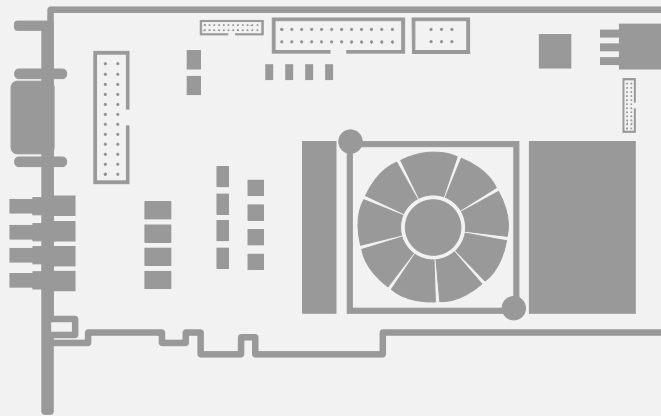


eGrabber

Using CustomLogic

3602 Coaxlink Octo

3603 Coaxlink Quad CXP-12
3603-4 Coaxlink Quad CXP-12



This documentation is provided with **eGrabber 24.04.0** (doc build **2187**).
www.euresys.com

This documentation is subject to the General Terms and Conditions stated on the website of **EURESYS S.A.** and available on the webpage <https://www.euresys.com/en/Menu-Legal/Terms-conditions>. The article 10 (Limitations of Liability and Disclaimers) and article 12 (Intellectual Property Rights) are more specifically applicable.

Contents

1. Introduction	4
1.1. Principles	5
1.2. Availability	6
1.3. Framework	8
2. Interfaces	9
2.1. Global Signals	10
2.2. Data (Pixel) Stream Interface	11
2.3. Metadata Interface	15
2.4. On-Board Memory Interface	20
2.5. Memento Event Interface	30
2.6. Control/Status Interface	32
2.7. General Purpose I/O Interface	33
3. Reference Design	36
3.1. Introduction	37
3.2. Available Reference Modules	38
3.3. CustomLogic Delivery	43
3.4. Reference Design Build Procedure	44
4. Debugging	45
5. Simulation Testbench	46
6. GenApi Features	47
6.1. CustomLogicControlAddress	48
6.2. CustomLogicControlData	49
7. Managing Firmware	50
7.1. What's Firmware?	51
7.2. Firmware Manager Tools	52
7.3. Updating and Installing Firmware	54
7.4. Special Firmware Procedures	55
7.5. Firmware Recovery Switch	58

1. Introduction

1.1. Principles	5
1.2. Availability	6
1.3. Framework	8

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	Revision
1-camera, custom-logic	<ul style="list-style-type: none">• 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	416
2-camera, line-scan, custom-logic	<ul style="list-style-type: none">• CXP-6 DIN 4 CoaXPress interface• Two 1- or 2-connection line-scan cameras• 2 GB RAM DDR4 on-board memory• 8-lane Gen 3 PCI Express interface	416

3603 Coaxlink Quad CXP-12 and 3603-4 Coaxlink Quad CXP-12

Firmware Variant	Description	Revision
1-camera, custom-logic	<ul style="list-style-type: none"> • CXP-12 HD-BNC 4 CoaXPress interface • One 1- or 2- or 4-connection area-scan camera • 2 GB (3603) or 4GB (3603-4) RAM DDR4 on-board memory • 8-lane Gen 3 PCI Express interface 	416
1-camera, line-scan, custom-logic	<ul style="list-style-type: none"> • CXP-12 HD-BNC 4 CoaXPress interface • One 1- or 2- or 4-connection line-scan camera • 2 GB (3603) or 4GB (3603-4) RAM DDR4 on-board memory • 8-lane Gen 3 PCI Express interface 	416
2-camera, custom-logic	<ul style="list-style-type: none"> • CXP-12 HD-BNC 4 CoaXPress interface • Two 1- or 2-connection area-scan cameras • 2 GB (3603) or 4GB (3603-4) RAM DDR4 on-board memory • 8-lane Gen 3 PCI Express interface 	416
2-camera, line-scan, custom-logic	<ul style="list-style-type: none"> • CXP-12 HD-BNC 4 CoaXPress interface • Two 1- or 2-connection line-scan cameras • 2 GB (3603) or 4GB (3603-4) RAM DDR4 on-board memory • 8-lane Gen 3 PCI Express interface 	416
4-camera, custom-logic	<ul style="list-style-type: none"> • CXP-12 HD-BNC 4 CoaXPress interface • Four 1-connection area-scan cameras • 2 GB (3603) or 4GB (3603-4) RAM DDR4 on-board memory • 8-lane Gen 3 PCI Express interface 	416

3625 Coaxlink QSFP+

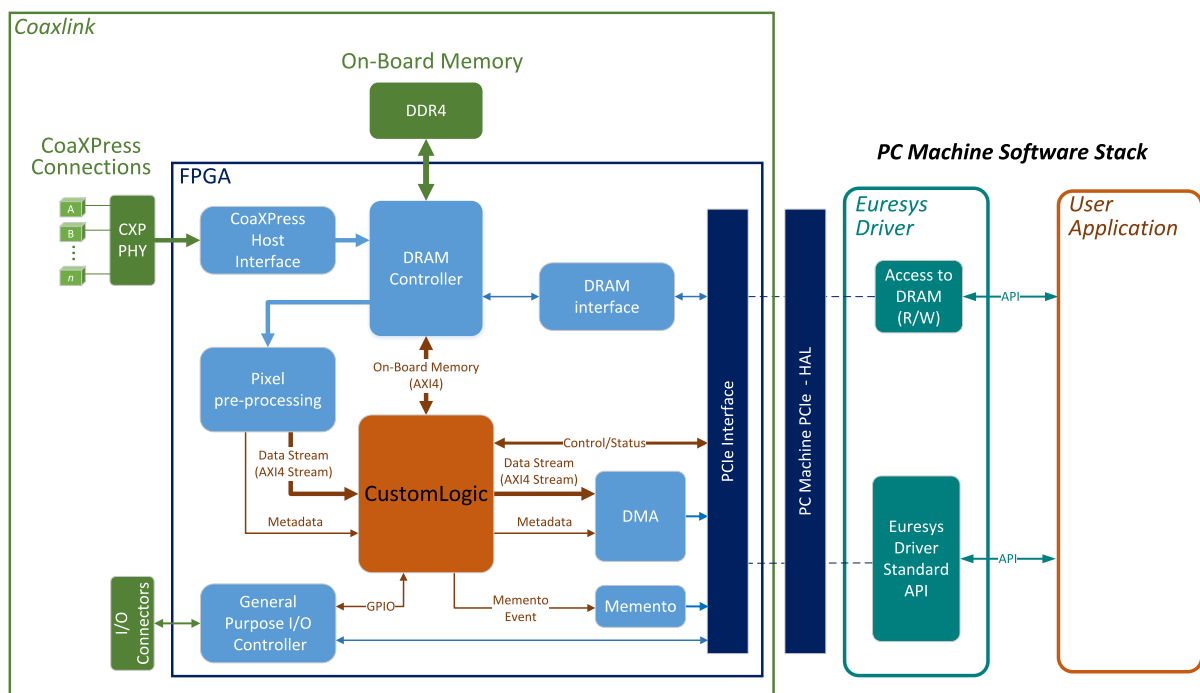
Firmware Variant	Description	Revision
1-camera, custom-logic	<ul style="list-style-type: none"> • CoaXPress-over-Fiber CoF-10 interface • One 1- or 2- or 4-connection area-scan camera • 4 GB RAM DDR4 on-board memory • 8-lane Gen 3 PCI Express interface 	4416

1.3. Framework

A Coaxlink **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



CustomLogic model

2. Interfaces

2.1. Global Signals	10
2.2. Data (Pixel) Stream Interface	11
2.3. Metadata Interface	15
2.4. On-Board Memory Interface	20
2.5. Memento Event Interface	30
2.6. Control/Status Interface	32
2.7. General Purpose I/O Interface	33

2.1. Global Signals

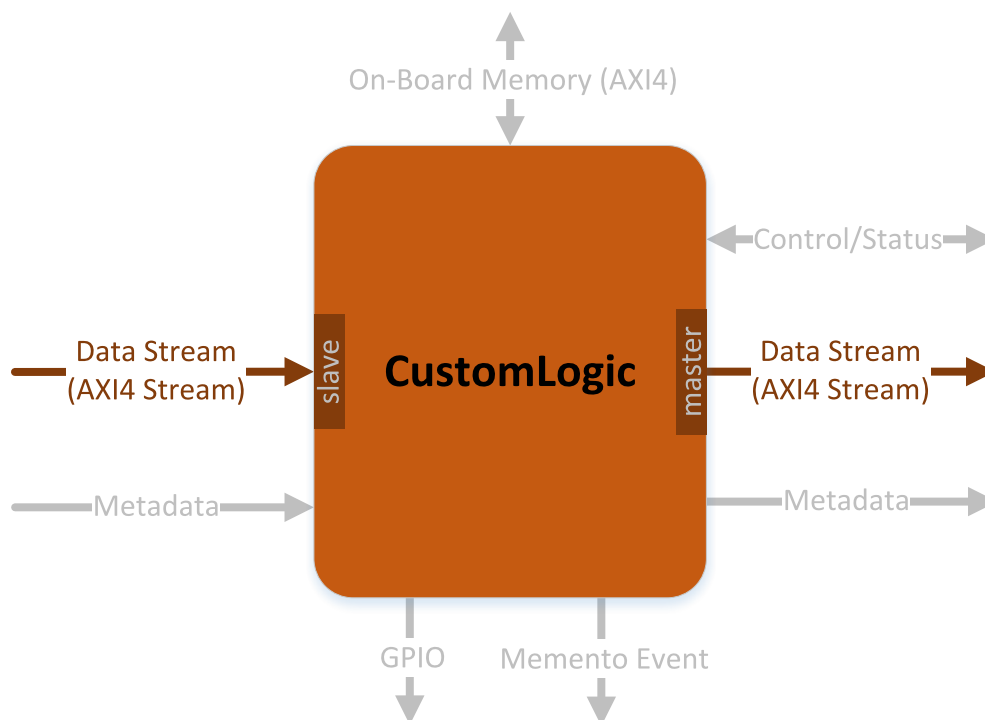
All available interfaces are in the same clock domain of 250 MHz and the **CustomLogic** global signals are the following:

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

In this topic:

- "Interface signals" on page 11
- "Slave interface signals" on page 12
- "Master interface signals" on page 13
- "Timing diagram" on page 14



Interface signals

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 AMBA® 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 is transmitted alongside the data stream. The TUSER content is encoded depending on the variant type. <ul style="list-style-type: none"> • For area-scan variants: <ul style="list-style-type: none"> □ TUSER [0] => Start-of-Frame (SOF) □ TUSER [1] => Start-of-Line (SOL) □ TUSER [2] => End-of-Line (EOL) □ TUSER [3] => End-of-Frame (EOF) • For line-scan variants: <ul style="list-style-type: none"> □ TUSER [0] => Start-of-Scan (SOS) □ TUSER [1] => Start-of-Line (SOL) □ TUSER [2] => End-of-Line (EOL) □ TUSER [3] => End-of-Scan (EOS)



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 ;
 - (2-camera, line-scan, custom-logic) => N = 2; W = 64;
- **3603 Coaxlink Quad CXP-12 and 3603-4 Coaxlink Quad CXP-12**
 - (1-camera, custom-logic) => N = 1; W = 256;
 - (1-camera, line-scan, custom-logic) => N=1; W=256;
 - (2-camera, custom-logic) => N = 2; W = 128;
 - (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 is transmitted alongside the data stream. The TUSER content is encoded depending on the variant type. <ul style="list-style-type: none"> • For area-scan variants: <ul style="list-style-type: none"> □ TUSER [0] => Start-of-Buffer (SOB) □ TUSER [1] => Reserved □ TUSER [2] => Reserved □ TUSER [3] => End-of-Buffer (EOB) • For line-scan variants: <ul style="list-style-type: none"> □ TUSER [0] => Start-of-Scan (SOS) □ TUSER [1] => Start-of-Line (SOL) □ TUSER [2] => End-of-Line (EOL) □ TUSER [3] => End-of-Scan (EOS)



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 ;
 - (2-camera, line-scan, custom-logic) => N = 2; W = 64;
- **3603 Coaxlink Quad CXP-12 and 3603-4 Coaxlink Quad CXP-12**
 - (1-camera, custom-logic) => N = 1; W = 256;
 - (1-camera, line-scan, custom-logic) => N=1; W=256;
 - (2-camera, custom-logic) => N = 2; W = 128;
 - (4-camera, custom-logic) => N = 4; W = 64;

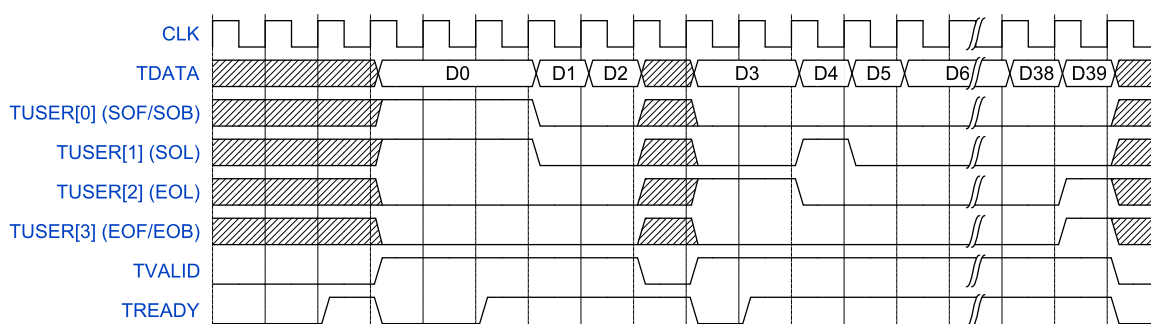


NOTE

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 depending on the variant type.

- For area-scan variants:
 - *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.
- For line-scan variants:
 - *Start-of-Scan*: A cycle containing this flag starts a new buffer. When the remaining space of a buffer is not sufficient to store an image line data, the acquisition continues into a new buffer and the filled buffer is made available to the application for processing.
 - *End-of-Scan*: 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.

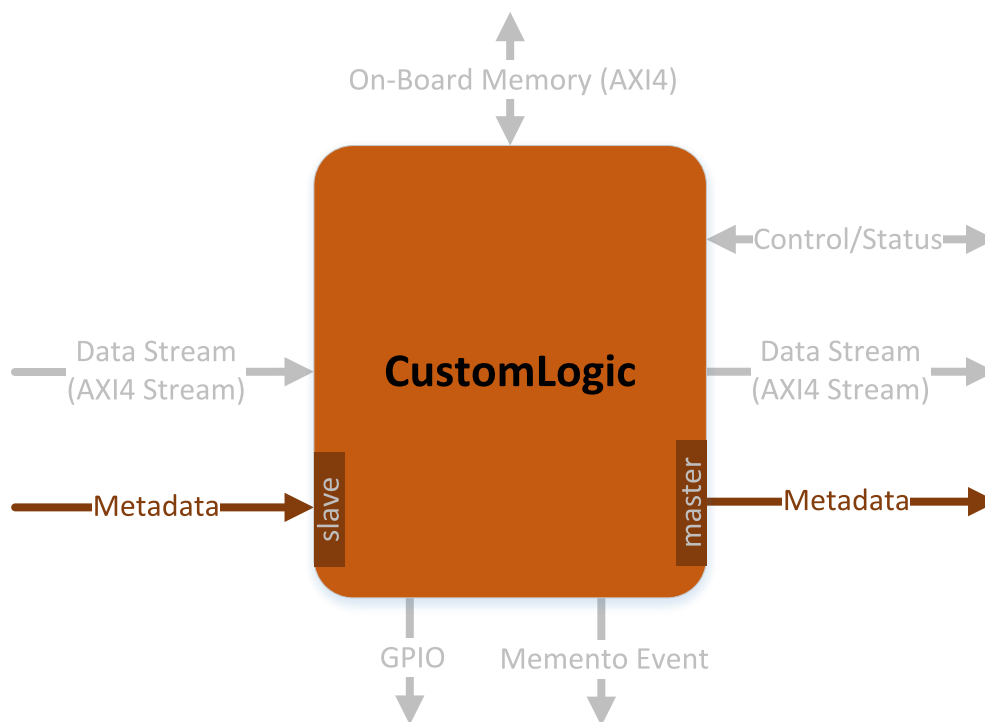
See also: For more information about the **AMBA® AXI4-Stream** protocol, please refer to Xilinx “AXI Reference Guide (UG1037)” at www.xilinx.com and **AMBA® AXI4-Stream Protocol Specification** at www.amba.com.

2.3. Metadata Interface

In this topic:

- "Interface signals" on page 16
- "CoaXPress Image Header group" on page 17
- "CustomLogic group" on page 18
- "Timing diagram" on page 19

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;
 - (2-camera, line-scan, custom-logic) => N = 2;
- **3603 Coaxlink Quad CXP-12 and 3603-4 Coaxlink Quad CXP-12**
 - (1-camera, custom-logic) => N = 1;
 - (1-camera, line-scan, custom-logic) => N=1;
 - (2-camera, custom-logic) => N = 2;
 - (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: <ul style="list-style-type: none"> • 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.

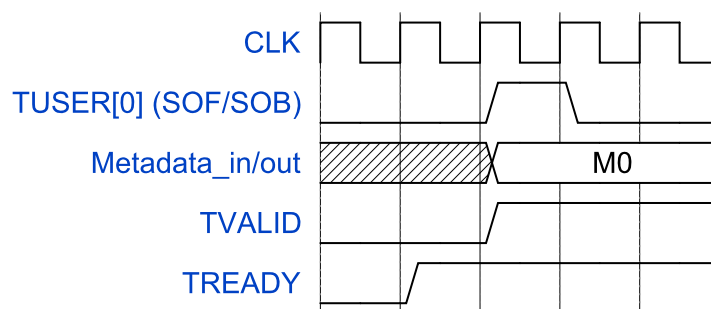


NOTE

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) => $N = 1$;
 - (2-camera, line-scan, custom-logic) => $N = 2$;
- **3603 Coaxlink Quad CXP-12 and 3603-4 Coaxlink Quad CXP-12**
 - (1-camera, custom-logic) => $N = 1$;
 - (1-camera, line-scan, custom-logic) => $N = 1$;
 - (2-camera, custom-logic) => $N = 2$;
 - (4-camera, custom-logic) => $N = 4$;

Timing diagram



Metadata timing diagram

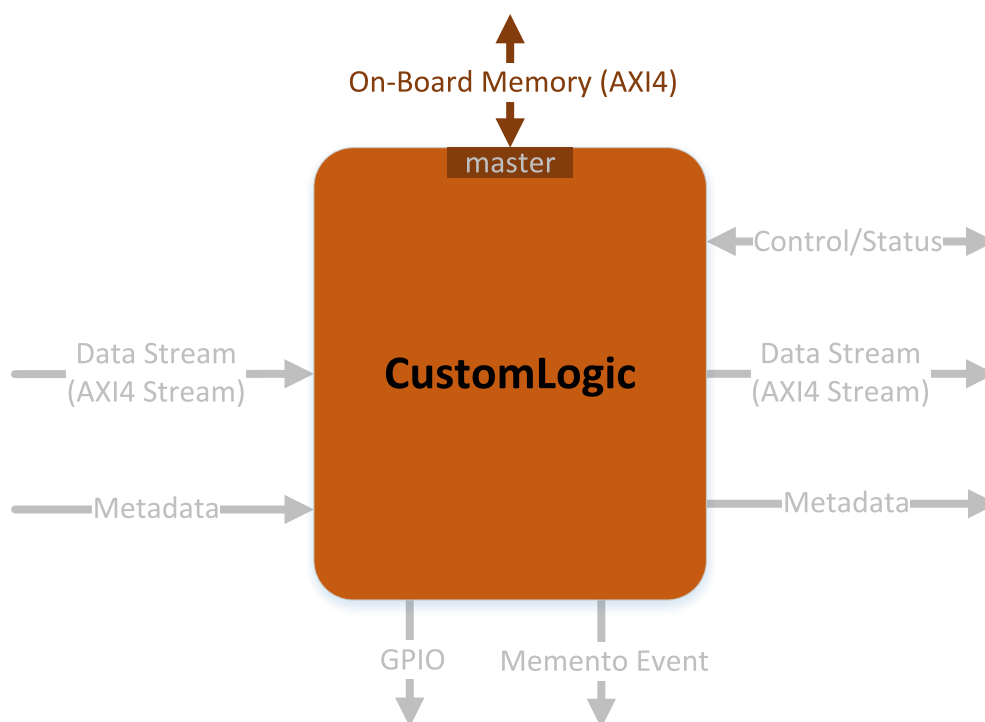
The Metadata signals are updated every time the TUSER flag SOF/SOB is asserted.

2.4. On-Board Memory Interface

In this topic:

- "On-board memory partitions" on page 21
- "AMBA® AXI4 channel architecture" on page 22
- "AMBA® AXI4 interface signals" on page 23
 - "Master interface global signals" on page 23
 - "Master write address channel interface signals" on page 24
 - "Master write data channel interface signals" on page 25
 - "Master write response channel interface signals" on page 26
 - "Master read address channel interface signals" on page 27
 - "Master read data channel interface signals" on page 28
- "AMBA® AXI4 timing diagram" on page 29

The on-board memory interface gives access to the memory resources available on the Euresys frame grabbers. It is based on **AMBA® AXI4**, an industry-standard protocol described in the **AMBA® AXI and ACE Protocol Specification**.



On-board memory partitions

The on-board memory has 2 partitions: the **CustomLogic** partition and the *FIFO Buffer partition*.

FIFO Buffer partition

This part of the on-board memory resources is dedicated to the frame grabber for the temporary storage of image data.

CustomLogic partition

This part of the on-board memory resources is dedicated to **CustomLogic**.



WARNING

Write and read operations outside of the **CustomLogic** partition must be avoided. Any write outside of the **CustomLogic** partition can corrupt data being acquired by the frame grabber.

The following parameters provide the base address and the size of the **CustomLogic** partition:

Signal	Width	Direction	Description
onboard_mem_base	32	Input	Indicates the base address of the CustomLogic partition in the On-Board Memory
onboard_mem_size	32	Input	Indicates the size in bytes of the CustomLogic partition in the On-Board Memory

The partitions sizes are product-specific:

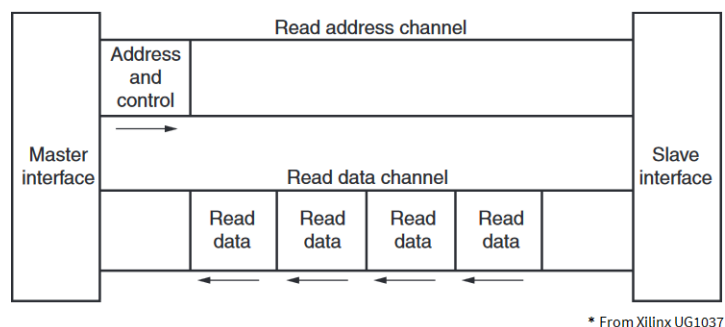
Product	Memory partition size [Gigabytes]	
	CustomLogic	FIFO Buffer
3602 Coaxlink Octo	1 GB	1 GB
3603 Coaxlink Quad CXP-12	1 GB	1 GB
3603-4 Coaxlink Quad CXP-12	2 GB	2 GB

AMBA® AXI4 channel architecture

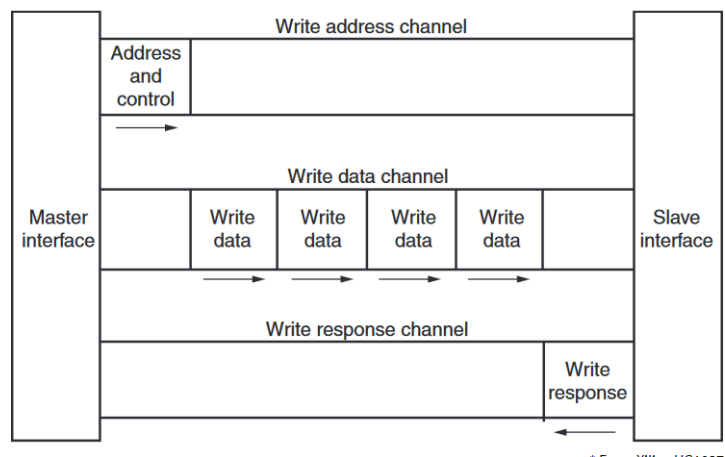
AMBA® AXI4 is a memory-mapped interface that consists of five channels:

- 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 AMBA® AXI4 is a burst transaction of up to 256 data transfers.



Channel Architecture of Reads



Channel Architecture of Writes

AMBA® AXI4 interface signals

The following sections briefly describe the **AMBA® 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 AMBA® 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_awsz	3	Output	Burst size. This signal indicates the size in bytes of each transfer in the burst. Burst_Size = 2^AWSZ[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: <ul style="list-style-type: none"> <input type="checkbox"/> AWCACHE[0] Bufferable <input type="checkbox"/> AWCACHE[1] Cacheable <input type="checkbox"/> AWCACHE[2] Read-allocate <input type="checkbox"/> 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: <ul style="list-style-type: none"> <input type="checkbox"/> AWPROT[0] Privileged <input type="checkbox"/> AWPROT[1] Non-secure <input type="checkbox"/> AWPROT[2] Instruction
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.

**WARNING**

It is strongly recommended to set `m_axi_awqos` values below 8 to not disturb the other agents connected to the On-Board Memory.

**NOTE**

The descriptions are excerpts from [AMBA® AXI and ACE Protocol Specification](#).

Master write data channel interface signals

Signal	Width	Direction	Description
<code>m_axi_wdata</code>	<code>W</code>	Output	Write data.
<code>m_axi_wstrb</code>	<code>W/8</code>	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.
<code>m_axi_wlast</code>	1	Output	Write last. This signal indicates the last transfer in a write burst.
<code>m_axi_wvalid</code>	1	Output	Write valid. This signal indicates that valid write data and strobes are available.
<code>m_axi_wready</code>	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 ;
 - (2-camera, line-scan, custom-logic) => `W` = 256;
- **3603 Coaxlink Quad CXP-12 and 3603-4 Coaxlink Quad CXP-12**
 - (1-camera, custom-logic) => `W` = 256;
 - (1-camera, line-scan, custom-logic) => `W`=256;
 - (2-camera, custom-logic) => `W` = 256;
 - (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: <ul style="list-style-type: none"> □ "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 in bytes 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: <ul style="list-style-type: none"> □ 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: <ul style="list-style-type: none"> □ ARPROT[0] Privileged □ ARPROT[1] Non-secure □ ARPROT[2] Instruction
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.



WARNING

It is strongly recommended to set `m_axi_awqos` values below 8 to not disturb the other agents connected to the On-Board Memory.



NOTE

The descriptions are excerpts from [AMBA® AXI and ACE Protocol Specification](#).

Master read data channel interface signals

Signal	Width	Direction	Description
<code>m_axi_rdata</code>	W	Input	Read data.
<code>m_axi_rresp</code>	2	Input	Read response. This signal indicates the status of the read transfer. Response: "00" = OKAY; "01" = EXOKAY; "10" = SLVERR; "11" = DECERR
<code>m_axi_rlast</code>	1	Input	Read last. This signal indicates the last transfer in a read burst.
<code>m_axi_rvalid</code>	1	Input	Read valid. This signal indicates that the channel is signaling the required read data.
<code>m_axi_rready</code>	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.

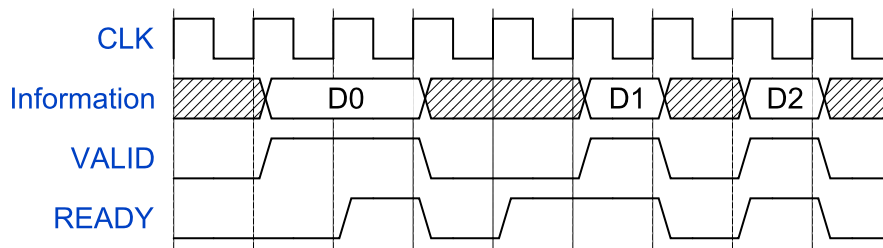
- **3602 Coaxlink Octo**
 - (1-camera, custom-logic) => $W = 128$;
 - (2-camera, line-scan, custom-logic) => $W = 256$;
- **3603 Coaxlink Quad CXP-12 and 3603-4 Coaxlink Quad CXP-12**
 - (1-camera, custom-logic) => $W = 256$;
 - (1-camera, line-scan, custom-logic) => $W=256$;
 - (2-camera, custom-logic)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



NOTE

The descriptions are excerpts from [AMBA® AXI and ACE Protocol Specification](#).

AMBA® AXI4 timing diagram



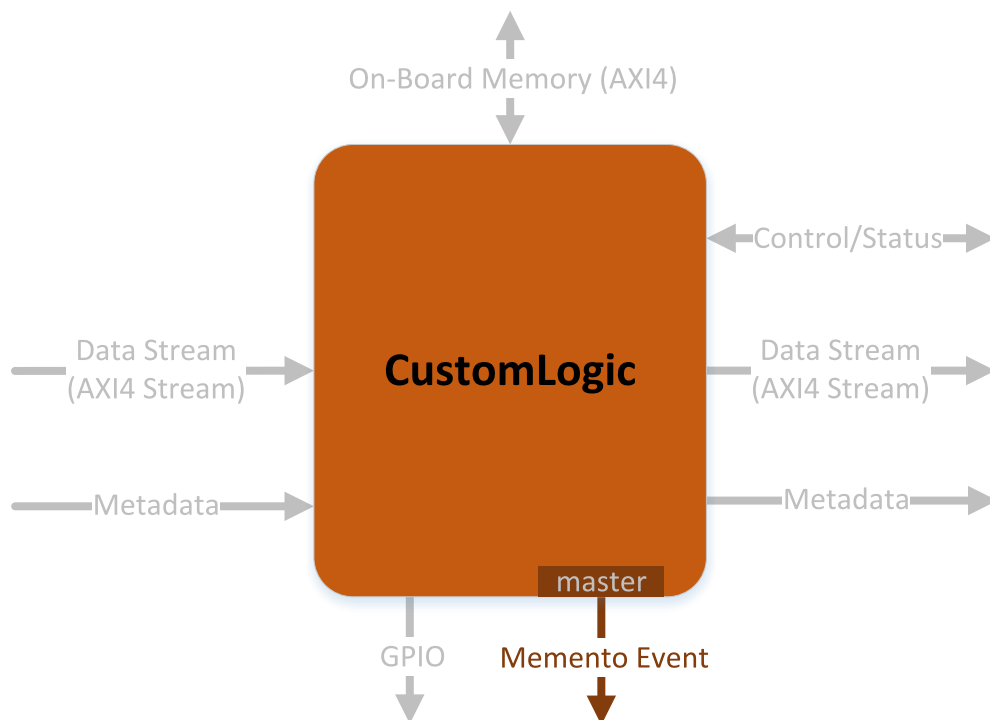
VALID/READY handshake timing diagram

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 μ s.

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

```
[ts:0195.350699] Message from CustomLogic: ARG_0 ARG_1  
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.



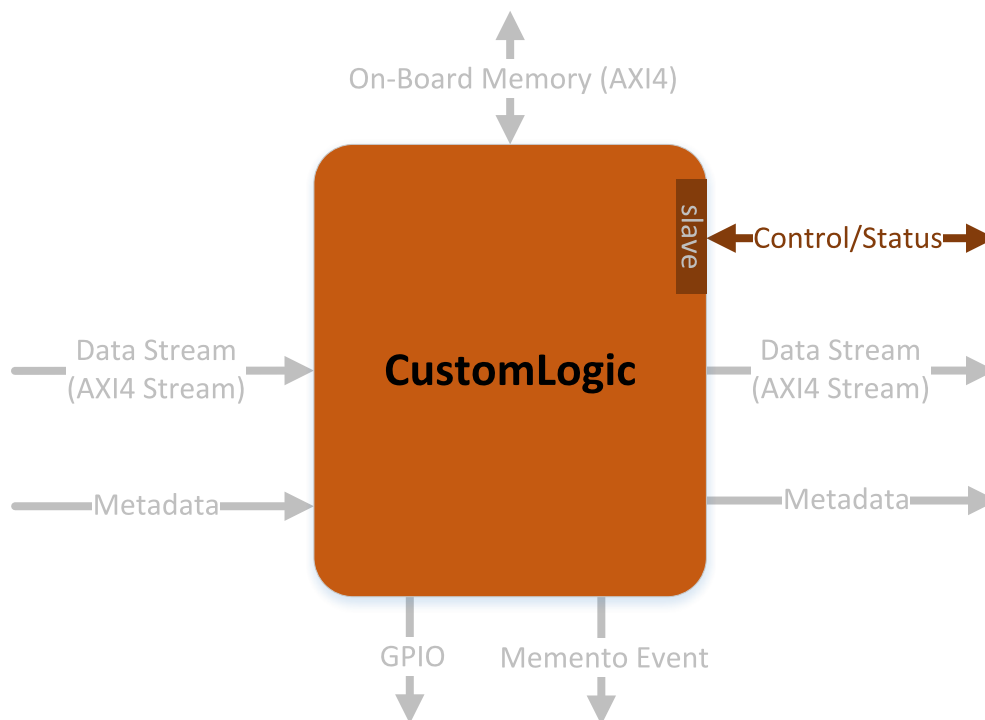
NOTE

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) => $N = 1$;
 - (2-camera, line-scan, custom-logic) => $N = 2$;
- **3603 Coaxlink Quad CXP-12 and 3603-4 Coaxlink Quad CXP-12**
 - (1-camera, custom-logic) => $N = 1$;
 - (1-camera, line-scan, custom-logic) => $N = 1$;
 - (2-camera, custom-logic) => $N = 2$;
 - (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 ["GenApi Features"](#) on page 47.



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`.

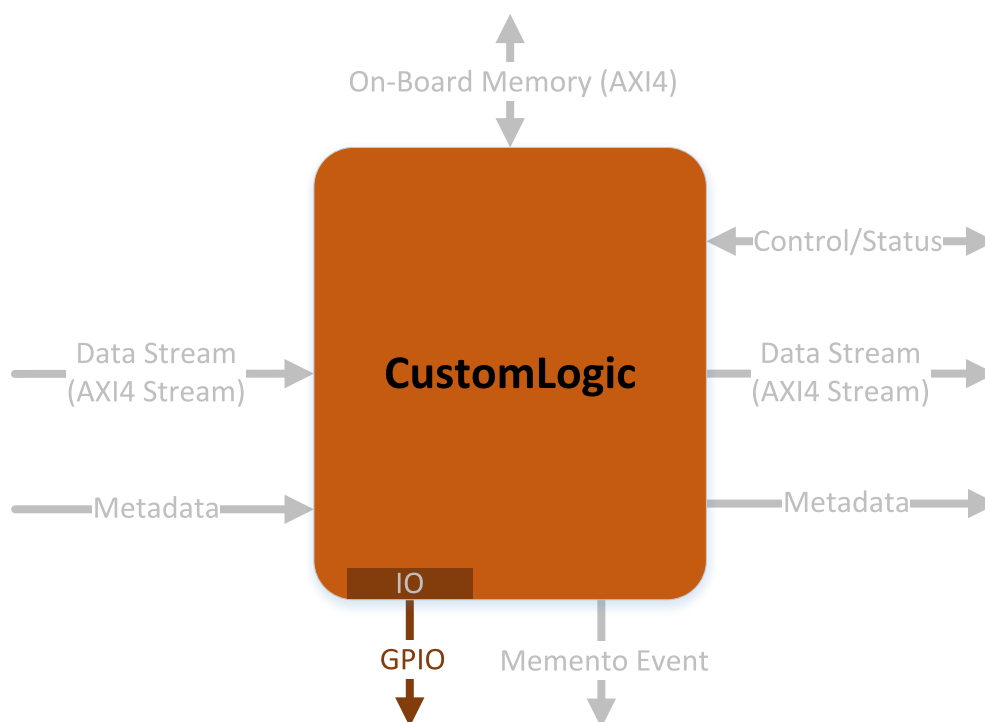
Interface signals

Signal	Width	Direction	Description
<code>s_ctrl_addr</code>	16	Input	16-bit WR/RD Address. The WR/RD Address selects the register to be read/written.
<code>s_ctrl_data_wr_en</code>	1	Input	Pulse of one cycle indicating an update in <code>s_ctrl_data_wr</code> .
<code>s_ctrl_data_wr</code>	32	Input	32-bit Write Data. Write a 32-bit vector into a selected register.
<code>s_ctrl_data_rd</code>	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 frame grabber. 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 *Functional Guide* and the *Hardware Manual* sections in the *Coaxlink series mini-site* of the eGrabber online documentation.



Interface signals

Signal	Width	Direction	Description
user_output_ctrl	16	Output	<p>Control the User Output Register:</p> <ul style="list-style-type: none"> □ 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 <p>Each UserOutput register can be controlled independently¹.</p> <p>The 'Ctrl' fields are encoded as follows:</p> <ul style="list-style-type: none"> □ "01" => UserOutputx <= '1' □ "10" => UserOutputx <= '0' □ Others => No change
custom_logic_output_ctrl	32	Output	<p>Control the general purpose outputs (TTLIO, IOOUT, MIO) available on the frame grabber:</p> <ul style="list-style-type: none"> □ Ctrl[0] => CustomLogicOutput0 □ Ctrl[1] => CustomLogicOutput1 □ Ctrl[2] => CustomLogicOutput2 □ Ctrl[3] => CustomLogicOutput3 □ Ctrl[4] => CustomLogicOutput4 □ Ctrl[5] => CustomLogicOutput5 □ Ctrl[6] => CustomLogicOutput6 □ Ctrl[7] => CustomLogicOutput7 □ Ctrl[8] => CustomLogicOutput8 □ Ctrl[9] => CustomLogicOutput9 □ Ctrl[10] => CustomLogicOutput10 □ Ctrl[11] => CustomLogicOutput11 □ Ctrl[12] => CustomLogicOutput12 □ Ctrl[13] => CustomLogicOutput13 □ Ctrl[14] => CustomLogicOutput14 □ Ctrl[15] => CustomLogicOutput15 □ Ctrl[16] => CustomLogicOutput16 □ Ctrl[17] => CustomLogicOutput17 □ Ctrl[18] => CustomLogicOutput18 □ Ctrl[19] => CustomLogicOutput19 □ Ctrl[20] => CustomLogicOutput20 □ Ctrl[21] => CustomLogicOutput21 □ Ctrl[22] => CustomLogicOutput22 □ Ctrl[23] => CustomLogicOutput23 □ Ctrl[24] => CustomLogicOutput24 □ Ctrl[25] => CustomLogicOutput25 □ Ctrl[26] => CustomLogicOutput26 □ Ctrl[27] => CustomLogicOutput27 □ Ctrl[28] => CustomLogicOutput28 □ Ctrl[29] => CustomLogicOutput29 □ Ctrl[30] => CustomLogicOutput30 □ Ctrl[31] => CustomLogicOutput31 <p>The 'Ctrl' fields are encoded as follows:</p> <ul style="list-style-type: none"> □ '0' => CustomLogicOutputx <= '0' □ '1' => CustomLogicOutputx <= '1'
user_output_status	8	Input	<p>User Output Register status:</p> <ul style="list-style-type: none"> □ Status[0] => UserOutput0 □ Status[1] => UserOutput1 □ Status[2] => UserOutput2 □ Status[3] => UserOutput3 □ Status[4] => UserOutput4 □ Status[5] => UserOutput5 □ Status[6] => UserOutput6

¹ 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 eGrabber driver API even if the corresponding 'Ctrl' field is constantly forced to "01".

Signal	Width	Direction	Description
			<ul style="list-style-type: none"> □ Status[7] => UserOutput7
standard_io_set1_status	10	Input	Standard I/O set #1 status: <ul style="list-style-type: none"> □ 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: <ul style="list-style-type: none"> □ 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: <ul style="list-style-type: none"> □ Status[0] => MIO1 □ Status[1] => MIO2 □ ... □ Status[39] => MIO40
qdc_status	Q*32	Input	Position of the Quadrature Decoder Tool



NOTE

In the Width column: “Q” refers to the total number of Quadrature Decoder tools available in the **CustomLogic** variant.

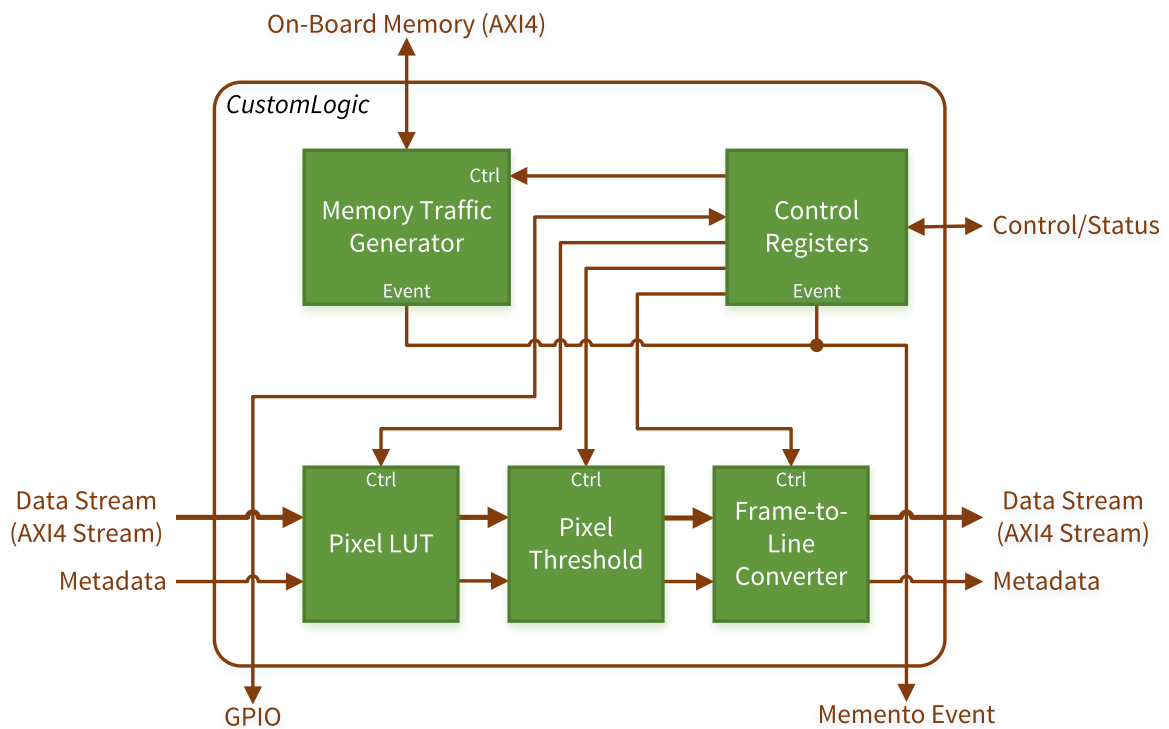
3. Reference Design

3.1. Introduction	37
3.2. Available Reference Modules	38
3.3. CustomLogic Delivery	43
3.4. Reference Design Build Procedure	44

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

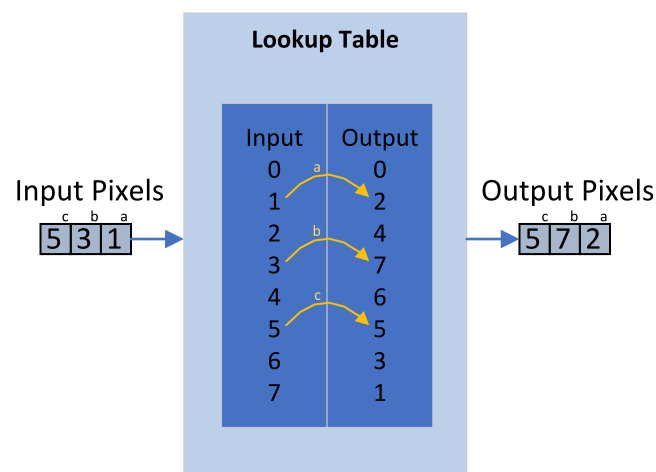
3.2. Available Reference Modules

In this topic:

- "Pixel LUT 8-bit " on page 38
- "Pixel Threshold" on page 39
- "Frame-to-Line Converter" on page 39
- "Memory Traffic Generator" on page 40
- "Memento Events" on page 40
- "General Purpose I/O" on page 40
- "Control Registers" on page 41

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** and **3603-4 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:

```
OutputPixel = 255 when InputPixel ≥ Th;
OutputPixel = 0 when InputPixel < Th;
where Th is the Threshold level.
```

The Pixel Threshold computes 16 (for **3602 Coaxlink Octo**) or 32 (for **3603 Coaxlink Quad CXP-12** and **3603-4 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 .



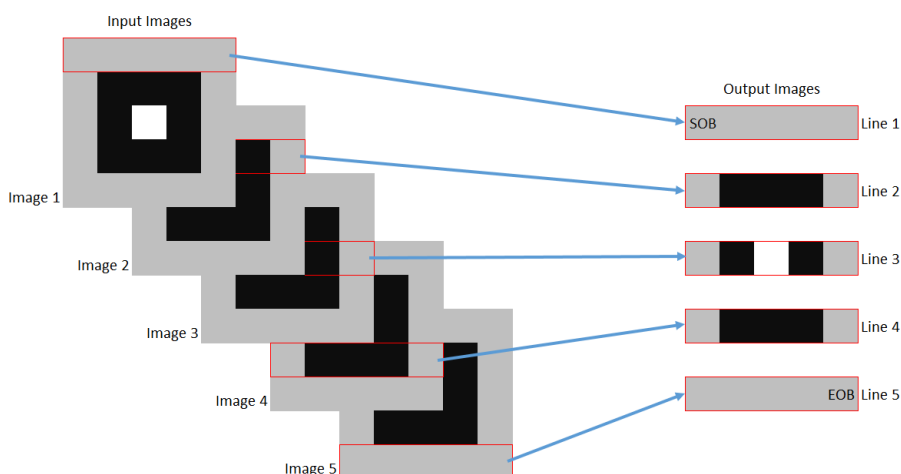
NOTE

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 0x00000000 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) <ul style="list-style-type: none"> □ 32-bit scratch pad (reset value => 0x00000000)
MemTrafficGen	0x0001	Bit 0 (R/W) <ul style="list-style-type: none"> □ 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: <ul style="list-style-type: none"> □ 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
MioSetAStatus	0x0006	Bit 0 (R) => MIO1 state Bit 1 (R) => MIO2 state ... Bit 31 (R) => MIO32 state

Register	Address	Description
MioSetBStatus	0x0007	Bit 0 (R) => MIO33 state Bit 1 (R) => MIO34 state ... Bit 7 (R) => MIO40 state
CLogicOutCtrl	0x0018	Bit 0 (R/W) => CustomLogicOutput0 Bit 1 (R/W) => CustomLogicOutput1 ... Bit 31 (R/W) => CustomLogicOutput31 Bit field encoding: <ul style="list-style-type: none"> □ When '0' => CustomLogicOutputx <= '0' □ When '1' => CustomLogicOutputx <= '1'
Frame2Line	0x1n00	Bit 0 (R/W) <ul style="list-style-type: none"> □ 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) <ul style="list-style-type: none"> □ 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) <ul style="list-style-type: none"> □ when '1' => Starts a new write sequence of coefficients Bit 4 (R) <ul style="list-style-type: none"> □ when '1' => Indicates the end of a write sequence of coefficients (reset value => '0') Bits 9:8 (R/W) <ul style="list-style-type: none"> □ 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) <ul style="list-style-type: none"> □ 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) <ul style="list-style-type: none"> □ when 0x00 => No change □ when others => Set the Pixel Threshold level (reset value => 0x01) Bits 9:8 (R/W) <ul style="list-style-type: none"> □ 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 **CustomLogic** 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_1cam
	2-camera, line-scan, custom-logic	CoaxlinkOcto_2cam_linescan
3603 Coaxlink Quad CXP-12 3603-4 Coaxlink Quad CXP-12	1-camera, custom-logic	CoaxlinkQuadCxp12_1cam
	1-camera, line-scan, custom-logic	CoaxlinkQuadCxp12_1cam_linescan
	2-camera, custom-logic	CoaxlinkQuadCxp12_2cam
	4-camera, custom-logic	CoaxlinkQuadCxp12_4cam

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).
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

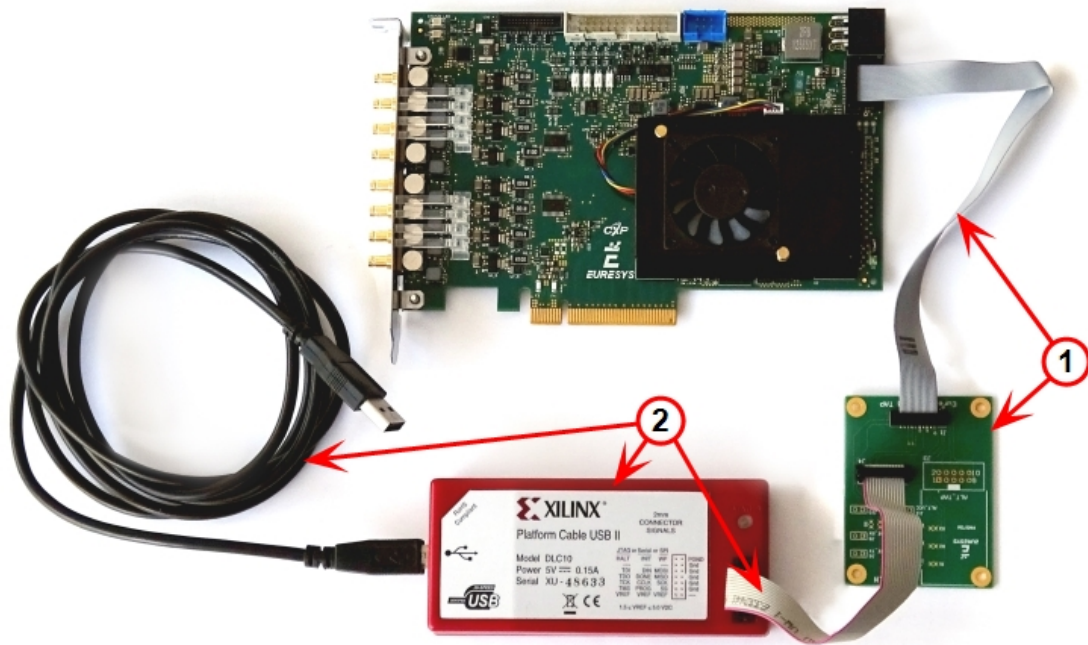
- This function requires a Xilinx JTAG programmer.
- This step is optional.

8. The generated bit-stream can also be programmed in a non-volatile way via the "[Firmware Manager Tools](#)" on page 52.

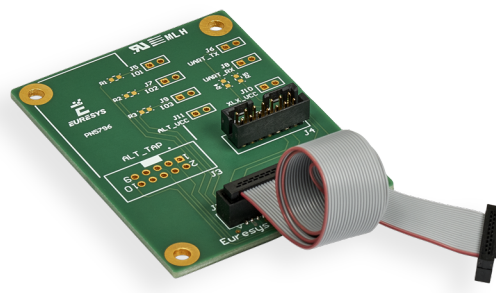
4. Debugging

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 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.

6. GenApi Features

6.1. CustomLogicControlAddress	48
6.2. CustomLogicControlData	49

See also: "GenApi Features > Interface Module > CustomLogic Category" section in the Coaxlink series Handbook

6.1. CustomLogicControlAddress

[Feature Info](#)

Module	Category Path	Type	Access
Interface	Root → CustomLogic	Integer	RW

[Value Info](#)

Minimum value: 0

Maximum value: 65535

[Short Description](#)

Custom Logic Control Address.

[Selected Features](#)

- ["CustomLogicControlData"](#) on page 49

6.2. CustomLogicControlData

[Feature Info](#)

Module	Category Path	Type	Access
Interface	Root → CustomLogic	Integer	RW

[Value Info](#)

Minimum value: 0

Maximum value: 4294967295

[Short Description](#)

Custom Logic Control Data.

7. Managing Firmware

- 7.1. What's Firmware? 51
- 7.2. Firmware Manager Tools 52
- 7.3. Updating and Installing Firmware 54
- 7.4. Special Firmware Procedures 55
- 7.5. Firmware Recovery Switch 58

7.1. What's Firmware?

Firmware

In this context, "firmware" designates the content of the FPGA (Field Programmable Gate Array) device of a card.

It defines the functionality of the card including the PCI Express end-point.

Firmware EEPROM

The FPGA used on **Coaxlink and Grablink Duo frame grabbers** is RAM-based; it needs to be loaded every power up.

Considering that a PCI Express end point must be ready within 150 milliseconds of the power-up time, the FPGA content, must be loaded quickly after having applied power to the card. Therefore, the firmware is stored into a non-volatile flash EEPROM allowing a fast start-up of the FPGA.



NOTE

This situation differs from other Euresys products, such as MultiCam cards, where the FPGA content is loaded by the MultiCam driver when it starts or at any time if a FPGA configuration change is requested during operation.



TIP

The **eGrabber** driver will never modify the content of the FPGA during operation.

Firmware modifications

Any modification of the FPGA content requires a two-step operation:

1. The new firmware is written into the Flash EEPROM of the card using a firmware management tool.
2. The new firmware is activated by cycling the system power.

7.2. Firmware Manager Tools

eGrabber is delivered with two firmware management tools:

- "Firmware Manager - GUI mode" on page 52 : A graphical user interface tool in **eGrabber Studio**,
- "Firmware Manager - Command line mode" on page 52 : A command-line tool named **Firmware Manager Console**.

Firmware Manager - GUI mode

To open the **Firmware Manager** in GUI mode, select one of the following methods:

- From the Windows Start Menu: click on Firmware Manager shortcut in the **Euresys eGrabber** folder
- From the **Welcome Screen** of **eGrabber Studio**, click on the **Firmware Manager** button.

See also: "Firmware Manager (GUI mode)" section in the eGrabber Studio User Guide for a detailed description.

Firmware Manager - Command line mode

Access

The command-line tool is named `coaxlink-firmware.exe`. It is located in the firmware sub-folder of the eGrabber installation folder.

On Windows, to open the **Firmware Manager** in command-line mode, select one of the following methods:

- From the Windows Start Menu: click on Firmware Manager console shortcut in the **Euresys eGrabber** folder
- Open a command prompt and open in the `C:\Program Files\Euresys\eGrabber\firmware` folder

On Linux, to open the **Firmware Manager** in command-line mode:

- Open a command shell in the `/opt/euresys/egrabber/firmware` folder

On macOS, to open the **Firmware Manager** in command-line mode:

- Open a command shell in the `/usr/local/opt/euresys/egrabber/firmware` folder.

Main commands

- Executing `coaxlink-firmware --help` displays a help message describing all the command options.
- Executing `coaxlink-firmware gui` starts the **Firmware Manager (Deprecated)** graphical user interface.

- Executing `coaxlink-firmware list` lists the properties of the firmware installed on each card present in the system.
- Executing `coaxlink-firmware update` updates the firmware.
- Executing `coaxlink-firmware install` installs a new firmware variant.

Unless specified with a `--firmware=FILE` option, the tool uses the embedded library.

7.3. Updating and Installing Firmware



WARNING

Prior to executing this procedure, read the "Important Notices" section of the release notes!

The **eGrabber** driver comes with all the firmware variants for all the **Coaxlink and Grablink Duo frame grabbers**.

1. Determine the firmware variant that fulfills the functional requirements of your application: e.g. '1-camera', '1-camera, line-scan', '2-camera'. Therefore, check the *Firmware Variants per Product* section of the release notes for the firmware variants that are applicable to your card.
2. Launch a [Firmware Manager tool](#) to perform a firmware *update* or to *install* a specific firmware variant on your card(s) using the **Firmware Manager** tool in *GUI mode* with **eGrabber Studio** or the **Firmware Manager Console** in *command-line mode*:
 - a. In **eGrabber Studio**, open the **Firmware Manager** pane:
 - Select the card to update
 - Select the firmware variant to install
 - Proceed with the installation
 - b. In command-line mode, to *update* a variant:
coaxlink-firmware update
 - c. In command-line mode, to *install* another firmware variant:
coaxlink-firmware install '[variant-name]'
3. Wait until completion of the firmware update



WARNING

Avoid turning off your PC during the firmware update procedure!

4. Repeat the procedure on all your **Coaxlink and Grablink Duo frame grabbers**.
5. **Power off completely your PC** and restart it to activate the newly loaded firmware.

7.4. Special Firmware Procedures

In this topic:

- "GUI mode downgrade procedure" on page 55
- "Command-line mode downgrade procedure " on page 56
- "Command-line mode recovery procedure" on page 56
- "Recovery procedure with recovery switch" on page 57

Directives

- Execute either the "GUI mode downgrade procedure" on page 55 or the "Command-line mode downgrade procedure " on page 56 only when the application absolutely requires an older firmware version!
- Execute either the "Command-line mode recovery procedure" on page 56 or the "Recovery procedure with recovery switch" on page 57 only in case in case of card malfunction after installation of a new firmware!

GUI mode downgrade procedure



WARNING

For **Coaxlink and Grablink Duo frame grabbers** having a Serial Number above or equal to 10,000: this procedure must be executed on a PC with a driver version 10.0.0 or higher installed!

1. Open the **Firmware Manager** pane in **eGrabber Studio**
2. In the **Details** view, click on the **File** button to select an alternate firmware source
3. Select the coaxlink-firmware.exe file delivered with the old driver required by the application
4. In the **Cards** view:
 - a. Select the card to downgrade
 - b. Select the firmware variant to install
 - c. Proceed with the installation

Command-line mode downgrade procedure



WARNING

For **Coaxlink and Grablink Duo frame grabbers** having a Serial Number above or equal to 10,000: this procedure must be executed on a PC with a driver version 10.0.0 or higher installed!

From the **Firmware Manager Console** executes one of the following commands:

- Keeping the same firmware variant:
`coaxlink-firmware update --firmware=PATH_TO_FILE`
- Changing also the firmware variant:
`coaxlink-firmware install VARIANT_TO_INSTALL --firmware=PATH_TO_FILE`

PATH-TO_FILE is the path to the coaxlink-firmware.exe file delivered with the old Coaxlink driver required by the application.

Command-line mode recovery procedure



WARNING

For **Coaxlink and Grablink Duo frame grabbers** having a Serial Number above or equal to 10,000: this procedure has to be executed on a PC with a driver version 10.1.2 or higher installed!

1. From the **Firmware Manager Console**, execute the bank selection command:
`coaxlink-firmware bank-select --next=ALTERNATE`
The command displays a status indicating that the next firmware after boot is the other bank:
[BANK0: current firmware][BANK1: alternate/next firmware] or
[BANK0: alternate/next firmware][BANK1: current firmware]
2. Power off the PC
3. Power on the PC

Recovery procedure with recovery switch



WARNING

(*) For **Coaxlink and Grablink Duo frame grabbers** having a Serial Number above or equal to 10,000: this procedure has to be executed on a PC with a driver version 10.1.2 or higher installed!

1. Power off the PC:
 - a. Remove the card from the PC
 - b. Set the "[Firmware Recovery Switch](#)" on [page 58](#) of the card to the "Recovery" position
2. Power off a PC*
 - a. Insert the card into the PC
 - b. Power on the PC
 - c. Execute a "[Updating and Installing Firmware](#)" on [page 54](#)
 - d. Power off the PC
 - e. Remove the card
 - f. Set back the "[Firmware Recovery Switch](#)" on [page 58](#) to the "Normal" position

7.5. Firmware Recovery Switch

Switch types and location

The *firmware recovery switch* is implemented with one of the following components:

- 3-pin header and a jumper
- 2-way DIP switch

See also: Board and Bracket Layouts in the Coaxlink series Handbook or in the Grablink Duo Handbook to locate the firmware recovery switch. These drawings show its normal position.

Switch positions



The *firmware recovery switch* has two positions:

Normal position (factory default)

At the next power ON, the latest firmware successfully written into the Flash EEPROM is used to program the FPGA. After FPGA startup completion, the card exhibits the *standard PCI ID* and the driver allows normal operation.

Recovery position

At the next power ON, the last but one firmware successfully written into the Flash EEPROM is used to program the FPGA. After FPGA startup completion, the card exhibits the *recovery PCI ID* and the driver inhibits image acquisition.

Switch type	Normal position	Recovery position
3-pin header and a jumper		
2-way DIP switch	