

# 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