AXI4 to AXI4-Lite Bridge (Beta Release)

Version 1.0

May 3, 2023

Contents

IP Summary

Introduction

An AXI to AXI Lite bridge is a type of interface that allows communication between a master that uses the AXI protocol and a slave that uses the AXI Lite protocol. The bridge acts as a translator, converting the signals and commands from one protocol to the other, allowing the two devices to communicate with each other. This is typically used in system designs where different IP blocks may use different protocols for communication. This AXI4 compliant bridge can be used in a number of IPs that support either AXI4 or AXILite interface.

Features

- Configurable Data Width for the AXI Master and Slave interfaces.
- Configurable Address Width for AXI Master and Slave interfaces.
- Configurable ID Width for AXI Slave interface.
- Conversion from AXI to AXI-lite with no performance loss.
- Maintains the one-clock per transition speed of AXI.
- Buffer to handle bursts that can maintain full speed of the AXI.

Overview

AXI4 to AXI4-Lite Bridge

An AXI to AXI-Lite bridge serves as a communication link between two different bus protocols in a system-on-chip (SoC) design. The AXI protocol is designed for high-performance and high-bandwidth communication between devices like processors, memory controllers, and high-speed peripherals. In contrast, the AXI-Lite protocol is a simplified version of AXI and is used for connecting low-bandwidth peripherals and control registers. The AXI to AXI-Lite bridge acts as a mediator, enabling communication between the two protocols. It receives AXI transactions from the high-speed bus and converts them into AXI-Lite transactions that can be understood by the low-bandwidth bus. Similarly, it receives AXI-Lite transactions from the low-bandwidth bus and converts them into AXI transactions that can be understood by the high-speed bus. Aside from protocol translation, the AXI to AXI-Lite bridge also performs additional functions like address mapping, data buffering, and protocol optimization to ensure efficient and reliable data transfer between the different bus types. The AXI to AXI-Lite bridge is an essential component in a system-on-chip design or an FPGA that facilitates smooth integration of high-speed and low-speed peripherals. A block diagram for the AXI to AXI-Lite Bridge IP is shown in Figure 1.

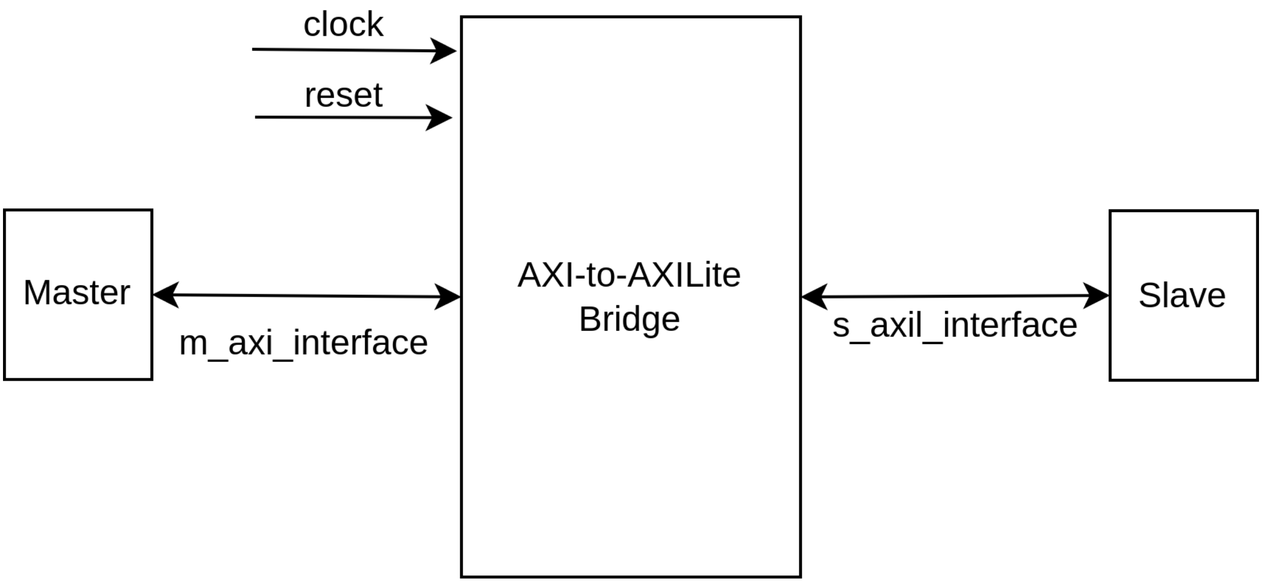

Figure 1: AXI4 to AXI4-Lite Bridge Block Diagram

IP Specification

The AXI to AXI-Lite bridge is a hardware component that facilitates communication between devices operating with the AXI protocol and those operating with the AXI-Lite protocol in a system-on-chip (SoC) design. The bridge acts as a mediator and translator, enabling seamless integration of high-speed and low-speed peripherals in an SoC. The working of the bridge is detailed as below:

- **Protocol Translation**
  The AXI to AXI-Lite bridge supports protocol translation from the AXI protocol to the AXI-Lite protocol and vice versa. It is capable of receiving AXI transactions from the high-speed bus and converting them into AXI-Lite transactions that can be understood by the low-speed bus. Similarly, it is able to receive AXI-Lite transactions from the low-speed bus and convert them into AXI transactions that can be understood by the high-speed bus.
- **Address Mapping**
  The AXI to AXI-Lite bridge supports address mapping, enabling it to map AXI addresses to AXI-Lite addresses and vice versa. This is necessary because the AXI and AXI-Lite protocols have different address widths, and mapping the addresses enables smooth communication between the devices.
- **Data Buffering**
  The AXI to AXI-Lite bridge supports data buffering to avoid data loss due to congestion. When the high-speed bus is sending data to the low-speed bus, the bridge should buffer the data and release it to the low-speed bus at a rate that the bus can handle. Similarly, when the low-speed bus is sending data to the high-speed bus, the bridge buffers the data and release it at a rate that the high-speed bus can handle.
- **Protocol Optimization**
  The AXI to AXI-Lite bridge supports protocol optimization to ensure efficient and reliable data transfer between the two buses. The bridge optimizes the AXI transactions and AXI-Lite transactions to minimize latency and maximize throughput.
- **Interrupt Handling**
  The AXI to AXI-Lite bridge supports interrupt handling, enabling it to generate interrupts to the low-speed bus when events occur on the high-speed bus. The bridge is capable of generating interrupts based on pre-defined criteria and provide a mechanism for the low-speed bus to acknowledge and clear the interrupts.

The internal block diagram can be seen in Figure 2.

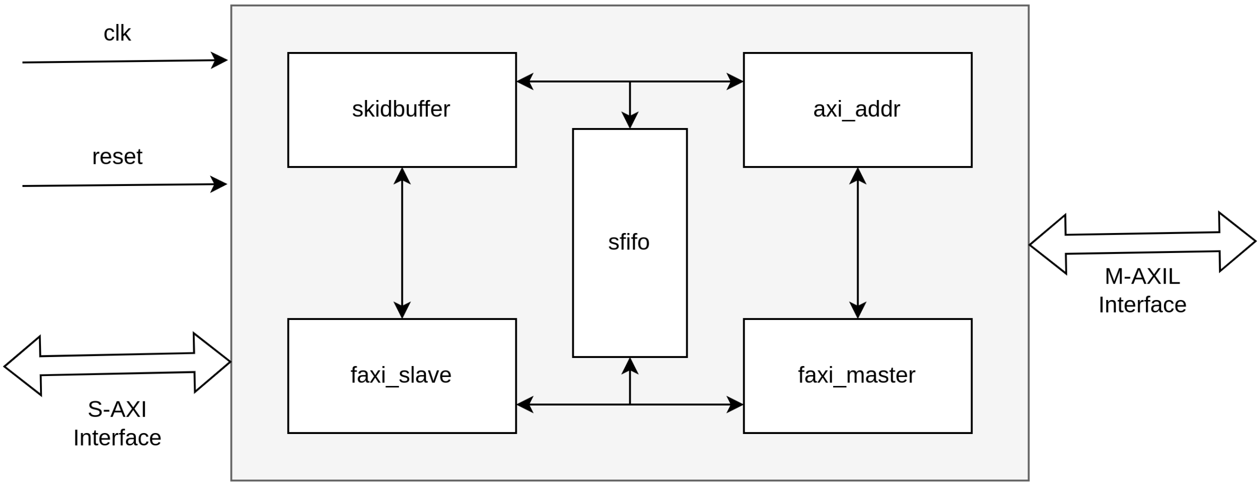

Figure 2: AXI to AXI-Lite Bridge Internal Diagram

Brief details of the internal modules is given below:

- **skidbuffer**
  Skid buffers are required for high throughput AXI code, since the AXI specification requires that all outputs be registered. This means that, if there are any stall conditions calculated, it will take a clock cycle before the stall can be propagated up stream. This means that the data will need to be buffered for a cycle until the stall signal can make it to the output. Handling that buffer is the purpose of this core. On one end of this core, you have the i_valid and i_data inputs to connect to your bus interface. There's also a registered o_ready signal to signal stalls for the bus interface. The other end of the core has the same basic interface, but it isn't registered. This allows you to interact with the bus interfaces as though they were combinatorial logic, by interacting with this half of the core. If at any time the incoming !stall signal, i_ready, signals a stall, the incoming data is placed into a buffer. Internally, that buffer is held in r_data with the r_valid flag used to indicate that valid data is within it.

- **axi_addr**
  The AXI (full) standard has some rather complicated addressing modes, where the address can either be FIXED, INCRementing, or even where it can WRAP around some boundary. When in either INCR or WRAP modes, the next address must always be aligned. In WRAP mode, the next address calculation needs to wrap around a given value, and that value is dependent upon the burst size (i.e. bytes per beat) and length (total numbers of beats). Since this calculation can be non-trivial, and since it needs to be done multiple times, the logic below captures it for every time it might be needed.

- **sfifo**
  It is just a synchronous data FIFO to synchronize the transactions with the clock.

- **faxil_slave**
  This module defines the formal properties of an AXI lite slave to describe the transactions behaviour for the AXI lite slave.

- **faxil_master**
  This module defines the formal properties of an AXI lite master to describe the transactions behaviour for the AXI lite master.

Standards

The AXI4 and AXI4-Lite interfaces are compliant with the AMBA® AXI Protocol Specifications.

IP Support Details

The Table 1 gives the support details for AXI4 to AXI4-Lite Bridge.

Compliance		IP Resources					Tool Flow		
Device	Interface	Source Files	Constraint File	Testbench	Simulation Model	Software Driver	Analyze and Elaboration	Simulation	Synthesis
GEMINI	AXI4/AXI4-Lite	Verilog	SDC	Python	Cocotb	Icarus	Raptor	Raptor	Raptor

Table 1: IP Details

Parameters

Table 2 lists the parameters of the AXI4 to AXI4-Lite Bridge.

Parameter	Values	Default Value	Description
DATA WIDTH	8, 16, 32, 64, 128, 256	32	Bridge Data Width
ADDR WIDTH	6 - 32	16	Bridge Address Width
ID WIDTH	1 - 32	2	Bridge ID Width

Table 2: Parameters

Resource Utilization

The parameters for computing the maximum and minimum resource utilization are given in Table 3, remaining parameters have been kept at their default values.

<table>
<tr><th>Tool</th><td colspan="4">Raptor Design Suite</td></tr>
<tr><th>FPGA Device</th><td colspan="4">GEMINI</td></tr>
<tr><th colspan="3">Configuration</th><th colspan="2">Resource Utilized</th></tr>
<tr><td rowspan="4">Minimum Resource</td><th>Options</th><th>Configuration</th><th>Resources</th><th>Utilized</th></tr>
<tr><td>DATA WIDTH</td><td>8</td><td rowspan="3">LUTs
Registers</td><td rowspan="3">417
487</td></tr>
<tr><td>ADDR WIDTH</td><td>6</td></tr>
<tr><td>ID WIDTH</td><td>1</td></tr>
<tr><td rowspan="4">Maximum Resource</td><th>Options</th><th>Configuration</th><th>Resources</th><th>Utilized</th></tr>
<tr><td>DATA WIDTH</td><td>256</td><td rowspan="3">LUTs
Registers
Adder Carry</td><td rowspan="3">2022
3110
26</td></tr>
<tr><td>ADDR WIDTH</td><td>32</td></tr>
<tr><td>ID WIDTH</td><td>32</td></tr>
</table>

Table 3: Resource Utilization

Port List

Table 4 lists the top interface ports of the AXI4 to AXI4-Lite Bridge.

Signal Name	I/O	Description
Clock and Reset		
s_axi_aclk	I	System Clock
s_axi_aresetn	I	Active Low Reset
Slave AXI Write Address Channel		
s_axi_awaddr	I	AXI4 write address
s_axi_awprot	I	AXI4 write address protection data qualifier
s_axi_awvalid	I	AXI4 valid write address
s_axi_awready	O	AXI4 write address ready
s_axi_awburst	I	AXI4 write address burst mode
s_axi_awlen	I	AXI4 write address burst length
s_axi_awsize	I	AXI4 write address burst size
s_axi_awlock	I	AXI4 write address lock qualifier
s_axi_awcache	I	AXI4 write address cache qualifier
s_axi_awregion	I	AXI4 write address region
s_axi_awid	I	AXI4 write address ID
s_axi_awuser	I	AXI4 write address user qualifier
s_axi_awqos	I	AXI4 write address quality of service qualifier
Slave AXI Write Data Channel		
s_axi_wdata	I	AXI4 write data
s_axi_wstrb	I	AXI4 write strobe
s_axi_wvalid	I	AXI4 data valid
s_axi_wready	O	AXI4 data ready
s_axi_wlast	I	AXI4 write last qualifier
s_axi_wuser	I	AXI4 write user qualifier
Slave AXI Write Response Channel		
s_axi_bresp	O	AXI4 write response
s_axi_bvalid	O	AXI4 write valid response
s_axi_bready	I	AXI4 write ready response
s_axi_bid	O	AXI4 write response ID
s_axi_buser	O	AXI4 write response user qualifier
Slave AXI Read Address Channel		
s_axi_araddr	I	AXI4 read address
s_axi_arprot	I	AXI4 read address protection data qualifier
s_axi_arvalid	I	AXI4 read address valid
s_axi_arready	O	AXI4 read address ready
s_axi_arburst	I	AXI4 read address burst mode
s_axi_arlen	I	AXI4 read address burst length
s_axi_arsize	I	AXI4 read address burst size
s_axi_arlock	I	AXI4 read address lock qualifier

s_axi_arcache	I	AXI4 read address cache qualifier
s_axi_arqos	I	AXI4 read address quality of service qualifier
s_axi_arregion	I	AXI4 read address region
s_axi_arid	I	AXI4 read address ID qualifier
s_axi_aruser	I	AXI4 read address user qualifier
Slave AXI Read Data Channel		
s_axi_rdata	O	AXI4 read data
s_axi_rresp	O	AXI4 read response
s_axi_rvalid	O	AXI4 read data valid
s_axi_rready	I	AXI4 read data ready
s_axi_rlast	O	AXI4 read last qualifier
s_axi_rid	O	AXI4 read ID qualifier
s_axi_ruser	O	AXI4 read user qualifier
Master AXI-Lite Write Address Channel		
m_axi_awaddr	O	AXI4-Lite write address
m_axi_awprot	O	AXI4-Lite write address protection data qualifier
m_axi_awvalid	O	AXI4-Lite valid write address
m_axi_awready	I	AXI4-Lite write address ready
Master AXI-Lite Write Data Channel		
m_axi_wdata	O	AXI4-Lite write data
m_axi_wstrb	O	AXI4-Lite write strobe
m_axi_wvalid	O	AXI4-Lite data valid
m_axi_wready	I	AXI4-Lite data ready
Master AXI-Lite Write Response Channel		
m_axi_bresp	I	AXI4-Lite write response
m_axi_bvalid	I	AXI4-Lite write valid response
m_axi_bready	O	AXI4-Lite write ready response
Master AXI-Lite Read Address Channel		
m_axi_ardata	O	AXI4-Lite read address
m_axi_arprot	O	AXI4-Lite read address protection qualifier
m_axi_arvalid	O	AXI4-Lite read address valid
m_axi_arready	I	AXI4-Lite read address ready
Master AXI-Lite Read Data Channel		
m_axi_rdata	I	AXI4-Lite read data
m_axi_rresp	I	AXI4-Lite read response
m_axi_rvalid	I	AXI4-Lite read data valid
m_axi_rready	O	AXI4-Lite read data ready

Table 4: AXI4 to AXI4-Lite Bridge Interface

Design Flow

IP Customization and Generation

AXI4 to AXI4-Lite Bridge IP core is a part of the Raptor Design Suite Software. A customized AXI4 to AXI4-Lite Bridge can be generated from the Raptor's IP configurator window as shown in Figure 3.

Figure 3: IP list

Parameters Customization

From the IP configuration window, the parameters of the AXI4 to AXI4-Lite Bridge can be configured and AXI4 to AXI4-Lite Bridge features can be enabled for generating a customized AXI4 to AXI4-Lite Bridge IP core that suits the user application requirement as shown in Figure 4. After IP Customization, all the source files are made available to the user with a top wrapper that instantiates a parameterized instance of the AXI4 to AXI4-Lite Bridge.

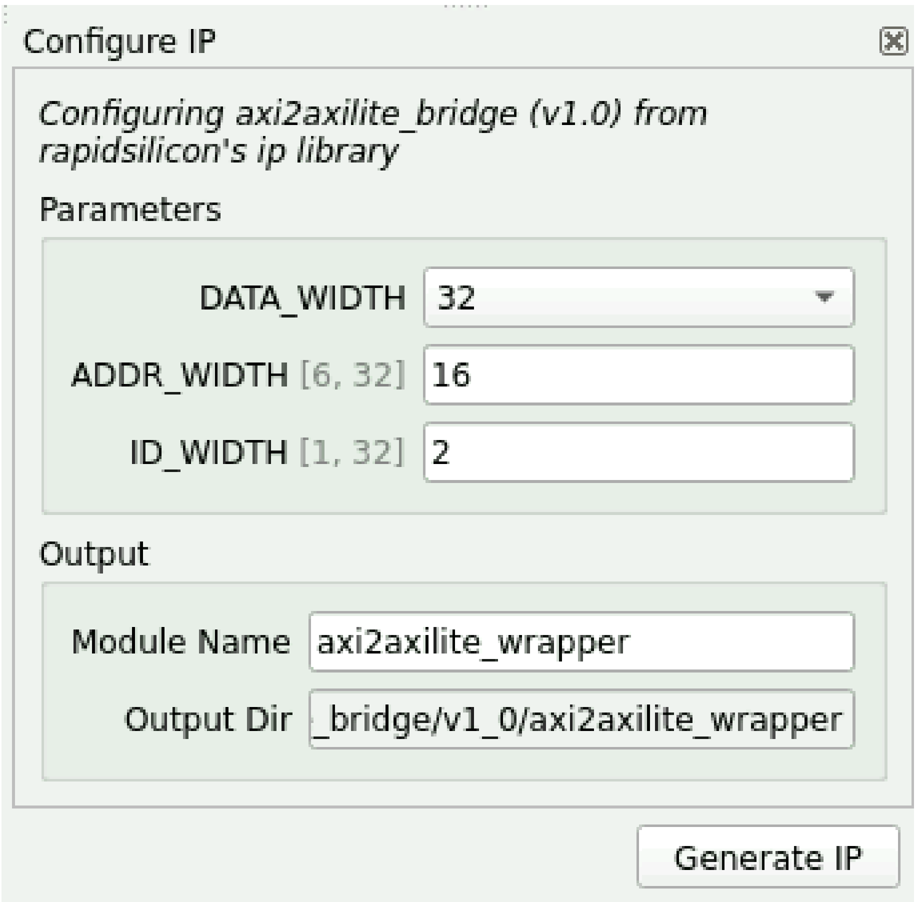

Figure 4: IP Configuration

Example Design

Overview

This AXI to AXI-Lite Bridge IP can be utilized in a system that requires the translation between AXI and AXI-Lite protocols to integrate both high-speed and low-speed IPs together. AXI to AXI-Lite is a crucial component in many electronic systems, enabling communication between different types of peripherals, essentially making an AXI slave connect with a AXI-Lite Master. One such example design of this AXI to AXI-Lite Bridge, where an AXI capable processor is connected with AXI-Lite peripherals via the AXI to AXI-Lite bridge, can be visualized in Figure 5.

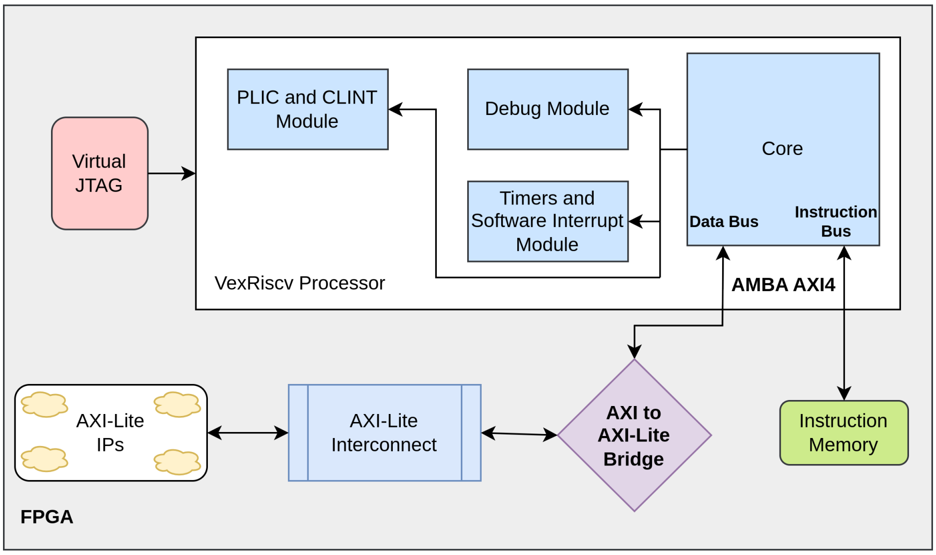

Figure 5: AXI to AXI-Lite Bridge embedded within an FPGA

Simulating the Example Design

The IP being Verilog HDL, can be simulated via a bunch of industry standard stimulus. For instance, it could be simulated via writing a Verilog Test-bench, or incorporating a soft processor that can stimulate this AXI4 to AXI4-Lite Bridge. The bundled example design is stimulated via a Cocotb based environment on Icarus, that iteratively stimulates the soft IP by performing a number of writing and reading operations on this IP on various different addresses to ensure the complete working of the bridge. A stress test is also performed at the end to ensure that the bridge can handle the high speed traffic of the AXI protocol.

Synthesis and PR

Raptor Design Suite is armed with tools for Synthesis along with Post and Route capabilities and the generated post-synthesis and post-route and place net-lists can be viewed and analyzed from within the Raptor. The generated bit-stream can then be uploaded on an FPGA device to be utilized in hardware applications.

Test Bench

The included testbench for the AXI to AXI-Lite Bridge IP is a Cocotb based Python testbench that performs various read and write operations on the IP core to stimulate all the modules of the IP. Python is used to simulate the IP inside the Cocotb library for this purpose and a total of 25 tests are performed, 12 tests perform read operations and the other 12 perform write operations on the IP at various different addresses to make sure that the protocol translation is working as expected. A stress test is also performed at the end of the simulation to make sure that the IP doesn't break under high speed burst conditions of the AXI protocol. The internal modules such as skidbuffer and FIFOs make sure that the AXI-Lite transactions are always in line with the higher spec AXI transactions and vice versa. The waveforms are also dumped in the format of .vcd for in-depth analysis of the whole operation and can be viewed on the integrated waveviewer. Being written in simple Python, the testbenches are easily modifiable to provide maximum coverage of the AXI to AXI-Lite Bridge IP.

Release

Release History

Date	Version	Revisions
May 3, 2023	0.01	Initial version AXI4 to AXI4-Lite Bridge User Guide Document