

# AXIS\_1553\_ENCODER



November 14, 2024

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 Encoding Method . . . . .	3
<b>3 Building</b>	<b>4</b>
3.1 fusesoc . . . . .	5
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 . . . . .	6
<b>4 Simulation</b>	<b>7</b>
4.1 iverilog . . . . .	7
4.2 cocotb . . . . .	7
<b>5 Module Documentation</b>	<b>8</b>
5.1 axis_1553_encoder . . . . .	9

# 1 Usage

## 1.1 Introduction

AXIS 1553 Encoder is a core for taking AXIS data and encoding for output to the PMOD1553 device. The output is a TTL differential signal. Meaning when diff[0] is 1 diff[1] is 0. This core also includes a diff enable which allows for mux switching to the transmit (encoder) when active.

## 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:utility:helper:1.0.0
- dep\_tb
  - AFRL:simulation:axis\_stimulator
  - AFRL:utility:sim\_helper

## 1.3 In a Project

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

TDATA input should contain the 16 bit data payload. TUSER is a 8 bit command register that takes a description what type of data it is (command or data) and other options described below.

TUSER = TYY,NA,D,I,P (7 downto 0)

- TYY = TYPE OF DATA
  - 000 N/A
  - 001 REG (NOT IMPLEMENTED)
  - 010 DATA
  - 100 CMD/STATUS

- NA = RESERVED FOR FUTURE USE.
- D = DELAY ENABLED
  - 1 = 4 us delay enabled.
  - 0 = no delay between transmissions.
- I = INVERT DATA
  - 1 = Invert data.
  - 0 = Normal data.
- P = PARITY
  - 1 = ODD
  - 0 = EVEN

## 2 Architecture

This core is made up of a single module.

- **axis\_1553\_encoder** Interface AXIS to PMOD1553 device (see core for documentation).

### 2.1 Encoding Method

This core has 4 always blocks that are sensitive to the positive clock edge. They are

- **pause counter** Inserts delays between transmits to comply with 4us spacing.
- **axis data input** Deals with AXIS bus input data to the core based on its current state.
- **data processing** In charge of the state machine and processing of input data to 1553 non-differential bitstream.
- **differential data output** Output processed data from a non-differential bitstream into a differential one.

Pause counter simply gets reset at each transmit state to its initial value. Then once it is out of the transmit state it starts its countdown. The data processing block checks the pause counter, if it is not 0 we will not transition to the transmit state.

AXIS data input does exactly as it says. Take input data in the data capture state and register it. All other states simply ignore the input data.

Data processing block does most of the work. It is in charge of the state machine state and generating the 1553 non-differential bitstream. Essentially the state machine does the following:

1. Startup in error state, which does to the default handler that puts it into data capture.
2. Wait for data input from the AXIS streaming input, once valid data is input. Then go to the data invert state.
3. Check the TUSER command if a data invert is necessary. Always go to the parity generation state.
4. Generate parity bit based upon the input data using xor method. Then move on to process state.
5. In process state take the TUSER and TDATA input and process it into register. This register will contain a non-differential 1553 bitstream.
  - Check if pause check is enabled, if it isn't skip it.
  - Based on TUSER insert the needed sync pulse of command or data. 0 is inserted if a invalid selection is made.
  - Check parity option for odd or even and insert parity bit based upon selection and generated parity bit.
  - Generate machester data using XOR method, this is done with a synthetic clock xor with the input data over the number of samples needed. The sythetic clock is contained in the register used for the result.
6. Check for pause (if it wasn't skipped) when counter is 0, move to transmit state.
7. In transmit state wait for differential data output to process all of the register data. Once all conditions are meet, move to the data capture state to wait for more input data to encode.

Differential data output waits for the transmit state, once it is reached the core will begin outputing the registered data generated by data processing in the process state. It will output a TTL differential version of the data and keep enable diff high so a mux can be switched for transmit mode. Once the data, and by extension counters, are exhausted the block will wait for the next time transmit state is reached.

### **3 Building**

The AXIS 1553 Encoder 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 fusesoc\_info File List**

- src
  - Type: verilogSource
  - src/axis\_1553\_encoder.v
- tb
  - 'tb/tb\_1553\_enc.v': 'file\_type': 'verilogSource'

### **3.3 Targets**

#### **3.3.1 fusesoc\_info Targets**

- default
  - Info: Default for IP integration.
  - src
  - dep
- sim
  - Info: Simulation using icarus as the default.
  - src
  - dep
  - tb
  - dep\_tb

### 3.4 Directory Guide

Below highlights important folders from the root of BUS UART.

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

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

The next sections document the module in great detail.

# axis\_1553\_encoder.v

---

## AUTHORS

---

JAY CONVERTINO

---

## DATES

---

2021/05/17

---

## INFORMATION

---

### Brief

---

AXIS MIL-STD-1553 ENCODER

### License MIT

---

Copyright 2021 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\_encoder

---

```
module axis_1553_encoder #(
    parameter
    CLOCK_SPEED
    =
    2000000,
    parameter
    SAMPLE_RATE
    =
    2000000
) ( input aclk, input arstn, input [15:0] s_axis_tdata, input s_axis_tvalid,
```

AXI streaming to MIL-STD-1553 encoder. This encoder can be used at 2 Mhz or above. TDATA is 16 bit data to be transmitted. TUSER sets how the core works.

## Parameters

<b>CLOCK_SPEED</b> parameter	This is the aclk frequency in Hz, must be 2 MHz or above.
<b>SAMPLE_RATE</b> parameter	2 MHz or above rate that is an even divisor of CLOCK_SPEED

## Ports

<b>aclk</b>	Clock for all logic
<b>arstn</b>	Negative reset
<b>s_axis_tdata</b>	Input data for 1553 encoder.
<b>s_axis_tvalid</b>	When set active high the input data is valid.
<b>s_axis_tuser</b>	Information about the AXIS data {TYY,NA,I,P} Bits explained below:

```
- TYY = TYPE OF DATA
  - 000 = NA
  - 001 = REG (NOT IMPLIMENTED)
  - 010 = DATA
  - 100 = CMD/STATUS
- D = DELAY ENABLED
- I = INVERT DATA
- P = PARITY
  - 1 = ODD
  - 0 = EVEN
```

<b>s_axis_tready</b>	When active high the device is ready for input data.
<b>diff</b>	Output data in TTL differential format.
<b>en_diff</b>	When diff is valid data, this is active high and can be used to switch a mux.

## base\_1553\_clock\_rate

```
localparam integer base_1553_clock_rate = 1000000
```

1553 base clock rate

## samples\_per\_mhz

```
localparam integer samples_per_mhz = SAMPLE_RATE / base_1553_clock_rate
```

sample rate 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

## **samples\_to\_skip**

---

```
localparam integer samples_to_skip = (  
    (cycles_per_mhz > samples_per_mhz) ? cycles_per_mhz / samples_per_mhz -  
    1  
    :  
    0  
)
```

calculate the number of samples to skip

## **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\_pulse\_len**

---

```
localparam integer sync_pulse_len = bit_rate_per_mhz * 3
```

sync pulse length

## **bits\_per\_trans**

---

```
localparam integer bits_per_trans = 20
```

bits per transmission

## **synth\_bits\_per\_trans**

---

```
localparam integer synth_bits_per_trans = (  
    bits_per_trans*bit_rate_per_mhz  
)
```

synth bits per trans

## **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_bits_per_trans-1:0]synth_clk = {  
  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 [sync_pulse_len-1:0]sync_cmd_stat = {  
  sync_pulse_len/2{1'b0}},  
  sync_pulse_len/2{1'b1}}  
}
```

sync pulse command

## sync\_data

---

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

sync pulse data

## cmd\_data

---

```
localparam cmd_data = 3'b010
```

tuser decode for data

## cmd\_data

---

tuser decode for command

## cmd\_data

---

enable diff output

## STATE MACHINE

---

Constants that makeup the encoder state machine.

### data\_cap

---

```
localparam data_cap = 3'd1
```

data capture

### data\_invert

---

```
localparam data_invert = 3'd2
```

invert data

### parity\_gen

---

```
localparam parity_gen = 3'd3
```

parity generator

### process

---

```
localparam process = 3'd4
```

command processor

### pause\_ck

---

```
localparam pause_ck = 3'd5
```

check for pause (4us)

### trans

---

```
localparam trans = 3'd6
```

transmit data

### error

---

```
localparam error = 3'd0
```

someone made a whoops