AXIS_DATA_WIDTH_CONVERTER

November 20, 2024

Jay Convertino

Contents

1 Usage **2**
- 1.1 Introduction . . . 2
- 1.2 Dependencies . . . 2
  - 1.2.1 fusesoc_info Depenecies . . . 2
- 1.3 In a Project . . . 2

2 Architecture **2**

3 Building **4**
- 3.1 fusesoc . . . 4
- 3.2 Source Files . . . 4
  - 3.2.1 fusesoc_info File List . . . 4
- 3.3 Targets . . . 5
  - 3.3.1 fusesoc_info Targets . . . 5
- 3.4 Directory Guide . . . 9

4 Simulation **10**
- 4.1 iverilog . . . 10
- 4.2 cocotb . . . 10

5 Module Documentation **11**
- 5.1 axis_data_width_converter . . . 12

1 Usage

1.1 Introduction

This data width converter is for even integer divides of slave to master or master to slave. Example this core can go from 4 bytes to 2 bytes or 2 bytes to 4 bytes. It can not go from 5 bytes to 2 bytes or 2 bytes to 5 bytes. 4/2 is 2, a round number. 5/2 is a fractional number that will not work with this core.

1.2 Dependencies

The following are the dependencies of the cores.

- fusesoc 2.X
- iverilog (simulation)
- cocotb (simulation)

1.2.1 fusesoc_info Depenecies

- dep
  - AFRL:utility:helper:1.0.0
- dep_tb
  - AFRL:simulation:axis_stimulator
  - AFRL:simulation:clock_stimulator
  - AFRL:utility:sim_helper

1.3 In a Project

Simply use this core between a sink and source AXIS devices. This will convert from one BUS size to another. Check the code to see if others will work correctly.

2 Architecture

The only module is the axis_data_width_converter module. It is listed below.

- **axis_data_width_converter** Implement an algorithm to convert BUS data interfaces in even multiples (see core for documentation).

The data width converter uses a generate block to select between three possible scenarios. First is they are equal, which just assigns the fields to each other. The second is slave is smaller then the master. This uses a register build up method by building up data till the correct number of bytes is reached. The final method is slave is larger than the master. This works by slicing the incoming data into chunks for the slave.

The slave is smaller than master block has the following steps.

1. Create registers used to buffer data and signals
2. Generate a backpressure tready signal for the axis input using the current tready master and its previous state.
3. The output is valid if the register has valid.
4. Last is set if register has last and the data is valid.
5. always block processes data in the following manner.
    (a) Once out of reset, check if the output device is ready. If it is clear out the data tlast and valid registers and set the previous tready to 0.
    (b) If the input is valid, and we were previously not ready or are currently ready start processing slave data.
        i. build up slave data in buffer
        ii. build up last buffer
        iii. increment counter, or decrement if reversed byte order.
        iv. Once counter hits its threshold, reset counter to initial value, and the buffer data is now valid so register for valid is set to active high.

The slave is larger than master block has the following steps.

1. Use a for loop to generate an assignment to take input data and slice it into a wire that is segmented into the size of the output data.
2. Ready happens when counter hits its count and the proper signals are set. Backpressure is needed since going larger to smaller will take more clock cycles.
3. For the output, if the register for valid is active high, output the proper signal or data.
4. always block processes data in the following manner.
    (a) Once out of reset, check if the output device is ready. If it is, register for valid is set to 0 and the previous tready is cleared.

(b) If the input is valid, and we were previously not ready or are currently ready, and the counter has reach the start count, start processing slave data.
   i. In an unrolled for loop take the split input data and store it in a buffer.
   ii. increment counter, or decrement if reversed byte order.
   iii. Data is valid, so set it active high
   iv. previous tready is set to active high, since the core has to be ready to take data.

(c) Check the counter, and check if the destination device is ready, if it is, decrement the counter and reassert the valid and previous tready.

Please see 5 for more information.

3 Building

The AXIS data width converter core is written in Verilog 2001. They should synthesize in any modern FPGA software. The core comes as a fusesoc packaged core and can be included in any other core. Be sure to make sure you have meet the dependencies listed in the previous section.

3.1 fusesoc

Fusesoc is a system for building FPGA software without relying on the internal project management of the tool. Avoiding vendor lock in to Vivado or Quartus. These cores, when included in a project, can be easily integrated and targets created based upon the end developer needs. The core by itself is not a part of a system and should be integrated into a fusesoc based system. Simulations are setup to use fusesoc and are a part of its targets.

3.2 Source Files

3.2.1 fusesoc_info File List

- src

  Type: verilogSource

  - src/axis_data_width_converter.v

- tb

  - 'tb/tb_axis.v': 'file_type': 'verilogSource'

3.3 Targets

3.3.1 fusesoc_info Targets

- default

  Info: Default for IP intergration.

  - src
  - dep

- sim

  Info: Test 1:1 conversion.

  - src
  - dep
  - tb
  - dep_tb
  - IN_FILE_NAME
  - OUT_FILE_NAME
  - RAND_READY
  - MASTER_WIDTH
  - SLAVE_WIDTH

- sim_reduce

  Info: Test data reduction.

  - src
  - dep
  - tb
  - dep_tb
  - IN_FILE_NAME
  - OUT_FILE_NAME
  - RAND_READY
  - MASTER_WIDTH=2
  - SLAVE_WIDTH=4

- sim_rand_data_reduce

  Info: Test data reduction with random data

  - src
  - dep
  - tb

  - dep_tb
  - IN_FILE_NAME=random.bin
  - OUT_FILE_NAME=out_random.bin
  - RAND_READY
  - MASTER_WIDTH=2
  - SLAVE_WIDTH=4
- sim_rand_ready_rand_data_reduce

  Info: Test data reduction with random ready and random data.
  - src
  - dep
  - tb
  - dep_tb
  - IN_FILE_NAME=random.bin
  - OUT_FILE_NAME=out_random.bin
  - RAND_READY=1
  - MASTER_WIDTH=2
  - SLAVE_WIDTH=4
- sim_8bit_count_data_reduce

  Info: Test data reduction with counter data.
  - src
  - dep
  - tb
  - dep_tb
  - IN_FILE_NAME=8bit_count.bin
  - OUT_FILE_NAME=out_8bit_count.bin
  - RAND_READY
  - MASTER_WIDTH=2
  - SLAVE_WIDTH=4
- sim_rand_ready_8bit_count_data_reduce

  Info: Test data reduction with counter data, and random ready.
  - src
  - dep

  - tb
  - dep_tb
  - IN_FILE_NAME=8bit_count.bin
  - OUT_FILE_NAME=out_8bit_count.bin
  - RAND_READY=1
  - MASTER_WIDTH=2
  - SLAVE_WIDTH=4
- sim_increase

  Info: Test data increase.
  - src
  - dep
  - tb
  - dep_tb
  - IN_FILE_NAME
  - OUT_FILE_NAME
  - RAND_READY
  - MASTER_WIDTH=4
  - SLAVE_WIDTH=2
- sim_rand_data_increase

  Info: Test data increase with random data.
  - src
  - dep
  - tb
  - dep_tb
  - IN_FILE_NAME=random.bin
  - OUT_FILE_NAME=out_random.bin
  - RAND_READY
  - MASTER_WIDTH=4
  - SLAVE_WIDTH=2
- sim_rand_ready_rand_data_increase

  Info: Test data increase with random data, and random ready.
  - src
  - dep

  - tb
  - dep_tb
  - IN_FILE_NAME=random.bin
  - OUT_FILE_NAME=out_random.bin
  - RAND_READY=1
  - MASTER_WIDTH=4
  - SLAVE_WIDTH=2
- sim_8bit_count_data_increase

  Info: Test data increase with count data.
  - src
  - dep
  - tb
  - dep_tb
  - IN_FILE_NAME=8bit_count.bin
  - OUT_FILE_NAME=out_8bit_count.bin
  - RAND_READY
  - MASTER_WIDTH=4
  - SLAVE_WIDTH=2
- sim_rand_ready_8bit_count_data_increase

  Info: Test data increase with count data, and random ready.
  - src
  - dep
  - tb
  - dep_tb
  - IN_FILE_NAME=8bit_count.bin
  - OUT_FILE_NAME=out_8bit_count.bin
  - RAND_READY=1
  - MASTER_WIDTH=4
  - SLAVE_WIDTH=2

3.4 Directory Guide

Below highlights important folders from the root of the directory.

1. **docs** Contains all documentation related to this project.
    - **manual** Contains user manual and github page that are generated from the latex sources.
2. **src** Contains source files for the core
3. **tb** Contains test bench files for iverilog and cocotb
    - **cocotb** testbench files

4 Simulation

There are a few different simulations that can be run for this core.

4.1 iverilog

iverilog is used for simple test benches for quick verification, visually, of the core.

4.2 cocotb

Future simulations will use cocotb. This feature is not yet implemented.

5 Module Documentation

There is a single async module for this core.

- **axis_data_width_converter** AXIS data width converter, converts from one BUS data size to another.

The next sections document the module in great detail.

axis_data_width_converter.v

AUTHORS

JAY CONVERTINO

DATES

2021/06/21

INFORMATION

Brief

AXIS DATA WIDTH CONVERTER

License MIT

axis_data_width_converter

```
module axis_data_width_converter #(
parameter
SLAVE_WIDTH
 =
1,
parameter
MASTER_WIDTH
 =
1,
parameter
REVERSE
```

```
  =
 0
) ( input aclk, input arstn, output [(MASTER_WIDTH*8)-1:0] m_axis_tdata, out
```

Change size of streaming bus in even integers of. 1/2 2/1 2/4 4/2 etc.

Parameters

SLAVE_WIDTH parameter	Width of the slave input bus in bytes
MASTER_WIDTH parameter	Width of the master output bus in bytes
REVERSE parameter	Change byte order

Ports

aclk	Clock for AXIS
arstn	Negative reset for AXIS
m_axis_tdata	Output data
m_axis_tvalid	When active high the output data is valid
m_axis_tready	When set active high the output device is ready for data.
m_axis_tlast	Indicates last word in stream.
s_axis_tdata	Input data
s_axis_tvalid	When set active high the input data is valid
s_axis_tready	When active high the device is ready for input data.
s_axis_tlast	Is this the last word in the stream (active high).