3_rx_prog_full
- ▶ odi_3_rx_fifo_depth(9:0)
- ▶ odi_3_rx_ovf
- ▶ odi_1_rx_fc_pc_status(14:0)
- ▶ odi_3_rx_fc_pc_status(14:0)
- ▶ dpu2dpu_1_rx_overflow
- ▶ dpu2dpu_1_rx_depth(7:0)
- ▶ dpu2dpu_2_rx_overflow
- ▶ dpu2dpu_2_rx_depth(7:0)
- ▶ sync_pulse
- ▶ clear_one
- ▶ clear_three
- ▶ gp_trig_in(2:0)
- ▶ S_BSCAN_drck
- ▶ S_BSCAN_shift
- ▶ S_BSCAN_tdi
- ▶ S_BSCAN_update
- ▶ S_BSCAN_sel
- ▶ S_BSCAN_tms
- ▶ S_BSCAN_tck
- ▶ S_BSCAN_runttest
- ▶ S_BSCAN_reset
- ▶ S_BSCAN_capture
- ▶ S_BSCAN_bscanid_en
- odi_1_tx_ARESETn ▶
- odi_1_rx_ARESETn ▶
- odi_3_tx_ARESETn ▶
- odi_3_rx_ARESETn ▶
- dpu2dpu_1_tx_ARESETn ▶
- dpu2dpu_1_rx_ARESETn ▶
- dpu2dpu_2_tx_ARESETn ▶
- dpu2dpu_2_rx_ARESETn ▶
- odi_3_tx + ▶
- odi_1_tx + ▶
- dpu2dpu_2_tx + ▶
- dpu2dpu_1_tx + ▶
- odi_1_tx_fc_pc_command(14:0) ▶
- odi_3_tx_fc_pc_command(14:0) ▶
- gp_trig_out(2:0) ▶
- hs_trig_out(39:0) ▶
- S_BSCAN_tdo ▶

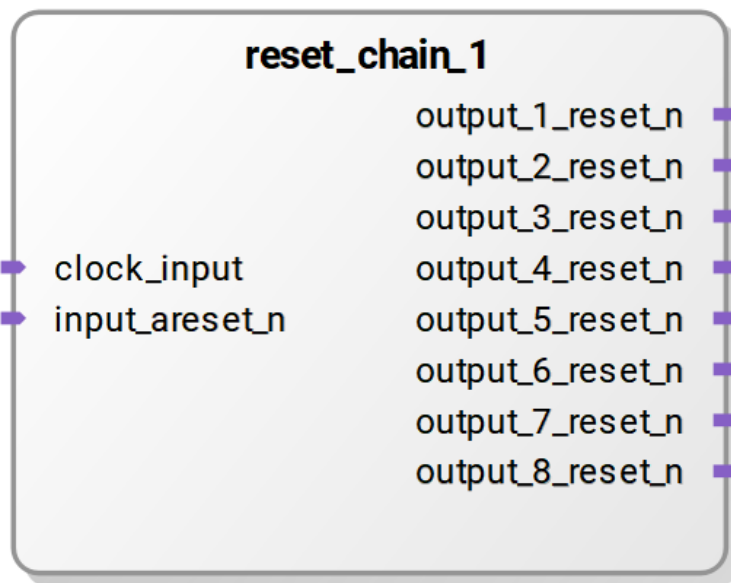
Common

reset_chain

With release 4.2, a new ip component has been introduced that will appear in the IP Catalog BSP entry for all three configurations.



When dragged into the design area it appears so



This component can be used to help insert flip flop pipeline stages to reset signals. One issue with the large size of the Customer Accessible Area partial reconfiguration area, is that it spans across 3 Super Logic Regions (SLR).

To help a design meet timing during the implementation stage, this component will create a chain of flip flops that the placement algorithms can use to route reset signals that have to reach long distances within the partial region.

The source code has suitable attributes to prevent shift register extraction to preserve the individual flip flops.

Also, the input is treated as an asynchronous input with appropriate synchronization, so it could be used to create a startup reset synchronous to Clock_ODI by feeding the ip component input "input_areset_n" to the AXI4LITE_REGS_ResetN signal, and connecting the ip component input "clock_input" to the Clock_ODI signal.

Each of the subsequent "output_X_reset_n" signal will transition 2 periods of the clock_input signal after the previous "output_X_reset_n" signals.

"output_1_reset_n" will transition between 3 and 4 clock periods after the "input_areset_n" signal changes depending on the relationship between the reset signal transition arrival and the clock.

Example Designs

Six templates are provided within the M8132A FSP for M8131A/M8132A Release 4.2

There are a pair of templates for each of the three sandboxes, FPGA A with sandbox 1 for Configuration A, FPGA A with sandbox 1 for Configuration B, and FPGA B with sandbox 2 for Configuration B.

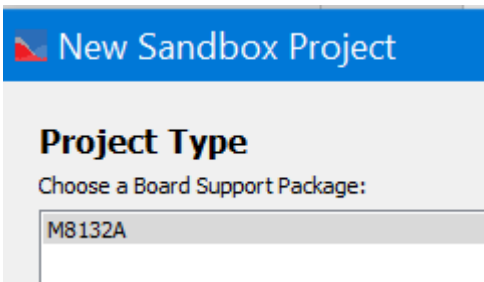
The table below lists which templates apply to which configurations

Configuration	Template	Notes
M8132A_A_Sandbox_A	M8132A_FPGA_A_sandbox_a_template_design_4_2	Uses the appropriate IP component to provide the same functionality as the default startup bit file for FPGA A in 4 ODI operation mode
	M8132A_FPGA_A_register_example_4_2	Simple register example using only general purpose pathwave fpga components
M8132A_A_Sandbox_B	M8132A_FPGA_A_sandbox_b_template_design_4_2	Uses the appropriate IP component to provide the same functionality as the default startup bit file for FPGA A in 3 ODI plus eight 10GbE operation mode
	M8132A_FPGA_A_config_b_trigger_example_4_2	Simple register example using general purpose pathwave fpga components to drive the HS_Trigger_OUT interface which can be connected to the M8132A front panel SMA connector "Trig Out"
M8132A_B_Sandbox_B	M8132A_FPGA_B_sandbox_b_template_design_4_2	Uses the appropriate IP component to provide the same functionality as the default

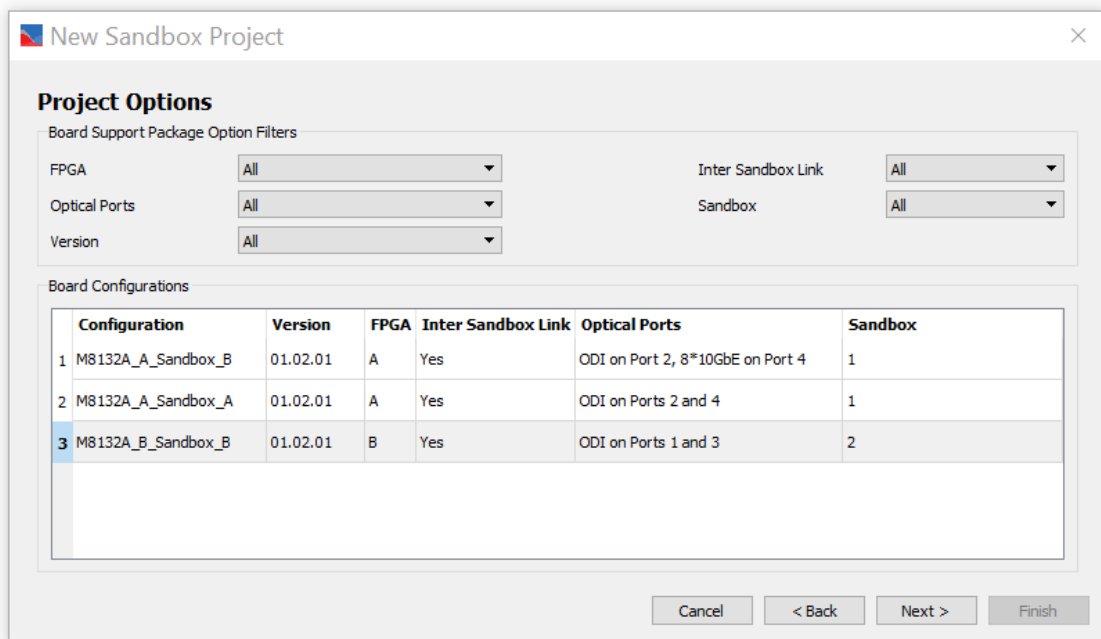
Configuration	Template	Notes
		startup bit file for FPGA B in either operation modes
	M8132A_FPGA_B_register_example_4_2	Simple register example using only general purpose pathwave fpga components

These templates can be loaded when a new design is started within the Pathwave FPGA tool when the M8132A BSP is selected during the new sandbox project dialog.

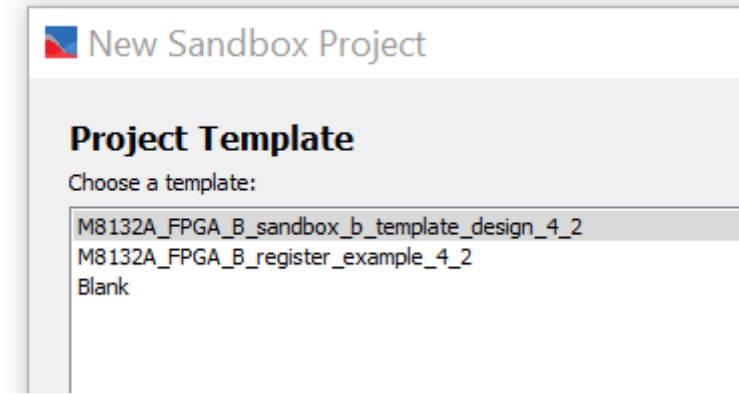
First select the M8132A BSP



Then, for say FPGA B, Sandbox 2, select the entry in the Board Configuration table relating to FPGA B Sandbox 2



And then, at the template selection stage, chose the M8132A_FPGA_B_sandbox_a_template_design_4_2, or the M8132A_FPGA_B_register_example_4_2



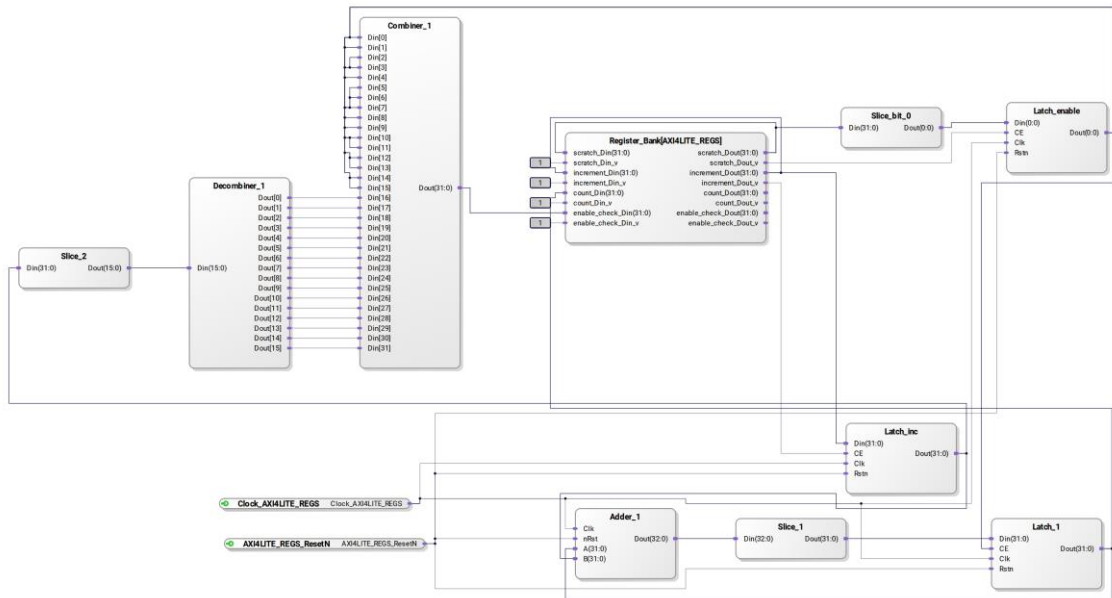
This will load an example project, using the appropriate IP component described above, with all the necessary connections to the top level IO of the sandbox already made.

This design can be built by running the Pathwave FPGA build process as normal

Here is a screen grab of a project created by loading the template sandbox example design for FPGA B, Sandbox 2, M8132A_FPGA_B_sandbox_b_template_design_4_2



Here is a screenshot of a project created by loading the template register example design for FPGA B, Sandbox 2, M8132A_FPGA_B_register_example_4_2



M8132A Template Example Design Description

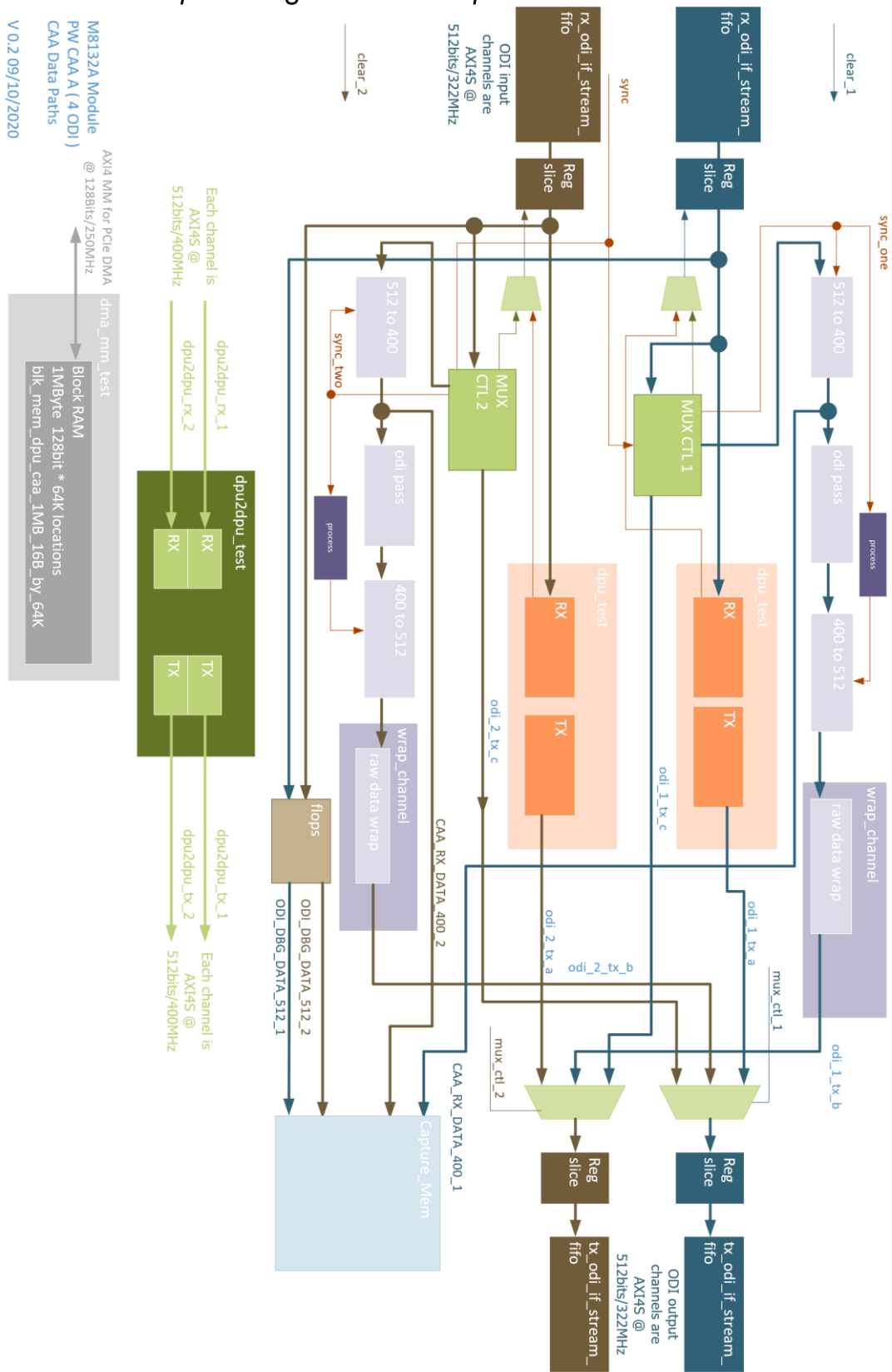
The example full sandbox content designs for the M8132A Pathwave FPGA sandboxes have some of the same functionality and some different functionality. One is specifically for use with sandbox 1 in FPGA A for the 4 ODI operation mode. One is specifically for use with sandbox 1 in FPGA A for the 3 ODI plus 8 10GbE operation mode, and the other is specifically for use with sandbox 2 in FPGA B for either operation modes. This chapter describes the basic functionality of the logic, and how to use it. Example C++ programs for this design are available via the installer for the M8131A/M8132A SFP.

The following paragraphs of this chapter consist of an overview of the main data flow paths of the example design, descriptions of the main components, and then memory maps and register descriptions, with procedures for setup and operation to help explain the details.

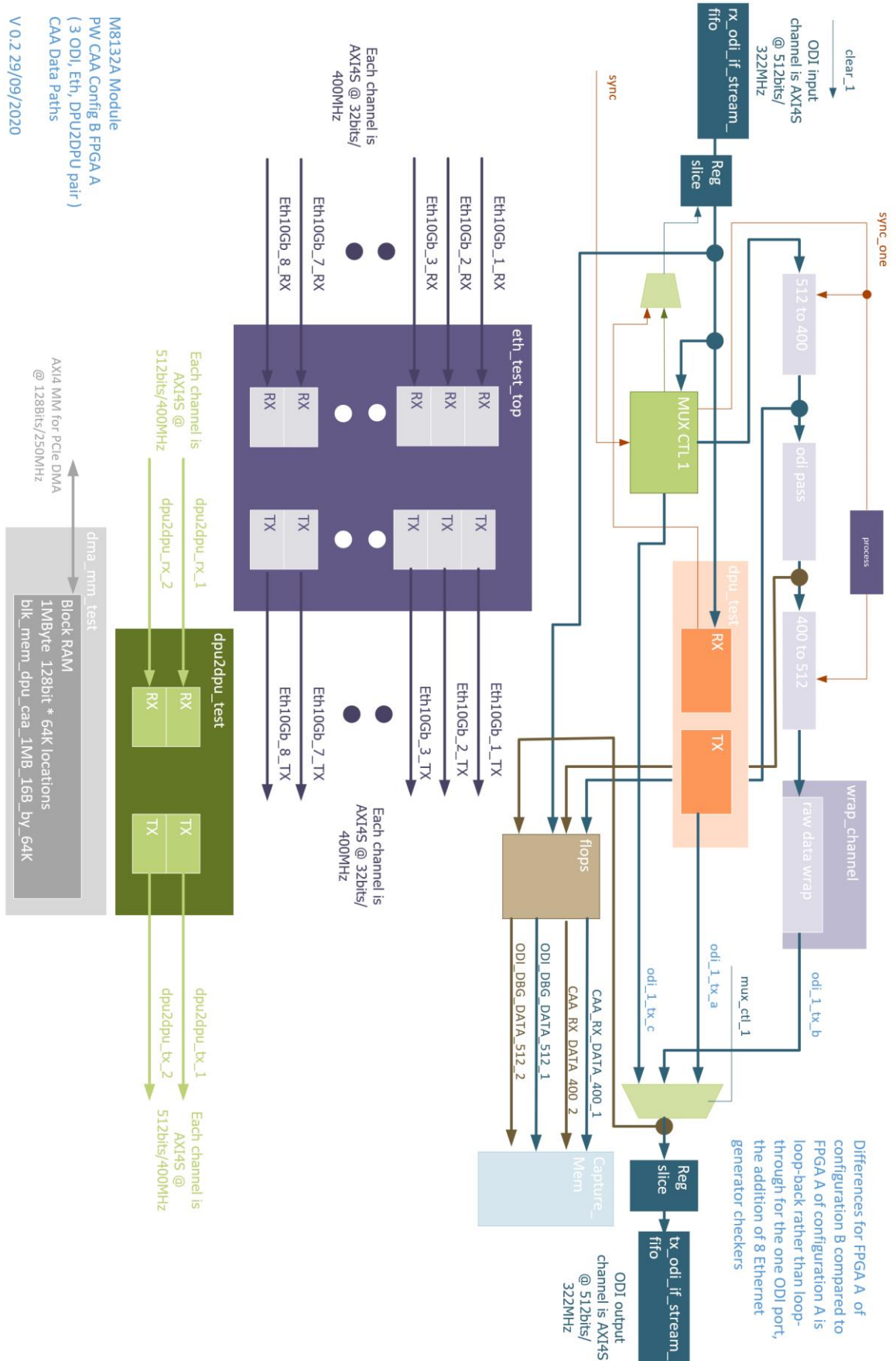
General

The three diagrams below give simplified views of the main data flow paths within the example designs. In the following paragraphs, the component names are mentioned in brackets, e.g. (Capture_Mem)

FPGA A example design for 4 ODI operation



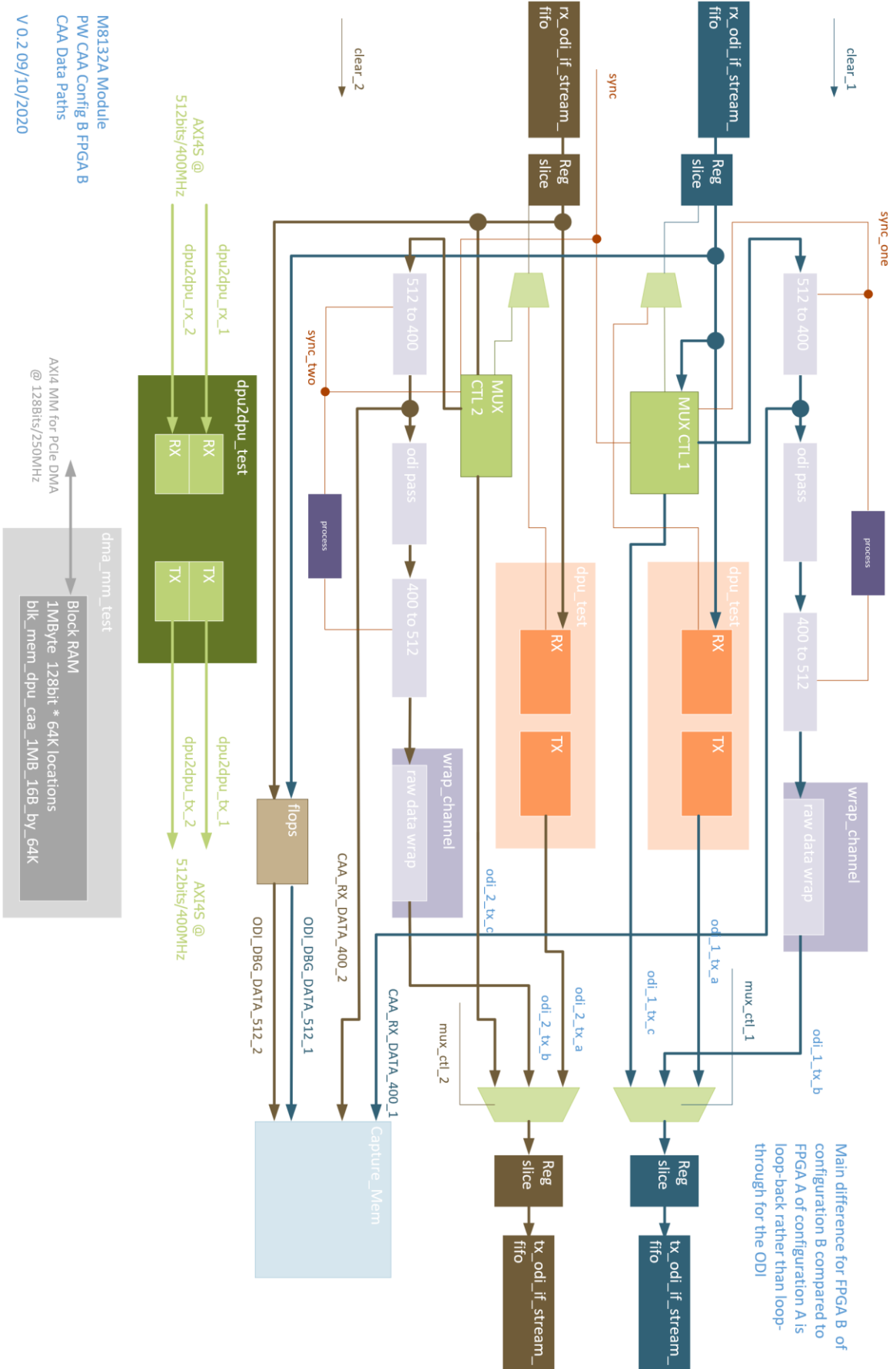
FPGA A example design for 3 ODI plus 8 10GbE operation



Differences for FPGA A of configuration B compared to FPGA A of configuration A is loop-back rather than loop-through for the one ODI port, the addition of 8 Ethernet generator checkers

M8132A Module
PW_CAA Config B FPGA A
(3 ODI, Eth, DPUDPU pair)
CAA Data Paths
V 0.2.29/09/2020

FPGA B example design for either modes of operation



ODI

General

Each sandbox design has either one or two test payload generators and checkers, one or two 512bit pass-through or loop-back paths with various modes of operation, and one or two 400bit pass-through or loop-back paths, with various modes of operation.

The checkers and generators and pass-through and loop-back components are setup to handle the ODI connections. The transmit and receive test pattern generator/checkers work with one port's transmit receive pair. The pass-through components work with the receiver of one port, and the transmitter of the other port. The loop-back components work with one port's transmit receive pair.

FPGA A, Sandbox 1 is ultimately connected to the front panel ODI ports 2 and 4 (port 4 is only an ODI port when the 4 ODI startup option is selected. If 3 ODI plus 10GbE operation is selected, then front panel optical port 4 is not connected to ODI)

FPGA B, Sandbox 2 is ultimately connected to the front panel ODI ports 1 and 3

Logic within the static regions of FPGA A and FPGA B hidden from the sandbox user, handle conversion of the AXI4S streaming interfaces to and from the ODI/Interlaken cores for transmission and reception via high speed transceivers within the FPGA

Basic Operation

The AXI4S ODI streams arrive and leave the sandbox synchronous to the 332MHz ODI clock

A Note on ODI over AXI4S

The ODI AXI4S interfaces are basically standard AXI4S signals with a few restrictions and alterations. User logic should obey tready and tvalid as normal. tlast indicates the last transfer of a packet is contained within this current transfer.

The only valid values for tstrb are 0xFFFFFFFFFFFFFFFF or 0xFFFFFFFF00000000 since valid ODI packets should be some multiple of 32bytes, starting at 64bytes as a minimum.

ODI is an earliest sample at the top protocol, hence when a packet ends on an odd multiple of 32bytes long, it is the upper 32bytes that are valid.

New packets must start at the top of the bus, and since the packet length must be greater than or equal to 64 bytes, the first tstrb value of a new packet must be 0xFFFFFFFFFFFFFFFF

The tables below give some examples

a 128byte packet

Data(511 downto 256)	Data(255 downto 0)	tstrb	tlast
first 32 bytes of data	32 bytes of data	0xFFFFFFFFFFFFFFFF	0
32 bytes of data	last 32 bytes of data	0xFFFFFFFFFFFFFFFF	1

a 192byte packet

Data(511 downto 256)	Data(255 downto 0)	tstrb	tlast
32 bytes of data first	32 bytes of data	0xFFFFFFFFFFFFFFFF	0
32 bytes of data	32 bytes of data	0xFFFFFFFFFFFFFFFF	0
32 bytes of data last	32 bytes of don't care	0xFFFFFFFF00000000	1

The M8132A ODI interfaces in the static region have no knowledge of the data format for either transmit or receive. It is up to the user of the customer accessible area to provide the processing necessary to create or decode data mapped to whatever format is being delivered on an input receive port, or expected on an output transmit port.

A Note on the 400bit bus operation

The 400MHz with 400Bit bus within the example design is a 40 times super sampled 10bit per sample data stream.

The lowest 10bits on the bus (9:0) are the earliest samples in time order, the highest ten bits, (399:390) are the latest samples in time order

The 4 ODI FPGA A Example Design ODI features

Slightly customized, and wrapped with further logic, AXI4S streaming fifos; (rx_odi_if_stream_fifo), are used to pass the data from the 332MHz clock to the 400MHz clock, the main processing clock in this example design for receive paths. Further slightly differently customized and wrapped AXI4S streaming fifos; (tx_odi_if_stream_fifo), are used to pass data from the 400MHz main processing clock to the 332MHz clock for transmit paths.

AXI4S register slice IP components; (Reg slice), are used to pipeline the data in the 400MHz domain to aid timing in place and route.

The DPU test components; (dpu_test), contain a transmit and receive pair of test pattern generator and checker for driving ODI data to and from a port. The generators allow some control over the size of ODI packets and the gap between each new packet, and hence the average data rate. At this time only RAW ODI packets can be generated or checked i.e. this design has no processing for Vita 49 packets.

The mux control blocks; (MUX CTL 1/2), do, as the name suggests, control the input and output multiplexors, to select the sink and source for the receive and transmit ODI streams. They are also involved in handling the conversion from AXI4S receive streams to simple data and data valid for the gearboxes. As the names suggest the 512bit passthrough mode retains the ODI data as it arrived which for certain data formats, may directly contain the original source data. These components also handle the 512 to 400 bit gearboxes; (512 to 400), and recover the base 512 data from the AXI4S stream to feed to the gearbox as required.

These gearboxes (512 to 400) are designed to recover the original samples from the M8131A Digitizer when it operates in 16GSa/s direct mode with no decimation. They recover the 400bit bus, with 40 times super-sampled 10 bit data at 400MHz. The gearboxes correctly regenerate the earliest 10 bit sample at the bottom format for the 400bit bus from the earliest sample at the top 512 bit bus assumed to have arrived over an ODI transport link. For correct operation, the input reference clock of the M8132A DSP module should be connected to the output reference clock of the M8131A digitizer, and the reference clock source set to external via the Soft Front Panel, or via SCPI, so that the 400MHz clock in the DSP module is synchronized to the 400MHz clock in the digitizer module. This prevents differences in the clock rates from causing FIFO overflows or underflows.

When operating in 400 bit pass through mode, the input and output gearboxes require synchronization. The two small components shown; (process), connected to the sync_one and sync_two signals contain some sequential statements to handle the sync signals, and some pipelining of the data and data valid signals required to keep the 512 to 400 and 400 to 512 gearboxes aligned.

In release 4.2 the components used for the 512 to 400 gearbox (gb_512_to_400.vhd) for the example template designs, (and within the static with default CAA contents designs loaded by software at SFP startup), have been replaced by gb_512_to_400_b.vhd. These perform the same function as before, but with better routability, at the cost of 1 extra clock period of delay through the component. (the previous component, gb_512_to_400.vhd is still available)

The component between the two gearboxes; (odi_pass), is the location that any customer 40 times super-sampled 10bit per sample at 400MHz processing could be performed. At the 400MHz 400bit processing, the data valid is false before any data arrives and the then goes true, and stays true.

In the supplied example design, *odi_pass* simply contains a register stage on the input, and a register stage on the output, for the data, and the data valid. If the customer wishes to try some signal processing, the only requirement is that the output is still 40 times super-sampled, and 10 bits per sample, and the data valid delay matches the data delay.

To prepare the 400bit data and clock enable signal for transmission two components are required. A gearbox; (400 to 512), turns the continuous clock enable 400bits at 400MHz, into a gapped clock enabled 512bit data and clock enable. Again, this gearbox correctly translates the earliest sample at the bottom 10bit per sample bus to the earliest sample at the top 512 bit bus suitable for ODI. This then is processed in the *wrap_channel* component where the necessary additional signals are added to create an AXI4S ODI stream interface.

Various parameters are set in the *wrap_channel* component, including the ODI/Interlaken channel, and the desired packet size.

After the output multiplexors select the desired source of data for driving an interface, a pipeline stage; (Reg Slice), helps the design meet place and route timing requirements, and the a stream fifo; (tx_odi_if_stream_fifo), passes the AXI4S ODI streaming signal from the 400MHz clock to the ODI 332MHz clock.

Four of the datapaths; one for each channel, are routed to a storage component; (Capture_Mem) which can be used to capture up to around 1 Mbytes worth of storage, of either 10bit samples from the 400bit bus; (CAA_RX_DATA_400_1/2), or 16bit samples from the 512bit bus; (ODI_DBG_DATA_512_1/2). The maximum number of samples depends on the capture mode.

Detailed operation of the various components is discussed in the register description sections of each module below, along with explanation of the operation of the *clear_1/2* and *sync* signals.

The 3 ODI FPGA A Example Design features

In terms of ODI operation, the 3 ODI FPGA A design is a single channel version of the 2 channel 4 ODI FPGA A Example Design, with a couple of other differences. The main difference is in the connections of the 512 and 400 bit paths. In this case since there is only one transmit/receive pair, the passthrough has been changed into a loop back. Further, since there is only channel, the 2nd pair of capture inputs fed to the (Capture_Mem) component are connected to the 400 bit path at the output of the (odi_pass) component and the final 512bit path after the output multiplexor. This would allow a user to insert 400 bit processing logic in the (odi_pass) component and capture both the input and output data for comparison. Likewise, the input 512 data, and the output 512 data could both be captured for comparison

The FPGA B Example Design features

In terms of ODI operation, the FPGA B Example design is almost the same as the 4 ODI FPGA A Example Design described above. The main difference is the connection of the 512 and 400 bit paths. In this case, a loop-back arrangement is setup, rather than a pass-through, just by virtue of which data paths are connected to which output multiplexors, all other aspects of ODI operation remain the same

Source Code

The source code for the example design, in particular the IP component used within it, is available within the BSP install directory of the M8132A, i.e., within C:\Program Files\Keysight\M8131\FSP is an ip directory, with sub folders

ip → common

ip → common → vhdl

ip → common → xci

ip → fpga_a_sandbox_1

ip → fpga_a_config_b_sandbox_1

ip → fpga_b_config_b_sandbox_2

ip → test_benches

The source vhdl files common between the example for fpga a and fpga b are within the vhdl folder

The Xilinx Vivado XCI files used for Xilinx IP components are within the xci folder

The top level vhdl for fpga a sandbox 1 for 4 ODI operation is within the fpga_a_sandbox_1 folder

The top level vhdl for fpga a sandbox 1 for 3 ODI + 8 10GbE operation is within the fpga_a_config_b_sandbox_1 folder

the top level vhdl for fpga b sandbox 2 is within the fpga_b_config_b_sandbox_2 folder

some example testbenches (described later) are within the test_benches folder

As of release 4.2 there is now a further folder within the common folder

ip → common → reset_chain

This contains the source code for the reset_chain component that appears in the BSP sub folder of the IP Catalog

Memory Map



M8132A front panel

The M8132A has four ODI optical ports numbered ODI 1, ODI 2, ODI 3, ODI 4.

FPGA A sandbox 1 is connected to ports 2 and 4

FPGA B sandbox 2 is connected to ports 1 and 3

The address maps of the two sandbox partial region example designs are essentially the same from an ODI point of view.

The one difference is that the FPGA A example design for 3 ODI plus Ethernet only has the first of the pairs of register sets relating to ODI IF 2.

Addresses in the detailed register description tables further below are given as individual offsets from the base address of each group given in the table directly below, and repeated at each table

ODI IF	Component	Base Address
1:2	ODI Test Ct/Stat	0x0000
3:4	ODI Test Ct/Stat	0x0100
1:2	ODI Pass Through Ct/Stat	0x0300
3:4	ODI Pass Through Ct/Stat	0x0400
1:2	ODI Mux control Ct/Stat	0x0500
3:4	OD Mux Control Ct/Stat	0x0600
1,3:2,4	Capture Memory	0x0700

ODI Test Control & Status

These registers control and monitor test payload generation and checking and there is a set in each of the *dpu_test* components

Port 1 (2) Register Base : 0x000000

Port 3 (4) Register Base : 0x000100

Address Offset	Type	Description
0x00	RW	Interlaken Channel
0x04	RW	TX Packet Length
0x08	RW	unused
0x0C	RW	TX Packet Start Gap
0x10	RW	TX Packet Count
0x14	RW	TX Control
0x18	RO	TX Status
0x1C	RO	Unused
0x20	RO	Unused
0x24	RO	Unused
0x28	RO	Unused
0x2C	RW	Expected Interlaken Channel
0x30	RW	RX Mark Space Ratio
0x34	RW	Unused
0x38	RO	RX Packet Count
0x3C	RW	RX Control
0x40	RO	RX Status
0x44	RO	RX Active Clocks Lower
0x48	RO	RX Active Clocks Upper
0x4C	RO	RX Total Clocks Lower
0x50	RO	RX Total Clocks Upper
0x54	RO	Segment Errors
0x58	RO	Core clock Rate
0x5C	RO	TX clock crossing fifo depth
0x60	RO	TX clock crossing fifo depth max
0x64	RO	RX clock crossing fifo depth
0x68	RO	RX clock crossing fifo depth max
0x6C	RO	RX flow control fifo depth
0x70	RO	RX flow control fifo depth max

ODI Mux Control Ctl/Stat

These registers control the multiplexing of input and output ODI signals and monitor the statistics/status for the direct 512 bit pass through and the 512 to 400 input gearbox. See the individual register descriptions for more details. There is a set in each of the *MUX_CTL 1* and *MUX_CTL 2* components. There are two copies of the logic; the set for Port 1(2), control the pass through of the data received in ODI port 1(2), and control the choice of the port 1(2) transmit output. The set for Port 3(4), control the pass through of the data received in ODI port 3(4), and control the choice of the port 3(4) transmit output. The port 1(3) receive pass through data is routed to the port 3(1) transmit output, likewise, the port 2(4) receive pass through data is routed to the port 4(2) transmit output.

Port 1 (2) Register Base : 0x000500

Port 3 (4) Register Base : 0x000600

Address Offset	Type	Description
0x00	RW	DPU MUX Select
0x04	RW	DPU MUX Control
0x08	RO	DPU MUX Status
0x0C	RO	DPU MUX DATA COUNT_LOWER
0x10	RO	DPU_MUX_DATA_COUNT_UPPER
0x14	RO	DPU_MUX_TX_FIFO_DEPTH
0x18	RO	DPU_MUX_RX_FIFO_DEPTH
0x1C	RW	DPU_MUX_SPARE
0x20	RW	DPU_MUX_STARTUP_DEPTH
0x24	RW	DPU_MUX_EMPTY_DEPTH
0x28	RW	DPU_MUX_ODI_PACKET_LENGTH

ODI Pass Through Ctl/Stat

These registers control/monitor the output stages of the 400bit pass through path preparing the data for the transmit ODI interfaces within the sandbox, and there is one set in each of the *wrap_channel* components.

Port 1 (2) Register Base : 0x000300

Port 3 (4) Register Base : 0x000400

Address Offset	Type	Description
0x00	RW	DPU_BB_ODI_CHANNEL
0x04	RW	DPU_BB_ODI_PACKET_LENGTH
0x08	RW	DPU_BB_CONTROL
0x0C	RO	DPU_BB_STATUS
0x10	RO	DPU_BB_DATA_COUNT
0x14	RO	UNUSED
0x18	RO	DPU_BB_CORE_CLOCK_RATE

Capture Memory

Capture Memory Registers Base : x100700

The capture system can capture from both channels simultaneously . For both channels only one of two paths can be chosen.

Captures can be made on either the 512 passthrough data path, or the 400 bit passthrough data path

Address Offset	Type	Name	Description
0x00	RW	CAPTURE_CONTROL	Capture control register
0x04	RW	CAPTURE_DATA_VECTS	Number of data vectors to capture
0x08	RW	READ_REQ_ADDR	Read request address
0x0C	RW	READ_REQ_SAMPLES	Number of samples to read
0x10	RW	READ_REQ_CTRL	Read request control
0x14	RO	READ_DATA	Read requested data
0x18	RW	ABORT	Abort read command
0x1C	RO	CAPTURE_MEM_STATUS	Capture status
0x20	RW	DDC_ACCESS_CTRL	Data from 400bit Interface or 512bit interface
0x30:6C	RO	READ_DATA	Burst Read requested data for Wishbone Interface

Register Descriptions

Test Packet Generation Registers

Duplicated for each ODI Port, 2 per FPGA/Sandbox where appropriate, 1 for the FPGA A Configuration B

Interlaken Channel

Bits:	Function	Description:
3:0	Interlaken TX Channel	Set before a "run" Determines which Interlaken channel will be used Default 0x0 Top bits fed to core are forced to 0x0

Only the bottom 4 bits can be set.

TX Packet Length

Bits:	Function	Description:
19:0	TX Packet Length	Set before a "run" Units of 32 bytes Determines length of packets generated in a run Default value 0x00008 , i.e. 256 bytes

TX Packet Start Gap

Bits:	Function	Description:
31:0	TX Packet Start GAP	Set before a “run” Units of 64 bytes Determines gap between successive start-of-packets. Must be larger than TX Packet Length * 2 Default value 0x0800 , i.e. 2048d, ie. 400Mb/s data rate if the default packet size of 256 bytes is used

The traffic generator runs at 400MHz, and has a data width of 512 bits. This is higher than the possible ~160Gb/s total ODI rate for 14.1G and a 2048 burst size. The combination of packet size and packet size gap should be set to achieve the desired average rate. Setting the rate higher will not have any adverse effects, but the achieved rate will be limited by the capacity of the link. A warning flag is set in the transmit status, “overflow”, to warn that this situation has occurred.

The generator can produce a new 512bit word every 400MHz clock tick

$$512 \text{ bits} * 400 \text{ MHz} = 204.8 \text{ Gb/s}$$

However, this is modified by the ratio of the packet size and the packet size gap

E.G. 1.

If a packet size of 128 is used, and a packet start gap of 250.

The units of packet size are 32bytes

The units of packet size gap are the 400MHz clock ticks

$$400\text{E}6 * (128 * 32 * 8 / 250) = 52.4288 \text{ Gb/s average rate}$$

E.G.2.

If a packet size of 1000 is used, and a packet start gap of 640 .

The units of packet size are 32 bytes

The units of packet start gap are the 400MHz clock ticks

$$400\text{E}6 * (1000 * 32 * 8 / 640) = 160 \text{ Gb/s}$$

TX Packet Count

Bits:	Function	Description:
31:0	TX Packet Count	Set before a “run” Determines how many packets are sent in a run, a 0 implies run until a stop action bit is set (see below) When the internal count reaches 0 and the current packet is complete, the transmitter stops generating packets

TX Control

Bits:	Function	Description:
0	Reset	Reset, i.e. put all registers/logic counters into known state for this data generator. Default/Reset value ‘1’
1	Stop	Stop, i.e. halt transmission of packets after any current packet in progress completes Used to halt continuous mode, or to stop counted mode before the count is complete.
2	Start	Start, i.e. begin packet transmission
3	Mode	0 = traditional M8132A test pattern 1 = Simple counting test pattern
4	Errin	0 = normal 1 = Error Sets the TUSER(1) bit to test the errin and errout paths
5	Reset_clock_cross_max	0 = normal 1 = reset the fifo depth max detector for the tx clock crossing fifo

TX Status

Bits:	Function	Description:
0	Running	If read as a 1, a tx run is in progress, if read as a 0, either the run has not started, or a started counted packet run has finished
1	Overflow	Latched status bit, indicates that a packet not was complete before the next packet was due to start, either the packet count and packet length and packet start gap are incorrectly programmed, or the back pressure exerted by the receiver of this tx, is set too low for the current “tx rate”

Transmit General

After startup, the transmitter is held in reset. This reset propagates to the ODI transmit reset signal to apply a reset to the AXI4S interface of the ODI transmitter logic within the static area.

(note this does not reset the core logic, just the AXI4S interface logic)

Writing a 0 to bit 0 (Reset) of the TX Control register will remove the reset.

Setup the desired packet size, and packet start gap, and then when the receiver is ready, and the appropriate setting have been made in the ODI Mux Control area, a 0 to 1 transition on the “Start” bit (bit 2) of the TX Control register will start the transmitter generating packets.

Expected Interlaken Channel

Bits:	Function	Description:
3:0	Interlaken RX Channel Expected	Set before a “run” Determines which Interlaken/ODI channel is the expected one, compared with the channel reported by the Interlaken core, a latched “history bit” per segment is reported, see below Default value 0x0, top bits for check are forced to 0x0

RX Mark Space Ratio

Bits:	Function	Description:
31:0	RX Mark Space Ratio	Apply a maximum limiting mark space ratio to the data draw from the receive side during reception Default 0xffffffff run as fast as possible Used to load a 32 bit circulating shift register, just program the mark space ratio (pattern) you want, when a run starts, the shift register is loaded, the register is clocked each possible transfer time, and data only received if the bottom bit is a 1 i.e. for 1:1 mk:sp program with 0xAAAAAAAA i.e. for 3:1 mk:sp program with 0xEEEEEEEE i.e. to read the effective rate for 16GS @ 10bit, try 0xF7777777 i.e. to read the effective rate of 16GS @ 10bit decimate by 2, then 10bit to 16 converted, i.e 320/512 , try 0xDADADADA i.e. to read the effective rate of 16GS : 10bit, decimate by 4, then 10bit to 16 converted, i.e. 160/512, try 0x92489248 The effect of this will be to increase the fill depth of the fifo between the Interlaken core receive “output” ports and the data checker. If the fifo goes above the treshold level, and flow control is enabled for the particular ODI link, then the flow control XON will be changed to XOFF for the link

RX Packet Count

Bits:	Function	Description:
31:0	RX Packet Count Actual	Reset before a “run” Count of how many end-of-packet control signals have been seen since the last “reset”

RX Control

Bits:	Function	Description:
0	Reset	Reset, i.e. put all registers/logic/counters/alarm status in to known state. Default/Powerup value '1'
1	Stop	Stop, i.e. halt counting packets or errors after any current packet in progress completes
2	Start	Start, i.e. begin counting packets or errors
3	Hold Register Value	Set to a '1' to halt updating of the registers from the counters. i.e. the counters still run, so nothing is "lost" but the registers will not be updated while this bit is set to a "1". This allows a coherent set of values to be read, in particular the, the 64 bit counts will not have the risk of being read splitting a lower ½ roll over and giving strange results. Put back to a '0' to allow normal updates. Defaults to "0"
4	Wait for Sync	Defaults to 0 If set to 1, the RX start doesn't take effect until the sync pulse has been seen
5	Mode	0 = traditional M8132A test pattern 1 = simple counting test pattern
6	Reset_clock_cross_max	0 = normal 1 = reset the rx clock crossing fifo max detector
7	Reset_flow_fifo_max	0 = normal 1 = reset the max detector for the flow control fifo depth

RX Status

Bits:	Function	Description:
0	Running	Run is in progress
1	Wrong Interlaken Channel latched	Latched alarm bits where the "Expected Interlaken Channel" register did not match with the channel presented by the core
2	Fifo overflow latched	Latched alarm bit reporting the fifo the test receiver listens to has reported overflow
3	Errout	0 = Normal 1 = Error Reports the current state of the errout / TUSER(1) received

Receive General

After startup, the receiver is held in reset. This reset propagates to the ODI receive reset signal to apply a reset to the AXI4S interface of the ODI receiver logic within the static area.

(note this does not reset the core logic, just the AXI4S interface logic)

Writing a 0 to bit 0 (Reset) of the RX Control register will remove the reset.

Setup the expected ODI channel, the desired mark space ratio, and then, when appropriate settings have been made in the ODI Mux Control area, a 0 to 1 transition on the “Start” bit (bit 2) of the RX Control register will start the receiver monitoring for incoming packets.

RX Active Clocks Lower

Bits:	Function	Description:
31:0	RX Active Clocks lower	Number of 400MHz domain clock cycles when data was transferred between the start and stop, lower 32 of a 64 bit count. Activated by start, stopped by stop

RX Active Clocks Upper

Bits:	Function	Description:
31:0	RX Active Clocks upper	Number of 400MHz domain clock cycles when data was transferred between the start and stop, upper 32 of a 64 bit count. Activated by start, stopped by stop

RX Total Clocks Lower

Bits:	Function	Description:
31:0	RX Total Clocks lower	Total number of 400MHz domain clock cycles between the start and stop. Lower 32 of a 64 bit count

RX Total Clocks Upper

Bits:	Function	Description:
31:0	RX Total Clocks upper	Total number of 400MHz domain clock cycles between the start and stop, upper 32 of a 64 bit count

Segment Errors

Bits:	Function	Description:
3:0	Fixed bits error latched	One bit for each segment, set true if an error occurs in any of the fixed bits for a particular segment
15:8	Fixed bits count	8 bit rolling count of number of fixed bit errors across all segments, 1 count means one 64 byte transfer had one or more errors
19:16	Count bits error latch	One bit for each segment, set true if an error occurs in any of the counter bits for a particular segment
31:24	Count bit count	8 bit rolling count of number of counter bit errors across all segments, 1 count means one 64 byte transfer had one or more errors

Core Clock Rate

Bits:	Function	Description:
8:0	stat_clock_rate	Reports the clock rate of the core clock in MHz, as measured using a 1us repetitive pulse generated from the 150MHz wishbone clock, just a rough measure, should report either 399, 400, or 401

TX Clock Crossing fifo depth

Bits:	Function	Description:
15:0	Tx_clock_fifo_depth	Reports the current depth of the AXI4S fifo for the transmit direction

TX Clock Crossing fifo depth max

Bits:	Function	Description:
15:0	Tx_clock_fifo_depth_max	Reports the maximum depth of the AXI4S fifo for the transmit direction since the last operation of the Reset_clock_cross_max bit in the TX Control Register

RX Clock Crossing fifo depth

Bits:	Function	Description:
15:0	Rx_clock_fifo_depth	Reports the current depth of the AXI4S fifo for the receive direction

RX Clock Crossing fifo depth max

Bits:	Function	Description:
15:0	Rx_clock_fifo_depth_max	Reports the maximum depth of the AXI4S fifo for the receive direction since the last operation of the Reset_clock_cross_max bit in the RX Control Register

RX Flow Control fifo depth

Bits:	Function	Description:
15:0	Rx_flow_fifo_depth	Reports the current depth of the rx flow control fifo

RX Flow Control fifo depth max

Bits:	Function	Description:
15:0	Rx_flow_fifo_depth_max	Reports the maximum depth of the rx flow control fifo since the last operation of the Reset_flow_fifo_max bit in the RX Control Register

ODI Pass Through Ctl/Stat

Duplicated for each ODI Port, 2 per DPU, 4 per M8132A in the case of 4 ODI startup operation

FPGA A will only have 1 if the 3 ODI plus 10GbE mode is desired

General

As can be seen in the simplified data flow diagram above, this component takes the output 512bit data from the 400 to 512 gearbox. The gearbox will take 400bit @ 400MHz continuous clock enable data and produce 512 bit @ 400MHz data with a gapped clock enable appropriate to it's operation.

This component also changes the earliest sample at the bottom format of the 400 bit bus into an earliest sample at the top 512 bus suitable for ODI transport.

This component then turns that data and clock enable into an AXI4S stream suitable for driving to a transmit ODI Stream fifo, which in turn could be connected to the ODI interface of the sandbox, for passing to the ODI core logic of the static area.

DPU_BB_ODI_CHANNEL

Bits:	Function	Description:
7:0	channel	Sets the ODI channel to use on the tx ODI interface

DPU_BB_ODI_PACKET_LENGTH

Bits:	Function	Description:
16:0	packet_length	Sets the size of the packets to use, to wrap the raw data into, for use on the tx ODI interface

DPU_BB_CONTROL

Bits:	Function	Description:
0	start	A 0 to 1 transition of this bit starts the wrapper component wrapping the raw data and passing packets to the ODI interface for eventual transmission
1	stop	A 0 to 1 transition of this bit stops the wrapper component wrapping the raw data and passing packets to the ODI interface. Any packets in progress will be completed, so make sure enough data is passed
2	reset	Setting to a 1 force the odi wrapper, and the tx odi interface into the default startup state
3	clr_alarm	Setting to a 1 resets any of the latched alarms within the bb stats block

DPU_BB_STATUS

Bits:	Function	Description:
0	running	Confirmation that the start has been seen by the wrapper component, and it is accepting raw data for wrapping to transmit to the ODI
1	overflow	The wrapper component needed to pass data to the ODI interface, but couldn't since the ready signal from the ODI wrapper was false.
2	fifo_overflow	This reports that the fifo within the ODI TX interface was overflowed. This should not occur. The reset control signal in the control register needs driven to clear this error state.

General

Apply a reset in the DPU_BB_CONTROL register to put the circuit back to its startup state.

A 0 to 1 transition on the start bit enables the circuit to start receiving the data from the 400 to 512 gearbox and converting it into an AXI4S interface signal set.

A stop will finish on the end of the current packet.

DPU_BB_DATA_COUNT

Bits:	Function	Description:
31:0	data_count	32 bit rolling count of the number of 64byte transfers of raw data to the ODI interface

DPU_BB_CORE_CLOCK_RATE

Bits:	Function	Description:
8:0	core_clock_count	9 bit count of the number of ticks of the 400MHz clock in the last us. I.E. reports the approximate frequency of the core_cck in MHz

ODI Mux Control Ctl/Stat

Duplicated for each ODI Port, 2 per DPU, 4 per M8132A for the 4 ODI startup operation. FPGA A will only have 1 in the 3 ODI plus Ethernet operatinogal mode

These registers are within the sandbox example design, within the dpu_caa_mux component, but some of the outputs reach the top level of the example design to control various features, such as the input mux, or output mux.

DPU_MUX_SELECT

Bits:	Function	Description:
1:0	output_mux_ctl	“00” default selects the 512 to 400 to 512 pass through “01” selects the direct 512 pass through “10” selects the TX Test payload generation
3:2	input_select_ctl	“00” default selects the 512 to 400 to 512 pass through “01” selects the direct 512 pass through This has to be selected to allow the 512 bit ODI_DBG_DATA for capture “10” selects the RX test payload receiver

There are no restrictions in the fpga logic on what the user can program into the output and the input mux control. Likewise; there are no restrictions in the fpga logic on what the user programs in one channel vs the other channel in a design that has 2 channels.

That being said, only certain combinations of output mux and input select make sense, (and would work correctly)

The tables below show the valid combinations of output and input control for the two ports within one FPGA/Sandbox in the case of FPGA A for 4 ODI operation

Valid Combinations for in 2 and out 4

Input Select 1	Output select 2	Operation
00	00	512 to 400 to 512 passthrough
01	01	Direct 512 pass through Capture 512 ODI RX 2
00	10	Tx test payload 4 out Capture 400 bit ODI RX 2
01	10	Tx test payload 4 out Capture 512 bit ODI RX 2
10	10	RX test payload 2 in TX Test payload 4 out

Valid Combinations for in 4 and out 2

Input Select 2	Output select 1	Operation
00	00	512 to 400 to 512 passthrough
01	01	Direct 512 pass through Capture 512 ODI RX 4
00	10	Tx test payload 2 out Capture 400 bit ODI RX 4
01	10	Tx test payload 2 out Capture 512 bit ODI RX 4
10	10	RX test payload 4 in TX Test payload 2 out

In summary,

- if a test payload checker is used on ODI RX 2 then you can only use test payload generator on TX ODI 4
- to get the 512 bit capture, 512 bit pass through has to be selected
- to get the 400 bit capture, the 400 bit pass through has to be selected
- selecting a pass through mode on TX 2 means the matching passthrough mode has to be selected on the RX 4 and vice versa
- if the pass through mode is selected on an rx input, but only the capture is actually required, the test payload tx can be used on the associated passthrough tx port if desired

Valid Combinations for FPGA B or FPGA A in 3 ODI plus 10GbE operation

Since these designs have loopback rather than passthrough signal routing, the only combinations that make sense are to match the settings of the input and output mux control, since the output mux of a particular channel is only connected to the signals from the input mux of the same channel

Input Select 1	Output select 1	Operation
00	00	512 to 400 to 512 loopback 1 ↔ 1, 2 ↔ 2, 3 ↔ 3
01	01	Direct 512 pass through 1 ↔ 1, 2 ↔ 2, 3 ↔ 3
10	10	TX test payload out 1, 2, 3 RX test payload in 1, 2, 3

NOTES:

The wrap channel component/controls have to be setup and configured to run before the mux_ctl component/controls are set to run if the 512 to 400 to 512 path is being used. If just the 400 bit output for capture is required, and not the wrapping, then just the mux_ctl components need configured.

Also, if a 400bit pass through, or 400 bit capture is required, then the M8132A DSP module must be set to use the external reference, and this front panel input must be connected to the reference output port of the M8131A digitizer, to ensure the two 400M clocks are locked together, and no data overflow or underflow occurs

DPU_MUX_CONTROL

Bits:	Function	Description:
0	Start	A 0 to 1 transition of this bit starts the mux component reading the rx odi interface and passing data to the 512 to 400 gearbox or the 512 pass through.
1	stop	A 0 to 1 transition of this bit stops the mux component reading the rx odi interface and stops passing data to the 512 to 400 gearbox or the 512 pass through
2	reset	Setting to a 1 forces the mux control into the startup state
3	clr_alarm	Setting to a 1 resets any of the latched alarms within the mux stats block
4	Halt_update	Set to a 1 to hold the counter registers at their current reading, to allow SW to gather a coherent set, release to 0 to allow the normal updates again
5	resize_en	When set to a 1, the 512 pass through mode will resize the Interlaken/ODI packets to the size set in the DPU_MUX_ODI_PACKET_LENGTH register. Only works for raw Interlaken/ODI, does not handle vita-49 so any traffic with vita will be broken by this operation. ONLY works for a stream where all the packets are a multiple of 64 bytes. Defaults to 0, i.e. OFF
6	use_sync	For the 512 path, the statemachine will not progress to reading, until a sync pulse arrives , 0 = Off, just start by sw, 1 = wait , needs the sw start, and then a sync pulse
7	fifo_depth_sync	If set to a 1, the depth if the rx odi stream if fifo will create the sync, if set to 0, the default, the real external sync is the source

DPU_MUX_STATUS

Bits:	Function	Description:
0	Running	Confirmation that the start has been seen by the mux control component, and it is accepting data from the RX ODI port
1	rx fifo_overflow	This reports that the fifo within the ODI RX interface was overflowed. This should not occur. The reset control signal in the control register needs driven to clear this error state.
2	rx underflow	Latched alarm, indicates that there was less data available from the RX ODI interface than the threshold level to pass to the 512 to 400 gearbox when required. This is an error condition and should not occur if the original source of the ODI is running 400bit 400M flat out
3	tx_overflow	The odi tx interface reported it's fifo overflowed An error condition, the reset control signal in the control register needs driven to clear this error state
4	resize_error	The 512 path was active, and the resize mode was enabled, and a packet which was not an even multiple of 32bytes was seen, so there will be issues where the output is sent to. latched alarm
5	Waiting	The 512 or the 400 path machines have been started, but are sitting waiting for the depth_good signal to indicate there is sufficient data in the fifo to start reading

DPU_MUX_DATA_COUNT_LOWER

Bits:	Function	Description:
31:0	data_count_lower	Lower 32bit portion of the 64bit rolling count of the number of 64byte transfers of raw data to the ODI interface

When the 512 direct pass through is selected, this is a count of the number of transfers of 64bytes.

DPU_MUX_DATA_COUNT_UPPER

Bits:	Function	Description:
31:0	data_count_upper	Upper 32 bit portion of the 64 bit rolling count of the number of 64byte transfers of raw data to the ODI interface

When the 512 direct pass through is selected, this is a count of the number of transfers of 64bytes.

DPU_MUX_TX_FIFO_DEPTH

Bits:	Function	Description:
9:0	odi_tx_fifo_depth	10 bit count of the number of 64byte entries within the tx fifo of the ODI interface, a guide only; the reported value is a few clock cycles behind the actual value due to clock domain crossing etc

This is used when the direct 512 pass-through is enabled

If the 512 to 400 to 512 path is enabled, then the register within the wrap channel has the value instead.

DPU_MUX_RX_FIFO_DEPTH

Bits:	Function	Description:
9:0	odi_rx_fifo_depth	10 bit count of the number of 64byte entries within the rx fifo of the ODI interface, a guide only; the reported value is a few clock cycles behind the actual value due to clock domain crossing etc

This is used when the direct 512 or the 512-400-512 pass-through is enabled

DPU_MUX_STARTUP_DEPTH

Bits:	Function	Description:
9:0	rx_start_depth	10 bit value to set the startup check depth of the rx odi IF fifo.# I.E. the 512 to 400 gearbox control state machine will not start to read the data till this amount of data is reported by the odi rx IF in it's fifo. This will ensure for 400bit @ 400M applications, that the fifo never runs dry during operation flat out Defaults to 0x020 (32d)

DPU_MUX_EMPTY_DEPTH

Bits:	Function	Description:
9:0	rx_empty_depth	10 bit value to set the depth at which if the rx odi fifo interface fifo is reported as being below, when the 512-400-512 path is active and running, when an underflow will be reported Defaults to 0x002 (2d)

DPU_MUX_ODI_PACKET_LENGTH

Bits:	Function	Description:
17:0	resize_packet_length	Sets the size of the packet on the output path that the state machine uses to re-size the incoming Interlaken/raw ODI packets Only affects 512 pass-through mode, and only if the resize_en bit is set. In multiples of 64. Defaults to 0x00004 (4d)

Capture Control

General

Each FPGA Sandbox has a two channel capture RAM built using the UltraScale Plus URAM block memories. It can be fed from different parts of the circuit to capture samples

Either or both of the two 512 pass through data buses can be captured if available, or either or both of the two 400bit outputs of the 512 to 400 gearboxes can be captured if available.

In the FPGA A Configuration B example, the 2nd 400 and 512 inputs are connected to the first channel further through the processing path

Depending on the mode between 128K and 160K samples can be captured

Capture Control

Capture Control Bit	Description	Bit
Format	Leave at 0	0
Channel4/3	1: Channel to capture 0: No Capturing	1
Channel2/1	1: Channel to capture 0: No Capturing	2
Start	Start capture (set separately)	3

Format

Data Format	Description	Value
	Leave at 0	0

NOTE: Depending on the DPU, different selected channels are accessed. DPU A is connected to channels 2 and 4, DPU B to channels 1 and 3.

Capture Data Vectors

Number of data vectors to capture in the capture memory.

(31:0)	
DataVectors(31:0)	Number of vectors to capture

The range of data vectors is 1 to 4096. In 10-bit format, the number of samples in a vector are 40 and in 16 bit mode a vector contains 32 samples.

This means in 16bit mode: 131,072 samples can be captured

(4096 vectors * 32 samples per vector)

and in 10bit mode: 163,840 samples can be captured

(4096 vectors * 40 samples per vector)

Read Request Addr

Offset(15:0)	Offset
Start Addr (31:16)	Start address

StartAddr and Offset

The combined Start address and offset is a 32-bit aligned register in the FPGA AXI4Lite memory map. The Address is in the upper 16bit portion, and the offset is in the lower 16bit portion

The address range is 0 to 0x0fff.

10-bit format: Start vector address range is 0 to 4095 and offset range is 0 to 39 (allowed values are 0, 2, 4 ...).

16-bit format: Start vector address range is 0 to 4095 (allowed values are 0, 4, 8 ...) and offset range is 0 to 15

Read Request Samples

Samples(31:0)	Number of samples to read
----------------------	----------------------------------

Samples

Number of samples range is 1 to 32768 (16 bits)

10-bit format: No of samples 1 to 4*32768 (allowed values are 0, 2, 4 ...)

Read Request Control

Read Control Bit	Description	Bit
Format	Data format for capturing	0
Channel4/3	1: Channel to capture 0: No Capturing	1
Channel2/1	1: Channel to capture 0: No Capturing	2
Start	Start reading from memory (set separately)	3

Format

Data Format	Description	Value
	Leave at 0	0

NOTE: Depending on the DPU selected channels are accessed.
DPU A is connected to channels 1 and 3, DPU B to channels 2 and 4.

Read Data

Read the requested data

Return value

Depending on the data size the return value consists of two (Data10) or two (Data16) data values. The most significant bits are filled with zeros for the 10bit samples.

10-bit Data Size

Parameter 1(15:0)	Parameter 2(15:0)
0b000000 & Data10(9:0)	0b000000 & Data10(9:0)

16-bit Data Size

Parameter 1(15:0)	Parameter 2(15:0)
Data16(15:0)	Data16(15:0)

Abort

Abort read from the capture memory.

Parameter 1(15:0)	Parameter 2(15:0)
Abort(0)	Set to abort a read
31:1	Unused

Capture Memory Status Register

CaptureMemStatus	Description	Bit
0	4,3, Previous data capture command has not completed, yet.	0
1	2,1 Previous data capture command has not completed, yet.	1
31:2	Not used	

DDC Access Control Register

Default value after DPU reset is 0x00000000.

DDCAccessControl	Description	Bit
DDCdata	0: 10-bit data 1: 16bit data	0

DDCdata

Set to 0 for capturing and reading back 10bit data

Set to 1 for capturing and reading back 16bit data

ReadData(0 - F) Read only

Read the requested data

ReadData0

ReadData1

ReadData2

ReadData3

ReadData4

ReadData5

ReadData6

ReadData7

ReadData8

ReadData9

ReadDataA

ReadDataB

ReadDataC

ReadDataD

ReadDataE

ReadDataF

The user can just keep reading the data by repeatedly reading the single ReadData register.

Alternatively, the user can read from the above 16 ReadDataN registers one after the other to bring up 16 pairs of sample data.

However there is no real advantage in this, in that the AXI4Lite bus does not permit burst reads, so it doesn't gain anything over just reading the ReadData register 16 times.

They are all fed from a single fifo, so there is no order requirement, the data will always appear in the reads in the same order, irrespective of the order of reading the registers above

General

If an M8131A Digitizer module is the source of the ODI data being captured within the M8132A DSP module, then the data recovered is in one of two formats

In full 16GSa/s (32 GSa/s) operation, the data is captured as 10bit samples, 40 samples per 400MHz clock for a 400bit bus. (80 samples per 400MHz for an 800 bit bus) For transmission over ODI each 400bit bus is gearboxed to a 512bit wide bus. On reception in the DSP module, the 512 to 400 gearbox path should be selected to recover the original 40 samples per 400Mhz clock.

In this mode the 400 bit capture should be selected

If one of the direct decimation or DDC (Digital down conversion) modes is selected the data is converted into 16bit samples, and fits naturally on the 512bit bus for transmission over ODI.

In this case the 512/DDC mode for capture should be selected.

There are some complications due to the sample ordering on the various busses that is explained in the next section.

In general; to read back samples, three approaches could be taken.

1. Set the start address and offset to 0
2. Set the number of samples to read, trigger the read, and then just keep reading for the matching number of samples.

Or

1. Set the start address and offset to 0
2. Set the number of samples to read as 32 or 40 (16 bit mode or 10 bit mode)
3. Trigger the read
4. Read 40 samples in 10bit mode, as 20 reads of 2 samples per read
5. Read 32 samples in 16 bit mode, as 16 reads of 2 samples per read
6. Increment the start address by 1, (leave the offset at 0)
7. Leave the number of samples to read as before
8. Trigger the read
9. Read 40 samples in 10bit mode, as 20 reads of 2 samples per read
10. Read 32 samples in 16 bit mode, as 16 reads of 2 samples per read
11. And repeat from 6 until all the captured samples have been read back

Or

1. Set the start address and offset to 0
2. Set the number of samples to read to 1
3. Trigger the read
4. Read one sample
5. Increment the offset
6. Trigger the read
7. Read one sample
8. Increment the offset
9. Trigger the read
10. Read one sample

11. Until either 32 or 40 samples have been read, as appropriate, for 16bit or 10 bit data capture
12. Then increment the address by 1, and set the offset back to 0
13. repeat from 3

Ethernet

The Ethernet example design within the FPGA A Configuration B example contains an octal generator and an octal checker. They can be programmed to drive and monitor the static area Ethernet interface circuitry. They are both register driven.

As can be seen in the diagram for FPGA A Example Design for 3 ODI plus 10GbE, the (eth_test_top) component is directly connected to the 8 transmit and 8 receive channels.

The ethernet test and check blocks are capable of generating and checking streams of Ethernet packets. Each generator can generate a programmable burst of packets each of the same programmable length. Alternatively, they can generate a continuous stream of packets until stopped.

The generated packets are generated with one of a small set of Source and destination addresses. The Ethertype is also programmable. The rest of the packet contains an incrementing number pattern.

The checker is designed to work closely with the generator. It has the same small set of source/destination addresses and a programmable ethertype. It checks the length of every packet is as expected and compares the source & destination addresses, Ethertype and incrementing data pattern against expected values.

Lastly the checker can be told, from which channel the data has been transmitted. This too, is checked.

There are 8 virtually identical sets of registers to drive the octal generator & checkers. One set for each Ethernet link numbered from 1 to 8.

The 8 sets of Ethernet Test Registers are offset by 0x40 from each other.

Ethernet Test Channel 1 Base	0x8000
Ethernet Test Channel 2 Base	0x8040
Ethernet Test Channel 3 Base	0x8080
Ethernet Test Channel 4 Base	0x80C0
Ethernet Test Channel 5 Base	0x8100
Ethernet Test Channel 6 Base	0x8140
Ethernet Test Channel 7 Base	0x8180
Ethernet Test Channel 8 Base	0x81C0

There are 13 registers used per Ethernet link at the following offsets from the above base addresses.

Address	Offset	Type	Description
0x00		RW	ET_CONTROL_REG
0x04		RW	ET_TX_ADDRESS_REG
0x08		RW	ET_TX_LENGTH_REG
0x0C		RW	ET_TX_IPG_REG
0x10		RW	ET_TX_REPEATS
0x14		RW	ET_RX_ADDRESS_REG
0x18		RW	ET_RX_LENGTH_REG
0x1C		RW	ET_CORE_STATS_REG
0x20		RO	ET_STAT_ERRORED
0x24		RO	ET_STAT_LENGTH_ERR
0x28		RO	ET_STAT_DATA_ERR
0x2C		RO	ET_TX_ACK_COUNT
0x30		RO	ET_RX_ACK_COUNT

The contents of each register are as follows:-

ET Control

Bit 12 (Global Freeze) can only be written at Channel1's offset (0) . All 8 channels will read this one bit.

Bits:	Function	Description:
0	TX packet Start	Set to 1 to start packet generation. The 0 to 1 transition on the register bit causes packets to start generating.
4	TX Update	Only when set to one, do the TX setup registers get sent to the generator. This means that the individual registers can all be set up then the cumulative effect applied to the generator on writing tx_update to 1. Writing it back to zero again before updating the other registers again is a good idea. This allows the software to have a new set of packets ready for when the first repeat sequence finishes.
8	RX freeze	Write to 1 to freeze the RX results for this channel so that a coherent set can be read When set to zero the registers will read "live"
12	Global Freeze	Like RX freeze but applies to all 8 channels simultaneously.
16	Enable	Set to one, to enable generator. This should be set to one before setting TX packet start. A 1 to 0 transition will cause a current packet to complete before the generator stops.
20	Continuous	Set to one to generate continuous packets. Repeats should be set to 0 in continuous mode.

Power up value 0

ET TX Address

Bits:	Function	Description:
2:0	TX Destination Address Index	A three bit index into a set of 8 Destination Addresses. See Below
6:4	TX Source Address Index	A three bit index into a set of 8 Destination Addresses. See Below
31:16	TX Ethertype	16 bit Ethertype value inserted into Ethertype Field. Does not support VLAN so just don't set to a VLAN value.

Power up value 0x88B50045

ET TX Length

Bits:	Function	Description:
13:0	TX Packet Length	Length of packets to be generated

Power up value 0x40

ET TX IPG

Bits:	Function	Description:
31:0	Inter Packet Gap	Interpacket Gap count. IPG is Count * 3.2ns + minimum (possibly 16ns)

Power up value 0x100

ET TX Repeats

Bits:	Function	Description:
31:0	Repeats	Number of identical packets to generate.

Power up value 0x40

ET RX Address

Bits:	Function	Description:
2:0	RX Destination Address Index	A three-bit index into a set of 8 Destination Addresses. See Below for details
6:4	RX Source Address Index	A three-bit index into a set of 8 Destination Addresses. See below for details
10:8	RX Channel	Program with the TX channel number attached to this lane minus 1. Allows cross wiring of channels.
12	Check loopback	Set to one to check a channel that is receiving an already looped back channel. Swaps over the SA and DA in the checker compared with the non-loopback check.
31:16	RX Ethertype	Program with expected Ethertype value. Does not support VLAN.

Power up value 0x88B50n45. (n = channel -1)

ET RX Length

Bits:	Function	Description:
13:0	RX Packet Length	Program with the expected packet length -1. e.g. for 64 byte packets program to 0x003F (63)

Power up value 0x3F (to expect a packet length of 64)

ET Rx Errored Count

Bits:	Function	Description:
31:0	RX Errored Count	Count of undersized, oversized, FCS errored and code errored packets detected by the core. A coherent set will be obtained if the results are frozen using RX freeze Rolls over at 2 ³²

Power up value 0

ET Rx Length Errored Count

Bits:	Function	Description:
31:0	RX Length Errored Count	Count of packets received that were not the programmed expected length. A coherent set will be obtained if the results are frozen using RX freeze Rolls over at 2 ³²

Power up value 0

ET Rx Data Errored Count

Bits:	Function	Description:
31:0	RX Data Errored Count	Count of packets that did not have the expected SA, DA Ethertype or data values. Counts 1 for SA,DA or Ethertype error and 1 for each 32 bit data word that contains any errors. Length errors will likely induce data errors as well. A coherent set will be obtained if the results are frozen using RX freeze Rolls over at 2^{32}

Power up value 0

ET Tx Ack Count

Bits:	Function	Description:
31:0	Tx Ack Count	Counts one for each set of packets sent. A repeat of 100 counts 1

Power up value 0

ET RX Ack Count

Bits:	Function	Description:
31:0	RX Ack Count	Counts one for every packet received. Will count 100 for a repeat of 100 packets

Power up value 0

Ethernet Test Operation and procedures

Addresses

The generator and checker use a common set of source and destination address values. There are 8 of each. A three-bit source address and destination address index is used to select an address.

Source Addresses

The source addresses are selected using the Source Address Index. The transmitted bottom nibble of the source address is changed to the channel number (Range 0-7). This is checked against the programmed Rx Channel in the tester. It must match to avoid counting data errors.

Index	Address	Type
0	FE:FF:FF:FF:FF:F0	
1	00:00:00:00:00:00	
2	10:20:30:40:50:60	
3	12:34:56:78:9A:B0	
4	14:FE:B5:DD:9A:82	
5	56:54:53:52:51:50	
6	FE:DC:BA:98:76:50	
7	CA:57:90:0D:AD:D0	

Destination Addresses

Destination addresses have certain reserved bits that indicate that the packet should be broadcast or multicast. Some of the addresses in the set of 8 are broadcast and multicast and should be correctly counted in broadcast and multicast counters of a transmitter and receiver.

When testing loopback, it is not advisable to use broadcast and multicast addresses.

Index	Address	Type
0	FF:FF:FF:FF:FF:FF	Broadcast
1	01:00:5E:01:02:03	Multicast
2	33:33:01:02:03:04	Multicast
3	01:C2:03:00:00:03	Multicast
4	00:10:20:30:40:50	Unicast
5	D6:D4:D3:D2:D1:D0	Unicast
6	FE:DC:BA:98:76:50	Unicast
7	CA:57:90:0D:AD:D0	Unicast

Source Address Override

The default operation of an M8132A is to override the source address with an address programmed into a fixed area register. In order for the test blocks to work, this must be turned off to allow the generated Source Address to reach the Ethernet core. This source address override can be turned off in one of two ways.

1. On the Soft Front Panel click on the advanced settings and turn off the override SA for each channel.
2. Send the following SCPI command "ETHERnet:PORT[1:8]:FRAME:MACSource OFF" for each of channels 1-8

If not turned off, the tests will return 2 data errors for every packet containing an overridden source address.

Also don't forget to turn the channels on.

Examples

Two examples are given. In example 1 a loopback dongle is attached to port 4. This will wire channel 1 TX to channel 1 RX, channel 2 TX to channel 2 RX etc. Example 2 uses the Ethernet breakout cable plus some connectors to wire the channels in a more realistic manner

Loopback Dongle

With the loopback dongle plugged into port 4, all the ethernet channels are looped back upon themselves. The default values in the Ethernet test control registers have been chosen so that this configuration will work as soon as the generator is started without changing any registers other than the control register.

The default register settings are not applied to the generator until the TX Update bit is set. This is achieved by setting the ET control register as follows:-

ET TX Control Chan 0

<i>TX Packet Start <= 0</i>	<i>Do not start until everything is applied</i>
<i>TX Continuous <- 0</i>	<i>A single burst of packets will be sent so continuous is off.</i>
<i>TX Enable <- 1</i>	<i>Enable the Generator</i>
<i>TX-Update <-1</i>	<i>Setting update applies all the TX default values to the generator</i>

The update bit can be set on the same write as the TX_enable and TX_continuous are changed. It should be written back to 0 afterwards. The generator will not start generating packets yet. In order for this to happen, the TX packet start bit needs to be set. This should occur on a write AFTER the one that sets everything else.

The UPDATE bit can be cleared on the write that starts the generator.

ET TX Control Chan 0

<i>TX Packet Start <= 1</i>	<i>Start the generator</i>
<i>TX Continuous <- 0</i>	<i>A burst of packets will be sent so continuous is off.</i>
<i>TX Enable <- 1</i>	<i>Enable the Generator</i>
<i>TX-Update <-0</i>	<i>Update can now be cleared.</i>

The generator will now generate a burst of 64 64-byte packets with an IPG of just over 256 (0x100) byte times. The default values in the checker will check for the default TX packets.

There should be no errors as long as the SA override was switched off.

Breakout cable

For the purposes of this example. It is assumed that externally numbered channels 1 and 2 are wired back to back. i.e. what is Transmitted on 1 is received on 2 and vice versa.

Connectors exist that allow LC TX/RX pairs to be connected together so that this can be achieved with the Ethernet breakout cable

Internally, at the registers, the channels are numbered from 0 to 7 as opposed to the 1-8 customer channel numbering so we need to program channels 0 and 1. To avoid confusion I0 and I1 will be used for internally numbered channels and E1 and E2 for the external customer numbers.

The intent is to transmit a burst of 100 packet from channel I0 to I1 of the minimum length of 64, and send a continuous packet stream from I1 to I0 of the maximum length of 4000.

When setting up a generator and checker it makes sense to set up the checker first.

To set up the channel I1 receiver to receive packets generated by transmitter I0

ET Rx Address Chan 1

Rx DA Index <- 5 *Set to what is being Tx'd*
RX SA Index <- 4 *Set to what is being Tx'd*
RX channel <-0 *TX'd from channel I0*
Check Loopback <- 0 *No loopback swapping needed*
RX-Ethertype <- 0x600 *Set to what is being Tx'd*
ET RX Length Chan 1
RX packet Length <- 63 *Packet length is 64. Prog. with length -1*

Now do the same for the RX on channel I0

ET Rx Address Chan 0

Rx DA Index <- 7 *Set to what is being Tx'd*
RX SA Index <- 6 *Set to what is being Tx'd*
RX channel <-1 *TX'd from channel I1*
Check Loopback <- 0 *No loopback swapping needed*
RX-Ethertype <- 0x600 *Set to what is being Tx'd*
ET RX Length Chan 0
RX packet Length <- 3999 *Packet length is 4000. Prog. with length -1*

RX side settings are set immediately.

Now set up the generator on Channel I0

ET TX Address Chan 0

TX DA Index <- 5 *DA 5*
TX SA Index <- 4 *SA 4*
TX-Ethertype <- 0x600 *Ethertype value*
ET TX Length Chan 0
TRX packet Length <- 64 *Packet length is 64.*
ET TX IPG Chan 0
TX IPG <- 100 *IPG = 100*
ET TX Repeats Chan 0
TX Repeats <- 100 *TX Burst of 100 packets*

The above TX settings are not applied until the TX Update bit is set. The ET control register is now set.

ET TX Control Chan 0

TX Packet Start <= 0 *Do not start until everything is applied*
TX Continuous <- 0 *A single burst of packets will be sent so continuous is off.*

TX Enable <- 1 Enable the Generator

TX-Update <-1 Setting update applies all the TX settings to the generator

The update bit can be set on the same write as the enable and continuous are changed. It should be written back to 0 afterwards. The generator will not start generating packets yet. In order for this to happen, the TX packet start bit needs to be set. This should occur on a write AFTER the one that sets everything else.

The UPDATE bit can be cleared here.

ET TX Control Chan 0

TX Packet Start <= 1 Start the generator

TX Continuous <- 0 A burst of packets will be sent so continuous is off.

TX Enable <- 1 Enable the Generator

TX-Update <-0 Update can now be cleared.

The generator will now generate a burst of 100 64 byte packets with an IPG of just over 400 byte times.

Channel 1 now gets set up.

ET TX Address Chan 1

TX DA Index <- 7 DA 7

TX SA Index <- 6 SA 6

TX-Ethertype <- 0x600 Ether type value

ET TX Length Chan 1

TRX packet Length <- 4000 Packet length is 4000.

ET TX IPG Chan 1

TX IPG <- 100 IPG = 0

ET TX Repeats Chan 1

TX Repeats <- 0 Repeats should be set to 0 for continuous mode

As before, the ET control register is now set up.

ET TX Control Chan 1

TX Packet Start <= 0 Do not start until everything is applied

TX Continuous <- 1 continuous packets will be sent so continuous is on.

TX Enable <- 1 Enable the Generator

TX-Update <-1 Setting update applies all the TX settings to the generator

And the generator is then started

ET TX Control Chan 1

TX Packet Start <= 1 Start the generator

TX Continuous. <- 1 continuous packets will be sent so continuous is on.

TX Enable <- 1 Enable the Generator

TX-Update <-0 Update can now be cleared.

By now channel 0 will have completed its sending of 100 packets but channel 1 will continue to send packets until TX *continuous* and TX *enable* are cleared (with TXupdate = 1)

The results of channel 1 can be read by simply reading the RX errored, data errors, length errors and RX Ack Counts. The RX ack counter should read pack 100 since it should have received 100 packets. Everything else should read back 0.

Channel 0 results are a little more complex. Channel 1 is continuously feeding packets. To read a live set of results, the RX freeze bit in the RX control register can be set. This will freeze all the counters allowing a coherent set to be obtained. Rx Freeze must stay on until the reading is complete. It is not an edge sensitive signal.

ET TX Control Chan 1

TX Packet Start <= 0

Do not start until everything is applied

TX Continuous <- 0

continuous packets will be turned off.

TX Enable <- 0

Disable the Generator

TX-Update <-1

Setting update applies these changes to the generator

ET TX Control Chan 1

TX Packet Start <= 0

Do not start until everything is applied

TX Continuous <- 0

continuous packets will be turned off.

TX Enable <- 0

Disable the Generator

TX-Update <-0

Leave update = 0

It is good practice to always reset update back to zero, ready for the next test.

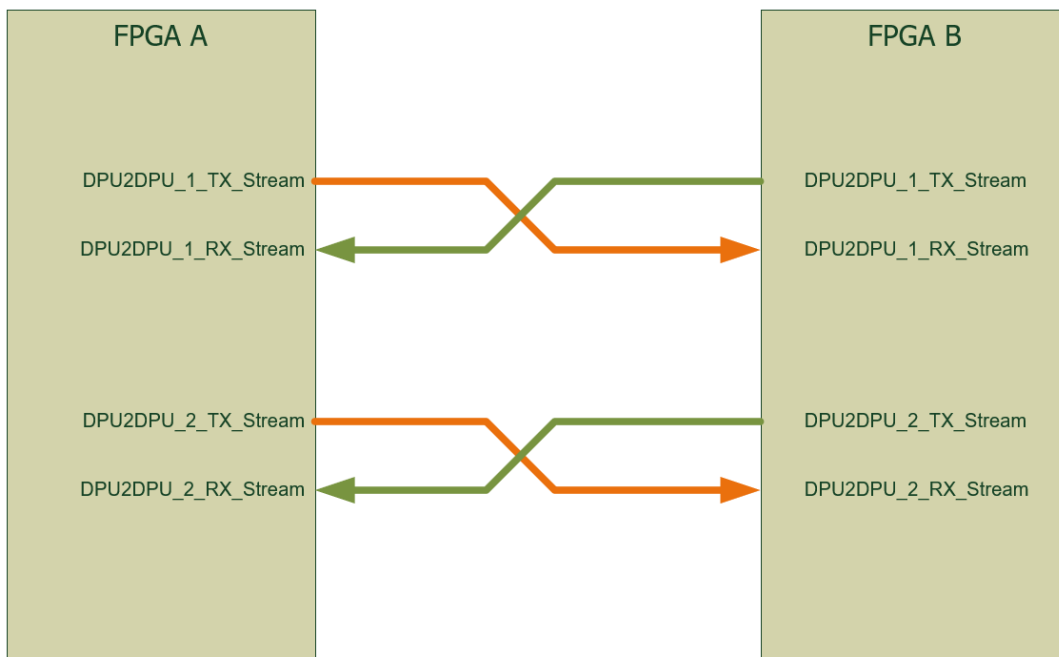
DPU2DPU

General

All the 3 example designs contain circuitry to transmit and receive packets over the DPU2DPU links.

As can be seen in the three diagrams above the (dpu2dpu_test) component of the example design just directly connects to the 2 transmit and 2 receive channels.

There are no front panel connections associated with the DPU2DPU links, the static areas contain Xilinx Aurora core IP components connected to transceivers which with associated logic take the 512bit AXI4S streams and connects them via the PCB of the module to the other FPGA, the diagram below shows a simplified diagram of the connections.



The packet size can be set to any integer value of bytes from 1 to 2,097,152 (2M)

The transmit and receive channels are entirely independant within one fpga

The two channels are entirely independant within one fpga

The only limitation is maximum speed on the link, smaller packets, becuae of the Aurora protocol overhead used to transfer the data bewteen the two FPGA will not acheive the same maximum rate as larger packets.

An 8192 byte packet (8K) is large enough to sustain a 160Gb/s data rate over the dpu2dpu links, this would be a 128 beat transfer of 64 bytes per beat during each packet

A 131,072 byte packet (128K) is large enough to sustain a 160Gb/s data rate over the dpu2dpu links, this would be a 2048 beat transfer of 64bytes per beat during each packet.

The logic within the example design is capable of driving well over the ~160Gb/s limit of the aurora connections between the two fpga, normal AXI4S tready pushback would occur if the test transmitter is set at too high an average traffic level

A note on the DPU2DPU AXI4S stream

The DPU2DPU links are more like standard AXI4S links that support continuous unaligned, but only support null bytes at the end of the packet, not at the start

a 128byte packet

Data(511 downto 0)	tkeep	tlast
first 64 bytes of data	0xFFFFFFFFFFFFFFFF	0
last 64 bytes of data	0xFFFFFFFFFFFFFFFF	1

a 129byte packet

Data(511 downto 0)	tkeep	tlast
first 64 bytes of data	0xFFFFFFFFFFFFFFFF	0
64 bytes of data	0xFFFFFFFFFFFFFFFF	0
last 1 byte of data	0x0000000000000001	1

Memory Map

There is one (dpu2dpu_test) component instantiated within the example designs, it contains two sets of transmit and receiver generator and checker logic, two sets of statistics and counters, and two sets of registers

The registers are identical for each of the two channels, and the registers appear at the same place within the memory map for all 3 example designs

DPU2DPU IF	Base Address
1	0x0A00
2	0x0B00

DPU2DPU Test Control & Status

These registers control and monitor test payload generation and checking and there are two sets in each of the (dpu2dpu_test) components

Address Offset	Type	Description
0x00	RW	TX Packet Length
0x04	RW	TX Packet Start Gap
0x08	RW	TX Packet Count
0x0C	RW	TX Control
0x10	RO	TX Status
0x14	RO	RX Packet Count
0x18	RW	RX Control
0x1C	RO	RX Status
0x20	RO	RX Active Clocks Lower
0x24	RO	RX Active Clocks Upper
0x28	RO	RX Total Clocks Lower
0x2C	RO	RX Total Clocks Upper
0x30	RO	Errors
0x34	RO	Core clock Rate
0x38	RO	RX fifo depth
0x3C	RO	RX fifo depth max

Register Descriptions

DPU2DPU Test Packet Generation Registers

Duplicated for each DPU2DPU Channel, 2 per FPGA Customer Accessible Area

TX Packet Length

Bits:	Function	Description:
21:0	TX Packet Length	Set before a "run" Units of bytes Determines length of packets in a run Default value 0x000080 , i.e. 128 bytes, two full width transfers Max value is 0x200000

TX Packet Start Gap

Bits:	Function	Description:
31:0	TX Packet Start GAP	Set before a “run” Units of 64 bytes Determines gap between successive start of packets. Must be larger than TX Packet Length * 2 Default value 0x0800 , i.e. 2048d, ie. 200Mb/s data rate if used with teh default packet size

The traffic generator runs at 400MHz, and has a data width of 512 bits. This is higher than the possible ~160Gb/s total dpu2dpu link.

The data will be sent out during a packet as fast as the back pressure from the DPU2DPU circuitry in tee static area will allow.

Once a packet has been finished, the generator will wait until it is time to send a new packet, as determined by the value set in the Tx Packet Start GAP.

As the name implies, the gap is time in clock ticks between the start of each new packet.

The average data rate calculation is thus

Bit Rate = ((Tx Packet Length * 8) / Tx Packet Start Gap) * 400E6 in bit/s

TX Packet Count

Bits:	Function	Description:
31:0	TX Packet Count	Set before a “run” Determines how many packets are sent in a run, a 0 implies run until a stop action bit is set (see below)

TX Control

Bits:	Function	Description:
0	Reset	Reset, i.e. put all registers/logic counters into known state for this data generator. Default/Reset value ‘1’
1	Stop	Stop, i.e. halt transmission of packets after any current packet in progress completes Used to halt continuous mode, or to stop counted mode before the count is complete, and stops the Active and Total clock counters
2	Start	Start, i.e. begin packet transmission, and start the Active and Total clock counters

TX Status

Bits:	Function	Description:
0	Running	If read as a 1, a tx run is in progress, if read as a 0, either the run has not started, or a started counted packet run has finished
1	Overflow	Latched status bit, indicates that a packet not was complete before the next packet was due to start, the packet count and packet length and packet start gap are incorrectly programmed

RX Packet Count

Bits:	Function	Description:
31:0	RX Packet Count Actual	Reset before a “run” Count of how many end control signals have been seen since the last “reset”

RX Control

Bits:	Function	Description:
0	Reset	Reset, i.e. put all registers/logic/counters/alarm status in to known state. Default/Powerup value '1'
1	Stop	Stop, i.e. halt counting packets or errors after any current packet in progress completes
2	Start	Start, i.e. begin counting packets or errors
3	Hold Register Value	Set to a '1' to halt updating of the registers from the counters. i.e. the counters still run, so nothing is "lost" but the registers will not be updated while this bit is set to a "1". This allows a coherent set of values to be read, in particular the, the 64 bit counts will not have the risk of being read splitting a lower 1/2 roll over and giving strange results. Put back to a '0' to allow normal updates. Defaults to "0"

RX Status

Bits:	Function	Description:
0	Running	Run is in progress
1	Fifo overflow latched	Latched alarm bit reporting the fifo the test receiver listens to has reported overflow
2	Errout	0 = Normal 1 = Error Reports the fact an error was seen in the checker

RX Active Clocks Lower

Bits:	Function	Description:
31:0	RX Active Clocks lower	Number of bytes transferred between the start and stop, lower 32 of a 64 bit count. Activated by start, stopped by stop

RX Active Clocks Upper

Bits:	Function	Description:
31:0	RX Active Clocks upper	Number of bytes transferred between the start and stop, upper 32 of a 64 bit count. Activated by start, stopped by stop,

RX Total Clocks Lower

Bits:	Function	Description:
31:0	RX Total Clocks lower	Total number of 400MHz domain clock cycles between the start and stop. Lower 32 of a 64 bit count

RX Total Clocks Upper

Bits:	Function	Description:
31:0	RX Total Clocks upper	Total number of 400MHz domain clock cycles between the start and stop, upper 32 of a 64 bit count

Core Clock Rate

Bits:	Function	Description:
8:0	stat_clock_rate	Reports the clock rate of the core clock in MHz, as measured using a 1us repetitive pulse generated from the 150MHz wishbone clock, just a rough measure

RX fifo depth

Bits:	Function	Description:
7:0	Rx_fifo_depth	Reports the current depth of the AXI4S fifo for the receive direction in the dpu2dpu IF connected to this rx tester

RX fifo depth_max

Bits:	Function	Description:
7:0	Rx_fifo_depth_max	Reports the highest depth of the AXI4S fifo for the receive direction in the dpu2dpu IF connected to this rx tester since the last issue of the reset of the rx control

Test Benches

While not strictly part of the FSP, nor the IP component, to aid understanding of the example design operation, some VHDL test benches have been provided for use.

These are located within the IP folder of the BSP install directory of the M8132A, i.e. C:\Program Files\Keysight\M8131\FSP\ip\test_benches\

m8132a_pw_a_sandbox_a_example_4_2_tb.vhd

This testbench doesn't stimulate anything, but exists to allow the user to load up the top level of the IP component for fpga a sandbox 1 for configuration A with the 2 ODI ports.

This would then allow the user to make sure they have all the necessary files below that top level and xilinx libraries compiled within their simulation environment to allow the top level to be compiled cleanly, and loaded into the simulator of choice

All the source files within C:\Program Files\Keysight\M8131\FSP\ip\common\vhd\ should be loaded into a suitable simulation project; (strictly they are not all needed, but for simplicity, take them all)

The eight XCI files within C:\Program Files\Keysight\M8131\FSP\ip\common\xci should be loaded into a suitable Xilinx Vivado IP project, and have the generation products created to produce the required simulation files, which should then be added to the simulation project; (again, strictly they are not all needed for this particular test bench, but for simplicity, take them all)

There are five Xilinx libraries that will have to be compiled inside Vivado for your simulator of choice. See the Xilinx Vivado help for details on how to compile simulation libraries for the simulator of your choice

The libraries required for the example design are "xpm", "unisim", "axis_register_slice_v1_1_20", "axi_register_slice_v2_1_20", "axi_amm_bridge_v1_0_10"

These should be compiled for Vivado 2019.2, the version supported by the M8132A FSP

m8132a_pw_d2d_b_sandbox_b_example_4_2_tb.vhd

This testbench doesn't stimulate anything, but exists to allow the user to load up the top level of the IP component for fpga b sandbox 2 for configuration B.

This would then allow the user to make sure they have all the necessary files below that top level and xilinx libraries compiled within their simulation environment to allow the top level to be compiled cleanly, and loaded into the simulator of choice

All the source files within C:\Program Files\Keysight\M8131\FSP\ip\common\vhd\ should be loaded into a suitable simulation project; (strictly they are not all needed, but for simplicity, take them all)

The eight XCI files within C:\Program Files\Keysight\M8131\FSP\ip\common\xci should be loaded into a suitable Xilinx Vivado IP project, and have the generation products created to produce the required simulation files, which should then be added to the simulation project; (again, strictly they are not all needed for this particular test bench, but for simplicity, take them all)

There are five Xilinx libraries that will have to be compiled inside Vivado for your simulator of choice. See the Xilinx Vivado help for details on how to compile simulation libraries for the simulator of your choice

The libraries required for the example design are "xpm", "unisim", "axis_register_slice_v1_1_20", "axi_register_slice_v2_1_20", "axi_amm_bridge_v1_0_10"

These should be compiled for Vivado 2019.2, the version supported by the M8132A FSP

m8132a_pw_d2d_eth_a_sandbox_b_example_4_2_tb.vhd

This testbench doesn't simulate anything, but exists to allow the user to load up the top level of the IP component for fpga a sandbox 1 for configuration B with 1 ODI and 8 Ethernet ports.

This would then allow the user to make sure they have all the necessary files below that top level and xilinx libraries compiled within their simulation environment to allow the top level to be compiled cleanly, and loaded into the simulator of choice.

All the source files within C:\Program Files\Keysight\M8131\FSP\ip\common\vhd\ should be loaded into a suitable simulation project; (strictly they are not all needed, but for simplicity, take them all)

The eight XCI files within C:\Program Files\Keysight\M8131\FSP\ip\common\xci should be loaded into a suitable Xilinx Vivado IP project, and have the generation products created to produce the required simulation files, which should then be added to the simulation project; (again, strictly they are not all needed for this particular test bench, but for simplicity, take them all)

There are five Xilinx libraries that will have to be compiled inside Vivado for your simulator of choice. See the Xilinx Vivado help for details on how to compile simulation libraries for the simulator of your choice

The libraries required for the example design are "xpm", "unisim", "axis_register_slice_v1_1_20", "axi_register_slice_v2_1_20", "axi_amm_bridge_v1_0_10"

These should be compiled for Vivado 2019.2, the version supported by the M8132A FSP

dpu_caa_pw_example_4_2_data_512_capture_tb.vhd

This testbench also operates at the top level of the example design for the fpga a sandbox 1 configuration A IP component. In this case, the component is stimulated with signals and the outputs are checked for correct operation.

the ODI receive input 4 is driven with a counting pattern in the data

The AXI4LITE register bus is driven to write and read registers within the IP component

The design is configured to operate in the 512 pass through mode, I.E. the output ODI transmit 2 is checked to see that the data driven in on ODI receive input 4 is correctly passed back out with no errors

The Capture Memory is configured for 512 bit capture operation, and the capture is started.

The Capture Memory is checked to see that the capture process has started

The data input is started, to drive data into the ODI receive input 4

The Capture Memory is checked, after a suitable delay, to see that it has completed its capture of the requested number of vectors of the 512 data bus

32 samples are read back from the capture ram

The first sample is checked for the correct value, it should be a straightforward exercise for the user to extend the check to the other 31 captured values

The amount of data passed back out is checked against the expect value

The number of packets passed back out is checked against the expected value

Each of the 512bit data vectors passed back out is checked against the expected value

There are five Xilinx libraries that will have to be compiled inside Vivado for your simulator of choice. See the Xilinx Vivado help for details on how to compile simulation libraries for the simulator of your choice

The libraries required for the example design are "xpm", "unisim", "axis_register_slice_v1_1_20", "axi_register_slice_v2_1_20", "axi_amm_bridge_v1_0_10"

These should be compiled for Vivado 2019.2, the version supported by the M8132A FSP

capture_mem_top_tb.vhd

This test bench focuses in on one of the components inside the example design IP, in particular, the capture memory.

It operates at the level of the capture_mem_top component.

A set of 40 different count values are concatenated to create the 400 bit bus,

The capture memory is setup to capture on one of the 400 bit input buses.

The status is read back to ensure the capture is running

The input bus is driven with changing data each clock cycle for a set number of cycles

The status is read again to check the capture is complete

The data is read back with a double loop

The outer loop repeats 4096 times

The inner loop repeats 20 times recovering 2 16 bit values each time which contain the captured 10bit samples

The captured data is stored in an array

Then the full array is checked to ensure it contains the expected original data

There are two Xilinx libraries that will have to be compiled inside Vivado for your simulator of choice. See the Xilinx Vivado help for details on how to compile simulation libraries for the simulator of your choice

The libraries required for the example design are "xpm" and "unisim"

These should be compiled for Vivado 2019.2, the version supported by the M8132A FSP

dpu_caa_pw_example_4_2_ethernet_tb.vhd

This testbench also operates at the top level of the example design for the fpga a sandbox 1 IP component for configuration B, the version with 1 ODI port and 8 10GbE ports. In this case, the component is stimulated with signals and the outputs are checked for correct operation.

The testbench generates a stream of packets on each of 8 ethernet channels. Packets range in size from 64 to 4000 bytes. The output transmit channels are looped back to become the input receive channels at the top level of the test bench.

A table of packet description records is used to define the length, IPG, repeat count, address and ethertype on each stream. The testbench converts these values to register writes and sets up the checker and generator. The packets are then started. After an appropriate amount of time, the checker registers are checked for correct values.

The test reports a pass if all the checker registers return expected values. It reports a fail otherwise.

There are five Xilinx libraries that will have to be compiled inside Vivado for your simulator of choice. See the Xilinx Vivado help for details on how to compile simulation libraries for the simulator of your choice

The libraries required for the example design are "xpm", "unisim", "axis_register_slice_v1_1_20", "axi_register_slice_v2_1_20", "axi_amm_bridge_v1_0_10"

These should be compiled for Vivado 2019.2, the version supported by the M8132A FSP

dpu_caa_pw_example_4_2_data_400_path_tb.vhd

This testbench operates at the top level of the example design for the fpga a sandbox 1 IP component for the configuration A, the version with 2 ODI ports. In this case, the component is stimulated and the outputs are checked for correct operation.

The testbench generates a pattern on the receive ODI port 4

The design is configured to operate in the 400 bit pass through mode, and the output ODI 2 transmit port is checked to see that the data driven in on ODI receive input 4 is correctly passed back out with no errors.

The AXI4LITE register bus is driven to write and read registers within the IP component to setup the desired operation.

A counting pattern is created to drive into the receive ODI port at 512bits wide, with a mark space ratio on the tvalid to be of the correct average data rate to feed the 512 to 400 bit gearbox within the example design at the correct rate to produce the continuous 400bit at 400MHz bus. In this case the data has no real meaning at 400bits wide. The 400 to 512 gearbox within the example design then returns the data to a 512 bit bus with a mark space ratio that will give the expected average data rate.

A reference pattern is created separately that then checks that the output ODI data matches the expected value, i.e. a delayed in time, but identical pattern to what was sent into the design.

There are five Xilinx libraries that will have to be compiled inside Vivado for your simulator of choice. See the Xilinx Vivado help for details on how to compile simulation libraries for the simulator of your choice

The libraries required for the example design are "xpm", "unisim", "axis_register_slice_v1_1_20", "axi_register_slice_v2_1_20", "axi_amm_bridge_v1_0_10"

These should be compiled for Vivado 2019.2, the version supported by the M8132A FSP

dpu_caa_pw_b_example_4_2_data_400_loopback_tb.vhd

This testbench operates at the top level of the example design for the fpga b sandbox 2 configuration B IP component. This common FPGA B is used for both 4 ODI and 3 ODI plus 8 10GbE operation.

The testbench generates a pattern on the receive ODI port 1

The design is configured to operate in the 400 bit loop back mode, and the output ODI 1 transmit port is checked to see that the data driven in on ODI 1 receive input 1 is correctly looped back out with no errors

The AXI4Lite register bus is driven to write and read registers within the IP component to setup the desired operation.

A counting pattern is created to drive into the receive ODI port at 512bits wide, with a mark space ratio on the tvalid to be of the correct average data rate to feed the 512 to 400 bit gearbox within the example design at the correct rate to produce the continuous 400bit at 400MHz bus. In this case the data has no real meaning at 400bits wide. The 400 to 512 gearbox within the example design then returns the data to a 512 bit bus with a mark space ratio that will give the expected average data rate.

A reference pattern is created separately that then checks that the output ODI data matches the expected value, i.e. a delayed in time, but identical pattern to what was sent into the design.

There are five Xilinx libraries that will have to be compiled inside Vivado for your simulator of choice. See the Xilinx Vivado help for details on how to compile simulation libraries for the simulator of your choice

The libraries required for the example design are "xpm", "unisim", "axis_register_slice_v1_1_20", "axi_register_slice_v2_1_20", "axi_amm_bridge_v1_0_10"

These should be compiled for Vivado 2019.2, the version supported by the M8132A FSP

dpu_caa_pw_example_4_2_data_400_loopback_tb.vhd

This testbench operates at the top level of the example design for the fpga a sandbox 1 configuration B IP component. This FPGA A is used for the 3 ODI plus 8 10GbE Operation

The testbench generates a pattern on the receive ODI port 2

The design is configured to operate in the 400 bit loop back mode, and the output ODI 2 transmit port is checked to see that the data driven in on ODI 2 receive input is correctly looped back out with no errors

The AXI4Lite register bus is driven to write and read registers within the IP component to setup the desired operation.

A counting pattern is created to drive into the receive ODI port at 512bits wide, with a mark space ratio on the tvalid to be of the correct average data rate to feed the 512 to 400 bit gearbox within the example design at the correct rate to produce the continuous 400bit at 400MHz bus. In this case the data has no real meaning at 400bits wide. The 400 to 512 gearbox within the example design then returns the data to a 512 bit bus with a mark space ratio that will give the expected average data rate.

A reference pattern is created separately that then checks that the output ODI data matches the expected value, i.e. a delayed in time, but identical pattern to what was sent into the design.

There are five Xilinx libraries that will have to be compiled inside Vivado for your simulator of choice. See the Xilinx Vivado help for details on how to compile simulation libraries for the simulator of your choice

The libraries required for the example design are "xpm", "unisim", "axis_register_slice_v1_1_20", "axi_register_slice_v2_1_20", "axi_amm_bridge_v1_0_10"

These should be compiled for Vivado 2019.2, the version supported by the M8132A FSP

dpu_caa_pw_a_example_4_2_dpu2dpu_tb.vhd

This testbench operates at the top level of the example design for the fpga a sandbox 1 configuration A IP component.

This test bench is for the DPU2DPU link, and uses an XPM fifo at the top level of the testbench to feed back the dpu2dpu_1 channel from transmit back to receive.

The testbench drives the AXI4Lite register interface to configure the DPU2DPU transmit test pattern generator and receive test payload checker to create test payload for transmission, and then receive that data back via the loophrough fifo at the top level of the testbench

The testbench turns on the receiver payload checkers, and ensure the status reports they are running. The transmit pattern generators are then started, and the status is checked to ensure they started up.

The testbench then waits for a pre-determined time, and then checks that the transmitters had finished sending the configured amount of packets, by reporting they have changes from reporting running, to reporting stopped.

The receive packet count is then checked to ensure the receiver saw the correct number of packets. The receiver is then stopped, and the status checked to ensure they report stopped, and also to ensure no data check errors were reported.

There are five Xilinx libraries that will have to be compiled inside Vivado for your simulator of choice. See the Xilinx Vivado help for details on how to compile simulation libraries for the simulator of your choice

The libraries required for the example design are "xpm", "unisim", "axis_register_slice_v1_1_20", "axi_register_slice_v2_1_20", "axi_amm_bridge_v1_0_10"

These should be compiled for Vivado 2019.2, the version supported by the M8132A FSP

dpu_caa_pw_d2d_b_example_4_2_dpu2dpu_tb.vhd

This testbench is an equivalent of the dpu_caa_pw_a_example_4_2_dpu2dpu_tb.vhd but in this case for the fpga b sandbox 2 configuration B IP component

dpu_caa_pw_example_4_2_dpu2dpu_tb.vhd

This testbench is an equivalent of the dpu_caa_pw_a_example_4_2_dpu2dpu_tb.vhd but in this case is for the fpga a sandbox 1 configuration B IP component

Bus Widths and Sample Ordering

The most obvious difference in the two passthrough modes are the width of the busses.

The 512 bus is, as its name implies; a 512 bit bus, for passing whatever data arrived on a receive ODI interface, back to the corresponding pass through transmit ODI interface.

The data is un-touched. One option in the mux control block is to resize the ODI packets, but this does not change the data, just the packaging.

The 400 bit bus, as the name implies is 400bits, and is created by the 512 to 400 gearbox, with the assumption, that the original data before it was transmitted over ODI was a 400bit 400MHz bus.

As such, in the M8131A Digitizer module, it would be 40 samples of 10bits each.

Moreover, the earliest sample in time is placed in the lowest 10bits, i.e. bits 9 to 0 of the bus.

The latest sample in time is placed in the highest 10bit, ie. Bits 399 to 390 of the bus.

(assuming a bus numbered (399 down to 0) in VHDL speak)

The ODI bus at 512, is not only just 512 bits, but is ordered with the earlier sample at the top of the bus, and the latest sample at the bottom of the bus following the ODI standard.

If for instance the M8131A Digitiser Module is running in a decimated mode, say 8GSa/s where the sample size changes from 10 to 16 bits, before transmission on ODI, the samples are arranged to appear on the ODI interface bus with the earliest sample in time on the top bits, (511 to 496) and the latest sample in time on the bottom bits, (15 to 0), but there is no gearbox involved, the 16 bit samples sit cleanly within the 512 bit bus

Similarly, if the M8131A Digitiser Module is running in a DDC mode, where the samples are now pairs of 16bit I & Q samples the 32bit combined I&Q samples are transmitted on ODI in the clear, the 32bits sit within the 512 bit bus with no gearbox. The ODI order is the normal ODI setup, the earliest samples in time are placed at the top of the 512bit bus, the latest samples in time are placed at the bottom of the 512 bit bus

The following three diagrams show these bus width and sample arrangements

400Bit to 512Bit Gearbox and ODI Transmission

When the M8131A Digitiser is running in 16Gs/s mode with no decimation nor DDC, the samples within the Digitiser are 10bits, a 400 bit, 40 times super sampled bus runs at 400MHz continuously.

For transmission over ODI the 400bit bus with continuous data valid is gearboxed to a 512 bit bus with a mark space ratio on the data valid. Also, to follow the ODI standard, the samples are shifted to have the earliest in time placed at the top of the 512 bit bus, the oldest in time are placed at the bottom of the 512 bit bus.

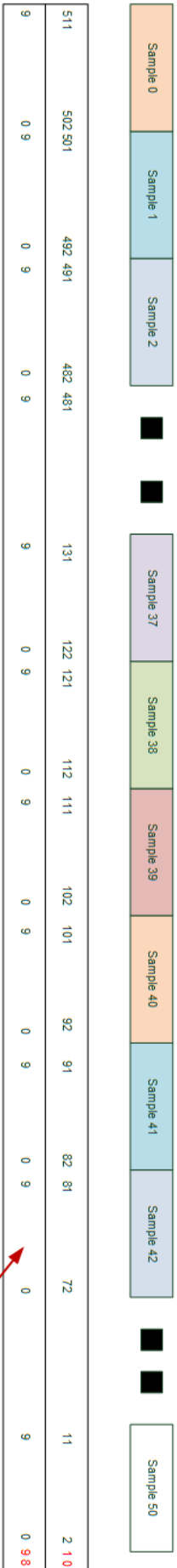
400bit and 512 bit gearbox arrangements

Inside the M8131A Digitiser

The 400bit, 40 samples @ 10 bit per sample bus The earliest sample is at the bottom of the bus, the latest sample is at the top of the bus



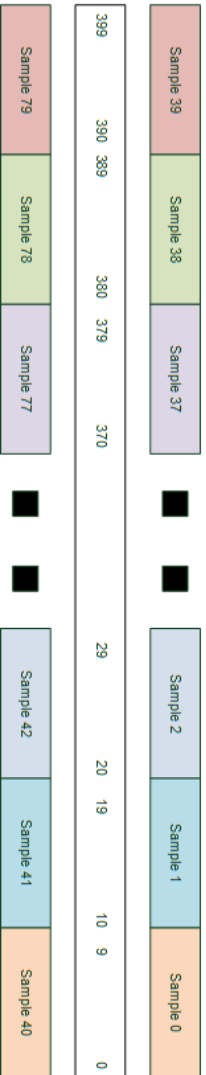
The 400 to 512 gearbox, as well as changing the bus width, it changes the sample order to follow the ODI earliest sample at the top, and latest sample at the bottom ordering



This is how the data is presented for transmission as an raw ODI stream, 512 bits with the earliest sample in time at the top of the bus

Inside the M8132A DSP

The 512 to 400 gearbox, as well as changing the bus width, it changes the sample order to recover the original Digitiser earliest sample at the bottom, and latest sample at the top ordering



This particular bit arrangement only exists in the first of the 5 repeating patterns of alignment of the 10 bit samples in the 512 bit bus.

512Bit and ODI Transmission

When a simple decimated mode is selected within the M8132A digitiser, there is no need for a gearbox, the 16bit samples can be placed directly into the 512bit ODI bus. The only alteration required for preparation for ODI transmission is to re-order

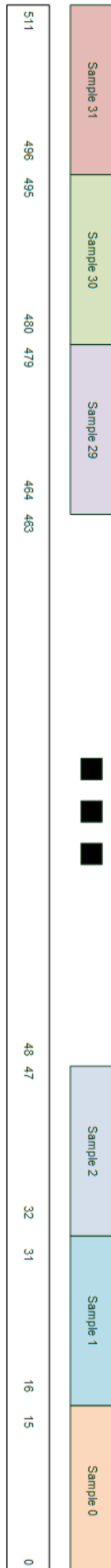
the samples to have the earliest samples in time placed at the top of the bus, and the latest samples in time placed at the bottom of the bus. The order of the samples on the bus is reversed, but the order of the bits within each sample is not changed.

The upper most bit within one sample is the most significant, the lower most bit within one sample is the least significant.

512 bit and ODI for 16 bit samples

Inside the M8131A Digitiser

The 512bit, 32 samples @ 16 bit per sample bus The earliest sample is at the bottom of the bus, the latest sample is at the top of the bus



Before passing to the ODI Interface, the sample order is reversed to follow the ODI earliest sample at the top, and latest sample at the bottom ordering

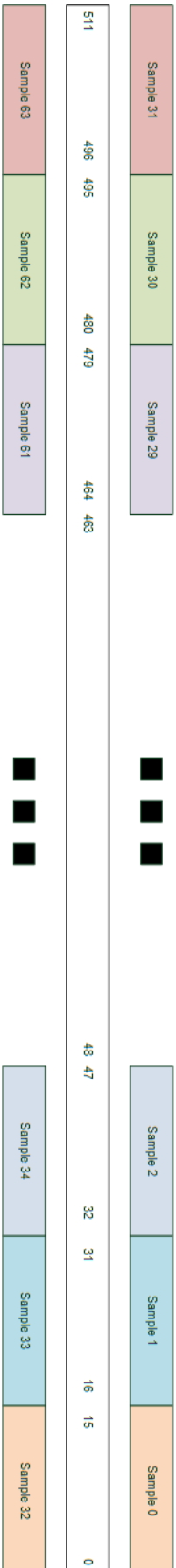


511	496 495	480 479	464 463	48 47	32 31	16 15	0
15	0 15	0 15	0 15	0 15	0 15	0 15	0

This is how the data is presented for transmission as an raw ODI stream, 512 bits with the earliest sample in time at the top of the bus

Inside the M8132A DSP

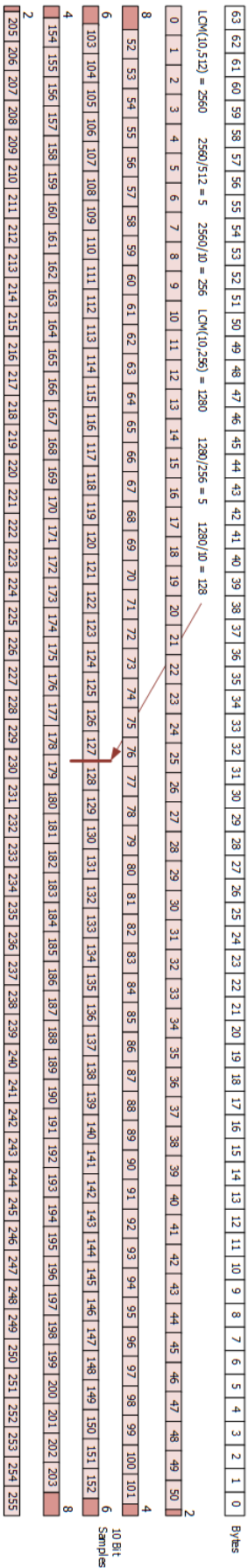
The feed of the 512 data bus, to the capture ram is reversed on a 16bit sample basis, to restore the orders of samples to the original M8131A Digitizer format



This diagram shows more details of the repeating pattern on the 512bit bus as produced by the 400 to 512 gearbox within the example design, and the M8131A Digitiser. If the 512 to 400 gearbox within the M8132A example design is used,

this gearboxing and sample reversal is all taken care of.

10bit Samples for M8131A/M8132A 512 bit ODI transmission



This diagram shows the alignment of 10 bit samples at 16GSa/s from a 400Bt @ 400MHz bus when gearboxed to a 512 bit bus for transmission over ODI

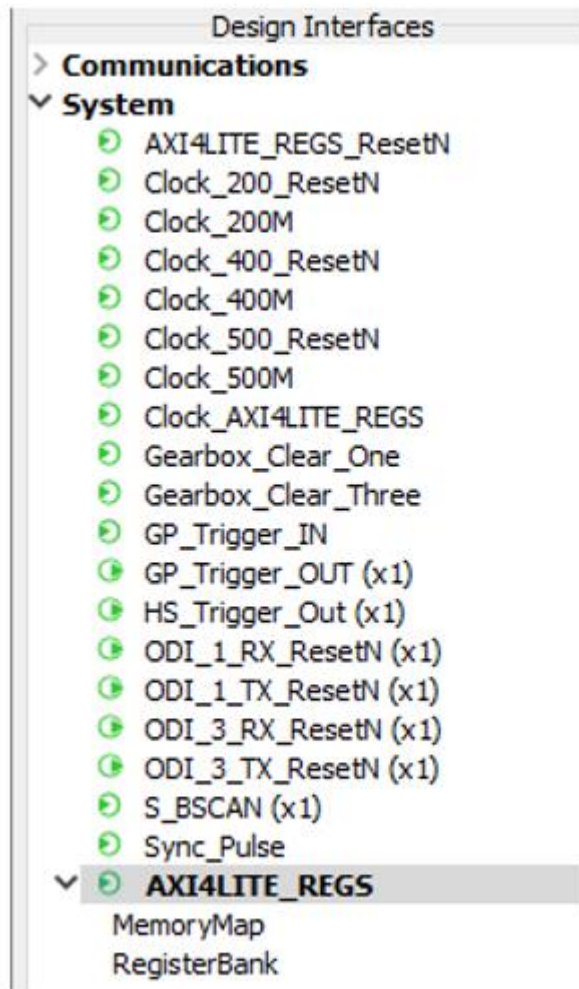
The diagram above shows how, for 10bit samples and a 512 bit bus, the sample to bus width alignment repeats on a 5 transaction basis. The M8131A is setup to create ODI packets of a length that match a multiple of $5 * 512$ bits, so that at the start of each ODI packet the MSB of the earliest sample is in the top bit of the ODI 512bit bus. The 512 to 400 gearbox in the example design expects this arrangement of data, and will not recover the 10bit samples at the 400bit output, if the ODI alignment is different.

M8132A Register Example

This simpler example shows the use of the Pathwave FPGA Register Bank so the SW examples can include use of the SCPI or C++ API commands to interrogate for the existence of registers, and how to interact with them. The example consists of one 4 register register bank, that is automatically connected to the AXI4LITE register bus. The 4 separate registers are named, scratch, increment, count, and enable_check.

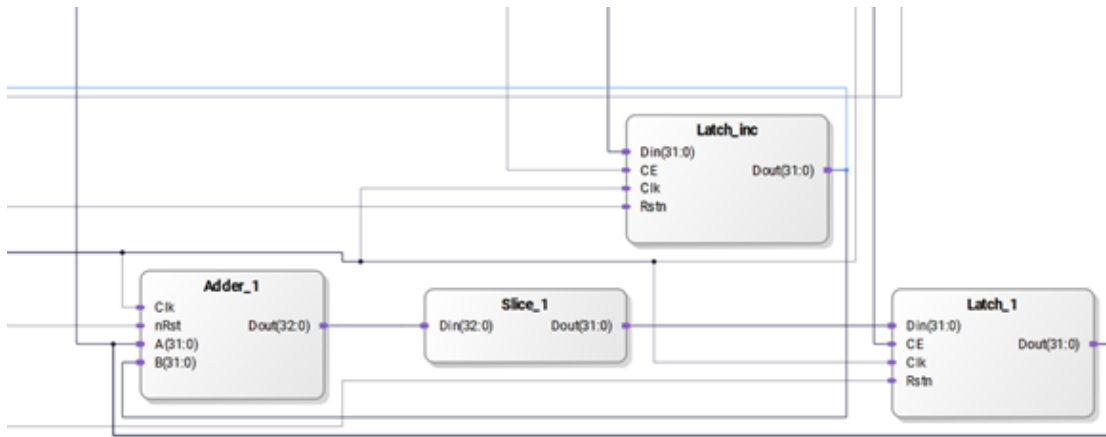
This design is built purely from Pathwave FPGA components from the IP Catalog

The Register bank components are found under the Design Interfaces section, then AXI4LITE_REGS, then Register Bank



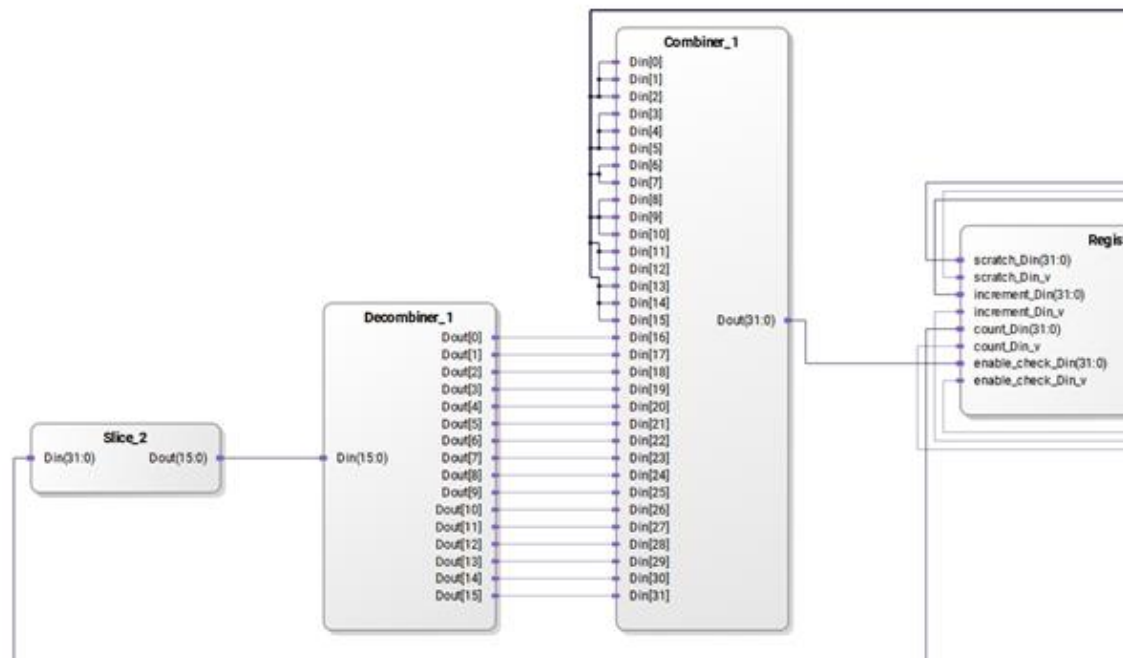
This design also uses the Clock_AXI4LITE_REGS signal and the AXI4LITE_REGS_ResetN signal, the appropriate clock and active low reset for this design which purely run on the 125M register clock.

The main area of the actual logic is shown here.



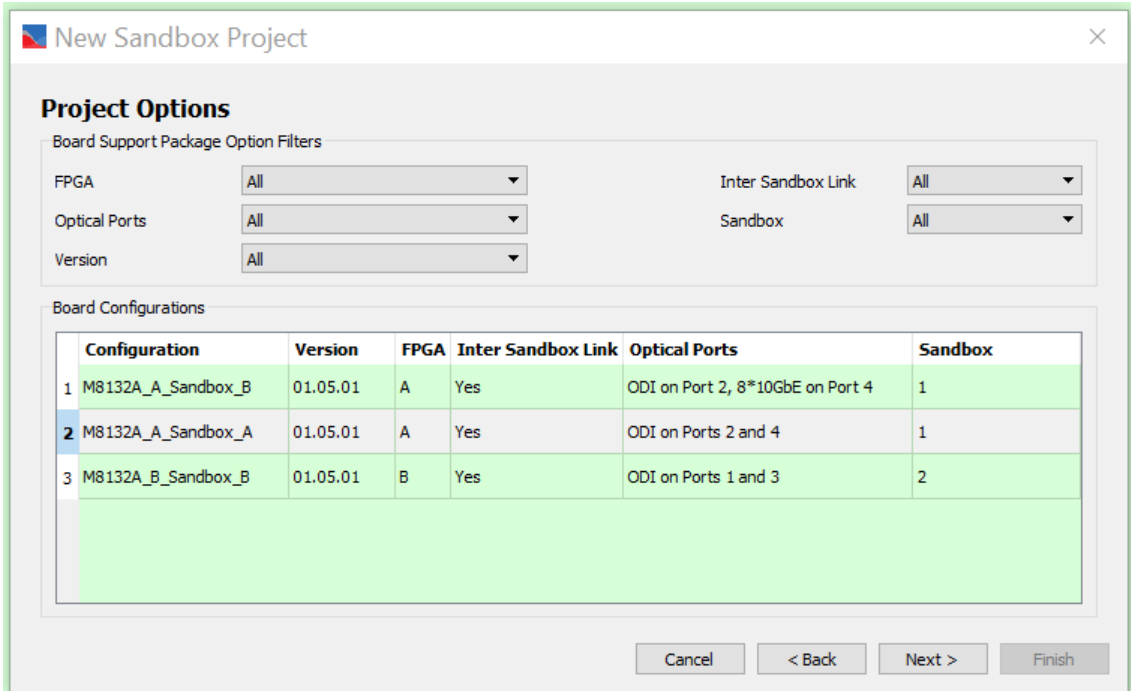
The Increment feeds into port Din of the Latch_inc block which is enabled by the increment_Dout_v signal from the Register bank. The Increment value feeds to port B of the Adder_1 component. The output of the adder is sliced from 33 bits to 32, and then latched in the Latch_1 component. The CE signal of the Latch_1 component is fed by the bottom bit of the scratch register. I.E when the value in the scratch register is odd, the accumulator will increment by the value in the increment register. When the value in the scratch register is even (i.e. bit 0 is a 0), then the latch will not be enabled, and the value of the accumulator will halt increasing.

The output of the Latch_1 component, Dout is fed back to the count register so it can be read as well as feeding the Adder_1 A input. The enable_check register is fed with two parts. The bottom 15 bits will all be 1 if the scratch register bottom bit is a 1, else the bottom 15 bits of the enable_check register will be 0 when the scratch register bottom bit is 0. The upper 16 bits of the enable check register are fed with a slice of the bottom 16 bits of the increment value

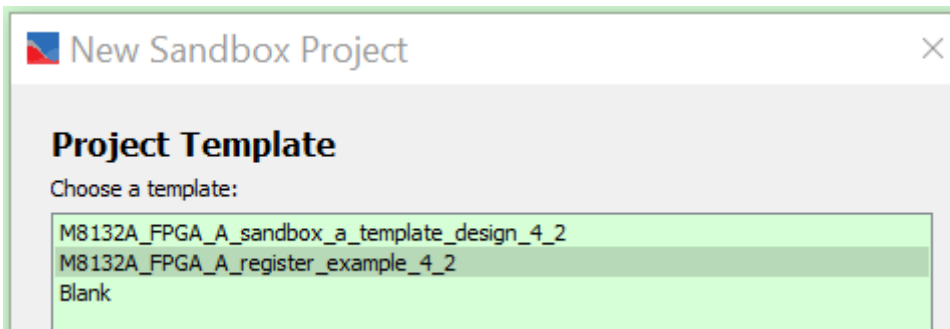


There are two versions of the same design supplied as template example projects.

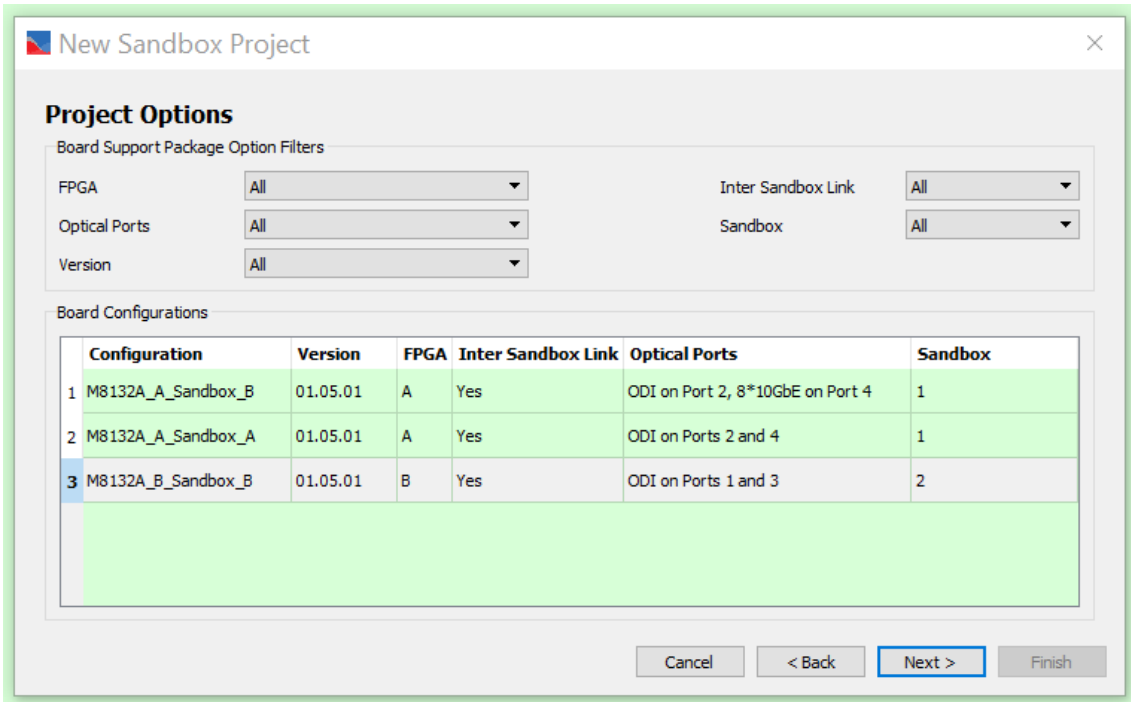
To create the FPGA A configuration A example register design for the 4 ODI setup, create a new Pathwave FPGA project, select the M8132A BSP, select the M8132A_A_Sandbox_A Configuration,



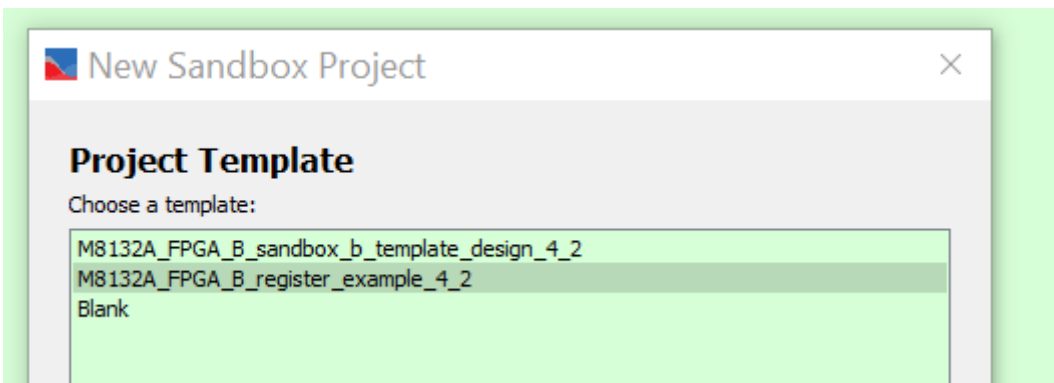
and then select the M8132A_FPGA_A_register_example_4_2 Project Template.



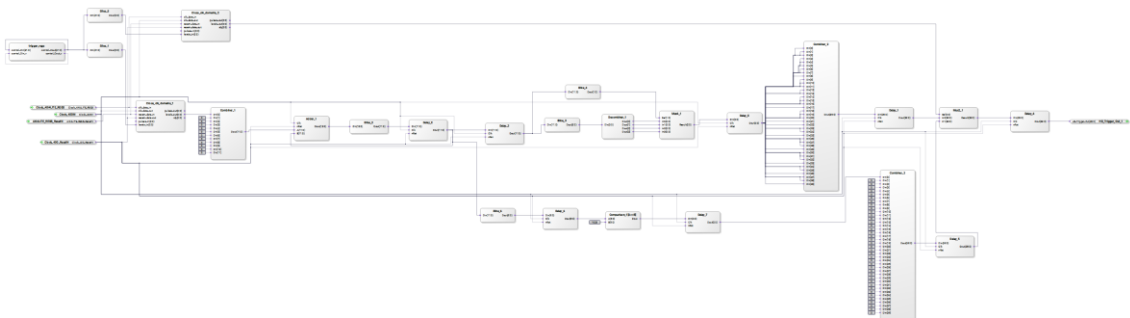
To create the FPGA B configuration B example register design for the 4 ODI setup, create a new Pathwave FPGA project, select the M8132A BSP, select the M8132A_B_Sandbox_B Configuration,



and then select the M8132A_FPGA_B_register_example_4_2 Project Template.

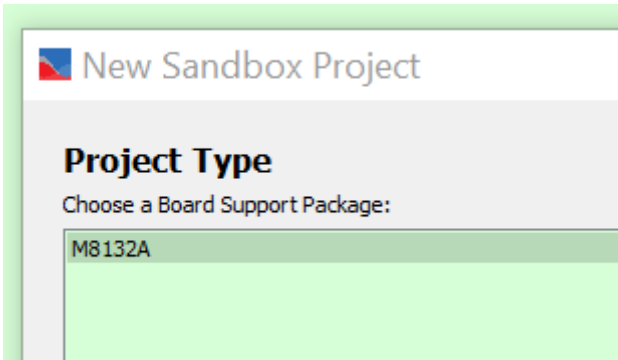


M8132A Trigger Out Example

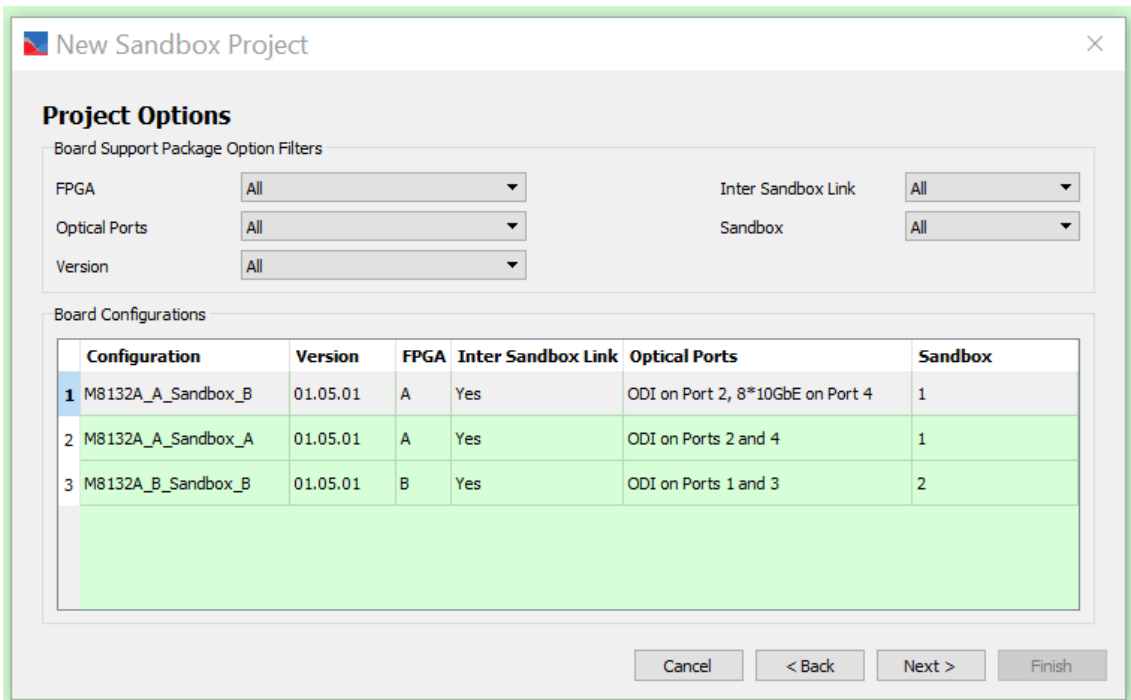


This alternative register based example shows use of the Pathwave FPGA Register Bank and the HS_Trigger_Out port.

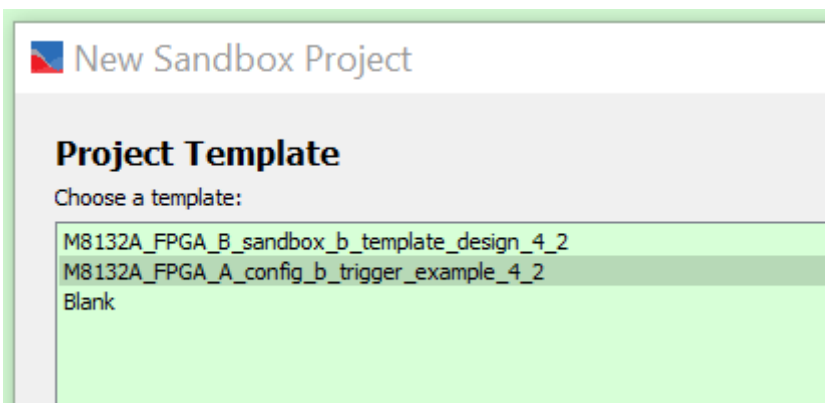
To create the FPGA A Configuration B example Register and Trigger design, create a new Pathwave FPGA project, select the M8132A BSP,



Select the M8132A_A_Sandbox_B Configuration,



and then select the M8132A_FPGA_A_config_b_trigger_example_4_2 Project Template



This design requires that the M8132A SFP is started up in the "3 ODI + 8 10GbE" mode, see the M8132A User Guide for details, (the default M8132A SFP startup gives the "4 ODI" mode).

If this design is built and downloaded into the FPGA A Customer Accessible Area, a few SCPI commands are required to control the module to allow the described behaviour to be observed on an oscilloscope connected to the "trig out" SMA output port.

See the M8132A User Guide for complete details of downloading and writing registers, and controlling the trigger output, but the poke and peek, and trigger control commands required are listed here for convenience.

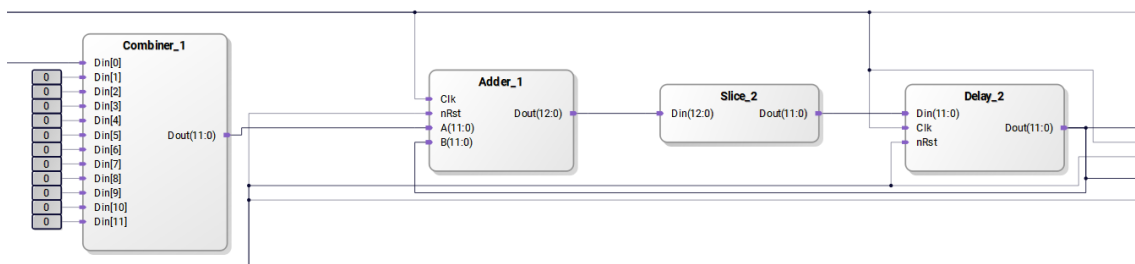
Command	Action performed
:instrument:sandbox1:poke 0,1	set the register at address 0 to the value 0x00000001
:instrument:sandbox1:poke 0,0	set the register at address 0 to the value 0x00000000
:instrument:sandbox1:poke 0,3	set the register at address 0 to the value 0x00000003
:trigger:output:source A	select FPGA A sandbox 1 as the source of the trigger output signal
:trigger:output:mode A,bypass	set the output trigger to be the unmodified 40 bit values sent from the FPGA A sandbox
:trigger:output:mode A,fsm	set the output trigger to be a pulse stretched version of the 40 bit values sent from the FPGA A sandbox

It has one register; (trigger_regs), with 2 active bits; (bits 1 and bit 0).

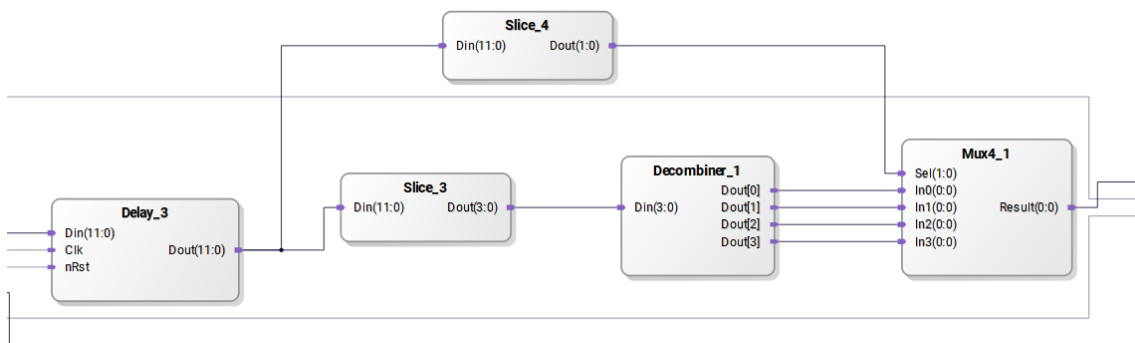
The registers run on the Clock_AXI4LITE_REGS and 125MHz. The main logic runs on Clock_400M at 400MHz, so Cross_clk_domains components are used to transfer the two control bits safely between the clock domains

In this case a 12 bit accumulator is setup to increment by 0 or 1 depending on the value of bit 0 of the trigger_regs register.

The accumulator is implemented by latching the output value of an adder which is set to add together the latched value, and a constant, 0 or 1



Then an arrangement of bit slices, decimbiners and a multiplexor presents a single bit selected from the top 4 bits of the accumulator value, picking up each of those 4 bits in turn once for each different value it presents.



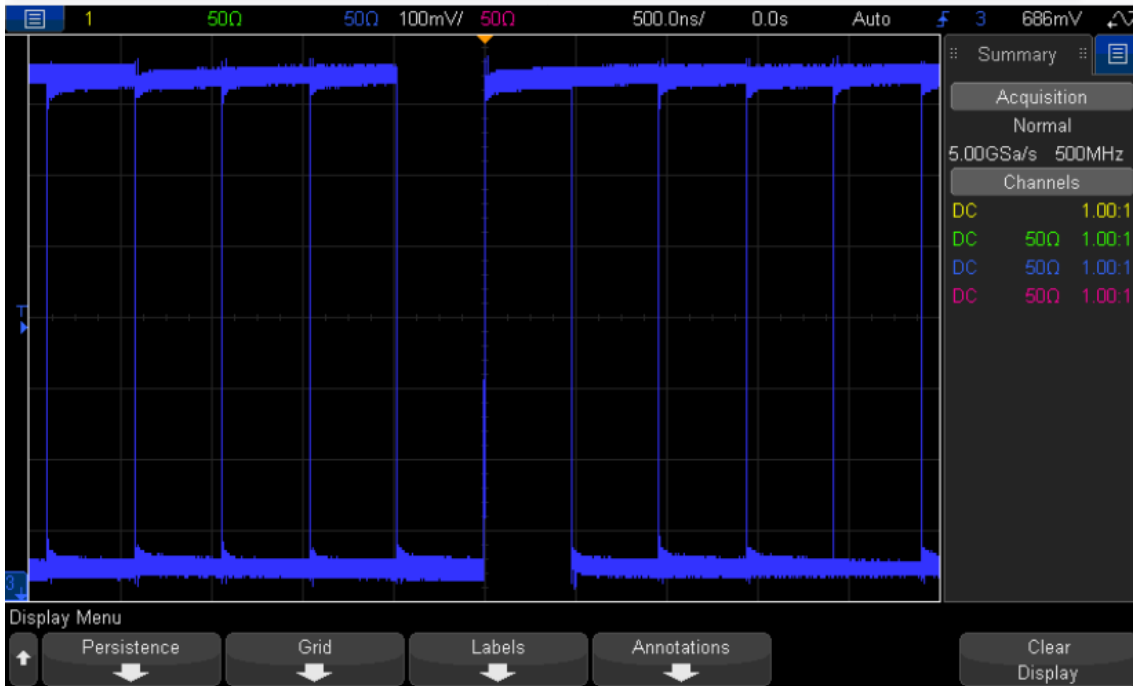
This value is then fanned out 40 times to create a slowly changing pattern of 1s and 0s for feeding out the 40 bit HS_TRIGGER_OUT bus.

In the static area, this 400MHz 40bit bus is serialised in a transceiver to drive out the front panel SMA connector "trig out" with a 16GHz serial bit rate.

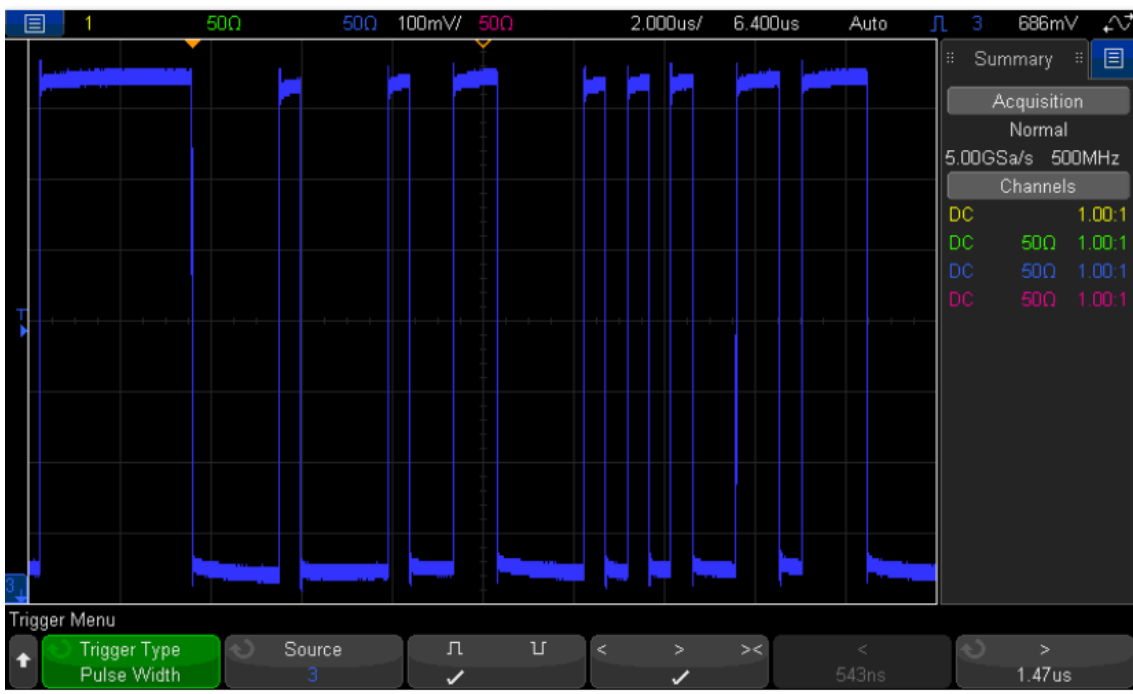
If a 1 is written to the bottom bit, the accumulator will start running, and a pattern can be observed on the "trig out" SMA.

Using the appropriate SCPI commands from the list above, select FPGA A sandbox 1 as the source of the trigger, and set the mode to be "bypass", and write the register to 0x00000001.

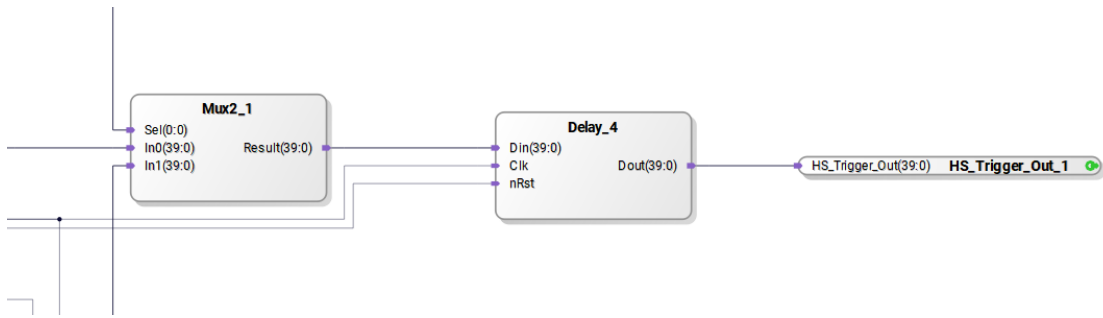
A simple rising edge trigger will display a waveform something similar to the one shown in the screenshot below



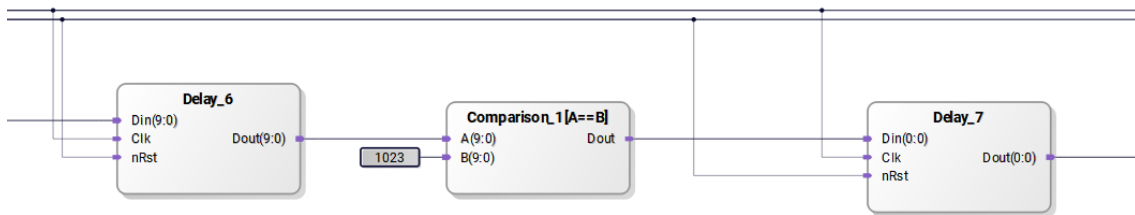
If an active high pulse width trigger is chosen, with a minimum pulse width of around 1.47us then a repeating 4 bit binary count can be observed



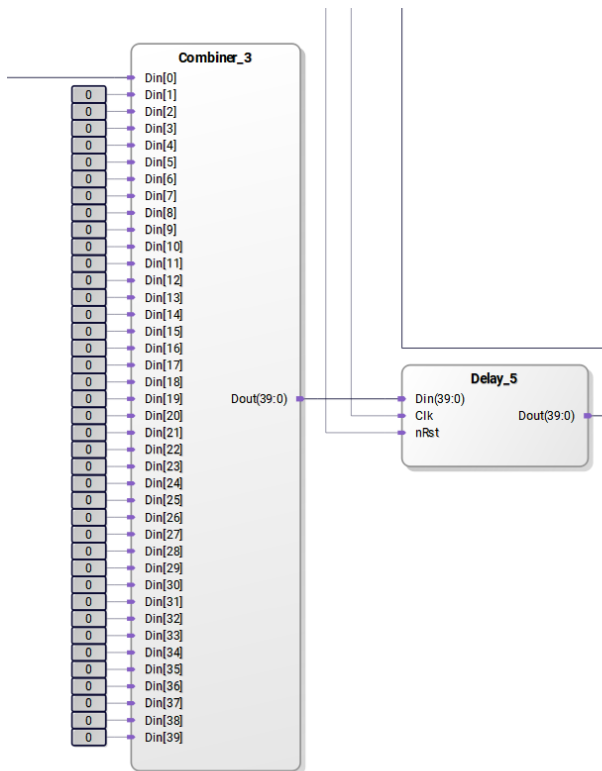
The 2nd bit of the control register, bit 1 controls the final output stage multiplexor



When this bit is set to a 1, a different pattern is presented on the 40 bit bus for serialization.
 The same accumulator output sliced to 10 bits and fed into a comparator.



This produces a '1' output from the comparator when the bottom 10 bits of the accumulator equal 0x03FF. This single bit signal which is mainly '0' with an occasional '1' is fed to a 40 bit combiner, where the other 39 bits are fixed at '0'.



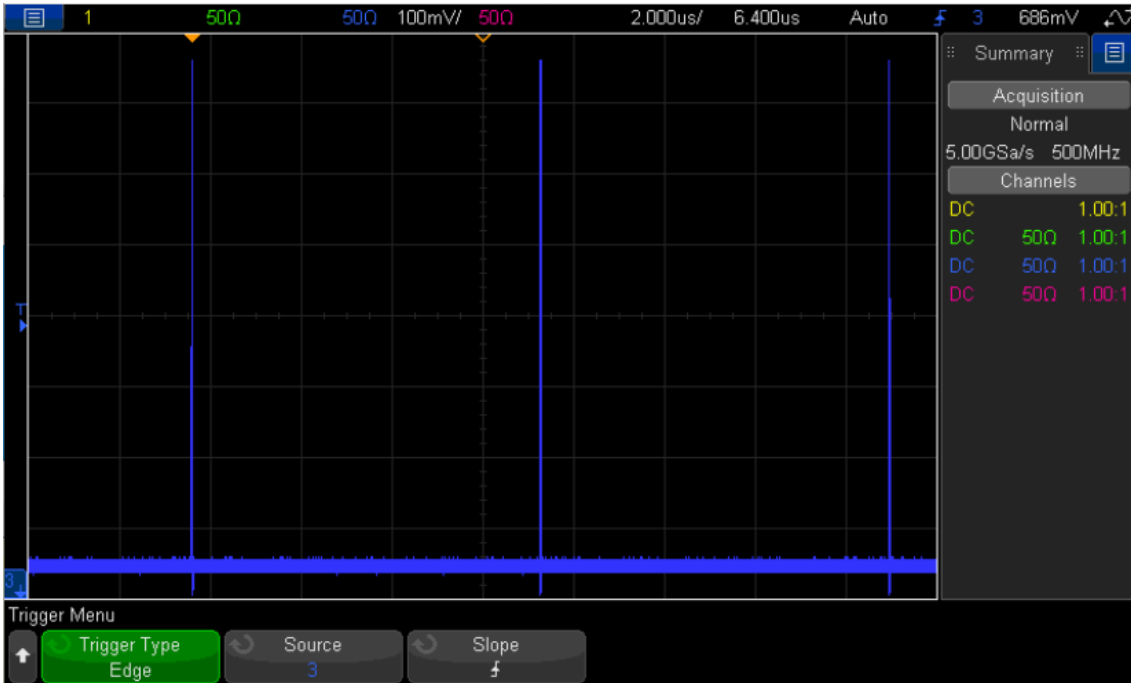
The output is fed through a flop flop, and is the 2nd input to the output multiplexer.

If the "trigger_regs" register is then set to the value 0x00000003 then the combination of the two bottom bits being a '1' will create a repeating single '1' on the serial output at the 16Gbit/s.

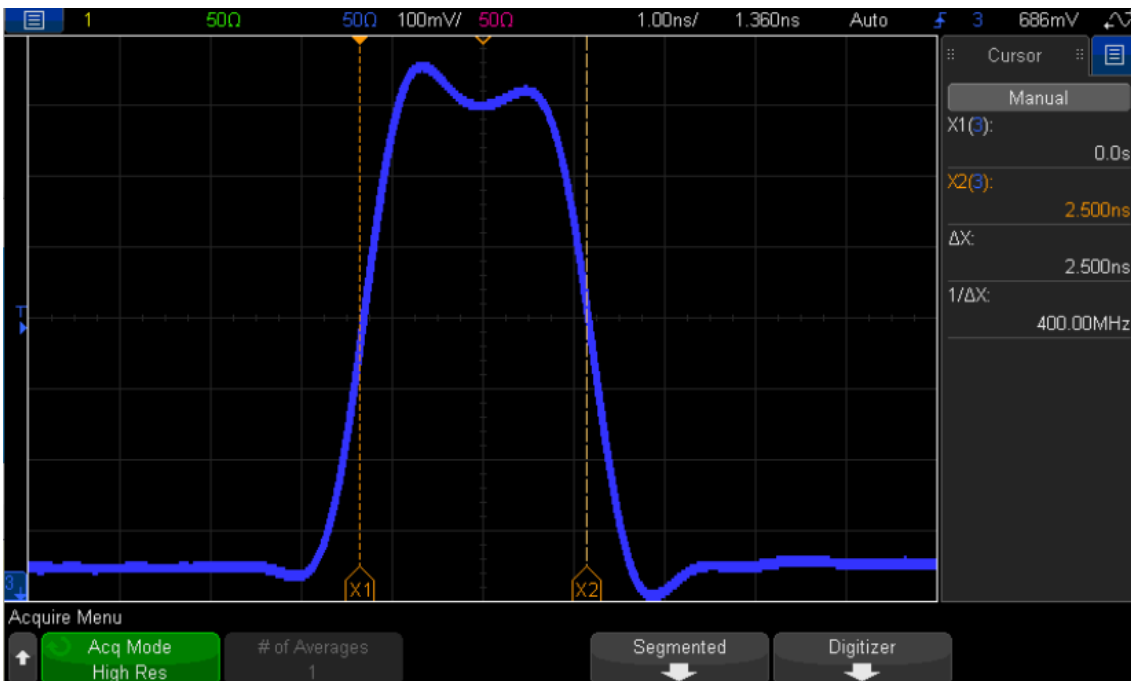
Use the appropriate SCPI command from the table above to set the register at address 0 to the value 0x00000003

A high speed oscilloscope would be required to capture this narrow pulse, (62.5ps).

However, if the trigger output mode is changed from 'bypass' to 'fsm' with the appropriate SCPI command, then the repeating pattern shown below can be observed



If the first pulse is zoomed into on the oscilloscope, then it can be observed that the pulses have been stretched to around 2.5ns



32GS/s TX/RX ODI Connection

The data rate capacity of the individual ODI connections is a little over 160Gb/s. This works fine for individual 16GSa/s at 10bits per sample data streams. However; for operating with M8131A digitisers that in some configurations can work at 32GSa/s 10bits per sample a 320Gb/s capacity link is required. This is achieved by running two ODI links in parallel. For the 32GSa/s digitiser input 1, ODI ports 1 and 3 are treated together as combined output ports with 320Gb/s capacity. For the 32GSa/s input 2, ODI ports 2 and 4 are treated together as combined output ports with 320Gb/s capacity. In the M8132A DSP module, this means it's corresponding ODI input ports 1 and 3 must work together to receive 320Gb/s, and ODI input ports 2 and 4 must work together to receive 320Gb/s. The existing mux control blocks in the example IP design show, in essence, how this could be achieved. At the moment, the two input channels work separately. A combined design where one mux control handles two ODI inputs is easily envisaged. The logic would be enhanced to drive both sync pulses to the 512 to 400 gearboxes from one controller. The controller would monitor both fifo depths to ensure both were above the minimum required level before issuing the sync pulse, if the Fifo Depth control method is required. If the sync pulse method is required, then the input sync pulse is passed to both 512 to 400 gearboxes at the same time. This would start the process of reading the 512 data from both receive fifos at the same time, driving both 512 to 400 gearboxes at the same time, producing two 400 bit buses in synchronism. The arrangement in the M8131A digitiser in 32GSa/s mode is that the even samples will be sent out on ODI port 1 (or ODI port 2 for the second sampler input) and the odd samples will be sent out on ODI port 3 (or ODI port 4 for the second sampler input). This means that in the M8132A DSP module, when handling a 320Gb/s stream from an M8131A Digitiser operating in 32GSa/s mode, the even samples will arrive on ODI channel 1 (or 2) and the odd samples will arrive on ODI channel 3 (or 4) assuming the ODI ports on the Digitizer and DSP module have port 1 connected to port 2, and port 3 connected to 3 etc.