AXIS_TINY_FIFO

May 20, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

Pipeline method axis fifo. This fifo uses a pipeline to create a fifo. Meaning it has a latency the size of the depth. If the output receiving core is not ready the core will build up data till it is full. All data will have a latency of the depth.

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 buffer data from one bus to another. Check the code to see if others will work correctly.

2 Architecture

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

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

  The always process is the logic behind the tiny fifo.

1. When out of reset, input new data if the output is ready and the valid buffer contains a 0 in the valid buffer. Otherwise assert current data.
2. Unrolled for loop that performs the following operations.
   (a) NAND all valids lower then current register. This results in all data being shifted below and including that index.
   (b) if any have 0 for valid (no data) shift. Also shift if ready, since destination is ready to take data anyways.

Please see 5 for more information.

3 Building

The AXIS tiny FIFO 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 the lint target using verible.

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_tiny_fifo.v': 'file_type': 'verilogSource'
- 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: Constant data value with file check.

- sim_rand_data

  Info: Feed random data input with file check

- sim_rand_ready_rand_data

  Info: Feed random data input, and randomize the read ready on the output. Perform output file check.

- sim_8bit_count_data

  Info: Feed a counter data as input, perform file check.

- sim_rand_ready_8bit_count_data

  Info: Feed a counter data a input, and randomize the read ready on the output. Perform output file check.

- 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 FIFO. 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:buffer:
    ↪ axis_tiny_fifo:1.0.0 --BUS_WIDTH=8 --FIFO_DEPTH
    ↪ =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
    ↪ %32+1)); fusesoc run --target sim_cocotb AFRL:
    ↪ buffer:axis_tiny_fifo:1.0.0 --BUS_WIDTH=$i --
    ↪ FIFO_DEPTH=$RY; echo "BUS␣WIDTH:" $i "FIFO␣DEPTH:
    ↪ " $RY; done
```

5 Code Documentation

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

- **axis_tiny_fifo** AXIS tiny fifo will buffer data from input to output.
- **tb_axis** Verilog test bench.
- **tb_cocotb verilog** Verilog test bench base for cocotb.
- **tb_cocotb python** cocotb unit test functions.

axis_tiny_fifo.v

AUTHORS

JAY CONVERTINO

DATES

2021/06/04

INFORMATION

Brief

AXIS TINY FIFO

License MIT

axis_tiny_fifo

```
module axis_tiny_fifo #(
parameter
FIFO_DEPTH
 =
4,
parameter
BUS_WIDTH
 =
8
) ( input aclk, input arstn, output [(BUS_WIDTH*8)-1:0] m_axis_tdata, output
```

AXIS fifo that uses a shift register to buffer data. This Adds latency to the design in the amount of the FIFO_DEPTH. Though if the destination isn't ready it will build up data to that FIFO_DEPTH and overwrite any

non-valid data inserted.

Parameters

FIFO_DEPTH parameter	Number of transactions to buffer.
BUS_WIDTH parameter	Width of the input/output bus in bytes.

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_tlast	Indicates last word in stream.
m_axis_tready	When set active high the output device is ready for data.
s_axis_tdata	Input data
s_axis_tvalid	When set active high the input data is valid
s_axis_tlast	Is this the last word in the stream (active high).
s_axis_tready	When active high the device is ready for input data.

VARIABLES

s_axis_tready

```
assign s_axis_tready = (
                                                                    ~&

reg_valid_buffer ||
m_axis_tready
) & arstn
```

If any valid is 0, we are ready for data

m_axis_tdata

```
assign m_axis_tdata = reg_data_buffer[0]
```

assign output data as soon as its ready

m_axis_tvalid

```
assign m_axis_tvalid = reg_valid_buffer[0]
```

assign output data as soon as its ready

m_axis_tlast

```
assign m_axis_tlast = reg_last_buffer[0]
```

assign output data as soon as its ready

tb_axis.v

AUTHORS

JAY CONVERTINO

DATES

2024/12/09

INFORMATION

Brief

Test bench for axis_tiny_fifo 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
```

```
 FIFO_DEPTH
  =
 8
)
```

Test bench for axis_tiny_fifo. 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.
FIFO_DEPTH parameter	Number of transactions to buffer.

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_stim_clk), .rstnv(tb_stim_rstn), .rstv() )
```

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

slave_axis_stim

```
 slave_axis_stimulus #(
                                                                                .
 BUS_WIDTH(BUS_WIDTH),
                                                                                .
 USER_WIDTH(USER_WIDTH),
                                                                                .
 DEST_WIDTH(DEST_WIDTH),
                                                                                .
 FILE(IN_FILE_NAME)
) slave_axis_stim ( .m_axis_aclk(tb_stim_clk), .m_axis_arstn(tb_stim_rstn),
```

Device under test SLAVE stimulus module.

dut

```
 axis_tiny_fifo #(
                                                                                .
 FIFO_DEPTH(FIFO_DEPTH),
                                                                                .
 BUS_WIDTH(BUS_WIDTH)
) dut ( .aclk(tb_stim_clk), .arstn(tb_stim_rstn), .s_axis_tvalid(tb_stim_va
```

Device under test, axis_tiny_fifo

master_axis_stim

```
 master_axis_stimulus #(
                                                                                .
 BUS_WIDTH(BUS_WIDTH),
                                                                                .
 USER_WIDTH(USER_WIDTH),
                                                                                .
 DEST_WIDTH(DEST_WIDTH),
                                                                                .
 RAND_READY(RAND_READY),
                                                                                .
 FILE(OUT_FILE_NAME)
) master_axis_stim ( .s_axis_aclk(tb_stim_clk), .s_axis_arstn(tb_stim_rstn),
```

Devie under test MASTER stimulus module.

tb_coctb.v

AUTHORS

JAY CONVERTINO

DATES

2024/12/09

INFORMATION

Brief

Test bench wrapper for cocotb

License MIT

tb_cocotb

```
module tb_cocotb #(
parameter
FIFO_DEPTH
 =
4,
parameter
BUS_WIDTH
 =
8
) ( input aclk, input arstn, output [(BUS_WIDTH*8)-1:0] m_axis_tdata, output
```

Test bench for axis_tiny_fifo. 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

FIFO_DEPTH parameter	Number of transactions to buffer.
BUS_WIDTH parameter	Number of bytes for tdata width.

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_tlast	Indicates last word in stream.
m_axis_tready	When set active high the output device is ready for data.
s_axis_tdata	Input data
s_axis_tvalid	When set active high the input data is valid
s_axis_tlast	Is this the last word in the stream (active high).
s_axis_tready	When active high the device is ready for input data.

INSTANTIATED MODULES

dut

```
axis_tiny_fifo #(
                                                                                .
FIFO_DEPTH(FIFO_DEPTH),
                                                                                .
BUS_WIDTH(BUS_WIDTH)
) dut ( .aclk(aclk), .arstn(arstn), .s_axis_tvalid(s_axis_tvalid), .s_axis_t
```

Device under test, axis_tiny_fifo

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.

single_word

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

Coroutine that is identified as a test routine. This routine tests for writing a single word, and then reading a single word.

Parameters

dut Device under test passed from cocotb.

full_empty

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

Coroutine that is identified as a test routine. This routine tests for writing till the fifo is full, Then reading from the full FIFO.

Parameters

dut Device under test passed from cocotb.

random_ready

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