Zynq-7000 All Programmable SoC Verification IP v1.0

DS940 (v1.0) April 28, 2017 **Product Specification**

Summary

The Zynq®-7000 All Programmable SoC Verification Intellectual Property (VIP) supports the functional simulation of Zynq-7000 based applications. It is targeted to enable the functional verification of Programmable Logic (PL) by mimicking the Processor System (PS)-PL interfaces and OCM/DDR memories of PS logic. This VIP is delivered as a package of System Verilog modules. VIP operation is controlled by using a sequence of Verilog tasks contained in a Verilog-syntax file.

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family(1)	Zynq®-7000 All Programmable SoC, 7 Series
Supported User Interfaces	AXI4, AXI4-Lite, AXI3
Resources	Not Provided
Provided with Core	
Design Files	Not Provided
Example Design	Verilog
Test Bench	N/A
Constraints File	N/A
Simulation Model	Verilog
Supported S/W Driver(2)	N/A
Tested Design Flows(2)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Features

- Pin compatible and Verilog-based simulation model.
- Supports all AXI interfaces.
  - AXI 3.0 compliant.
- 32/64–bit Data-width for AXI_HP, 32-bit for AXI_GP and 64-bit for AXI_ACP.
- Sparse memory model (for DDR) and a RAM model (for OCM).
- System Verilog task-based API.
- Delivered in Vivado® Design Suite.
- Blocking and non-blocking interrupt support.
- ID width support as per the Zynq-7000 specification.
- Support for FIXED, INCR and WRAP transaction types.
- Support for all Zynq-7000 supported burst lengths and burst sizes.
- Protocol checking, provided by the AXI VIP models.
- Read/Write request capabilities.
- System Address Decode for OCM/DDR transactions.

Additional Features

- System Address Decode for Register Map Read transactions (only default value of the registers can be read).
- Support for static remap for AXI_GP0 and AXI_GP1.
- Configurable latency for Read/Write responses.
- First-level arbitration scheme based on the priority indicated by the AXI QoS signals.
- Datapath connectivity between any AXI master in PL and the PS memories and register map.
- Parameters to enable and configure AXI Master and Slave ports.
- APIs to set the traffic profile and latencies for different AXI Master and Slave ports.
- Support for FPGA logic clock generation.
- Soft Reset Control for the PL.
- API support to pre-load the memories, read/wait for the interrupts from PL, and checks for certain data pattern to be updated at certain memory location.
- All unused interface signals that output to the PL are tied to a valid value.
- Semantic checks on all other unused interface signals.

An example design that demonstrates the usage of this VIP is available for reference.

Limitations

The following features are not yet supported by Zynq7000 APSoC VIP:

- Exclusive Access transfers are not supported on any of the slave ports.
- Read/Write data interleaving is not supported.
- Write access to the Register Map is not supported.
- Support for in-order transactions only.

Applications

The Zynq-7000 VIP is used to provide a simulation environment for the Zynq-7000 PS logic, typically replacing the processing_system7 block in a design. The Zynq-7000 VIP models the following:

- Transactions originating from PS masters through the AXI VIP master API calls
- Transactions terminating through the PS slaves to models of the OCM and DDR memories through interconnect models FCLK reset and clocking support
- Input interrupts to the PS from PL
- PS register map

Functional Description

Zynq-7000 VIP consists of four main layers. Figure 1 shows the Zynq-7000 VIP architecture.

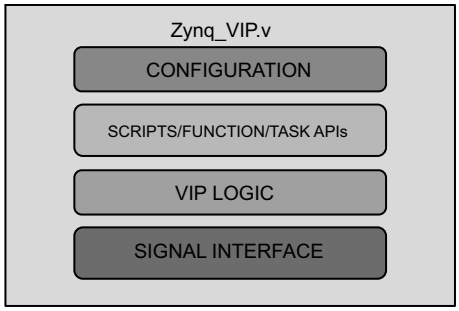

Figure 1: **Zynq 7000 VIP Architecture**

- **Configuration**

  Configuration is implemented using Verilog parameters and is used to configure the Zynq-7000 VIP. Some configuration must be done using configuration APIs.
- **Function and Task APIs**

  Verilog tasks and functions that help to set:
  - Datapath between processing system (PS) and programmable logic (PL) in memory mapped interfaces.
  - Control path between PS and PL in register interface.
  - Configure the traffic profiles for each ports.
- **VIP Logic**

  VIP logic has the PS-PL interface with supporting functionality that contains the AXI interfaces, sparse memory implementation, and the interconnect (arbiter) model as shown in Figure 2.
- **Signal Interface**

  The signal interface includes the typical Verilog input and output ports and associated signals.

Figure 2 shows the detailed architecture for the VIP logic.

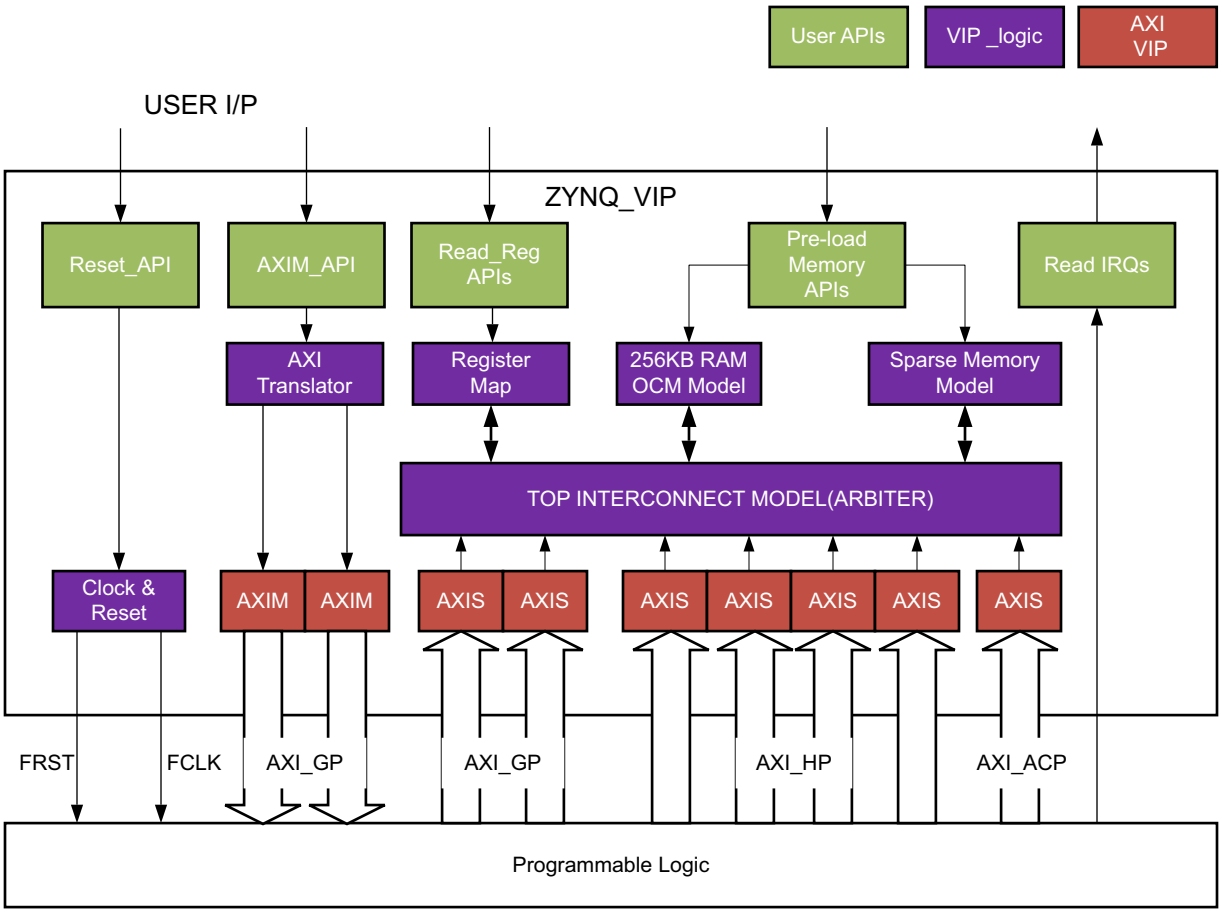

Figure 2: **Architecture Details**

Figure 3 show the Zynq-7000 VIP test bench.

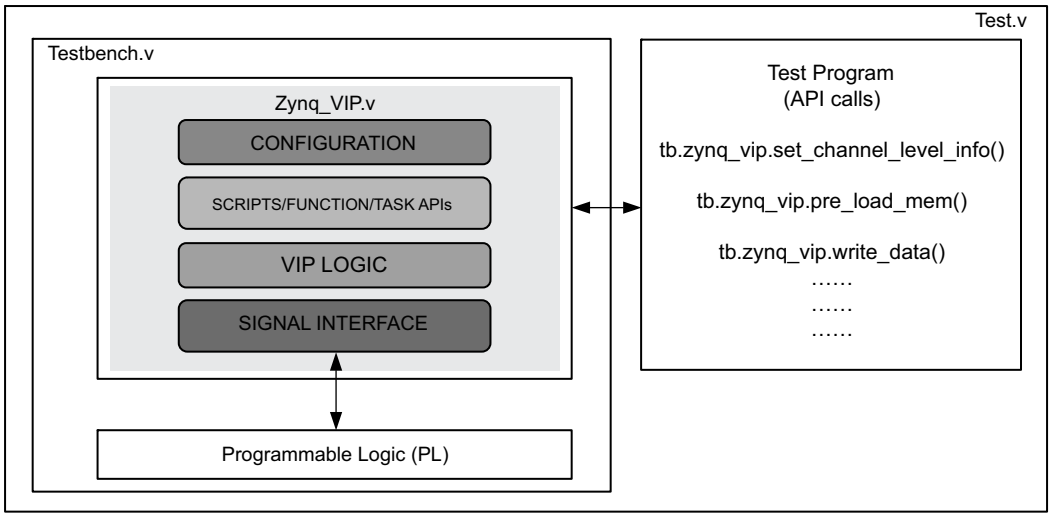

Figure 3: **Zynq-7000 VIP test bench**

Using Zynq-7000 Verification IP

This section details the configuration parameters and the APIs necessary for using the Zynq-7000 VIP. It also explains how to run the sample Verilog tests.

The interface models in the Zynq-7000 VIP are based on the AXI VIP LogiCORE™ IP module, which is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License.

Application Programming Interfaces

The application programming interfaces (APIs) specified in Table 1 can be used to configure the VIP and develop a test program. The order of inputs and outputs for a given API must be configured with inputs followed by outputs in the same order. The following APIs can be used only after releasing RESET to the VIP.

Table 1: **Zynq-7000 VIP APIs**

APIs	Inputs	Outputs
set_stop_on_error When set to value '1', Stop the simulation on error. The default value is 1.	**LEVEL**: A bit input for the info level.	None
set_debug_level_info When set to value '1', debug level info for Zynq-7000 VIP is reported, else no info is reported. The default value is 1.	**LEVEL**: A bit input for the info level.	None
set_arqos Set the ARQOS value to be used by Slave ports for first level arbitration scheme. The AXI Slave ARQOS input port value is enabled by default.	**Name**: S_ AXI_GP0, S_AXI_GP1, S_AXI_HP0, S_AXI_HP1, S_AXI_HP2, S_AXI_HP3, S_AXI_ACP **[3:0] val**: ARQoS value.	None
set_awqos Set the AWQOS value to be used by Slave ports for first level arbitration scheme. The AXI Slave AWQOS input port value is enabled by default.	**Name**: S_ AXI_GP0, S_AXI_GP1, S_AXI_HP0, S_AXI_HP1, S_AXI_HP2, S_AXI_HP3, S_AXI_ACP **[3:0] val**: AWQoS value.	None
fpga_soft_reset Issue/Generate soft reset for PL.	**[31:0] reset_ctrl** : 32-bit input indicating the reset o/p to be asserted for PL. (Details same as FPGA_RST_CTRL register defined in PS)	None

Table 1: **Zynq-7000 VIP APIs *(Cont'd)***

APIs	Inputs	Outputs
pre_load_mem_from_file Preload DDR/OCM with data from a file. Based on the address specified, the data is loaded in DDR/OCM. Only hexadecimal data format is supported. **DDR**: Address must be 32-bit aligned. **OCM**: Address must be 32-bit aligned.	**[1023:0] file_name**: File name (max. 128 characters) **[31:0] start_addr**: Start Address from where DDR/OCM should be initialized with data from the file. **no_of_bytes**: Number of data bytes to be loaded	None
pre_load_mem Preload DDR/OCM with random_data/all_zeros/all_ones. Based on the address specified, the data is loaded in DDR/OCM. DDR: Address must be 32-bit aligned. OCM: Address must be 32-bit aligned.	**[1:0] data_type**: Random, zeros or ones. 00 - Random 01- Zeros, 10 - Ones. **[31:0] start_addr**: Start Address from where DDR should be initialized with data from the file. **no_of_bytes**: Number of data bytes to be loaded	None
read_interrupt Users can use this API to poll the interrupt status at any given time. This is a non-blocking task which returns immediately with the current status of the interrupt pins.	None	**[15:0] irq_status**: Interrupts generated by PL.
wait_interrupt Users can use this API to wait for any of the interrupt to occur. This is a blocking task and returns only any of the interrupt pin is asserted.	**[3:0] irq**: Interrupt line number	**[15:0] irq_status**: Interrupts generated by PL.
wait_mem_update Users can use this API to poll for a specific location. This task is a blocking task and returns when some value is written in that location (either a specific value or a new value). If the value does not match the expected data pattern then a fatal error is issued and the simulation halts. Only one instance of this API should be used at a time.	**[31:0] addr**: 32-bit address (DDR/OCM) [**31:0] data_i**: expected data pattern	**[31:0] data_o**: data value that is updated in the memory
write_from_file Initiate a AXI write transaction on the GP master port. The write data is used from the file. Burst type used is INCR. This is a blocking task and returns only after the complete AXI WRITE transaction is done. Address has to 32-bit aligned.	**[1023:0] file_name**: File name that stores the write data (max. 128 characters). **[31:0] addr**: Write address **wr_size**: Number of data bytes to written	**RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
read_to_file Initiate a AXI read transaction on the GP master port. The read data is written to the file. Burst type used is INCR. This is a blocking task and returns only after the complete AXI READ transaction is done. Address has to 32-bit aligned.	**[1023:0] file_name**: File name that stores the read data (max. 128 characters). **[31:0] addr**: Read address **rd_size**: Number of data bytes to be read	**RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]

Table 1: **Zynq-7000 VIP APIs *(Cont'd)***

APIs	Inputs	Outputs
write_data Initiate a AXI write transaction on the GP master port. This task should be used when the transfer size is less or equal to 128 bytes and the write data is provided as an argument to the task call. Burst type used is INCR. This is a blocking task and returns only after the AXI write is completed through a write response (BRESP).	**[31:0] addr**: Write address **[7:0] wr_size**: Number of data bytes to be written **[1023:0] wr_data**: write data (max. 128 bytes).	**RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
read_data Initiate a AXI read transaction on the master port. This task should be used when the transfer size is less or equal to 128 bytes and the read data is returned as an output to the task call. Burst type used is INCR. This is a blocking task and returns only after the complete AXI READ transaction is done.	**[31:0] addr**: Write address **[7:0] rd_size**: Number of data bytes to be read	**DATA**: Valid data transferred by the Slave **RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
write_burst Initiate a single AXI write burst transaction on the master port. This calls the AXI VIP API. This task returns when the complete write transaction is complete.	**ADDR**: Write Address **LEN**: Burst Length **SIZE**: Burst Size **BURST**: Burst Type **LOCK**: Lock Type **CACHE**: Cache Type **PROT**: Protection Type **DATA**: Data to send **DATASIZE**: The size in bytes of the valid data contained in the input data vector.	**RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
write_burst_concurrent Initiate a single AXI write burst transaction on the master port. This calls the AXI VIP API. This task performs write phase and address phases concurrently and returns when the complete write transaction is complete.	**ADDR**: Write Address **LEN**: Burst Length **SIZE**: Burst Size **BURST**: Burst Type **LOCK**: Lock Type **CACHE**: Cache Type **PROT**: Protection Type **DATA**: Data to send **DATASIZE**: The size in bytes of the valid data contained in the input data vector.	**RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
read_burst Initiate a single AXI Read burst transaction on the master port. This is hook to call the AXI VIP API. This task returns when the complete read transaction is complete.	**ADDR**: Write Address **LEN**: Burst Length **SIZE**: Burst Size **BURST**: Burst Type **LOCK**: Lock Type **CACHE**: Cache Type **PROT**: Protection Type	**DATA**: Valid data transferred by the slave **RESPONSE**: This is a vector that is created by concatenating all slave read responses together
read_register Read Register.	**[31:0] addr**: Register address	**DATA**: Valid register read data

Table 1: **Zynq-7000 VIP APIs *(Cont'd)***

APIs	Inputs	Outputs
read_register_map Read a chunk of registers. 32 registers can be read at a time using this API.	**[31:0] addr**: Starting Register address **size**: Number of registers to be read.	**DATA**: Valid register read data
set_slave_profile Set the Slave Profile for each slave port. The latency to be used on all slave ports is set using this API.	**Name**: S_AXI_GP0 , S_AXI_GP1, S_AXI_HP0, S_AXI_HP1, S_AXI_HP2, S_AXI_HP3, S_AXI_ACP or ALL **[1:0] latency**: latency type **00**: Best case **01**: Average case **10**: Worst case **11**: Random	None
wait_reg_update This API can be used to wait for a particular address in PL to be updated with a specific pattern. Zynq-7000 VIP issues an continuous AXI Read command with the specified address until the read data matches the expected data pattern. You can decide the interval of time between the reads. This is a blocking task and returns only when the read data matches the expected data pattern or timeout. Only One API instance per GP Master port should be used at a time.	**[31:0] addr**: Register address **[31:0] data_i**: expected data pattern. **[31:0] mask_i**: Mask indicating the bits that are masked (1- indicates bit is masked from read/write). **time_interval**: number of clock cycles to wait in between reads. **time_out**: number of clock cycles to wait for the register update.	**[31:0] data_o**: data value that is updated in the register.
peek_mem_to_file() Back door read to file from the DDR/OCM memory. Based on the address specified, the data is read from DDR/OCM. The read data is written to a file. DDR: Address must be 32-bit aligned. OCM: Address must be 32-bit aligned.	**[1023:0] file_name**: File name (max. 128 characters; Read data is written to this file). **[31:0] start_addr**: Start Address to read the data from. **no_of_bytes**: Number of data bytes to be read.	None

Table 1: **Zynq-7000 VIP APIs *(Cont'd)***

APIs	Inputs	Outputs
write_mem() Back door write to the DDR/OCM memory. Based on the address specified, the data is written to DDR/OCM. DDR: Address must be 32-bit aligned. OCM: Address must be 32-bit aligned.	**[1023:0] data**: Write data to be written to memory. **[31:0] start_addr**: Start Address to write the data from. **no_of_bytes**: Number of data bytes to be write (max. 128 bytes).	None
read_mem() Back door read from the DDR/OCM memory. Based on the address specified, the data is read from DDR/OCM. DDR: Address must be 32-bit aligned. OCM: Address must be 32-bit aligned.	**[31:0] start_addr**: Start Address to read the data from. **no_of_bytes**: Number of data bytes to be read (max. 128 bytes).	**[1023:0] data**: Read data from memory

Integrating Zynq-7000 VIP

The Zynq-7000 VIP will automatically be delivered as part of the Zynq Processing System block simulation source.

Configuration Options

Table 2 shows the configuration options that are passed to the VIP through Verilog parameters.

Table 2: **Configuration Options Using Verilog Parameters**

VIP Parameter	Default Value	Description
C_FCLK_CLK0_FREQ	50	PL Clock Frequency in MHz for FCLK0.
C_FCLK_CLK1_FREQ	50	PL Clock Frequency in MHz for FCLK1.
C_FCLK_CLK2_FREQ	50	PL Clock Frequency in MHz for FCLK2.
C_FCLK_CLK3_FREQ	50	PL Clock Frequency in MHz for FCLK3.
C_HIGH_OCM_EN	0	When set to '1', enables the high address range for OCM.
C_USE_S_AXI_HP0	0	When set to '1', enables the S_AXI_HP0 Slave port.
C_USE_S_AXI_HP1	0	When set to '1', enables the S_AXI_HP1 Slave port.
C_USE_S_AXI_HP2	0	When set to '1', enables the S_AXI_HP2 Slave port.
C_USE_S_AXI_HP3	0	When set to '1', enables the S_AXI_HP3 Slave port.
C_S_AXI_HP0_DATA_WIDTH	32	Set the data-width for S_AXI_HP0 port.
C_S_AXI_HP1_DATA_WIDTH	32	Set the data-width for S_AXI_HP1 port.
C_S_AXI_HP2_DATA_WIDTH	32	Set the data-width for S_AXI_HP2 port.
C_S_AXI_HP3_DATA_WIDTH	32	Set the data-width for S_AXI_HP3 port.

Table 2: **Configuration Options Using Verilog Parameters *(Cont'd)***

VIP Parameter	Default Value	Description
C_USE_M_AXI_GP0	0	When set to '1', enables the M_AXI_GP0 Master port.
C_USE_M_AXI_GP1	0	When set to '1', enables the M_AXI_GP1 Master port.
C_M_AXI_GP0_THREAD_ID_WIDTH	12	Possible values are 6 and 12. This gets set to '6' with Static remap enabled.
C_M_AXI_GP1_THREAD_ID_WIDTH	12	Possible values are 6 and 12. This gets set to '6' with Static remap enabled.
C_M_AXI_GP0_ENABLE_STATIC_REMAP	0	When set to '1', enables the M_AXI_GP0 Static remap.
C_M_AXI_GP1_ENABLE_STATIC_REMAP	0	When set to '1', enables the M_AXI_GP1 Static remap.
C_USE_S_AXI_GP0	0	When set to '1', enables the S_AXI_GP0 Slave port.
C_USE_S_AXI_GP1	0	When set to '1', enables the S_AXI_GP1 Slave port.
C_USE_S_AXI_ACP	0	When set to '1', enables the S_AXI_ACP Slave port.

Example Design

An example design and test bench is provided as part of the Vivado Zynq example design.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for the Zynq-7000 All Programmable SoC VIP Core:

AR67207

Licensing and Ordering Information

The Zynq-7000 Verification IP module is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Technical Support

Xilinx provides technical support at the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

1. *Zynq-7000 All Programmable SoC Technical Reference Manual*(UG585)
2. *Xilinx AXI Reference Guide (UG761)*
3. *LogiCORE IP AXI Interconnect Product Guide (PG059)*
4. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
5. *Vivado Design Suite User Guide: Designing with IP* (UG896)
6. *Vivado Design Suite User Guide: Getting Started* (UG910)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
9. *Vivado Design Suite User Guide: Implementation* (UG904)

Revision History

The following table shows the revision history for this document:

Date	Version	Description
04/28/2017	1.0	Initial Xilinx release.

Please Read: Important Legal Notices