Frame Check Sequence (FCS) module

By: Mohamed Ayman Elsayed Mahmoud

Outline

Introduction

A frame check sequence (FCS) is an error-detecting code added to a frame in a communication protocol. Frames are used to send payload data from a source to a destination.

The FCS field contains a 16-bit ITU-T CRC. The FCS is calculated over MAC Header and MAC payload parts of the frame

The FCS shall be calculated using the following standard generator polynomial of degree 16

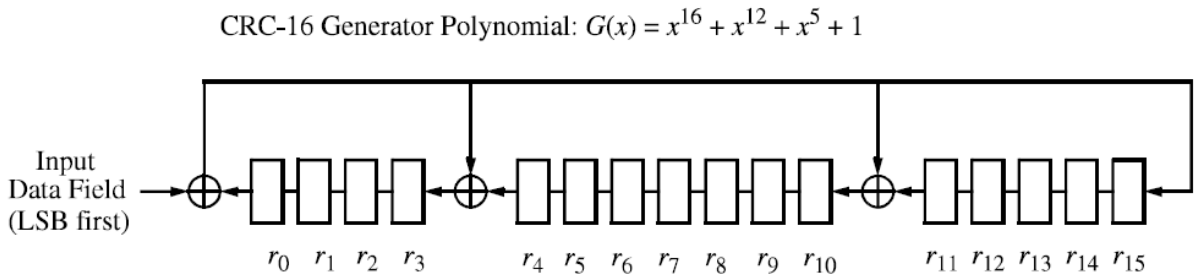

Figure 1 - FCS algorithm diagram

Block Interface

- Serial data are received by Input_Data port (MSB first).
- RST is active low asynchronous reset signal.
- CLK is system clock signal.
- Valid_Data signal is high for one clock cycle only when there is a new frame to detect first bit (MSB).
- Data_Size is the size of the frame (64 to 1024 Bits).
- Serial output data is sent by “OUT” port.
- Done signal is high for when the operation is done to detect last serial output bit.
- Valid_OUT signal is high for one clock cycle only to detect first output bit (LSB of FCS result)
- Busy signal is high during the operation until is done.

Figure 2 - Block interface

Ports list

Port	Direction	Width	Description
CLK	Input	1	System Clk
RST	Input	1	Active low asynchronous reset signal
Valid_Data	Input	1	Valid Signal
Data_Size	Input	10	Size of input data
Input_Data	Input	1	Serial Input Data Field
OUT	Output	1	Output Serial Data
Done	Output	1	The shifted operation is done
Valid_OUT	Output	1	Valid signal for output Data
Busy	Output	1	Busy signal

Block Diagram

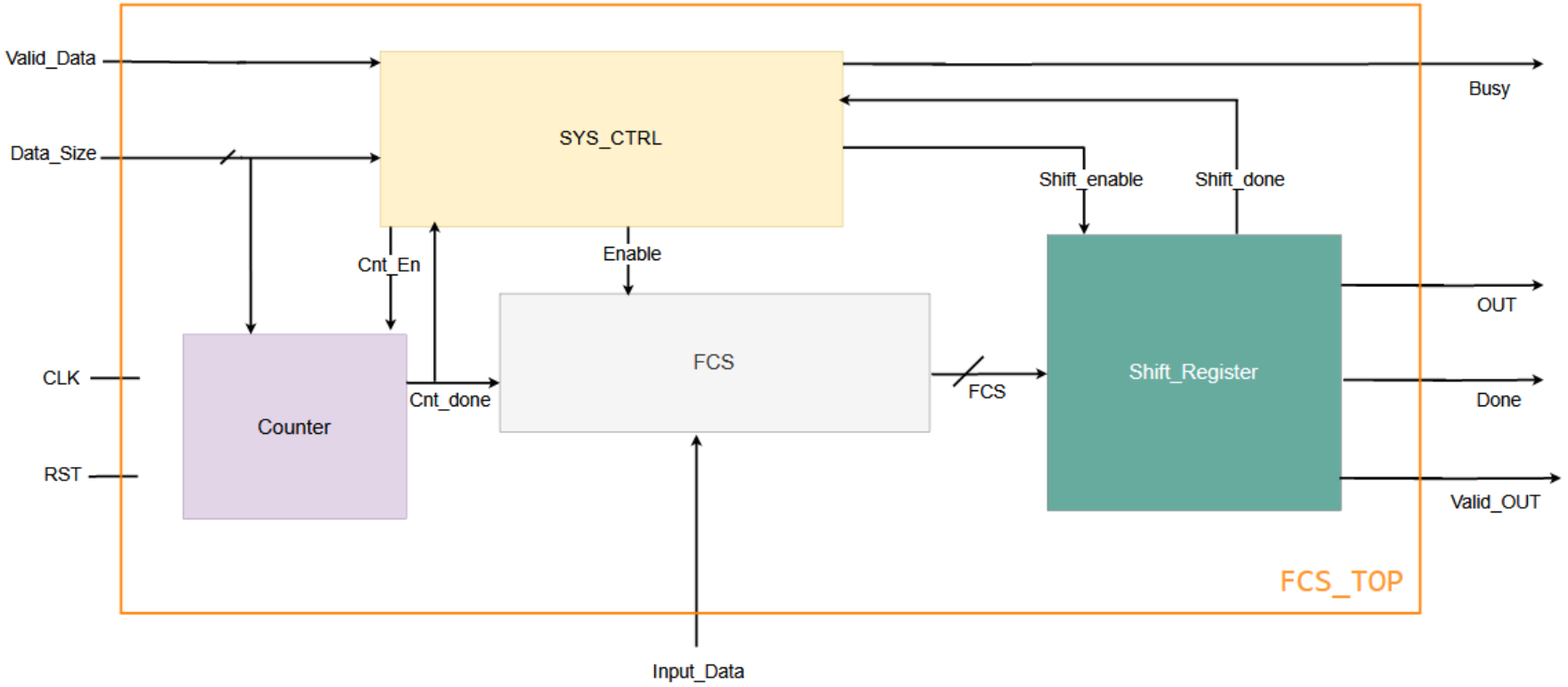

Figure 3 - Block Diagram

System Controller

System Controller has three states of operation:

- Idle State
- FCS operation State
- Shift operation State

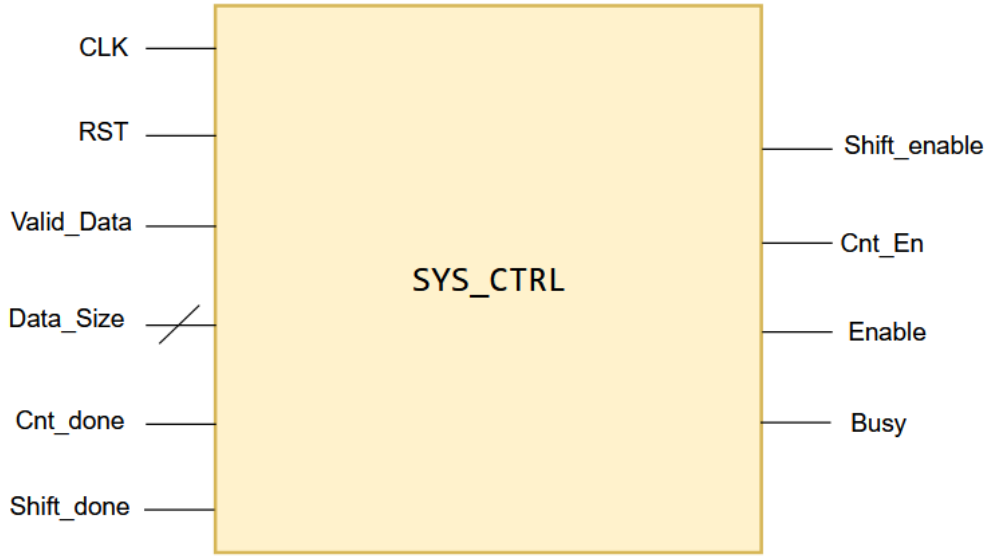

Figure 4 – System Controller Block

System Controller

Finite state Machine for the system controller operation:

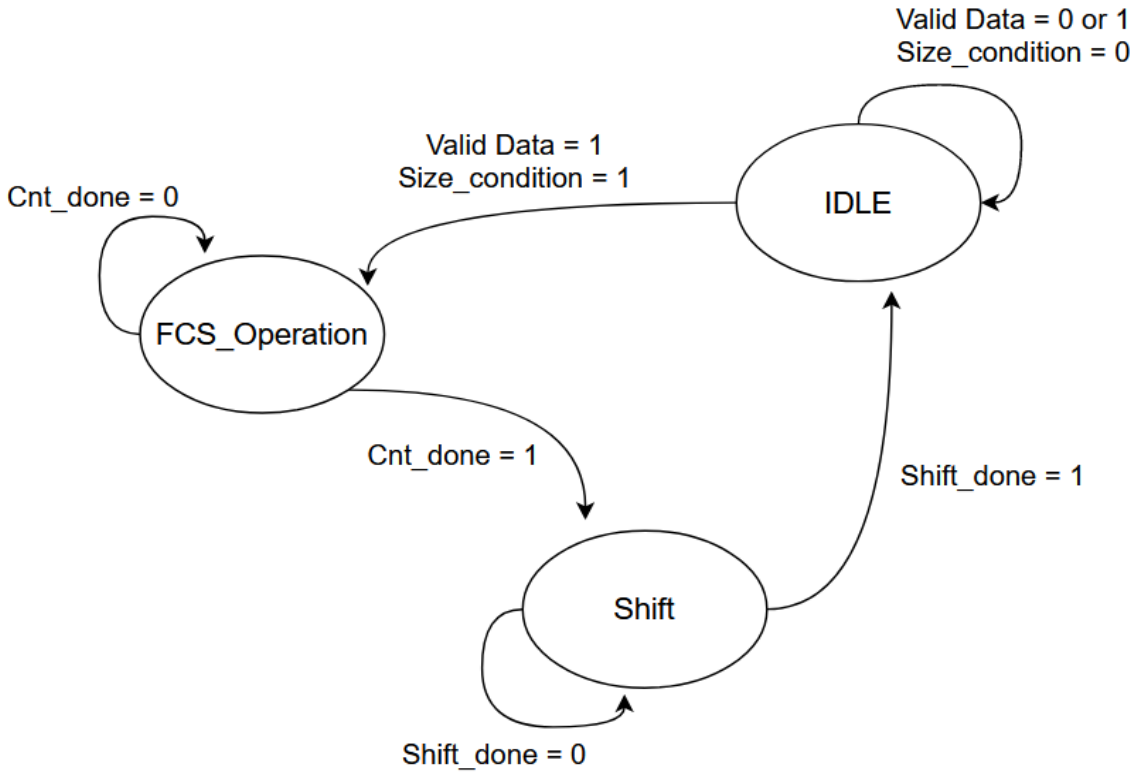

Figure 5 – Finite State Machine for System Controller

System Controller

Outputs in each state:

States	Busy	Enable	Cnt_Enable	Shift_Enable
IDLE	0	0	0	0
FCS_Operation	1	1	1	0
Shift	1	0	0	1
Default	0	0	0	0

System Controller: Ports list

Port	Direction	Width	Description
CLK	Input	1	System Clk
RST	Input	1	Active low asynchronous reset signal
Valid_Data	Input	1	Valid Signal
Data_Size	Input	10	Size of input data
Cnt_done	Input	1	The FCS Counter is done
Shift_done	Input	1	Shift operation Done
Shift_enable	Output	1	Enable for Shift operation
Cnt_En	Output	1	FCS Counter Enable
Enable	Output	1	Enable for FCS operation
Busy	Output	1	Busy signal

Counter

The Counter starts count when the system control send enable signal to the counter (Cnt_En = 1) and that happened when Valid Data = 1 and frame width is less than 1024 bits and greater than 64 bits (Size condition) -> “FCS operation state”

The counter counts until Counter value = (Data_width-1) then Counter_done signal is high.

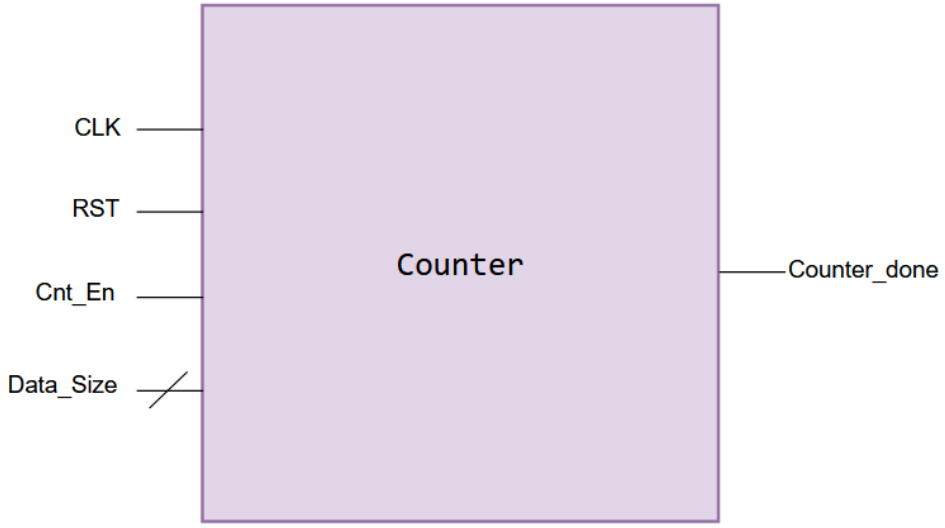

Figure 6 – Counter Block

```
assign Cnt_done = ( Counter == Data_Size-1 ) ? 1'b1 : 1'b0 ;
```

Counter: Ports list

Port	Direction	Width	Description
CLK	Input	1	System Clk
RST	Input	1	Active low asynchronous reset signal
Data_Size	Input	10	Size of input data
Cnt_En	Input	1	FCS Counter Enable
Cnt_done	Output	1	The FCS Counter is done

Frame Check Sequence

After the valid signal is high and size condition of data is true, the system controller send “Enable” signal to FSC to start the operation.

Stand_Gen is the standard generator polynomial of degree 16

$$G_{16}(x) = X^{16} + X^{12} + X^5 + 1$$

```
Stand_Gen     = 17'b10001000000100001;
```

After the counter is done (Cnt_done = 1), the FCS result is ready to be shifted.

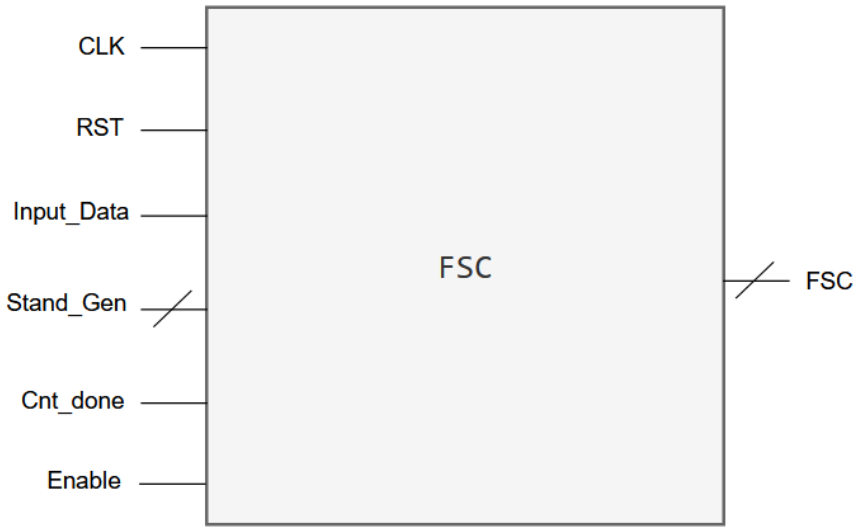

Figure 7 – FCS Block

FCS: Ports list

Port	Direction	Width	Description
CLK	Input	1	System Clk
RST	Input	1	Active low asynchronous reset signal
Input_Data	Input	1	Serial Input Data Field
Stand_Gen	Input	17	Standard Generator bits
Cnt_done	Input	1	The FCS Counter is done
Enable	Input	1	Enable for FCS operation
FSC	Output	16	FCS Result

Shift Register

After the FSC counter is done, the system controller sends “Shift_enable” to Shift Register to start the shift operation.

The Shift register Counter starts to count from 1 until counter value equal (reminder width) then Shift_done signal is high and the last bit of FCS result is transmitted through OUT port.

```
assign  Shift_done_value    = ( Counter  == Rem_WIDTH ) ? 1'b1 : 1'b0 ;
```

Note: Reminder width = Stand_Gen_width – 1 = 16

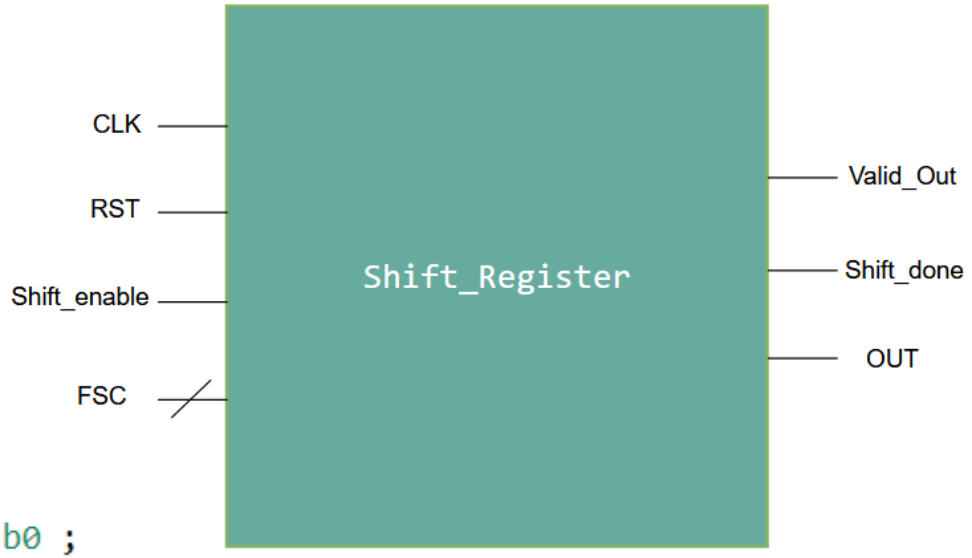

Figure 8 – Shift Register Block

Shift Register: Ports list

Port	Direction	Width	Description
CLK	Input	1	System Clk
RST	Input	1	Active low asynchronous reset signal
FSC	Input	16	FCS Result
Shift_enable	Input	1	Enable for Shift operation
OUT	Output	1	Output Serial Data
Valid_OUT	Output	1	Valid signal for output Data
Shift_done	Output	1	Shift operation Done

System Specifications

- The module is working with 32MHz system clock
- The input length may vary between 64:1024 bits.

Test Bench

Test bench interface with the top module:

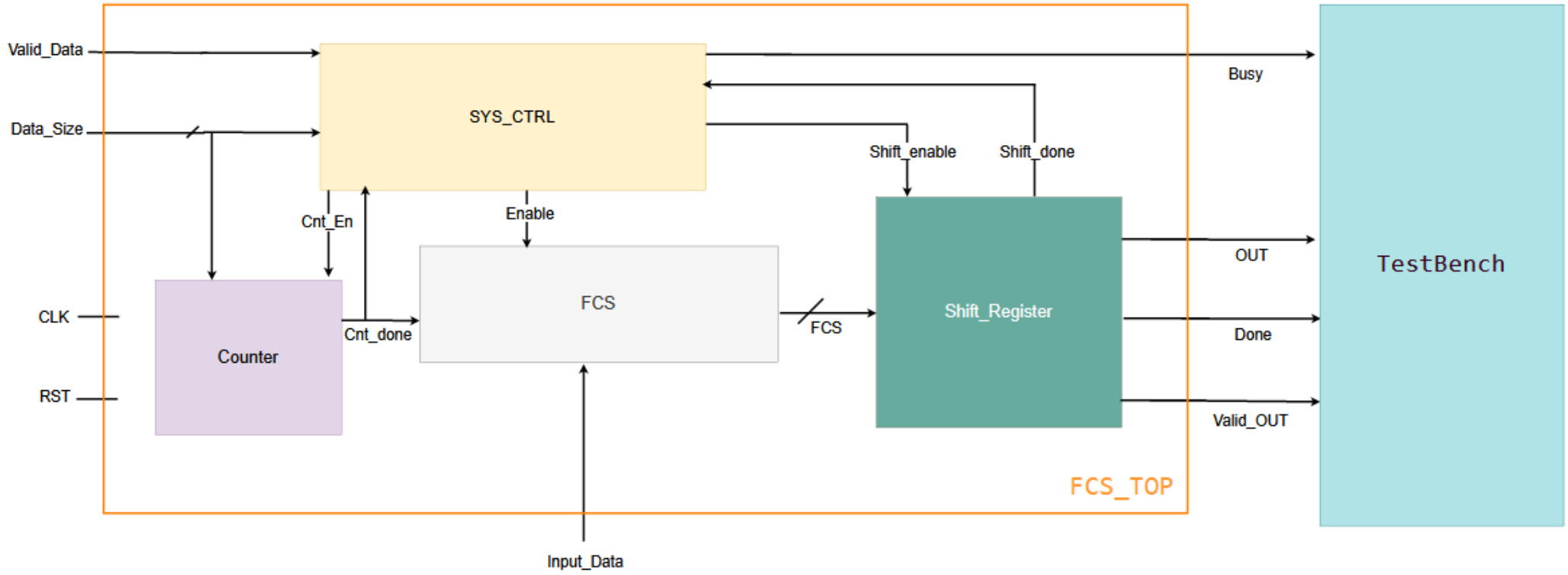

Figure 9 – TestBench

Test Bench

The test bench consists of four tasks:

- Initialize task:

This task responsible for initialize the input ports of top module.

- Reset task:

This task responsible for reset the active low asynchronous RST signal.

- Operation task:

This task responsible for take the data from input port (MSB first) to make FCS operation

- Test task:

This task responsible for test the OUT data.

Simulations Results

The test case and the expected output:

As an example, consider an acknowledgment frame with no payload and the following 3 byte MHR:

0100 0000 0000 0000 0101 0110 [leftmost bit (b_0) transmitted first in time]

b_0.....b_{23}

The FCS for this case would be the following:

0010 0111 1001 1110 [leftmost bit (r_0) transmitted first in time]

r_0.....r_{15}

```
DATA          = 'b00000000_00000000_00000000_00000000_00000000_01000000_00000000_01010110;
Expected_OUT  = 'b0010011110011110;
Data_Size_tb  = 'd64;
```

Figure 10 – Test Case

Simulations Results

Simulation Results for the pervious test case:

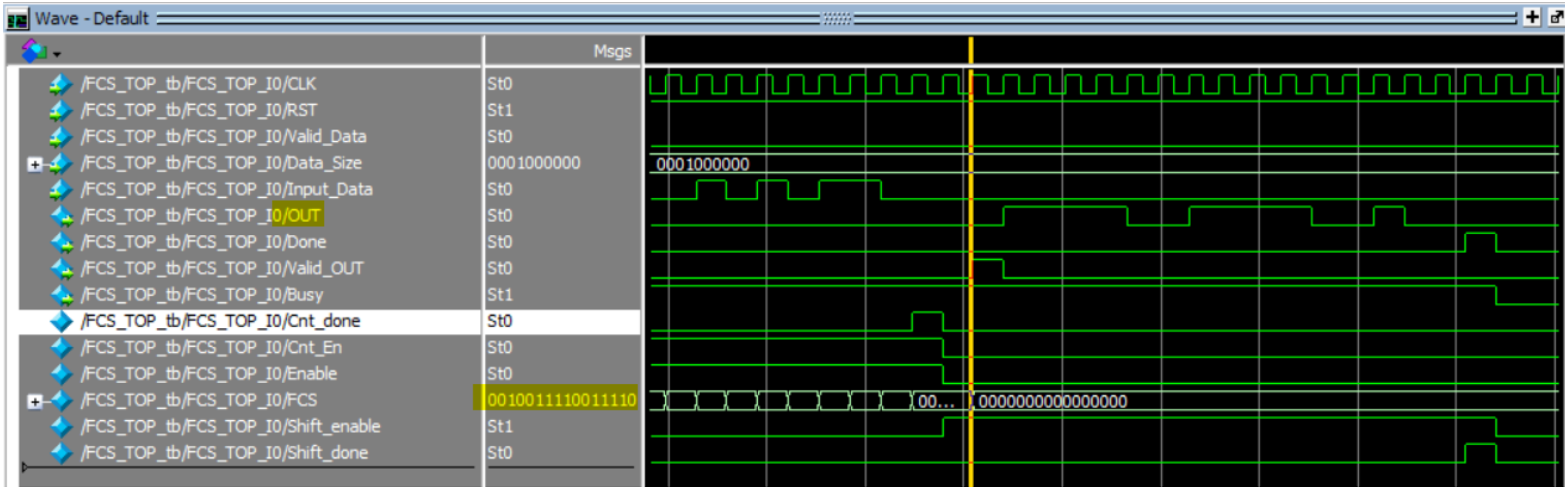

Figure 11 – Simulation results

Simulations Results

Simulation Results for the pervious test case:

```
VSIM 28> run -all
# OUT [0] IS Correct
# OUT [1] IS Correct
# OUT [2] IS Correct
# OUT [3] IS Correct
# OUT [4] IS Correct
# OUT [5] IS Correct
# OUT [6] IS Correct
# OUT [7] IS Correct
# OUT [8] IS Correct
# OUT [9] IS Correct
# OUT [10] IS Correct
# OUT [11] IS Correct
# OUT [12] IS Correct
# OUT [13] IS Correct
# OUT [14] IS Correct
# OUT [15] IS Correct
```

Figure 12 – Test OUT bits