AXIS_DATA_WIDTH_CONVERTER

May 20, 2025

Jay Convertino

Contents

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 **??** 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. Linting is performed by verible using the lint target.

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

- tb_cocotb
  - 'tb/tb_cocotb.py': 'file_type': 'user', 'copyto': '.'
  - 'tb/tb_cocotb.v': 'file_type': 'verilogSource'

3.3 Targets

3.3.1 fusesoc_info Targets

- default

  Info: Default for IP intergration.

- lint

  Info: Lint with Verible

- sim

  Info: Test 1:1 conversion.

- sim_reduce

  Info: Test data reduction.

- sim_rand_data_reduce

  Info: Test data reduction with random data

- sim_rand_ready_rand_data_reduce

  Info: Test data reduction with random ready and random data.

- sim_8bit_count_data_reduce

  Info: Test data reduction with counter data.

- sim_rand_ready_8bit_count_data_reduce

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

- sim_increase

  Info: Test data increase.

- sim_rand_data_increase

  Info: Test data increase with random data.

- sim_rand_ready_rand_data_increase

Info: Test data increase with random data, and random ready.

- sim_8bit_count_data_increase

  Info: Test data increase with count data.

- sim_rand_ready_8bit_count_data_increase

  Info: Test data increase with count data, and random ready.

- sim_cocotb

  Info: Cocotb unit tests

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

4 Simulation

There are a few different simulations that can be run for this core. All currently use iVerilog (icarus) to run. The first is iverilog, which uses verilog only for the simulations. The other is cocotb. This does a unit test approach to the testing and gives a list of tests that pass or fail.

4.1 iverilog

All simulation targets that do NOT have cocotb in the name use a verilog test bench with verilog stimulus components. These all read in a file and then write a file that has been processed by the data width converter. Then the input and output file are compared with a MD5 sum to check that they match. If they do not match then the test has failed. All of these tests provide fst output files for viewing the waveform in the there target build folder.

4.2 cocotb

To use the cocotb tests you must install the following python libraries.

```
$ pip install cocotb
$ pip install cocotbext-axi
```

Then you must use the cocotb sim target. In this case it is sim_cocotb. This target can be run with various bus and fifo parameters.

```
$ fusesoc run --target sim_cocotb AFRL:
    ↪ streaming_converter:axis_data_width_converter
    ↪ :1.0.1 --SLAVE_WIDTH=8 --MASTER_WIDTH=32
```

The following is an example command to run through various parameters without typing them one by one.

```
$ for i in {1..32}; do sleep 5; export RY=$((RANDOM*2)
    ↪ %32)); fusesoc run --target sim_cocotb AFRL:
    ↪ streaming_converter:axis_data_width_converter
    ↪ :1.0.1 --SLAVE_WIDTH=$i --MASTER_WIDTH=$RY; echo
    ↪ "SLAVE WIDTH:" $i "MASTER WIDTH:" $RY; done
```

5 Code Documentation

Natural docs is used to generate documentation for this project. The next lists the following sections.

- **axis_data_width_converter** AXIS data width converter, converts from one BUS data size to another.
- **tb_axis** Verilog test bench.
- **tb_cocotb verilog** Verilog test bench base for cocotb.
- **tb_cocotb python** cocotb unit test functions.

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).

tb_axis.v

AUTHORS

JAY CONVERTINO

DATES

2024/12/09

INFORMATION

Brief

Test bench for axis_data_width_converter using axis stim and clock stim.

License MIT

tb_axis

```
module tb_axis #(
parameter
IN_FILE_NAME
 =
in.bin,
parameter
OUT_FILE_NAME
 =
out.bin,
parameter
RAND_READY
 =
0,
parameter
```

```
 SLAVE_WIDTH
  =
 2,
 parameter
 MASTER_WIDTH
  =
 4
)()
```

Test bench for axis_data_width_converter. This will run a file through the system and write its output. These can then be compared to check for errors. If the files are identical, no errors. A FST file will be written.

Parameters

IN_FILE_NAME parameter	File name for input.
OUT_FILE_NAME parameter	File name for output.
RAND_READY parameter	0 = no random ready. 1 = randomize ready.
SLAVE_WIDTH parameter	Width of the slave input bus in bytes
MASTER_WIDTH parameter	Width of the master output bus in bytes

INSTANTIATED MODULES

clk_stim

```
clk_stimulus #(
                                                                    .
CLOCKS(1),
                                                                    .
CLOCK_BASE(1000000),
                                                                    .
CLOCK_INC(1000),
                                                                    .
RESETS(1),
                                                                    .
RESET_BASE(2000),
                                                                    .
RESET_INC(100)
) clk_stim ( .clkv(tb_dut_clk), .rstnv(tb_dut_rstn), .rstv() )
```

Generate a 50/50 duty cycle set of clocks and reset.

slave_axis_stim

```
slave_axis_stimulus #(
                                                                    .
BUS_WIDTH(SBUS_WIDTH),
                                                                    .
USER_WIDTH(USER_WIDTH),
                                                                    .
DEST_WIDTH(DEST_WIDTH),
                                                                    .
FILE(IN_FILE_NAME)
```

```
) slave_axis_stim ( .m_axis_aclk(tb_dut_clk), .m_axis_arstn(tb_dut_rstn), .r
```

Device under test SLAVE stimulus module.

dut

```
 axis_data_width_converter #(
                                                                                .
 MASTER_WIDTH(MBUS_WIDTH),
                                                                                .
 SLAVE_WIDTH(SBUS_WIDTH)
) dut ( .aclk(tb_dut_clk), .arstn(tb_dut_rstn), .m_axis_tdata(tb_dut_data),
```

Device under test, axis_data_width_converter

master_axis_stim

```
 master_axis_stimulus #(
                                                                                .
 BUS_WIDTH(MBUS_WIDTH),
                                                                                .
 USER_WIDTH(USER_WIDTH),
                                                                                .
 DEST_WIDTH(DEST_WIDTH),
                                                                                .
 RAND_READY(RAND_READY),
                                                                                .
 FILE(OUT_FILE_NAME)
) master_axis_stim ( .s_axis_aclk(tb_dut_clk), .s_axis_arstn(tb_dut_rstn),
```

Devie under test MASTER stimulus module.

tb_cocotb.v

AUTHORS

JAY CONVERTINO

DATES

2024/12/11

INFORMATION

Brief

Test bench wrapper for cocotb

License MIT

tb_cocotb

```
module tb_cocotb #(
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
```

Test bench for data width converter. This will run a file through the system and write its output. These can then be compared to check for errors. If the files are identical, no errors. A FST file will be written.

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).

INSTANTIATED MODULES

dut

```
 axis_data_width_converter #(
                                                                    .
 MASTER_WIDTH(MASTER_WIDTH),
                                                                    .
 SLAVE_WIDTH(SLAVE_WIDTH)
) dut ( .aclk(aclk), .arstn(arstn), .m_axis_tdata(m_axis_tdata), .m_axis_tva
```

Device under test, axis_data_width_converter

tb_cocotb.py

AUTHORS

JAY CONVERTINO

DATES

2024/12/09

INFORMATION

Brief

Cocotb test bench

License MIT

FUNCTIONS

random_bool

```
def random_bool()
```

Return a infinte cycle of random bools

Returns: List

start_clock

```
def start_clock(
dut
)
```

Start the simulation clock generator.

Parameters

dut Device under test passed from cocotb test function

reset_dut

```
async def reset_dut(
dut
)
```

Cocotb coroutine for resets, used with await to make sure system is reset.

Parameters

dut Device under test passed from cocotb.

conversion_test

```
@cocotb.test()
async def conversion_test(
dut
)
```

Coroutine that is identified as a test routine. This routine tests for conversion based on current input to output size conversion.

Parameters

dut Device under test passed from cocotb.

conversion_test_random_ready

```
@cocotb.test()
async def conversion_test_random_ready(
dut
)
```

Coroutine that is identified as a test routine. This routine tests for randomized ready from the sink.

Parameters

dut Device under test passed from cocotb.

in_reset

```
@cocotb.test()
async def in_reset(
dut
```

```
)
```

Coroutine that is identified as a test routine. This routine tests if device stays in unready state when in reset.

Parameters

dut Device under test passed from cocotb.

no_clock

```
@cocotb.test()
async def no_clock(
dut
)
```

Coroutine that is identified as a test routine. This routine tests if no ready when clock is lost and device is left in reset.

Parameters

dut Device under test passed from cocotb.