euresys
Empowering Computer Vision

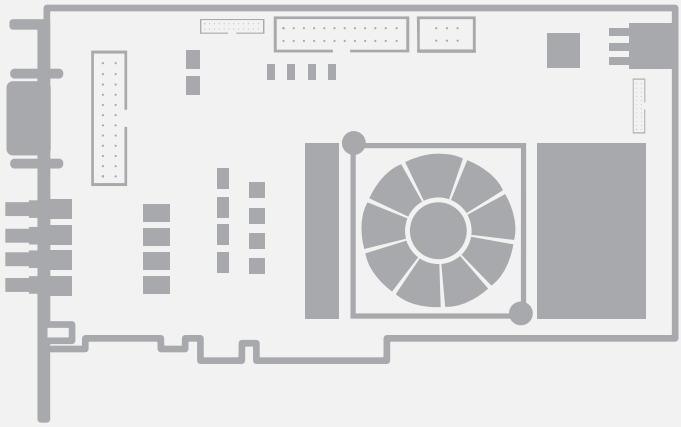
USER GUIDE

CustomLogic

CustomLogic 12.5 User Guide

3602 Coaxlink Octo

3603 Coaxlink Quad CXP-12



Terms of Use

This documentation is provided with CustomLogic 12.5.4 (doc build 2109).
www.euresys.com

Contents

1. Introduction to CustomLogic

1.1. Principles

CustomLogic allows users to add custom on-board image processing in the FPGA (Field Programmable Gate Array) of Euresys frame grabbers fitted with a "CustomLogic" firmware variant.

CustomLogic includes a design framework providing documented interfaces, which are used to interconnect the custom image processing functions with the frame grabber.

1.2. Availability

NOTE
The CustomLogic installation package is only available on request to your local Sales office.

CustomLogic is available for the following products and firmware variants:

3602 Coaxlink Octo

Firmware Variant	Description
1-camera, custom-logic	◻ CXP-6 DIN 4 CoaXPress interface ◻ One 1- or 2- or 4-connection area-scan camera ◻ 2 GB RAM DDR4 on-board memory ◻ 8-lane Gen 3 PCI Express interface

3603 Coaxlink Quad CXP-12

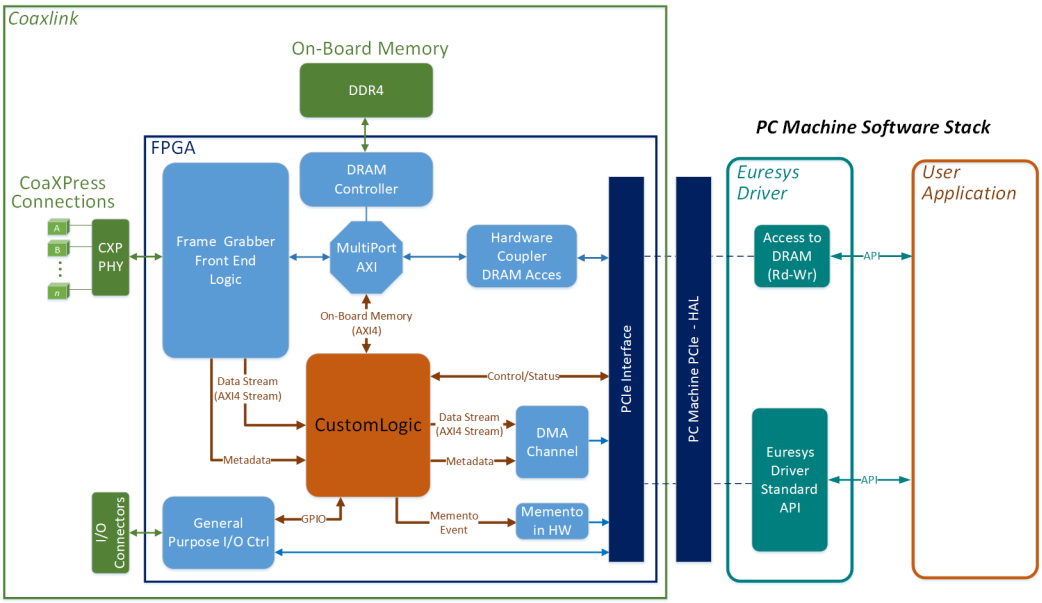
Firmware Variant	Description
1-camera, custom-logic	◻ CXP-12 HD-BNC 4 CoaXPress interface ◻ One 1- or 2- or 4-connection area-scan camera ◻ 2 GB RAM DDR4 on-board memory ◻ 8-lane Gen 3 PCI Express interface
4-camera, custom-logic	◻ CXP-12 HD-BNC 4 CoaXPress interface ◻ Four 1-connection area-scan cameras ◻ 2 GB RAM DDR4 on-board memory ◻ 8-lane Gen 3 PCI Express interface

1.3. Framework

A Coaxlink framework provides the following built-in modules:

1. Full featured CoaXPress frame grabber.
2. On-board memory interface.
3. PCI Express interface with a DMA back-end channel.
4. Memento in Hardware event logging system.
5. User registers access via Euresys Driver API.

Sample block diagram



Coaxlink CustomLogic model

2. CustomLogic Interfaces

2.1. Global Signals

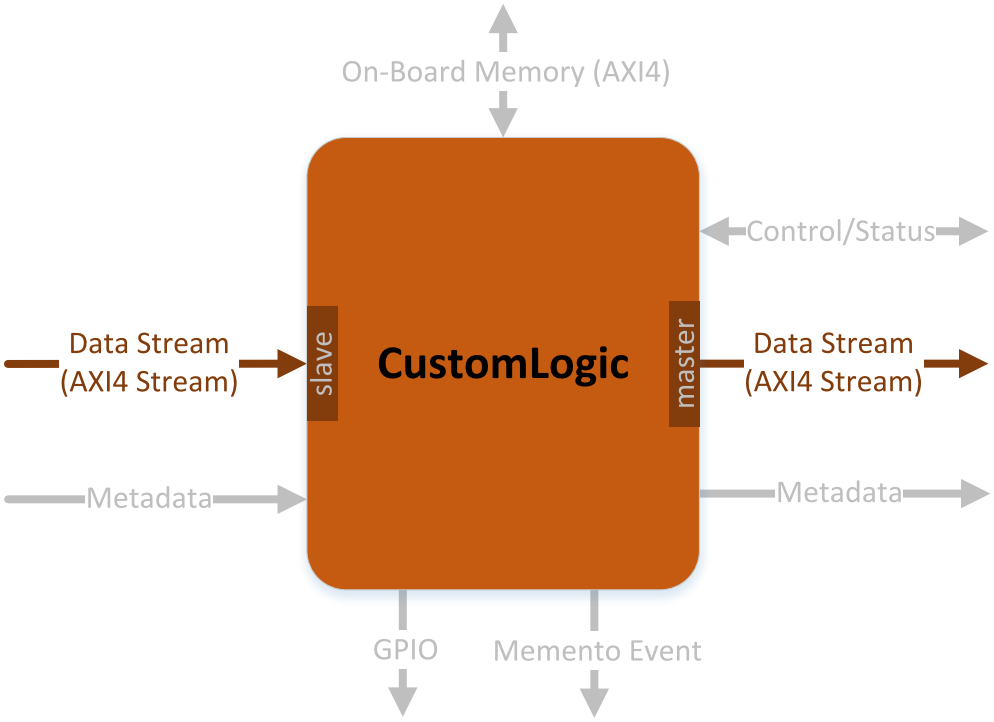
All available interfaces are in the same clock domain of 250 MHz and the *CustomLogic* global signals are the following:

Signal	Direction	Description
clk250	Input	250 MHz clock source common to all *CustomLogic* interfaces.
srst250	Input	Synchronous reset (clk250) asserted during a PCI Express reset.

2.2. Data (Pixel) Stream Interface

The Data Stream interface is based on the AMBA AXI4-Stream Protocol Specification:

- At the slave side, the *CustomLogic* receives images acquired from a CoaXPress Device (for example a CoaXPress camera)
- At the master side, the Data Stream interface transfers the resulting images/data generated by the *CustomLogic* to the PCI Express DMA Back-End channel.



Slave interface signals

Signal	Width	Direction	Description
s_axis_aresetn	N*1	Input	ARESETn resets the AXI4-Stream interface. This pulse is asserted when a Stop Acquisition command (`DSStopAcquisition`) is executed. This signal should be used to clear the *CustomLogic* internal Data Stream pipeline.
s_axis_tvalid	N*1	Input	TVALID indicates that a corresponding master interface is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
s_axis_tready	N*1	Output	TREADY indicates that the *CustomLogic* can accept a transfer in the current cycle.
s_axis_tdata	N*W	Input	TDATA is the primary payload that is used to provide the data passing across the interface.
s_axis_tuser	N*4	Input	TUSER is user defined sideband information that can be transmitted alongside the data stream. The TUSER content is encoded as follows: □ TUSER [0] => Start-of-Frame (SOF) □ TUSER [1] => Start-of-Line (SOL) □ TUSER [2] => End-of-Line (EOL) □ TUSER [3] => End-of-Frame (EOF)

NOTE

In the Width column: "*N*" refers to the total number of interface slots, which is the number of devices/cameras supported by the *CustomLogic* variant, and "*W*" refers to STREAM_DATA_WIDTH, the stream data width per device/camera.

- **3602 Coaxlink Octo** (1-camera, custom-logic) => *N* = 1; *W* = 128 ;
- **3603 Coaxlink Quad CXP-12** (1-camera, custom-logic) => *N* = 1; *W* = 256;
- **3603 Coaxlink Quad CXP-12** (4-camera, custom-logic) => *N* = 4; *W* = 64;

Master interface signals

Signal	Width	Direction	Description
m_axis_tvalid	N*1	Output	TVALID indicates that the *CustomLogic* is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
m_axis_tready	N*1	Input	TREADY indicates that the PCI Express DMA Back-End can accept a transfer in the current cycle.
m_axis_tdata	N*W	Output	TDATA is the primary payload that is used to provide the data passing across the interface.
m_axis_tuser	N*4	Output	TUSER is user defined sideband information that can be transmitted alongside the data stream. The TUSER content is encoded as follows: □ axis_tuser_out[0] => Start-of-Buffer (SOB) □ axis_tuser_out[1] => *Reserved* □ axis_tuser_out[2] => *Reserved* □ axis_tuser_out[3] => End-of-Buffer (EOB)

NOTE

In the Width column: "*N*" refers to the total number of interface slots, which is the number of devices/cameras supported by the *CustomLogic* variant, and "*W*" refers to STREAM_DATA_WIDTH, the stream data width per device/camera.

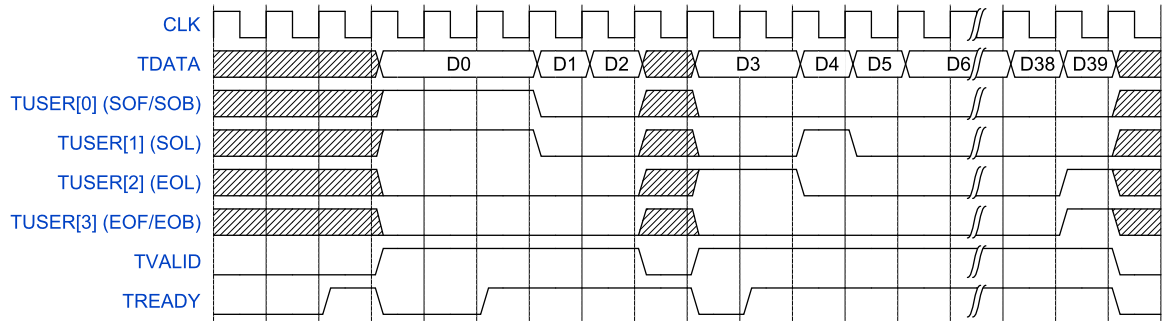
- **3602 Coaxlink Octo** (1-camera, custom-logic) => *N* = 1; *W* = 128 ;
- **3603 Coaxlink Quad CXP-12** (1-camera, custom-logic) => *N* = 1; *W* = 256;
- **3603 Coaxlink Quad CXP-12** (4-camera, custom-logic) => *N* = 4; *W* = 64;

NOTE

At the *CustomLogic* master side, the `m_axis_tuser` signal has the function of controlling the PCI Express DMA Back-End. The flags carried by `m_axis_tuser` are interpreted as follows:

- *Start-of-Buffer*: A cycle containing this flag starts a new buffer.
- *End-of-Buffer*: A cycle containing this flag ends a buffer even if it still has available space to accommodate new transfers.

Timing diagram



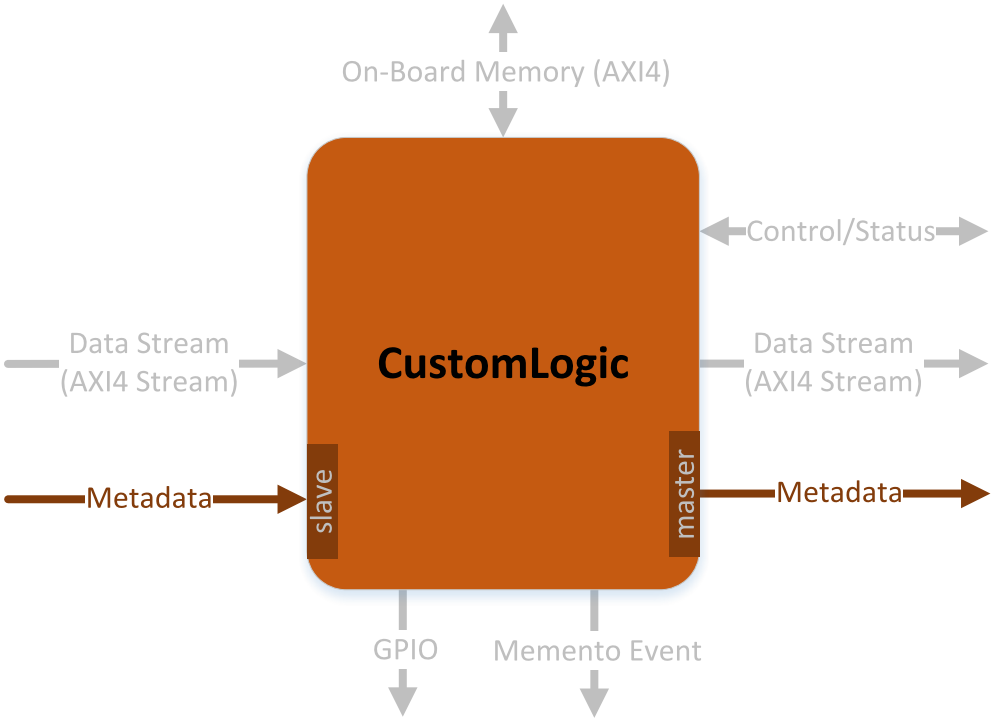
TVALID/TREADY handshake and TUSER flags timing diagram

In this example, we consider that the *LinePitch* is 64 bytes (4 transfer cycles of 16 bytes each) and the full frame is composed of 10 lines/packet.

For more information about the AXI4-Stream Protocol, please refer to Xilinx “AXI Reference Guide (UG1037)” at www.xilinx.com and “AMBA AXI4-Stream Protocol Specification” at www.amba.com.

2.3. Metadata Interface

The CoaXPress Image Header generated by the CoaXPress Device is exposed at the Metadata interface. In addition to the CoaXPress Image Header, the Metadata interface also provides information regarding pixel alignment, time-stamp...



Interface signals

At the slave side, the Metadata interface presents the prefix *s* and has the direction *Input*.

At the master side, it presents the prefix *the prefix m and has the direction Output.*

There are two groups of signals in the Metadata interface:

- *CoaXPress Image Header* group – signals containing a copy of the CoaXPress Rectangular Image Header issued by the CoaXPress Device.
- *CustomLogic* group – signals informing time-stamp and data stream characteristics.

CoaXPress Image Header group

Signal	Width	Description
(m/s)_mdata_StreamId	N*8	Unique stream ID.
(m/s)_mdata_SourceTag	N*16	16 bit source image index. Incremented for each transferred image, wraparound to 0 at 0xFFFF. The same number shall be used by each stream containing data relating to the same image.
(m/s)_mdata_Xsize	N*24	24 bit value representing the image width in pixels.
(m/s)_mdata_Xoffs	N*24	24 bit value representing the horizontal offset in pixels of the image with respect to the left hand pixel of the full Device image.
(m/s)_mdata_Ysize	N*24	24 bit value representing the image height in pixels. This value shall be set to 0 for line scan images.
(m/s)_mdata_Yoff	N*24	24 bit value representing the vertical offset in pixels of the image with respect to the top line of the full Device image.
(m/s)_mdata_DsizeL	N*24	24 bit value representing the number of data words per image line.
(m/s)_mdata_PixelF	N*16	16 bit value representing the pixel format.
(m/s)_mdata_TapG	N*16	16 bit value representing the tap geometry.
(m/s)_mdata_Flags	N*8	Image flags.

NOTE

In the Width column: "*N*" refers to the total number of interface slots, which is the number of devices/cameras supported by the *CustomLogic* variant.

- **3602 Coaxlink Octo** (1-camera, custom-logic) => *N* = 1;
- **3603 Coaxlink Quad CXP-12** (1-camera, custom-logic) => *N* = 1;
- **3603 Coaxlink Quad CXP-12** (4-camera, custom-logic) => *N* = 4;

NOTE

The descriptions are excerpts from CoaXPress Standard Version 2.0.

CustomLogic group

Signal	Width	Description
(m/s)_mdata_Timestamp	N*32	Timestamp of the Device's readout start event.
(m/s)_mdata_PixProcFlgs	N*8	Pixel processing flags: □ PixProcFlgs[0] => RGB to BGR swap enabled. □ PixProcFlgs[1] => MSB pixel alignment enabled. □ PixProcFlgs[2] => Packed acquisition enabled. □ PixProcFlgs[7:4] => LUT configuration.
(m/s)_mdata_Status	N*32	32-bit vector that can be used by the *CustomLogic* to report its status.

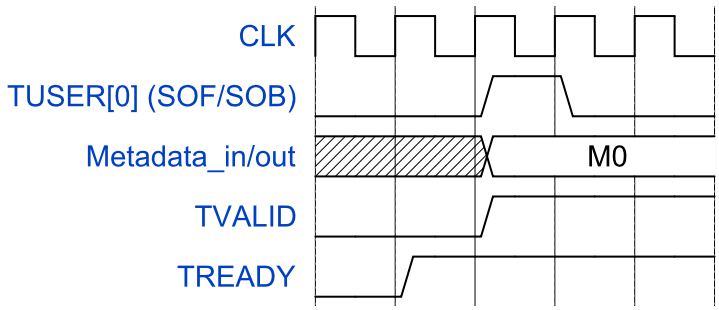
NOTE
At the slave side the Status value is 0x00000000.

NOTE

In the Width column: "*N*" refers to the total number of interface slots, which is the number of devices/cameras supported by the *CustomLogic* variant.

- **3602 Coaxlink Octo** (1-camera, custom-logic) => *N* = 1;
- **3603 Coaxlink Quad CXP-12** (1-camera, custom-logic) => *N* = 1;
- **3603 Coaxlink Quad CXP-12** (4-camera, custom-logic) => *N* = 4;

Timing diagram

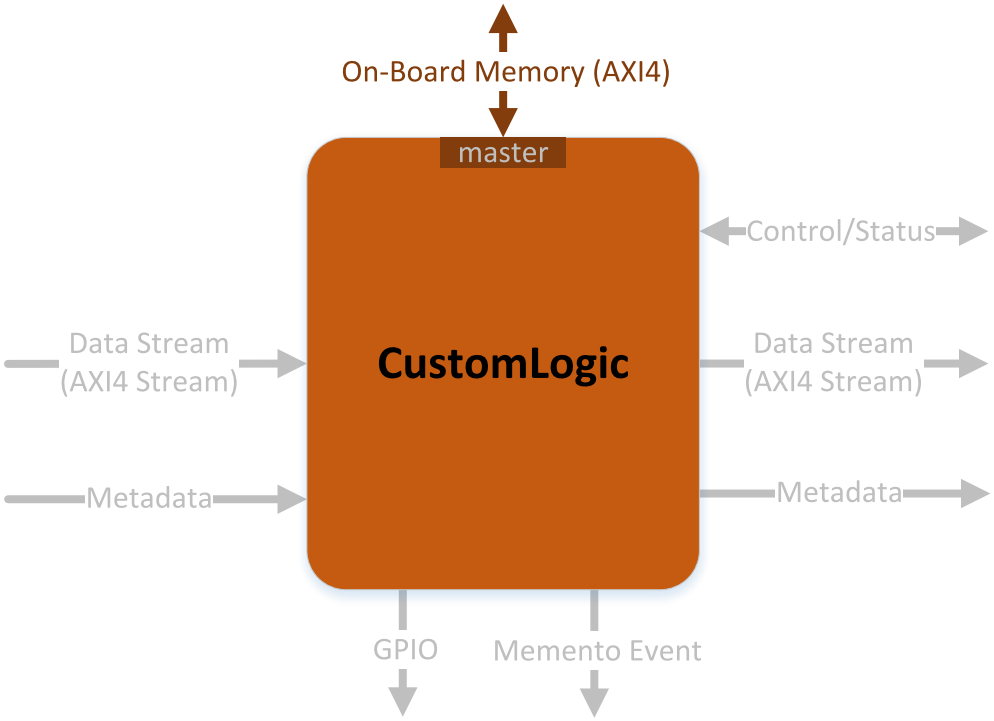


Metadata timing diagram

The Metadata signals are updated every time the *TUSER* flag SOF/SOB is asserted.

2.4. On-Board Memory Interface

The on-board memory interface is based on the AMBA AXI4-Stream Protocol Specification.

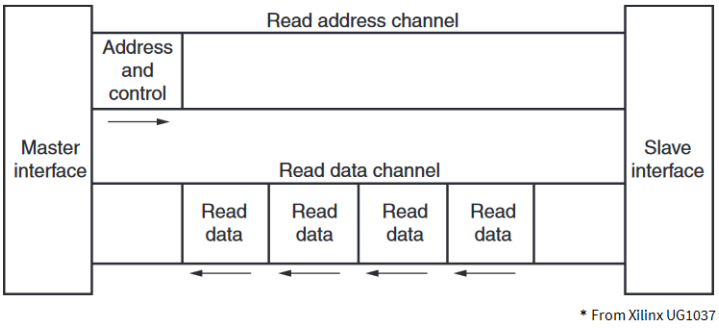


The AXI4 is a memory-mapped interface that consists of five channels:

- Write Address Channel
- Write Data Channel
- Write Response Channel
- Read Address Channel
- Read Data Channel

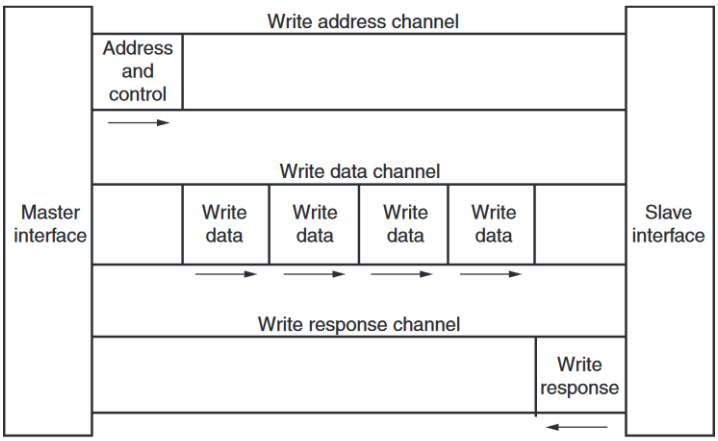
Data can move in both directions between the master and slave simultaneously, and data transfer sizes can vary. The limit in AXI4 is a burst transaction of up to 256 data transfers.

AXI4 channel architecture



* From Xilinx UG1037

Channel Architecture of Reads



* From Xilinx UG1037

Channel Architecture of Writes

AXI4 signals description

The following sections briefly describe the AXI4 signals.

> **NOTE**
> For a complete view of signal, interface requirements and transaction attributes, please refer to AMBA AXI and ACE Protocol Specification document at www.amba.com.

Master interface global signals

Signal	Width	Direction	Description
m_axi_resetn	1	Input	RESETn resets the AXI4 interface.

Master write address channel interface signals

Signal	Width	Direction	Description
m_axi_awaddr	32	Output	Write address. The write address gives the address of the first transfer in a write burst transaction.
m_axi_awlen	8	Output	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. Burst_Length = AWLEN[7:0] + 1
m_axi_awsize	3	Output	Burst size. This signal indicates the size of each transfer in the burst. Burst_Size = 2^AWSIZE[2:0]
m_axi_awburst	2	Output	Burst type. The burst type and the size information, determine how the address for each transfer within the burst is calculated. Burst_Type: "00" = FIXED; "01" = INCR; "10" = WRAP
m_axi_awlock	1	Output	Lock type. Provides additional information about the atomic characteristics of the transfer. Atomic_Access: '0' Normal; '1' Exclusive
m_axi_awcache	4	Output	Memory type. This signal indicates how transactions are required to progress through a system. Memory_Attributes: □ AWCACHE[0] Bufferable □ AWCACHE[1] Cacheable □ AWCACHE[2] Read-allocate □ AWCACHE[3] Write-allocate
m_axi_awprot	3	Output	Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. Access_Permissions: □ AWPROT[0] Privileged □ AWPROT[1] Non-secure □ AWPROT[2] Instruction

Signal	Width	Direction	Description
m_axi_awqos	4	Output	Quality of Service, QoS. The QoS identifier sent for each write transaction. Quality_of_Service: Priority level
m_axi_awvalid	1	Output	Write address valid. This signal indicates that the channel is signaling valid write address and control information.
m_axi_awready	1	Input	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.

NOTE
The descriptions are excerpts from AMBA AXI and ACE Protocol Specification.

Master write data channel interface signals

Signal	Width	Direction	Description
m_axi_wdata	W	Output	Write data.
m_axi_wstrb	W/8	Output	Write strobes. This signal indicates which byte lanes hold valid data. There is one write strobe bit for each eight bits of the write data bus.
m_axi_wlast	1	Output	Write last. This signal indicates the last transfer in a write burst.
m_axi_wvalid	1	Output	Write valid. This signal indicates that valid write data and strobes are available.
m_axi_wready	1	Input	Write ready. This signal indicates that the slave can accept the write data.

NOTE

In the Width column: "*W*" refers to MEMORY_DATA_WIDTH, the data width of the write data channel.

- **3602 Coaxlink Octo** (1-camera, custom-logic) => *W* = 128 ;
- **3603 Coaxlink Quad CXP-12**(1-camera, custom-logic) => *W* = 256;
- **3603 Coaxlink Quad CXP-12** (4-camera, custom-logic) => *W* = 256;

NOTE
The descriptions are excerpts from AMBA AXI and ACE Protocol Specification.

Master write response channel interface signals

Signal	Width	Direction	Description
m_axi_bresp	2	Input	Write response. This signal indicates the status of the write transaction. Response: □ "00" = OKAY □ "01" = EXOKAY □ "10" = SLVERR □ "11" = DECERR
m_axi_bvalid	1	Input	Write response valid. This signal indicates that the channel is signaling a valid write response.
m_axi_bready	1	Output	Response ready. This signal indicates that the master can accept a write response.

NOTE
The descriptions are excerpts from AMBA AXI and ACE Protocol Specification.

For m_axi_bresp:

- *OKAY*: Normal access success. Indicates that a normal access has been successful. Can also indicate an exclusive access has failed. See OKAY, normal access success.
- *EXOKAY*: Exclusive access okay. Indicates that either the read or write portion of an exclusive access has been successful.
- *SLVERR*: Slave error. Used when the access has reached the slave successfully, but the slave wishes to return an error condition to the originating master.
- *DECERR*: Decode error. Generated, typically by an interconnect component, to indicate that there is no slave at the transaction address.

Master read address channel interface signals

Signal	Width	Direction	Description
m_axi_araddr	32	Output	Read address. The read address gives the address of the first transfer in a read burst transaction.
m_axi_arlen	8	Output	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. Burst_Length = ARLEN[7:0] + 1
m_axi_arsize	3	Output	Burst size. This signal indicates the size of each transfer in the burst. Burst_Size = 2^ARSIZE[2:0]
m_axi_arburst	2	Output	Burst type. The burst type and the size information, determine how the address for each transfer within the burst is calculated. Burst_Type: "00" = FIXED; "01" = INCR; "10" = WRAP
m_axi_arlock	1	Output	Lock type. Provides additional information about the atomic characteristics of the transfer. Atomic_Access: '0' Normal; '1' Exclusive
m_axi_arcache	4	Output	Memory type. This signal indicates how transactions are required to progress through a system. Memory_Attributes: □ ARCACHE[0] Bufferable □ ARCACHE[1] Cacheable □ ARCACHE[2] Read-allocate □ ARCACHE[3] Write-allocate
m_axi_arprot	3	Output	Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. Access_Permissions: □ ARPROT[0] Privileged □ ARPROT[1] Non-secure □ ARPROT[2] Instruction

Signal	Width	Direction	Description
m_axi_arqos	4	Output	Quality of Service, QoS. The QoS identifier sent for each write transaction. Quality_of_Service: Priority level
m_axi_arvalid	1	Output	Read address valid. This signal indicates that the channel is signaling valid read address and control information.
m_axi_arready	1	Input	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals.

NOTE
The descriptions are excerpts from AMBA AXI and ACE Protocol Specification.

Master read data channel interface signals

Signal	Width	Direction	Description
m_axi_rdata	W	Input	Read data.
m_axi_rresp	2	Input	Read response. This signal indicates the status of the read transfer. Response: "00" = OKAY; "01" = EXOKAY; "10" = SLVERR; “11” = DECERR
m_axi_rlast	1	Input	Read last. This signal indicates the last transfer in a read burst.
m_axi_rvalid	1	Input	Read valid. This signal indicates that the channel is signaling the required read data.
m_axi_rready	1	Output	Read ready. This signal indicates that the master can accept the read data and response information.

NOTE

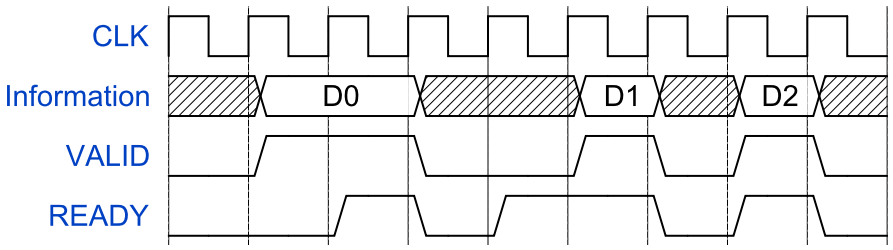
In the Width column: "*W*" refers to MEMORY_DATA_WIDTH, the data width of the write data channel.

- **3602 Coaxlink Octo** (1-camera, custom-logic) => *W* = 128 ;
- **3603 Coaxlink Quad CXP-12**(1-camera, custom-logic) => *W* = 256;
- **3603 Coaxlink Quad CXP-12** (4-camera, custom-logic) => *W* = 256;

NOTE

The descriptions are excerpts from AMBA AXI and ACE Protocol Specification.

Timing diagram



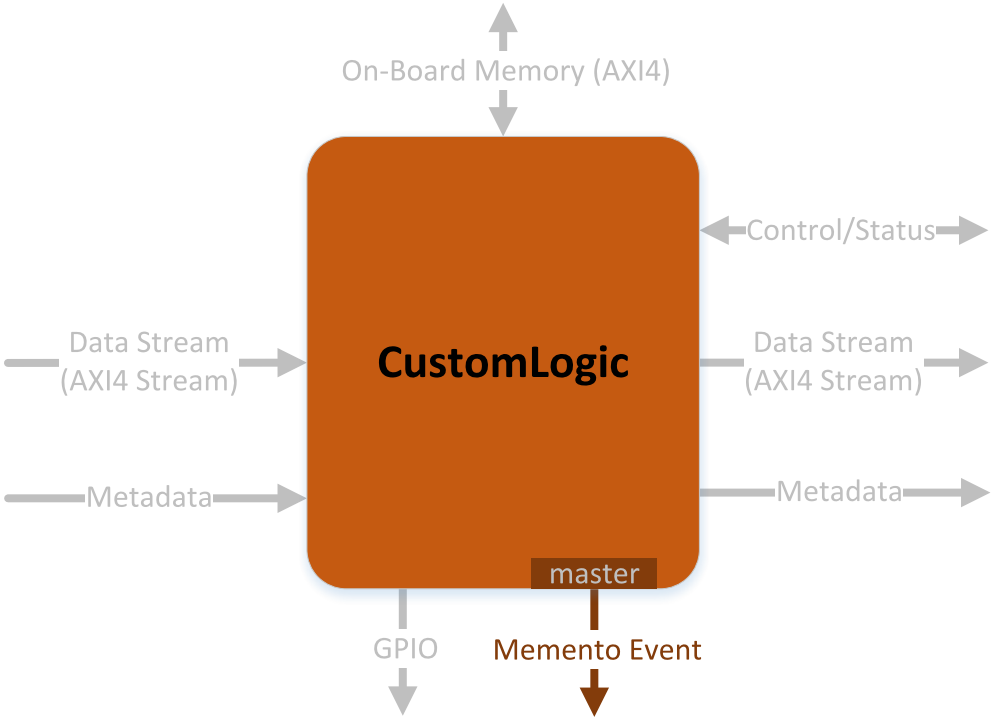
VALID/READY handshake timing diagram

2.5. Memento Event Interface

The Memento Event interface allows the *CustomLogic* to send timestamped events to the Memento Logging tool with a precision of 1 µs.

Along with the timestamped event, two 32-bit arguments are reported in Memento as follows:

```
                                              ARG_0      ARG_1
[ts:0195.350699] Message from CustomLogic: 0x00000003 0x00000002
```



Master interface signals

Signal	Width	Direction	Description
m_memento_event	N*1	Output	Pulse of one cycle indicating a *CustomLogic* event.
m_memento_arg0	N*32	Output	32-bit argument that is reported in Memento Logging tool along with the corresponding *CustomLogic* event.
m_memento_arg1	N*32	Output	32-bit argument that is reported in Memento Logging tool along with the corresponding *CustomLogic* event.

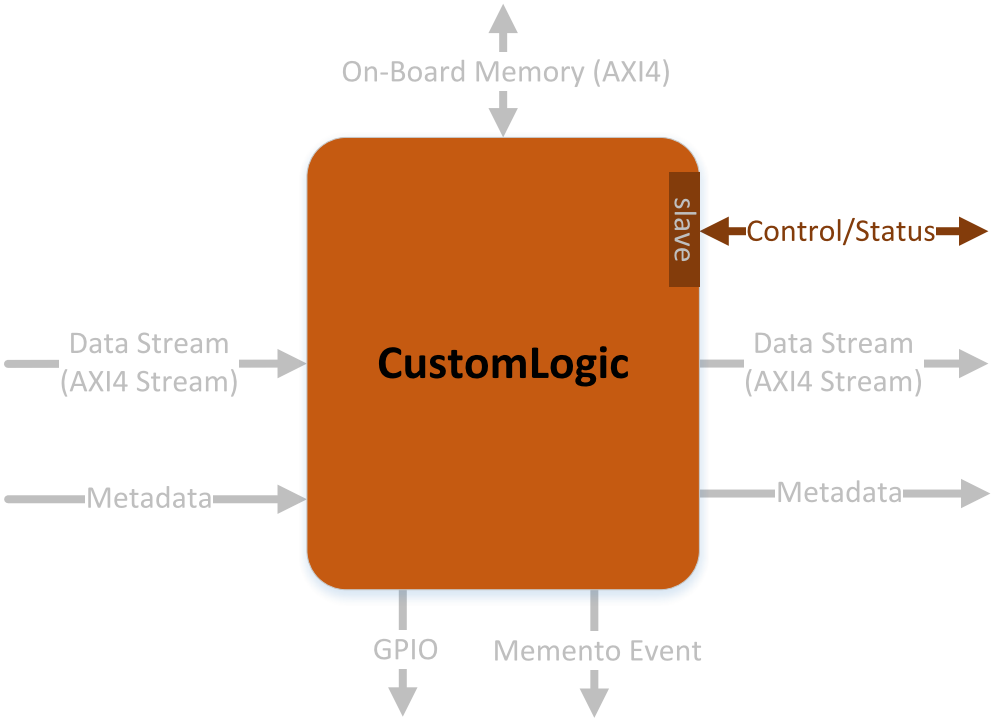
NOTE

In the Width column: “N” refers to the total number of interface slots, which is the number of devices/cameras supported by the *CustomLogic* variant.

- **3602 Coaxlink Octo** (1-camera, custom-logic) => *N* = 1;
- **3603 Coaxlink Quad CXP-12** (1-camera, custom-logic) => *N* = 1;
- **3603 Coaxlink Quad CXP-12** (4-camera, custom-logic) => *N* = 4;

2.6. Control/Status Interface

The Control/Status interface allows you to read or write registers inside the *CustomLogic* via the Coaxlink Driver API.



The use of this interface strongly depends on how the *CustomLogic* defines the Control/Status interface. The recommended definition is to use this interface as Address/Data Control Registers as illustrated in the reference design file `control_registers.vhd`.

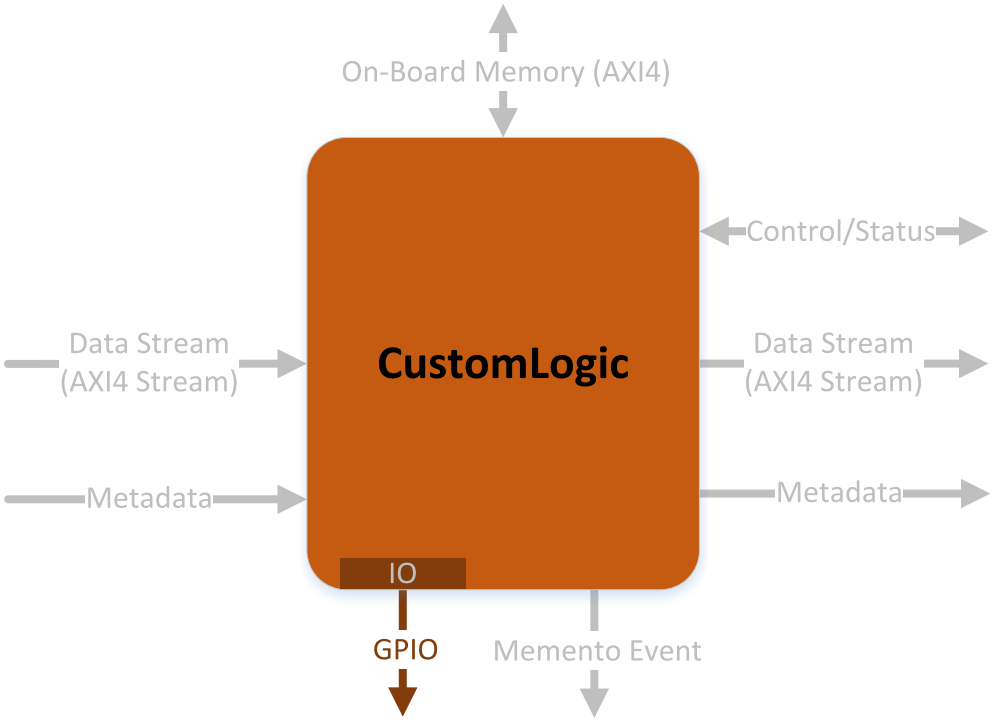
Interface signals

Signal	Width	Direction	Description
s_ctrl_addr	16	Input	16-bit WR/RD Address. The WR/RD Address selects the register to be read/written.
s_ctrl_data_wr_en	1	Input	Pulse of one cycle indicating an update in s_ctrl_data_wr.
s_ctrl_data_wr	32	Input	32-bit Write Data. Write a 32-bit vector into a selected register.
s_ctrl_data_rd	32	Output	32-bit Read Data. Copy a 32-bit vector from a selected register.

2.7. General Purpose I/O Interface

The General Purpose I/O (GPIO) interface gives access to the status of all I/Os available in the Coaxlink boards. It also allows the user to control the 'User Output Register'.

For more information on the General Purpose I/O and the User Output Register, please refer to the Coaxlink Functional Guide and the Coaxlink Hardware Manual.



Interface signals

Signal	Width	Direction	Description
user_output_ctrl	16	Output	Control the User Output Register: □ Ctrl[1:0] => UserOutput0 □ Ctrl[3:2] => UserOutput1 □ Ctrl[5:4] => UserOutput2 □ Ctrl[7:6] => UserOutput3 □ Ctrl[9:8] => UserOutput4 □ Ctrl[11:10] => UserOutput5 □ Ctrl[13:12] => UserOutput6 □ Ctrl[15:14] => UserOutput7 Each UserOutput register can be controlled independently. The 'Ctrl' fields are encoded as follows: □ "01" => UserOutputx <= '1' □ "10" => UserOutputx <= '0' □ Others => No change **NOTE** Each 'Ctrl' field is evaluated only once every time a change in the field is detected. It means that a UserOutput state can be changed via the Coaxlink Driver API even if the corresponding 'Ctrl' field is constantly forced to "01".
user_output_ status	8	Input	User Output Register status: □ Status[0] => UserOutput0 □ Status[1] => UserOutput1 □ Status[2] => UserOutput2 □ Status[3] => UserOutput3 □ Status[4] => UserOutput4 □ Status[5] => UserOutput5 □ Status[6] => UserOutput6

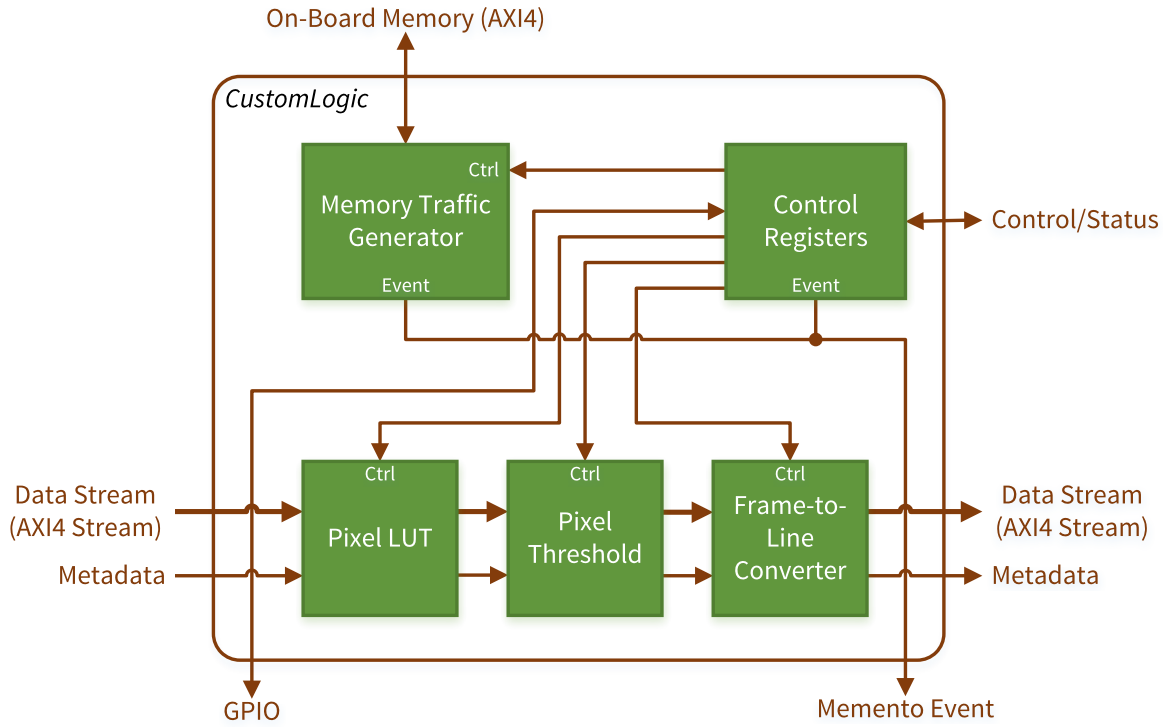
Signal	Width	Direction	Description
			◻ Status[7] => UserOutput7
standard_io_set1_status	10	Input	Standard I/O set #1 status: ◻ Status[0] => DIN11 ◻ Status[1] => DIN12 ◻ Status[2] => IIN11 ◻ Status[3] => IIN12 ◻ Status[4] => IIN13 ◻ Status[5] => IIN14 ◻ Status[6] => IOUT11 ◻ Status[7] => IOUT12 ◻ Status[8] => TTLIO11 ◻ Status[9] => TTLIO12
standard_io_set2_status	10	Input	Standard I/O set #2 status: ◻ Status[0] => DIN21 ◻ Status[1] => DIN22 ◻ Status[2] => IIN21 ◻ Status[3] => IIN22 ◻ Status[4] => IIN23 ◻ Status[5] => IIN24 ◻ Status[6] => IOUT21 ◻ Status[7] => IOUT22 ◻ Status[8] => TTLIO21 ◻ Status[9] => TTLIO22
module_io_set_status	40	Input	I/O Extension Module status: ◻ Status[0] => MIO1 ◻ Status[1] => MIO2 ◻ … ◻ Status[39] => MIO40

3. CustomLogic Reference Design

3.1. Introduction

The CustomLogic package is delivered with a reference design intended to be used as a template for the *CustomLogic*. The reference design exposes all interfaces available in the *CustomLogic*.

The reference design is delivered as set of VHDL files with the following block diagram:

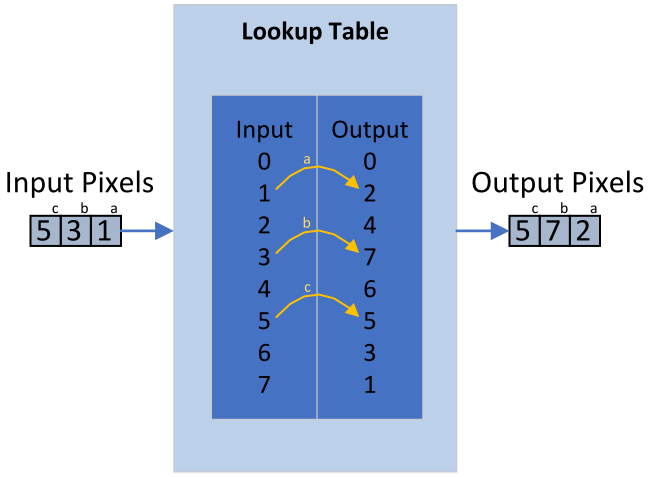


Reference design block diagram

3.2. Available Reference Modules

Pixel LUT 8-bit

The Pixel LUT 8-bit provides a Lookup Table operator in the *CustomLogic* reference design pipeline. A Lookup Table operator can change any input pixel value by a predefined value on its table. There are many applications for a Lookup Table, e.g., gamma correction and contrast enhancement. The following figure illustrate a Lookup Table operator:



The Pixel LUT 8-bit can compute 16 (for **3602 Coaxlink Octo**) or 32 (for **3603 Coaxlink Quad CXP-12**) 8-bit pixels per clock cycle. The Control Registers module is used to control and upload the Lookup Table values.

NOTE
This module only supports Mono8 pixel format.

Pixel Threshold

The Pixel Threshold provides a Threshold operator in the *CustomLogic* reference design pipeline. For each input pixel, the Threshold operator outputs 0 or 255 according to the formulas:

$$\text{OutputPixel} = 255 \text{ when } \text{InputPixel} \geq Th;$$
$$\text{OutputPixel} = 0 \text{ when } \text{InputPixel} < Th;$$

where Th is the Threshold level.

The Pixel Threshold compute compute 16 (for **3602 Coaxlink Octo**) or 32 (for **3603 Coaxlink Quad CXP-12** 8-bit pixels per clock cycle. The Control Registers module is used to control Pixel Threshold module.

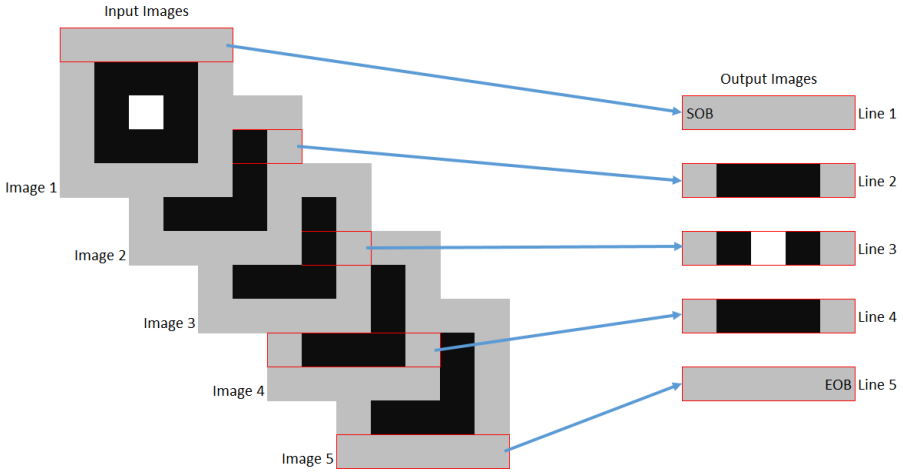
NOTE
This module was generated by Vivado HLS using C++ code as input. To regenerate this module, please follow the procedure described in the `/05_ref_design_hls/HLS_README.txt` file .

NOTE
This module only supports Mono8 pixel format.

Frame-to-Line Converter

The Frame-to-Line Converter outputs one line for each input image. The outputted lines are extracted from the input images in the following way:

- From the first input image, we extract the first line.
- From the second input image, we extract the second line and so on.
- When the Frame-to-Line Converter extracts the last line of the input image (that is the number of input images is equal to the image *Ysize*), it enables the flag *End-of-Buffer* at the last transfer of this line and starts a new cycle of acquisition.



The Frame-to-Line Converter latches the input Metadata (source side) of the first image in a sequence and transfers it to the output Metadata (destination side).

This module can be controlled via the Control Registers reference design.

Memory Traffic Generator

The Memory Traffic Generator writes data bursts of 1024 bytes incrementing the address from 0x0000000 and wrapping around at 0x40000000 (1 GB). The written data consists of an 8-bit counter.

After each burst of 1024 bytes, the Memory Traffic Generator reads back the data at the same address. It also reports the number of address wraparounds that have occurred.

This module can be controlled via the Control Registers reference design.

Memento Events

There are two sources of Memento Events in the reference design:

- One is via the Control Registers where the `m_memento_arg0` vector can be defined.
- The other event is generated when an address wraparound occurs in the Memory Traffic Generator. In this case, `m_memento_arg1` receives the value of the address wraparound counter.

General Purpose I/O

The status of all I/Os available in the General Purpose I/O interface can be read via the Control Register. It is also possible to control the UserOutput registers and read back their status via the Control Register.

Control Registers

The Control Registers module provides a mechanism to control/configure modules implemented in the *CustomLogic* via the Control/Status Interface. The reference register map is the following:

Register	Address	Description
Scratchpad	0x0000	Bits 31:0 (R/W) □ 32-bit scratch pad (reset value => 0x00000000)
MemTrafficGen	0x0001	Bit 0 (R/W) □ when '0' => Memory Traffic Generator is disabled (reset value) □ when '1' => Memory Traffic Generator is enabled
UserOutCtrl	0x0002	Bits 1:0 (R/W) => UserOutput0 Bits 3:2 (R/W) => UserOutput1 Bits 5:4 (R/W) => UserOutput2 Bits 7:6 (R/W) => UserOutput3 Bits 9:8 (R/W) => UserOutput4 Bits 11:10 (R/W) => UserOutput5 Bits 13:12 (R/W) => UserOutput6 Bits 15:14 (R/W) => UserOutput7 Bit fields encoding: □ When "01" => UserOutputx <= '1' □ When "10" => UserOutputx <= '0' □ Others => No change
UserOutStatus	0x0003	Bit 0 (R) => UserOutput0 state Bit 1 (R) => UserOutput1 state Bit 2 (R) => UserOutput2 state Bit 3 (R) => UserOutput3 state Bit 4 (R) => UserOutput4 state Bit 5 (R) => UserOutput5 state Bit 6 (R) => UserOutput6 state Bit 7 (R) => UserOutput7 state
IoSet1Status	0x0004	Bit 0 (R) => DIN11 state Bit 1 (R) => DIN12 state Bit 2 (R) => IIN11 state Bit 3 (R) => IIN12 state Bit 4 (R) => IIN13 state Bit 5 (R) => IIN14 state Bit 6 (R) => IOUT11 state Bit 7 (R) => IOUT12 state Bit 8 (R) => TTLIO11 state Bit 9 (R) => TTLIO12 state
IoSet2Status	0x0005	Bit 0 (R) => DIN21 state Bit 1 (R) => DIN22 state Bit 2 (R) => IIN21 state Bit 3 (R) => IIN22 state Bit 4 (R) => IIN23 state Bit 5 (R) => IIN24 state Bit 6 (R) => IOUT21 state Bit 7 (R) => IOUT22 state Bit 8 (R) => TTLIO21 state Bit 9 (R) => TTLIO22 state

Register	Address	Description
MioSetAStatus	0x0006	Bit 0 (R) => MIO1 state Bit 1 (R) => MIO2 state … Bit 31 (R) => MIO32 state
MioSetBStatus	0x0007	Bit 0 (R) => MIO33 state Bit 1 (R) => MIO34 state … Bit 7 (R) => MIO40 state
Frame2Line	0x1n00	Bit 0 (R/W) □ when "01" => Frame-to-Line Converter bypass is enabled (reset value) □ when "10" => Frame-to-Line Converter bypass is disabled
MementoEvent	0x1n01	Bits 31:0 (R/W) □ Any write in this register generates a Memento event and the 32-bit vector defined here is copied into CustomLogic_event_arg0
PixelLut	0x1n02	Bit 0 (W, auto-clear) □ when '1' => Starts a new write sequence of coefficients Bit 4 (R) □ when '1' => Indicates the end of a write sequence of coefficients (reset value => '0') Bits 9:8 (R/W) □ when "01" => Pixel LUT bypass is enabled (reset value) □ when "10" => Pixel LUT bypass is disabled □ when others => No change
PixelLutCoef	0x1n03	Bits 7:0 (W) □ Writes a coefficient into the Pixel LUT. Each write into this register increments the coefficient index from 0 to 255.
PixelThreshold	0x1n04	Bits 7:0 (R/W) □ when 0x00 => No change □ when others => Set the Pixel Threshold level (reset value => 0x01) Bits 9:8 (R/W) □ when "01" => Pixel Threshold bypass is enabled (reset value) □ when "10" => Pixel Threshold bypass is disabled □ when others => No change

NOTE
In the Address column "n" is a 4-bit field, which corresponds to the selector of the device/camera channel.

3.3. CustomLogic Delivery

The Coaxlink package targets Vivado 2018.3 and contains the following folders:

- `<variant short-name>/01_readme`
  Brief description how to generate a Vivado project from the Coaxlink *CustomLogic* Package.
- `<variant short-name>/02_coaxlink`
  Collection of proprietary files (encrypted HDL, netlists, and TCL scripts) necessary to build the *CustomLogic* framework.
  Note: These files shall not be modified.
- `<variant short-name>/03_scripts`
  Collection of scripts to help developing on CustomLogic.
- `<variant short-name>/04_ref_design`
  Reference design source files.
- `<variant short-name>/05_ref_design_hls`
  HLS reference design source files.
- `<variant short-name>/06_release`
  Pre-built reference design bitstream file.

Product	Variant full name	Variant short-name
3602 Coaxlink Octo	1-camera, custom-logic	CoaxlinkOcto_1-cam
3603 Coaxlink Quad CXP-12	1-camera, custom-logic	CoaxlinkQuadCxp12_1-cam
3603 Coaxlink Quad CXP-12	4-camera, custom-logic	CoaxlinkQuadCxp12_1-cam

3.4. Reference Design Build Procedure

To build the reference design:

1. Decompress the package in a folder respecting Vivado requirements (no special characters in the path). For example: `c:/workspace/CustomLogic`
2. Start Vivado
3. Execute the script "create_vivado_project.tcl" in the Tcl Console.
   TCL command: `source c:/workspace/CustomLogic/03_scripts/create_vivado_project.tcl`
   As result, a Vivado project is created at the folder `07_vivado_project`. For example: `c:/workspace/CustomLogic/07_vivado_project`.
4. Run Implementation.
   TCL command: `launch_runs impl_1`
5. Execute the script "customlogic_functions.tcl" in the Tcl Console.
   TCL command: `source c:/workspace/CustomLogic/03_scripts/customlogic_functions.tcl`
   This script makes the following two functions available:
   - *customlogic_bitgen*: Generate .bit file.
   - *customlogic_prog_fpga*: Program FPGA via JTAG (volatile).

NOTE
This function requires a Xilinx JTAG programmer.

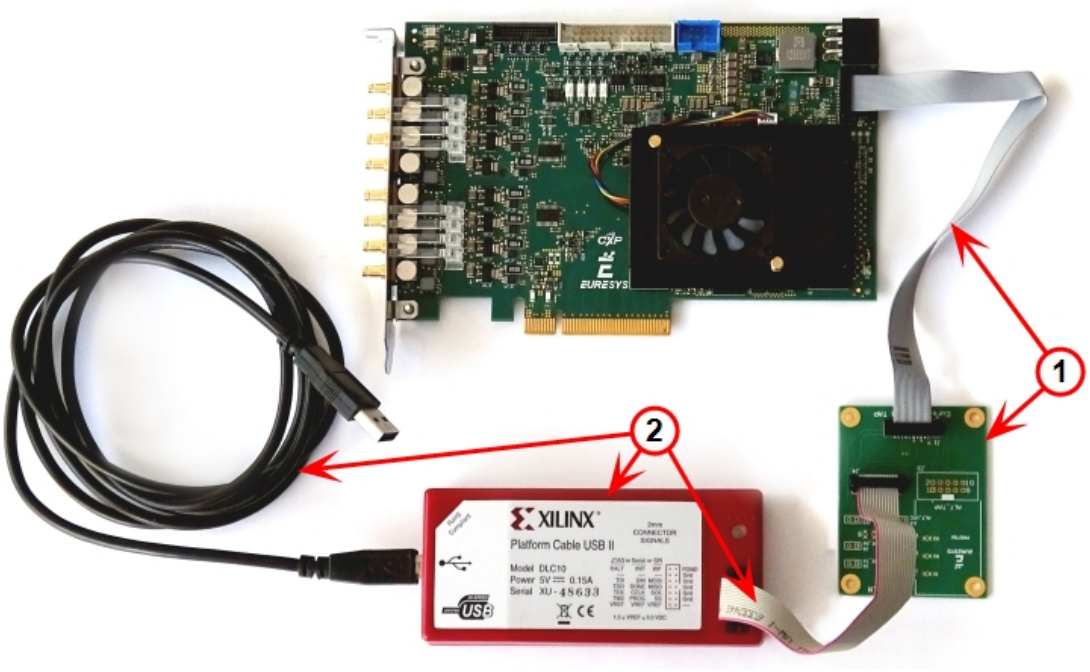
6. After completion of the implementation, run the function “customlogic_bitgen” in the TCL console.
   TCL command: `customlogic_bitgen`
   This function updates the bitstream file in the folder `06_release`.
7. After the bitstream is generated, update the FPGA by executing the function `customlogic_prog_fpga` in the TCL console.
   TCL command: `customlogic_prog_fpga`

NOTE
This step is optional.

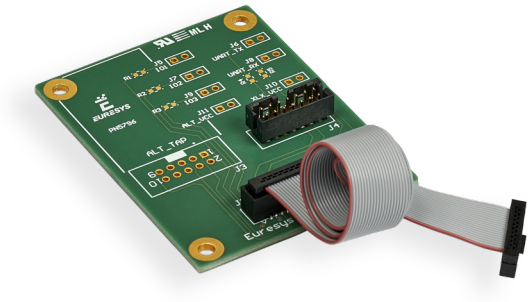
4. Debugging

CustomLogic does not require any additional hardware to program the FPGA.

However, to use the debugging feature of Vivado (ChipScope), you may purchase the *3613 JTAG Adapter Xilinx for Coaxlink* (1) to connect the *Xilinx Platform Cable USB II programmer* (2) to the Coaxlink FPGA.



Xilinx Platform Cable USB II programmer (2) connected to 3602 Coaxlink Octo through a 3613 JTAG Adapter Xilinx for Coaxlink (1)



3613 JTAG Adapter Xilinx for Coaxlink

5. Simulation Testbench

CustomLogic is delivered with a simulation testbench capable to stimulate all CustomLogic interfaces. It also captures data at backend side from the Data Stream, Metadata, and Memento Event interfaces. The results (captured data) are stored in files with extension '.dat' in the folder: <variant short-name>/07_vivado_project/CustomLogic.sim/sim_1/behave/xsim

The testbench is integrated in the Vivado project created by the script 'create_vivado_project.tcl'. To start the simulation, enter the command 'launch_simulation' into Vivado's Tcl Console.

The file 'SimulationCtrl_tb.vhd' allows the user to control the testbench. This file contains a process called 'Simulation' where is possible to create a sequence of actions for the testbench via a set of commands as in the following example:

```
Simulation : process
Begin
       -- Enable Data Stream at channel 0
       EnableDataStream        (clk,status,ctrl, 0);

       -- Request 5 frames (256x10 Mono8) at channel 0.
       FrameRequest            (clk,status,ctrl, 0, 5, 256, 10, Mono8);

       -- Disable Data Stream at channel 0
       DisableDataStream       (clk,status,ctrl, 0);

       -- End simulation
       std.env.finish;
end process;
```

A description of all available commands can be found in the file 'SimulationCtrl_tb.vhd' located in the folder: <variant short-name>/04_ref_design/sim

NOTE
Regarding the 'On-Board Memory Interface', the storage size of the test bench model is limited to 2 MB.