USER GUIDE

CustomLogic

Coaxlink Driver 11.0.0

3602 Coaxlink Octo

3603 Coaxlink Quad CXP-12

Terms of Use

This documentation is provided with CustomLogic 11.0.0 (doc build 9012).

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

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

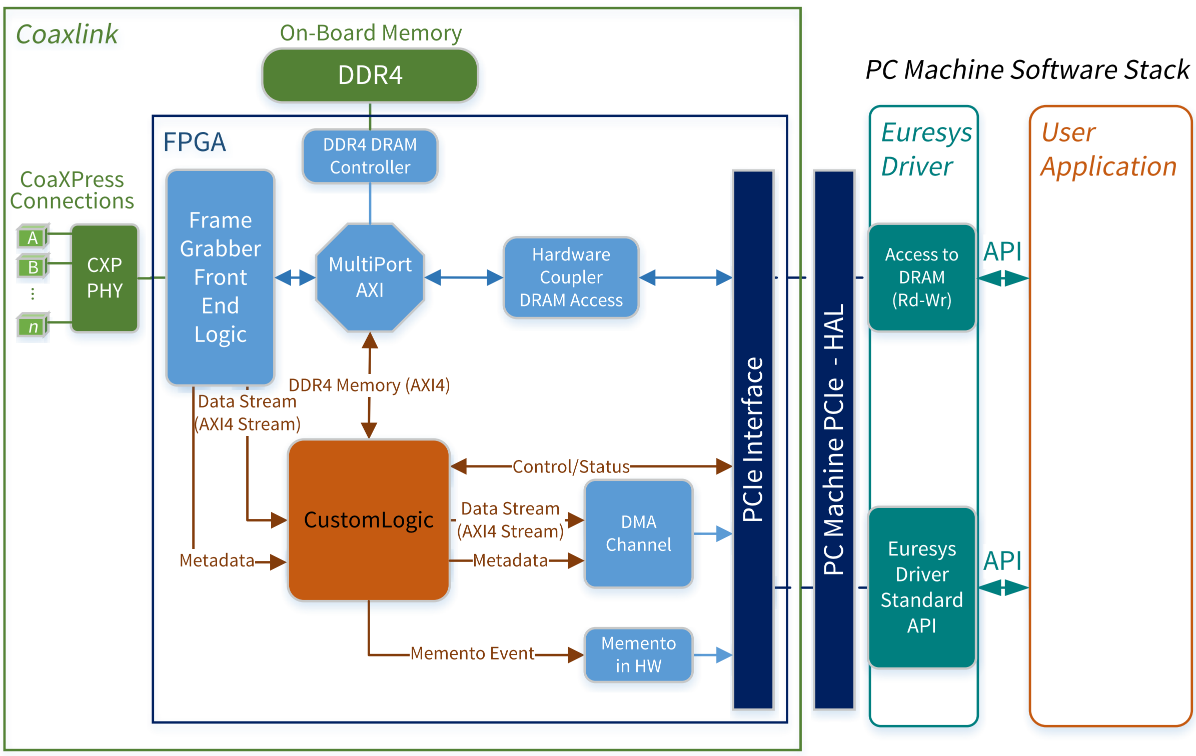
Firmware Variant	Description
1-camera, custom-logic	◻ CXP-12 HD-BNC 4 CoaXPress interface ◻ One 1- or 2- or 4-connection area-scan camera ◻ 2 GB RAM DDR4 on-board memory ◻ 8-lane Gen 3 PCI Express interface

1.3. Framework

A CustomLogic 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



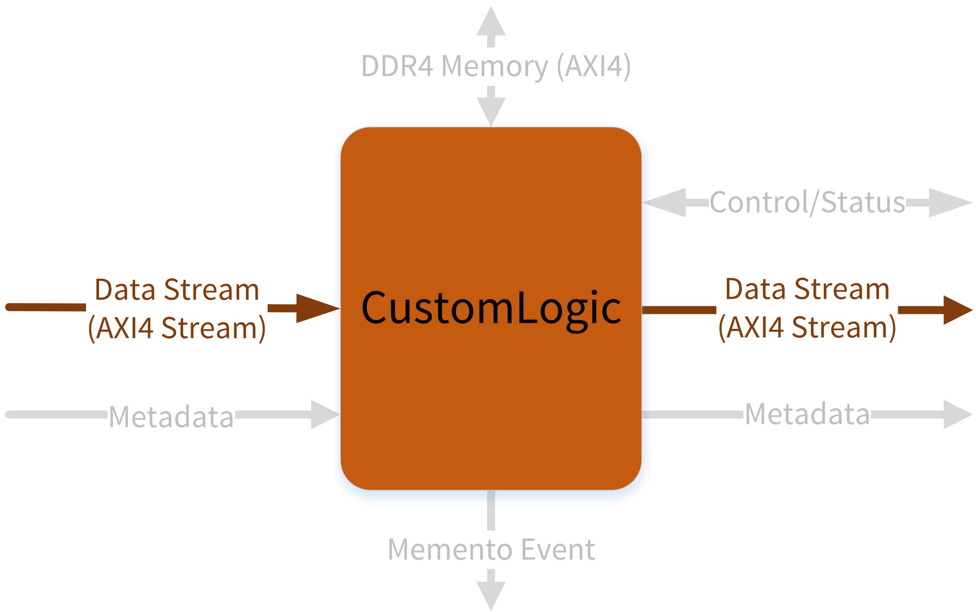
Coaxlink CustomLogic model

2. CustomLogic Interfaces

2.1. Data (Pixel) Stream Interface

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

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



Source interface signals

Signal	Direction	Description
axis_tvalid_in	IN	TVALID indicates that the **source** is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
axis_tready_in	OUT	TREADY indicates that the *CustomLogic* can accept a transfer in the current cycle.
axis_tdata_in [*]	IN	TDATA is the primary payload that is used to provide the data passing across the interface. *[*] = [127:0] for* Octo *or [255:0] for* QuadCXP12
axis_tuser_in [3:0]	IN	TUSER is user defined sideband information that can be transmitted alongside the data stream. The TUSER content is encoded as follows: □ axis_tuser_in[0] => Start-of-Frame (SOF) □ axis_tuser_in[1] => Start-of-Line (SOL) □ axis_tuser_in[2] => End-of-Line (EOL) □ axis_tuser_in[3] => End-of-Frame (EOF)

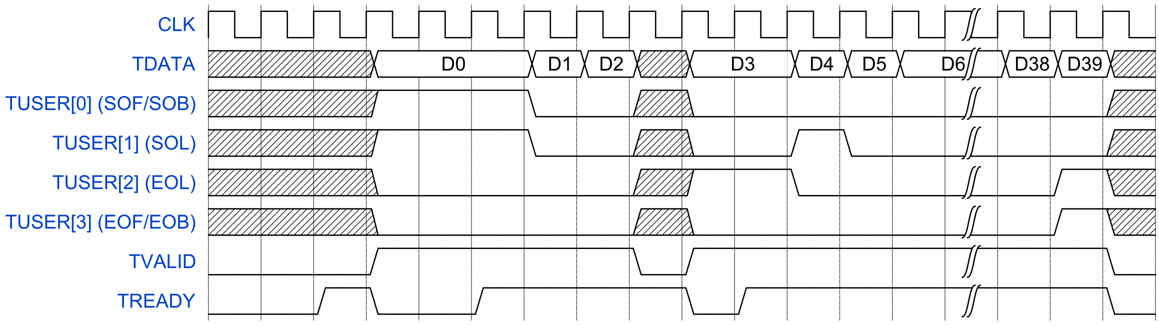
Destination interface signals

Signal	Direction	Description
axis_tvalid_out	OUT	TVALID indicates that the *CustomLogic* is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
axis_tready_out	IN	TREADY indicates that the **PCI Express DMA Back-End** can accept a transfer in the current cycle.
axis_tdata_out [*]	OUT	TDATA is the primary payload that is used to provide the data passing across the interface. *[*] = [127:0] for* Octo *or [255:0] for* QuadCXP12
axis_tuser_out [3:0]	OUT	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)

At the *CustomLogic* destination side, the *axis_tuser_out* signal has the function of controlling the PCI Express DMA Back-End. The flags carried by the *axis_tuser_out* 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



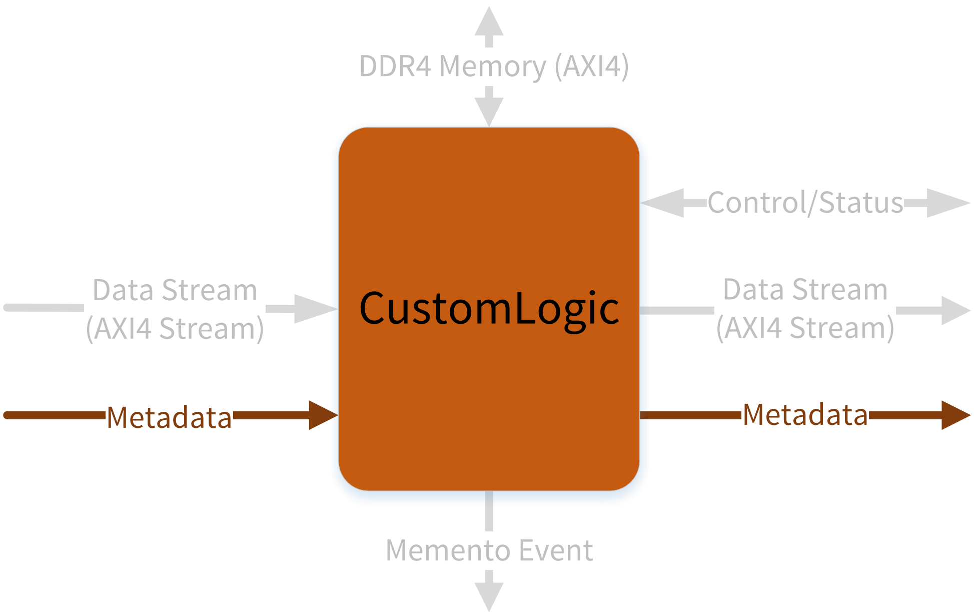
TVALID/TREADY handshake and TUSER flags timing diagram

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.

2.2. 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

The Metadata interface signals are presented as VHDL Record Type named *Metadata_rec* (the definition of this record type can be found in the *CustomLogicPkg* package (`CustomLogicPkg.vhd`).

- At the source side, the Metadata interface presents the prefix *Metadata_in* and has the direction *IN*.
- At the destination side, it presents the prefix *Metadata_out* and has the direction *OUT*.

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

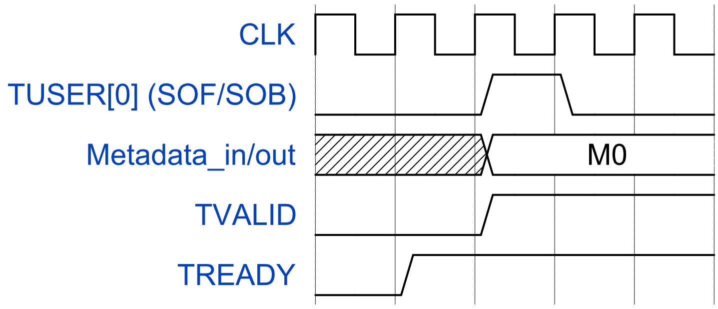
Signal	Description
[prefix].StreamId [7:0]	*Unique stream ID.*
[prefix].SourceTag [15:0]	*16 bit source image index. Incremented for each transferred image, wraparound to 0 at 0xFFFF. The same number shall be used by each stream containing data relating to the same image.*
[prefix].Xsize [23:0]	*24 bit value representing the image width in pixels.*
[prefix].Xoffs [23:0]	*24 bit value representing the horizontal offset in pixels of the image with respect to the left hand pixel of the full Device image.*
[prefix].Ysize [23:0]	*24 bit value representing the image height in pixels. This value shall be set to 0 for line scan images.*
[prefix].Yoff [23:0]	*24 bit value representing the vertical offset in pixels of the image with respect to the top line of the full Device image.*
[prefix].DsizeL [23:0]	*24 bit value representing the number of data words per image line.*
[prefix].PixelF [15:0]	*16 bit value representing the pixel format.*
[prefix].TapG [15:0]	*16 bit value representing the tap geometry.*
[prefix].Flags [15:0]	*Image flags.*

Note: *The descriptions in italic green are excerpts from CoaXPress Standard Version 1.1.1.*

CustomLogic group

Signal	Description
[prefix].Timestamp [31:0]	Timestamp of the Device's readout start event.
[prefix].PixProcessingFlgs [7:0]	Pixel processing flags: □ PixProcessingFlgs[0] => RGB to BGR swap enabled. □ PixProcessingFlgs[1] => MSB pixel alignment enabled. □ PixProcessingFlgs[2] => Packed acquisition enabled. □ PixProcessingFlgs[7:4] => LUT configuration.
[prefix].Status [31:0]	32-bit vector that can be used by the *CustomLogic* to report its status. **Note:** *At the source side the Status value is 0x00000000.*

Timing diagram



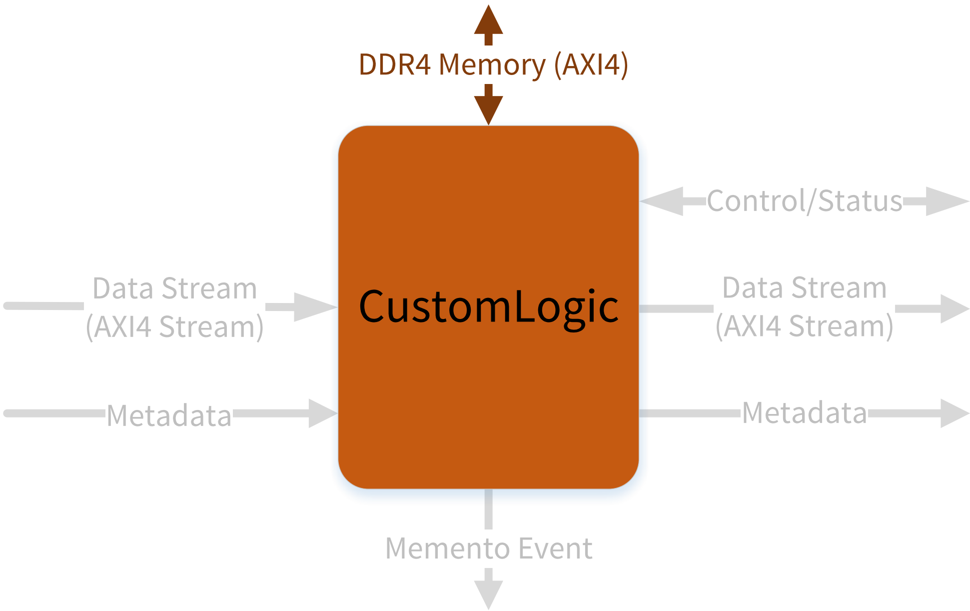
Metadata timing diagram

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

> **Important:** *The CustomLogic should not modify the contents of the signals included in the record type Metadata_rec.*

2.3. DDR4 Memory Interface

The DDR4 on-bard memory interface is based on the AMBA AXI4-Stream Protocol Specification.

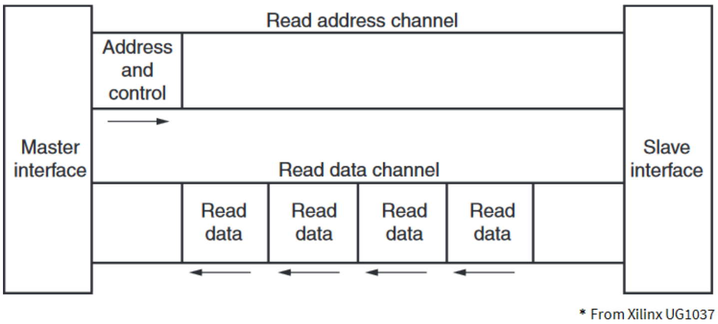


The AXI4 is a memory-mapped interface that consists of five channels:

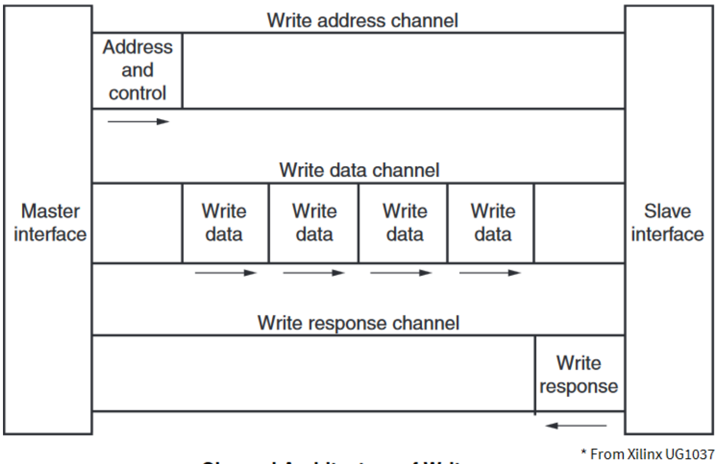
- Write Address Channel
- Write Data Channel
- Write Response Channel
- Read Address Channel
- Read Data Channel

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



Channel Architecture of Reads



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 write address channel interface signals

<table>
<tr><th>Signal</th><th>Direction</th><th>Description</th></tr>
<tr><td>m_axi_awaddr [31:0]</td><td>OUT</td><td>Write address. The write address gives the address of the first transfer in a write burst transaction.</td></tr>
<tr><td>m_axi_awlen [7:0]</td><td>OUT</td><td>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</td></tr>
<tr><td>m_axi_awsize [2:0]</td><td>OUT</td><td>Burst size. This signal indicates the size of each transfer in the burst.

Burst_Size = 2^AWSIZE[2:0]</td></tr>
<tr><td>m_axi_awburst [1:0]</td><td>OUT</td><td>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</td></tr>
<tr><td>m_axi_awlock</td><td>OUT</td><td>Lock type. Provides additional information about the atomic characteristics of the transfer.

Atomic_Access: '0' Normal; '1' Exclusive</td></tr>
<tr><td>m_axi_awcache [3:0]</td><td>OUT</td><td>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</td></tr>
<tr><td>m_axi_awprot [2:0]</td><td>OUT</td><td>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</td></tr>
<tr><td>m_axi_awqos [3:0]</td><td>OUT</td><td>Quality of Service, QoS. The QoS identifier sent for each write transaction.

Quality_of_Service: Priority level</td></tr>
</table>

Signal	Direction	Description
m_axi_awvalid	OUT	*Write address valid. This signal indicates that the channel is signaling valid write address and control information.*
m_axi_awready	IN	*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	Direction	Description
m_axi_wdata [*]	OUT	*Write data.* *[*] = [127:0] for* Octo *or [255:0] for* QuadCXP12
m_axi_wstrb [**]	OUT	*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.* *[**] = [15:0] for* Octo *or [31:0] for* QuadCXP12
m_axi_wlast	OUT	*Write last. This signal indicates the last transfer in a write burst.*
m_axi_wvalid	OUT	*Write valid. This signal indicates that valid write data and strobes are available.*
m_axi_wready	IN	*Write ready. This signal indicates that the slave can accept the write data.*

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

Master write response channel interface signals

Signal	Direction	Description
m_axi_bresp [1:0]	IN	*Write response. This signal indicates the status of the write transaction.* *Response:* □ *"00" = OKAY* □ *"01" = EXOKAY* □ *"10" = SLVERR* □ *"11" = DECERR*
m_axi_bvalid	IN	*Write response valid. This signal indicates that the channel is signaling a valid write response.*
m_axi_bready	OUT	*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	Direction	Description
m_axi_araddr[31:0]	OUT	*Read address. The read address gives the address of the first transfer in a read burst transaction.*
m_axi_arlen[7:0]	OUT	*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[2:0]	OUT	*Burst size. This signal indicates the size of each transfer in the burst.* *Burst_Size = 2^ARSIZE[2:0]*
m_axi_arburst[1:0]	OUT	*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	OUT	*Lock type. Provides additional information about the atomic characteristics of the transfer.* *Atomic_Access: '0' Normal; '1' Exclusive*
m_axi_arcache[3:0]	OUT	*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[2:0]	OUT	*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*
m_axi_arqos[3:0]	OUT	*Quality of Service, QoS. The QoS identifier sent for each write transaction.* *Quality_of_Service: Priority level*

Signal	Direction	Description
m_axi_arvalid	OUT	*Read address valid. This signal indicates that the channel is signaling valid read address and control information.*
m_axi_arready	IN	*Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals.*

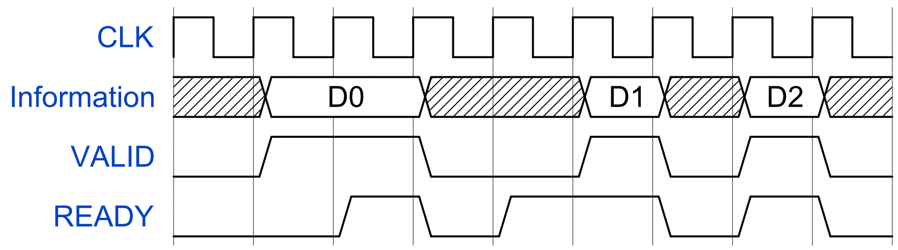
Note: *The descriptions are excerpts from AMBA AXI and ACE Protocol Specification.*

Master read data channel interface signals

Signal	Direction	Description
m_axi_rdata [*]	IN	*Read data.* *[*] = [127:0] for* Octo *or [255:0] for* QuadCXP12
m_axi_rresp [1:0]	IN	*Read response. This signal indicates the status of the read transfer.* *Response: "00" = OKAY; "01" = EXOKAY; "10" = SLVERR; "11" = DECERR*
m_axi_rlast	IN	*Read last. This signal indicates the last transfer in a read burst.*
m_axi_rvalid	IN	*Read valid. This signal indicates that the channel is signaling the required read data.*
m_axi_rready	OUT	*Read ready. This signal indicates that the master can accept the read data and response information.*

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

Timing diagram



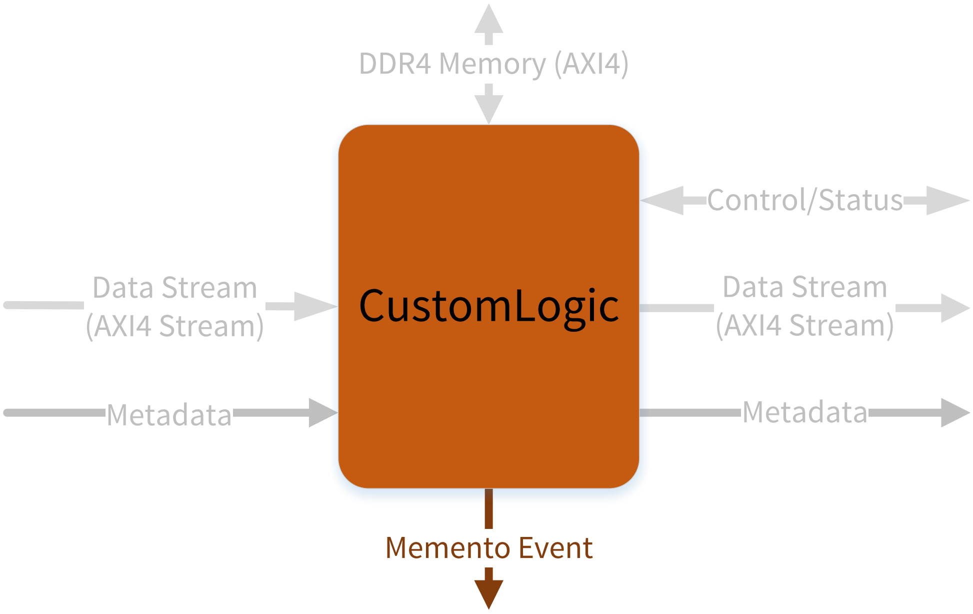
VALID/READY handshake timing diagram

2.4. 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
```

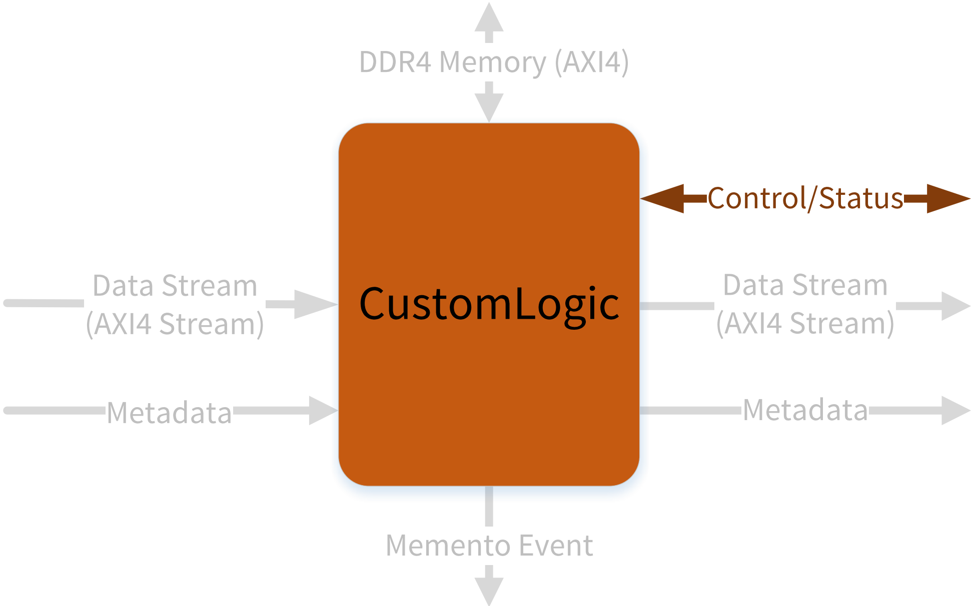


Interface signals

Signal	Direction	Description
CustomLogic_event	OUT	Pulse of one cycle indicating a *CustomLogic* event.
CustomLogic_event_arg0 [31:0]	OUT	32-bit argument that is reported in Memento Logging tool along with the corresponding *CustomLogic* event.
CustomLogic_event_arg1 [31:0]	OUT	32-bit argument that is reported in Memento Logging tool along with the corresponding *CustomLogic* event.

2.5. 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`.

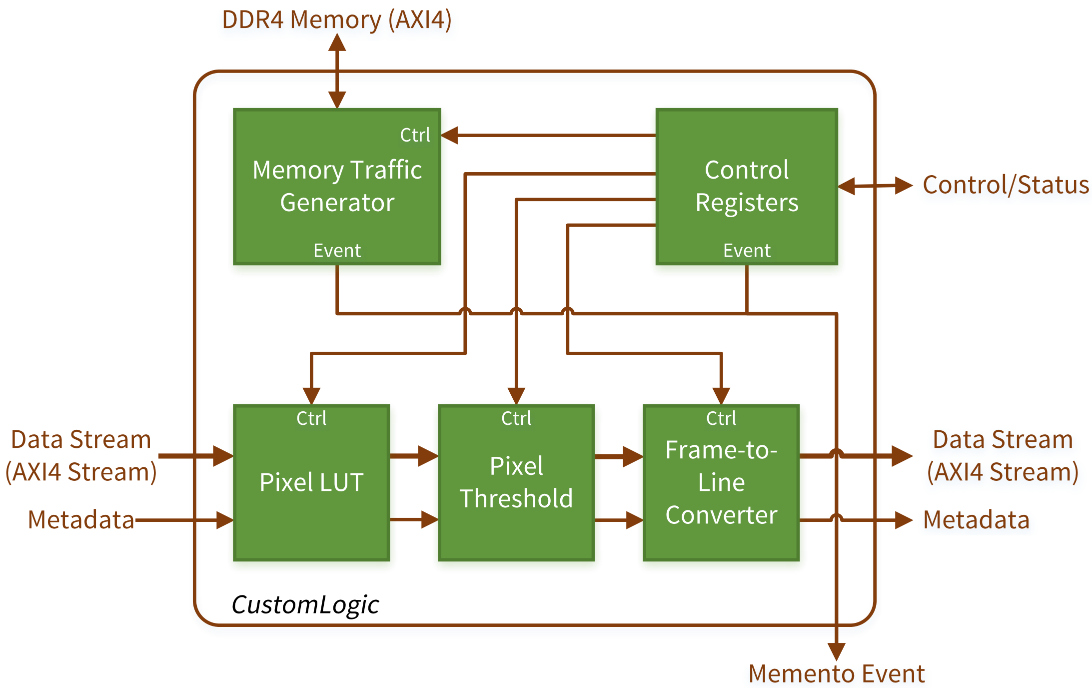
Interface signals

Signal	Direction	Description
CustomLogic_ctrl_addr [15:0]	IN	16-bit WR/RD Address. The WR/RD Address selects the register to be read/written.
CustomLogic_ctrl_data_in_ce	IN	Pulse of one cycle indicating an update in CustomLogic_ctrl_data_in.
CustomLogic_ctrl_data_in [31:0]	IN	32-bit Write Data. Write a 32-bit vector into a selected register.
CustomLogic_ctrl_data_out[31:0]	OUT	32-bit Read Data. Copy a 32-bit vector from a selected register.

3. CustomLogic Reference Design

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

3.1. Global Signals

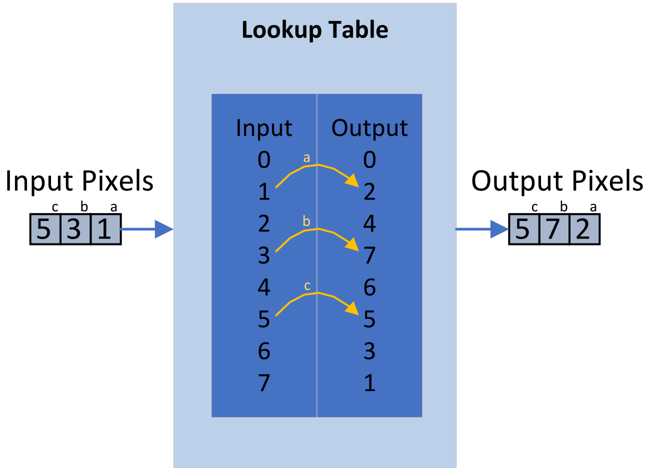
All available interfaces are in the same clock domain of 250 MHz and the *CustomLogic* global signals are the following:

Signal	Direction	Description
clk250	IN	250 MHz clock source common to all *CustomLogic* interfaces.
srst250	IN	Synchronous reset (clk250) asserted during a PCI Express reset.
PipelineClear	IN	Pulse asserted when a Stop Acquisition command (DSStopAcquisition) is executed. This signal should be used to clear the *CustomLogic* internal pipeline along the Data Stream.

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 Octo) or 32 (for QuadCXP12) 8-bit pixels per clock cycle. The Control Registers module is used to control and upload the Lookup Table values.

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:

```
OutputPixel= 255 when &InputPixel≥Th;
 OutputPixel= 0 when &InputPixel<Th;
```

where `Th` is the Threshold level.

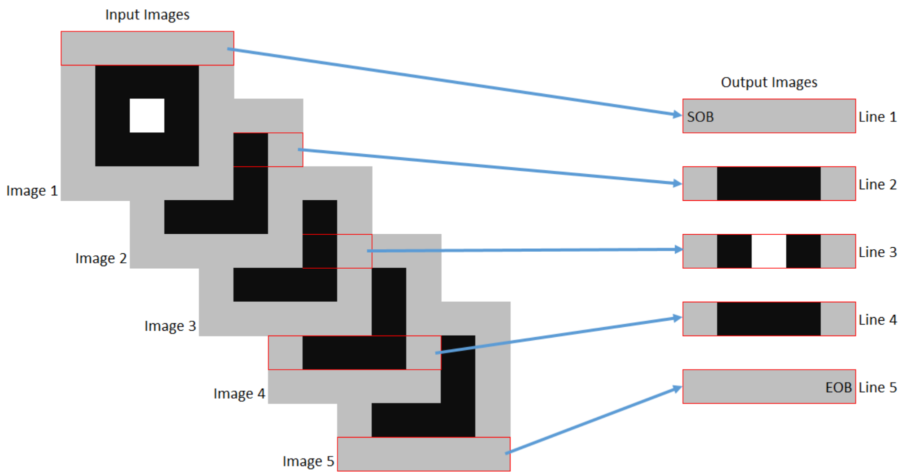
The Pixel Threshold compute 16 (for Octo) or 32 (for QuadCXP12) 8-bit pixels per clock cycle. The Control Registers module is used to control Pixel Threshold module.

> **Note:** *This module was generated based on a C++ code using Vivado HLS. To regenerate this module, please follow the procedure described in the* `/05_ref_design_hls/HLS_README.txt` *file .*

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 *CustomLogic_event_arg0* vector can be defined.
- The other event is generated when an address wraparound occurs in the Memory Traffic Generator. In this case, *CustomLogic_event_arg1* receives the value of the address wraparound counter.

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)
Frame2Line	0x0001	Bit 0 (R/W) □ when ‘0’ => Frame-to-Line Converter bypass is disabled □ when ‘1’=> Frame-to-Line Converter bypass is enabled (reset value)
MemTrafficGen	0x0002	Bit 0 (R/W) □ when ‘0’ => Memory Traffic Generator is disabled (reset value) □ when ‘1’ => Memory Traffic Generator is enabled
MementoEvent	0x0003	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	0x0004	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
PixelLutCoeff	0x0005	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	0x0006	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

3.3. CustomLogic Delivery

The CustomLogic package targets Vivado 2018.3 and contains the following files:

- `/README.txt`
  Brief description how to generate a Vivado project from the Coaxlink *CustomLogic* Package.
- `/01_doc/D903EN-User Guide-CustomLogic-xxx.pdf`
  CustomLogic User Guide (this document)
- `/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.
- `/03_scripts/`
  - `customlogic_functions.tcl`
    Vivado tcl script containing functions to generate the .bit file, program FPGA via JTAG.
  - `configure_fgrabber.js`
    Sample script to configure all reference design modules.
  - `create_vivado_project.tcl`
    Vivado tcl script to create a Vivado project for *CustomLogic*.
- `/04_ref_design/`
  - `CustomLogic.vhd`
    Reference Design wrapper. Users can include their own logic in this file.
  - `CustomLogic.xdc`
    Target constraint file for Vivado project.
  - `CustomLogicPkg.vhd`
    Package containing the definition of the record type Metadata_rec.
  - `CustomLogicTop.vhd`
    Top-level of the project.
  - `frame_to_line.vhd`
    Frame-to-Line Converter.
  - `mem_traffic_gen.vhd`
    Memory Traffic Generator.
  - `control_registers.vhd`
    Control Registers.
  - `pix_lut8b.vhd`
    Pixel LUT 8-bit.
  - `ip/lut_bram_8x256/lut_bram_8x256.xci`
    Xilinx IP used in the Pixel LUT 8-bit module.

- `pix_threshold.vhd`
  Pixel Threshold (HLS IP).
- `pix_threshold_wrp.vhd`
  Pixel Threshold wrapper.

- `/05_ref_design_hls/`
  - `HLS_README.txt`
    Brief description how to generate the HLS sample IPs.
  - `scripts/run_hls.tcl`
    Script to generate the HLS sample IPs.
  - `srcs/CustomLogic.h`
    Global header for *CustomLogic*.
  - `srcs/pix_threshold.h`
    Header for the Pixel Threshold example.
  - `srcs/pix_threshold.cpp`
    C++ code of the Pixel Threshold example.
  - `srcs/pix_threshold_test.cpp`
    C++ testbench code for the Pixel Threshold example.
- `/06_release/`
  - `CoaxlinkOcto_1-cam.bit`
    Pre-built FPGA bitstream for 3602 Coaxlink Octo
  - `CoaxlinkQuadCxp12_1-cam.bit`
    Pre-built FPGA bitstream for 3603 Coaxlink Quad CXP-12
  - `CoaxlinkOcto_1-cam.ltx`
    Pre-built Chipscope probes (empty) for 3602 Coaxlink Octo
  - `CoaxlinkQuadCxp12_1-cam.ltx`
    Pre-built Chipscope probes (empty) for 3603 Coaxlink Quad CXP-12

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.