AXIS_1553_DECODER

November 21, 2024

Jay Convertino

Contents

1 Usage

1.1 Introduction

AXIS 1553 decoder is a core for decoding MIL-STD-1553 signals demodulated by the PMOD1553 device to AXIS data. The input is a TTL differential signal. Meaning when diff[0] is 1 diff[1] is 0.

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:utility:helper:1.0.0
- dep_tb
  - AFRL:simulation:axis_stimulator

1.3 In a Project

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

TDATA output will contain the 16 bit data payload. TUSER is a 8 bit command register that outputs 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_decoder** Interface AXIS to PMOD1553 device (see core for documentation).

2.1 Decoding Method

This core has 5 always blocks that are sensitive to the positive clock edge. They are

- **pause counter** Checks for delays between receives to comply with 4us spacing.
- **axis data output** Deals with AXIS bus output data from the core based on its current state.
- **data processing** In charge of the state machine and processing of 1553 non-differential bitstream to AXIS output data.
- **skip counter** Aligns the sample counters based on the differential signal so sampling is aligned correctly.
- **differential data input** Process data from a differential input into a non-differential one.

Pause counter simply gets reset when not in the data capture or diff wait states. In those states the counter checks if the diff is equal (no input data) and counts down till it reaches zero. This allows for detections of badly formed packets that ignore the spacing.

AXIS data output does exactly as it says. Once in the transmit state, in this case this means we can transmit the decoded 1553 data over the AXIS bus. Once we are out of this state the data is wiped.

Data processing block does most of the work. It is in charge of the state machine state and decoding the 1553 non-differential bit-stream. Command bit refers to the TUSER output. Essentially the state machine does the following:

1. Startup in error state, which does to the default handler that puts it into diff wait.
2. In diff wait check for a change in the diff input signals. Once that has happened goto the data capture state.
3. In data capture wait for the trans_counter to reach zero. Once at zero check the sync pulse type. Then go to data reduce, unless the pause_counter timed out in the data capture state, then we go back to the diff wait.
4. In data reduce use the bit slice offset to capute a bit from the stream of data. Capture the parity bit, and set the output command bits for invert and delay to the correct state. Move on to parity gen.
5. In parity gen check the parity bit against the parity bit generated by the module. Set the command bit to tell the user if there is an error or not.
6. Trans just goes right back to diff wait. Only needs a cycle to let the AXIS data output to do its thing.

Skip counter is a set of conditions to reset skip counter to 0. If it is not reset the skip counter will increase. This allows us to sample the correct bit from the oversampled stream. The conditions to reset are:

1. If in diff wait state and no difference in diff input, hold skip counter at current value.
2. If pause counter is 0, and no difference in diff input, reset skip counter to 0.
3. If there is a positive edge on diff, reset skip counter to 0.
4. If there is a negative edge of diff, reset skip counter to 0.
5. Error? Reset registered to all ones and skip counter to 0.

Differential data input waits for the data capture state. In this state the capture is started and the signal is sampled at the input clock. This is reduced using a counter to pick the samples needed via the skip counter being compared to sample select. Captured data is then shifted into a register and the transmit counter is decremented. This continues till the transmit counter is 0, or the pause counter is 0 and the diff signal is equal. All other states reset the data registers for capturing data to its synthetic clock and the transmit counter back to its initial value before decrement.

3 Building

The AXIS 1553 decoder 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.

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_decoder.v
- tb
  - 'tb/tb_1553_dec.v': 'file_type': 'verilogSource'

3.3 Targets

3.3.1 fusesoc_info Targets

- default

  Info: Default for IP intergration.

- sim

  Info: Simulation only, defaults to icarus.

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 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_decoder** Interfaces AXIS to the PMOD1553.

The next sections document the module in great detail.

axis_1553_decoder.v

AUTHORS

JAY CONVERTINO

DATES

2021/05/24

INFORMATION

Brief

AXIS MIL-STD-1553 DECODER

License MIT

axis_1553_decoder

```
module axis_1553_decoder #(
parameter
CLOCK_SPEED
 =
20000000,
parameter
SAMPLE_RATE
 =
2000000,
parameter
BIT_SLICE_OFFSET
```

```
  =
0,
parameter
INVERT_DATA
  =
0,
parameter
SAMPLE_SELECT
  =
0
) ( input aclk, input arstn, output [15:0] m_axis_tdata, output m_axis_tval
```

This core is a MIL-STD-1553 to AXI streaming decoder. It uses the postive edge of a clock to sample data. This restricts the core to 2 Mhz and above for a sample clock.

Parameters

CLOCK_SPEED This is the aclk frequency in Hz, must be 2 MHz or above.
parameter

SAMPLE_RATE 2 MHz or above rate that is an even divisor of CLOCK_SPEED
parameter

BIT_SLICE_OFFSET Changes the bit that is selected for data reduction.
parameter

INVERT_DATA Will invert all decoded data.
parameter

SAMPLE_SELECT Changes the bit that is sampled for data capture.
parameter

Ports

aclk Clock for all logic

arstn Negative reset

m_axis_tdata Output data for 1553 decoder.

m_axis_tvalid When active high the output data is valid.

m_axis_tuser 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
- NA  = RESERVED FOR FUTURE USE.
- D   = DELAY BEFORE DATA
      - 1 = Delay of 4us or more before data
      - 0 = No delay between data
- P   = PARITY
      - 1 = GOOD
      - 0 = BAD
```

m_axis_tready When active high the destination device is ready for data.

diff Output data in TTL differential format.

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 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

delay_time

```
localparam integer delay_time = cycles_per_mhz * 4
```

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

samples_to_skip

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

calculate the number of samples to skip

round_SAMPLE_SELECT

```
localparam integer round_SAMPLE_SELECT = (
                                  (SAMPLE_SELECT == 0) ? samples_to_skip/
2
:
SAMPLE_SELECT % samples_to_skip
)
```

SAMPLE_SELECT rounded

bit_rate_per_mhz

```
localparam integer bit_rate_per_mhz = samples_per_mhz
```

bit rate per mhz

round_BIT_SLICE_OFFSET

```
localparam integer round_BIT_SLICE_OFFSET = (
                                  (cycles_per_mhz > samples_per_mhz) ? ((
BIT_SLICE_OFFSET == 0) ? bit_rate_per_mhz/4 : BIT_SLICE_OFFSET % bit_rate_p
:
0
)
```

pick the middle of the samples generated by default

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
)
```

sync bits per trans

sync_cmd_stat

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

Command sync pulse

sync_data

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

```
                                                                                {
sync_pulse_len/2{1'b0}}
}
```

Data sync pulse

cmd_data

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

data tuser encode

cmd_data

command tuser encode

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_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.

STATE MACHINE

Constants that makeup the decoder state machine.

diff_wait

```
localparam diff_wait = 5'h01
```

wait for diff

data_cap

```
localparam data_cap = 5'h03
```

data capture

data_reduce

```
localparam data_reduce = 5'h07
```

reduce data

parity_gen

```
localparam parity_gen = 5'h0F
```

parity generator

trans

```
localparam trans = 5'h1F
```

transmit data

error

```
localparam error = 5'h00
```

someone made a whoops