

BUS\_1553



November 13, 2024

Jay Convertino

# Contents

<b>1 Usage</b>	<b>2</b>
1.1 Introduction . . . . .	2
1.2 Dependencies . . . . .	2
1.2.1 axi_lite_1553 Depenecies . . . . .	2
1.2.2 wishbone_classic_1553 Depenecies . . . . .	2
1.2.3 up_1553 Depenecies . . . . .	3
1.3 In a Project . . . . .	3
<b>2 Architecture</b>	<b>3</b>
<b>3 Building</b>	<b>4</b>
3.1 fusesoc . . . . .	4
3.2 Source Files . . . . .	4
3.2.1 axi_lite_1553 File List . . . . .	4
3.2.2 wishbone_classic_1553 File List . . . . .	4
3.2.3 up_1553 File List . . . . .	5
3.3 Targets . . . . .	5
3.3.1 axi_lite_1553 Targets . . . . .	5
3.3.2 wishbone_classic_1553 Targets . . . . .	7
3.3.3 up_1553 Targets . . . . .	7
3.4 Directory Guide . . . . .	7
<b>4 Simulation</b>	<b>8</b>
4.1 iverilog . . . . .	8
4.2 cocotb . . . . .	8
<b>5 Module Documentation</b>	<b>9</b>
5.1 axi_lite_1553 . . . . .	10
5.2 wishbone_classic_1553 . . . . .	14
5.3 up_1553 . . . . .	18
5.3.1 Registers . . . . .	20

# 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 Classic, 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 Dependencies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device:up\_1553:1.0.0
  - AD:common:up\_axi:1.0.0
- dep\_tb
  - AFRL:simulation:axis\_stimulator
  - AFRL:utility:sim\_helper

### 1.2.2 wishbone\_classic\_1553 Dependencies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device:up\_1553:1.0.0
  - AFRL:bus:up\_wishbone\_classic:1.0.0

### 1.2.3 up\_1553 Dependencies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:device\_converter:axis\_1553\_encoder:1.0.0
  - AFRL:device\_converter:axis\_1553\_decoder: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\_encoder** Encodes data from the RX FIFO and sends it to the PMOD1553 (see core for documentation).
- **axis\_1553\_decoder** Decodes data from the PMOD1553 and sends it to the TX FIFO (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\_classic** A wishbone classic to uP converter core (see core for documentation).
- **up\_1553** Takes uP bus and converts 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 met 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\_1553 File List

- src
  - Type: verilogSource
  - src/axi\_lite\_1553.v
- tb
  - Type: verilogSource
  - tb/tb\_1553.v

#### 3.2.2 wishbone\_classic\_1553 File List

- src
  - Type: verilogSource
  - src/wishbone\_classic\_1553.v
- tb
  - Type: verilogSource
  - tb/tb\_wishbone\_slave.v

### 3.2.3 up\_1553 File List

- src  
Type: verilogSource
  - src/up\_1553.v
- tb  
Type: verilogSource
  - tb/tb\_up\_1553.v

## 3.3 Targets

### 3.3.1 axi\_lite\_1553 Targets

- default  
Info: Default for IP intergration.
  - src
  - dep
- sim  
Info: Base simulation using icarus as default.
  - src
  - dep
  - tb
  - dep\_tb
  - IN\_FILE\_NAME
  - OUT\_FILE\_NAME
  - RAND\_READY
- sim\_rand\_data  
Info: Use random data as sim input.
  - src
  - dep
  - tb
  - dep\_tb
  - IN\_FILE\_NAME=random.bin
  - OUT\_FILE\_NAME=out\_random.bin
  - RAND\_READY

- FIFO\_DEPTH
- sim\_rand\_ready\_rand\_data
  - Info: Use random data with a random ready as sim input.
  - src
  - dep
  - tb
  - dep\_tb
  - IN\_FILE\_NAME=random.bin
  - OUT\_FILE\_NAME=out\_random.bin
  - RAND\_READY=1
  - FIFO\_DEPTH
- sim\_8bit\_count\_data
  - Info: Use counter data as sim input.
  - src
  - dep
  - tb
  - dep\_tb
  - IN\_FILE\_NAME=8bit\_count.bin
  - OUT\_FILE\_NAME=out\_8bit\_count.bin
  - RAND\_READY
  - FIFO\_DEPTH
- sim\_rand\_ready\_8bit\_count\_data
  - Info: Use counter data with a random ready as sim input.
  - src
  - dep
  - tb
  - dep\_tb
  - IN\_FILE\_NAME=8bit\_count.bin
  - OUT\_FILE\_NAME=out\_8bit\_count.bin
  - RAND\_READY=1
  - FIFO\_DEPTH

### 3.3.2 wishbone\_classic\_1553 Targets

- default
  - Info: Default for IP intergration.
  - src
  - dep
- sim
  - Info: Base simulation using icarus as default.
  - src
  - dep
  - tb

### 3.3.3 up\_1553 Targets

- default
  - Info: Default for IP intergration.
  - src
  - dep
- sim
  - Info: Base simulation using icarus as default.
  - src
  - dep
  - tb

## 3.4 Directory Guide

Below highlights important folders from the root of BUS1553.

1. **docs** Contains all documentation related to this project.
  - **manual** Contains user manual and github page that are generated from the latex sources.
2. **src** Contains source files for the core
3. **tb** Contains test bench files for iverilog and cocotb
  - **cocotb** testbench files



## **4 Simulation**

There are a few different simulations that can be run for this core.

### **4.1 iverilog**

iverilog is used for simple test benches for quick verification, visually, of the core.

### **4.2 cocotb**

Future simulations will use cocotb. This feature is not yet implemented.

## 5 Module Documentation

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\_classic\_1553 module adds a Wishbone Classic to uP (microprocessor) bus converter. This converter was designed for Wishbone Classic 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

---

Copyright 2024 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## axi\_lite\_1553

---

```
module axi_lite_1553 #(
  parameter
  ADDRESS_WIDTH
  =
  32,
  parameter
  CLOCK_SPEED
  =
  100000000,
  parameter
  SAMPLE_RATE
```

```

=
2000000,
parameter
BIT_SLICE_OFFSET
=
0,
parameter
INVERT_DATA
=
0,
parameter
SAMPLE_SELECT
=
0
) ( input aclk, input arstn, input s_axi_awvalid, input [ADDRESS_WIDTH-1:0]

```

AXI Lite based 1553 communications device.

## Parameters

<b>ADDRESS_WIDTH</b> parameter	Width of the axi address bus
<b>CLOCK_SPEED</b> parameter	This is the aclk frequency in Hz
<b>SAMPLE_RATE</b> parameter	Rate of in which to sample the 1553 bus. Must be 2 MHz or more and less than aclk. This is in Hz. BIT_SLICE_OFFSET- Adjust where the sample is taken from the input.
<b>INVERT_DATA</b> parameter	Invert all 1553 bits coming in and out.
<b>SAMPLE_SELECT</b> parameter	Adjust where in the array of samples to select a bit.

## Ports

<b>aclk</b>	Clock for all devices in the core
<b>arstn</b>	Negative reset
<b>s_axi_awvalid</b>	Axi Lite aw valid
<b>s_axi_awaddr</b>	Axi Lite aw addr
<b>s_axi_awprot</b>	Axi Lite aw prot
<b>s_axi_awready</b>	Axi Lite aw ready
<b>s_axi_wvalid</b>	Axi Lite w valid
<b>s_axi_wdata</b>	Axi Lite w data
<b>s_axi_wstrb</b>	Axi Lite w strb
<b>s_axi_wready</b>	Axi Lite w ready
<b>s_axi_bvalid</b>	Axi Lite b valid
<b>s_axi_bresp</b>	Axi Lite b resp
<b>s_axi_bready</b>	Axi Lite b ready
<b>s_axi_arvalid</b>	Axi Lite ar valid
<b>s_axi_araddr</b>	Axi Lite ar addr
<b>s_axi_arprot</b>	Axi Lite ar prot
<b>s_axi_arready</b>	Axi Lite ar ready
<b>s_axi_rvalid</b>	Axi Lite r valid

<b>s_axi_rdata</b>	Axi Lite r data
<b>s_axi_rresp</b>	Axi Lite r resp
<b>s_axi_rready</b>	Axi Lite r ready
<b>i_diff</b>	Input differential signal for 1553 bus
<b>o_diff</b>	Output differential signal for 1553 bus
<b>en_o_diff</b>	Enable output of differential signal (for signal switching on 1553 module)
<b>irq</b>	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-3:0] 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-3:0] up_waddr
```

uP write bus address

## up\_wdata

---

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

uP write bus data

## INSTANTIATED 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\_1553

---

```
up_1553 #(
    .ADDRESS_WIDTH(ADDRESS_WIDTH),
    .CLOCK_SPEED(CLOCK_SPEED),
    .SAMPLE_RATE(SAMPLE_RATE),
    .BIT_SLICE_OFFSET(BIT_SLICE_OFFSET),
    .INVERT_DATA(INVERT_DATA),
    .SAMPLE_SELECT(SAMPLE_SELECT)
) inst_up_1553 ( .clk(aclk), .rstn(arstn), .up_rreq(up_rreq), .up_rack(up_ra
```

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

---

Copyright 2024 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## wishbone\_classic\_1553

---

```
module wishbone_classic_1553 #(
    parameter
    ADDRESS_WIDTH
    =
    32,
    parameter
    BUS_WIDTH
    =
    4,
    parameter
    CLOCK_SPEED
```

```

=
100000000,
parameter
SAMPLE_RATE
=
2000000,
parameter
BIT_SLICE_OFFSET
=
0,
parameter
INVERT_DATA
=
0,
parameter
SAMPLE_SELECT
=
0
) ( input clk, input rst, input s_wb_cyc, input s_wb_stb, input s_wb_we, in

```

Wishbone Calssic based 1553 communications device.

## Parameters

<b>ADDRESS_WIDTH</b> parameter	Width of the address bus in bits
<b>BUS_WIDTH</b> parameter	Width of the data bus in bytes.
<b>CLOCK_SPEED</b> parameter	This is the aclk frequency in Hz
<b>SAMPLE_RATE</b> parameter	Rate of in which to sample the 1553 bus. Must be 2 MHz or more and less than aclk. This is in Hz. BIT_SLICE_OFFSET- Adjust where the sample is taken from the input.
<b>INVERT_DATA</b> parameter	Invert all 1553 bits coming in and out.
<b>SAMPLE_SELECT</b> parameter	Adjust where in the array of samples to select a bit.

## Ports

<b>clk</b>	Clock for all devices in the core
<b>rst</b>	Positive reset
<b>s_wb_cyc</b>	Bus Cycle in process
<b>s_wb_stb</b>	Valid data transfer cycle
<b>s_wb_we</b>	Active High write, low read
<b>s_wb_addr</b>	Bus address
<b>s_wb_data_i</b>	Input data
<b>s_wb_sel</b>	Device Select
<b>s_wb_bte</b>	Burst Type Extension
<b>s_wb_cti</b>	Cycle Type
<b>s_wb_ack</b>	Bus transaction terminated
<b>s_wb_data_o</b>	Output data
<b>s_wb_err</b>	Active high when a bus error is present



<b>i_diff</b>	Input differential signal for 1553 bus
<b>o_diff</b>	Output differential signal for 1553 bus
<b>en_o_diff</b>	Enable output of differential signal (for signal switching on 1553 module)
<b>irq</b>	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-3:0] 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-3:0] up_waddr
```

uP write bus address

## up\_wdata

---

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

uP write bus data

## INSTANTIATED MODULES

---

### inst\_up\_wishbone\_classic

---

```
up_wishbone_classic #(
    ADDRESS_WIDTH(ADDRESS_WIDTH),
    BUS_WIDTH(BUS_WIDTH)
) inst_up_wishbone_classic ( .clk(clk), .rst(rst), .s_wb_cyc(s_wb_cyc), .s_v
```

Module instance of up\_wishbone\_classic for the Wishbone Classic bus to the uP bus.

### inst\_up\_1553

---

```
up_1553 #(
    ADDRESS_WIDTH(ADDRESS_WIDTH),
    BUS_WIDTH(BUS_WIDTH),
    CLOCK_SPEED(CLOCK_SPEED),
    SAMPLE_RATE(SAMPLE_RATE),
    BIT_SLICE_OFFSET(BIT_SLICE_OFFSET),
    INVERT_DATA(INVERT_DATA),
    SAMPLE_SELECT(SAMPLE_SELECT)
) inst_up_1553 ( .clk(aclk), .rstn(arstn), .up_rreq(up_rreq), .up_rack(up_rack)
```

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

---

Copyright 2024 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## up\_1553

---

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

```

=
100000000,
parameter
SAMPLE_RATE
=
2000000,
parameter
BIT_SLICE_OFFSET
=
0,
parameter
INVERT_DATA
=
0,
parameter
SAMPLE_SELECT
=
0
) ( input clk, input rstn, input up_rreq, output up_rack, input [ADDRESS_WID

```

uP based 1553 communications device.

## Parameters

<b>ADDRESS_WIDTH</b> parameter	Width of the uP address port.
<b>BUS_WIDTH</b> parameter	Width of the uP bus data port.
<b>CLOCK_SPEED</b> parameter	This is the aclk frequency in Hz
<b>SAMPLE_RATE</b> parameter	Rate of in which to sample the 1553 bus. Must be 2 MHz or more and less than aclk. This is in Hz. BIT_SLICE_OFFSET- Adjust where the sample is taken from the input.
<b>INVERT_DATA</b> parameter	Invert all 1553 bits coming in and out.
<b>SAMPLE_SELECT</b> parameter	Adjust where in the array of samples to select a bit.

## Ports

<b>clk</b>	Clock for all devices in the core
<b>rstn</b>	Negative reset
<b>up_rreq</b>	uP bus read request
<b>up_rack</b>	uP bus read ack
<b>up_raddr</b>	uP bus read address
<b>up_rdata</b>	uP bus read data
<b>up_wreq</b>	uP bus write request
<b>up_wack</b>	uP bus write ack
<b>up_waddr</b>	uP bus write address
<b>up_wdata</b>	uP bus write data
<b>i_diff</b>	Input differential signal for 1553 bus
<b>o_diff</b>	Output differential signal for 1553 bus
<b>en_o_diff</b>	Enable output of differential signal (for signal switching on 1553 module)
<b>irq</b>	Interrupt when data is received

## FIFO\_DEPTH

```
localparam FIFO_DEPTH = 16
```

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

## 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
```

Defines the address offset for RX FIFO

RX FIFO REGISTER		
31:24	23:16	15:0
UNUSED	STATUS DATA	RECEIVED DATA

Valid bits are from 23:0. Bits 23:16 are information about the data. Bit 15:0 are data.

## TX\_FIFO\_REG

```
localparam TX_FIFO_REG = 4'h4
```

Defines the address offset to write the TX FIFO.

TX FIFO REGISTER		
31:24	23:16	15:0
UNUSED	STATUS DATA	TRANSMIT DATA

Valid bits are from 23:0. Bits 23:16 are information about the data. Bit 15:0 are data.

## STATUS\_REG

```
localparam STATUS_REG = 4'h8
```

Defines the address offset to read the status bits.

STATUS REGISTER								
31:8	7	6	5	4	3	2	1	0
UNUSED	PC	DI	Delay	irq_en	tx_full	tx_empty	rx_full	rx_valid

## Status Register Bits

<b>PC</b>	7, Parity check passed?
<b>DI</b>	6, Build time option to invert data from the core, 1 is active.
<b>Delay</b>	5, Message had a 4uS delay.
<b>irq_en</b>	4, 1 when the IRQ is enabled by <b>CONTROL_REG</b>
<b>tx_full</b>	3, When 1 the tx fifo is full.
<b>tx_empty</b>	2, When 1 the tx fifo is empty.
<b>rx_full</b>	1, When 1 the rx fifo is full.
<b>rx_valid</b>	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

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

## INSTANTIATED MODULES

### inst\_axis\_1553\_encoder

```
axis_1553_encoder #(
    CLOCK_SPEED(CLOCK_SPEED),
    SAMPLE_RATE(SAMPLE_RATE)
) inst_axis_1553_encoder ( .aclk(clk), .arstn(rstn), .s_axis_tdata(tx_rdata)
```

Encode incoming AXIS data into a differential 1553 data stream

## inst\_axis\_1553\_decoder

```
axis_1553_decoder #(
    CLOCK_SPEED(CLOCK_SPEED),
    SAMPLE_RATE(SAMPLE_RATE),
    BIT_SLICE_OFFSET(BIT_SLICE_OFFSET),
    INVERT_DATA(INVERT_DATA),
    SAMPLE_SELECT(SAMPLE_SELECT)
) inst_axis_1553_decoder ( .aclk(clk), .arstn(rstn), .m_axis_tdata(m_axis_tdata),
```

Decode incoming differential 1553 data stream to AXIS data format.

## inst\_rx\_fifo

```
fifo #(
    FIFO_DEPTH(FIFO_DEPTH),
    BYTE_WIDTH(BUS_WIDTH),
    COUNT_WIDTH(8),
    FWFT(1),
    RD_SYNC_DEPTH(0),
    WR_SYNC_DEPTH(0),
    DC_SYNC_DEPTH(0),
    COUNT_DELAY(0),
    COUNT_ENA(0),
    DATA_ZERO(0),
    ACK_ENA(0),
    RAM_TYPE("block")
) inst_rx_fifo ( .rd_clk(clk), .rd_rstn(rstn & r_rstn_rx_delay[0]), .rd_en(
```

Buffer up to 16 items output from the axis\_1553\_encoder.

## inst\_tx\_fifo

```
fifo #(
    FIFO_DEPTH(FIFO_DEPTH),
    BYTE_WIDTH(BUS_WIDTH),
```

```

COUNT_WIDTH(8),
FWFT(1),
RD_SYNC_DEPTH(0),
WR_SYNC_DEPTH(0),
DC_SYNC_DEPTH(0),
COUNT_DELAY(0),
COUNT_ENA(0),
DATA_ZERO(0),
ACK_ENA(0),
RAM_TYPE("block")
) inst_tx_fifo ( .rd_clk(clk), .rd_rstn(rstn & r_rstn_tx_delay[0]), .rd_en(s

```

Buffer up to 16 items to input to the axis\_1553\_decoder.