BUS SPI MASTER

August 1, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

BUS SPI Master core emulates and extends the Altera SPI IP core. This is a SPI master device only. It is currently available as a AXI Lite, Wishbone Standard, and uP bus IP core. The Altera core this is based on has Linux and uboot drivers, by mimicking it this core has instant access to its support software. There is a extension register for control that allows for changes to the SPI Master the Altera IP does not have. The CPOL/CPHA can be changed. A new speed registers controls the speed of the device outside of the initial setting. The following is information on how to use the device in an FPGA, software, and in simulation.

1.2 Dependencies

The following are the dependencies of the cores.

- fusesoc 2.X
- iverilog (simulation)
- cocotb (simulation)

1.2.1 axi_lite_spi_master Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device:up_spi_master:1.0.0
  - AD:common:up_axi:1.0.0
- dep_tb
  - AFRL:simulation:axis_stimulator
  - AFRL:utility:sim_helper

1.2.2 wishbone_standard_spi_master Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device:up_spi_master:1.0.0
  - AFRL:bus:up_wishbone_standard:1.0.0

1.2.3 up_spi_master Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device_converter:axis_spi_master:1.0.0
  - AFRL:buffer:axis_fifo:1.0.0

1.3 In a Project

First, pick a core that matches the target bus in question. Then connect the BUS SPI MASTER core to that bus. Once this is complete the SPI pins will need to be routed to the slave SPI devices. The core can be used with a 32 bit or 16 bit databus (4 or 2 bytes). Any other size is not supported and will result in a core that acts strange or don't build at all.

2 Architecture

This core is made up of other cores that are documented in detail in their source. The cores this is made up of are:

- **axis_spi_master** Interface with SPI master and present the data over AXIS interface (see core for documentation).
- **up_axi** An AXI Lite to uP converter core (see core for documentation).
- **up_wishbone_standard** A wishbone standard to uP converter core (see core for documentation).
- **up_spi_master** Takes uP bus and coverts it for interfacing with the AXIS SPI core (see module documentation for information 5).

2.1 Registers

For register bit documentation please see up_spi_master subsetion registers in 5

Interrupts for this core are enabled in the control register. First the general error IE bit, interrupt enable for all errors, is set to 1. All errors will now generate a interrupt. IOE, IROE, etc will not need to be activated. This interrupt goes active high (1) when a condition becomes true. Starting with the interrupt end of packet bit (IEOP), setting this active will enable the interrupt to go off when the status EOP bit is true. This will stay that way till the tx or rx register is cleared of the

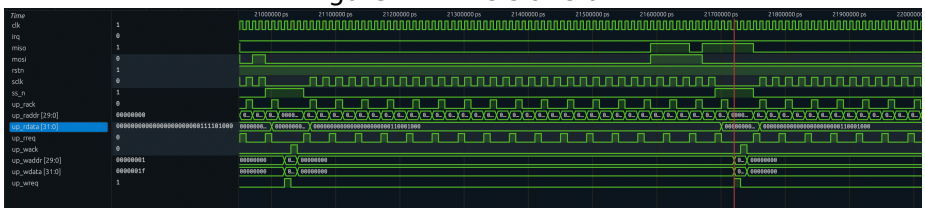
EOP word. Interrupt read ready (IRRDY) when set active will trigger an interrupt when the status bit RRDY is active. This will stay tripped till a word is read. Interrupt transmit ready (ITRDY) when set active will trigger an interrupt when the status bit TRDY is active. This will stay tripped till a word is written. Interrupt transmit overrun (ITOE) when set active will trigger an interrupt when the status bit TOE is active. This will stay tripped till the status register is written. Interrupt receive underrun (IROE) when set active will trigger an interrupt when the status bit ROE is active. ITOE, TOE, IROE, ROE, and E are only cleared by writing to the status register. The status register does NOT write any actual data to the register, status bits are not directly affected. It simply resets ROE/TOE to 0. See 5 up_spi_master for more detail on the location and function of register bits.

2.2 Waveforms

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

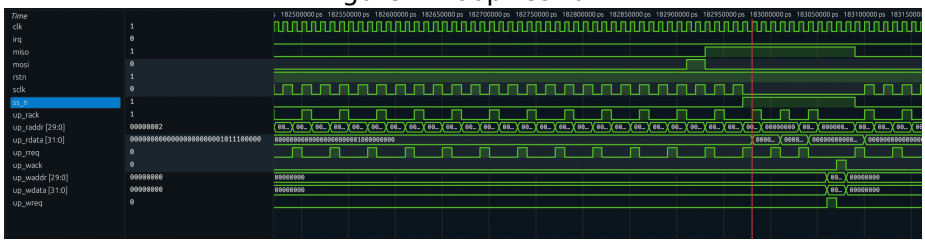
Write slave is the first uP test that looks at writing to a slave device using the SPI master without reading the received data.

Figure 1: write slave uP



Loop test checks for reads and writes on the uP bus.

Figure 2: loop test uP



Next are various interrupt enable tests to see if they respond as they should.

Figure 3: read ready interrupt enabled

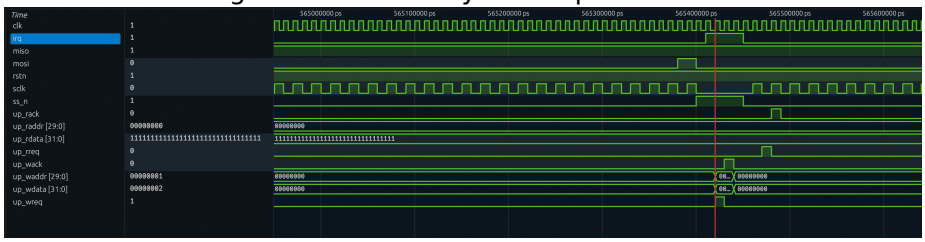
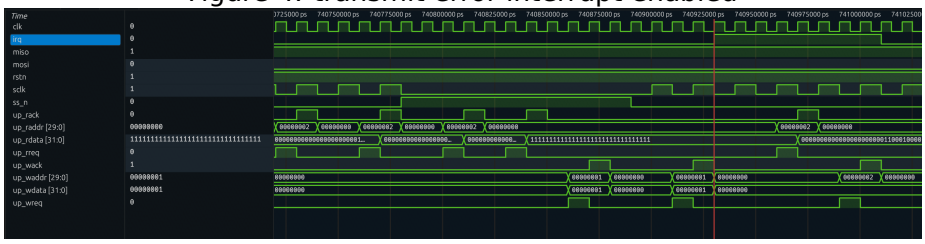
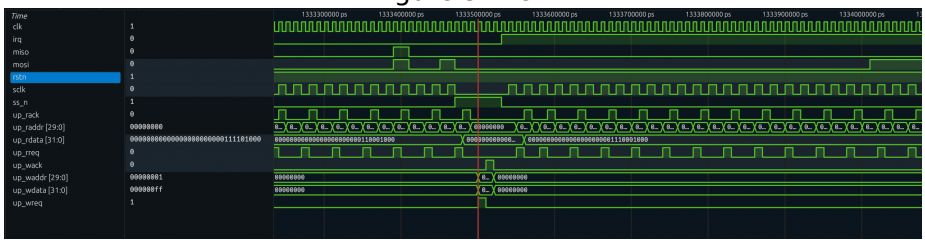


Figure 4: transmit error interrupt enabled



End of packet is a feature added to the Altera IP core in 2019. It is not present in the linux drivers. Its functionality is questionable when it comes to being useful. Essentially it allows the device to signal bits or a irq if a EOP word matches the transmit or receive register contents. It is cleared when those contents change and are not the EOP.

Figure 5: EOP



The last two waveforms show the uP to bus adapters in a loopback test. Since it takes two writes to get the SPI slave to echo a value written in two previous writes, the read value will be two writes behind.

Figure 6: loop test AXI Lite

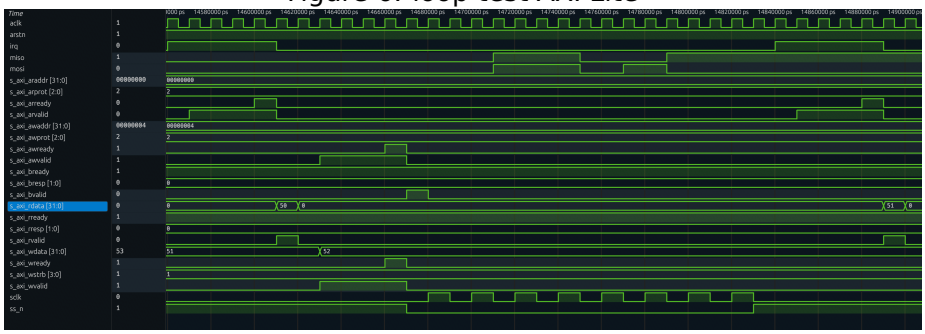
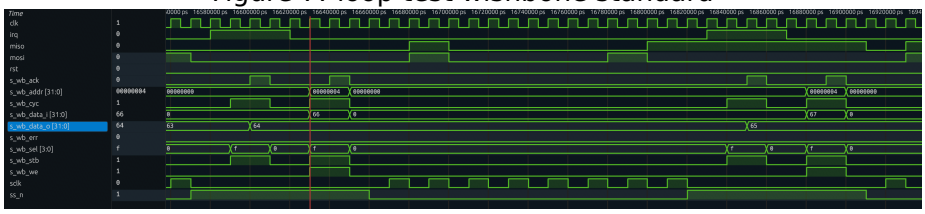


Figure 7: loop test wishbone standard



3 Building

The BUS SPI MASTER is written in Verilog 2001. It should synthesize in any modern FPGA software. It 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 axi_lite_spi_master File List

- src
  - src/axi_lite_spi_master.v
- tb_cocotb
  - 'tb/tb_cocotb_axi_lite.py': 'file_type': 'user', 'copyto': '.'
  - 'tb/tb_cocotb_axi_lite.v': 'file_type': 'verilogSource'
- tb
  - tb/tb_uart.v

3.2.2 wishbone_standard_spi_master File List

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

3.2.3 up_spi_master File List

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

3.3 Targets

3.3.1 axi_lite_spi_master Targets

- default

  Info: Default for IP intergration.

- lint

Info: Lint with Verible

- sim_cocotb

  Info: Cocotb unit tests

3.3.2 wishbone_standard_spi_master Targets

- default

  Info: Default for IP intergration.

- lint

  Info: Lint with Verible

- sim_cocotb

  Info: Cocotb unit tests

3.3.3 up_spi_master Targets

- default

  Info: Default for IP intergration.

- lint

  Info: Lint with Verible

- 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 cocotb

Cocotb is the only method for simulating the various interations of the bus_spi_master core. At the moment there is a axi_lite, wishbone_standard, and uP based versions. This is currently set to use icarus as the sim tool for cocotb. The uP testbench is the one that will test all the various register bits and interrupt options. Others will only do loop back tests.

To run the wishbone sim use the command below.

```
fusesoc run --target sim_cocotb AFRL:device:wishbone_standard_spi_master:1.0.0
```

To run the axi_lite sim use the command below.

```
fusesoc run --target sim_cocotb AFRL:device:axi_lite_spi_master:1.0.0
```

To run the uP sim use the command below.

```
fusesoc run --target sim_cocotb AFRL:device:up_spi_master:1.0.0
```

5 Module Documentation

up_spi_master is the module that integrates the AXIS SPI MASTER core. This uses inputs/outputs for data tied directly to registers mapped in the uP bus. The uP bus is the microprocessor bus based on Analog Devices design. It resembles a APB bus in design, and is the bridge to other buses BUS SPI MASTER can use. This makes changing for AXI Lite, to Wishbone to whatever quick and painless.

axi_lite_spi module adds a AXI Lite to uP (microprocessor) bus converter. The converter is from Analog Devices.

wishbone_standard_spi module adds a Wishbone Standard to uP (microprocessor) bus converter. This converter was designed for Wishbone Standard only, NOT pipelined or Registered (cti).

The next sections document these modules. up_spi_master contains the register map explained, and what the various bits do.

axi_lite_spi_master.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/30

INFORMATION

Brief

AXI Lite SPI Master is a core for interfacing with SPI Slave devices.

License MIT

axi_lite_spi_master

```
module axi_lite_spi_master #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
WORD_WIDTH
 =
4,
parameter
```

```
 CLOCK_SPEED
  =
 100000000,
 parameter
 SELECT_WIDTH
  =
 16,
 parameter
 DEFAULT_RATE_DIV
  =
 0,
 parameter
 DEFAULT_CPOL
  =
 0,
 parameter
 DEFAULT_CPHA
  =
 0,
 parameter
 FIFO_ENABLE
  =
 0
)
                                                                    (
 input
 wire
 aclk,
 input
 wire
 arstn,
 input
 wire
 s_axi_awvalid,
 input
 wire
  [ADDRESS_WIDTH-1:0]
 s_axi_awaddr,
 input
 wire
  [ 2:0]
 s_axi_awprot,
 output
 wire
 s_axi_awready,
 input
 wire
 s_axi_wvalid,
 input
 wire
  [(BUS_WIDTH*8)-1:0]
 s_axi_wdata,
 input
 wire
  [ 3:0]
 s_axi_wstrb,
 output
 wire
 s_axi_wready,
 output
 wire
 s_axi_bvalid,
 output
 wire
  [ 1:0]
 s_axi_bresp,
```

```
 input
 wire
 s_axi_bready,
 input
 wire
 s_axi_arvalid,
 input
 wire
  [ADDRESS_WIDTH-1:0]
 s_axi_araddr,
 input
 wire
  [ 2:0]
 s_axi_arprot,
 output
 wire
 s_axi_arready,
 output
 wire
 s_axi_rvalid,
 output
 wire
  [(BUS_WIDTH*8)-1:0]
 s_axi_rdata,
 output
 wire
  [ 1:0]
 s_axi_rresp,
 input
 wire
 s_axi_rready,
 output
 wire
 irq,
 output
 wire
 sclk,
 output
 wire
 mosi,
 input
 wire
 miso,
 output
 wire
  [SELECT_WIDTH-1:0]
 ss_n
)
```

AXI Lite based SPI Master device. BUS_WIDTH is 4 bytes.

Parameters

ADDRESS_WIDTH parameter	Width of the uP address port, max 32 bit.
BUS_WIDTH parameter	Width of the uP bus data port, only valid values are 2 or 4.
WORD_WIDTH parameter	Width of each SPI Master word. This will also set the bits used in the TX/RX data registers. Must be less than or equal to BUS_WIDTH. VALID: 1 to 4.
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.
SELECT_WIDTH parameter	Bit width of the slave select, defaults to 16 to match altera spi ip.

DEFAULT_RATE_DIV parameter	Default divider value of the main clock to use for the spi data output clock rate. 0 is 2 (2^(X+1) X is the DEFAULT_RATE_DIV)
DEFAULT_CPOL parameter	Default clock polarity for the core (0 or 1).
DEFAULT_CPHA parameter	Default clock phase for the core (0 or 1).
FIFO_ENABLE parameter	Enable a 16 word fifo for RX and TX. The chip select will stay asserted between words.

Ports

aclk input wire	Clock for all devices in the core
arstn input wire	Negative reset
s_axi_awvalid input wire	Axi Lite aw valid
s_axi_awaddr input wire [ADDRESS_WIDTH- 1:0]	Axi Lite aw addr
s_axi_awprot input wire [2:0]	Axi Lite aw prot
s_axi_awready output wire	Axi Lite aw ready
s_axi_wvalid input wire	Axi Lite w valid
s_axi_wdata input wire [(BUS_WIDTH* 8)- 1:0]	Axi Lite w data
s_axi_wstrb input wire [3:0]	Axi Lite w strb
s_axi_wready output wire	Axi Lite w ready
s_axi_bvalid output wire	Axi Lite b valid
s_axi_bresp output wire [1:0]	Axi Lite b resp
s_axi_bready input wire	Axi Lite b ready
s_axi_arvalid input wire	Axi Lite ar valid
s_axi_araddr input wire [ADDRESS_WIDTH- 1:0]	Axi Lite ar addr
s_axi_arprot input wire [2:0]	Axi Lite ar prot
s_axi_arready output wire	Axi Lite ar ready
s_axi_rvalid output wire	Axi Lite r valid
s_axi_rdata output wire [(BUS_WIDTH* 8)- 1:0]	Axi Lite r data
s_axi_rresp output wire [1:0]	Axi Lite r resp
s_axi_rready input wire	Axi Lite r ready
irq	Interrupt when data is received

```
output wire
```

sclk `output wire`	spi clock, should only drive output pins to devices.
mosi `output wire`	transmit for master output
miso `input wire`	receive for master input
ss_n `output wire [SELECT_WIDTH- 1:0]`	slave select output

up_rreq

```
wire up_rreq
```

uP read bus request

up_rack

```
wire up_rack
```

uP read bus acknowledge

up_raddr

```
wire [ADDRESS_WIDTH-(
BUS_WIDTH
                                                                    /
2
)-1:0] up_raddr
```

uP read bus address

up_rdata

```
wire [31:0] up_rdata
```

uP read bus request

up_wreq

```
wire up_wreq
```

uP write bus request

up_wack

```
wire up_wack
```

uP write bus acknowledge

up_waddr

```
wire [ADDRESS_WIDTH-(
BUS_WIDTH
                                                                    /
2
)-1:0] up_waddr
```

uP write bus address

up_wdata

```
wire [31:0] up_wdata
```

uP write bus data

INSTANTIANTED MODULES

inst_up_axi

Module instance of up_axi for the AXI Lite bus to the uP bus.

inst_up_spi_master

Module instance of up_spi_master creating a Logic wrapper for spi master axis bus cores to interface with uP bus.

wishbone_standard_spi_master.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/30

INFORMATION

Brief

Wishbone Standard SPI Master core.

License MIT

wishbone_standard_spi_master

```
module wishbone_standard_spi_master #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
WORD_WIDTH
 =
4,
parameter
```

```
 CLOCK_SPEED
  =
 100000000,
 parameter
 SELECT_WIDTH
  =
 16,
 parameter
 DEFAULT_RATE_DIV
  =
 0,
 parameter
 DEFAULT_CPOL
  =
 0,
 parameter
 DEFAULT_CPHA
  =
 0,
 parameter
 FIFO_ENABLE
  =
 0
)
                                                                    (
 input
 wire
 clk,
 input
 wire
 rst,
 input
 wire
 s_wb_cyc,
 input
 wire
 s_wb_stb,
 input
 wire
 s_wb_we,
 input
 wire
  [ADDRESS_WIDTH-1:0]
 s_wb_addr,
 input
 wire
  [BUS_WIDTH*8-1:0]
 s_wb_data_i,
 input
 wire
  [BUS_WIDTH-1:0]
 s_wb_sel,
 output
 wire
 s_wb_ack,
 output
 wire
  [BUS_WIDTH*8-1:0]
 s_wb_data_o,
 output
 wire
 s_wb_err,
 output
 wire
 irq,
 output
```

```
 wire
 sclk,
 output
 wire
 mosi,
 input
 wire
 miso,
 output
 wire
  [SELECT_WIDTH-1:0]
 ss_n
)
```

Wishbone Standard based SPI Master device.

Parameters

ADDRESS_WIDTH parameter	Width of the uP address port, max 32 bit.
BUS_WIDTH parameter	Width of the uP bus data port, only valid values are 2 or 4.
WORD_WIDTH parameter	Width of each SPI Master word. This will also set the bits used in the TX/RX data registers. Must be less than or equal to BUS_WIDTH. VALID: 1 to 4.
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.
SELECT_WIDTH parameter	Bit width of the slave select, defaults to 16 to match altera spi ip.
DEFAULT_RATE_DIV parameter	Default divider value of the main clock to use for the spi data output clock rate. 0 is 2 (2^(X+1) X is the DEFAULT_RATE_DIV)
DEFAULT_CPOL parameter	Default clock polarity for the core (0 or 1).
DEFAULT_CPHA parameter	Default clock phase for the core (0 or 1).
FIFO_ENABLE parameter	Enable a 16 word fifo for rx and tx. All words put into the fifo together will keep chip select low.

Ports

clk input wire	Clock for all devices in the core
rst input wire	Positive reset
s_wb_cyc input wire	Bus Cycle in process
s_wb_stb input wire	Valid data transfer cycle
s_wb_we input wire	Active High write, low read
s_wb_addr input wire [ADDRESS_WIDTH- 1:0]	Bus address
s_wb_data_i input wire [BUS_WIDTH* 8- 1:0]	Input data
s_wb_sel input wire [BUS_WIDTH- 1:0]	Device Select
s_wb_ack output wire	Bus transaction terminated

s_wb_data_o `output wire [BUS_WIDTH* 8- 1:0]`	Output data
s_wb_err `output wire`	Active high when a bus error is present
irq `output wire`	Interrupt when data is received
sclk `output wire`	spi clock, should only drive output pins to devices.
mosi `output wire`	transmit for master output
miso `input wire`	receive for master input
ss_n `output wire [SELECT_WIDTH- 1:0]`	slave select output

up_rreq

```
wire up_rreq
```

uP read bus request

up_rack

```
wire up_rack
```

uP read bus acknowledge

up_raddr

```
wire [ADDRESS_WIDTH-(
BUS_WIDTH
                                                                    /
2
)-1:0] up_raddr
```

uP read bus address

up_rdata

```
wire [31:0] up_rdata
```

uP read bus request

up_wreq

```
wire up_wreq
```

uP write bus request

up_wack

```
wire up_wack
```

uP write bus acknowledge

up_waddr

```
wire [ADDRESS_WIDTH-(
BUS_WIDTH
                                                                    /
2
)-1:0] up_waddr
```

uP write bus address

up_wdata

```
wire [31:0] up_wdata
```

uP write bus data

INSTANTIANTED MODULES

inst_up_wishbone_standard

Module instance of up_wishbone_standard for the Wishbone Classic Standard bus to the uP bus.

inst_up_spi_master

Module instance of up_spi_master creating a Logic wrapper for spi master axis bus cores to interface with uP bus.

up_spi_master.v

AUTHORS

JAY CONVERTINO

DATES

2024/04/29

INFORMATION

Brief

uP Core for interfacing with axis spi that emulates the ALTERA SPI IP in MASTER mode.

License MIT

up_spi_master

```
module up_spi_master #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
WORD_WIDTH
 =
4,
parameter
```

```
 CLOCK_SPEED
  =
 100000000,
 parameter
 SELECT_WIDTH
  =
 16,
 parameter
 DEFAULT_RATE_DIV
  =
 0,
 parameter
 DEFAULT_CPOL
  =
 0,
 parameter
 DEFAULT_CPHA
  =
 0,
 parameter
 FIFO_ENABLE
  =
 0
)
                                                                    (
 input
 wire
 clk,
 input
 wire
 rstn,
 input
 wire
 up_rreq,
 output
 wire
 up_rack,
 input
 wire
  [ADDRESS_WIDTH-(BUS_WIDTH/2)-1:0]
 up_raddr,
 output
 wire
  [(BUS_WIDTH*8)-1:0]
 up_rdata,
 input
 wire
 up_wreq,
 output
 wire
 up_wack,
 input
 wire
  [ADDRESS_WIDTH-(BUS_WIDTH/2)-1:0]
 up_waddr,
 input
 wire
  [(BUS_WIDTH*8)-1:0]
 up_wdata,
 output
 wire
 irq,
 output
 wire
 sclk,
 output
```

```
 wire
 mosi,
 input
 wire
 miso,
 output
 wire
  [SELECT_WIDTH-1:0]
 ss_n
)
```

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

Parameters

ADDRESS_WIDTH parameter	Width of the uP address port, max 32 bit.
BUS_WIDTH parameter	Width of the uP bus data port, only valid values are 2 or 4.
WORD_WIDTH parameter	Width of each SPI Master word. This will also set the bits used in the TX/RX data registers. Must be less than or equal to BUS_WIDTH, VALID: 1 to 4.
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.
SELECT_WIDTH parameter	Bit width of the slave select, defaults to 16 to match altera spi ip.
DEFAULT_RATE_DIV parameter	Default divider value of the main clock to use for the spi data output clock rate. 0 is 2 (2^(X+1) X is the DEFAULT_RATE_DIV)
DEFAULT_CPOL parameter	Default clock polarity for the core (0 or 1).
DEFAULT_CPHA parameter	Default clock phase for the core (0 or 1).
FIFO_ENABLE parameter	Enable a 16 word (byte) fifo for RX/TX. Not a standard part of the Altera IP core.

Ports

clk input wire	Clock for all devices in the core
rstn input wire	Negative reset
up_rreq input wire	uP bus read request
up_rack output wire	uP bus read ack
up_raddr input wire [ADDRESS_WIDTH-(BUS_WIDTH/ 2)- 1:0]	uP bus read address
up_rdata output wire [(BUS_WIDTH* 8)- 1:0]	uP bus read data
up_wreq input wire	uP bus write request
up_wack output wire	uP bus write ack
up_waddr input wire [ADDRESS_WIDTH-(BUS_WIDTH/ 2)- 1:0]	uP bus write address
up_wdata input wire [(BUS_WIDTH* 8)- 1:0]	uP bus write data

irq `output wire`	Interrupt when data is received
sclk `output wire`	spi clock, should only drive output pins to devices.
mosi `output wire`	transmit for master output
miso `input wire`	receive for master input
ss_n `output wire [SELECT_WIDTH- 1:0]`	slave select output

DIVISOR

```
localparam DIVISOR = BUS_WIDTH/2
```

Divide the address register default location for 1 byte access to multi byte access. (register offsets are byte offsets).

REG_SIZE

```
localparam REG_SIZE = 8
```

Number of bits for the register address

FIFO_DEPTH

```
localparam FIFO_DEPTH = 16
```

Depth of the fifo, matches UART LITE (xilinx), so I kept this just cause

REGISTER INFORMATION

Core has 7 registers at the offsets that follow when at a full 32 bit bus width, Internal address is OFFSET >> BUS_WIDTH/2 (32bit would be h4 >> 2 = 1 for internal address).

RX_DATA_REG	h00
TX_DATA_REG	h04
STATUS_REG	h08
CONTROL_REG	h0C
RESERVED	h10
SLAVE_SELECT_REG	h14
EOP_VALUE_REG	h18
CONTROL_EXT_REG	h1C
SPEED_EXT_REG	h20

RX_DATA_REG

```
localparam RX_DATA_REG = 8'h0 >> DIVISOR
```

Defines the address offset for RX DATA OUTPUT

RX DATA REGISTER	
31:N	N:0
UNUSED	RECEIVED DATA

Valid bits are from WORD_WIDTH*8-1:0, which are data.

TX_DATA_REG

```
localparam TX_DATA_REG = 8'h4 >> DIVISOR
```

Defines the address offset to write the TX DATA INPUT.

TX DATA REGISTER	
31:N	N:0
UNUSED	TRANSMIT DATA

Valid bits are from WORD_WIDTH*8-1:0, which are data.

STATUS_REG

```
localparam STATUS_REG = 8'h8 >> DIVISOR
```

Defines the address offset to read the status bits.

STATUS REGISTER								
31:10	9	8	7	6	5	4	3	2:0
UNUSED	EOP	E	RRDY	TRDY	TMT	TOE	ROE	UNUSED

Status Register, 1 is considered active.

EOP 9, This bit is active(1) when the EOP_VALUE_REG is equal to RX_DATA_REG or TX_DATA_REG.

E 8, Logical or of TOE and ROE (Clear by writing status).

RRDY 7, Receive is ready (full) when the bit is 1, empty when the bit is 0.

TRDY 6, Transmit is ready (empty) when the bit is 1, full when the bit is 0.

TMT 5, Transmit shift register empty is set to 1 when all bits have been output.

TOE 4, Transmit overrun is set to 1 when a TX_DATA_REG write happens whne TRDY is 1 (Clear by writing status reg).

ROE 3, Receive overrun is set to 1 when RRDY is 1 and a new received word is going to be written to RX_DATA_REG (Clear by writing status reg)

CONTROL_REG

```
localparam CONTROL_REG = 8'hC >> DIVISOR
```

Defines the address offset to set the control bits.

CONTROL REGISTER									
31:11	10	9	8	7	6	5	4	3	2:0
UNUSED	SSO	IEOP	IE	IRRDY	ITRDY	UNUSED	ITOE	IROE	UNUSED

Control Register, 1 is considered active. **All zeros on reset.**

SSO 10, Setting this to 1 will force all ss_n lines to 0 (selected).

IEOP 9, Generate a interrupt on EOP status bit going active if set to 1.

IE 8, Generate a interrupt on ANY error, active if set to 1.

IRRDY 7, Generate a interrupt on RRDY status bit going active if set to 1.

ITRDY 6, Generate a interrupt on TRDY status bit going active if set to 1.

ITOE 4, Generate a interrupt on TOE status bit going active if set to 1.

IROE 3, Generate a interrupt on ROE status bit going active if set to 1.

RESERVED

```
localparam RESERVED = 8'h10 >> DIVISOR
```

Defines the address offset that is not used.

SLAVE_SELECT_REG

```
localparam SLAVE_SELECT_REG = 8'h14 >> DIVISOR
```

Defines the address offset to set the slave select value

SLAVE SELECT REGISTER	
31:N	N:0
UNUSED	SLAVE SELECT

Valid bits are from SELECT_WIDTH-1:0, which are the slave select output lines to drive low during data transmission.

EOP_VALUE_REG

```
localparam EOP_VALUE_REG = 8'h18 >> DIVISOR
```

Defines the address offset to set the end of packet match value

EOP REGISTER	
31:N	N:0
UNUSED	EOP

Valid bits are from BUS_WIDTH*8:0, which are used to check for a word match between rx and/or tx and update status.

CONTROL_EXT_REG

```
localparam CONTROL_EXT_REG = 8'h1C >> DIVISOR
```

Defines the address offset for control register extensions

CONTROL REGISTER EXTENDED		
31:2	1	0
UNUSED	CPHA	CPOL

Control Extension to add capabilities to Altera IP core.

CPHA 1, Clock Phase Bit, 0 or 1 per SPI specs (default value set by IP parameter).

CPOL 0, Clock Polarity bit, 0 or 1 per SPI specs (default value set by IP parameter).

SPEED_EXT_REG

```
localparam SPEED_EXT_REG = 8'h20 >> DIVISOR
```

Defines the address offset for speed control reg extension

SPEED CONTROL REGISTER
31:0
SPI OUTPUT CLOCK IN HZ

Valid bits are from BUS_WIDTH*8-1:0, which is the speed of the spi core in HZ.

INSTANTIATED MODULES

inst_axis_spi

SPI Master instance with AXIS interface

inst_axis_tx_fifo

SPI trasnmit data fifo.

inst_axis_rx_fifo

SPI received data fifo.

tb_cocotb_wishbone_standard.py

AUTHORS

JAY CONVERTINO

DATES

2025/04/30

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.

loop_data

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

Coroutine that is identified as a test routine. Use echo slave to loop data, check write wishbone equals spi slave contents, bus writes equal bus reads.

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_wishbone_standard.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/30

INFORMATION

Brief

Test bench wrapper for cocotb

License MIT

tb_cocotb

```
module tb_cocotb #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
WORD_WIDTH
 =
4,
parameter
```

```
 CLOCK_SPEED
  =
 100000000,
 parameter
 SELECT_WIDTH
  =
 16,
 parameter
 DEFAULT_RATE_DIV
  =
 0,
 parameter
 DEFAULT_CPOL
  =
 0,
 parameter
 DEFAULT_CPHA
  =
 0
)
                                                                    (
 input
 clk,
 input
 rst,
 input
 s_wb_cyc,
 input
 s_wb_stb,
 input
 s_wb_we,
 input
  [ADDRESS_WIDTH-1:0]
 s_wb_addr,
 input
  [BUS_WIDTH*8-1:0]
 s_wb_data_i,
 input
  [BUS_WIDTH-1:0]
 s_wb_sel,
 output
 s_wb_ack,
 output
  [BUS_WIDTH*8-1:0]
 s_wb_data_o,
 output
 s_wb_err,
 output
 irq,
 output
 sclk,
 output
 mosi,
 input
 miso,
 output
  [SELECT_WIDTH-1:0]
 ss_n
)
```

Wishbone Standard based SPI Master device.

Parameters

ADDRESS_WIDTH Width of the uP address port, max 32 bit.
parameter

BUS_WIDTH parameter	Width of the uP bus data port(can not be less than 2 bytes, max tested is 4).
WORD_WIDTH parameter	Width of each SPI Master word. This will also set the bits used in the TX/RX data registers. Must be less than or equal to BUS_WIDTH 1 to 4.
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.
SELECT_WIDTH parameter	Bit width of the slave select, defaults to 16 to match altera spi ip.
DEFAULT_RATE_DIV parameter	Default divider value of the main clock to use for the spi data output clock rate. 0 is 2 (2^(X+1) X is the DEFAULT_RATE_DIV)
DEFAULT_CPOL parameter	Default clock polarity for the core (0 or 1).
DEFAULT_CPHA parameter	Default clock phase for the core (0 or 1).

Ports

clk input	Clock for all devices in the core
rst input	Positive reset
s_wb_cyc input	Bus Cycle in process
s_wb_stb input	Valid data transfer cycle
s_wb_we input	Active High write, low read
s_wb_addr input [ADDRESS_WIDTH- 1:0]	Bus address
s_wb_data_i input [BUS_WIDTH* 8- 1:0]	Input data
s_wb_sel input [BUS_WIDTH- 1:0]	Device Select
s_wb_ack output [BUS_WIDTH- 1:0]	Bus transaction terminated
s_wb_data_o output [BUS_WIDTH* 8- 1:0]	Output data
s_wb_err output [BUS_WIDTH* 8- 1:0]	Active high when a bus error is present
irq output [BUS_WIDTH* 8- 1:0]	Interrupt when data is received
sclk output [BUS_WIDTH* 8- 1:0]	spi clock, should only drive output pins to devices.
mosi output [BUS_WIDTH* 8- 1:0]	transmit for master output
miso input [BUS_WIDTH* 8- 1:0]	receive for master input
ss_n output [SELECT_WIDTH- 1:0]	slave select output

INSTANTIATED MODULES

dut

Device under test, wishbone_standard_spi_master

tb_cocotb_axi_lite.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/04

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.

loop_data

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

Coroutine that is identified as a test routine. Use echo slave to loop data, check write axi equals spi slave contents, axi writes equal axi reads.

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_axi_lite.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/30

INFORMATION

Brief

Test bench wrapper for cocotb

License MIT

tb_cocotb

```
module tb_cocotb #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
WORD_WIDTH
 =
4,
parameter
```

```
 CLOCK_SPEED
  =
 100000000,
 parameter
 SELECT_WIDTH
  =
 16,
 parameter
 DEFAULT_RATE_DIV
  =
 0,
 parameter
 DEFAULT_CPOL
  =
 0,
 parameter
 DEFAULT_CPHA
  =
 0
)
                                                                    (
 input
 aclk,
 input
 arstn,
 input
 s_axi_awvalid,
 input
  [ADDRESS_WIDTH-1:0]
 s_axi_awaddr,
 input
  [ 2:0]
 s_axi_awprot,
 output
 s_axi_awready,
 input
 s_axi_wvalid,
 input
  [(BUS_WIDTH*8)-1:0]
 s_axi_wdata,
 input
  [ 3:0]
 s_axi_wstrb,
 output
 s_axi_wready,
 output
 s_axi_bvalid,
 output
  [ 1:0]
 s_axi_bresp,
 input
 s_axi_bready,
 input
 s_axi_arvalid,
 input
  [ADDRESS_WIDTH-1:0]
 s_axi_araddr,
 input
  [ 2:0]
 s_axi_arprot,
 output
 s_axi_arready,
 output
 s_axi_rvalid,
 output
  [(BUS_WIDTH*8)-1:0]
```

```
 s_axi_rdata,
 output
  [ 1:0]
 s_axi_rresp,
 input
 s_axi_rready,
 output
 irq,
 output
 sclk,
 output
 mosi,
 input
 miso,
 output
  [SELECT_WIDTH-1:0]
 ss_n
)
```

AXI Lite based SPI Master device.

Parameters

ADDRESS_WIDTH parameter	Width of the uP address port, max 32 bit.
BUS_WIDTH parameter	Width of the uP bus data port, only valid values are 2 or 4.
WORD_WIDTH parameter	Width of each SPI Master word. This will also set the bits used in the TX/RX data registers. Must be less than or equal to BUS_WIDTH 1 to 4.
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.
SELECT_WIDTH parameter	Bit width of the slave select, defaults to 16 to match altera spi ip.
DEFAULT_RATE_DIV parameter	Default divider value of the main clock to use for the spi data output clock rate. 0 is 2 (2^(X+1) X is the DEFAULT_RATE_DIV)
DEFAULT_CPOL parameter	Default clock polarity for the core (0 or 1).
DEFAULT_CPHA parameter	Default clock phase for the core (0 or 1).

Ports

aclk input	Clock for all devices in the core
arstn input	Negative reset
s_axi_awvalid input	Axi Lite aw valid
s_axi_awaddr input [ADDRESS_WIDTH- 1:0]	Axi Lite aw addr
s_axi_awprot input [2:0]	Axi Lite aw prot
s_axi_awready output [2:0]	Axi Lite aw ready
s_axi_wvalid input [2:0]	Axi Lite w valid
s_axi_wdata input [(BUS_WIDTH* 8)- 1:0]	Axi Lite w data

s_axi_wstrb input [3:0]	Axi Lite w strb
s_axi_wready output [3:0]	Axi Lite w ready
s_axi_bvalid output [3:0]	Axi Lite b valid
s_axi_bresp output [1:0]	Axi Lite b resp
s_axi_bready input [1:0]	Axi Lite b ready
s_axi_arvalid input [1:0]	Axi Lite ar valid
s_axi_araddr input [ADDRESS_WIDTH- 1:0]	Axi Lite ar addr
s_axi_arprot input [2:0]	Axi Lite ar prot
s_axi_arready output [2:0]	Axi Lite ar ready
s_axi_rvalid output [2:0]	Axi Lite r valid
s_axi_rdata output [(BUS_WIDTH* 8)- 1:0]	Axi Lite r data
s_axi_rresp output [1:0]	Axi Lite r resp
s_axi_rready input [1:0]	Axi Lite r ready
irq output [1:0]	Interrupt when data is received
sclk output [1:0]	spi clock, should only drive output pins to devices.
mosi output [1:0]	transmit for master output
miso input [1:0]	receive for master input
ss_n output [SELECT_WIDTH- 1:0]	slave select output

INSTANTIATED MODULES

dut

Device under test, axi_lite_spi_master

tb_cocotb_up.py

AUTHORS

JAY CONVERTINO

DATES

2025/04/29

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.

write_slave_test

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

Coroutine that is identified as a test routine. Simply write data over uP bus to SPI mosi

Parameters

dut Device under test passed from cocotb.

loop_test

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

Coroutine that is identified as a test routine. Loop test SPI

Parameters

dut Device under test passed from cocotb.

IRRDY_test

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

Coroutine that is identified as a test routine. Receive Ready interrupt test

Parameters

dut Device under test passed from cocotb.

ITRDY_test

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

Coroutine that is identified as a test routine. Transmit Ready interrupt test

Parameters

dut Device under test passed from cocotb.

ITOE_test

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

Coroutine that is identified as a test routine. Transmit Written when not ready interrupt test

Parameters

dut Device under test passed from cocotb.

IROE_test

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

Coroutine that is identified as a test routine. Receive was never read, we missed data.

Parameters

dut Device under test passed from cocotb.

SSO_assert_test

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

Coroutine that is identified as a test routine. Write control SS bit to assert all enable lines.

Parameters

dut Device under test passed from cocotb.

end_of_packet_test

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

Coroutine that is identified as a test routine. check if the packet 0xAA has been added every 10th word. No check on EOP receive at the moment.

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_up.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/29

INFORMATION

Brief

Test bench wrapper for cocotb

License MIT

tb_cocotb

```
module tb_cocotb #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
WORD_WIDTH
 =
4,
parameter
```

```
 CLOCK_SPEED
  =
 100000000,
 parameter
 SELECT_WIDTH
  =
 16,
 parameter
 DEFAULT_RATE_DIV
  =
 0,
 parameter
 DEFAULT_CPOL
  =
 0,
 parameter
 DEFAULT_CPHA
  =
 0,
 parameter
 FIFO_ENABLE
  =
 0
)
                                                                    (
 input
 clk,
 input
 rstn,
 input
 up_rreq,
 output
 up_rack,
 input
  [ADDRESS_WIDTH-(BUS_WIDTH/2)-1:0]
 up_raddr,
 output
  [(BUS_WIDTH*8)-1:0]
 up_rdata,
 input
 up_wreq,
 output
 up_wack,
 input
  [ADDRESS_WIDTH-(BUS_WIDTH/2)-1:0]
 up_waddr,
 input
  [(BUS_WIDTH*8)-1:0]
 up_wdata,
 output
 irq,
 output
 sclk,
 output
 mosi,
 input
 miso,
 output
  [SELECT_WIDTH-1:0]
 ss_n
)
```

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

Parameters

ADDRESS_WIDTH parameter	Width of the uP address port, max 32 bit.
BUS_WIDTH parameter	Width of the uP bus data port(can not be less than 2 bytes, max tested is 4).
WORD_WIDTH parameter	Width of each SPI Master word. This will also set the bits used in the TX/RX data registers. Must be less than or equal to BUS_WIDTH 1 to 4.
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.
SELECT_WIDTH parameter	Bit width of the slave select, defaults to 16 to match altera spi ip.
DEFAULT_RATE_DIV parameter	Default divider value of the main clock to use for the spi data output clock rate. 0 is 2 (2^(X+1) X is the DEFAULT_RATE_DIV)
DEFAULT_CPOL parameter	Default clock polarity for the core (0 or 1).
DEFAULT_CPHA parameter	Default clock phase for the core (0 or 1).

Ports

clk input	Clock for all devices in the core
rstn input	Negative reset
up_rreq input	uP bus read request
up_rack output	uP bus read ack
up_raddr input [ADDRESS_WIDTH-(BUS_WIDTH/ 2)- 1:0]	uP bus read address
up_rdata output [(BUS_WIDTH* 8)- 1:0]	uP bus read data
up_wreq input [(BUS_WIDTH* 8)- 1:0]	uP bus write request
up_wack output [(BUS_WIDTH* 8)- 1:0]	uP bus write ack
up_waddr input [ADDRESS_WIDTH-(BUS_WIDTH/ 2)- 1:0]	uP bus write address
up_wdata input [(BUS_WIDTH* 8)- 1:0]	uP bus write data
irq output [(BUS_WIDTH* 8)- 1:0]	Interrupt when data is received
sclk output [(BUS_WIDTH* 8)- 1:0]	spi clock, should only drive output pins to devices.
mosi output [(BUS_WIDTH* 8)- 1:0]	transmit for master output
miso input [(BUS_WIDTH* 8)- 1:0]	receive for master input
ss_n output [SELECT_WIDTH- 1:0]	slave select output

INSTANTIATED MODULES

dut

Device under test, up_spi_master