AXIS_1553

June 25, 2025

Jay Convertino

Contents

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 interace that uses tuser to extend the bus to allow for command and data syncs to be choosen, 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 themeselves 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 sythetically 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 Depenecies

- 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,TYY (4 downto 0)

- TYY = 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-machester 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 contatenated with a pre-defined sync that is choosen

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

axis_1553

```
module axis_1553 #(
parameter
CLOCK_SPEED
 =
2000000,
parameter
RX_BAUD_DELAY
 =
0,
parameter
TX_BAUD_DELAY
 =
0
) ( input wire aclk, input wire arstn, output wire parity_err, output wire t
```

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

Parameters

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

Ports

aclk	Clock for AXIS
arstn	Negative reset for AXIS
parity_err	Indicates error with parity check for receive (active high)
frame_err	Indicates the diff line went to no diff before data catpure finished.
s_axis_tdata	Input data for UART TX.
s_axis_tuser	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 immediatly
- 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 caputre 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 outputing 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

uart_baud_gen_tx

Generates TX BAUD rate for UART modules using modulo divide method.

uart_baud_gen_rx

```
mod_clock_ena_gen #(
                                                                                .
CLOCK_SPEED(CLOCK_SPEED),
                                                                                .
DELAY(RX_BAUD_DELAY)
) uart_baud_gen_rx ( .clk(aclk), .rstn(arstn), .start0(1'b0), .clr(s_clr_cl
```

Generates RX BAUD rate for UART modules using modulo divide method.

inst_sipo

```
sipo #(
                                                                                .
BUS_WIDTH(TOTAL_SYNTH_BYTES_PER_TRANS)
) inst_sipo ( .clk(aclk), .rstn(arstn), .ena(ena_rx), .rev(1'b0), .load(r_
```

Captures RX data for 1553 receive

inst_piso

```
piso #(
                                                                                .
BUS_WIDTH(TOTAL_SYNTH_BYTES_PER_TRANS),
                                                                                .
DEFAULT_RESET_VAL(0),
```

```
                                                                                .
 DEFAULT_SHIFT_VAL(0)
) inst_piso ( .clk(aclk), .rstn(arstn), .ena(ena_tx), .rev(1'b0), .load(s_ax
```

Generates TX data for 1553 transmit

tb_cocotb.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 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

tb_cocotb

```
module tb_cocotb #(
parameter
CLOCK_SPEED
 =
2000000,
parameter
RX_BAUD_DELAY
 =
0,
parameter
TX_BAUD_DELAY
 =
0
) ( input wire aclk, input wire arstn, output wire parity_err, output wire f
```

Parameters

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

Ports

aclk	Clock for AXIS
arstn	Negative reset for AXIS
parity_err	Indicates error with parity check (active high)
frame_err	Indicates the diff line went to no diff before data catpure finished.
s_axis_tdata	Input data for UART TX.
s_axis_tuser	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 immediatly
- 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

```
 axis_1553 #(
                                                                                .
 CLOCK_SPEED(CLOCK_SPEED),
                                                                                .
 RX_BAUD_DELAY(RX_BAUD_DELAY),
                                                                                .
 TX_BAUD_DELAY(TX_BAUD_DELAY)
) dut ( .aclk(aclk), .arstn(arstn), .parity_err(parity_err), .frame_err(fram
```

Device under test, axis_1553