BeastLink v1.0

Cesys Product Guide for BeastLink v1.0 IP Core.

Introduction

The BeastLink IP Core is an efficient bridge between the Cypress FX3 USB controller's slave FIFO Interface (GPIF-II) and ARM's Advanced Extensible Interface (AXI-4) bus system. The IP Core facilitates an easy and efficient connectivity between a host PC with Superspeed Universal Serial Bus (USB3.0) and a number of on-board peripherals controlled by an FPGA. The IP provides a ready solution for systems requiring USB communication. This is accomplished by interfacing the widely used Cypress FX3's slave FIFOs and AXI4-Memory-Mapped peripherals, for example, Memory Interface Generator, Block RAM Controller, General Purpose IOs, AXI-XADC etc. Such peripherals are readily available in the Vivado Ultrafast design methodology IP catalog. Users can also interface their custom AXI4 protocol based peripherals (IPs) with the BeastLink IP. The IP Core is intended to be used in FPGA designs where an interface between FX3's slave FIFOs and AXI4-Bus system is required. The BeastLink v1.0 IP Core is designed to be used in Xilinx's Vivado Design Suite.

The BeastLink v1.0 IP Core is a part of the Cesys BeastLink Library that provides a complete solution for systems requiring Highspeed USB communication between a host PC and an FPGA Design. The host-software can read/write all the functional blocks of the design by simply calling read/write API functions of the Cesys BeastLink Library. The referenced addresses are embedded in the BeastLink protocol, transferred serially over the Universal Serial Bus (USB) and are stored intermediately in the FX3's slave FIFOs. The BeastLink IP Core, configured as the master, reads/writes the slave FIFOs in FX3 over a General Programmable Interface (GPIF-II) and translates the commands/data into AXI4-Full cycles to access the on-chip registers or on-board peripherals.

Features

- Supports AXI-4 Memory-Mapped Interface specifications
- Supports FX3's GPIF-II slave FIFO interface
- Uses DMA to transfer data to and from Memory-mapped peripherals
- Supports transfer sizes of upto 8 Megabytes
- Connects seamlessly with AXI based peripherals
- Supports AXI-4 Full and Lite protocols
- Provides a 4G address space
- Supported on all Xilinx 7-series FPGA parts

IP Facts

IP Information

Supports	Further Details
Target Language	VHDL
Devices	Xilinx 7-Series FPGAs
User Interfaces	AXI4-Memory-Mapped and GPIF-II
Tool	Vivado 2017.4 onwards

Inclusions

Particular	Evaluation Version	Full Version
Design Files	Encrypted IP	VHDL
Example Design	VHDL	
Simulation files	Verilog + VHDL	
Constraints files	XDC	
Support	http://cesys.com/download/fpga/	

Resource Utilization

The BeastLink IP-Core is synthesized and implemented in Vivado version 2017.4. It was tested on the "XC7A200TFBG676-2" part embedded on the Cesys EFM03 Beastboard. The Post-Implementation Resource Utilization report can be seen in the below table.

Resource	Utilization
LUT	1797
LUTRAM	176
FF	2030
BRAM	26.50
IO	261
BUFG	2

Applications

- High performance PC peripherals
- Image processing
- DSP co-processing
- Embedded Control
- Custom test equipment
- Data acquisition daughter cards

Advantages

The IP Core has several advantages like:

1. Provides a ready solution for systems using Superspeed USB (USB-3.0)
2. Greatly reduces development time in USB-based designs
3. Interfaces 2 widely used technologies; AXI bus system and FX3 Chipset
4. Facilitates high data-rates upto 315 MB/s for Read and Write accesses
5. Proven stable performance for long durations with rigorous tests
6. Backward compatibility with USB2.0

Quick Start

This section presents a brief explanation about obtaining the IP Core and adding the design files to the IP Catalog and eventually, using it in the project. To know more about the IP Core, generating its outputs and running synthesis, implementation and bitstream generation, please go through the "Overview" and "Example Design" sections of this document.

Obtaining the IP-Core files

The BeastLink v1.0 IP Core is available as part of the Cesys BeastLink Library that can be evaluated or purchased. The core's design files and an example design are packed in an archive "BeastLink_IP.zip". Using the login information that you received from our customer support, please download the archive from our website (https://www.cesys.com).

Adding the BeastLink IP to the IP Catalog

The IP core can be added to the Vivado IP Catalog by following the below steps:

1. Create a new Vivado project or open the Vivado project in which you want to add the BeastLink IP
2. Make sure the project's target language is VHDL
3. In the **Flow Navigator** window, under **Project Manager** tab, open **IP Catalog**
4. In the IP Catalog Window, Right-Click and select **Add Repository...** option
5. Navigate to the location where you have unzipped the above mentioned archive
6. Select BeastLink_IP folder and press **Select**

Now, in the **IP Catalog** window, you must be able to see a section called User Repository. Under **User Repository->UserIP** one can see BeastLink v1.0. To add the IP to a design as source (without block design), follow the below steps:

1. In the **Flow Navigator** window, under **Project Manager** tab, open **IP Catalog**
2. Under **User Repository->UserIP**, double click on BeastLink v1.0 IP
3. In the customization window, press **OK**
4. In the **Generate Output Products** popup, under *Synthesis Options*, select **Global** and press **Generate**
5. The IP Core will be added to your design

To add the BeastLink v1.0 IP to a block design in your project, follow the below

steps:

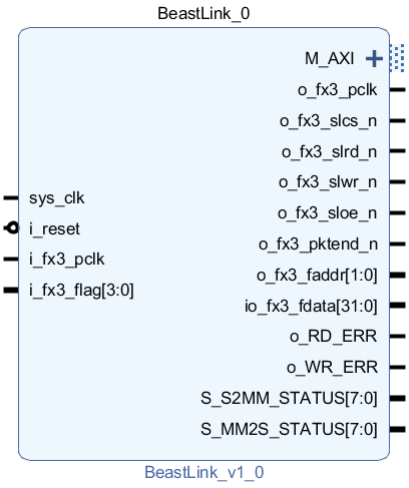
1. Select **Create Block design** under **IP Integrator** tab in **Flow Navigator** window
2. If you already have an existing Block design open, move to the **Diagram** window
3. Right click on the canvas and select **Add IP...** option
4. In the IP selection dropbox, search for or navigate through the list to find BeastLink v1.0.
5. Select BeastLink v1.0 and the IP should appear on the canvas
6. Make necessary connections and save the block design. Under **IP Integrator** Click on **Generate Block Design**
7. In the **Generate Output Products** popup, under *Synthesis Options*, select **Global** and press **Generate**
8. The IP Core will be added to your design

Overview

A detailed description of the BeastLink IP Core is presented in this chapter. The IP's ports, the internal structure and functionality are described here.

Port Description

The below figure shows the BeastLink v1.0 IP core GUI symbol.



The BeastLink v1.0 IP-Core GUI symbol

BeastLink Port Description

Signal	Interface	I/O	Default	Description
sys_clk	Clock	I	NA	System Clock. This is a 100MHz clock.
i_fx3_pclk	Clock	I	NA	FX3 GPIF state machine clock. This clock can either externally be provided or the on-board loop-back clock can be used. Please Refer Hardware reference guide for Cesys EFM03 Beastboard for further details. This is a 100MHz clock.
i_reset	Reset	I	0	System Reset, active Low.

Signal	Interface	I/O	Default	Description
i_fx3_flag (3:0)		I	0000	Flags from the FX3 slave FIFOs. They can be programmed to display different levels of the slave FIFO. For this IP core, the flags must be programmed as follows: Bit '0': FX3_POP_IS_EMPTY FX3 slave FIFOs are empty Bit '1': FX3_POP_POSSIBLE FX3 slave FIFO has at least 8Kbyte data to be read Bit '2': FX3_PUSH_IS_FULL FX3 slave FIFOs are full Bit '3': FX3_PUSH_POSSIBLE FX3 slave FIFO has at least 8Kbyte space
o_fx3_faddr (1:0)	Slave FIFO Address	O	00	FX3 slave FIFO Address. The FX3 slave consists of 4 slaves which can be used to achieve high performance. For this IP, only 2 slave FIFOs are utilized. Bit '0': Always 0 Bit '1': 0 -> Host-2-Peripheral transfers Use slave FIFO socket-0 for H2P transfers Bit '1': 1 -> Peripheral-2-Host transfers Use slave FIFO socket-2 for P2H transfers
io_fx3_fdata	Slave FIFO Data bus	IO	Z	FX3 controller's data bus. This bus is a bidirectional bus. It needs an additional signal to control the data direction. See o_fx3_sloe_n.
o_fx3_slwr_n	write enable	O	1	The FX3 slave Interface's Write Enable signal. The IP core asserts this signal when it has data to write to FX3 slave FIFO. This signal is asserted after the o_fx3_slcs_n signal is asserted. This is an active low signal.
o_fx3_slrd_n	read enable	O	1	The FX3 slave Interface's Read Enable

Signal	Interface	I/O	Default	Description
				signal. The IP core asserts this signal when it wants to read data from the FX3 slave FIFOs. This signal is asserted after the o_fx3_slcs_n and o_fx3_sloe_n signals are asserted. This is an active low signal.
o_fx3_sloe_n	output enable	O	1	The FX3 slave Interface's Output Enable signal. The IP core asserts this signal when it wants to read data from the FX3 slave FIFOs. This signal is asserted after the o_fx3_slcs_n signal and before the o_fx3_slrd_n signals are asserted. This is an active low signal. This signal is the output enable for the Bidirectional data port. '0' -> io_fx3_fdata behaves as input port The slave drives the data bus. If there is no input from the slave, the IP core pulls the data bus into a HIGH Impedence state (Z). '1' -> io_fx3_fdata behaves as output port. The master drives the data bus. The slave pulls the data bus into a HIGH Impedence state (Z).
o_fx3_slcs_n	Slave FIFO Chip Select	O	1	FX3 controller's Chip Select Signal. This is an active low signal.
o_fx3_pktend_n	Slave FIFO packet end signal	O	1	FX3 controller's Packet end signal. This design does not make use of packet end signalling. This signal is always deasserted in this IP-Core. This is an active low signal.
o_fx3_pclk	PCLK for FX3	O	NA	FX3 controller's Clock Signal. Maximum of 100MHz clock frequency can be supplied to the FX3 controller. This IP Core provides a 100MHz Clock through this port. If the FX3 firmware

Signal	Interface	I/O	Default	Description
				changes the FX3 input clock frequency, this port must be configured to provide the appropriate clock frequency.
M_AXI	AXI Master	NA	NA	AXI4-Full Master bus. These signals follow AXI Specifications as mentioned in the *Appendix A* of the Vivado AXI reference Guide (UG1037)for AXI4, AXI4-Lite and AXI-Stream Signals.

Clocking

The BeastLink v1.0 IP-Core operates on the **sys_clk**. It is a **100MHz** Clock. This clock is internally connected to the "o_fx3_pclk" output which drives the on-board FX3. Users must make sure that the "o_fx3_pclk" output is looped back in hardware to an FPGA ball. This looped back clock must drive the "i_fx3_pclk" port in the BeastLink IP Core.

Important!

Use the same clock signal to drive the AXI bus system in the design and the sys_clk of BeastLink IP.

Resets

The BeastLink v1.0 IP-Core is reset when the **i_reset** is asserted. This signal is an **active-LOW** reset.

Important!

*The reset on the Cesys EFM03 Beastboard provides an active-HIGH reset to the FPGA on pin **K15**. However, the reset port **i_reset** in the BeastLink v1.0 IP-Core is an **active-LOW** reset. Users have to convert the active-HIGH signal to an active-LOW reset.*

Hint!

*If an MMCM is used to generate the 100MHz sys_clk, then the i_reset port can be connected to MMCM's **clock locked** signal. If a Memory Interface Generator (MIG) IP Core is used in the design, use the 100MHz **ui_clk** output of the MIG to drive the sys_clk port. Further, if MIG IP is used, then the **i_reset** input must be an active-LOW*

signal resulting from a combnination of the init_calib_complete, mmcm_locked and the ui_clk_sync_rst signals. If any other components used in the design have to wait for initialization, then appropriate signals must be added to the above combination. For example, if the user design contains a STARTUPE2 primitive, then the ***End Of Startup (EOS)*** *signal of the STARTUPE2 primitive must also be included in the above mentioned combinatorial logic. This ensures that the BeastLink IP is held in reset until all the dependent modules have initialized properly.*

Functional Description

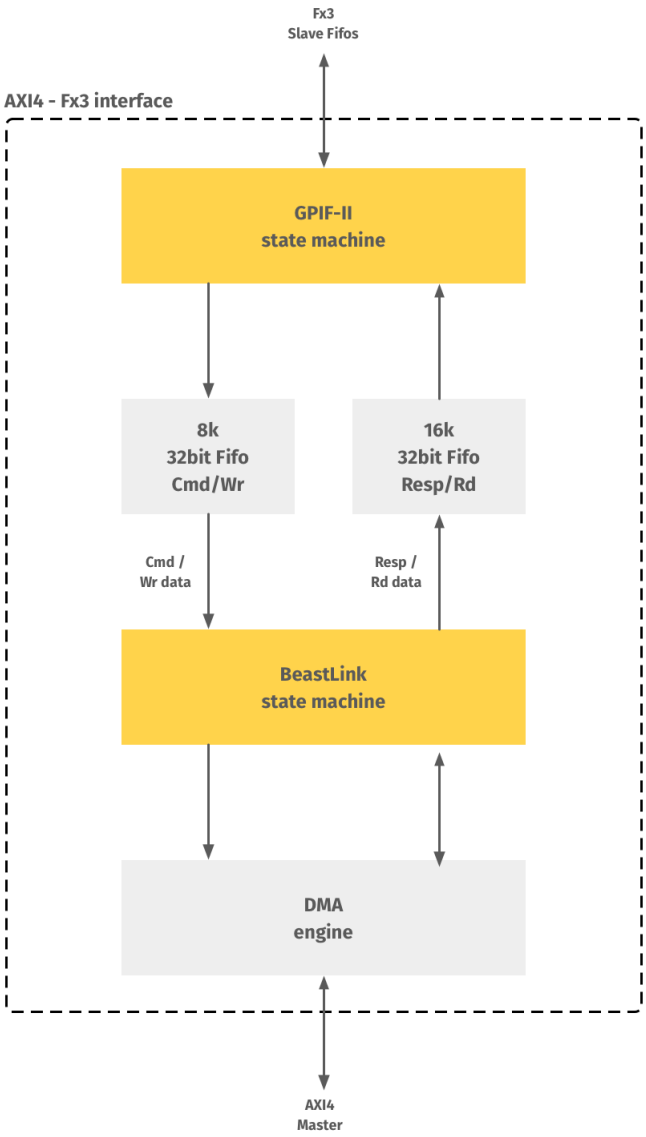
Current FPGA design trends based on Xilinx parts have mostly moved towards the block design methodology. The BeastLink IP Core fits perfectly into this trend and provides an easy and efficient translation between the FX3's GPIF-II slave FIFO interface and the widely used AXI4 bus system. The IP Core communicates with the FX3 slave FIFOs over a GPIF-II interface, interprets the commands issued by the host using the BeastLink protocol (BeastLink Protocol Structure and also **AN101** for *UDK3 Transfer Protocol*). It then issues DMA transfers to perform AXI4 accesses to the peripherals present in the design. This section describes the internal structure of the BeastLink IP and its functionality in detail.

Internal Structure

The BeastLink IP core, as shown in figure below, consists of the following important components and state-machines:

- GPIF-II State Machine
- CMD/Write and Resp/Read FIFOs
- UDK3 State machine
- DMA Engine

The figure below gives an overview of the internal structure of the BeastLink IP core. Each of these components and state-machine functionality are described in detail in the upcoming sections.

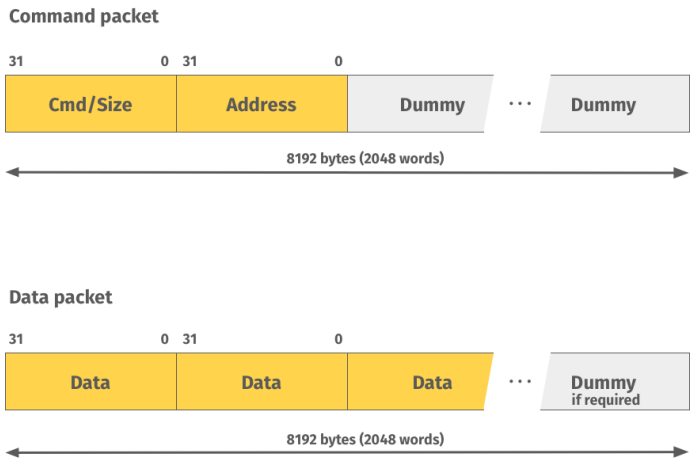


Broad overview of the internal structure of BeastLink v1.0 IP-Core

BeastLink Protocol Structure

The following description of the BeastLink data structure is for information only. It is used internally to serialize the address-based transfers. The structure can be treated as a black box when using the BeastLink in own designs. The data structure is similar to the structure of the previous UDK/UDK3 data structures. The read/write APIs in the

Cesys BeastLink Library pack the commands and data to be transferred in 8Kbyte blocks as shown in the figure below.



BeastLink protocol used by the Cesys BeastLink Library for transferring data

On response to an API call in the Host software, the Host sends an 8Kbyte Command packet as shown in the figure above. Based on the transfer direction bit in CMD/SIZE field, the Host or the Device sends the Data packet. If the data packet is less than 8192 bytes, then the Cesys BeastLink Library or BeastLink IP automatically inserts dummy data to the remainder of the packet.

The Command and Data packet fields are explained in the table below.

Field	Size	Default	Description
CMD/SIZE	32 bits	0	Bit 31: Transfer Direction '1' -> Host-2-Peripheral(H2P) '0' -> Peripheral-2-Host(P2H) Bit 30: Addressing Mode '1' -> Auto-increment (API Default) '0' -> Constant Addressing Bit 29:24: RESERVED Bit 23:0 : Payload size in WORDS

Field	Size	Default	Description
ADDRESS	32 bits	0	Start Address of the current transfer. The BeastLink IP expects Word-aligned address only. This is a limit imposed by the DMA.
DUMMY		0	Dummy WORDS to fill the 8Kbyte block. Only complete blocks are transferred. No short packet implementation is supported.
DATA	4 to 8192	0	This field contains the payload of the current transfer. The transferred amount of data must always be a multiple of the transfer block size (8Kbyte). Unused buffer space must be filled with DUMMY data. Dummy data is automatically inserted in data packets by the BeastLink IP or the Cesys BeastLink Library based on Transfer direction.

GPIF-II State Machine

The Cypress FX3 USB controller transfers data to its master over a slave FIFO interface and this is called the 2nd generation General Programmable Interface (GPIF-II). The FX3 controller sets output flags to indicate the slave FIFO levels to the master requesting data from or providing data to it. The BeastLink IP Core is designed to be the master. Based on the FX3's flags, the IP reads the FX3's slave FIFOs if a Host-2-Peripheral command is issued and stores the command in the CMD/Write Data FIFO. If a Peripheral-2-Host command is issued, the IP checks the FX3's FIFO levels and if there is an 8Kbyte space available, it initiates a read transfer to FX3 slave FIFOs.

CMD/Write and Resp/Read FIFOs

The IP-Core contains an 8K deep (CMD/Write FIFO) and a 16K deep FIFO (Resp/Read FIFO), both with a data-width of 32-bits. Therefore, each of the FIFOs can hold atleast 4 complete 8Kbyte BeastLink protocol frames (See BeastLink Protocol Structure). The GPIF-II State machine writes the commands or write data to the CMD/Write FIFO and reads peripheral data through the Resp/Read Data FIFO. These FIFOs are setup to hold atleast one complete USB3.0 Data field of size 8Kbyte at a time. The FIFOs are configured as "Independent Clocks, Block RAM" so that they operate at independent Read and Write clocks and instantiates Block RAMs for storage. The transaction begins as soon as there is data in the CMD/Write Data FIFO. It is the

responsibility of the user to provide 8Kbyte frames.

UDK3 State machine

This module decodes the Cesys UDK3 protocol (See also **AN101** for *UDK3 api specifications*), issues commands and provides data to the DMA engine. Based on the commands issued and the data given, the DMA Engine initiates a transfer to or from the addressed peripheral.

If the host issues a Host-2-Peripheral command, then the command is followed by data. The UDK3 state machine reads the CMD/Write Data FIFO and decodes the command. It then reads out the dummy bytes stuffed to the 8Kbyte command packet and awaits data. As soon as the data is seen in the CMD/Write Data FIFO, the UDK3 state-machine issues a WRITE command to the DMA engine. The DMA engine then reads data directly from the Write Data FIFO and transfers it to the Peripheral which was addressed in the command.

If the host issues a Peripheral-2-Host command, then the GPIF-II stores this command in the CMD/Write Data FIFO. The UDK3 State-machine reads this command and issues a READ command to the DMA Engine. The DMA engine in turn reads the Peripheral addressed by the command and stores the read data directly in the Resp/Read Data FIFO.

According to the UDK3 protocol, each transfer is 8Kbyte long and if the whole 8Kbyte is not used, the remaining data fields must be stuffed with DUMMY values (0x00). For a CMD/Write transfer, the DUMMY insertion has to be done by the HOST. For Resp/Read transfers, the UDK3 State-machine does this automatically once the requested number of bytes has been written into the Resp/Read Data FIFO.

Elimination of Read-Write Collision

The Cesys BeastLink Library sends commands and data to (or expects data from) the FPGA design sequentially over a USB interface. The DMA Engine on the other hand, consists of separate FIFOs for both, Read and Write accesses, thus facilitating actual parallel execution of Read/Write accesses which is a characteristic feature of the AXI4 Bus. However, most peripheral modules consist of a single bus to Read/Write to the on-board peripheral and do not support parallel Read/Write accesses. When a slower peripheral is connected to the system and a Write followed by a Read access is executed, a Read-Write collision may occur at the AXI-slave Peripheral. This is because

a Write access is still in progress and a Read access is serviced by the AXI slave (parallelism). This collision occurs especially when writing to very slow peripherals such as a Flash Controller, over AXI-Lite Interface, in which write accesses typically takes 10-15us to complete.

When an AXI-Lite slave is connected to the master interface of AXI Interconnect, the interconnect automatically instantiates an AXI Protocol Converter (auto PC) in order to translate AXI-Full bus cycles (from DMA engine) into AXI-Lite cycles. The auto PC has a latency of 2 AWREADY-AWVALID handshakes (or ARREADY-ARVALID). Such an implementation may cause a Read-Write collision between the last write and the first read access at the peripheral's AXI slave Interface which is not in the BeastLink IP's control.

The UDK3 state machine eliminates any collision that can be caused at the BeastLink IP's master which is driving the AXI Interconnect. It is acheived by ensuring a sequential execution of host commands on the DMA's Memory-Map write interface(M_AXI_S2MM). This is accomplished by simply waiting for the DMA Engine to complete the current write access to the AXI Interconnect. Only after The DMA Engine signals the end of the write transfer (xfer_cmplt), the next command will be processed. However, when an AXI4-Lite interface is connected to the master interface of the AXI Interconnect, as explained earlier, is out of scope of the UDK3 state machine. The slave Peripheral should not accept read accesses when write accesses are present at the interface.

In custom peripherals based on AXI-Lite slave interface, users can acheive sequential accesses by waiting for the write transfers to complete before asserting the 'ready' signal of the Read Address channel (ARREADY) of the slave. The read accesses will be stalled until the slave has finished writing to the peripherals. An example code is shown here:

```
if (axi_arready = '0' and S_AXI_ARVALID = '1'  -- Previous handshake completed
  and  peripheral_ready = '1'                  -- peripheral is ready
  and S_AXI_AWVALID = '0' and axi_awready = '0') then  -- when no writes (SEQ)
  -- slave has accepted valid RD_addr
  axi_arready <= '1';
  axi_araddr  <= S_AXI_ARADDR;
else
  axi_arready <= '0';
end if;
```

The above code snippet is an edited extraction from the AXI-Lite slave interface to generate the ARREADY signal. It is generated by Xilinx Vivado tool. The line commented as 'SEQ' has been manually inserted and it ensures that the read access is

accepted only when there are no more writes at the AXI-slave interface.

The DMA Engine

The DMA Engine used in this IP Core converts AXI4 Streaming data to AXI4 Memory-mapped protocol and vice versa, based on the commands issued to it. The data for a Host-2-Peripheral transfer is provided by the CMD/Write Data FIFO. The UDK3 State-machine provides the FIFO data to the DMA engine over an AXI4 streaming protocol. The DMA engine writes this data to the peripherals via an AXI-Interconnect based on AXI4-Full protocol. After it has transferred the write data completely onto the AXI-Interconnect, the DMA Engine signals the end of transfers (xfer_cmplt) signal to the AXI-FX3-Interface and waits for the next command. For a Peripheral-2-Host transfer, the DMA Engine reads the peripherals over AXI4-Full protocol and provides this data to the UDK3 State-machine over an AXI4 streaming interface. The DMA generates a 'tlast' (AXI4-Streaming interface) signal when it is transferring the last 32-bit data which signals the end of transfer during a read access.

Example Design

This chapter contains information about the example design provided with the BeastLink v1.0 IP-Core. It serves as a starting point for user designs. The design is structured in such a way that it reduces development time in the user's custom designs. One can easily expand the example design by adding custom blocks/IPs to the AXI bus system or by modifying the existing design. The example design is delivered as a set of VHDL sources with the BeastLink IP Core. The example design can be simulated, synthesized, implemented and a bitstream can be generated for validation on hardware. This design was developed using Vivado Design Suite 2017.4 and and tested on the Cesys EFM03 Beastboard.

The example design consists of the following components and primitives:

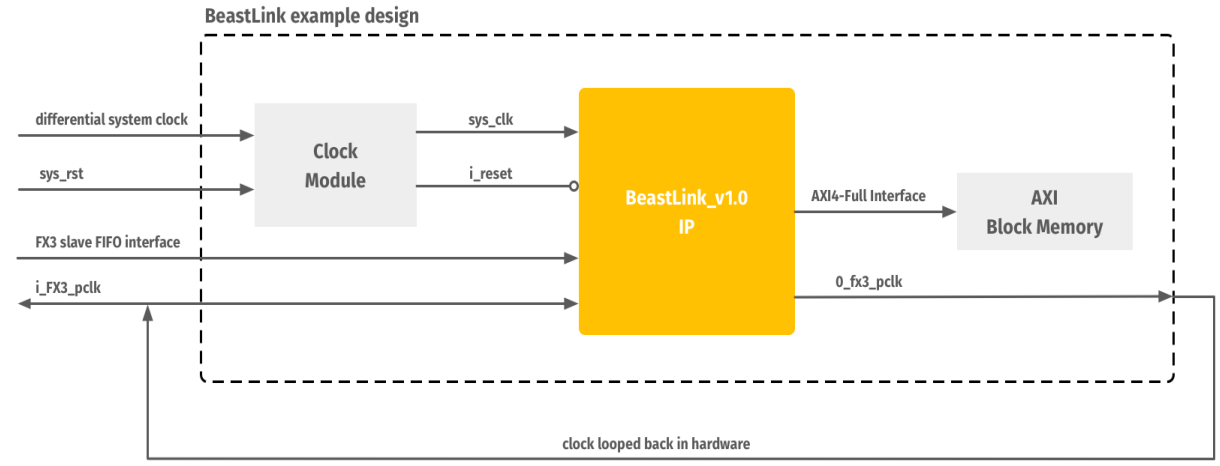
- BeastLink top module
- BeastLink v1.0 IP Core
- Block Memory Generator with AXI interface
- An MMCM primitive

Using the example design

The top module instantiates all components and IP-cores that are essential for implementing the example design on . The example design is named as **BeastLink_0_ex.xpr** and can be opened by following the below steps:

1. Add the BeastLink_IP to the project as described in the section **Quick Start**
2. Make sure the project's target language is VHDL
3. Under **IP Integrator**, click on *Generate Block Design* or find the IP in *sources* window and *right-click -> Generate Output Products...*
4. In *Generate Output Products...* popup, **Global** under **Synthesis Options** and press **Generate**
5. Right-click on BeastLink_IP and select **Open IP example design...**
6. A new Vivado instance will open with the BeastLink example design

The example design receives commands from the host, and based on the received command, performs a READ or a WRITE operation on the AXI Block-RAM peripheral. There are 2 different file sets in this example design, one for simulation and one for synthesis. Each of them is explained in detail in the upcoming sections. A brief explanation is provided below about the components used. The figure below gives an overview of the example design and the components it consists.



Block Diagram of the example design.

Clock Module

The example design instantiates a Mixed Mode Clocking Manager (MMCM) primitive with a Differential-to-Single-ended Buffer (BUFDS) primitive. The BUFDS module converts the differential 200MHz clock (FPGA balls R3/P3) from the board to a Single-ended 200MHz clock which is provided to the input clock of the MMCM primitive. The MMCM primitive generates a 100MHz single ended clock to drive all the components in the example design. The clock module also takes in the system reset (FPGA ball K15) signal from the board. The active-LOW reset for all the components in the design is driven by the MMCM's **clock locked** signal. The MMCM is configured as shown below.

```
-- Jitter programming (OPTIMIZED, HIGH, LOW)
BANDWIDTH          => "OPTIMIZED",

CLKFBOUT_MULT_F    => 5.0, -- Multiply value for all CLKOUT (2.000-64.000).
CLKFBOUT_PHASE     => 0.0, -- Phase offset in degrees of CLKFB (-360.000-360.000).
CLKIN1_PERIOD      => 5.0, -- Input clock period in ns to ps resolution (i.e. 5.0 is
200 MHz).
-- CLKOUT0_DIVIDE - CLKOUT6_DIVIDE:
-- Divide amount for each CLKOUT (1-128)
CLKOUT1_DIVIDE     => 1,
CLKOUT2_DIVIDE     => 1,
CLKOUT3_DIVIDE     => 1,
CLKOUT4_DIVIDE     => 1,
CLKOUT5_DIVIDE     => 1,
CLKOUT6_DIVIDE     => 1,
CLKOUT0_DIVIDE_F   => 10.000, -- Divide amount for CLKOUT0 (1.000-128.000).
```

```
-- CLKOUT0_DUTY_CYCLE - CLKOUT6_DUTY_CYCLE:
-- Duty cycle for each CLKOUT (0.01-0.99).
CLKOUT0_DUTY_CYCLE => 0.5,
CLKOUT1_DUTY_CYCLE => 0.5,
CLKOUT2_DUTY_CYCLE => 0.5,
CLKOUT3_DUTY_CYCLE => 0.5,
CLKOUT4_DUTY_CYCLE => 0.5,
CLKOUT5_DUTY_CYCLE => 0.5,
CLKOUT6_DUTY_CYCLE => 0.5,

-- CLKOUT0_PHASE - CLKOUT6_PHASE:
-- Phase offset for each CLKOUT (-360.000-360.000).
CLKOUT0_PHASE    => 0.0,
CLKOUT1_PHASE    => 0.0,
CLKOUT2_PHASE    => 0.0,
CLKOUT3_PHASE    => 0.0,
CLKOUT4_PHASE    => 0.0,
CLKOUT5_PHASE    => 0.0,
CLKOUT6_PHASE    => 0.0,
CLKOUT4_CASCADE => FALSE,  -- Cascade CLKOUT4 counter with CLKOUT6 (FALSE, TRUE)

DIVCLK_DIVIDE => 1,    -- master division value (1-106)
REF_JITTER1 => 0.0,    -- Reference input jitter in UI (0.000-0.999).
STARTUP_WAIT => FALSE -- Delays DONE until MMCM is locked (FALSE, TRUE)
```

BRAM Controller and Block Memory Generator

The example design instantiates Block RAM using the *Block Memory Generator* IP Core available in Vivado. This IP uses Block RAM primitives present on the FPGA fabric. In this example design, a 16Kbyte (4096 * 32 bit data width) Block RAM module is instantiated. The Block Memory Generator in this example design is instantiated with the following settings:

Window	Parameter	Settings
BASIC	Interface Type	AXI4
	Memory Type	Simple Dual Port
	Algorithm	Minimum Area
AXI4	AXI Type	AXI4
	AXI Slave Type	Memory slave
	ID Width	4
Port A Opt	Width	32
	Depth	4096
Port B Opt	Identical to Port A	
Other Options	Enable Safety Circuit	Deselected

Window	Parameter	Settings
	Collision Warnings	All

The AXI based Block Memory Generator has a 16K address range starting at 0x00000000 up until 0x00003FFF. For further details on the Block Memory Generator IP-core, please refer to *PG058* programming guides from Xilinx.

Note!

Only the options shown in the above table are modified. All other options remain default. The Interface Type option can be set to AXI4 ***only when the core is instantiated from the IP Catalog****.*

Simulating the design

The BeastLink example design consists of two file sets, one for synthesis/ implementation and another for simulation. This section describes the design in the simulation workflow.

The simulation workflow contains the same components as the synthesis workflow, the only difference being that the simulation workflow contains Verilog files for Block Memory Generator, whereas the synthesis workflow contains VHDL files. The design is buit so because Xilinx supports only Verilog for simulation. Apart from this difference, the design is same in both workflows and they are functionally identical. The simulation file set also contains a VHDL test bench *TB_beastlink_exdes_top.vhd* which instantiates the top module as DUT. This test bench provides the 200MHz differential clock and the active-HIGH system reset. It receives the "o_fx3_pclk" from the DUT and loops it back the DUT as "i_fx3_pclk". The test bench also provides Commands and Data to the DUT. It writes to the entire 16Kbyte BRAM address space and then reads the BRAM contents back. No verification is done during read back.

The test-bench is designed in such a way as to provide commands and data to the design. It follows the BeastLink protocol structure (See BeastLink Protocol Structure section). The test-bench provides all the clocks and resets required to simulate the design.

At time 0s, the test-bench asserts and holds the reset signal to '1' (active-HIGH in top module). It waits for 50 clock cycles (250ns) before deasserting it. The test-bench then sets the FX3 Flags for a write operation ("0010") and writes the header and dummy bytes followed by 16384 data bytes to the BRAM starting from address 0x00000000. After writing the data, the FX3 Flags are set to "0000" which signifies no

opertion. After 25us, the FX3 Flags are set to "0010" in order to transfer a command to read the previously written 16384 bytes from BRAM starting from address 0x00000000. After transferring the read command, the FX3 Flags are set to "1000" which signifies that the FPGA can transfer the 16384 bytes read from BRAM over the FX3 data interface. The BeastLink v1.0 IP-Core inserts the required amount of dummy bytes while transferring the data over the FX3 data interface.

The simulation design is completely built and the user can analyze the waveforms by just running the simulation. To simulate the design, please click on **Run Simulation** under **SIMULATION** section in the **Flow Navigator** window of Vivado. Please execute simulation for at least 250 micro seconds in order to analyze the write and read operations.

Synthesis and Implementation

For synthesizing the design, click on **Run Synthesis** in **Flow Navigator** under **Synthesis**.

For implementing (Place And Route) the design, click on **Run Implementation** in **Flow Navigator** under **Implementation**.

In order to use the FPGA Design in Cesys BeastLink Library, the generated bitstream must be in "**<filename>.bin**". A binary file can be genrated in the Vivado tool chain during bitstream generation. This can be accomplished by selecting **-bin_file** option under **Flow Navigator -> PROJECT MANAGER -> Settings -> Bitstream**. If Vivado TCL console is being used, then the option "**-bin_file**" must be added to the "**write_bitstream**" command.

To generate a bitstream, click on **Generate Bitstream** in **Flow Navigator** under **Program and Debug**.

Verifying the design

After generating the bitstream in Vivado toolset follow the below steps to verify the design:

1. Copy the generated **beastlink_exdes_top.bin** and paste it in the **<installation path>/udk3-tools-windows-1.5.1/** folder.
2. Run the **UDK3perfMon.exe** tool in the same folder
3. Select *EFM03(XCA200T)SOC Block RAM* or *EFM03(XCA200T)SOC DDR3L RAM* in **Preset** section.
4. In **Device**, select EFM03

5. Browse and select the **beastlink_exdes_top**.bin in **FPGA design file** field (it is also selected by default)
6. Select the Transfer Block Size either by clicking on the options or by entering the size manually
7. Make sure the Transfer Block Size is a power of 2 or Area Size is a multiple of Transfer Block Size
8. Select Input to read, Output to write or all three to write data and then read and verify the written data.
9. Press **Start**; Data transfer should begin and run continuously without any errors.

Constraints

The example design is delivered with a Xilinx Design Constraints (XDC) file and it can be found under:

<installation_path>/BeastLink_0_ex/imports/beastlink_constr.xdc.

The constraints in this file were written in accordance with the Cesys EFM03 Beastboard. This file contains all the IO and timing constraints required in order to acheive the best performance with this example design. The most important constraints are described below.

The constraints below describe the system clock, reset, output clock for FX3 and the looped byck input clocks. It establishes their locations, IO Standards and the timing information of the clocks. These constraints set up the **system clock** at R3/P3 pins as a 200MHz differential clock and the **system reset** at FPGA ball location K15. Further, these constraints state that the output clock to FX3 (o_fx3_pclk; N21) is generated from the sys_clk of the BeastLink IP Core and sets up the looped back clock (i_fx3_pclk; M21) to be a 100MHz clock with 50% duty cycle.

```
# System Clock - 200MHz Differential Clock @ R3/P3
create_clock -period 5.000 -name sys_clk -waveform {0.000 2.500} [get_ports
sys_clk_p]
set_property PACKAGE_PIN R3 [get_ports sys_clk_p]
set_property IOSTANDARD DIFF_SSTL135 [get_ports sys_clk_p]

# System Reset - ACTIVE HIGH input @ K15
set_property PACKAGE_PIN K15 [get_ports sys_rst]
set_property IOSTANDARD LVCMOS33 [get_ports sys_rst]

# Output clock to FX3 Chipset @ M21.
# This clock is looped back in hardware
set_property PACKAGE_PIN N21 [get_ports o_fx3_pclk]
set_property IOSTANDARD LVCMOS33 [get_ports o_fx3_pclk]

# Input Clock looped back in Hardware @ N21
```

```
# This is important for the functionality of this IP
set_property PACKAGE_PIN M21 [get_ports i_fx3_pclk]
set_property IOSTANDARD LVCMOS33 [get_ports i_fx3_pclk]
create_clock -period 10.000 -name i_fx3_pclk -waveform {0.000 5.000} [get_ports
i_fx3_pclk]

# Generated clock constraint for the output clock "o_fx3_pclk"
# This is nothing but the "sys_clk" given to the BeastLink IP
create_generated_clock -name o_pclk -source [get_pins -filter REF_PIN_NAME=~sys_clk*
-of [get_cells -hier inst_beastlink*]] -multiply_by 1 [get_ports o_fx3_pclk]
```

Caution!

Users may modify the pin locations as per their own design requirements. However, it is recommended ***not to modify the timing constraints****.*

These constraints set up the FPGA configuration mode. They state that the FPGA will be configured by an external master device, in this case the FX3 chipset. These constraints are very important and have to be present if the FPGA needs to be configured by host PC through using Cesys BeastLink Library. Also, the FPGA is instructed to use the 90MHz External master Configuration Clock (EMCCLK) during configuration and not the default 3MHz configuration clock (CCLK).

```
# Configuration Mode Slave Select Map 16 bit data bus
set_property CONFIG_MODE S_SELECTMAP16 [current_design]

# Use 90MHz External Master Configuration Clock during FPGA Configuration
set_property BITSTREAM.CONFIG.EXTMASTERCCLK_EN DIV-1 [current_design]
```

The FX3 Chipset has a tight setup/hold timing requirement on its input data bus and has a large delay before it outputs data to FPGA. The FPGA design must comply to these timing requirements. The below *set_input_delay* constraints state that FX3 data is valid at the FPGA input port only after 8ns and is valid for 2 ns. The *set_output_delay* constraints establish the setup/hold requirement of the FX3's control input and data input ports.

```
# FX3 Data Output Delay Constraints
set_input_delay -clock i_fx3_pclk -max 8 [get_ports io_fx3_fdata]
set_input_delay -clock i_fx3_pclk -min 2 [get_ports io_fx3_fdata]

# FX3 Control In and Data In setup/hold requirements
# SETUP is 2 ns, so max is 2
# HOLD is 0.5 ns, so min is -0.5
set_output_delay -clock o_pclk -max 2.000 [get_ports io_fx3_fdata]
set_output_delay -clock o_pclk -min -0.500 [get_ports io_fx3_fdata]

set_output_delay -clock o_pclk -max 2.000 [get_ports o_fx3_slcs_n]
set_output_delay -clock o_pclk -min -0.500 [get_ports o_fx3_slcs_n]
```

```
set_output_delay -clock o_pclk -max 2.000 [get_ports o_fx3_slwr_n]
set_output_delay -clock o_pclk -min -0.500 [get_ports o_fx3_slwr_n]

set_output_delay -clock o_pclk -max 2.000 [get_ports o_fx3_slrd_n]
set_output_delay -clock o_pclk -min -0.500 [get_ports o_fx3_slrd_n]

set_output_delay -clock o_pclk -max 2.000 [get_ports o_fx3_sloe_n]
set_output_delay -clock o_pclk -min -0.500 [get_ports o_fx3_sloe_n]

set_output_delay -clock o_pclk -max 2.000 [get_ports o_fx3_pktend_n]
set_output_delay -clock o_pclk -min -0.500 [get_ports o_fx3_pktend_n]

set_output_delay -clock o_pclk -max 2.000 [get_ports o_fx3_faddr]
set_output_delay -clock o_pclk -min -0.500 [get_ports o_fx3_faddr]
```

Warning!

These timing constraints are crucial for the functionality of the example design (eventually any design using BeastLink IP Core). Ignoring/modifying these constraints may result in undefined behaviour.

The below constraints deal with the location and IO standards of the control and data pins of the FPGA's slave FIFO Interface. The FPGA output data ports are constrained for a drive strength of 12mA and for a fast slew-rate so that the data is available at the FX3's inputs as soon as possible and with very less transition time. All address/ control ports are configured for a higher drive strength of 16mA.

```
# FX3 Control pin Locations
set_property PACKAGE_PIN L19 [get_ports o_fx3_pktend_n]
set_property PACKAGE_PIN M22 [get_ports o_fx3_slcs_n]
set_property PACKAGE_PIN K17 [get_ports o_fx3_sloe_n]
set_property PACKAGE_PIN K16 [get_ports o_fx3_slrd_n]
set_property PACKAGE_PIN M19 [get_ports o_fx3_slwr_n]
set_property PACKAGE_PIN N22 [get_ports {o_fx3_faddr[1]}]
set_property PACKAGE_PIN P20 [get_ports {o_fx3_faddr[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports o_fx3_pktend_n]
set_property IOSTANDARD LVCMOS33 [get_ports o_fx3_slcs_n]
set_property IOSTANDARD LVCMOS33 [get_ports o_fx3_sloe_n]
set_property IOSTANDARD LVCMOS33 [get_ports o_fx3_slrd_n]
set_property IOSTANDARD LVCMOS33 [get_ports o_fx3_slwr_n]
set_property IOSTANDARD LVCMOS33 [get_ports {o_fx3_faddr[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_fx3_faddr[0]}]

set_property DRIVE 16 [get_ports o_fx3_slcs_n]
set_property DRIVE 16 [get_ports o_fx3_sloe_n]
set_property DRIVE 16 [get_ports o_fx3_slrd_n]
set_property DRIVE 16 [get_ports o_fx3_slwr_n]
set_property DRIVE 16 [get_ports o_fx3_pktend_n]
set_property DRIVE 16 [get_ports {o_fx3_faddr[1]}]
```

```
set_property DRIVE 16 [get_ports {o_fx3_faddr[0]}]

set_property PACKAGE_PIN T24 [get_ports {io_fx3_fdata[0]}]
set_property PACKAGE_PIN L23 [get_ports {io_fx3_fdata[1]}]
set_property PACKAGE_PIN L22 [get_ports {io_fx3_fdata[2]}]
set_property PACKAGE_PIN L24 [get_ports {io_fx3_fdata[3]}]
set_property PACKAGE_PIN N16 [get_ports {io_fx3_fdata[4]}]
set_property PACKAGE_PIN N17 [get_ports {io_fx3_fdata[5]}]
set_property PACKAGE_PIN R16 [get_ports {io_fx3_fdata[6]}]
set_property PACKAGE_PIN R17 [get_ports {io_fx3_fdata[7]}]
set_property PACKAGE_PIN N18 [get_ports {io_fx3_fdata[8]}]
set_property PACKAGE_PIN K25 [get_ports {io_fx3_fdata[9]}]
set_property PACKAGE_PIN K26 [get_ports {io_fx3_fdata[10]}]
set_property PACKAGE_PIN M20 [get_ports {io_fx3_fdata[11]}]
set_property PACKAGE_PIN L20 [get_ports {io_fx3_fdata[12]}]
set_property PACKAGE_PIN L25 [get_ports {io_fx3_fdata[13]}]
set_property PACKAGE_PIN M24 [get_ports {io_fx3_fdata[14]}]
set_property PACKAGE_PIN M25 [get_ports {io_fx3_fdata[15]}]
set_property PACKAGE_PIN R23 [get_ports {io_fx3_fdata[16]}]
set_property PACKAGE_PIN T23 [get_ports {io_fx3_fdata[17]}]
set_property PACKAGE_PIN R22 [get_ports {io_fx3_fdata[18]}]
set_property PACKAGE_PIN T22 [get_ports {io_fx3_fdata[19]}]
set_property PACKAGE_PIN P26 [get_ports {io_fx3_fdata[20]}]
set_property PACKAGE_PIN R26 [get_ports {io_fx3_fdata[21]}]
set_property PACKAGE_PIN T25 [get_ports {io_fx3_fdata[22]}]
set_property PACKAGE_PIN M26 [get_ports {io_fx3_fdata[23]}]
set_property PACKAGE_PIN N26 [get_ports {io_fx3_fdata[24]}]
set_property PACKAGE_PIN P25 [get_ports {io_fx3_fdata[25]}]
set_property PACKAGE_PIN R25 [get_ports {io_fx3_fdata[26]}]
set_property PACKAGE_PIN R21 [get_ports {io_fx3_fdata[27]}]
set_property PACKAGE_PIN R20 [get_ports {io_fx3_fdata[28]}]
set_property PACKAGE_PIN P24 [get_ports {io_fx3_fdata[29]}]
set_property PACKAGE_PIN P23 [get_ports {io_fx3_fdata[30]}]
set_property PACKAGE_PIN N19 [get_ports {io_fx3_fdata[31]}]

# Slew Rate and drive strength and IO Stds for FX3 data
set_property DRIVE 12 [get_ports {io_fx3_fdata[0]}]
set_property SLEW FAST [get_ports {io_fx3_fdata[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {io_fx3_fdata[0]}]
# ...
set_property DRIVE 12 [get_ports {io_fx3_fdata[31]}]
set_property SLEW FAST [get_ports {io_fx3_fdata[31]}]
set_property IOSTANDARD LVCMOS33 [get_ports {io_fx3_fdata[31]}]
```

Caution!

Users may modify the pin locations as per their own design requirements. However, it is recommended not to modify the Drive Strength and Slew rates of those pins.

Performace benchmarking

The design and its performance was thoroughly tested using the UDK3 Performance Monitor tool available with the UDK3 toolchain under **<UDK3_installation_path>/udk3-tools-windows-1.5.x** folder. The Performace Monitor was run on an Intel Core i7-4790 processor running a 64-bit Windows-7 Ultimate operating system. The design's performance was recorded and is presented in the table below.

Peripheral	**Transfer Size** (Bytes)	**Area Size**	**Write Speed** (MBps)	**Read Speed** (MBps)	**Read & verify** (MBps)
Block RAM	2048	0x0001_0000	32.10	15.60	21.00
Block RAM	8192	0x0001_0000	127.85	62.60	84.20
Block RAM	16384 (16K)	0x0001_0000	152.30	75.50	130.25

The performance of the BeastLink IP Core was observed to be dependent on the Transfer Size. Since the BRAM in the example design was limited to 16Kbyte, another benchmarking was performed with the Cesys EFM03 Beastboard where the on-board DDR3L RAM which facilitates a larger address space was targeted. The results were recorded and is presented in the table below.

Peripheral	**Transfer Size** (Bytes)	**Area Size**	**Write Speed** (MBps)	**Read Speed** (MBps)	**Read & verify** (MBps)
DDR3L RAM	8192	0x8000_0000	124.10	62.25	82.65
DDR3L RAM	65536 (64K)	0x8000_0000	266.75	222.50	242.50
DDR3L RAM	262144 (256K)	0x8000_0000	290.25	289.50	282.25
DDR3L RAM	1048576 (1M)	0x8000_0000	304.30	320.50	302.50

Copyright Notice

Disclaimer

This file contains confidential and proprietary information of Cesys GmbH and is protected under international copyright and other intellectual property laws.

Disclaimer

This disclaimer is not a license and does not grant any rights to the materials distributed herewith. Except as otherwise provided in a valid license issued to you by Cesys, and to the maximum extent permitted by applicable law:

(1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND WITH ALL FAULTS, AND CESYS HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE;

and

(2) Cesys shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under or in connection with these materials, including for any direct, or any indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Cesys had been advised of the possibility of the same.

CRITICAL APPLICATIONS

CESYS products are not designed or intended to be fail-safe, or for use in any application requiring fail-safe performance, such as life-support or safety devices or systems, Class III medical devices, nuclear facilities, applications related to the deployment of airbags, or any other applications that could lead to death, personal injury, or severe property or environmental damage (individually and collectively, "Critical Applications"). Customer assumes the sole risk and liability of any use of Cesys products in Critical Applications, subject only to applicable laws and regulations governing limitations on product liability.

THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS PART OF THIS FILE AT ALL TIMES.

Address

CESYS Gesellschaft für angewandte Mikroelektronik mbH
Gustav-Hertz-Str. 4
D - 91074 Herzogenaurach - GERMANY

Table of Contents

Introduction....2
Features....2
IP Facts....3
IP Information....3
Inclusions....3
Resource Utilization....3
Applications....4
Advantages....4
Quick Start....5
Obtaining the IP-Core files....5
Adding the BeastLink IP to the IP Catalog....5
Overview....7
Port Description....7
BeastLink Port Description....7
Clocking....10
Resets....10
Functional Description....11
Internal Structure....11
BeastLink Protocol Structure....12
GPIF-II State Machine....14
CMD/Write and Resp/Read FIFOs....14
UDK3 State machine....15
Elimination of Read-Write Collision....15
The DMA Engine....17
Example Design....18
Using the example design....18
Clock Module....19
BRAM Controller and Block Memory Generator....20
Simulating the design....21
Synthesis and Implementation....22
Verifying the design....22
Constraints....23
Performace benchmarking....27
Copyright Notice....28
Disclaimer....28
Disclaimer....29
CRITICAL APPLICATIONS....29
Address....29
Revision history....31

Revision history

Version	Date	Details	Author	Approved by
1.0	Feb, 2018	Initial Release	vv	??