AXIS_MOVING_AVERAGE

December 11, 2024

Jay Convertino

Contents

1 Usage

1.1 Introduction

This core provides the AXIS Moving Average function. This impliments the moving average algorithm in an efficent way for FPGAs. Efficent since it uses powers of two to calculate its weights. This formula implimented is

$$\frac{X_0 + ...X_n}{n}$$

where n is constrained to a power of two and X is a unsigned number. This also works as a low pass filter or a smothing algorithm.

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 has been tested with unsigned data types. Check the code to see if others will work correctly.

2 Architecture

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

- **axis_moving_average** Impliment moving average algorithm (see core for documentation).

This core only uses a combinatoral method to divide the accumulator. Since all weights are powers of two this is done with a part select based on bit position.

The always block has the following steps.

1. If there is valid data, sum the new data into the accumulator and remove the top element in the buffer from the accumulator.
2. Insert the new element into the buffer.
3. Shift the buffer to so that old elements at the top of the buffer are shifted out.

Please see **??** for more information.

3 Building

The AXIS Moving Average 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

- src/axis_moving_average.v

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

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

3.3 Targets

3.3.1 fusesoc_info Targets

- default

  Info: Default for IP intergration.

- sim

  Info: Default for simulation using icarus.

- 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:math:
    ↪ axis_moving_average:1.0.1 --BUS_WIDTH=8 --WEIGHT
    ↪ =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=$((RANDOM*2)
    ↪ %32)); fusesoc run --target sim_cocotb AFRL:math:
    ↪ axis_moving_average:1.0.1 --BUS_WIDTH=$i --WEIGHT
    ↪ =$RY; echo "BUS␣WIDTH:" $i "WEIGHT:" $RY; done
```

5 Code Documentation

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

- **axis_moving_average** AXIS moving average core.
- **tb_axis** Verilog test bench.
- **tb_cocotb verilog** Verilog test bench base for cocotb.
- **tb_cocotb python** cocotb unit test functions.

axis_moving_average.v

AUTHORS

JAY CONVERTINO

DATES

2023/02/01

INFORMATION

Brief

AXIS moving average for unsigned numbers.

License MIT

axis_moving_average

```
module axis_moving_average #(
parameter
BUS_WIDTH
 =
1,
parameter
WEIGHT
 =
1
) ( input aclk, input arstn, output [8*BUS_WIDTH-1:0] m_axis_tdata, output r
```

AXIS moving average for unsigned numbers.

Parameters

BUS_WIDTH parameter — Width of the BUS in bytes.

WEIGHT parameter — How many elements, rounded to a power of two, to accumulate.

Ports

aclk	Clock for AXIS
arstn	Negative reset for AXIS
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.
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.

VARIABLES

m_axis_tdata

```
assign m_axis_tdata = r_accumulator[(
                                                                8*BUS_WIDTH+
c_WEIGHT_POWER
                                                                -
1
):c_WEIGHT_POWER]
```

Trim and shift data to get amount, this is the divide out.

m_axis_tvalid

```
assign m_axis_tvalid = r_always_valid
```

Single clock edge valid

s_axis_tready

```
assign s_axis_tready = m_axis_tready
```

We are ready if the destination is ready

tb_axis.v

AUTHORS

JAY CONVERTINO

DATES

2024/12/11

INFORMATION

Brief

Test bench for axis_moving_average using axis stim and clock stim.

License MIT

tb_axis

```
module tb_axis
```

Test bench for axis_moving_average. 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.

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("random.bin")
) slave_axis_stim ( .m_axis_aclk(tb_stim_clk), .m_axis_arstn(tb_stim_rstn),
```

Device under test SLAVE stimulus module.

dut

```
 axis_moving_average #(
                                                                                .
 BUS_WIDTH(BUS_WIDTH),
                                                                                .
 WEIGHT(8)
) dut ( .aclk(tb_stim_clk), .arstn(tb_stim_rstn), .m_axis_tdata(tb_dut_data)
```

Device under test, axis_moving_average

slave_axis_stim

Device under test SLAVE 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
BUS_WIDTH
 =
1,
parameter
WEIGHT
 =
1
) ( input aclk, input arstn, output [(BUS_WIDTH*8)-1:0] m_axis_tdata, output
```

Test bench for axis moving average. 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

BUS_WIDTH parameter	Width of the bus input/output
WEIGHT parameter	Divisor for moving average, rounded to the highest power of two.

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.
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.

INSTANTIATED MODULES

dut

```
axis_moving_average #(
                                                                        .
BUS_WIDTH(BUS_WIDTH),
                                                                        .
WEIGHT(WEIGHT)
) dut ( .aclk(aclk), .arstn(arstn), .m_axis_tdata(m_axis_tdata), .m_axis_tva
```

Device under test, axis_moving_average

tb_cocotb.py

AUTHORS

JAY CONVERTINO

DATES

2024/12/09

INFORMATION

Brief

Cocotb test bench

License MIT

FUNCTIONS

moving_average

```
async def moving_average(
dut
)
```

Emulate verilog moving average function for unsigned numbers only, this is a coroutine that runs at the same time as the main.

Parameters

dut device under test from cocotb test.

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_rand_ready

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

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.