AXIS_MOVING_AVERAGE

November 21, 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 5 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
  - '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.

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_moving_average** AXIS Moving Average to uP converter

The next sections document the module in great detail.

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 = s_axis_tvalid
```

Single clock edge valid

s_axis_tready

```
assign s_axis_tready = m_axis_tready
```

We are ready if the destination is ready