BUS SPI MASTER

AIR FORCE RESEARCH LABORATORY

May 1, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

BUS UART is a core for interfacing over RS232 UART to a bus of choice. The core will process data to and from the UART. The data can then be accessed over a BUS, currently AXI lite or Wishbone Standard, and processed as needed. All input and output over the bus goes into FIFOs that is then tied to the AXIS UART core. 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

1.3 In a Project

First, pick a core that matches the target bus in question. Then connect the BUS UART core to that bus. Once this is complete the UART pins will need to be routed so they match the UART device or other.

2 Architecture

This core is made up of other cores that are documented in detail in there source. The cores this is made up of are the,

- **axis_uart** Interface with UART and present the data over AXIS interface (see core for documentation).
- **fifo** Used for RX and TX FIFO instances. Set to 16 words buffer max (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_uart** Takes uP bus and coverts it to interface with the RX/TX FIFOs and the AXIS UART (see module documentation for information 5).

For register documentation please see up_uart in 5

3 Building

The BUS UART 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 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.
- sim_cocotb

  Info: Cocotb unit tests

3.3.2 wishbone_standard_spi_master Targets

- default

  Info: Default for IP intergration.

- sim_cocotb

  Info: Cocotb unit tests

3.3.3 up_spi_master Targets

- default

  Info: Default for IP intergration.

- 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_UART 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.

To run the wishbone sim use the command below.

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

To run the axi_lite sim use the command below.

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

To run the uP sim use the command below.

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

5 Module Documentation

up_uart is the module that integrates the AXIS UART core. This includes FIFO's that have there inputs/outputs for data tied 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 UART can use. This makes changing for AXI Lite, to Wishbone to whatever quick and painless.

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

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

The next sections document these modules in great detail. up_uart 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
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]
```

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(can not be less than 2 bytes, max tested is 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	Clock for all devices in the core
arstn	Negative reset
s_axi_awvalid	Axi Lite aw valid
s_axi_awaddr	Axi Lite aw addr
s_axi_awprot	Axi Lite aw prot
s_axi_awready	Axi Lite aw ready
s_axi_wvalid	Axi Lite w valid
s_axi_wdata	Axi Lite w data
s_axi_wstrb	Axi Lite w strb
s_axi_wready	Axi Lite w ready
s_axi_bvalid	Axi Lite b valid
s_axi_bresp	Axi Lite b resp
s_axi_bready	Axi Lite b ready
s_axi_arvalid	Axi Lite ar valid
s_axi_araddr	Axi Lite ar addr
s_axi_arprot	Axi Lite ar prot
s_axi_arready	Axi Lite ar ready

s_axi_rvalid	Axi Lite r valid
s_axi_rdata	Axi Lite r data
s_axi_rresp	Axi Lite r resp
s_axi_rready	Axi Lite r ready
irq	Interrupt when data is received
sclk	spi clock, should only drive output pins to devices.
mosi	transmit for master output
miso	receive for master input
ss_n	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

```
up_axi #(
                                                                         .
AXI_ADDRESS_WIDTH(ADDRESS_WIDTH)
) inst_up_axi ( .up_rstn (arstn), .up_clk (aclk), .up_axi_awvalid(s_axi_awv
```

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

inst_up_spi_master

```
up_spi_master #(
                                                                         .
ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                         .
BUS_WIDTH(BUS_WIDTH),
                                                                         .
CLOCK_SPEED(CLOCK_SPEED),
                                                                         .
SELECT_WIDTH(SELECT_WIDTH),
                                                                         .
DEFAULT_RATE_DIV(DEFAULT_RATE_DIV),
                                                                         .
DEFAULT_CPOL(DEFAULT_CPOL),
                                                                         .
DEFAULT_CPHA(DEFAULT_CPHA)
) inst_up_spi_master ( .clk(aclk), .rstn(arstn), .up_rreq(up_rreq), .up_rack
```

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
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, inp
```

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(can not be less than 2 bytes, max tested is 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	Clock for all devices in the core
rst	Positive reset
s_wb_cyc	Bus Cycle in process
s_wb_stb	Valid data transfer cycle
s_wb_we	Active High write, low read
s_wb_addr	Bus address
s_wb_data_i	Input data
s_wb_sel	Device Select
s_wb_ack	Bus transaction terminated
s_wb_data_o	Output data
s_wb_err	Active high when a bus error is present
irq	Interrupt when data is received
sclk	spi clock, should only drive output pins to devices.
mosi	transmit for master output
miso	receive for master input
ss_n	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

```
up_wishbone_standard #(
                                                                          .
ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                          .
BUS_WIDTH(BUS_WIDTH)
) inst_up_wishbone_standard ( .clk(clk), .rst(rst), .s_wb_cyc(s_wb_cyc), .s_
```

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

inst_up_spi_master

```
up_spi_master #(
                                                                          .
ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                          .
BUS_WIDTH(BUS_WIDTH),
                                                                          .
CLOCK_SPEED(CLOCK_SPEED),
                                                                          .
SELECT_WIDTH(SELECT_WIDTH),
                                                                          .
DEFAULT_RATE_DIV(DEFAULT_RATE_DIV),
                                                                          .
DEFAULT_CPOL(DEFAULT_CPOL),
                                                                          .
DEFAULT_CPHA(DEFAULT_CPHA)
) inst_up_spi_master ( .clk(clk), .rstn(~rst), .up_rreq(up_rreq), .up_rack(
```

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

up_spi.v

AUTHORS

JAY CONVERTINO

DATES

2024/04/29

INFORMATION

Brief

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

License MIT

up_spi_master

```
module up_spi_master #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_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 rstn, input up_rreq, output up_rack, input [ADDRESS_WII
```

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).
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	Clock for all devices in the core
rstn	Negative reset
up_rreq	uP bus read request
up_rack	uP bus read ack
up_raddr	uP bus read address
up_rdata	uP bus read data
up_wreq	uP bus write request
up_wack	uP bus write ack
up_waddr	uP bus write address
up_wdata	uP bus write data
irq	Interrupt when data is received
sclk	spi clock, should only drive output pins to devices.
mosi	transmit for master output
miso	receive for master input
ss_n	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

REGISTER INFORMATION

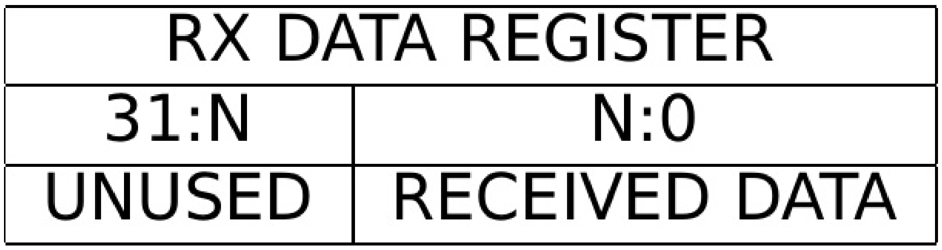
Core has 7 registers at the offsets that follow when at a full 32 bit bus width, Internal address is OFFSET >> BUS_WIDTH/2 (32bit would be h4 >> 2 = 1 for internal address).

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

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 BUS_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 BUS_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, Enable (1) or disable(0) all interrupts that are active.
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

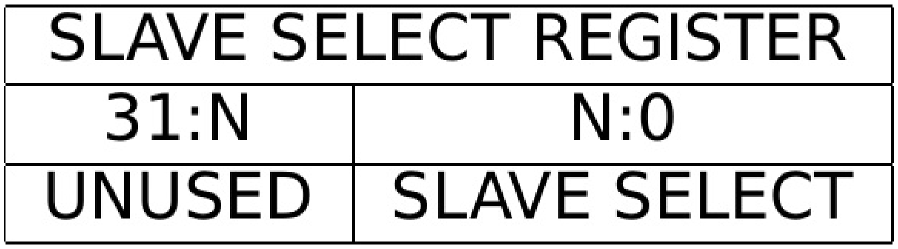
```
localparam RESERVED = 8'h10 >> DIVISOR
```

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



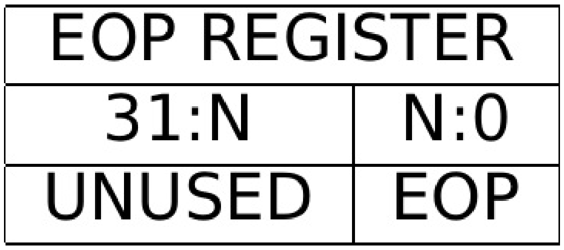
SLAVE SELECT REGISTER	
31:N	N:0
UNUSED	SLAVE SELECT

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



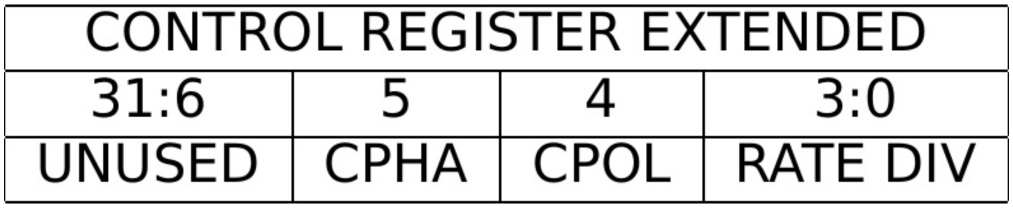
EOP REGISTER	
31:N	N:0
UNUSED	EOP

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			
31:6	5	4	3:0
UNUSED	CPHA	CPOL	RATE DIV

Control Extension to add capabilities to Altera IP core.

CPHA	5, Clock Phase Bit, 0 or 1 per SPI specs (default value set by IP parameter).
CPOL	4, Clock Polarity bit, 0 or 1 per SPI specs (default value set by IP parameter).
RATE_TOP	3, Top bit for rate control. Divider values are 0 to 15 (2^X+1 where X is the divider value).
RATE_BOT	0, Bottom bit for rate control.

INSTANTIATED MODULES

inst_axis_spi

```
axis_spi_master #(
                                                                            .
CLOCK_SPEED(CLOCK_SPEED),
                                                                            .
BUS_WIDTH(BUS_WIDTH),
                                                                            .
SELECT_WIDTH(SELECT_WIDTH)
) inst_axis_spi_master ( .aclk(clk), .arstn(rstn), .s_axis_tdata(r_tx_wdat
```

SPI Master instance with AXIS interface

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
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, inp
```

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(can not be less than 2 bytes, max tested is 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	Clock for all devices in the core
rst	Positive reset
s_wb_cyc	Bus Cycle in process
s_wb_stb	Valid data transfer cycle
s_wb_we	Active High write, low read
s_wb_addr	Bus address
s_wb_data_i	Input data
s_wb_sel	Device Select
s_wb_ack	Bus transaction terminated
s_wb_data_o	Output data
s_wb_err	Active high when a bus error is present
irq	Interrupt when data is received
sclk	spi clock, should only drive output pins to devices.
mosi	transmit for master output
miso	receive for master input
ss_n	slave select output

INSTANTIATED MODULES

dut

```
wishbone_standard_spi_master #(
                                                                          .
ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                          .
BUS_WIDTH(BUS_WIDTH),
                                                                          .
CLOCK_SPEED(CLOCK_SPEED),
                                                                          .
SELECT_WIDTH(SELECT_WIDTH),
                                                                          .
DEFAULT_RATE_DIV(DEFAULT_RATE_DIV),
                                                                          .
DEFAULT_CPOL(DEFAULT_CPOL),
                                                                          .
DEFAULT_CPHA(DEFAULT_CPHA)
) dut ( .clk(clk), .rst(rst), .s_wb_cyc(s_wb_cyc), .s_wb_stb(s_wb_stb), .s_w
```

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
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]
```

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(can not be less than 2 bytes, max tested is 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	Clock for all devices in the core
arstn	Negative reset
s_axi_awvalid	Axi Lite aw valid
s_axi_awaddr	Axi Lite aw addr
s_axi_awprot	Axi Lite aw prot
s_axi_awready	Axi Lite aw ready
s_axi_wvalid	Axi Lite w valid
s_axi_wdata	Axi Lite w data
s_axi_wstrb	Axi Lite w strb
s_axi_wready	Axi Lite w ready
s_axi_bvalid	Axi Lite b valid
s_axi_bresp	Axi Lite b resp
s_axi_bready	Axi Lite b ready
s_axi_arvalid	Axi Lite ar valid
s_axi_araddr	Axi Lite ar addr
s_axi_arprot	Axi Lite ar prot
s_axi_arready	Axi Lite ar ready

s_axi_rvalid	Axi Lite r valid
s_axi_rdata	Axi Lite r data
s_axi_rresp	Axi Lite r resp
s_axi_rready	Axi Lite r ready
irq	Interrupt when data is received
sclk	spi clock, should only drive output pins to devices.
mosi	transmit for master output
miso	receive for master input
ss_n	slave select output

INSTANTIATED MODULES

dut

```
 axi_lite_spi_master #(
                                                                          .
 ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                          .
 BUS_WIDTH(BUS_WIDTH),
                                                                          .
 CLOCK_SPEED(CLOCK_SPEED),
                                                                          .
 SELECT_WIDTH(SELECT_WIDTH),
                                                                          .
 DEFAULT_RATE_DIV(DEFAULT_RATE_DIV),
                                                                          .
 DEFAULT_CPOL(DEFAULT_CPOL),
                                                                          .
 DEFAULT_CPHA(DEFAULT_CPHA)
) dut ( .aclk(aclk), .arstn(arstn), .s_axi_awvalid(s_axi_awvalid), .s_axi_aw
```

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
CLOCK_SPEED
 =
100000000,
parameter
```

```
SELECT_WIDTH
 =
16,
parameter
DEFAULT_RATE_DIV
 =
0,
parameter
DEFAULT_CPOL
 =
0,
parameter
DEFAULT_CPHA
 =
0
) ( input clk, input rstn, input up_rreq, output up_rack, input [ADDRESS_WID
```

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).
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	Clock for all devices in the core
rstn	Negative reset
up_rreq	uP bus read request
up_rack	uP bus read ack
up_raddr	uP bus read address
up_rdata	uP bus read data
up_wreq	uP bus write request
up_wack	uP bus write ack
up_waddr	uP bus write address
up_wdata	uP bus write data
irq	Interrupt when data is received
sclk	spi clock, should only drive output pins to devices.
mosi	transmit for master output
miso	receive for master input
ss_n	slave select output

INSTANTIATED MODULES

dut

```
up_spi_master #(
                                                                          .
ADDRESS_WIDTH(ADDRESS_WIDTH),
                                                                          .
BUS_WIDTH(BUS_WIDTH),
                                                                          .
CLOCK_SPEED(CLOCK_SPEED),
                                                                          .
SELECT_WIDTH(SELECT_WIDTH),
                                                                          .
DEFAULT_RATE_DIV(DEFAULT_RATE_DIV),
                                                                          .
DEFAULT_CPOL(DEFAULT_CPOL),
                                                                          .
DEFAULT_CPHA(DEFAULT_CPHA)
) dut ( .clk(clk), .rstn(rstn), .up_rreq(up_rreq), .up_rack(up_rack), .up_ra
```

Device under test, up_spi_master