

AXIS\_1553



June 26, 2025

Jay Convertino

# Contents

<b>1 Usage</b>	<b>2</b>
1.1 Introduction . . . . .	2
1.2 Dependencies . . . . .	2
1.2.1 fusesoc_info Depenecies . . . . .	2
1.3 In a Project . . . . .	2
<b>2 Architecture</b>	<b>3</b>
2.1 MIL-STD-1553 . . . . .	3
2.2 Encoding (transmit) Method . . . . .	3
2.3 Decoding (receive) Method . . . . .	4
<b>3 Building</b>	<b>4</b>
3.1 fusesoc . . . . .	4
3.2 Source Files . . . . .	5
3.2.1 fusesoc_info File List . . . . .	5
3.3 Targets . . . . .	5
3.3.1 fusesoc_info Targets . . . . .	5
3.4 Directory Guide . . . . .	5
<b>4 Simulation</b>	<b>6</b>
4.1 cocotb . . . . .	6
<b>5 Module Documentation</b>	<b>7</b>
5.1 axis_1553 . . . . .	8
5.2 tb_cocotb python . . . . .	15
5.3 tb_cocotb verilog . . . . .	18

# 1 Usage

## 1.1 Introduction

AXIS 1553 is a transmit and receive for the MIL-STD-1553 bus. This core can run at full duplex even though MIL-STD-1553 is half duplex. This core provides a simple axis streaming interface that uses tuser to extend the bus to allow for command and data syncs to be chosen, along with setting or indicating more than a 4us delay. There are additional signals for frame errors, sync only, and parity errors. These can be used to manage issues that present themselves to the core. Internally this core generates its own enables to cycle data out at its synthetic sample rate. Data is transmitted based on a 1 MHz clock, but is synthetically generated to 2 Mhz sample rate.

## 1.2 Dependencies

The following are the dependencies of the cores.

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

### 1.2.1 fusesoc\_info Dependencies

- dep
  - AFRL:clock:mod\_clock\_ena\_gen:1.1.1
  - AFRL:utility:helper:1.0.0
  - AFRL:simple:piso:1.0.1
  - AFRL:simple:sipo:1.0.1

## 1.3 In a Project

Connect the device to your AXIS bus. TUSER is used to set or check various data status such as command/data packet mode.

TDATA input contains the 16 bit data payload. TUSER is a 5 bit command register that can take input or provide output that is a description what type of sync (command or data) and other options described below.

TUSER = S,D,TTY (4 downto 0)

- TTY = TYPE OF DATA

- 000 N/A (IF DATA VALID, SYNC IS NOT VALID AND DATA IS NOT AS WELL)
- 001 REG (NOT IMPLEMENTED)
- 010 DATA
- 100 CMD/STATUS
- D = DELAY ENABLED
  - 1 = 4 us delay enabled.
  - 0 = no delay between transmissions.
- S = SYNC ONLY
  - 1 = Sync only detected
  - 0 = Standard message

## 2 Architecture

This core is made up of the following modules.

- **axis\_1553** Interface AXIS to PMOD1553 device.
- **mod\_clock\_ena\_gen** Generate enable pulse at required sample rate.
- **PISO** Parallel input serial output.
- **SIPO** Serial input parallel output.

### 2.1 MIL-STD-1553

In this core the data is encoded using Manchester II (G.E. Thomas) method. IEEE 802.3 uses a XOR with a clock and data. Manchester II uses a XNOR clock and data. The sync pulse is a non-manchester compliant part of the transmission that is only for detecting the start of the frame and what type of frame is incoming. The two types are data, and command/status. All messages are terminated with a odd parity bit.

### 2.2 Encoding (transmit) Method

MIL-STD-1553 data is generated using a combinatorial process for the XNOR. This XNOR is performed using the data and a synthetic clock of 1 MHz at a 2 MHz sample rate (minimum for the digital waveform). This is then concatenated with a pre-defined sync that is chosen

based upon TUSER, with a generated parity bit XNOR with the synth clock. This is then loaded into the PISO core and cycled out at the sample rate by the mod\_clock\_ena\_gen for tx enable pulse. If TUSER had set a delay the mod\_clock\_ena\_gen for tx will be put into a hold state till the counter has finished. Once all the data has been sent the AXIS input will become ready again. The transmit output will set the differential lines to zero meaning there is no difference in output and there is no data.

## **2.3 Decoding (receive) Method**

MIL-STD-1553 data is input into the SIPO core. The mod\_clock\_ena\_gen for RX enable is cleared and kept on hold till the receive input is in a differential state. The state of the signals being identical means there is no data being received (or transmitted). A few different conditions can arise. If the counter is less then the needed number of bits, and the diff in RX is no longer present then a sync only detection is made or a frame error has happened. If the total number of bits are captured then the combinatorial XNOR decoder has its output sampled and placed into tdata and tuser properly. If the delay has been longer then 4us since the last message the TUSER bit will be set to 1, otherwise there was no delay and it is 0.

# **3 Building**

The AXIS 1553 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 fusesoc\_info File List

- src
  - src/axis\_1553.v
- tb\_cocotb
  - 'tb/tb\_cocotb.py': 'file\_type': 'user', 'copyto': '.'
  - 'tb/tb\_cocotb.v': 'file\_type': 'verilogSource'

## 3.3 Targets

### 3.3.1 fusesoc\_info Targets

- default
  - Info: Default for IP intergration.
- lint
  - Info: Lint with Verible
- sim\_cocotb
  - Info: Cocotb unit tests

de

## 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 cocotb
  - **cocotb** testbench files

## 4 Simulation

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

### 4.1 cocotb

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

```
$ pip install cocotb
$ pip install cocotbext-axi
$ pip install cocotbext-mil_std_1553
```

Then you must use the cocotb sim target. The targets above can be run with various bus and fifo parameters.

```
$ fusesoc run --target sim_cocotb AFRL:device_converter:axis_1553:1.0.0
```

## 5 Module Documentation

- **axis\_1553** Interfaces AXIS to the PMOD1553.

The next sections document the module in great detail.



## axis\_1553.v

---

### AUTHORS

---

JAY CONVERTINO

---

### DATES

---

2025/06/24

---

### INFORMATION

---

#### Brief

---

AXIS 1553 core

#### License MIT

---

Copyright 2025 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.

## axis\_1553

---

```
module axis_1553 #(
  parameter
  CLOCK_SPEED
  =
  20000000,
  parameter
  RX_BAUD_DELAY
  =
  0,
  parameter
  TX_BAUD_DELAY
  =
  0
)
```

```

input
wire
aclk,
input
wire
arstn,
output
wire
parity_err,
output
wire
frame_err,
input
wire
[15:0]
s_axis_tdata,
input
wire
[ 4:0]
s_axis_tuser,
input
wire
s_axis_tvalid,
output
wire
s_axis_tready,
output
wire
[15:0]
m_axis_tdata,
output
wire
[ 4:0]
m_axis_tuser,
output
wire
m_axis_tvalid,
input
wire
m_axis_tready,
output
wire
tx_active,
output
wire
[ 1:0]
tx_diff,
input
wire
[ 1:0]
rx_diff
)

```

AXIS 1553, simple core for encoding and decoding 1553 bus messages.

## Parameters

<b>CLOCK_SPEED</b> parameter	This is the aclk frequency in Hz
<b>RX_BAUD_DELAY</b> parameter	Delay in rx baud enable. This will delay when we sample a bit (default is midpoint when rx delay is 0).
<b>TX_BAUD_DELAY</b> parameter	Delay in tx baud enable. This will delay the time the bit output starts.

## Ports

<b>aclk</b> input wire	Clock for AXIS
<b>arstn</b> input wire	Negative reset for AXIS
<b>parity_err</b> output wire	Indicates error with parity check for receive (active high)
<b>frame_err</b> output wire	Indicates the diff line went to no diff before data capture finished.
<b>s_axis_tdata</b> input wire [15:0]	Input data for UART TX.
<b>s_axis_tuser</b> input wire [4:0]	Information about the AXIS data {S,D,TYY} (4:0) Bits explained below:

```
- S = SYNC ONLY (4)
  - 1 = Send only a sync pulse specified by TYY
  - 0 = Send normal sync + data.
- D = DELAY ENABLED (3)
  - 1 = Make sure there is a delay of 4us
  - 0 = Send out immediately
- TYY = TYPE OF DATA (2:0)
  - 000 = NA
  - 001 = REG (NOT IMPLIMENTED)
  - 010 = DATA
  - 100 = CMD/STATUS
```

s\_axis\_tvalid - When set active high the input data is valid s\_axis\_tready - When active high the device is ready for input data. m\_axis\_tdata - Output data from UART RX m\_axis\_tuser - Information about the AXIS data {S,D,TYY} (4:0)

Bits explained below:

```
- S = SYNC ONLY (4)
  - 1 = Only received a sync pulse specified by TYY
  - 0 = Normal sync + data received.
- D = DELAY BEFORE DATA (3)
  - 1 = Delay of 4us or more before data
  - 0 = No delay between data
- TYY = TYPE OF DATA (2:0)
  - 000 NA
  - 001 REG (NOT IMPLIMENTED)
  - 010 DATA
  - 100 CMD/STATUS
```

m\_axis\_tvalid - When active high the output data is valid m\_axis\_tready - When set active high the output device is ready for data. tx\_active - Active high indicates transmit is in progress. tx\_diff - transmit for 1553 (output to RX) rx\_diff - receive for 1553 (input from TX)

## BASE\_1553\_CLOCK\_RATE

```
localparam integer BASE_1553_CLOCK_RATE = 1000000
```

1553 base clock rate

## BASE\_1553\_SAMPLE\_RATE

```
localparam integer BASE_1553_SAMPLE_RATE = 2000000
```

Sample rate to use for the 1553 bus, set to 2 MHz

## SAMPLES\_PER\_MHZ

---

```
localparam integer SAMPLES_PER_MHZ = BASE_1553_SAMPLE_RATE /  
    BASE_1553_CLOCK_RATE
```

sample rate of 2 MHz to capture transmission bits at

## cycles\_per\_mhz

---

```
localparam integer CYCLES_PER_MHZ = CLOCK_SPEED / BASE_1553_CLOCK_RATE
```

calculate the number of cycles the clock changes per period

## BIT\_RATE\_PER\_MHZ

---

```
localparam integer BIT_RATE_PER_MHZ = SAMPLES_PER_MHZ
```

bit rate per mhz

## DELAY\_TIME

---

```
localparam integer DELAY_TIME = CYCLES_PER_MHZ * 4
```

delay time, 4 is for 4 us (min 1553 time)

## SYNC\_BITS\_PER\_TRANS

---

```
localparam integer SYNC_BITS_PER_TRANS = 3
```

sync bits per transmission

## SYNTH\_SYNC\_BITS\_PER\_TRANS

---

```
localparam integer SYNTH_SYNC_BITS_PER_TRANS = SYNC_BITS_PER_TRANS *  
    BIT_RATE_PER_MHZ
```

sync pulse length

## PARITY\_BITS\_PER\_TRANS

---

```
localparam integer PARITY_BITS_PER_TRANS = 1
```

parity bits per transmission

## SYNTH\_PARITY\_BITS\_PER\_TRANS

---

```
localparam integer SYNTH_PARITY_BITS_PER_TRANS = PARITY_BITS_PER_TRANS *  
BIT_RATE_PER_MHZ
```

synth parity bits per transmission

## DATA\_BITS\_PER\_TRANS

---

```
localparam integer DATA_BITS_PER_TRANS = 16
```

data bits per transmission

## SYNTH\_DATA\_BITS\_PER\_TRANS

---

```
localparam integer SYNTH_DATA_BITS_PER_TRANS = DATA_BITS_PER_TRANS *  
BIT_RATE_PER_MHZ
```

synth data bits per transmission

## BITS\_PER\_TRANS

---

```
localparam integer BITS_PER_TRANS = DATA_BITS_PER_TRANS +  
PARITY_BITS_PER_TRANS
```

non sync bits per transmission

## TOTAL\_BITS\_PER\_TRANS

---

```
localparam integer TOTAL_BITS_PER_TRANS = DATA_BITS_PER_TRANS +  
PARITY_BITS_PER_TRANS + SYNC_BITS_PER_TRANS
```

bits per transmission with sync

## SYNTH\_BITS\_PER\_TRANS

---

```
localparam integer SYNTH_BITS_PER_TRANS = SYNTH_DATA_BITS_PER_TRANS +  
SYNTH_PARITY_BITS_PER_TRANS
```

synth bits per trans without sync

## TOTAL\_SYNTH\_BITS\_PER\_TRANS

---

```
localparam integer TOTAL_SYNTH_BITS_PER_TRANS =  
SYNTH_DATA_BITS_PER_TRANS + SYNTH_SYNC_BITS_PER_TRANS +  
SYNTH_PARITY_BITS_PER_TRANS
```

---

synth bits per trans with sync

## TOTAL\_SYNTH\_BYTES\_PER\_TRANS

---

```
localparam integer TOTAL_SYNTH_BYTES_PER_TRANS =  
    TOTAL_SYNTH_BITS_PER_TRANS/8
```

sync bits per trans with sync

## BIT\_PATTERN

---

```
localparam [(  
    BIT_RATE_PER_MHZ  
)-1:0]BIT_PATTERN = {{BIT_RATE_PER_MHZ/2{1'b1}}, {BIT_RATE_PER_MHZ/2{1'b0}}}
```

create the bit pattern. This is based on outputting data on the negative and positive. This allows the encoder to run down to 1 mhz.

## SYNTH\_CLK

---

```
localparam [SYNTH_DATA_BITS_PER_TRANS-1:0]SYNTH_CLK = {  
    DATA_BITS_PER_TRANS{BIT_PATTERN}  
}
```

synth clock is the clock constructed by the repeating the bit pattern. this is intended to be a representation of the clock. Captured at a bit\_rate\_per\_mhz of a 1mhz clock.

## SYNC\_CMD\_STAT

---

```
localparam [SYNTH_SYNC_BITS_PER_TRANS-1:0]SYNC_CMD_STAT = {  
    SYNTH_SYNC_BITS_PER_TRANS/2{1'b1}},  
    SYNTH_SYNC_BITS_PER_TRANS/2{1'b0}}}
```

sync pulse command

## SYNC\_DATA

---

```
localparam [SYNTH_SYNC_BITS_PER_TRANS-1:0]SYNC_DATA = {  
    SYNTH_SYNC_BITS_PER_TRANS/2{1'b0}},  
    SYNTH_SYNC_BITS_PER_TRANS/2{1'b1}}}
```

sync pulse data

## CMD\_DATA

---

```
localparam CMD_DATA = 3'b010
```

tuser decode for data

## CMD\_DATA

---

tuser decode for command

## INSTANTIATED MODULES

---

### clk\_gen\_tx

---

Generates TX clock at sample rate (BASE\_1553\_SAMPLE\_RATE).

### clk\_gen\_rx

---

Generates RX clock at sample rate (BASE\_1553\_SAMPLE\_RATE).

### inst\_sipo

---

Captures RX data for 1553 receive

### inst\_piso

---

Generates TX data for 1553 transmit

# tb\_cocotb.py

---

## AUTHORS

---

JAY CONVERTINO

---

## DATES

---

2025/03/04

---

## INFORMATION

---

### Brief

---

Cocotb test bench

### License MIT

---

Copyright 2025 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.

## FUNCTIONS

---

### random\_bool

---

```
def random_bool()
```

Return a infinite 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 tx

---

Coroutine that is identified as a test routine. This routine tests by sending a incrementing value as a command and then as data, no delay between the two is inserted by the core.

#### Parameters

**dut**     Device under test passed from cocotb.

### increment test rx

---

Coroutine that is identified as a test routine. This routine tests by sending a incrementing value as a command and then as data.

#### Parameters

**dut**     Device under test passed from cocotb.

### increment test tx delay

---

Coroutine that is identified as a test routine. This routine tests by sending a incrementing value as a command and then as data, delay between the two is inserted by the core.

#### Parameters

**dut**     Device under test passed from cocotb.

### increment test tx delay

---

Coroutine that is identified as a test routine. This routine tests by sending a incrementing value as a command and then as 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.v

---

## AUTHORS

---

JAY CONVERTINO

---

## DATES

---

2025/06/24

---

## INFORMATION

---

### Brief

---

Test bench wrapper for cocotb

### 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.

## tb\_cocotb

---

```
module tb_cocotb #(
  parameter
    CLOCK_SPEED
    =
    20000000,
  parameter
    RX_BAUD_DELAY
    =
    0,
  parameter
    TX_BAUD_DELAY
    =
    0
)
```

```

input
wire
aclk,
input
wire
arstn,
output
wire
parity_err,
output
wire
frame_err,
input
wire
[15:0]
s_axis_tdata,
input
wire
[ 4:0]
s_axis_tuser,
input
wire
s_axis_tvalid,
output
wire
s_axis_tready,
output
wire
[15:0]
m_axis_tdata,
output
wire
[ 4:0]
m_axis_tuser,
output
wire
m_axis_tvalid,
input
wire
m_axis_tready,
output
wire
tx_active,
output
wire
[ 1:0]
tx_diff,
input
wire
[ 1:0]
rx_diff
)

```

## Parameters

<b>CLOCK_SPEED</b> parameter	This is the aclk frequency in Hz
<b>RX_BAUD_DELAY</b> parameter	Delay in rx baud enable. This will delay when we sample a bit (default is midpoint when rx delay is 0).
<b>TX_BAUD_DELAY</b> parameter	Delay in tx baud enable. This will delay the time the bit output starts.

## Ports

<b>ack</b> input wire	Clock for AXIS
<b>arstn</b> input wire	Negative reset for AXIS
<b>parity_err</b> output wire	Indicates error with parity check (active high)
<b>frame_err</b> output wire	Indicates the diff line went to no diff before data capture finished.
<b>s_axis_tdata</b> input wire [15:0]	Input data for UART TX.
<b>s_axis_tuser</b> input wire [4:0]	Information about the AXIS data {S,D,TYY} (4:0) Bits explained below:

- S = SYNC ONLY (4)
  - 1 = Send only a sync pulse specified by TYY
  - 0 = Send normal sync + data.
- D = DELAY ENABLED (3)
  - 1 = Make sure there is a delay of 4us
  - 0 = Send out immediately
- TYY = TYPE OF DATA (2:0)
  - 000 = NA
  - 001 = REG (NOT IMPLIMENTED)
  - 010 = DATA
  - 100 = CMD/STATUS

s\_axis\_tvalid - When set active high the input data is valid s\_axis\_tready - When active high the device is ready for input data. m\_axis\_tdata - Output data from UART RX m\_axis\_tuser - Information about the AXIS data {S,D,TYY} (4:0)

Bits explained below:

- S = SYNC ONLY (4)
  - 1 = Only received a sync pulse specified by TYY
  - 0 = Normal sync + data received.
- D = DELAY BEFORE DATA (3)
  - 1 = Delay of 4us or more before data
  - 0 = No delay between data
- TYY = TYPE OF DATA (2:0)
  - 000 NA
  - 001 REG (NOT IMPLIMENTED)
  - 010 DATA
  - 100 CMD/STATUS

m\_axis\_tvalid - When active high the output data is valid m\_axis\_tready - When set active high the output device is ready for data. tx\_active - Active high indicates transmit is in progress. tx\_diff - transmit for 1553 (output to RX) rx\_diff - receive for 1553 (input from TX)

## INSTANTIATED MODULES

---

### dut

---

Device under test, axis\_1553