AXIS SPI MASTER

April 30, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

The intent of this core is to provide a base AXIS to SPI Master interface. It is capable of back to back transfers with zero wait time. The data can be output at any rate up to half the input clock. The core SPI clock is generated for external use only and should NOT be routed into any logic. This device also does the chip selection based on the current activity of the core. CPOL/CPHA can be altered at anytime.

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:clock:mod_clock_ena_gen:1.1.1
  - AFRL:utility:helper:1.0.0
  - AFRL:simple:piso:1.0.0
  - AFRL:simple:sipo:1.0.0
- dep_tb
  - AFRL:simulation:axis_stimulator
  - AFRL:utility:sim_helper
  - AFRL:simulation:clock_stimulator

1.3 In a Project

This core connects a SPI to the AXIS bus. Meaning this is a streaming device only. Connect the MOSI/MISO to the SPI device in question and connect the AXIS to its intended endpoints.

2 Architecture

The core for this contains the following:

- **axis_spi** Interface with SPI to AXIS interface.
- **mod_clock_ena_gen** Generate an enable used to sample data for piso/sipo.
- **piso** Take parallel data and output it in serial.
- **sipo** Take serial data and output it in parallel.

The main core is made to interface a AXIS bus to the SPI bus. This is done using the SIPO and PISO cores to change from serial to parallel data streams. In addition mod clock enable gen cores create the negative and positive enables based upon the input clock and set rate to sample the data. This is then glued together in the core with some logic to output the approtpriate SPI signals, including a generated clock. This generated clock is created by the mod clock gen enables only and is NOT used to clock any internal signals. Use only as a output clock. The core allows for any rate to be used up to half the input clock. The clock phase and polarity can be changed on the fly at anytime. All word transfers are the size of the AXIS bus. If multiple byte transfers of varing sizes are needed. It is recommened to set this to one byte width for the AXIS data bus and do back to back transfers for the number needed. Basically having data available to the core as soon as it can get it means there will be no gap in the spi output.

2.1 Ports

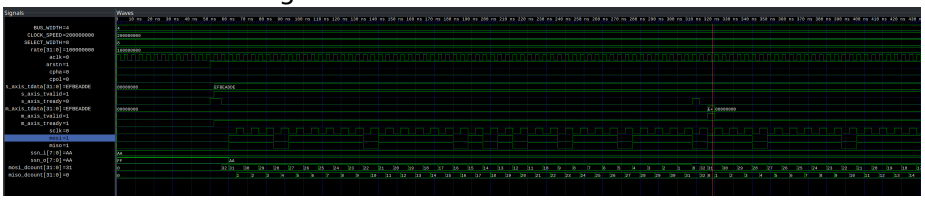
A port list is availabe, with specific signal information, in the 5. The generalized idea is a AXIS slave input for MOSI data. A AXIS master output for MISO data, and a master SPI interface with sclk, mosi, miso, and ss_n signals. The dcount outputs give insight into the status of the in/out data bits of the core.

2.2 Waveforms

The idealized simulation waveforms are shown below. The values reflect the results of using the icarus backend with GTKwaveform view tools.

Back to back transfers will naturally occur if data is available in time for the next ready. This allows for zero intertransmission gaps.

Figure 1: CPOL = 0 : CPHA = 0



The following figures are not back to back transfers. Meaning there is a some gap in the time a new word is available. This could be done on purpose due to SPI slave needs and can easily done by looking at the counter values of the core.

Figure 2: CPOL = 0 : CPHA = 0

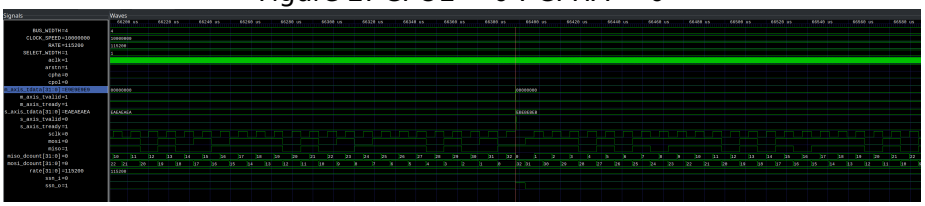


Figure 3: CPOL = 0 : CPHA = 1

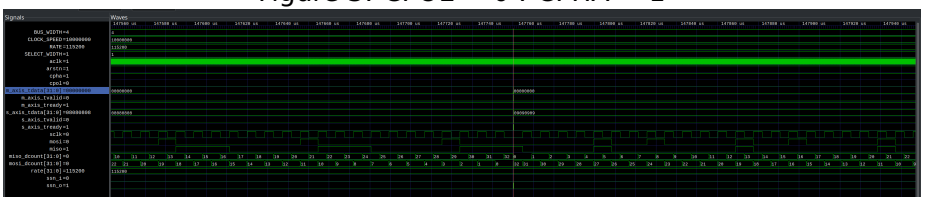


Figure 4: CPOL = 1 : CPHA = 0

Figure 5: CPOL = 1 : CPHA = 1

3 Building

The AXIS SPI is written in Verilog 2001. It 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_spi_master.v
- tb
  - tb/tb_spi.v
- 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.

- sim

  Info: Base simulation using icarus as default.

- sim_rand_data

  Info: Use random data as sim input.

- sim_8bit_count_data

  Info: Use counter data as sim input.

- 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
   - **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.

- **sim** Standard simulation of SPI looped, input/output verification.

This uses a axis stimulator cores for master/slave. This will run all the data in the slave axis SPI interface, which will output the data over the SPI interface. This is then looped into the SPI input that then puts the valid data out on the master axis SPI interface.

4.2 cocotb

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

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

Each module has a cocotb based simulation. These use the cocotb extensions made by Alex. The two extensions used are cocotbext-axi and cocotbext-spi. These provide outside verification of the implimentation. These tests consist of the following fusesoc targets.

- **sim_cocotb** Standard simulation of SPI data to and from cocotbexts this tests all CPOL/CPHA options.

Then you must use the cocotb sim target. The targets above can be run with the following:

```
$ fusesoc run --target sim_cocotb AFRL:device_converter:axis_spi:1.0.0
```

5 Module Documentation

- **axis_spi** Interfaces AXIS to SPI.
- **tb_spi** Verilog test bench.
- **tb_cocotb verilog** Verilog test bench base for cocotb.
- **tb_cocotb python** cocotb unit test functions.

axis_spi_master.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/22

INFORMATION

Brief

Stream SPI input/output data over AXIS bus.

License MIT

axis_spi_master

```
module axis_spi_master #(
parameter
CLOCK_SPEED
 =
2000000,
parameter
BUS_WIDTH
 =
4,
parameter
SELECT_WIDTH
 =
8
) ( input aclk, input arstn, input [BUS_WIDTH*8-1:0] s_axis_tdata, input s_a
```

SPI core with axis input/output data. Read/Write is size of BUS_WIDTH bytes. Write activates core for read.

Parameters

CLOCK_SPEED parameter	This is the aclk frequency in Hz, this is the the frequency used for the bus and is divided by the rate.
BUS_WIDTH parameter	AXIS data width in bytes.
SELECT_WIDTH parameter	Bit width of the slave select.

Ports

aclk	Clock for AXIS
arstn	Negative reset for AXIS
s_axis_tdata	Input data for SPI MOSI.
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 from SPI MISO
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.
sclk	spi clock, should only drive output pins to devices.
mosi	transmit for master output
miso	receive for master input
ssn_i	slave select input
ssn_o	slave select output
rate	output rate of spi core.
cpol	clock polarity of sclk
cpha	clock phase of sclk
miso_dcount	Current number of input bits available from parallel register.
mosi_dcount	current number of output bits available to serial shift output.

STATE MACHINE

Constants that makeup the data_state machine.

ready

```
localparam ready = 3'd1
```

ready and waiting for data

processing

```
localparam processing = 3'd3
```

data is being processed

error

```
localparam error = 3'd0
```

someone made a whoops

INSTANTIATED MODULES

inst_spi_output_clk

```
mod_clock_ena_gen #(
                                                                                .
CLOCK_SPEED(CLOCK_SPEED)
) inst_spi_output_clk ( .clk(aclk), .rstn(arstn), .start0(1'b0), .clr(spi_e
```

Generates enable at rate for spi output data.

inst_spi_input_clk

```
mod_clock_ena_gen #(
                                                                                .
CLOCK_SPEED(CLOCK_SPEED)
) inst_spi_input_clk ( .clk(aclk), .rstn(arstn), .start0(1'b1), .clr(spi_er
```

Generates enable at rate for spi input data.

inst_piso

```
piso #(
                                                                                .
BUS_WIDTH(BUS_WIDTH)
) inst_piso ( .clk(aclk), .rstn(arstn), .ena(spi_ena_mosi), .load(spi_mosi_
```

take axis input parallel data at bus size, and output the word to the spi bus.

inst_sipo

```
sipo #(
                                                                                .
BUS_WIDTH(BUS_WIDTH)
) inst_sipo ( .clk(aclk), .rstn(arstn), .ena(spi_ena_miso), .load(spi_miso_
```

take serial input data, and output the world to the parallel data bus.

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_00

```
@cocotb.test()
async def single_word_00(
dut
)
```

Coroutine that is identified as a test routine. This routine tests for writing a single word, and then reading a single word for cpol == 0 and cpha == 0.

Parameters

dut Device under test passed from cocotb.

single_word_10

```
@cocotb.test()
async def single_word_10(
dut
)
```

Coroutine that is identified as a test routine. This routine tests for writing a single word, and then reading a single word for cpol == 1 and cpha == 0.

Parameters

dut Device under test passed from cocotb.

single_word_01

```
@cocotb.test()
async def single_word_01(
dut
)
```

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

single word for cpol == 0 and cpha == 1.

Parameters

dut Device under test passed from cocotb.

single_word_11

```
@cocotb.test()
async def single_word_11(
dut
)
```

Coroutine that is identified as a test routine. This routine tests for writing a single word, and then reading a single word for cpol == 1 and cpha == 1.

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.

tb_cocotb.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/24

INFORMATION

Brief

Test bench wrapper for cocotb

License MIT

tb_cocotb

```
module tb_cocotb #(
parameter
CLOCK_SPEED
 =
2000000,
parameter
BUS_WIDTH
 =
4,
parameter
SELECT_WIDTH
 =
1,
parameter
```

```
 RATE
  =
 115200
) ( input aclk, input arstn, input [BUS_WIDTH*8-1:0] s_axis_tdata, input s_a
```

SPI core with axis input/output data. Read/Write is size of BUS_WIDTH bytes. Write activates core for read.

Parameters

CLOCK_SPEED parameter	This is the aclk frequency in Hz, this is the the frequency used for the bus and is divided by the rate.
BUS_WIDTH parameter	AXIS data width in bytes.
SELECT_WIDTH parameter	Bit width of the slave select.
RATE parameter	Select the data rate of the spi core.

Ports

aclk	Clock for AXIS
arstn	Negative reset for AXIS
s_axis_tdata	Input data for UART TX.
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 from UART RX
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.
sclk	spi clock, should only drive output pins to devices.
mosi	transmit for master output
miso	receive for master input
ssn_i	slave select input
ssn_o	slave select output
rate parameter	output rate of spi core.
cpol	clock polarity of spi_clk
cpha	clock phase of spi_clk
miso_dcount	Current number of input bits available from parallel register.
mosi_dcount	current number of output bits available to serial shift output.

INSTANTIATED MODULES

dut

```
axis_spi_master #(
.CLOCK_SPEED(CLOCK_SPEED),
.BUS_WIDTH(BUS_WIDTH),
.SELECT_WIDTH(SELECT_WIDTH)
```

```
) dut ( .aclk(aclk), .arstn(arstn), .s_axis_tdata(s_axis_tdata), .s_axis_tv
```

Device under test, axis_spi_master

tb_spi.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/22

INFORMATION

Brief

Test bench for AXIS SPI

License MIT