Note: This File contains the list of exclusions made while improving coverage

CTRL INSTANCE

1. Line number: 79

File name: `cfs_ctrl.v`

a. We have considered the CTRL size and offset to be 4,0 and size and offset of the incoming packets
is also 4,0 and we have also made md_tx_ready = 0.

b. Considering the push_ready = 0 because we have filled the TX FIFO and the push_ready from the
TX FIFO has become 0. push_valid is 1 because there are incoming data packets.

c. Once the last packet fills the TX FIFO:

i. Next packet is 4,0 then push_ready = 0 and push_valid = 1,
unaligned_bytes_processed = 4, unaligned_size = 4,
pop_valid and pop_ready = 1

ii. In if (pop_valid && pop_ready):
unaligned_bytes_processed = 0, unaligned_size = 4,
pop_ready = 0

iii. For the next packet:
push_ready = 0, push_valid = 1,
unaligned_bytes_processed = 0, unaligned_size = 4,
pop_valid = 1, pop_ready = 0

iv. The condition (unaligned_bytes_processed >= unaligned_size) fails and it executes the else block. It will not enter:
- if (pop_valid == 1 && pop_ready == 0), and
- it will also not fail at if (pop_valid == 1 && pop_ready == 1)

2. Line number: 145

File name: `cfs_ctrl.v`

a. We have considered the CTRL size and offset to be 2,0 and size and offset of the incoming packets
is also 4,0 and we have also made md_tx_ready = 1 (TX FIFO not full).

b. Considering the push_ready = 1 because we have not filled the TX FIFO and the push_ready from the
TX FIFO has become 1. push_valid is 1 because there are incoming data packets.

c.

i. Next packet is 4,0 then push_ready = 1 and push_valid = 1,
unaligned_bytes_processed = 2, unaligned_size = 4,
pop_valid = 1 and pop_ready = 0
(as incoming data size is more than CTRL size, pop_ready gets deasserted for 1 cycle in order for the data to be split and sent to the TX side)

ii. In if (pop_valid && pop_ready):
unaligned_bytes_processed = 2, unaligned_size = 4,
pop_ready = 0

iii. For the next packet:
push_ready = 1, push_valid = 1,
unaligned_bytes_processed = 2, unaligned_size = 4,
pop_valid = 1, pop_ready = 0

iv. The condition (unaligned_bytes_processed >= unaligned_size) fails and it executes the else block. It will not enter:
- if (pop_valid == 1 && pop_ready == 0), and
- it will also not fail at if (pop_valid == 1 && pop_ready == 1)

** The above two exclusions are made for the lines of code which are unhittable (pop_valid==1&&pop_ready==0)

3. Line number: 191

File name: `cfs_ctrl.v`

if (unaligned_bytes_processed >= unaligned_size) . This condition is not getting hit ; observed that it is unhittable after trying to send packets with various kinds of legal combinations with valid CTRL.SIZE and CTRL.OFFSET configuration. So all the branches and conditions inside this condition are excluded.

RX_FIFO INSTANCE

1)Expression (push_valid&&push_ready) is excluded because of the following reason:

—> Expression given in cfs_sync_fifo.v :
assign push_ready = push_valid & (~push_full)

ready=1 valid=1 not hitting which is clearly not possible from the above expression

TX_FIFO INSTANCE

1)Expression (pop_valid&&pop_ready) is excluded because of the following reason:

—> Expression given in cfs_sync_fifo.v :
assign pop_valid = ! pop_empty
—->Expression given in tx_ctrl.v:
assign pop_ready= pop_valid & md_tx_ready

ready=1 valid=1 not hitting which is clearly not possible from the above expression.

RX_CTRL INSTANCE

- Expression (md_rx_ready&&md_rx_valid&&md_rx_err) is excluded because the combinations with {Valid=0,Ready=1,Error=1}, {Valid=1, Ready=0, Error=1} is not hitting because inorder to detect an error packet and by the design and then assert error signal, the data sent (error data) must be sampled first which means valid should be high and then to receive the error packet by the design, the RX side must take the data into it; ready must be high.
- In short, to make error signal high , an error data must be sent which makes the valid signal to1 for data sampling and ready signal to 1 for data reception and then error signal is raised to 1.

TOGGLE COVERAGE EXCLUSIONS

a. Reserved bits mentioned in the design specification.
These bits always stay 0 and cannot be toggled to 1.

b. aligned_bytes_processed can have a maximum value up to three.
So aligned_bytes_processed[2] = 0 always;
cannot be toggled to 1 as aligned_bytes_processed = 4 is not possible.
There is no statement which is making aligned_bytes_processed = 4.