AXI4-Stream Accelerator Adapter v2.1

LogiCORE IP Product Guide

Vivado Design Suite

PG081 November 18, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The AXI4-Stream Accelerator Adapter is a soft Xilinx® LogiCORE™ Intellectual Property (IP) core used as a infrastructure block for connecting hardware accelerators to embedded CPUs.

It provides the AXI4-Stream interface to AXI4 infrastructure components and BRAM/FIFO interface towards Accelerator IP. This IP is used to improve the overall system-level performance for hardware accelerator IP in the FPGA logic.

Features

- Connects as a 32-bit slave on AXI4-Lite interface
- AXI4-Stream data width support of 32, 64, 128, and 256 bits
- BRAM/FIFO data width support of 8, 16, 32, and 64 bits
- Supports up to eight channels for Stream-to-Memory (S2M), Memory-to-Stream (M2S), and Memory-to-Memory (M2M)
- Supports up to four buffers per channel
- Supports asymmetric Multi-Buffer data width for AXI4-Stream and BRAM/FIFO interface
- Supports up to eight input scalars, eight output scalars, and eight inout scalars for scalar interface
- Increased bandwidth between Accelerator and Adapter
- Command queue transaction pipelining

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families, UltraScale™ Architecture, Zynq®-7000 All Programmable SoC, 7 Series
Supported User Interfaces	AXI4-Lite, AXI4-Stream
Resources	See Table 2-2.
Provided with Core	
Design Files	RTL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	XDC
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows(2)	
Design Entry	Vivado® Design Suite Vivado
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.

Overview

The Accelerator Adapter core is a key interconnect infrastructure IP that provides the functionality to improve the overall system-level performance and efficiency when connecting hardware accelerators to embedded CPUs. The Accelerator Adapter IP core complements accelerators created using the Xilinx® Vivado High Level Synthesis (HLS) tool, abstracting AXI4-Stream transport signaling to enable a clean, algorithmic style of C programming in HLS, and improves overall system performance with task-level pipelining of the accelerators.

By providing a pipelined control interface and hardware multi-buffers, the Accelerator Adapter core enables software to queue multiple task requests synchronized in hardware without tight coupling to the Accelerator Control registers, which can significantly reduce task latency for pipelined tasks. Each accelerator operates as an independent thread, synchronized in hardware at the transport level by AXI4-Stream handshaking, with the input arrival and accelerator hardware "start/done" synchronization barriers realized by the Accelerator Adapter core.

Figure 1-1 shows the AXI4-Stream Accelerator Adapter block integration with rest of the system. The Accelerator Adapter core provides AXI4-Stream interface towards AXI4 infrastructure components (for example, AXI4 Interconnect, AXI4 DMA, etc.) and Xilinx BRAM/simple FIFO interface towards Vivado HLS generated hardware accelerators.

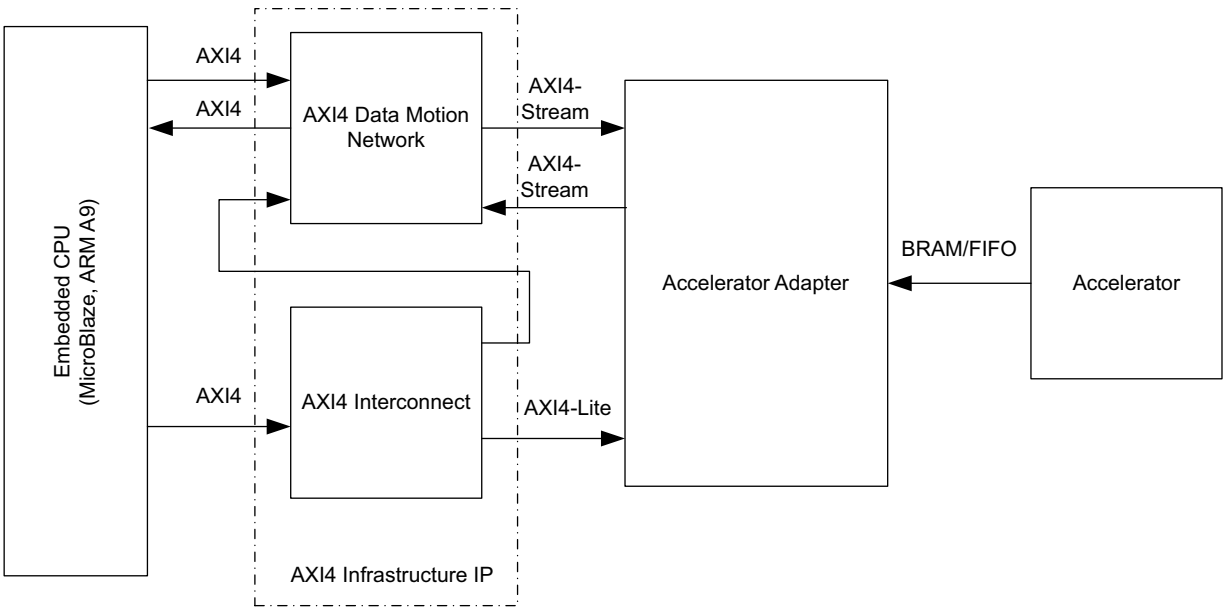

Figure 1-1: **Adapter System-Level Integration**

Figure 1-2 shows the Accelerator Adapter core block diagram and its integration with Data Motion Network and the HLS Accelerator. Accelerator Adapter core provides the following interfaces:

- **AXI4-Lite Slave Interfaces (All With FIFO Buffer)**
  - AXI4-Lite Task Control and Status registers
  - AXI4-Lite to Accelerator Input Scalar registers
  - Accelerator Output Scalar registers to AXI4-Lite
- **AXI4-Stream Slave Interfaces**
  - AXI4-Stream to BRAM (adapter is slave to accelerator)
  - AXI4-Stream to `acc_fifo_read` (adapter is master to accelerator)
- **AXI4-Stream Master Interfaces**
  - BRAM to AXI4-Stream (adapter is slave to accelerator)
  - `acc_fifo_write` to axis (adapter is slave to accelerator)
- **`acc_handshake` Master Interface for Accelerator Control**

Note: The interface names are based on the IP-XACT bus definitions.

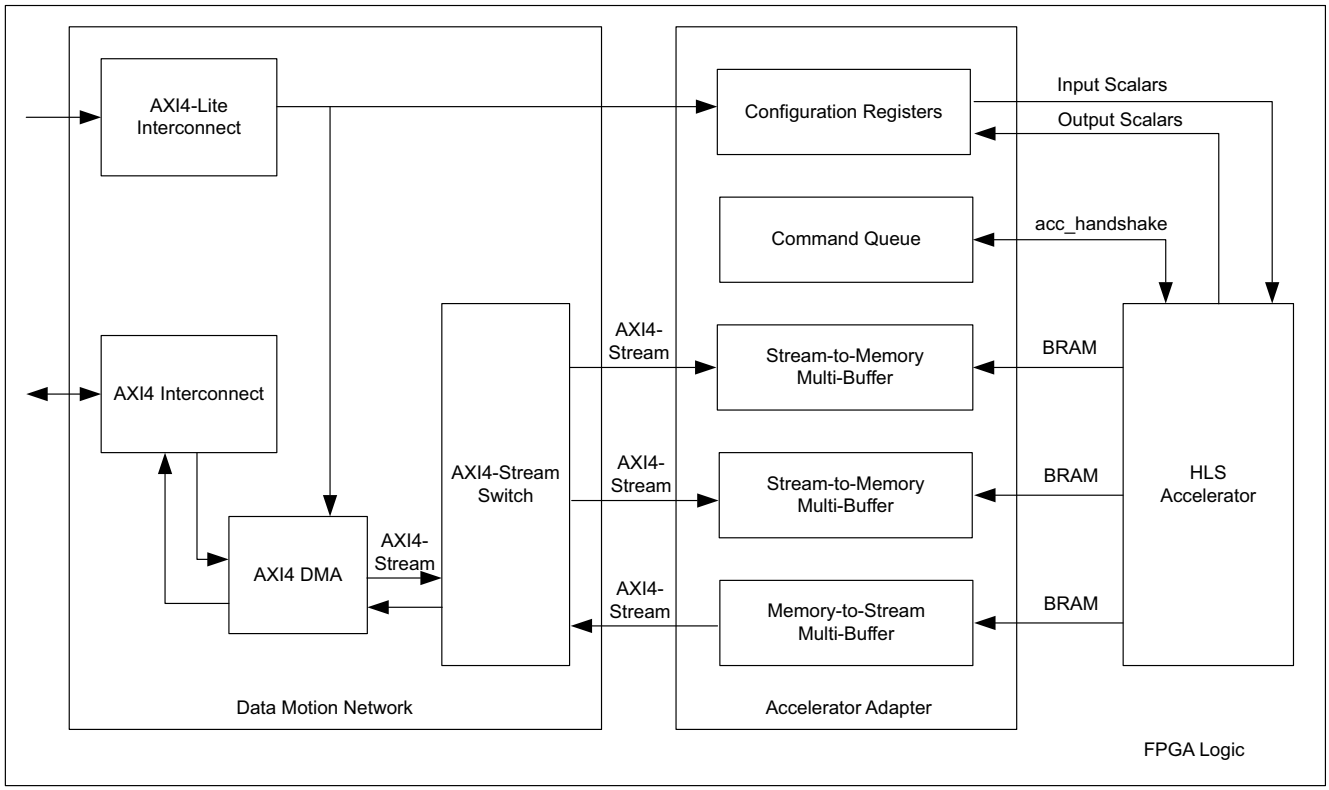

Figure 1-2: **AXI4-Stream Accelerator Adapter Block Diagram**

As shown in Figure 1-2, Accelerator Adapter core consist of the following major design blocks:

- AXI4-Stream to Block Memory Multi-Buffer
- AXI4-Stream to Accelerator FIFO
- Block Memory to AXI4-Stream Multi-Buffer
- Accelerator FIFO to AXI4-Stream
- Memory-to-Memory Multi-Buffer
- FIFO to FIFO Accelerator Communication
- Configuration Registers
- Command Queue
- Handshake Interface
- Scalar Interface

IMPORTANT: *Accelerator Adapter core requires AXI4-Stream signals to contain* `tvalid`, `tdata`, `tready`, *and* `tlast`.

AXI4-Stream to Block Memory Multi-Buffer

The Stream-to-Memory (S2M) Multi-Buffer exposes an AXI4-Stream slave interface to the data transport network, delivering input data into the Accelerator Adapter core buffers. To the accelerator, the Accelerator Adapter core presents a BRAM slave interface (`bram_rtl` IP-XACT bus definition) which gives the accelerator random access to the data set.

The Accelerator Adapter core ensures that the entire data packet has been received (as marked by `tlast` on the final data beat) before issuing a task-start signal to the accelerator over the `ap_ctrl` (`acc_handshake`) bus. The adapter also supports data-width conversion between the AXI4-Stream data channel and the BRAM word size. Therefore, the accelerator port data types and the AXI4-Stream data channels can be independently sized for higher system performance and efficiency.

AXI4-Stream to Accelerator FIFO

The Stream-to-FIFO-Read (S2F) exposes an AXI4-Stream slave interface to the data transport network, delivering input data into the Accelerator Adapter core buffers. To the accelerator, the Accelerator Adapter core presents a FIFO master interface (`acc_fifo_read` IP-XACT bus definition) which gives the accelerator in order read access to the data set.

Unlike the S2M interface, for which the Accelerator Adapter core waits until `tlast` is asserted before issuing a task-start signal to the accelerator, any input packet sent over an S2F interface is not treated as part of the input barrier group. The adapter also supports data-width conversion between the AXI4-Stream data channel and the `acc_fifo_read` word size. Therefore, the accelerator port data types and the AXI4-Stream data channels can be independently sized for higher system performance and efficiency.

The Accelerator Adapter core supports from zero to eight instances of any combination of the S2M Multi-Buffer and S2F adapters.

Block Memory to AXI4-Stream Multi-Buffer

The Memory-to-Stream (M2S) Multi-Buffer exposes an AXI4-Stream master interface to the data transport network, delivering accelerator output data from the Accelerator Adapter core buffers. To the accelerator, the Accelerator Adapter core presents a BRAM slave interface (`bram_rtl` IP-XACT bus definition) which gives the accelerator random access to the data set.

The Accelerator Adapter core does not initiate this data stream (by asserting TVALID) until the accelerator has issued the `ap_done` signal over the `acc_handshake` interface, and terminates the stream by asserting `tlast` on the final data beat. The adapter also supports data-width conversion between the BRAM and AXI4-Stream data channel word sizes. Therefore, the accelerator port data types and the AXI4-Stream data channels can be independently sized for higher system performance and efficiency.

Accelerator FIFO to AXI4-Stream

The FIFO-Write-to-Stream (F2S) exposes an AXI4-Stream master interface to the data transport network, delivering accelerator output data from the Accelerator Adapter core buffers. To the accelerator, the Accelerator Adapter core presents a FIFO slave interface (`acc_fifo_write` IP-XACT bus definition) which gives the accelerator in order write access of the data set.

Unlike the M2S interface, for which the Accelerator Adapter core waits until `ap_done` is asserted before initiating the data stream by asserting `tvalid`, the output stream is available to the AXI4-Stream transport network with a single data beat latency. The AXI4-Stream `tlast` signal is only asserted with the final data beat when the accelerator asserts task completion by ap_done over the `acc_handshake` bus. The adapter also supports data-width conversion between the `acc_fifo_write` and AXI4-Stream data channel word size. Therefore, the accelerator port data types and the AXI4-Stream data channels can be independently sized for higher system performance and efficiency.

The Accelerator Adapter core supports from zero to eight instances of any combination of the M2S Multi-Buffer and F2S adapters.

Memory-to-Memory Multi-Buffer

The Memory-to-Memory (M2M) Multi-Buffer is provided to connect two accelerators directly over AXI4-Stream transport. It provides a Multi-Buffer IP with two memory (block RAM) interfaces. Transport over the AXI4-Stream is not initiated until the first accelerator signals task completion by its `acc_handshake` bus.

FIFO to FIFO Accelerator Communication

The AXI Adapter supports streaming accelerator to accelerator communication over AXI4-Stream transport (F2F). Unlike the M2M Multi-Buffer, the AXI4-Stream is initiated as soon as data is available, with latency of a single data beat. The `tlast` packet terminator is inserted by the AXI-Stream master when the first accelerator asserts task completion over its `acc_handshake` bus. The F2F adapter provides an efficient and consistent task interface between accelerators created using Vivado HLS having `acc_fifo` interfaces which do not include hardware buffering, nor packetized data.

Configuration Registers

The Multi-Buffer has several Control and Status registers are mapped to an AXI4-Lite slave interface. These Control and Status registers are described in Chapter 2, Product Specification.

Command Queue

This is a software controllable command queue to control the execution of the accelerator. This supports task pipelining, where the software (processor) can submit multiple commands to the command queue. Accelerator Adapter core processes in order input and output configuration commands from the command queue until it finds an "execute" command that effectively defines a task barrier. When all pending input/output memory buffer and scalar requests have been met, the Accelerator Adapter core generates an accelerator `start` signal over the `acc_handshake` bus. It then waits the `done` signal before reading any additional commands from the queue. Generation of the accelerator `start` signal is based on two main inputs:

- Status of the input and output buffers and scalars (for example, `req/ack` signal from each Multi-Buffer signaling if there is data available in the input Multi-Buffers or if there is a Multi-Buffer free at the output).
- Software Control registers, which specify the input/output Multi-Buffers to consider when triggering the accelerator (that is, generate the `start` signal)

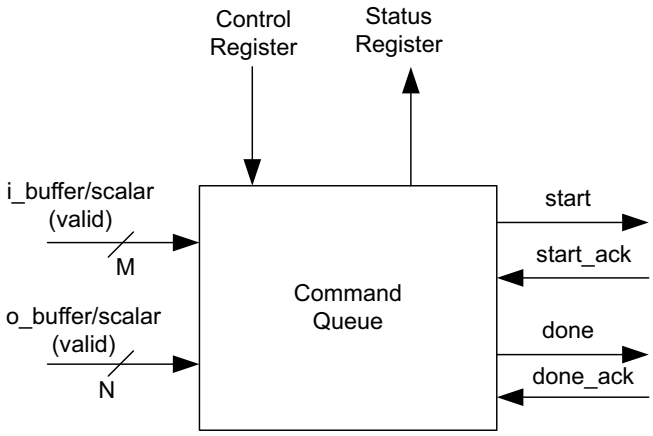

Figure 1-3: **Command Queue Interface**

This mechanism offloads the processor from the task of controlling the accelerator (`start` signal generation). The processor (software) can issue commands in advance to the command queue and the Accelerator Adapter core synchronizes input and output arrival with the accelerator task in hardware.

Additionally, there are commands to specify the activation of a specific input/output Multi-Buffers and/or scalars. This mechanism supports task pipelining by the means of specific command to the command queue.

This mechanism can be used to implement slow-changing data buffers (for example, it can be used to implement Quasi-Static buffers, which are buffers where the data does not change in each call to the accelerator) or applications where the output buffer(s) is used to accumulate multiple calls to the accelerator.

Handshake Interface

The handshake interfaces between Accelerator Adapter core and Accelerator IP to control accelerator task execution (`acc_handshake` IP-XACT bus definition). The handshake signals (`ap_start`, `ap_ready`, `ap_done`) are used to transfer the ownership of the data buffers (that is, from Accelerator Adapter core to accelerator and vice versa). Figure 1-4 shows a timing diagram for the handshake on `ap_start` and `ap_ready` signals—start asserted implies block owns input buffers.

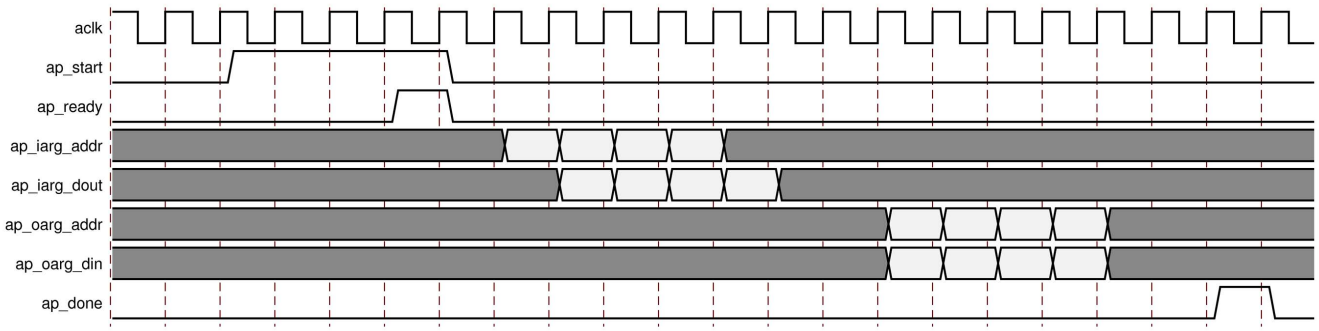

Figure 1-4: **Handshake Interface Timing**

Scalar Interface

The scalar interface is used to provide scalars parameter from software to hardware. There are a maximum eight input scalars (0-7), eight output scalars (0-7), and eight inout scalars which enable both input and output scalars in the range 8 to 15. Inout scalars data write/read address registers 0x00A0 to 0x00BC are used for data access for Input and Output scalars (8-15). Accelerator Adapter core provides a hardware buffer for each scalar to support task pipelining (new scalar values can be loaded in the background, while the accelerator is running using the previously loaded set of scalars). The Accelerator Adapter core support HLS accelerator supported I/O protocols (AP_NONE, AP_VLD, AP_HS). Each input/output scalar can be configured for these I/O protocol separately.

- **AP_NONE** – Only scalar signal, no valid and `ack` associated with scalars.
  - **Input Scalar** – Accelerator Adapter core to sample scalar at `ap_start`.
  - **Output Scalars** – Accelerator Adapter core samples output scalar at `ap_done`.
- **AP_VLD** – Scalar valid associated with each scalar.
  - **Input Scalars** – Accelerator Adapter core generates `ap_iscalar_vld` signal for each scalar.
  - **Output Scalars** – Accelerator Adapter core samples output scalar only at `ap_oscalar_vld` signal. `ap_done` signal is ignored in this case.
- **AP_HS** – Complete handshake interface. Scalar valid and `ack` associated with each scalar.
  - **Input Scalars** – Accelerator Adapter core samples scalar at `ap_iscalar_vld` and generates `ap_iscalar_ack`. The core can update the new scalar after receiving `ack`.
  - **Output Scalars** – Accelerator Adapter core samples scalar data at `ap_oscalar_vld` and generates `ap_oscalar_ack`.

Feature Summary

This section includes high-level features supported by the AXI-Stream Accelerator Adapter IP core.

Configurable Multi-Buffer Depth

Accelerator Adapter core supports a design-time configurable parameter **Buffer Depth**. The minimum depth should be 2 (that is, ping-pong buffer) and maximum supported depth is 4. This multi-buffer depth is used to hide the DMA communication overhead to enhance pipeline communication and computation.

Figure 1-5 shows examples of how overlapping communication and computation significantly improves performance.

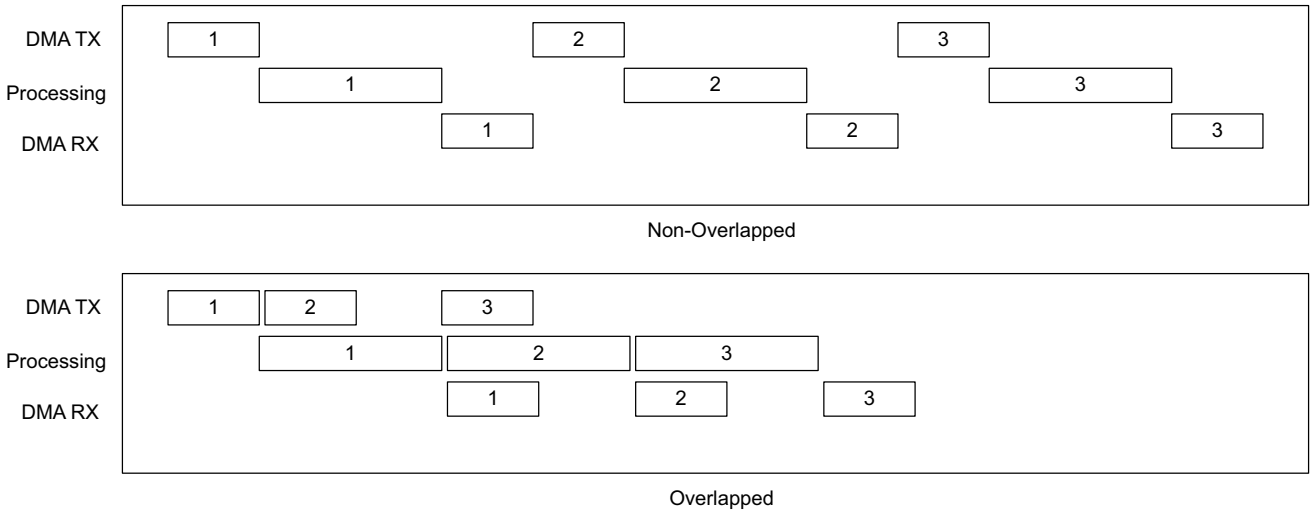

Figure 1-5: **Communication and Computation Example**

Asymmetric Multi-Buffer

The Accelerator Adapter core supports asymmetric data width for AXI4-Stream interface and block RAM interface towards accelerators. Accelerator Adapter core has a width conversion logic. This reduces the communication overhead (that is, transfer time) between AXI4-DMA and Adapter Multi-Buffer assuming that the AXI4-Stream bit width is wider than the memory (block RAM) interfaces.

Asynchronous Clock Domain

The Accelerator Adapter core supports asynchronous clock domain for AXI4-Stream and accelerator interfaces. Accelerator Adapter core logic works on the accelerator clock. This feature is useful in scenarios where accelerator logic is running faster than the AXI4 infrastructure IP or when the accelerator is performing pre/post-processing directly connected to an external I /O module using a clock domain asynchronous to the AXI4-Stream clock.

Applications

The following is a demonstration on how the Accelerator Adapter core can be combined with Vivado HLS to create hardware accelerators that support task pipelining. This is suitable for the Zynq®-7000 All Programmable SoC family and other FPGAs. This example focuses only on the Accelerator Adapter core interface. For more information on `#pragmas` and high-level synthesis, see the Vivado HLS User Guide.

In the following C++ source code for a matrix multiplier accelerator with top-level function `mmult`, matrices are represented as linearized one-dimensional arrays. In HLS, the hardware interfaces for the synthesized IP are specified by `pragmas`. The accelerator has been defined with BRAM interfaces on the inputs to enable task-pipelining with the Accelerator Adapter core S2M Multi-Buffers.

The accelerator algorithm produces output in linear order, so choosing a FIFO interface on the output argument out_C reduces latency significantly; data transfer can commence as soon as output results are available.

The **copy loops** pull inputs from the Accelerator Adapter core multi-buffer into the HLS block so that the `mmult_kernel` subfunction can further reshape the inputs into multi-ported memories for parallel access and higher performance (code omitted).

```
#include <stdio.h>
#include <stdlib.h>
#define A_NROWS 32
#define A_NCOLS 32
#define B_NCOLS 32
void mmult (float in_A[A_NROWS*A_NCOLS],
            float in_B[A_NCOLS*B_NCOLS],
            float out_C[A_NROWS*B_NCOLS])
{
#pragma HLS RESOURCE variable=in_A core=SPRAMD
#pragma HLS RESOURCE variable=in_B core=SPRAMD
#pragma HLS INTERFACE ap_fifo port=out_C
   int i, j;
   float a_buf[A_NROWS][A_NCOLS];
   float b_buf[A_NCOLS][B_NCOLS];
   // Transfer matrix A from multi-buffer into local RAM
   for(i=0; i<A_NROWS; i++) {
      for(j=0; j<A_NCOLS; j++) {
#pragma HLS PIPELINE II=1
         a_buf[i][j] = in_A[i * A_NCOLS + j];
      }
   }
   // Transfer matrix B from multi-buffer into local RAM
   for(i=0; i<A_NCOLS; i++) {
      for(j=0; j<B_NCOLS; j++) {
#pragma HLS PIPELINE II=1
         b_buf[i][j] = in_B[i * B_NCOLS + j];
      }
   }
   // Matrix multiply algorithm with reshaped inputs
   // Subfunction definition omitted for brevity
   mmult_kernel(a_buf, b_buf, out_C);
}
```

Figure 1-6 shows the Accelerator Adapter core connectivity with the Vivado IP integrator block diagram.

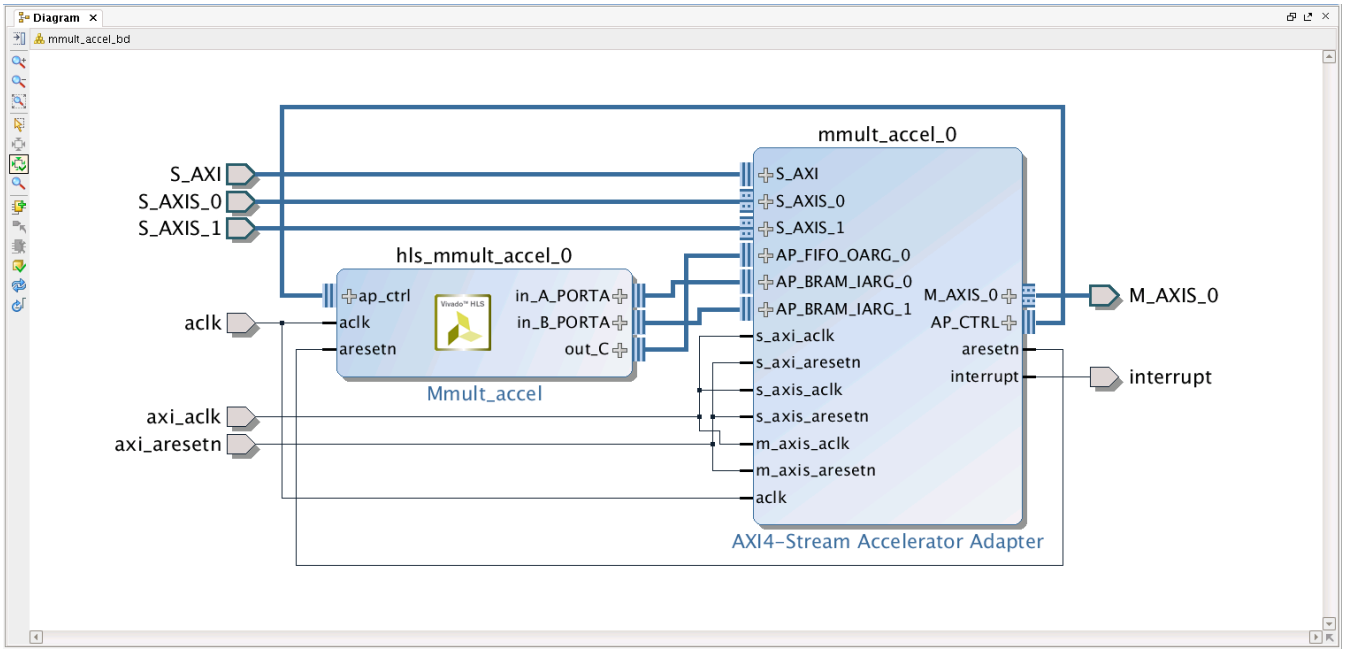

Figure 1-6: **Connectivity of Accelerator Adapter Core with Vivado IP Integrator**

Licensing and Ordering Information

This Xilinx LogiCORE™ 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 on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

This chapter contains resource usage data, signal descriptions and details about the registers.

Performance

Performance characterization of this core has been done using margin system methodology. The details of the margin system characterization methodology are available in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1].

Maximum Frequencies

Table 2-1 lists the maximum frequencies for this core.

Note: Maximum frequency numbers for UltraScale™ architecture and Zynq®-7000 All Programmable SoC devices are expected to be similar to 7 series device numbers.

Table 2-1: **Maximum Frequencies**

Family	Speed Grade	F_{Max} (MHz)		
		AXI4-Stream	Block RAM	AXI4-Lite
Virtex-7	–1	200	200	180
Kintex-7		200	200	180
Artix-7		150	150	120
Virtex-7	–2	240	240	200
Kintex-7		240	240	200
Artix-7		180	180	140
Virtex-7	–3	280	280	220
Kintex-7		280	280	220
Artix-7		200	200	160

Latency

The Accelerator Adapter core has 11 clocks latency from writing an execute command to the core to `ap_start` generation towards accelerator. The core has seven clocks latency from `ap_done` to output argument data available on AXI4-Stream channels.

Throughput

Throughput is 100% as long as the master/slave connected to this IP asserts ready High continuously.

Resource Utilization

Note: Resource numbers for UltraScale architecture and Zynq-7000 devices are expected to be similar to 7 series device numbers.

Virtex-7 FPGAs

Table 2-2 provides approximate resource counts for the various core options using Virtex®-7 FPGAs.

Table 2-2: **Device Utilization – Virtex-7 FPGAs**

Parameter Values							Design Resources		
Input Argument	Output Arguments	Streaming Width	Accelerator Interface	Accelerator Width	Input Scalar	Output Scalar	Slices Flip-Flops	LUTs	Block RAM
1	1	32	Block RAM	32	0	0	681	393	8
2	1	32	Block RAM	32	0	0	789	467	10
4	4	32	Block RAM	32	0	0	1,887	1,023	20
8	8	32	Block RAM	32	0	0	3,475	1,817	34
2	1	64	Block RAM	32	0	0	817	458	10
2	1	32	FIFO	32	0	0	1,063	532	3
2	1	64	FIFO	64	0	0	1,217	598	3
2	1	32	Block RAM	32	8	8	3,289	1,614	10

The maximum clock frequency results were obtained by double-registering input and output ports to reduce dependence on I/O placement. The inner level of registers used a separate clock signal to measure the path from the input registers to the first output register through the core. The results are post-implementation, using tool default settings except for high effort.

The resource usage results do not include the characterization registers and represent the true logic used by the core. LUT counts include SRL16s or SRL32s.

Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification. The maximum achievable clock frequency and the resource counts might also be affected by other tool options, additional logic in the FPGA, using a different version of Xilinx® tools, and other factors.

Resources required for the Accelerator Adapter core have been estimated for the Virtex-7 devices (Table 2-2). These values were generated using Vivado® IP catalog. They are derived from post-synthesis reports, and might change during implementation. Start by choosing the device, maximum frame size, and minimum block size of the core.

Port Descriptions

Table 2-3 lists the Accelerator Adapter core signals. The I/O signals include AXI interface signals through which the IP is configured, AXI4-Stream interface, and Accelerator interface signals.

Table 2-3: **I/O Signal Descriptions**

Signal Name	Interface	I/O	Initial State	Description
System Interface				
aclk	System	I	–	Accelerator Adapter core clock. Same as Accelerator domain clock.
interrupt	System	O	0x0	System interrupt output (Reserved)
aresetn	System	O	0x0	Active-Low reset output signal for Accelerator Adapter core.
AXI4-Lite Slave Interface				
s_axi_aclk	System	I	–	AXI4 interface clock
s_axi_aresetn	System	I	–	AXI4 reset. Active-Low.
s_axi_*	S_AXI	–	–	See Appendix A of the *Vivado AXI Reference Guide* (UG1037) [Ref 2] for a description of AXI4 signals.
AXI4-Stream Input Interface (Channel 0 to 8)				
s_axis_aclk	System	I	–	AXI4-Stream clock

Table 2-3: **I/O Signal Descriptions *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
s_axis_aresetn	System	I	–	AXI4-Stream interface. Active-Low.
s_axis_n_tdata	S_AXIS_n	I	–	Streaming data
s_axis_n_tvalid	S_AXIS_n	I	–	Streaming data valid
s_axis_n_tlast	S_AXIS_n	I	–	Last data beat of streaming packet
s_axis_n_tid	S_AXIS_n	I	–	Stream data identifier
s_axis_n_tready	S_AXIS_n	O	0x0	Indicates adapter ready to accept stream data
s_axis_n_tstrb	S_AXIS_n	I	–	Byte qualifier for streaming data
s_axis_n_tkeep	S_AXIS_n	I	–	Byte qualifier (data byte or a null byte)
s_axis_n_tuser	S_AXIS_n	I	–	Indicates user defined sideband signals
s_axis_n_tdest	S_AXIS_n	I	–	Indicates routing information for data stream
AXI4-Stream Output Interface (0 to 7)				
m_axis_aclk	System	I	–	AXI4-Stream clock
m_axis_aresetn	System	I	–	AXI4-Stream interface. Active-Low.
m_axis_n_tdata	M_AXIS_n	I	–	Streaming data
m_axis_n_tvalid	M_AXIS_n	O	0x0	Streaming data valid
m_axis_n_tlast	M_AXIS_n	O	0x0	Last data beat of streaming packet
m_axis_n_tid	M_AXIS_n	O	0x0	Stream data identifier
m_axis_n_tready	M_AXIS_n	I	–	Indicates slave ready to accept stream data
m_axis_n_tstrb	M_AXIS_n	O	0x0	Byte Qualifier for streaming data
m_axis_n_tkeep	M_AXIS_n	O	0x0	Byte qualifier (data byte or a null byte)
m_axis_n_tuser	M_AXIS_n	O	0x0	Indicates user defined sideband signals
m_axis_n_tdest	M_AXIS_n	O	0x0	Indicates routing information for data stream
Accelerator Input Argument Block RAM Interface (0 to 7)				
ap_iarg_n_clk	AP_BRAM_IARG_n	I	–	Block RAM interface clock. Port unused in this core.
ap_iarg_n_rst	AP_BRAM_IARG_n	I	–	Block RAM interface reset. Port unused in this core.
ap_iarg_n_ce	AP_BRAM_IARG_n	I	–	Block RAM interface chip enable
ap_iarg_n_we	AP_BRAM_IARG_n	I	–	Block RAM interface write enable
ap_iarg_n_addr	AP_BRAM_IARG_n	I	–	Block RAM interface address bus
ap_iarg_n_dout	AP_BRAM_IARG_n	O	0x0	Block RAM interface read data bus
ap_iarg_n_din	AP_BRAM_IARG_n	I	–	Block RAM interface write data bus
Accelerator Output Argument Block RAM Interface (0 to 7)				
ap_oarg_n_clk	AP_BRAM_OARG_n	I	–	Block RAM interface clock. Port unused in this core.

Table 2-3: **I/O Signal Descriptions *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
ap_oarg_n_rst	AP_BRAM_OARG_n	I	–	Block RAM interface reset. Port unused in this core.
ap_oarg_n_ce	AP_BRAM_OARG_n	I	–	Block RAM interface chip enable
ap_oarg_n_we	AP_BRAM_OARG_n	I	–	Block RAM interface write enable
ap_oarg_n_addr	AP_BRAM_OARG_n	I	–	Block RAM interface address bus
ap_oarg_n_dout	AP_BRAM_OARG_n	O	0x0	Block RAM interface read data bus
ap_oarg_n_din	AP_BRAM_OARG_n	I	–	Block RAM interface write data bus
Accelerator Input Argument FIFO Interface (0 to 7)				
ap_fifo_iarg_n_read	AP_FIFO_IARG_n	I	–	FIFO interface read enable
ap_fifo_iarg_n_dout	AP_FIFO_IARG_n	O	0x0	FIFO interface read data
ap_fifo_iarg_n_empty_n	AP_FIFO_IARG_n	O	0x0	FIFO interface FIFO empty indicator
Accelerator Output Argument FIFO Interface (0 to 7)				
ap_fifo_oarg_n_write	AP_FIFO_OARG_n	I	–	FIFO interface write enable
ap_fifo_oarg_n_din	AP_FIFO_OARG_n	I	–	FIFO interface write Data
ap_fifo_oarg_n_full_n	AP_FIFO_OARG_n	O	0x0	FIFO interface FIFO full indicator
Accelerator Control Interface				
ap_idle	AP_CTRL	I	–	Accelerator idle
ap_ready	AP_CTRL	I	–	Accelerator ready
ap_start	AP_CTRL	O	0x0	Start indicator for accelerator execution (Single Step)
ap_continue	AP_CTRL	O	0x0	Continuous start indicator for accelerator execution
ap_done	AP_CTRL	I	–	Accelerator generates done after task execution
Scalar Interface (0 to 15 for Output Scalars, 0 to 15 for Input Scalars)				
ap_iscalar_n_dout	AP_SCALAR	O	0x0	Input scalar to accelerator
ap_oscalar_n_din	AP_SCALAR	I	–	Output scalar from accelerator to Accelerator Adapter core
ap_iscalar_n_vld	AP_SCALAR	O	0x0	Input scalar valid signal. Used for scalar interface mode AP_VLD, AP_HS.
ap_iscalar_n_ack	AP_SCALAR	I	–	Input scalar acknowledgement from Accelerator. Used in AP_HS mode
ap_oscalar_n_vld	AP_SCALAR	I	–	Output scalar valid signal. Used for scalar interface mode AP_VLD, AP_HS.

Table 2-3: **I/O Signal Descriptions *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
ap_oscalar_n_ack	AP_SCALAR	O	0x0	Output scalar acknowledgment from adapter. Used for scalar interface mode AP_HS.

Notes:

1. Accelerator Adapter core supports up to eight input and output argument channels, eight Output Scalar interface, and up to 16 Input Scalar interfaces.

Register Space

This section details the registers and their reset values of Accelerator Adapter core. Table 2-4 shows all Accelerator Adapter core register and their address map.

Table 2-4: **Accelerator Adapter Core Register Map**

Register Index	Address Offset	Register Name	Access Type	Description
0	0x0000	CTRL	R/W	Control
1	0x0004	STATUS	R/W1C	Status
4	0x0010	IARG_RQT_EN	R/W	Input Argument Request Enable
5	0x0014	OARG_RQT_EN	R/W	Output Argument Request Enable
10	0x0028	CMD	WO	Command
15	0x003C	OARG_LENGTH_MODE	R/W	Output Argument Length Mode
16	0x0040	ISCALAR_FIFO_RST	WO	Input Scalar FIFO Reset
17	0x0044	OSCALAR_FIFO_RST	WO	Output Scalar FIFO Reset
18	0x0048	ISCALAR_RQT_EN	WO	Input Scalar Request Enable
19	0x004C	OSCALAR_RQT_EN	WO	Output Scalar Request Enable
32	0x0080	ISCALAR0_DATA	WO	Input Scalar-0 Write Data FIFO
33	0x0084	ISCALAR1_DATA	WO	Input Scalar-1 Write Data FIFO
34	0x0088	ISCALAR2_DATA	WO	Input Scalar-2 Write Data FIFO
35	0x008C	ISCALAR3_DATA	WO	Input Scalar-3 Write Data FIFO
36	0x0090	ISCALAR4_DATA	WO	Input Scalar-4 Write Data FIFO
37	0x0094	ISCALAR5_DATA	WO	Input Scalar-5 Write Data FIFO
38	0x0098	ISCALAR6_DATA	WO	Input Scalar-6 Write Data FIFO
39	0x009C	ISCALAR7_DATA	WO	Input Scalar-7 Write Data FIFO
40	0x00A0	ISCALAR8_DATA	WO	Inout Scalar-0 Write/Read Data FIFO
41	0x00A4	ISCALAR9_DATA	WO	Inout Scalar-1 Write/Read Data FIFO
42	0x00A8	ISCALAR10_DATA	WO	Inout Scalar-2 Write/Read Data FIFO

Table 2-4: **Accelerator Adapter Core Register Map *(Cont'd)***

Register Index	Address Offset	Register Name	Access Type	Description
43	0x00AC	ISCALAR11_DATA	WO	Inout Scalar-3 Write/Read Data FIFO
44	0x00B0	ISCALAR12_DATA	WO	Inout Scalar-4 Write/Read Data FIFO
45	0x00B4	ISCALAR13_DATA	WO	Inout Scalar-5 Write/Read Data FIFO
46	0x00B8	ISCALAR14_DATA	WO	Inout Scalar-6 Write/Read Data FIFO
47	0x00BC	ISCALAR15_DATA	WO	Inout Scalar-7 Write/Read Data FIFO
48	0x00C0	OSCALAR0_DATA	RO	Output Scalar-0 Read Data FIFO
49	0x00C4	OSCALAR1_DATA	RO	Output Scalar-1 Read Data FIFO
50	0x00C8	OSCALAR2_DATA	RO	Output Scalar-2 Read Data FIFO
51	0x00CC	OSCALAR3_DATA	RO	Output Scalar-3 Read Data FIFO
52	0x00D0	OSCALAR4_DATA	RO	Output Scalar-4 Read Data FIFO
53	0x00D4	OSCALAR5_DATA	RO	Output Scalar-5 Read Data FIFO
54	0x00D8	OSCALAR6_DATA	RO	Output Scalar-6 Read Data FIFO
55	0x00DC	OSCALAR7_DATA	RO	Output Scalar-7 Read Data FIFO
64	0x0100	IARG0_STATUS	RO	Input Argument-0 Buffer Status
65	0x0104	IARG1_STATUS	RO	Input Argument-1 Buffer Status
66	0x0108	IARG2_STATUS	RO	Input Argument-2 Buffer Status
67	0x010C	IARG3_STATUS	RO	Input Argument-3 Buffer Status
68	0x0110	IARG4_STATUS	RO	Input Argument-4 Buffer Status
69	0x0114	IARG5_STATUS	RO	Input Argument-5 Buffer Status
70	0x0118	IARG6_STATUS	RO	Input Argument-6 Buffer Status
71	0x011C	IARG7_STATUS	RO	Input Argument-7 Buffer Status
80	0x0140	OARG0_STATUS	RO	Output Argument-0 Buffer Status
81	0x0144	OARG1_STATUS	RO	Output Argument-1 Buffer Status
82	0x0148	OARG2_STATUS	RO	Output Argument-2 Buffer Status
83	0x014C	OARG3_STATUS	RO	Output Argument-3 Buffer Status
84	0x0150	OARG4_STATUS	RO	Output Argument-4 Buffer Status
85	0x0154	OARG5_STATUS	RO	Output Argument-5 Buffer Status
86	0x0158	OARG6_STATUS	RO	Output Argument-6 Buffer Status
87	0x015C	OARG7_STATUS	RO	Output Argument-7 Buffer Status
96	0x0180	ISCALAR0_STATUS	RO	Input Scalar-0 Status
97	0x0184	ISCALAR1_STATUS	RO	Input Scalar-1 Status
98	0x0188	ISCALAR2_STATUS	RO	Input Scalar-2 Status
99	0x018C	ISCALAR3_STATUS	RO	Input Scalar-3 Status
100	0x0190	ISCALAR4_STATUS	RO	Input Scalar-4 Status

Table 2-4: **Accelerator Adapter Core Register Map *(Cont'd)***

Register Index	Address Offset	Register Name	Access Type	Description
101	0x0194	ISCALAR5_STATUS	RO	Input Scalar-5 Status
102	0x0198	ISCALAR6_STATUS	RO	Input Scalar-6 Status
103	0x019C	ISCALAR7_STATUS	RO	Input Scalar-7 Status
104	0x01A0	ISCALAR8_STATUS	RO	Inout Input Scalar-0 Status
105	0x01A4	ISCALAR9_STATUS	RO	Inout Input Scalar-1 Status
106	0x01A8	ISCALAR10_STATUS	RO	Inout Input Scalar-2 Status
107	0x01AC	ISCALAR11_STATUS	RO	Inout Input Scalar-3 Status
108	0x01B0	ISCALAR12_STATUS	RO	Inout Input Scalar-4 Status
109	0x01B4	ISCALAR13_STATUS	RO	Inout Input Scalar-5 Status
110	0x01B8	ISCALAR14_STATUS	RO	Inout Input Scalar-6 Status
111	0x01BC	ISCALAR15_STATUS	RO	Inout Input Scalar-7 Status
112	0x01C0	OSCALAR0_STATUS	RO	Output Scalar-0 Status
113	0x01C4	OSCALAR1_STATUS	RO	Output Scalar-1 Status
114	0x01C8	OSCALAR2_STATUS	RO	Output Scalar-2 Status
115	0x01CC	OSCALAR3_STATUS	RO	Output Scalar-3 Status
116	0x01D0	OSCALAR4_STATUS	RO	Output Scalar-4 Status
117	0x01D4	OSCALAR5_STATUS	RO	Output Scalar-5 Status
118	0x01D8	OSCALAR6_STATUS	RO	Output Scalar-6 Status
119	0x01DC	OSCALAR7_STATUS	RO	Output Scalar-7 Status
120	0x01E0	OSCALAR8_STATUS	RO	Inout Output Scalar-0 Status
121	0x01E4	OSCALAR9_STATUS	RO	Inout Output Scalar-1 Status
122	0x01E8	OSCALAR10_STATUS	RO	Inout Output Scalar-2 Status
123	0x01EC	OSCALAR11_STATUS	RO	Inout Output Scalar-3 Status
124	0x01F0	OSCALAR12_STATUS	RO	Inout Output Scalar-4 Status
125	0x01F4	OSCALAR13_STATUS	RO	Inout Output Scalar-5 Status
126	0x01F8	OSCALAR14_STATUS	RO	Inout Output Scalar-6 Status
127	0x01FC	OSCALAR15_STATUS	RO	Inout Output Scalar-7 Status
128	0x0200	OARG0_LENGTH	WO	Output Argument-0 Length
129	0x0204	OARG1_LENGTH	WO	Output Argument-1 Length
130	0x0208	OARG2_LENGTH	WO	Output Argument-2 Length
131	0x020C	OARG3_LENGTH	WO	Output Argument-3 Length
132	0x0210	OARG4_LENGTH	WO	Output Argument-4 Length
133	0x0214	OARG5_LENGTH	WO	Output Argument-5 Length
134	0x0218	OARG6_LENGTH	WO	Output Argument-6 Length

Table 2-4: **Accelerator Adapter Core Register Map *(Cont'd)***

Register Index	Address Offset	Register Name	Access Type	Description
135	0x021C	OARG7_LENGTH	WO	Output Argument-7 Length
144	0x0240	OARG0_TDEST	R/W	Output Argument-0 TDEST
145	0x0244	OARG1_TDEST	R/W	Output Argument-1 TDEST
146	0x0248	OARG2_TDEST	R/W	Output Argument-2 TDEST
147	0x024C	OARG3_TDEST	R/W	Output Argument-3 TDEST
148	0x0250	OARG4_TDEST	R/W	Output Argument-4 TDEST
149	0x0254	OARG5_TDEST	R/W	Output Argument-5 TDEST
150	0x0258	OARG6_TDEST	R/W	Output Argument-6 TDEST
151	0x025C	OARG7_TDEST	R/W	Output Argument-7 TDEST

Notes:
1. W1C – Write 1 to clear the specific bit of this register.

Register Descriptions

Control Register (CTRL)

Control provides soft reset option for the Accelerator Adapter core.

Table 2-5: **Control Register (0x0000)**

Bits	Name	Reset Value	Access Type	Description
31:2	Reserved	N/A	N/A	Reserved
1	GIE	0x0	R/W	Global Interrupt enable bit. This bit is unused in this version of the core.
0	RST	0x0	R/W	Soft Reset. Resets adapter core logic.

Status Register (STATUS)

Status provides current status of accelerator control interface of Accelerator Adapter core.

Table 2-6: **Status Register (0x0004)**

Bits	Name	Reset Value	Access Type	Description
31:4	Reserved	N/A	N/A	Reserved
3	READY	0x1	R/W1C	Accelerator Ready
2	IDLE	0x0	R/W1C	Accelerator IDLE

Table 2-6: **Status Register (0x0004) *(Cont'd)***

Bits	Name	Reset Value	Access Type	Description
1	DONE	0x0	R/W1C	Accelerator Done
0	START	0x0	R/W1C	Accelerator Start

Notes:
1. W1C – Write 1 to clear the specific bit of this register.

Input Argument Request Enable Register (IARG_RQT_EN)

Input Argument Request Enable is used by the Accelerator Adapter core to issue new task execution (`ap_start` signal) to the accelerator. The core does not issue an `ap_start` until all input arguments have valid data (that is, data to process by accelerator). This register enables to select which input arguments is used in the generation of `start` signal. If the particular bit is 1, then respective input argument is used in generation of `start` signal when it has valid data. A value of 0 means that `start` generates independently of the data in the input argument. If the number of input argument is 0, then this register is not used.

Table 2-7: **Input Argument Request Enable Register (0x0010)**

Bits	Name	Reset Value	Access Type	Description
31:N	Reserved	N/A	N/A	Reserved
N – 1:0	IARG_EN	0x1	R/W	Request enable for input arguments [N – 1:0]

Output Argument Request Enable Register (OARG_RQT_EN)

Output Argument Request Enable is used by the Accelerator Adapter core to issue new task execution (`ap_start` signal) to the accelerator. The core does not issue an `ap_start` until there is space in the respective output buffer. This register enables to select which output arguments are used in the generation of `start` signal. If the particular bit is 1, then respective output argument is used in generation of `start` signal when it has buffer available. A value of 0 means that respective argument buffer is not considered for `start` generation. If the number of output argument is 0, then this register is not used.

Table 2-8: **Output Argument Request Enable Register (0x0014)**

Bits	Name	Reset Value	Access Type	Description
31:N	Reserved	N/A	N/A	Reserved
N – 1:0	OARG_EN	0x1	R/W	Request enable for output arguments [N – 1:0]

Command Register (COMMAND)

Command is used to input the commands into command queue.

Table 2-9: **Command Register (0x0028)**

Bits	Name	Reset Value	Access Type	Description
31:28	Reserved	N/A	N/A	Reserved
27:20	ISCALAR_MASK	0x0	WO	Scalar mask for inout scalars [7:0]
19:16	OPCODE	0x0	WO	Command Opcode 0000 = Update input argument 0001 = Update output argument 0010 = Execute step (Single iteration) 0011 = Reserved 0100 = Continuous run 0101 = Stop Continuous run 0110 to 1111 = Reserved
15:8	SCALAR_MASK	0x0	WO	Scalar Mask for [7:0] scalars
7:0	ARG_MASK	0x0	WO	Argument Mask

The Execute command is used to start the execution in the accelerator. After this command is in the command queue, the Accelerator Adapter core generates the `start` signal when there is valid data in input argument and scalars (that is, multi-buffer and FIFOs not empty) and there is free space in the output arguments and scalars.

The Update Input command is used to enable data reuse for input arguments (multi-buffer and scalar). That is, to use previously received data and avoid unnecessary data movement overhead. When a particular bit in ARG_MASK/SCALAR_MASK is set to 1, the corresponding input argument/scalar is moved to the next position in the multi-buffer/FIFO. When the bit is set to 0, the same buffer data is reused.

The Update Output command is used to enable data reuse for output arguments. When a particular bit is set to 1, at the end of the accelerator execution (`ap_done`), data in corresponding buffer sends on the output AXI4-Stream channel and output buffer moves to the next position. A value of 0 means no data sent on output AXI4-Stream channel.

The Continuous run command is used to reduce the processor time for every data transfer. After this command, the core generates the `start` signal when there is valid data in input argument and scalars (that is, multi-buffer and FIFOs not empty) and there is free space in the output arguments and scalars.

The Accelerator Adapter core continues to generate the start condition on every `ap_done` signal as long as this condition is met. The continuous run operation can be stopped by using the Stop Continuous command.

If the Accelerator Adapter core receives any commands except Stop continuous run when core is running in Continuous mode, the core operation halts. The only method to resume the core operation is by applying reset to the core.

Output Argument Length Mode Register (OARG_LENGTH_MODE)

Output Argument Length Mode is used to set the mode for generating the output AXI4-Stream length (that is, output arguments). Two modes are available for output stream generation:

- Hardware
- Software

In hardware mode (memory interface), the number of data beats in AXI4-Stream is A+1, where A is the highest address written during the accelerator execution. For hardware mode (FIFO interface), `tlast` signal on output AXI4-Stream is generated after the Accelerator Adapter core receives `done` signal.

In software mode, the number of data elements on output AXI4-Stream is set using OARGx_LENGTH_REG register. K-bit with a value 0 = hardware mode, 1 = software mode.

A value of 0 means that respective output argument is configured in hardware mode, 1 means output argument is configured in software mode. If the number of output argument is 0, then this register is not used.

Table 2-10: **Output Argument Length Mode Register (0x003C)**

Bits	Name	Reset Value	Access Type	Description
31:N	Reserved	N/A	N/A	Reserved
N – 1:0	LENGTH_MODE	0x0	R/W	Length mode for output arguments [N – 1:0]

Input Scalar FIFO Reset Register (ISCALAR_FIFO_RST)

Input Scalar FIFO Reset is used to reset the scalar FIFO. If the particular bit is 1, then respective input scalar FIFO resets. N represents number of input scalars and M represents number of Inout scalars plus maximum allowed input scalars. Maximum allowed input scalars are eight in the current version of the core.

Table 2-11: **Input Scalar FIFO Reset Register (0x0040)**

Bits	Name	Reset Value	Access Type	Description
31:M	Reserved	N/A	N/A	Reserved

Table 2-11: **Input Scalar FIFO Reset Register (0x0040) *(Cont'd)***

Bits	Name	Reset Value	Access Type	Description
M – 8:1	FIFO_RST	0x0	WO	Reset for Inout Scalar FIFOs [M – 8:1]
N – 1:0	FIFO_RST	0x0	WO	Reset for Input Scalar FIFOs [N – 1:0]

Output Scalar FIFO Reset Register (OSCALAR_FIFO_RST)

Output Scalar FIFO Reset is used to reset the scalar FIFO. If the particular bit is 1, then respective output scalar FIFO resets. N represents number of output scalars and M represents number of Inout scalars plus maximum allowed output scalars. Maximum allowed output scalars are eight in the current version of the core.

Table 2-12: **Output Scalar FIFO Reset Register (0x0044)**

Bits	Name	Reset Value	Access Type	Description
31:M	Reserved	N/A	N/A	Reserved
M – 8:1	FIFO_RST	0x0	WO	Reset for Inout Scalar FIFOs [M – 8:1]
N – 1:0	FIFO_RST	0x0	WO	Reset for Output Scalar FIFOs [N – 1:0]

Input Scalar Request Enable Register (ISCALAR_RQT_EN)

Input Scalar Request Enable is used to enable the scalars for accelerator `start` generation. When a particular scalar bit is set to 1 and corresponding scalar has data available, the Accelerator Adapter core generates the `start` to accelerator. If the scalar bit is set to 0, corresponding scalar is not considered for `start` generation. N represents number of input scalars and M represents number of Inout scalars plus maximum allowed input scalars. Maximum allowed input scalars are eight in the current version of the core.

Table 2-13: **Input Scalar Request Enable Register (0x0048)**

Bits	Name	Reset Value	Access Type	Description
31:M	Reserved	N/A	N/A	Reserved
M – 8:1	RQT_EN	0x1	WO	Request enable for Inout Scalars [M – 8:1]
N – 1:0	RQT_EN	0x1	WO	Request enable for Input Scalars [N – 1:0]

Output Scalar Request Enable Register (OSCALAR_RQT_EN)

Output Scalar Request Enable is used to enable the scalars for accelerator `start` generation. When a particular scalar bit is set to 1 and corresponding scalar FIFO has space available, the Accelerator Adapter core generates the `start` to accelerator. If the scalar bit is set to 0, corresponding scalar is not considered for `start` generation. N represents number of output scalars and M represents number of Inout scalars plus maximum allowed output scalars. Maximum allowed output scalars are eight in the current version of the core.

Table 2-14: **Output Scalar Request Enable Register (0x004C)**

Bits	Name	Reset Value	Access Type	Description
31:M	Reserved	N/A	N/A	Reserved
M – 8:1	RQT_EN	0x1	WO	Request enable for Inout Scalars [M – 8:1]
N – 1:0	RQT_EN	0x1	WO	Request enable for Output Scalars [N – 1:0]

Input Scalar Write Data Register (ISCALARn_DATA)

Input Scalar Write Data is used to provide scalars from the Accelerator Adapter core to accelerator. Each input scalar has 16 beat deep FIFO. The status of each scalar FIFO (occupancy, empty, full) is provided in the Input Scalar Status register. This is a write only register. N represents the width of respective scalar and n represents input scalar number.

Table 2-15: **Input Scalar Write Data Register (0x0080 to 0x009C)**

Bits	Name	Reset Value	Access Type	Description
31:N	Reserved	N/A	N/A	Reserved
N – 1:0	ISCALARn_DATA	0x0	WO	Input Scalars data [N – 1:0]

Inout Scalar Read/Write Data Register (IOSCALARn_DATA)

Inout Scalar Read/Write Data is used to provide scalars from the Accelerator Adapter core to accelerator and vice versa. Each inout scalar has 16 beat deep FIFO. The status of each scalar FIFO (occupancy, empty, full) is provided in the Inout Scalar Status register. This is a read/write register. N represents the width of respective scalar and n represents inout scalar number.

Table 2-16: **Inout Scalar Read/Write Data Register (0x00A0 to 0x00BC)**

Bits	Name	Reset Value	Access Type	Description
31:N	Reserved	N/A	N/A	Reserved
N – 1:0	IOSCALARn_DATA	0x0	R/W	Inout Scalars data [N – 1:0]

Output Scalar Read Data Register (OSCALARn_DATA)

Output Scalar Read Data is used to provide scalars received from accelerator to the Accelerator Adapter core. Each output scalar has 16 beat deep FIFO. The status of each scalar FIFO (occupancy, empty, full) is provided in the Output Scalar Status register. This is a read only register. N represents the width of respective output scalar and n represents output scalar number.

Table 2-17: **Output Scalar Read Data Register (0x00C0 to 0x00DC)**

Bits	Name	Reset Value	Access Type	Description
31:N	Reserved	N/A	N/A	Reserved
N – 1:0	OSCALARn_DATA	0x0	RO	Output Scalar Read Data [N – 1:0]

Input Buffer Status Register (IARGn_STATUS)

Input Buffer Status provides status of each input argument buffer such as number of available buffers, buffer empty, and buffer full. Multi-buffer status for each argument is provided in separate registers. The status provided by this register is valid only when argument type is configured for the block RAM interface.

Table 2-18: **Input Buffer Status Register (0x0100 to 0x011C)**

Bits	Name	Reset Value	Access Type	Description
31:6	Reserved	N/A	N/A	Reserved
5	IARGFULL	0x0	RO	Input argument buffer full
4	IARGEMPTY	0x1	RO	Input argument buffer empty
3:0	IARGBUF	0x0	RO	Number of used buffer in input argument

Output Buffer Status Register (OARGn_STATUS)

Output Buffer Status provides status of each output argument buffer such as number of available buffers, buffer empty, and buffer full. Multi-buffer status for each argument is provided in separate registers. The status provided by this register is valid only when argument type is configured for the block RAM interface.

Table 2-19: **Output Buffer Status Register (0x0140 to 0x015C)**

Bits	Name	Reset Value	Access Type	Description
31:6	Reserved	N/A	N/A	Reserved
5	OARGFULL	0x0	RO	Output argument buffer full
4	OARGEMPTY	0x1	RO	Output argument buffer empty
3:0	OARGBUF	0x0	RO	Number of used buffer in output argument

Input Scalar Status Register (ISCALARn_STATUS)

Input Scalar Status provides status of each input scalar FIFO such as number of positions available in scalar FIFO, FIFO empty, and FIFO full. Scalar FIFO status for each scalar is provided in separate registers.

Table 2-20: **Input Scalar Status Register (0x0180 to 0x019C)**

Bits	Name	Reset Value	Access Type	Description
31:6	Reserved	N/A	N/A	Reserved
5	ISCLRFULL	0x0	RO	Input scalar FIFO full. Any data writes to the Input scalar FIFO when this bit is set would be ignored.
4	ISCLREMPTY	0x1	RO	Input scalar FIFO empty
3:0	ISCLRBUF	0x0	RO	Number of positions used in input scalar FIFO

Inout Scalar Status Register (IOSCALARn_STATUS)

Inout Scalar Status provides status of each inout scalar FIFO such as number of positions available in scalar FIFO, FIFO empty, and FIFO full. Scalar FIFO status for each scalar is provided in separate registers.

Table 2-21: **Inout Scalar Status Register (0x01A0 to 0x01BC)**

Bits	Name	Reset Value	Access Type	Description
31:6	Reserved	N/A	N/A	Reserved
5	IOSCLRFULL	0x0	RO	Inout scalar FIFO full. Any data writes to the Inout scalar FIFO when this bit is set would be ignored.
4	IOSCLREMPTY	0x1	RO	Inout scalar FIFO empty
3:0	IOSCLRBUF	0x0	RO	Number of positions used in inout scalar FIFO

Output Scalar Status Register (OSCALARn_STATUS)

Output Scalar Status provides status of each output scalar FIFO such as number of positions available in scalar FIFO, FIFO empty, and FIFO full. Scalar FIFO status for each scalar is provided in separate registers.

Table 2-22: **Output Scalar Status Register (0x01C0 to 0x01DC)**

Bits	Name	Reset Value	Access Type	Description
31:6	Reserved	N/A	N/A	Reserved
5	OSCLRFULL	0x0	RO	Output scalar FIFO full. Any data writes to the Output scalar FIFO when this bit is set would be ignored.

Table 2-22: **Output Scalar Status Register (0x01C0 to 0x01DC) *(Cont'd)***

Bits	Name	Reset Value	Access Type	Description
4	OSCLREMPTY	0x1	RO	Output scalar FIFO empty
3:0	OSCLRBUF	0x0	RO	Number of positions used in output scalar FIFO

Inout Oscalar Status Register (Inout OSCALARn_STATUS)

Inout Oscalar Status provides status of each inout oscalar FIFO such as number of positions available in scalar FIFO, FIFO empty, and FIFO full. Scalar FIFO status for each scalar is provided in separate registers.

Table 2-23: **Inout Scalar Status Register (0x01E0 to 0x01FC)**

Bits	Name	Reset Value	Access Type	Description
31:6	Reserved	N/A	N/A	Reserved
5	IO_OSCLRFULL	0x0	RO	Inout oscalar FIFO full. Any data writes to the Inout scalar FIFO when this bit is set would be ignored.
4	IO_OSCLREMPTY	0x1	RO	Inout oscalar FIFO empty
3:0	IO_OSCLRBUF	0x0	RO	Number of positions used in inout oscalar FIFO

Output Argument Length Register (OARGn_LENGTH)

Output Argument Length is used to set length for argument for each output AXI4-Stream. This output argument length is used by the core only when output argument is configured for software mode. OARG_LENGTH_MODE register configures output argument mode. If the output argument is configured for hardware mode, this length value does not have any effect on the output stream data generation.

Table 2-24: **Output Argument Length Register (0x0200 to 0x021C)**

Bits	Name	Reset Value	Access Type	Description
31:16	Reserved	N/A	N/A	Reserved
15:0	OARG_LEN	0x0	WO	Number of data elements to stream in output stream. This length represents the number of data beats with respect to accelerator output argument data width.

Output Argument TDEST Register (OARGn_TDEST)

Output Argument TDEST is used to set the value on TDEST signal on output AXI4-Stream.

Table 2-25: **Output Argument TDEST Register (0x0240 to 0x025C)**

Bits	Name	Reset Value	Access Type	Description
31:N	Reserved	N/A	N/A	Reserved
N – 1:0	OARG_TDEST	0x0	R/W	Value to set on TDEST signal. N represents your configured TDEST width.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

General Design Guidelines

The following steps are recommended for all designs using the Accelerator Adapter core.

1. Instantiate the Accelerator Adapter core in the system as shown in Figure 1-1, page 5.
2. Configure the core for required number of input and output arguments.
3. Configure the core for required input and output scalars and their respective I/O protocols.
4. Connect proper clock and reset connection to all the available ports. Connect `aresetn` output port to the HLS accelerator reset input port.
5. Program the core as per the steps defined in Programming Sequence for Accelerator Adapter.

Programming Sequence for Accelerator Adapter

This section defines the programing sequence for the Accelerator Adapter core for two input, one output argument, one input Scalar, and one output scalar configuration.

1. Apply soft reset to the core by writing 0x00000001 to Control Register (0x0000).
2. Configure Input Argument Request Enable Register (0x0010) to define input buffer selection for `ap_start` generation. `start` is generated only if the selected input argument buffer has data available. By default, all input argument buffer are considered for `start` generation.
3. Configure Output Argument Request Enable Register (0x0014) to define output buffer selection for `ap_start` generation. `start` is generated only if the selected output argument buffer has space available. By default, all input argument buffer are considered for `start` generation.

4. If scalars are enable, configure the Input Scalar Request Enable Register (0x0048) and Output Scalar Request Enable Register (0x004C) to define the `start` generation. By default, all input and output scalars are considered for `start` generation.
5. Update the scalar data to be transmitted to Accelerator in Input and Inout Scalar Write Data.
6. Configure Output Argument Length Mode Register (0x003C) for hardware mode or software mode operation. In hardware mode, output stream length is generated based on the data received from the accelerator.

   In software mode, configure the Output Argument Length Mode Register (0x003C) by writing OARG_LENGTH for each output argument.
7. Write Update Output command to adapter by writing 0x00010001 to Command Register (0x0028). By writing 1 to the argument, mask moves the output buffer to the next position on every data output on stream channel.
8. Write TDEST value in Output Argument TDEST Register (0x0240 to 0x025C).
9. Write Execute command 0x00020000 in Command Register (0x0028) to start the operation.
10. After completing the Accelerator operation, done status is updated in the Status Register (0x0004). Output scalar data can be read now from OSCALAR_DATA and IO_OSCALAR_DATA.
11. Write Update Output command to adapter by writing 0x00000003 to Command Register (0x0028). By writing 1 to the argument, mask moves the input buffer pointer to the next position in multi-buffer. Writing 0 reuses the same buffer.

Clocking

Accelerator Adapter core supports synchronous clock domain between core clock, register interface clock, and streaming interface clock. The following clock in the same clock domains are supported:

- AXI4-Lite clock domain is clocked by `s_axi_aclk`
- Core and accelerator clock interface is clocked by `aclk`
- Input stream interface clock domain is clocked by `s_axis_aclk`
- Output stream interface clock domain is clocked by `m_axis_aclk`

Resets

Accelerator Adapter core supports active-Low reset signal for each available clock domain. Reset output signal `aresetn` is the active-Low reset for HLS accelerator IP.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the Vivado IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 3]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5]

Customizing and Generating the Core

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 3] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4].

Note: Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version.

Figure 4-1 shows the Accelerator Adapter core Customize IP with each option split into tabs. The Basic tab contains most of the configuration options.

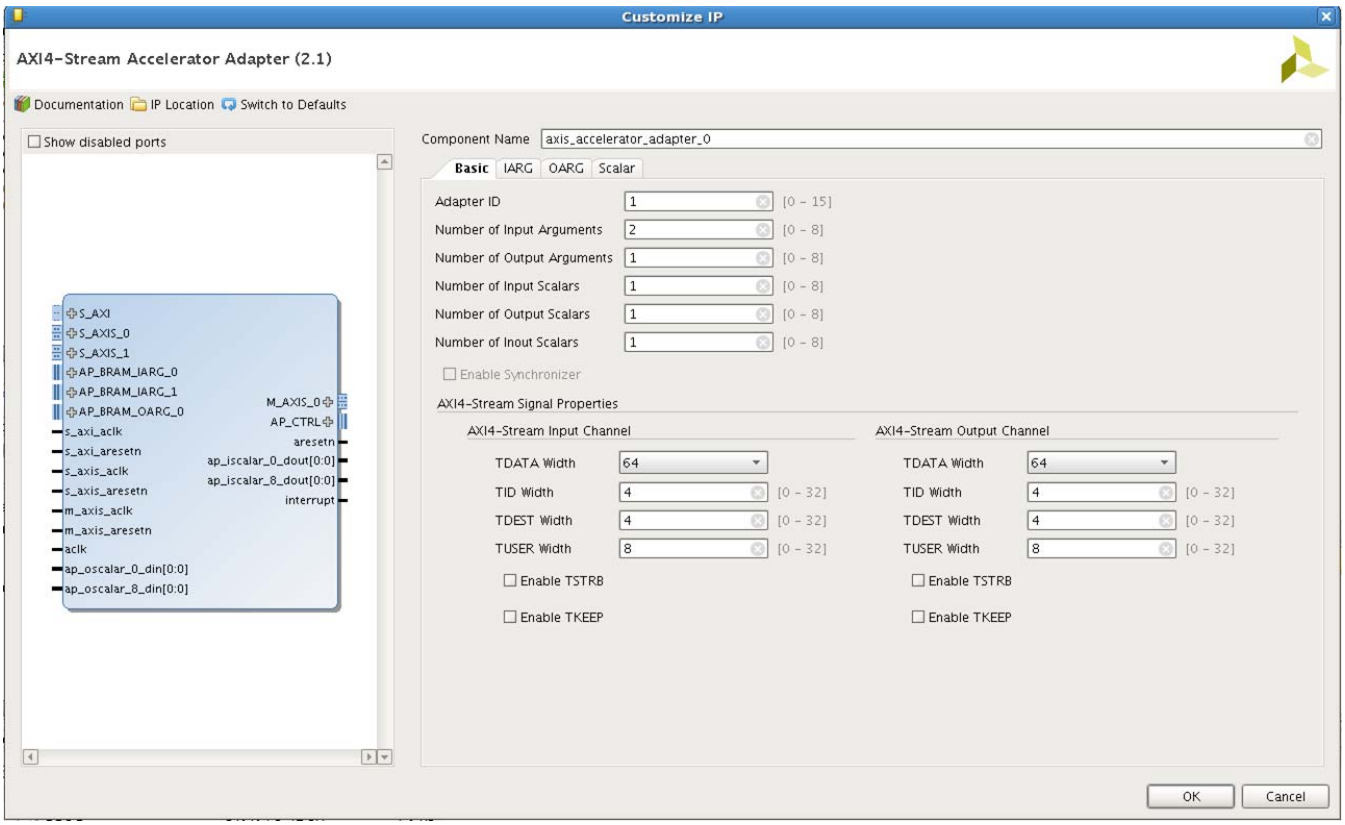

Figure 4-1: **Vivado Customize IP Dialog Box Basic Tab**

Figure 4-2 shows the Input Argument setting. The number of rows are added as per the number of input arguments configured in the Basic tab.

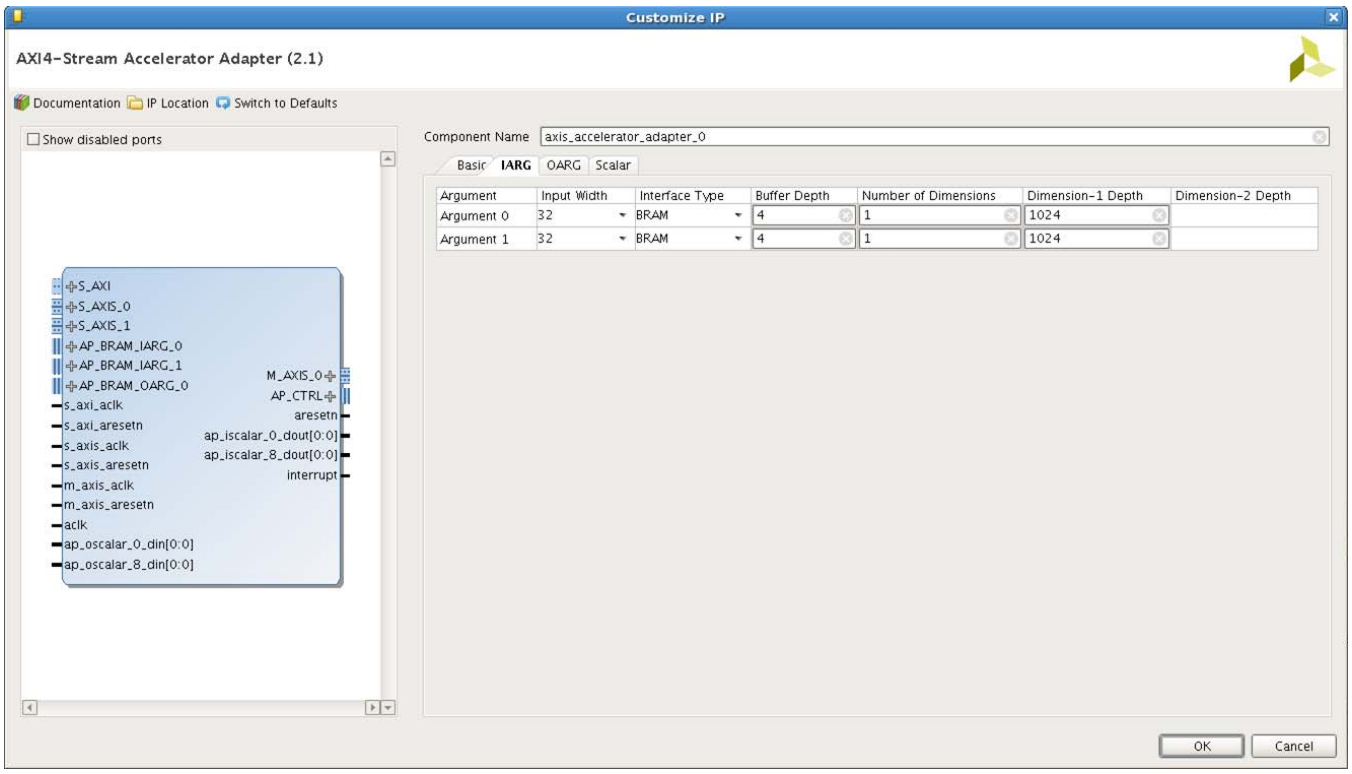

Figure 4-2: **Input Argument Setting**

Figure 4-3 shows the Output Argument setting. The number of rows are added as per the number of output arguments configured in the Basic tab.

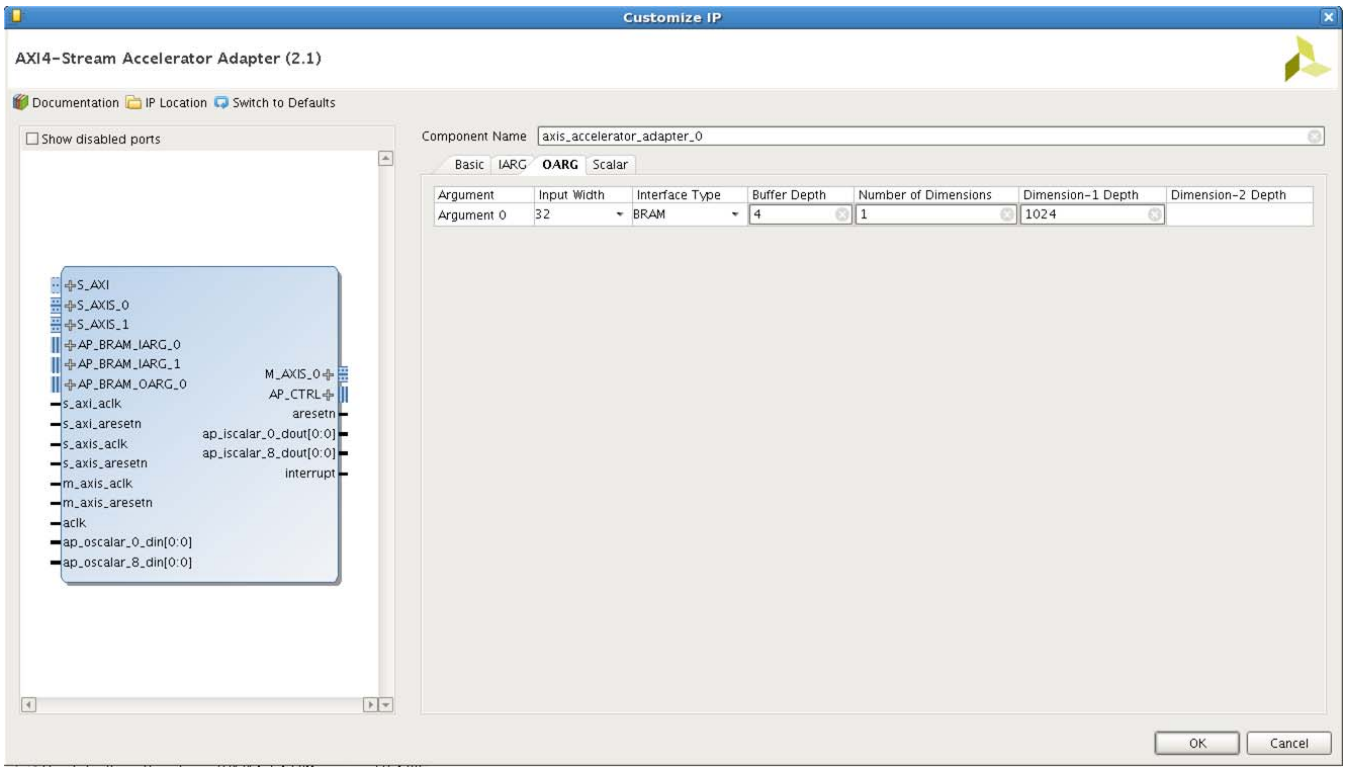

Figure 4-3: **Output Argument Setting**

Figure 4-4 shows the Input, Output, and Inout Scalar setting based on the number of scalars configured for each in the Basic tab.

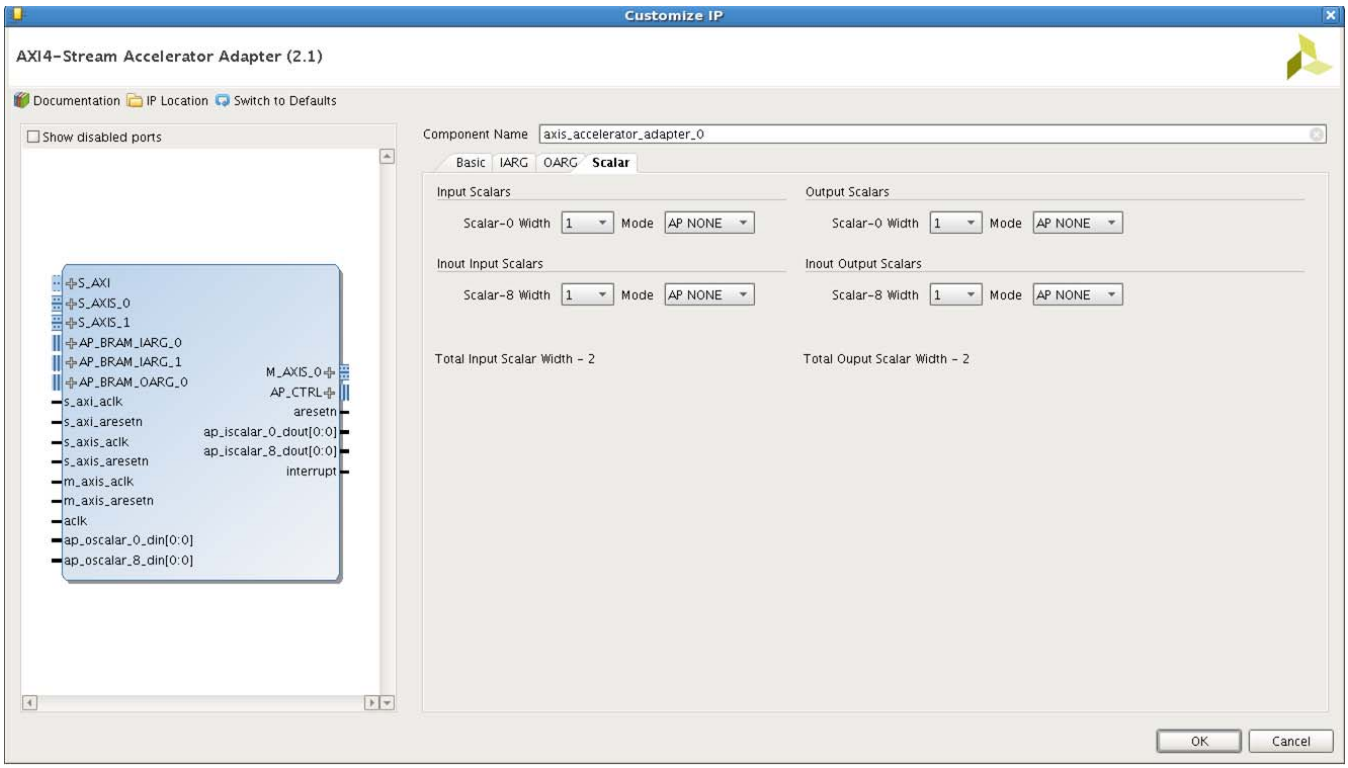

Figure 4-4: **Input, Output, and Inout Scalar Setting**

Figure 4-5 shows the Accelerator Adapter core Customize IP with IP integrator.

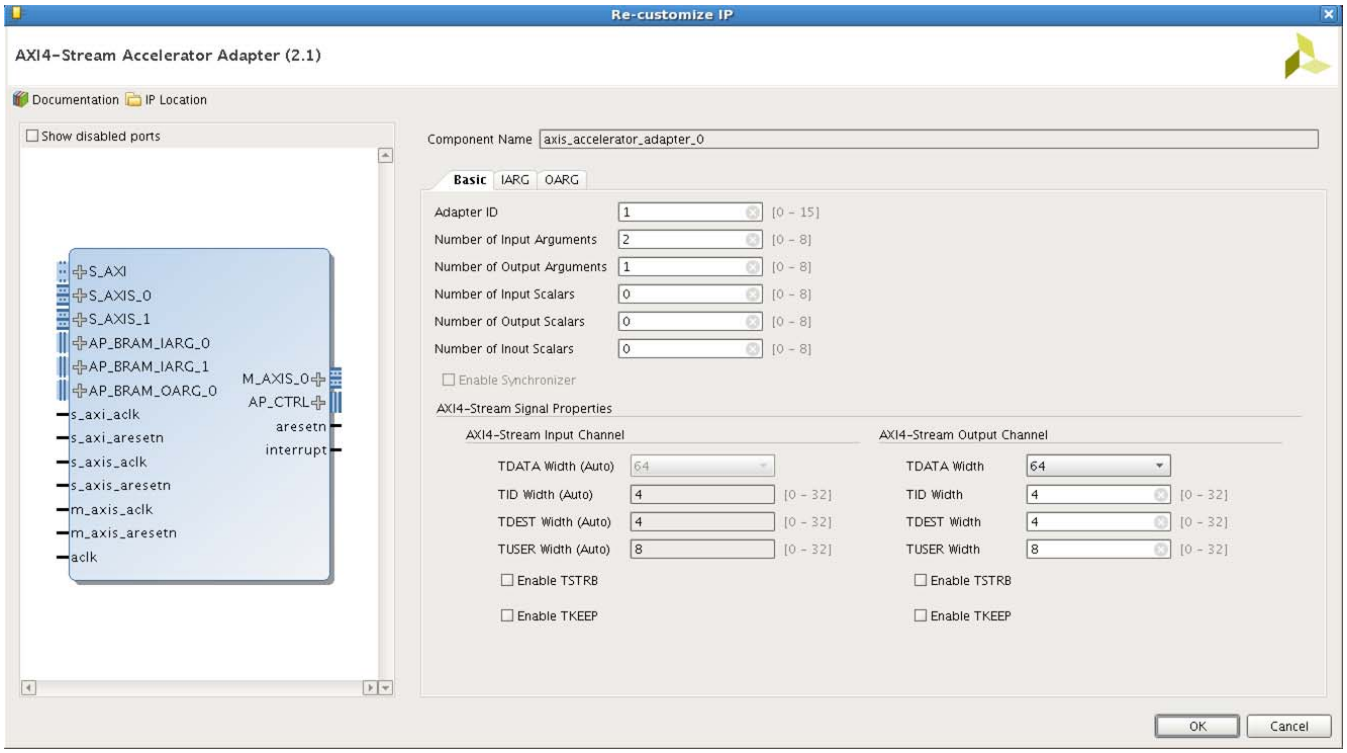

Figure 4-5: **Vivado IP Integrator**

The following parameters are set automatically in IP integrator block design and not available for configuration.

- AXI4-Stream Input Channel
- TDATA Width
- TID Width
- TDEST Width
- TUSER Width
- Enable TSTRB

IMPORTANT: *The Enable Synchronizer parameter is disabled in the current version.*

- **Component Name** – The base name of the output files generated for the core. Names must begin with a letter and can be composed of any of the following characters: a to z, 0 to 9, and "_".

Basic Options

This section describes the Vivado IDE options for the Accelerator Adapter core.

- **Adapter ID** – Static parameter to hold your configured ID for identification.
- **Number of Input Arguments** – This integer value specifies number of input argument to HLS accelerator. This selection enables AXI4-Stream slave interface and block RAM Slave interface or FIFO Master interface in Accelerator Adapter core. Supported number of input arguments is 0 to 8.
- **Number of Output Arguments** – This integer value specifies number of output argument from HLS accelerator. This selection enables AXI4-Stream Master interface and block RAM or FIFO Slave interface in Accelerator Adapter core. Supported number of output arguments is 0 to 8.
- **Number of Input Scalars** – This integer value specifies number of input scalars to HLS accelerator. Each input scalar can have different data width and can be configured with different I/O protocol. Valid values for number of input scalars are 0 to 8.
- **Number of Output Scalars** – This integer value specifies number of output scalars from HLS accelerator. Each output scalar can have different data width and can be configured with different I/O protocol. Valid values for number of output scalars are 0 to 8.
- **Number of Inout Scalars** – This integer value specifies number of inout scalars from/to HLS accelerator. Each inout scalar can have different data width and can be configured with different I/O protocol. Valid values for number of inout scalars are 0 to 8.
- **Enable Synchronizer** – This setting is disabled in the current version and is supported in future releases.

AXI4-Stream Channel

This section describes the Vivado IDE options for the AXI4-Stream Slave or Master interface of the Accelerator Adapter core.

- **TDATA Width** – AXI4-Stream interface data width in bits. Valid values are 8, 16, 32, 64, 128, and 256.
- **TDEST Width** – AXI4-Stream interface `tdest` signal width in bits. Valid values are 0 to 32. Setting value of 0 disables the port.
- **TID Width** – AXI4-Stream interface `tid` signal width in bits. Valid values are 0 to 32. Setting value of 0 disables the port.
- **TUSER Width** – AXI4-Stream interface `tdest` signal width in bits. Valid values are 0 to 32. Setting value of 0 disables the port.

- **Enable TSTRB** – Checking this option enables `tstrb` ports on AXI4-Stream Master and Slave interfaces.
- **Enable TKEEP** – Checking this option enables `tkeep` ports on AXI4-Stream Master and Slave interfaces.

Input/Output Argument Configuration

This section describes the Vivado IDE options for the input and output arguments of the Accelerator Adapter core.

- **Argument Width** – This integer value specifies argument width in bits for block RAM or FIFO interface. Valid values are 8, 16, 32, and 64.
- **Interface Type** – This setting allows you to set block RAM or FIFO interface towards accelerator.
- **Buffer Depth** – This integer value specifies number of multi-buffers for input and output arguments. Valid values are 1 to 4. To have better performance, specify the minimum value as 2.
- **Number of Dimensions** – This integer value specifies number dimension available for block RAM interface. This setting is available only for block RAM interface.
- **Dimension Depth** – This integer value specifies depth of each multi-buffer for block RAM interface. For FIFO interface, this field specifies the FIFO depth.
- **Scalar Width** – This integer value specifies width of scalar interface. Valid values are 1 to 32.
- **Scalar Mode** – Specifies the I/O protocol for each scalar port to match with the HLS accelerator scalar I/O protocol. Supported I/O interfaces are `ap_none`, `ap_vld`, and `ap_hs`.

User Parameters

Table 4-1 shows the relationship between the GUI fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value[(1)]	User Parameter/Value[(1)]	Default Value[(1)]
Adapter ID	C_AP_ADAPTER_ID	1
Number of Input Arguments	C_N_INPUT_ARGS	2
Number of Output Arguments	C_N_OUTPUT_ARGS	1
Number of Input Scalars	C_N_INPUT_SCALARS	0
Number of Output Scalars	C_N_OUTPUT_SCALARS	0
Number of Inout Scalars	C_N_INOUT_SCALARS	0

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value[1]
Enable Synchronizer	C_PRMRY_IS_ACLK_ASYNC	0
TDATA Width	C_S_AXIS_TDATA_WIDTH	64
TID Width	C_S_AXIS_TID_WIDTH	4
TDEST Width	C_S_AXIS_TDEST_WIDTH	4
TUSER Width	C_S_AXIS_TUSER_WIDTH	8
Enable TSTRB	C_S_AXIS_HAS_TSTRB	FALSE
Enable TKEEP	C_S_AXIS_HAS_TKEEP	FALSE
TDATA Width	C_M_AXIS_TDATA_WIDTH	64
TID Width	C_M_AXIS_TID_WIDTH	4
TDEST Width	C_M_AXIS_TDEST_WIDTH	4
TUSER Width	C_M_AXIS_TUSER_WIDTH	8
Enable TSTRB	C_M_AXIS_HAS_TSTRB	FALSE
Enable TKEEP	C_M_AXIS_HAS_TKEEP	FALSE
Input Argument-<0-7> Width	C_AP_IARG_N_DWIDTH	32
Interface Type	C_AP_IARG_N_TYPE	BRAM
Buffer Depth	C_AP_IARG_N_MB_DEPTH	4
Number of Dimensions	C_AP_IARG_O_N_DIM	1
Dimension-1 Depth	C_AP_IARG_O_DIM_1	1,024
Dimension-2 Depth	C_AP_IARG_O_DIM_2	1
Output Argument-<0-7> Width	C_AP_OARG_N_DWIDTH	32
Interface Type	C_AP_OARG_N_TYPE	BRAM
Buffer Depth	C_AP_OARG_N_MB_DEPTH	4
Number of Dimensions	C_AP_OARG_O_N_DIM	1
Dimension-1/Dimension-2 Depth	C_AP_OARG_O_DIM_<1-2>	1,024
Scalar-0 Width	C_INPUT_SCALAR_N_WIDTH	1
Mode	C_INPUT_SCALAR_N_MODE	AP HS
Scalar-8 Width	C_INPUT_SCALAR_<8-15>_WIDTH	1
Mode	C_INPUT_SCALAR_N_MODE	AP None
Scalar-0 Width	C_AP_OSCALAR_N_WIDTH	1
Mode	C_OUTPUT_SCALAR_N_MODE	AP None
Scalar-8 Width	C_OUTPUT_SCALAR_<8-15>_WIDTH	AP None
Mode	C_OUTPUT_SCALAR_N_MODE	AP None
Total Input Scalar Width	N/A	

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value[1]
Total Output Scalar Width	N/A	

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

In synchronous mode, all clocks run at the same frequency and are derived from the same source. There are no multicycle or FALSE paths in this design. No constraint is generated in this case.

These constraints are generated based on core configuration such as Number of input/output argument and argument types. The constraints are generated in `core.xdc` and `core_clock.xdc` files in the generated output products.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5].

IMPORTANT: *For cores targeting 7 series or Zynq-7000 AP SoC devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1].

Example Design

There is no example design for this IP core release.

Test Bench

There is no test bench for this IP core release.

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information on migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 6].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Debugging

This appendix includes details about resources available on the Xilinx® Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the AXI Accelerator Adapter, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the AXI Accelerator Adapter. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

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 this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the AXI Accelerator Adapter

AR: 57586

Technical Support

Xilinx provides technical support in 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.

Debug Tools

There are many tools available to address AXI Accelerator Adapter design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 7].

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. Vivado Lab Edition is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using Vivado Lab Edition for debugging the specific problems.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `locked` port.
- If your outputs go to 0, check your licensing.

Core-Specific Checks

Perform the following checks for further debugging process.

- Check if all streaming input and output arguments have proper clock and reset connection.
- Check the software configuration as described in Programming Sequence for Accelerator Adapter, page 34.

Interface Debug

AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. Output `s_axi_arready` asserts when the core is ready to receive address, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `s_axi_aclk and aclk` inputs are connected and toggling.
- The interface is not being held in reset, and `s_axi_areset` is an active-Low reset.
- The interface is enabled, and `s_axi_aclken` is active-High (if used).
- The main core clocks are toggling and that the enables are also asserted.
- If the simulation has been run, verify in simulation and/or a Vivado Lab Edition capture that the waveform is correct for accessing the AXI4-Lite interface.

AXI4-Stream Interfaces

If data is not being transmitted or received, check the following conditions:

- If transmit `m_axis_n_tready` is stuck Low following the `m_axis_n_tvalid` input being asserted, the core cannot send data.
- If the receive `m_axis_n_tvalid` is stuck Low, the core is not sending data.
- Check if `m_axis_n_aclk` input is connected and toggling.
- Check if `m_axis_n_aresetn` has proper connection.

Appendix C

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. *Vivado® Design Suite User Guide: Designing with IP* (UG896)
2. *Vivado AXI Reference Guide* (UG1037)
3. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
4. *Vivado Design Suite User Guide: Getting Started* (UG910)
5. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
6. *ISE® to Vivado Design Suite Migration Guide* (UG911)
7. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
8. *AMBA® AXI4-Stream Protocol Specification*
9. *Vivado Design Suite User Guide: Implementation* (UG904)
10. *Vivado Design Suite User Guide: High-Level Synthesis* (UG902)
11. *Vivado Design Suite Tutorial: High-Level Synthesis* (UG871)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/18/2015	2.1	Added support for UltraScale+ families.
04/01/2015	2.1	• Updated clock description in Latency section. • Updated Table 2-2: Device Utilization – Virtex-7 FPGAs. • Updated Asynchronous Clock Domain section. • Updated Bit[5] for Input Scalar Status Register (0x0180 to 0x019C) to Inout Scalar Status Register (0x01E0 to 0x01FC). • Added UNISIM important note in Simulation section.
10/01/2014	2.1	• Asynchronous clocking removed. • Updated Customizing and Generating the Core section. • Updated Figs. 4-1 to 4-5. • Vivado IDE update and User Parameter section for revision change.
04/02/2014	2.1	Initial Xilinx release.

Please Read: Important Legal Notices