BUS_1553

June 26, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

BUS1553 is a core for interfacing the PMOD1553 device to a bus of choice. The core will process data to and from the PMOD1553. 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 demodulation and modulation cores, which then send/recv the differential data to/from the PMOD1553 device. 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_1553 Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device:up_1553:1.0.0
  - AD:common:up_axi:1.0.0

1.2.2 wishbone_standard_1553 Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device:up_1553:1.0.0
  - AFRL:bus:up_wishbone_standard:1.0.0

1.2.3 up_1553 Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device_converter:axis_1553:1.0.0
  - AFRL:buffer:fifo

1.3 In a Project

First, pick a core that matches the target bus in question. Then connect the BUS1553 core to that bus. Once this is complete the PMOD pins will need to be routed so they match the PMOD1553 device. Please see the schematic of the PMOD1553 for electrical connection details. All I/O's are 3.3volt.

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_1553** Encodes and decodeds data from the TX/RX FIFO and sends/recvs it to the PMOD1553 (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_1553** Takes uP bus and coverts it to interface with the RX/TX FIFOs and the encoder/decoder (see module documentation for information 5).

For register documentation please see up_1553 in 5

3 Building

The BUS1553 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. 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_1553 File List

- src
  - src/axi_lite_1553.v
- tb_cocotb
  - 'tb/tb_cocotb_axi_lite.py': 'file_type': 'user', 'copyto': '.'
  - 'tb/tb_cocotb_axi_lite.v': 'file_type': 'verilogSource'

3.2.2 wishbone_standard_1553 File List

- src
  - src/wishbone_standard_1553.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_1553 File List

- src
  - src/up_1553.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_1553 Targets

- default

  Info: Default for IP intergration.

- lint

  Info: Lint with Verible

- sim_cocotb

  Info: Cocotb unit tests

3.3.2 wishbone_standard_1553 Targets

- default

  Info: Default for IP intergration.

- lint

  Info: Lint with Verible

- sim_cocotb

  Info: Cocotb unit tests

3.3.3 up_1553 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_1553 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_1553:1.0.0
```

To run the axi_lite sim use the command below.

```
fusesoc run --target sim_cocotb AFRL:device:axi_lite_1553:1.0.0
```

To run the uP sim use the command below.

```
fusesoc run --target sim_cocotb AFRL:device:up_1553:1.0.0
```

5 Module Documentation

up_1553 is the module that integrates the AXI streaming 1553 encoder/decoder. 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 BUS1553 can use. This makes changing for AXI Lite, to Wishbone to whatever quick and painless.

axi_lite_1553 module adds a AXI Lite to uP (microprocessor) bus converter. The converter is from Analog Devices.

wishbone_standard_1553 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_1553 contains the register map explained, and what the various bits do.

axi_lite_1553.v

AUTHORS

JAY CONVERTINO

DATES

2024/10/17

INFORMATION

Brief

AXI Lite 1553 is a core for interfacing with 1553 devices over the AXI lite bus.

License MIT

axi_lite_1553

```
module axi_lite_1553 #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
CLOCK_SPEED
 =
100000000
)
```

```
                                                                (
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,
input
 [1:0]
rx_diff,
output
 [1:0]
tx_diff,
output
tx_active,
output
irq
)
```

AXI Lite based 1553 communications device.

Parameters

ADDRESS_WIDTH
parameter
Width of the axi address bus, max 32 bit.

BUS_WIDTH
parameter
Width in bytes of the data bus.

CLOCK_SPEED
parameter
This is the aclk frequency in Hz

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

rx_diff
input [1:0]
Input differential signal for 1553 bus

tx_diff output [1:0]	Output differential signal for 1553 bus
tx_active output [1:0]	Enable output of differential signal (for signal switching on 1553 module)
irq output [1:0]	Interrupt when data is received

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_1553

Module instance of up_1553 creating a Logic wrapper for 1553 bus cores to interface with uP bus.

wishbone_classic_1553.v

AUTHORS

JAY CONVERTINO

DATES

2024/10/17

INFORMATION

Brief

wishbone classic to uP core for 1553 comms.

License MIT

wishbone_standard_1553

```
module wishbone_standard_1553 #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
CLOCK_SPEED
 =
100000000
)
```

```
                                                                    (
 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,
 input
  [1:0]
 rx_diff,
 output
  [1:0]
 tx_diff,
 output
 tx_active,
 output
 irq
)
```

Wishbone Stanard based 1553 communications device.

Parameters

ADDRESS_WIDTH — Width of the address bus in bits, max 32 bit.
parameter

BUS_WIDTH — Width of the data bus in bytes.
parameter

CLOCK_SPEED — This is the aclk frequency in Hz
parameter

Ports

clk — Clock for all devices in the core
input

rst — Positive reset
input

s_wb_cyc — Bus Cycle in process
input

s_wb_stb — Valid data transfer cycle
input

s_wb_we — Active High write, low read
input

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
rx_diff `input [1:0]`	Input differential signal for 1553 bus
tx_diff `output [1:0]`	Output differential signal for 1553 bus
tx_active `output [1:0]`	Enable output of differential signal (for signal switching on 1553 module)
irq `output [1:0]`	Interrupt when data is received

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_1553

Module instance of up_1553 creating a Logic wrapper for 1553 bus cores to interface with uP bus.

up_1553.v

AUTHORS

JAY CONVERTINO

DATES

2024/10/17

INFORMATION

Brief

uP Core for interfacing with simple 1553 communications.

License MIT

up_1553

```
module up_1553 #(
parameter
ADDRESS_WIDTH
 =
32,
parameter
BUS_WIDTH
 =
4,
parameter
CLOCK_SPEED
 =
100000000
)
```

```
                                                                    (
 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,
 input
  [1:0]
 rx_diff,
 output
  [1:0]
 tx_diff,
 output
 tx_active,
 output
 irq
)
```

uP based 1553 communications device.

Parameters

ADDRESS_WIDTH parameter	Width of the uP address port, max 32 bit.
BUS_WIDTH parameter	Width of the uP bus data port.
CLOCK_SPEED parameter	This is the aclk frequency in Hz

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
rx_diff `input [1:0]`	Input differential signal for 1553 bus
tx_diff `output [1:0]`	Output differential signal for 1553 bus
tx_active `output [1:0]`	Enable output of differential signal as this indicates tx is active (for signal switching on 1553 module)
irq `output [1:0]`	Interrupt when data is received

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).

FIFO_DEPTH

```
localparam FIFO_DEPTH = 16
```

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

DATA_BITS

```
localparam DATA_BITS = 21
```

Number of bits in RX/TX FIFO that are valid.

REGISTER INFORMATION

Core has 4 registers at the offsets that follow.

RX_FIFO_REG	h0
TX_FIFO_REG	h4
STATUS_REG	h8
CONTROL_REG	hC

RX_FIFO_REG

```
localparam RX_FIFO_REG = 4'h0 >> DIVISOR
```

Defines the address offset for RX FIFO

RX FIFO REGISTER		
31:20	20:16	15:0
UNUSED	STATUS DATA	RECEIVED DATA

Valid bits are from 20:0. Bits 20:16 are status bits information about the data. Bit 15:0 are data.

Status Bits

{S:1,D:1,TY:3}

TY Type is 3 bits, 000 NA, 001 = REG, 010 = DATA, 100 = CMD/STATUS

D Delay Enabled is 1 bit, 1 is there was be a delay of 4 us or more, or 0 no delay.

S Sync only when 1, normal is 0

TX_FIFO_REG

```
localparam TX_FIFO_REG = 4'h4 >> DIVISOR
```

Defines the address offset to write the TX FIFO.

TX FIFO REGISTER		
31:20	20:16	15:0
UNUSED	STATUS DATA	TRANSMIT DATA

Valid bits are from 20:0. Bits 20:16 are status bits information about the data. Bit 15:0 are data.

Status Bits

{S:1,D:1,TY:3}

TY Type is 3 bits, 000 NA, 001 = REG, 010 = DATA, 100 = CMD/STATUS

D Delay Enabled is 1 bit, 1 is there must be a delay of 4 us or more, or 0 no delay.

S Sync only when 1, normal is 0

STATUS_REG

```
localparam STATUS_REG = 4'h8 >> DIVISOR
```

Defines the address offset to read the status bits.

STATUS REGISTER							
31:7	6	5	4	3	2	1	0
UNUSED	Parity_err	frame_err	irq_en	tx_full	tx_empty	rx_full	rx_valid

Status Register Bits

parity_err 6, When 1 an error in the RX parity check has occured

frame_err 5, When 1 an error in the RX frame has occured (manchester data 2'b11 or 2'b00).
irq_en 4, When 1 the IRQ is enabled by CONTROL_REG
tx_full 3, When 1 the tx fifo is full.
tx_empty 2, When 1 the tx fifo is empty.
rx_full 1, When 1 the rx fifo is full.
rx_valid 0, When 1 the rx fifo contains valid data.

CONTROL_REG

```
localparam CONTROL_REG = 4'hC
```

Defines the address offset to set the control bits.

CONTROL REGISTER				
31:5	4	3:2	1	0
UNUSED	ENA_INTR_BIT	UNUSED	RST_RX_BIT	RST_TX_BIT

See Also: ENABLE_INTR_BIT, RESET_RX_BIT, RESET_TX_BIT

Control Register Bits

ENABLE_INTR_BIT 4, Control Register offset bit for enabling the interrupt.
RESET_RX_BIT 1, Control Register offset bit for resetting the RX FIFO.
RESET_TX_BIT 0, Control Register offset bit for resetting the TX FIFO.

inst_axis_1553

Decode/Encode differential 1553 data stream to AXIS data format.

inst_rx_fifo

Buffer up to 16 items output from the axis_1553_encoder.

inst_tx_fifo

Buffer up to 16 items to input to the axis_1553_decoder.

tb_cocotb_wishbone_standard.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.

increment_test_cmd_send

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

Coroutine that is identified as a test routine. Setup up to send 1553 commands

Parameters

dut Device under test passed from cocotb.

increment_test_cmd_recv

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

Coroutine that is identified as a test routine. Setup up to recv 1553 commands

Parameters

dut Device under test passed from cocotb.

increment_test_data_send

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

Coroutine that is identified as a test routine. Setup up to send 1553 data

Parameters

dut Device under test passed from cocotb.

increment_test_data_recv

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

Coroutine that is identified as a test routine. Setup up to recv 1553 data

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

2025/04/01

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
)
```

```
                                                                (
 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,
 input
  [1:0]
 rx_diff,
 output
  [1:0]
 tx_diff,
 output
 tx_active,
 output
 irq
)
```

Wishbone Stanard based 1553 communications device.

Parameters

ADDRESS_WIDTH
parameter
Width of the address bus in bits, max 32 bit.

BUS_WIDTH
parameter
Width of the data bus in bytes.

CLOCK_SPEED
parameter
This is the aclk frequency in Hz

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
rx_diff input [1:0]	Input differential signal for 1553 bus
tx_diff output [1:0]	Output differential signal for 1553 bus
tx_active output [1:0]	Enable output of differential signal (for signal switching on 1553 module)
irq output [1:0]	Interrupt when data is received

INSTANTIATED MODULES

dut

Device under test, wishbone_standard_1553

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.

increment_test_cmd_send

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

Coroutine that is identified as a test routine. Setup up to send 1553 commands

Parameters

dut Device under test passed from cocotb.

increment_test_cmd_recv

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

Coroutine that is identified as a test routine. Setup up to recv 1553 commands

Parameters

dut Device under test passed from cocotb.

increment_test_data_send

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

Coroutine that is identified as a test routine. Setup up to send 1553 data

Parameters

dut Device under test passed from cocotb.

increment_test_data_recv

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

Coroutine that is identified as a test routine. Setup up to recv 1553 data

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

2025/04/01

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
)
```

```
                                                                    (
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,
input
 [1:0]
rx_diff,
output
 [1:0]
tx_diff,
output
tx_active,
output
irq
)
```

AXI Lite slave to AXI Lite 1553 DUT

Parameters

ADDRESS_WIDTH
parameter
Width of the axi address bus, max 32 bit.

BUS_WIDTH
parameter
Width in bytes of the data bus.

CLOCK_SPEED
parameter
This is the aclk frequency in Hz

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

rx_diff
input [1:0]
Input differential signal for 1553 bus

tx_diff `output [1:0]`	Output differential signal for 1553 bus
tx_active `output [1:0]`	Enable output of differential signal (for signal switching on 1553 module)
irq `output [1:0]`	Interrupt when data is received

INSTANTIATED MODULES

dut

Device under test, axi_lite_1553

tb_cocotb_up.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.

increment_test_cmd_send

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

Coroutine that is identified as a test routine. Setup up to send 1553 commands ADDRESS MAP FOR uP: 0=0,4=1,8=2,C=3

Parameters

dut Device under test passed from cocotb.

increment_test_cmd_recv

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

Coroutine that is identified as a test routine. Setup up to recv 1553 commands ADDRESS MAP FOR uP: 0=0,4=1,8=2,C=3

Parameters

dut Device under test passed from cocotb.

increment_test_data_send

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

Coroutine that is identified as a test routine. Setup up to send 1553 data ADDRESS MAP FOR uP:

0=0,4=1,8=2,C=3

Parameters

dut Device under test passed from cocotb.

increment_test_data_recv

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

Coroutine that is identified as a test routine. Setup up to recv 1553 data ADDRESS MAP FOR uP: 0=0,4=1,8=2,C=3

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

2025/04/01

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
)
```

```
                                                                    (
 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,
 input
  [1:0]
 rx_diff,
 output
  [1:0]
 tx_diff,
 output
 tx_active,
 output
 irq
)
```

uP 1553 testbench

Parameters

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

BUS_WIDTH — Width of the uP bus data port.
parameter

CLOCK_SPEED — This is the aclk frequency in Hz
parameter

Ports

clk — Clock for all devices in the core
input

rstn — Negative reset
input

up_rreq — uP bus read request
input

up_rack — uP bus read ack
output

up_raddr — uP bus read address
input [ADDRESS_WIDTH-(BUS_WIDTH/ 2)- 1:0]

up_rdata — uP bus read data
output [(BUS_WIDTH* 8)- 1:0]

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
rx_diff `input [1:0]`	Input differential signal for 1553 bus
tx_diff `output [1:0]`	Output differential signal for 1553 bus
tx_active `output [1:0]`	Enable output of differential signal (for signal switching on 1553 module)
irq `output [1:0]`	Interrupt when data is received

INSTANTIATED MODULES

dut

Device under test, up_1553