AXI4-Lite I2C Master (Beta Release)

Version 0.1

November 14, 2023

Copyright

Copyright © 2021 Rapid Silicon. All rights reserved. This document may not, in whole or part, be reproduced, modified, distributed, or publicly displayed without prior written consent from Rapid Silicon (“Rapid Silicon”).

Trademarks

All Rapid Silicon trademarks are as listed at www.rapidsilicon.com. Synopsys and Synplify Pro are trademarks of Synopsys, Inc. Aldec and Active-HDL are trademarks of Aldec, Inc. Modelsim and Questa are trademarks or registered trademarks of Siemens Industry Software Inc. or its subsidiaries in the United States or other countries. All other trademarks are the property of their respective owners.

Disclaimers

NO WARRANTIES: THE INFORMATION PROVIDED IN THIS DOCUMENT IS “AS IS” WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF ACCURACY, COMPLETENESS, MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL RAPID SILICON OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (WHETHER DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION PROVIDED IN THIS DOCUMENT, EVEN IF RAPID SILICON HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF CERTAIN LIABILITY, SOME OF THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

Rapid Silicon may make changes to these materials, specifications, or information, or to the products described herein, at any time without notice. Rapid Silicon makes no commitment to update this documentation. Rapid Silicon reserves the right to discontinue any product or service without notice and assumes no obligation to correct any errors contained herein or to advise any user of this document of any correction if such be made. Rapid Silicon recommends its customers obtain the latest version of the relevant information to establish that the information being relied upon is current and before ordering any products.

Contents

IP Summary

Introduction

I2C (Inter-Integrated Circuit) is a communication protocol that allows multiple devices to communicate with a single master device using only two wires: a clock wire (SCL) and a data wire (SDA). The master device controls the clock and initiates communication with slave devices on the bus. Each slave device has a unique address, allowing the master to communicate with specific slaves. I2C is commonly used in embedded systems, such as microcontrollers, to communicate with sensors, displays, and other peripherals. This I2C IP acts as a Master to which multiple slave devices can be connected via the AXILite interface making it compatible with other AXI based systems.

Features

- Independently configurable read and write address widths.
- Supports configurable command address width.
- Configurable I2C prescale with an optional default prescale.
- Supports the AXI4-Lite interface specification.
- Independently configurable FIFO for read and write operations.
- Optional command FIFO for the I2C.
- AXI-Stream based FIFO for fast data transactions.

Overview

AXIL I2C Master

The I2C Master soft IP core is designed to be implemented in a Field Programmable Gate Array (FPGA) or an Application-Specific Integrated Circuit (ASIC) device. It provides a configurable interface that supports various operating modes and data transfer rates. The IP core is optimized for high-performance and low-power operation, and it can be customized to meet specific design requirements. The I2C Master soft IP core provides a flexible and efficient solution for implementing I2C communication in embedded systems. It simplifies the design process, reduces development time and cost, and enables the implementation of complex communication protocols in resource-constrained environments. A slave may not transmit data unless it has been addressed by the master. A block diagram for the AXI-Lite I2C Master IP is shown in Figure 1.

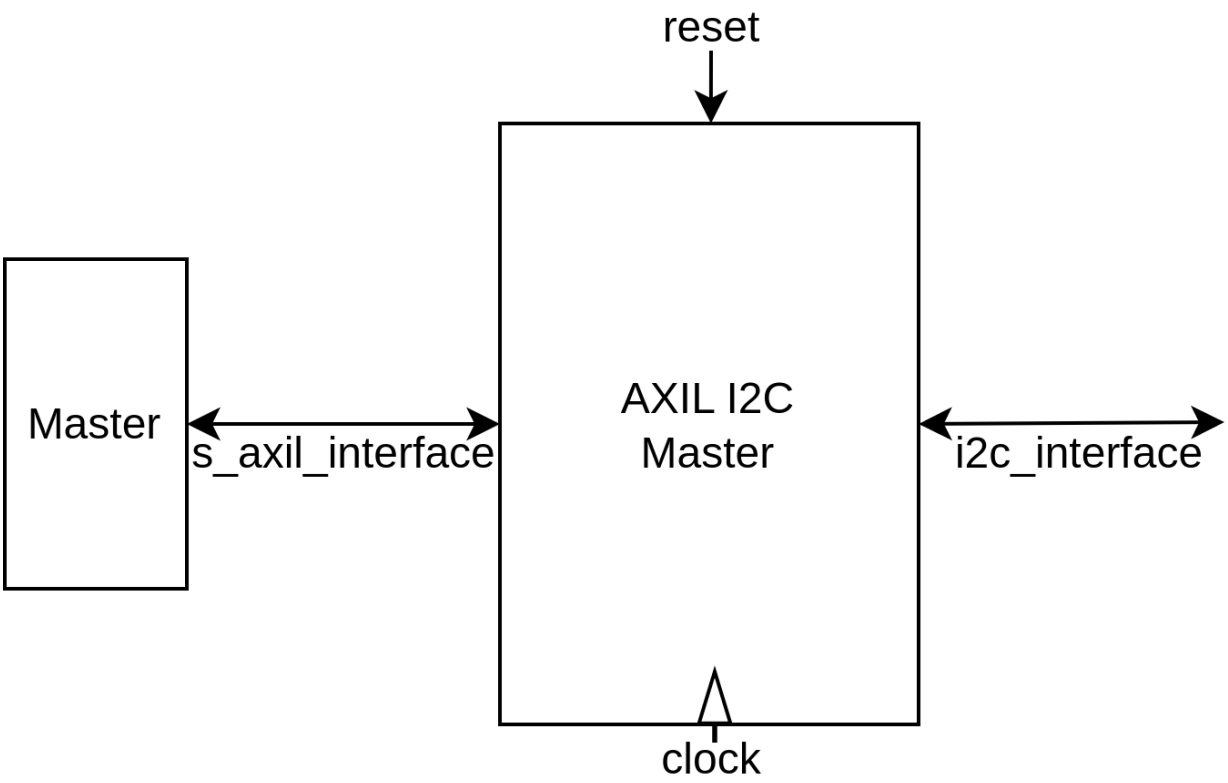

Figure 1: AXIL I2C Master Block Diagram

IP Specification

An I2C master is a device that initiates and controls the communication on the I2C bus. The master device generates the clock signal, initiates the start and stop conditions, and controls the flow of data on the bus. The master device can communicate with multiple slave devices on the same I2C bus.
The I2C master has the responsibility of sending data to the slave devices, receiving data from the slave devices, and controlling the communication flow. The master sends a start condition to initiate the communication and sends the slave device address along with the read/write bit to select the slave device. Once the slave device acknowledges the address, the master device can send or receive data from the slave device.
The I2C master device generates the clock signal that synchronizes the communication on the I2C bus. The master generates the clock pulses that are used to transmit and receive data between the master and slave devices on the I2C side. The clock speed is determined by the master device, and all slave devices on the I2C bus must support the same clock speed.
The master device controls the communication between these devices and allows for efficient data transfer between them. The top level embeds a FIFO and an I2C Master that are based on AXI-Strem protocol to ensure high throughput from within the soft IP. The internal block diagram can be seen in Figure 2.

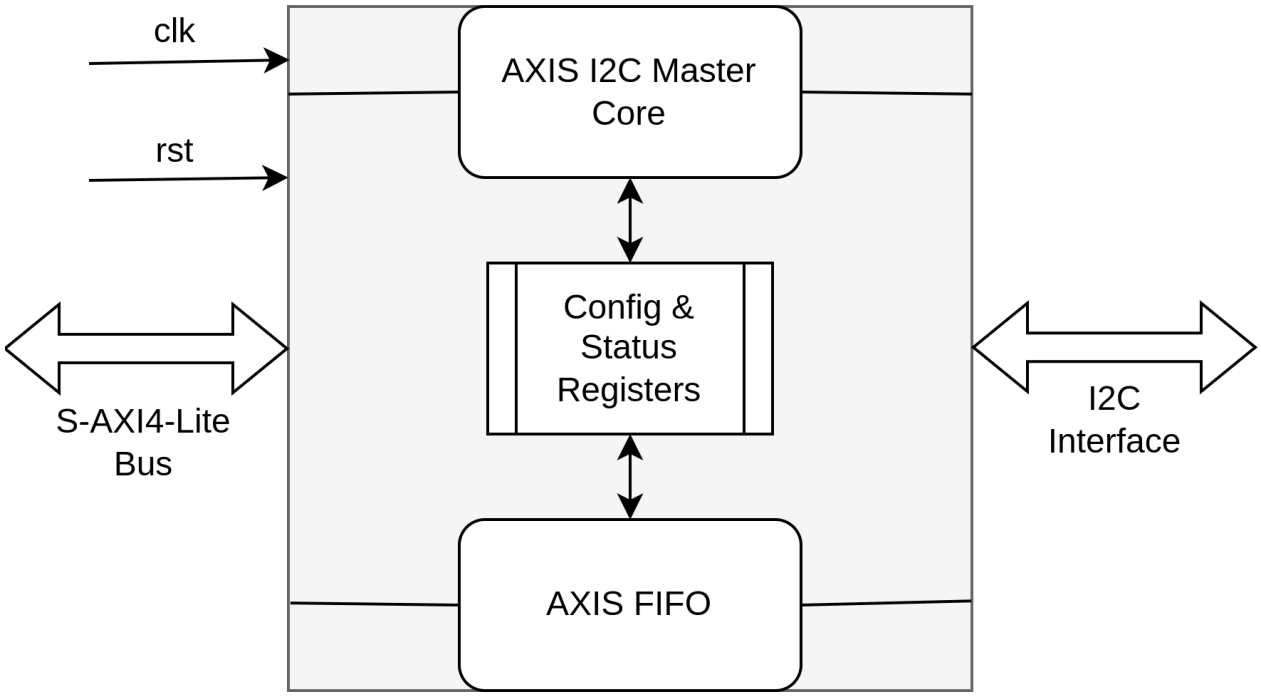

Figure 2: AXIL I2C Master Internal Diagram

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 Master.

Compliance		IP Resources				Tool Flow		
Device	Interface	Source Files	Constraint Files	Testbench	Simulation Model	Analyze and Elaboration	Simualtion	Synthesis
Gemini	AXI4-Lite / I2C	Verilog	-	Python / Verilog	MyHDL	Verific (Raptor)	Icarus (Raptor)	Raptor

Table 1: IP Details

Parameters

Table 2 lists the parameters of the AXIL I2C Master.

Parameter	Values	Default Value	Description
DEFAULT PRESCALE	0 / 1	1	I2C Default Prescale
FIXED PRESCALE	0 / 1	0	I2C Fixed Prescale
CMD FIFO	0 / 1	1	I2C FIFO Command Enable
WRITE FIFO	0 / 1	1	I2C FIFO Write Enable
READ FIFO	0 / 1	1	I2C FIFO Read Enable
CMD ADDR WIDTH	1 - 5	5	I2C FIFO Command Address Width
WRITE ADDR WIDTH	1 - 5	5	I2C FIFO Write Address Width
READ ADDR WIDTH	1 - 5	5	I2C FIFO Read Address Width

Table 2: Parameters

Port List

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

Signal Name	I/O	Description
AXI Clock and Reset		
clk	I	System Clock
rst	I	Active High Reset
Write Address Channel		
s_axil_awaddr	I	AXI4-Lite write address
s_axil_awprot	I	AXI4-Lite protection data qualifier
s_axil_awvalid	I	AXI4-Lite valid write address
s_axil_awready	O	AXI4-Lite write address ready
Write Data Channel		
s_axil_wdata	I	AXI4-Lite data
s_axil_wstrb	I	AXI4-Lite data stream identifier
s_axil_wvalid	I	AXI4-Lite data valid
s_axil_wready	O	AXI4-Lite data ready
Write Response Channel		
s_axil_bresp	O	AXI4-Lite transfer response
s_axil_bvalid	O	AXI4-Lite transfer valid response
s_axil_bready	I	AXI4-Lite transfer ready response
Read Address Channel		
s_axil_araddr	I	AXI4-Lite read address
s_axil_arprot	I	AXI4-Lite protection data qualifier
s_axil_arvalid	I	AXI4-Lite read address valid
s_axil_arready	O	AXI4-Lite read address ready
Read Data Channel		
s_axil_rdata	O	AXI4-Lite read data
s_axil_rresp	O	AXI4-Lite read response
s_axil_rvalid	O	AXI4-Lite read data valid
s_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

Table 3: AXIL I2C Master 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**
	DEFAULT PRESCALE	0	LUTs	228
	FIXED PRESCALE	0		
	CMD FIFO	0		
	WRITE FIFO	0	Registers	131
	READ FIFO	0		
Maximum Resource	**Options**	**Configuration**	**Resources**	**Utilized**
	DEFAULT PRESCALE	1	LUTs	264
	FIXED PRESCALE	1		
	CMD FIFO	1		
	WRITE FIFO	1	Registers	223
	READ FIFO	1		
	CMD ADDR WIDTH	5	BRAM	2
	WRITE ADDR WIDTH	5		
	READ ADDR WIDTH	5		

Table 4: Resource Utilization

Registers and Address Space

The Table 5 shows the available registers and their addresses in the I2C Master soft IP core.

Address	**Name**
0x00	Status
0x04	Command
0x08	Data
0x0C	Prescale

Table 5: Registers

The detail for each of these registers are given below:

- **Status Register**

  Table 6 highlights the addresses and usage of the Status Register.

Address	Name	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0x00	Data	-	-	-	-	-	-	-	-
Address	**Name**	**Bit 23**	**Bit 22**	**Bit 21**	**Bit 20**	**Bit 19**	**Bit 18**	**Bit 17**	**Bit 16**
0x00	Data	-	-	-	-	-	-	-	-
Address	**Name**	**Bit 15**	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 9**	**Bit 8**
0x00	Status	rf_full	rd_empty	wr_ovf	wr_full	wr_empty	cmd_ovf	cmd_full	cmd_empty
Address	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0x00	Status	-	-	-	-	miss_ack	bus_act	bus_cont	busy

Table 6: Status Register

- **busy:** high when module is performing an I2C operation
- **bus_cont:** high when module has control of active bus
- **bus_act:** high when bus is active
- **miss_ack:** set high when an ACK pulse from a slave device is not seen; write 1 to clear
- **cmd_empty:** command FIFO empty
- **cmd_full:** command FIFO full
- **cmd_ovf:** command FIFO overflow; write 1 to clear
- **wr_empty:** write data FIFO empty
- **wr_full:** write data FIFO full
- **wr_ovf:** write data FIFO overflow; write 1 to clear
- **rd_empty:** read data FIFO is empty
- **rd_full:** read data FIFO is full

Command Register

Table 7 highlights the addresses and usage of the Command Register.

Address	Name	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0x04	Data	-	-	-	-	-	-	-	-
Address	**Name**	**Bit 23**	**Bit 22**	**Bit 21**	**Bit 20**	**Bit 19**	**Bit 18**	**Bit 17**	**Bit 16**
0x04	Data	-	-	-	-	-	-	-	-
Address	**Name**	**Bit 15**	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 9**	**Bit 8**
0x04	Command	-	-	-	cmd_stop	cmd_wr_m	cmd_write	cmd_read	cmd_start
Address	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0x04	Command	-	cmd_address [6:0]						

Table 7: Command Register

- **cmd_address:** I2C address for command
- **cmd_start:** set high to issue I2C start, write to push on command FIFO
- **cmd_read:** set high to start read, write to push on command FIFO
- **cmd_write:** set high to start write, write to push on command FIFO
- **cmd_write_multiple:** set high to start block write, write to push on command FIFO

– **cmd_stop:** set high to issue I2C stop, write to push on command FIFO

Setting more than one command bit is allowed. Start or repeated start will be issued first, followed by read or write, followed by stop. Note that setting read and write at the same time is not allowed, this will result in the command being ignored.

- **Data Register**

Table 8 highlights the addresses and usage of the Data Register.

Address	**Name**	**Bit 31**	**Bit 30**	**Bit 29**	**Bit 28**	**Bit 27**	**Bit 26**	**Bit 25**	**Bit 24**
0x08	Data	-	-	-	-	-	-	-	-
Address	**Name**	**Bit 23**	**Bit 22**	**Bit 21**	**Bit 20**	**Bit 19**	**Bit 18**	**Bit 17**	**Bit 16**
0x08	Data	-	-	-	-	-	-	-	-
Address	**Name**	**Bit 15**	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 9**	**Bit 8**
0x08	Data	-	-	-	-	-	-	data_last	data_valid
Address	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0x08	Data	data [7:0]							

Table 8: Data Register

– **data:** I2C data, write to push on write data FIFO, read to pull from read data FIFO

– **data_valid:** indicates valid read data, must be accessed with atomic 16 bit reads and writes

– **data_last:** indicate last byte of block write (write_multiple), must be accessed with atomic 16 bit reads and writes

Prescale Register

Table 9 highlights the addresses and usage of the Prescale Register.

Address	**Name**	**Bit 31**	**Bit 30**	**Bit 29**	**Bit 28**	**Bit 27**	**Bit 26**	**Bit 25**	**Bit 24**
0x0C	Data	-	-	-	-	-	-	-	-
Address	**Name**	**Bit 23**	**Bit 22**	**Bit 21**	**Bit 20**	**Bit 19**	**Bit 18**	**Bit 17**	**Bit 16**
0x0C	Data	-	-	-	-	-	-	-	-
Address	**Name**	**Bit 15**	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 9**	**Bit 8**
0x0C	Prescale	prescale [15:8]							
Address	**Name**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0x0C	Prescale	prescale [7:0]							

Table 9: Prescale Register

- – **prescale:** set prescale value
  set prescale to 1/4 of the minimum clock period in units of input clk cycles

$$prescale = \frac{F_{clk}}{F_{I2Cclk} \times 4} \quad (1)$$

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 Master IP.

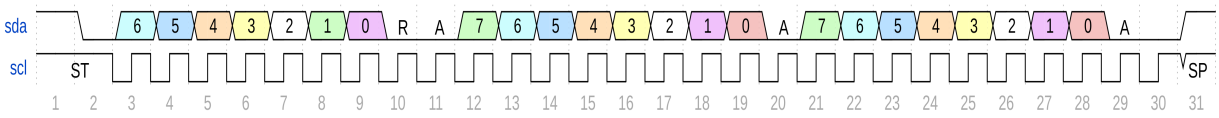

Figure 3: Read Cycle

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

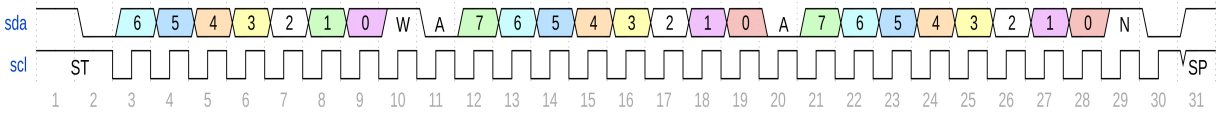

Figure 4: Write Cycle

The list of abbreviations used in the Figures 3 and 4 is given in Table 10.

Abbreviation	Definition
ST	Start Bit
SP	Stop Bit
W	Write
A	Acknowledge
R	Read

Table 10: List of Abbreviations

Commands

- **read**
  - read data byte
  - set start to force generation of a start condition
  - start is implied when bus is inactive or active with write or different address
  - set stop to issue a stop condition after reading current byte
  - set stop to issue a stop condition after reading current byte
  - if stop is set with read command, then data_out_last will be set
- **write**
  - write data byte
  - set start to force generation of a start condition
  - start is implied when bus is inactive or active with read or different address

  - set stop to issue a stop condition after writing current byte
- **write multiple**
  - write multiple data bytes (until data_in_last)
  - set start to force generation of a start condition
  - start is implied when bus is inactive or active with read or different address
  - set stop to issue a stop condition after writing block
- **stop**
  - issue stop condition if bus is active

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
- **missed_ack**
  - strobed when a slave ack is missed

Control Parameters

- **prescale**
  - set prescale to 1/4 of the minimum clock period in units of input clk cycles

$$prescale = \frac{F_{clk}}{F_{I2Cclk} \times 4} \tag{2}$$

- **stop_on_idle**
  - automatically issue stop when command input is not valid

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

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

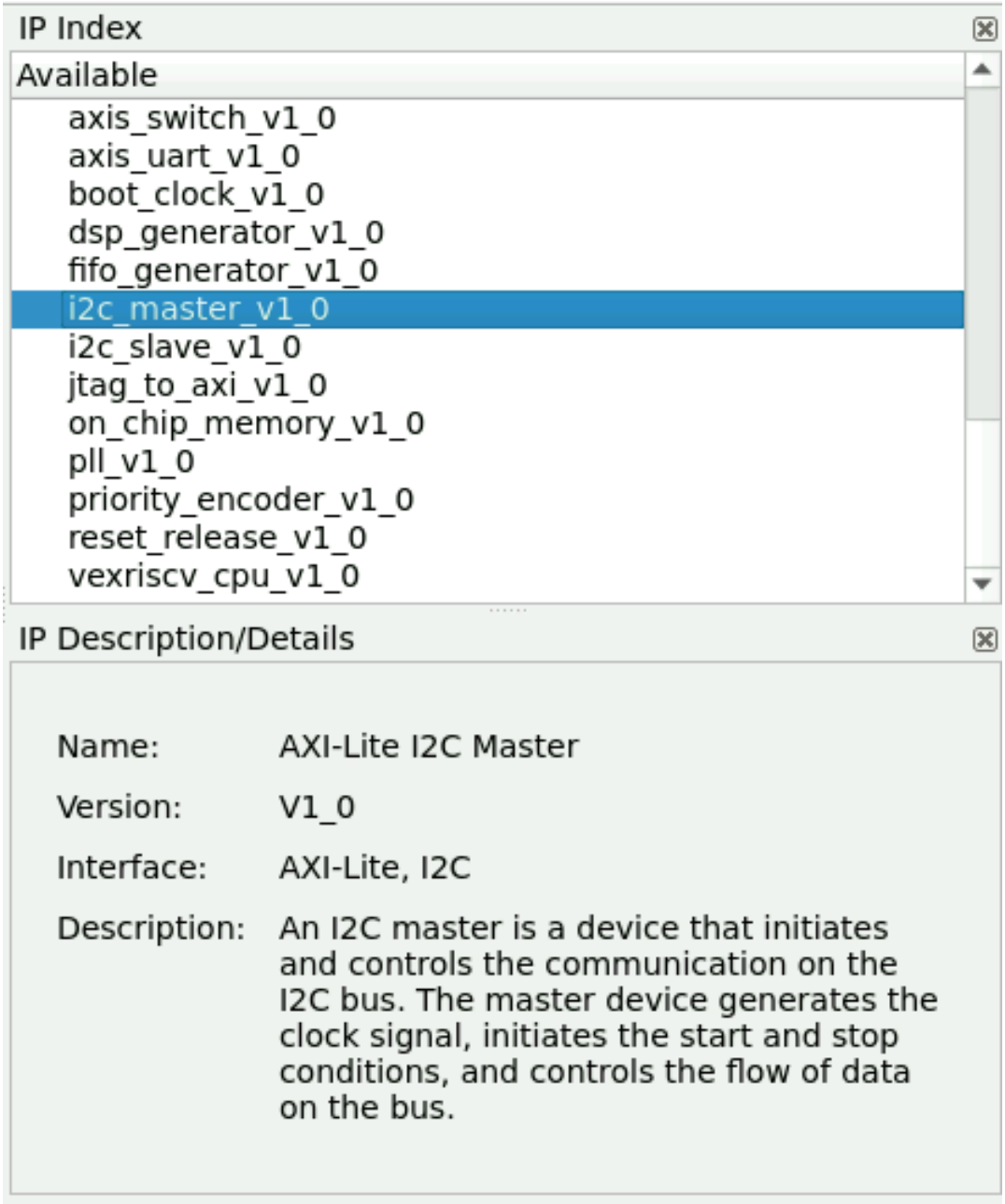

Figure 5: IP list

Parameters Customization

From the IP configuration window, the parameters of the I2C Master can be configured and I2C Master features can be enabled for generating a customized I2C Master 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 Master.

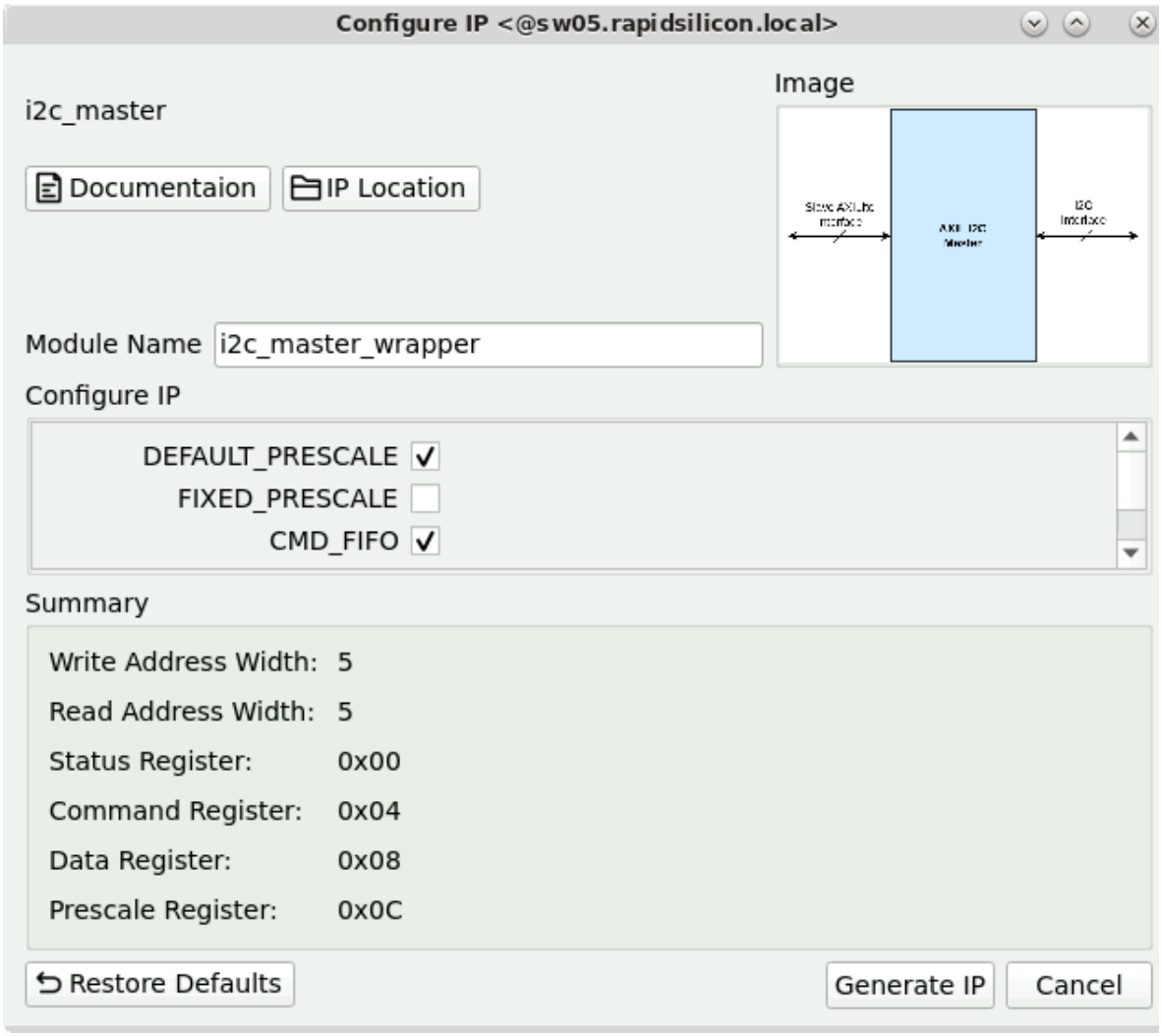

Figure 6: IP Configuration

Example Design

Overview

This AXIL I2C Master 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. 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 and is a part of the Raptor Design Suite.

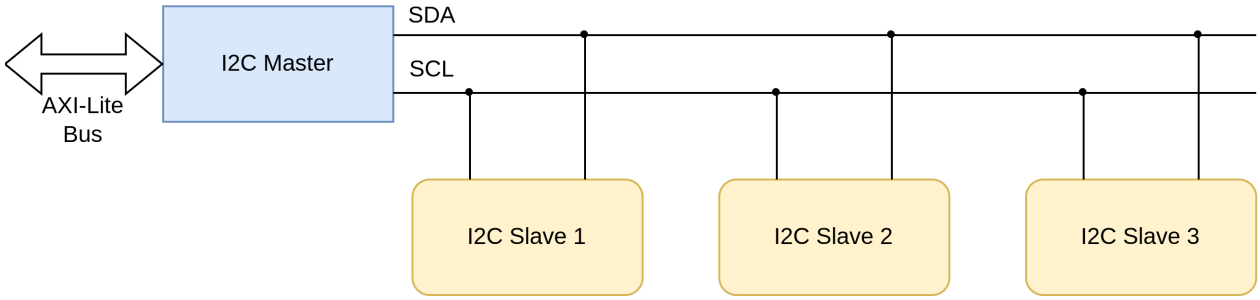

Figure 7: AXIL I2C Master connected with multiple Slaves

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 I2C 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.

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 Master 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_master 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 Master 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 Master 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 Master IP.

Release

Release History

Date	Version	Revisions
November 14, 2023	0.1	Initial version AXI4-Lite I2C Master User Guide Document