AXI4-Lite I2C Slave (Beta Release)

Version 0.1

May 4, 2023

Copyright

Trademarks

Disclaimers

Contents

IP Summary

Introduction

An I2C slave device is a device that communicates with an I2C master device over an I2C bus. The slave device does not control the clock signal and can only respond to commands and requests from the master device. Each slave device on the bus has a unique address that the master device uses to communicate with it. The slave device can be a sensor, a display, or other types of peripheral devices. In I2C communication protocol slave device can only respond to the master device, it doesn't initiate the communication. This I2C IP acts as a Slave that is commanded via a Master connected to the AXI-Lite interface making it compatible with other AXI based systems.

Features

- Configurable Data and Address Widths.
- Supports configurable filter length.
- Supports the AXI4-Lite interface specification.
- AXI-Stream based embedded I2C module for maximum throughput.
- Status signals on the output for acknowledging the current state of the I2C IP.
- Independent configuration signals for device specific operations.

Overview

AXIL I2C Slave

An I2C Slave is a device that receives commands and data from an I2C Master and responds accordingly. An I2C Slave Soft IP typically consists of a set of configurable registers that define the device's behavior and response to different commands from the Master. These registers can be programmed by the system designer to customize the Slave's behavior to suit the requirements of the specific application. The IP may also include additional features, such as interrupt support, that can enhance the Slave's functionality. It is used to transfer data between devices over short distances, and is a popular communication protocol in many embedded systems. One of the advantages of using an I2C Slave is its simplicity and low power consumption. It requires only two wires for communication, making it an ideal choice for many embedded systems. Additionally, the I2C protocol supports multiple Slaves on the same bus, allowing multiple devices to communicate with each other without the need for complex routing or switching. A block diagram for the AXI-Lite I2C Slave IP is shown in Figure 1.

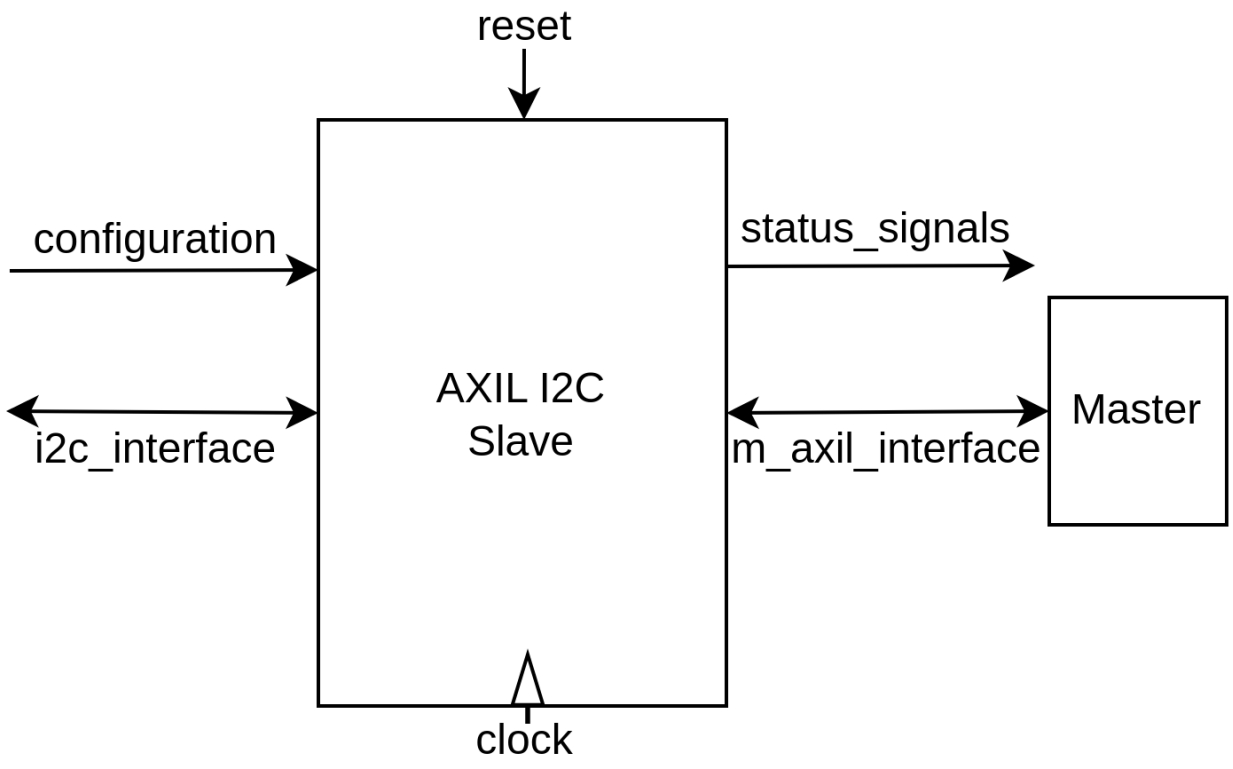

Figure 1: AXIL I2C Slave Block Diagram

IP Specification

The I2C Slave must support the I2C protocol as defined by the I2C specification, including clock synchronization, addressing, and data transfer. The I2C Slave must have a unique address on the I2C bus that can be set or configured by the system designer. The address should be a 7-bit value and must not conflict with any other I2C devices on the same bus. The I2C Slave must have a set of registers that define its behavior and response to different commands from the Master. These registers must be configurable and accessible by the system designer through the I2C bus. The I2C Slave should support interrupts to indicate the completion of a data transfer or the occurrence of an error condition. The Slave must be able to generate an interrupt signal that can be used to wake up the system from a low power state.The I2C Slave must be able to transfer data to and from the I2C Master using either read or write commands. The Slave must support both single-byte and multiple-byte data transfers. The I2C Slave must support clock stretching, which allows the Slave to hold the clock line low to slow down the data transfer rate if it needs more time to process data. The I2C Slave must comply with the timing requirements specified by the I2C specification, including maximum clock frequency and minimum setup and hold times for data and clock signals. The top level embeds an I2C Master that is based on AXI-Strem protocol to ensure high throughput from within the soft IP. The internal block diagram can be seen in Figure 2.

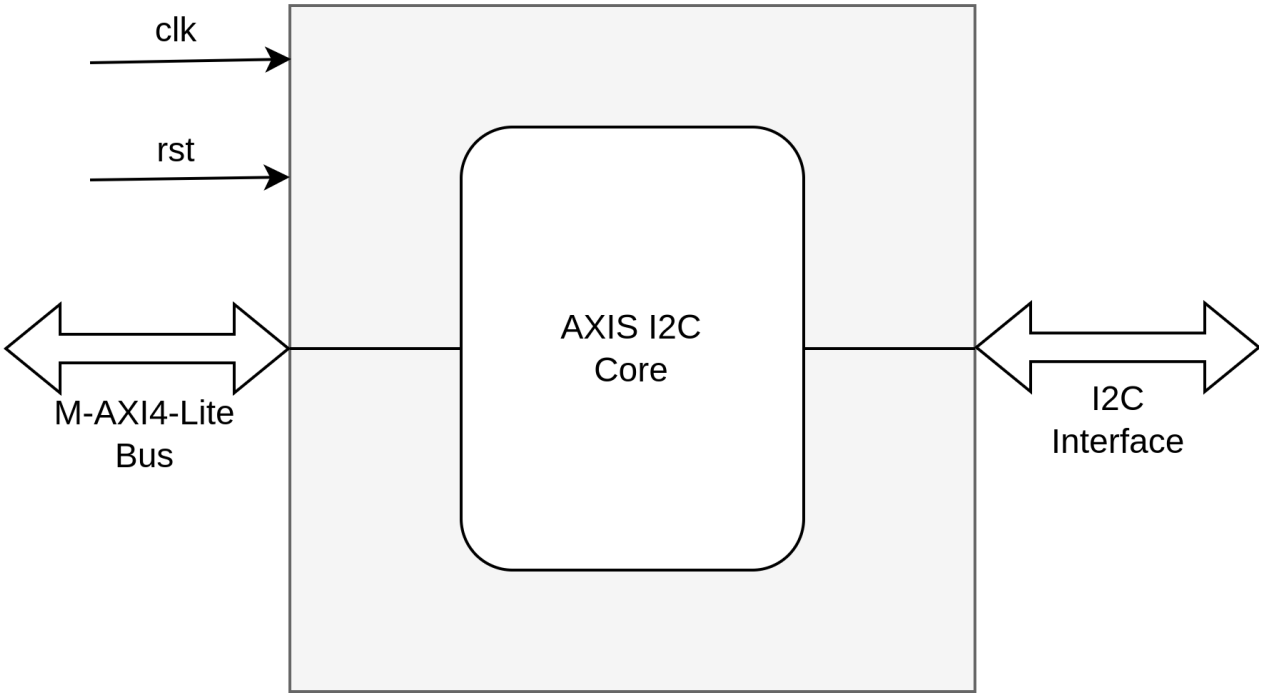

Figure 2: AXIL I2C Slave Internal Diagram

Standards

The AXI4-Lite interface is compliant with the AMBA® AXI Protocol Specification.

IP Support Details

The Table 1 gives the support details for AXIL I2C Slave.

Compliance		IP Resources				Tool Flow		
Device	Interface	Source Files	Constraint Files	Testbench	Simulation Model	Analyze and Elaboration	Simualtion	Synthesis
Gemini	AXI4-Lite	Verilog	-	Python / Verilog	MyHDL	Verific (Raptor)	Icarus (Raptor)	Raptor

Table 1: IP Details

Parameters

Table 2 lists the parameters of the AXIL I2C Slave.

Parameter	Values	Default Value	Description
DATA WIDTH	32 / 64	32	AXI Data Width
ADDR WIDTH	8 / 16 / 32	16	AXI Address Width
FILTER LEN	1 - 4	4	I2C Slave Filter Length

Table 2: Parameters

Port List

Table 3 lists the top interface ports of the AXIL I2C Slave.

Signal Name	I/O	Description
AXI Clock and Reset		
clk	I	System Clock
rst	I	Active High Reset
Write Address Channel		
m_axil_awaddr	I	AXI4-Lite write address
m_axil_awprot	I	AXI4-Lite protection data qualifier
m_axil_awvalid	I	AXI4-Lite valid write address
m_axil_awready	O	AXI4-Lite write address ready
Write Data Channel		
m_axil_wdata	I	AXI4-Lite data
m_axil_wstrb	I	AXI4-Lite data stream identifier
m_axil_wvalid	I	AXI4-Lite data valid
m_axil_wready	O	AXI4-Lite data ready
Write Response Channel		
m_axil_bresp	O	AXI4-Lite transfer response
m_axil_bvalid	O	AXI4-Lite transfer valid response
m_axil_bready	I	AXI4-Lite transfer ready response

Read Address Channel		
m_axil_araddr	I	AXI4-Lite read address
m_axil_arprot	I	AXI4-Lite protection data qualifier
m_axil_arvalid	I	AXI4-Lite read address valid
m_axil_arready	O	AXI4-Lite read address ready
Read Data Channel		
m_axil_rdata	O	AXI4-Lite read data
m_axil_rresp	O	AXI4-Lite read response
m_axil_rvalid	O	AXI4-Lite read data valid
m_axil_rready	I	AXI4-Lite read data ready
I2C Interface		
i2c_scl_i	I	Serial Clock Input
i2c_scl_o	O	Serial Clock Output
i2c_scl_t	O	Serial Clock for interfacing tristate pins
i2c_sda_i	I	Serial Data Input
i2c_sda_o	O	Serial Data Output
i2c_sda_t	O	Serial Data for interfacing tristate pins
Status Signals		
busy	O	Busy Device Signal
bus_addressed	O	Address of the Bus Activated
bus_active	O	Status of Bus Activation
Configuration		
enable	I	Enable bit for Slave
device_address	I	Address assignment for the Slave

Table 3: AXIL I2C Slave Interface

Resource Utilization

The parameters for computing the maximum and minimum resource utilization are given in Table 4, remaining parameters have been kept at their default values.

Tool	Raptor Design Suite			
FPGA Device	GEMINI			
Configuration			**Resource Utilized**	
Minimum Resource	**Options**	**Configuration**	**Resources**	**Utilized**
	DATA WIDTH	32	LUTs Registers	247 110
	ADDR WIDTH	8		
	FILTER LEN	1		
Maximum Resource	**Options**	**Configuration**	**Resources**	**Utilized**
	DATA WIDTH	64	LUTs Registers	393 176
	ADDR WIDTH	32		
	FILTER LEN	4		

Table 4: Resource Utilization

Operation

This module enables I2C control over an AXI lite bus, useful for enabling a design to operate as a peripheral to an external microcontroller or similar. The AXI lite interface is fully parametrizable, with the restriction that the bus must be divided into 2^m words of $8 * 2^n$ bits. Writing via I2C first accesses an internal address register, followed by the actual AXI lite bus. The first k bytes go to the address register, where

$$k = \left\lceil \log_2 \left(\text{ADDR\_WIDTH} + \log_2 \frac{\text{DATA\_WIDTH}}{\text{SELECT\_WIDTH}} \right) / 8 \right\rceil \quad (1)$$

The address pointer will automatically increment with reads and writes. For buses with word size > 8 bits, the address register is in bytes and unaligned writes will be padded with zeros. Writes to the same bus address in the same I2C transaction are coalesced and written either once a complete word is ready or when the I2C transaction terminates with a stop or repeated start.
Reading via the I2C interface immediately starts reading from the AXI lite interface starting from the current value of the internal address register. Like writes, reads are also coalesced when possible. One AXI lite read is performed on the first I2C read. Once that has been completey transferred out, another read will be performed on the start of the next I2C read operation.

Read and Write Transactions

A typical I2C transaction can be broken down into the following points:

- **Start Bit:** To start a transmission, the controller device sets the clock line (SCL) to a high state and pulls the data line (SDA) to a low state. This signals all peripheral

devices that a transmission is about to begin. In cases where multiple controllers try to access the bus simultaneously, the device that pulls the SDA line low first gains control of the bus.

- **Address Frame:** Every new communication sequence starts with the address frame, which includes a 7-bit address transmitted with the most significant bit (MSB) first, followed by a read/write (R/W) bit that specifies the operation type. Additionally, all frames (data or address) have a 9th bit called the NACK/ACK bit. After the first 8 bits of the frame are sent, the receiving device takes control of the data line (SDA). If the receiving device does not respond by pulling the SDA line low before the 9th clock pulse, it means that the device either did not receive the data or could not parse the message. At this point, the exchange stops, and the system controller decides how to proceed.
- **Data Frame:** Once the address frame has been sent, data transmission can begin. The controller device will send clock pulses at regular intervals, and the data will be transmitted through the data line (SDA) either by the controller or the peripheral device, depending on whether the R/W bit specified a read or write operation. The number of data frames that follow is not predetermined, and many peripheral devices have the capability to auto-increment the internal register, which means that subsequent reads or writes will be performed on the next register in sequence.
- **Repeated Start Condition:** At times, it's necessary for a controller device to send multiple messages without any interference from other controllers on the bus. To address this requirement, a repeated start condition has been defined. To initiate a repeated start, the data line (SDA) is allowed to go high while the clock line (SCL) is low, and then SCL is allowed to go high. Next, SDA is brought low again while SCL is high. Since there was no stop condition on the bus, the previous communication is not considered complete, and the current controller device retains control of the bus. After that, the next message can be sent, and the format of this new message is the same as any other message, consisting of an address frame followed by data frames. Multiple repeated starts are possible, and the controller device will retain control of the bus until it generates a stop condition.
- **Stop Bit:** After all the data frames have been transmitted, the controller device will generate a stop condition, which is defined as a transition from low to high on the data line (SDA) followed by a high state on the clock line (SCL), with SCL remaining high. It's important to note that during normal data writing operations, the value on SDA should remain constant when SCL is high to prevent any false stop conditions.

The Figure 3 shows a complete read cycle from the I2C Slave IP.

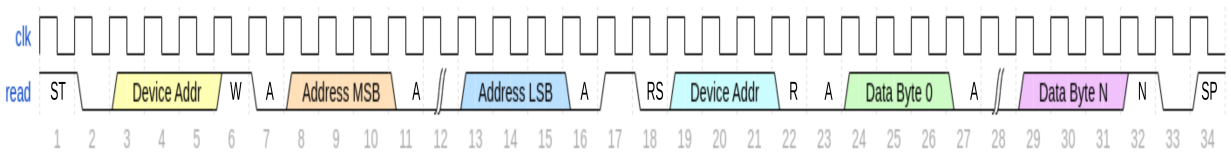

Figure 3: Read Cycle

The Figure 4 shows a complete write cycle from the I2C Slave IP.

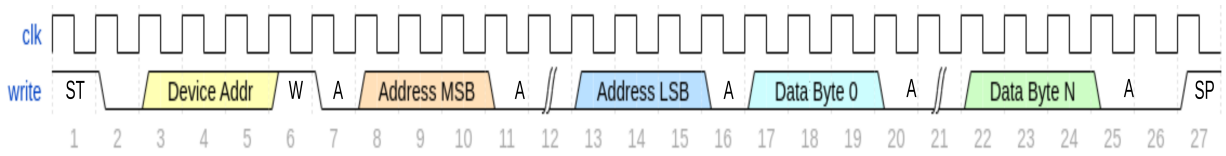

Figure 4: Write Cycle

The list of abbreviations used in the Figures 3 and 4 is given in Table 5.

Abbreviation	Definition
ST	Start Bit
SP	Stop Bit
W	Write
A	Acknowledge
R	Read
RS	Repeated Start
N	NACK/Not Acknowledged

Table 5: List of Abbreviations

Status Bits

- **busy:** module is communicating over the bus
- **bus_control:** module has control of bus in active state
- **bus_active:** bus is active, not necessarily controlled by this module

Control Parameters

- **device_address:** address of slave device

Interfacing the I2C Bus

- This will work for any tristate bus.

```
assign scl_i = scl_pin;
assign scl_pin = scl_t ? 1'bz : scl_o;
assign sda_i = sda_pin;
assign sda_pin = sda_t ? 1'bz : sda_o;
```

- Equivalent code that does not use *_t connections, we can get away with this because I2C is open-drain.

```
assign scl_i = scl_pin;
assign scl_pin = scl_o ? 1'bz : 1'b0;
assign sda_i = sda_pin;
assign sda_pin = sda_o ? 1'bz : 1'b0;
```

- Example of two interconnected I2C devices:

```
assign scl_1_i = scl_1_o & scl_2_o;
assign scl_2_i = scl_1_o & scl_2_o;
assign sda_1_i = sda_1_o & sda_2_o;
assign sda_2_i = sda_1_o & sda_2_o;
```

- Example of two I2C devices sharing the same pins:

```
assign scl_1_i = scl_pin;
assign scl_2_i = scl_pin;
assign scl_pin = (scl_1_o & scl_2_o) ? 1'bz : 1'b0;
assign sda_1_i = sda_pin;
assign sda_2_i = sda_pin;
assign sda_pin = (sda_1_o & sda_2_o) ? 1'bz : 1'b0;
```

- **Note:** scl_o should not be connected directly to scl_i, only via AND logic or a tristate I/O pin. This would prevent devices from stretching the clock period.

Design Flow

IP Customization and Generation

AXIL I2C Slave IP core is a part of the Raptor Design Suite Software. A customized AXIL I2C Slave can be generated from the Raptor's IP configurator window as shown in Figure 5.

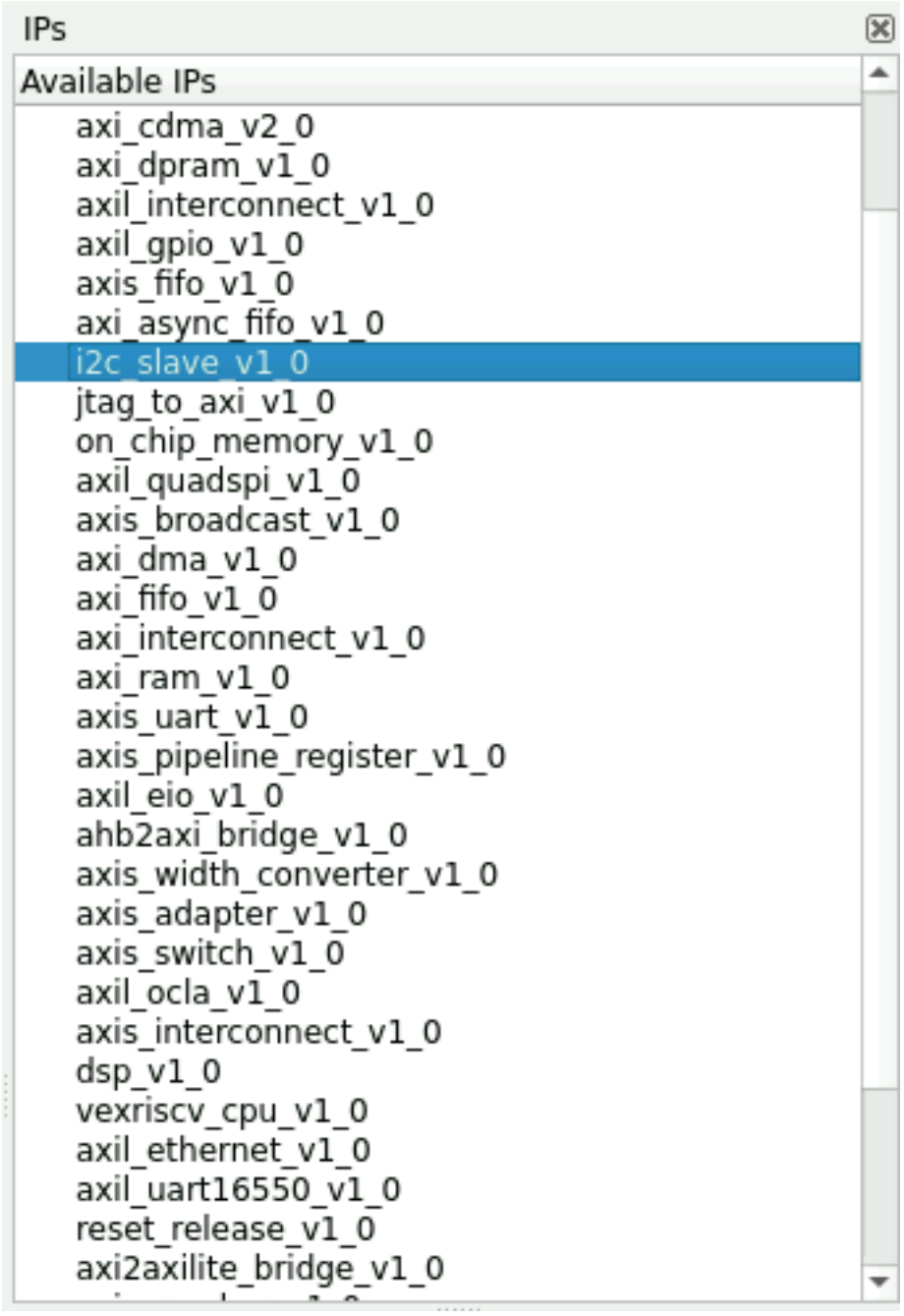

Figure 5: IP list

Parameters Customization

From the IP configuration window, the parameters of the I2C Slave can be configured and I2C Slave features can be enabled for generating a customized I2C Slave IP core that suits the user application requirement as shown in Figure 6. After IP Customization, all the source files are made available to the user with a top wrapper that instantiates a parameterized instance of the AXIL I2C Slave.

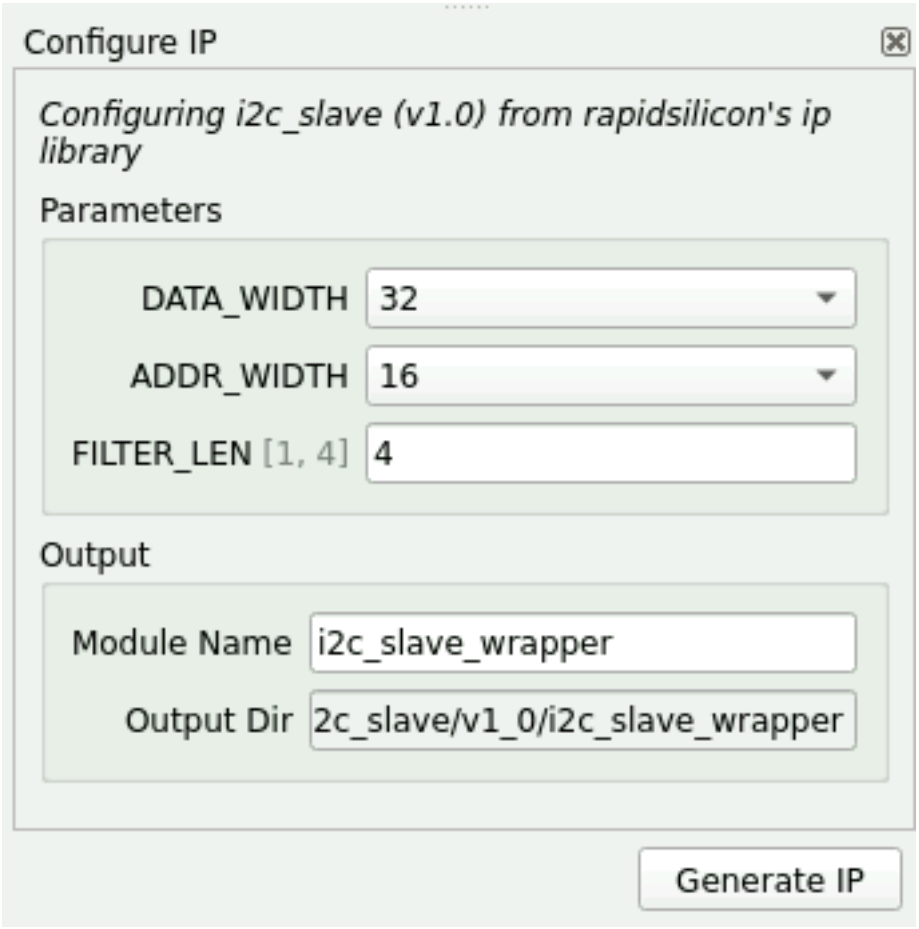

Figure 6: IP Configuration

Example Design

Overview

This AXIL I2C Slave IP can be utilized in a system that requires sequential transmission and reception of data in the form of reads and writes from the outside world with minimum wires and high througput. I2C is a crucial component in many electronic systems, enabling communication between the system and external devices through a serial interface. It can be embedded inside SoCs to enable two-way communication via the SoC in an AXI-Lite interface to maximize the efficiency with minimal overhead. One such example design of this AXIL I2C Master can be visualized in Figure 7.

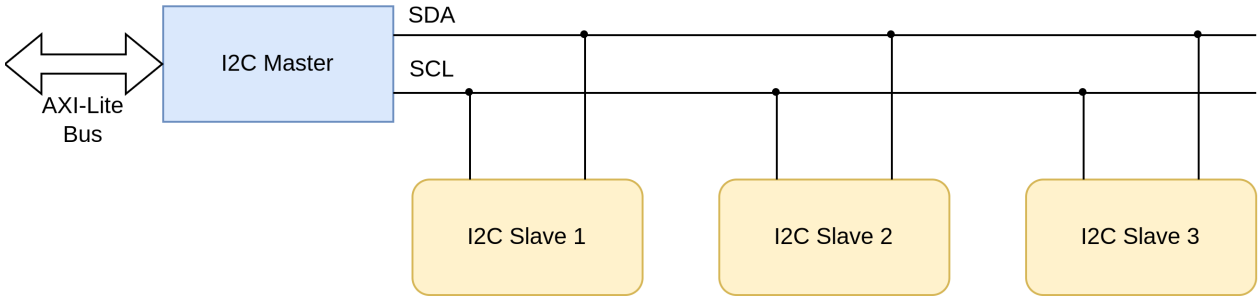

Figure 7: AXIL I2C Slaves connected with a Master

Simulating the Example Design

The IP being Verilog HDL, can be simulated via a bunch of industry standard simulator. For instance, it could be simulated via writing a Verilog Test-bench, or incorporating a soft processor that can stimulate this I2C Slave via a connected Master. The bundled example design is stimulated via a MyHDL based environment that iteratively stimulates the soft IP with the help of a Verilog testbench by performing various read and write operations on the I2C interface while also stimulating the AXI-Lite interface connected to the slave 2.

Synthesis and PR

Raptor 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 AXIL I2C Slave IP is a MyHDL based Python testbench that performs various read and write operations on the IP core with the help of pre-defined Verilog testbenches. Python is used to simulate the Verilog testbenches under the influence of MyHDL for this purpose and a total of 2 tests are performed, first one tests the embedded i2c_slave interface that is based on the AXI-Stream interface, this test is purely MyHDL with python without the specific Verilog testbench as this just makes sure that the i2c internal interface works as expected. The other tests the top level of the AXI-Lite I2C Slave in combination with a Verilog testbench that compares the write and read operations to make sure that the soft IP core for the AXIL I2C Slave is working as expected. The waveforms are also dumped in the format of .lxt for in-depth analysis of the whole operation. Being written in simple Verilog and Python, the testbenches are easily modifiable to provide maximum coverage of the AXIL I2C Slave IP.

Release

Release History

Date	Version	Revisions
May 4, 2023	0.1	Initial version AXI4-Lite I2C Slave User Guide Document