Digital Design Automation – EN0720 (KD7020)
Assignment 2017-18

Design and Verification of an FPGA-based Bit Error Rate Tester

Objectives:

- To create SystemVerilog (SV) design source descriptions for the various component parts of a Bit Error Rate Tester system for use in the testing of a digital communication system.
- To perform simulations of individual design modules using SV test-modules and the Vivado® Simulator.
- To perform simulations of the complete BERT design top-level module using a SV test-module and the Vivado® Simulator.
- Synthesise and Implement the BERT system, targeting a Field Programmable Gate Array development board (Artix-7 FPGA on Digilent Basys3 development board).
- To perform post-implementation timing simulations of an individual module in order to verify correct operation after implementation.
- Demonstrate the operation of the FPGA implementation of the BERT using the development board by means of a so-called 'loop-back' test, and optionally, using a simple visible light communication link.

Learning Outcomes:

See Assignment Specification document:
'EN0720_KD7020_assignment_2017_18_Specification.docx'.

Software and Hardware Resources:

- Xilinx Vivado ® FPGA/CPLD Design Suite. Available in labs E204/206 and as a free download (Vivado HLS WebPACK Edition) from: http://www.xilinx.com/products/design-tools/vivado/vivado-webpack.html

- Digilent Incorporated 'Basys3' Artix-7 FPGA Development Board. Reference Manual.

Supporting Documentation (available on eLearning portal)

- Powerpoint presentation on Register Transfer Level Design using the SystemVerilog hardware description language.

Total number of marks available: 310

Description of the BERT System

Figure 1a, below, shows a top-level block diagram of the Bit Error Rate Tester System and figure 1b shows the corresponding physical layout of the Basys3 development board. The system outputs a continuous stream of Manchester Encoded data bits on the 'man_tx' output pin, this signal is transmitted via the Visible Light Communication channel shown on the diagram. The received signal from the VLC is fed into the 'man_rx' input of the receiver part of the system, where it is decoded to produce the original transmitted data bits (RNRZ). By a process of synchronisation and comparison, the BERT system determines the number of bits that are received without error, for consecutive bursts of 1000 data bits, and displays this information on the 4-digit, 7-segment display.
Synchronisation between the transmitted and received Non-Return-to-Zero (NRZ) data bits is achieved by manually adjusting the delay applied to the transmitted NRZ data bits, via the 'Delay' input switches, until a LED (SyncLED) indicates that synchronisation is achieved.

The system is driven by a 100MHz crystal clock and an active-high reset push button, when pressed, resets the entire system.
The highest data bit-rate of the system is one tenth of the master clock frequency, i.e. 10MBits/second. It may be necessary to reduce this data rate in order to perform a test with a simple VLC link having a lower bandwidth.

This assignment involves the design, verification and implementation of the BERT system using a Field Programmable Gate Array (FPGA) device, this is achieved through a series of guided tasks. In addition, the characteristics of the VLC channel will be modelled and investigated by means of mixed-signal simulation.

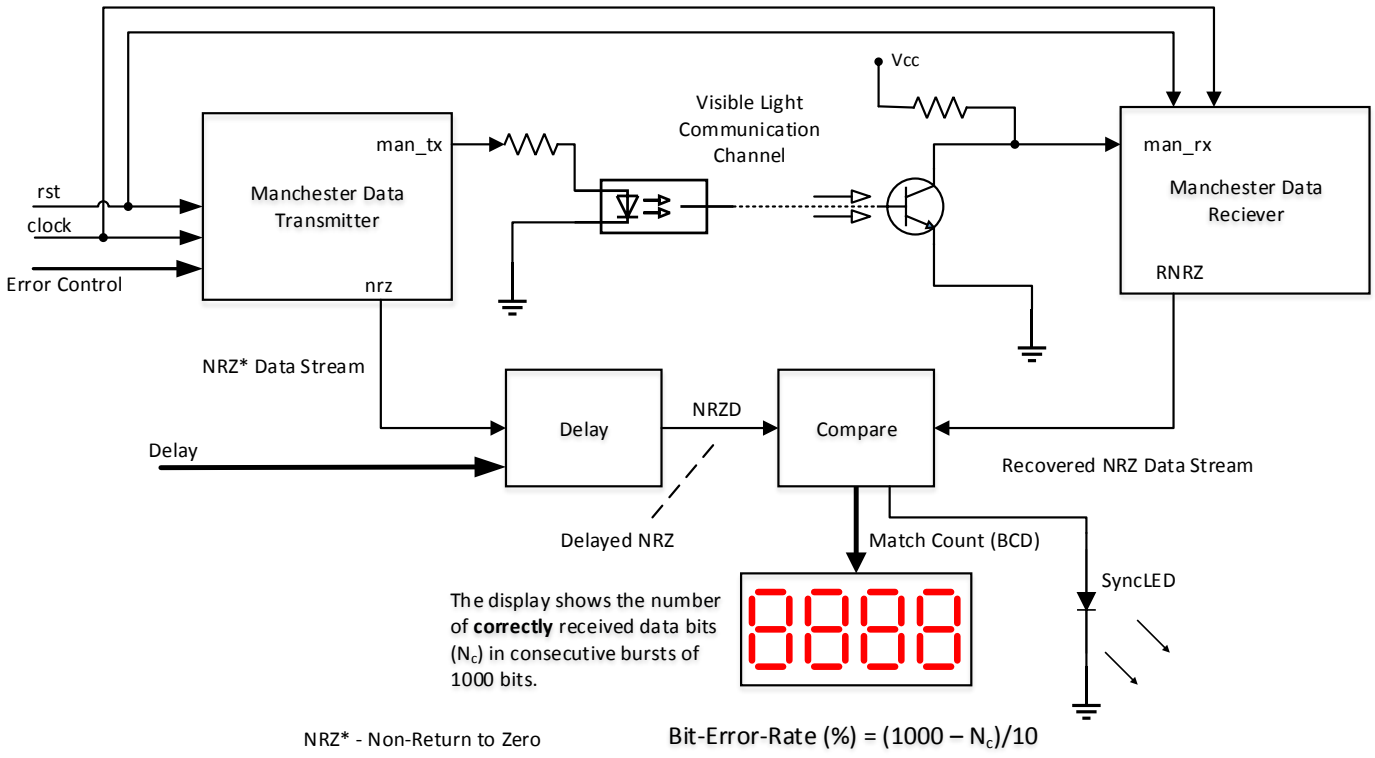


Figure 1a – Top-level block diagram of BERT system

The physical layout of the BERT system implemented using the Digilent® 'Basys3' development board, shown in figure 1b, illustrates the location of the main inputs and outputs. Eight sliding switches (SW0…SW7) are used to set the delay value when synchronising the transmitted and received data bits, while switches SW12 to SW15 are used to control error insertion.

A wire link is shown connecting ‘man_tx’ to ‘man_rx’ at the top right hand 10-way connector JB1, this constitutes the so-called ‘loop-back’ configuration.

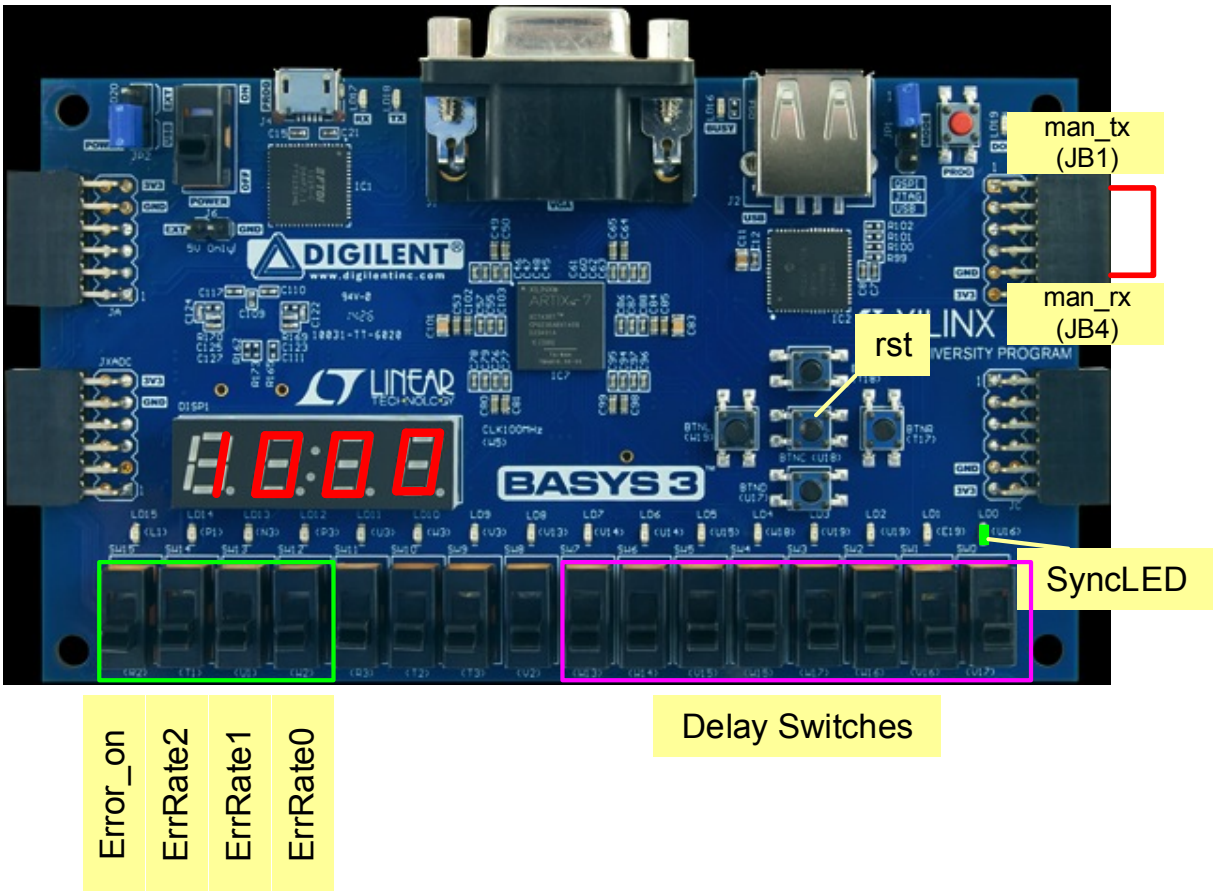


Figure 1b – Physical layout of BERT system

Figure 1c, below, shows the internal block diagram of the Bit Error Rate Tester System.

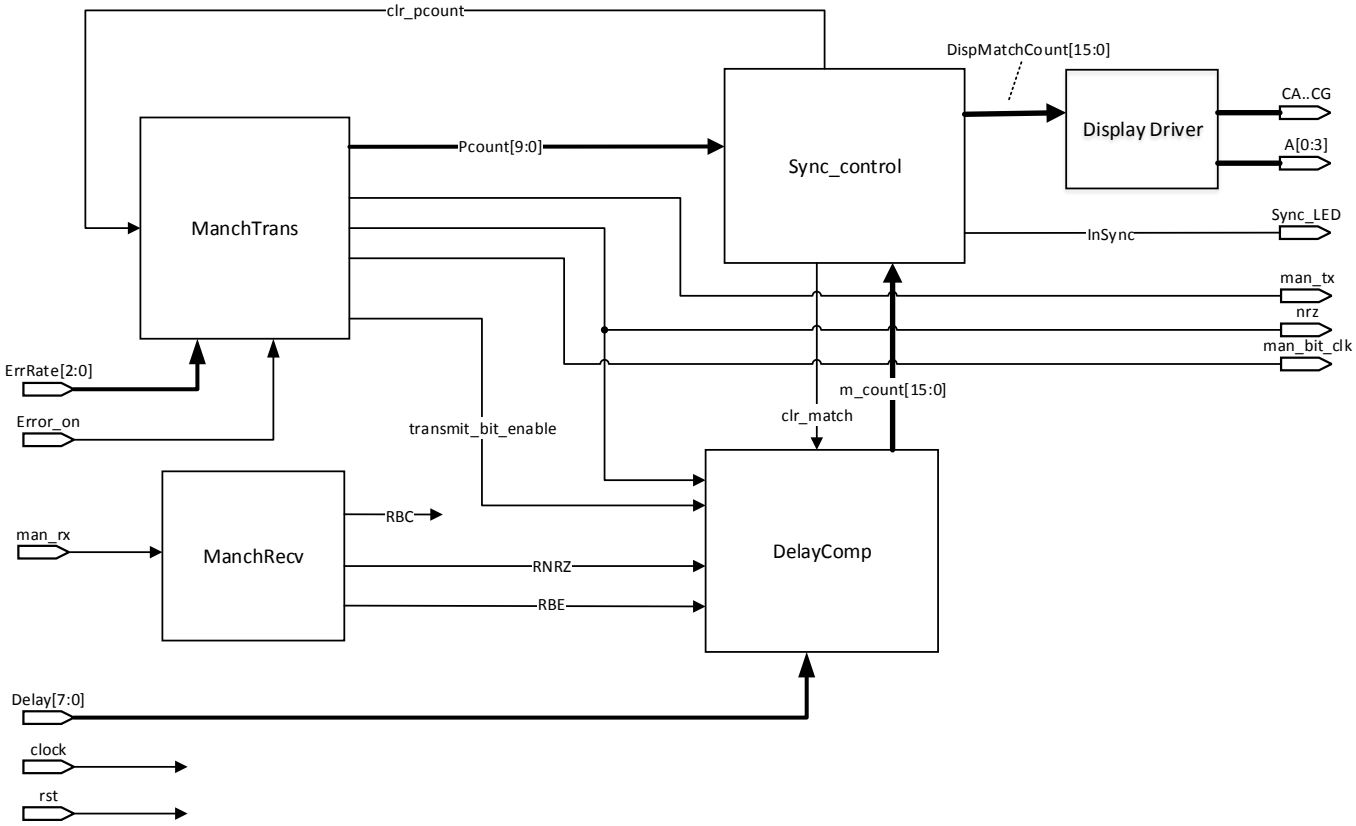


Figure 1c – Internal block diagram of the BERT System

The system is made up of four main blocks, the majority of which are clocked by the system clock input ('clock') and asynchronously reset by means of the 'rst' input push-button. A Display Driver logic circuit outputs the 'Match Count' value to the multiplexed 4-digit, 7-segment display. All of the blocks in Figure 1c are SystemVerilog modules and, with the exception of the 'Sync_control' block, contain instances of sub-modules. Wherever possible, the names of ports and interconnecting signals should be kept the same in order to exploit the automatic port connection feature of the SV language.
The system provides additional outputs 'nrz' and 'man_bit_clk' for testing purposes only, the Manchester encoded data bits being transmitted are on output 'man_tx'.
The Manchester Transmitter block contains a sub-module that allows errors to be inserted into the transmitted data stream corresponding to a range of bit-error rates from 1% to 50%, this allows the BERT system to perform self-testing when performing a so-called 'loop-back' test, i.e. 'man_tx' linked directly to 'man_rx'.

Task 1 – Create a SV source descriptions for the 'ManchTrans' block

The timing diagram of figure 2a below shows the behaviour of the signals associated with the Manchester Encoding block (assuming the error insertion logic is disabled). The NRZ data stream is to be a Pseudo-Random Bit Sequence (PRBS) containing (2^N – 1) bits, where 'N' is a **parameter** that sets the length of the PRBS shift register (set to 10 for implementation). The sequence continually repeats without interruption (apart from when the global system reset (rst) is asserted).
The 100MHz main system clock is to be divided internally by 10 to result in a data bit rate of 10MBits/second. This achieved by means of an internal signal, 'bit_en' (bit enable), that pulses high for one clock period every 10, as shown by the red waveform in figure 2a.

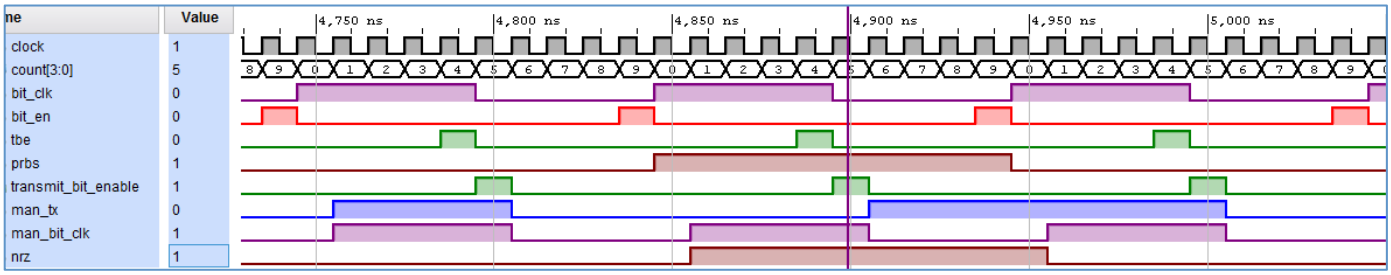


Figure 2a – Timing Waveforms for Manchester Encoding

Figure 2b shows the internal block diagram of the 'ManchTrans' block.

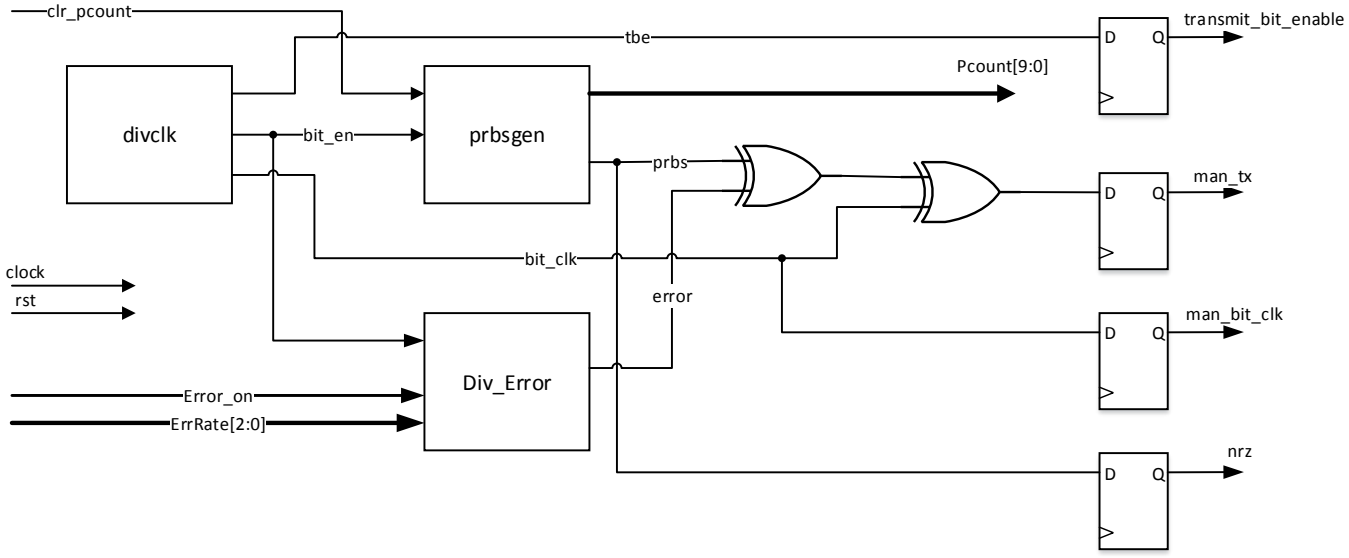


Figure 2b - Internal structure of the 'ManchTrans' block

As shown in the above diagram, a sub-module named ‘divclk’ generates the three basic timing signals (‘bit_en’, ‘bit_clk’ and ‘tbe’ in figure 2a) from a 4-bit, divide-by-10 counter signal (‘count’). Given that the signal ‘count’ goes from 0 to 9 repetitively:

‘bit_en’ is logic-1 when count equals 9
‘bit_clk’ is logic-1 when count is in the range 0 to 4 inclusive
‘tbe’ is logic-1 when count equals 4

Create a SV module for the clock divider ‘divclk’ having the module header provided below.

```
module divclk #(parameter DIVIDER = 10, N = 4)
   (
    input logic clock,
    input logic rst,   //asynchronous reset
    output logic bit_en,
    output logic bit_clk,
    output logic tbe
    );
```

The parameters ‘DIVIDER’ and ‘N’ allow the division ratio to be changed if necessary, ‘N’ being the length of the register required to divide by ‘DIVIDER’.

Create a Vivado® RTL project setup to target the FPGA device on the Basys3 development board (xc7a35tcpg236-1). Add the ‘divclk’ design source file you have created in task 1 to the project, making sure it is associated with both implementation and simulation. Verify correct operation of the ‘divclk’ module by performing an interactive behavioural simulation using the Vivado Simulator, you will need to set the ‘divclk’ module as the top-level simulation source if there are other sources in the simulation group. Add the relevant signals to the wave window and use the ‘Force’ commands to stimulate the inputs (Right click over signal name in waveform window and: ‘Force – Clock’/’Force - Constant’).
Capture the waveform results in order to clearly show correct operation of the module and insert them, along with the source listing, into your report. Comment on the results obtained.

[10 marks]

The output ‘bit_en’ from the ‘divclk’ module enables the pseudo-random bit sequence generator (prbsgen) to shift out the next random data bit on signal ‘prbs’ every 10 clock pulses, whilst also incrementing ‘Pcount’ by 1. The signal ‘Pcount’ keeps track of the number of bits output during each burst of pseudo-random bits. This count value can be reset to zero at any time by asserting the input ‘clr_pcount’, this occurs synchronously at the next positive-edge of the clock. Note that the PRBS data bit sequence is not affected by clearing ‘Pcount’.
The following image in figure 2c shows the ‘prbs’ and ‘Pcount’ output waveforms of the ‘prbsgen’ module (bottom 2). Both outputs ‘prbs’ and ‘Pcount’ are triggered by input ‘bit_en’ and the corresponding system clock edge.

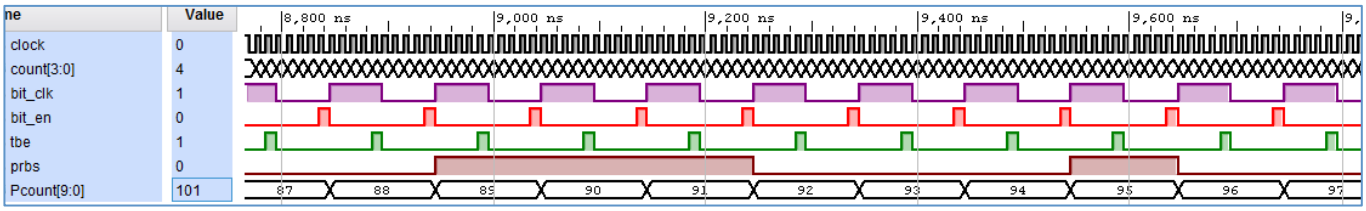


Figure 2c – Timing waveforms for outputs of ‘prbsgen’

The following partial listing is an outline of the ‘prbsgen’ SV description. Parameter ‘N’ sets the length of both the pseudo-random bit sequence register (prbsreg) and the data bit counter ‘Pcount’. Parameter ‘TapMask’ selects the bits of the pseudo-random bit sequence register that are Exclusive-Or’ed and fed back to the input of the least significant flip-flop (feedback taps).

```
module prbsgen #(parameter N = 8, TapMask = 8'b11100001)
   (input logic clock, rst, bit_en, clr_pcount,
    output logic prbs,
    output logic [N-1:0] Pcount);

logic [N-1:0] prbsreg;  //pseudo-random bit sequence register

always_ff @(posedge clock, posedge rst)
begin
   if (rst) begin
       //insert asynchronous reset action
   end else if (clr_pcount) begin
      //insert synchronous reset action
   end else if (bit_en) begin
      //insert synchronous clocked action
   end
end

assign prbs = ……;   //assign MSB of prbsreg to output

endmodule
```

Complete the above SV description of the 'prbsgen' module and verify its correct operation by performing a behavioural simulation using the Vivado Simulator (after adding it to the project).
It may be beneficial to combine the 'prbsgen' and 'divclk' modules into a test-module, given that 'bit_en' is generated by the latter. Add the relevant signals to the wave window and use the 'Force' commands (or the test-module) to stimulate the inputs.
Experiment with different values of 'prbsgen' parameters 'N' and 'TapMask', suitable feedback tap values may be found by searching the internet.
Capture the waveform results in order to clearly show correct operation of the module and insert them, along with the source listing, into your report. Comment on the results obtained.

[10 marks]

As shown in figure 2b, a pair of Exclusive-OR gates along with a D-type Flip-flop are used to generate the Manchester encoded bit-stream 'man_tx'. Additional flip-flops are included to synchronise the remaining outputs to the system clock. The signal labelled 'error' in figure 2b is Exor'd with the 'prbs' output of the 'prbsgen' module before being further Exor'd with the 'bit_clk' signal from the clock divider module. Explain the function of this part of the Manchester Encoder sub-system (the two Exor gates and the D-type flip-flop they feed).

[10 marks]

The module named 'Div_Error' in figure 2b operates as follows:

i. The input named 'Error_on' is used to turn on the error pulses, if it is logic-0 the 'error' output is at logic-0. If 'Error_on' is at logic-1 the error pulses are enabled.
ii. The 3-bit input named 'ErrRate' controls the interval (in terms of NRZ data bits) between 'error' bits (which invert one bit of the NRZ data stream) in order to simulate a range of bit-error rates, as per the following table:

ErrRate	Bits between errors*	Bit Error Rate (%)
0	99	1%
1	49	2%
2	19	5%
3	9	10%
4	4	20%
5	3	25%
6	2	33⅓%
7	1	50%

*1 less than bits per error

iii. An internal counter is incremented each time input ‘bit_en’ is asserted and resets to zero after reaching the required ‘bits between errors’ value.
iv. All sequential elements are reset asynchronously by ‘rst’ and clocked on the positive-edge of ‘clock’.

Create a SystemVerilog source description for the ‘Div_Error’ block based upon the information provided above. Verify its correct operation by performing a behavioural simulation using the Vivado Simulator, it may be beneficial to combine the ‘Div_Error’ and ‘divclk’ modules into a test-module, given that ‘bit_en’ is generated by the latter.
Capture the waveform results in order to clearly show correct operation of the module(s) and insert them, along with the source listing, into your report. Comment on the results obtained.

[15 marks]

Combine the modules created and verified above together, in order to create a SystemVerilog source description for the ‘ManchTrans’ block, using figure 2b as a guide. Attempt to use consistent signal and port names throughout, so that automatic port connections can be exploited, shown by the following line of SV source text:

```
Div_Error DE1(.*);  //instantiation of 'Div_Error' module with auto-connection
```

The module header for ‘ManchTrans’ is provided below:

```
module ManchTrans #(parameter N = 4, TapMask = 4'b1100) //passed to 'prbsgen'
             (input logic clock, rst, clr_pcount,
             input logic Error_on,
             input logic [2:0] ErrRate,
             output logic transmit_bit_enable, nrz, man_bit_clk, man_tx,
             output logic [N-1:0] Pcount);
```

the above module header includes **parameter**s to set the length and feedback tap positions of the PRBSG, for initial simulation purposes these will be set to the following values:

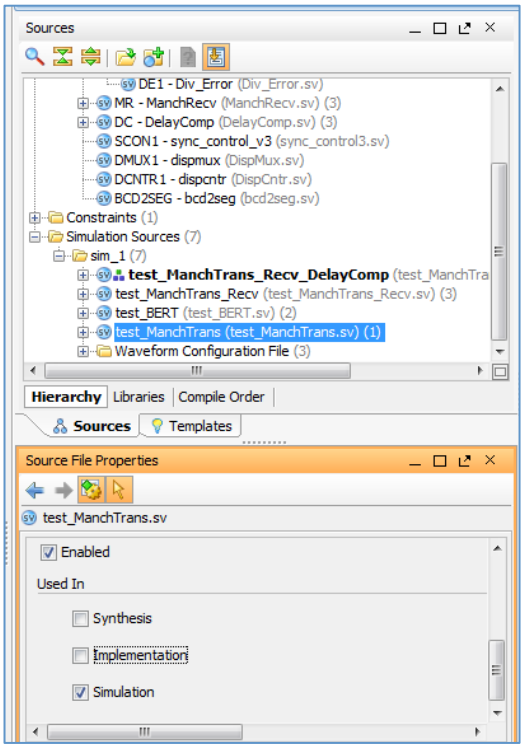
Length, N = 4
Feedback taps, TapMask = 4'b1100

Include a SystemVerilog source listing for the ‘ManchTrans’ module(s) in your report, neatly presented using appropriate use of reserved word highlighting and indentation.

[10 marks]

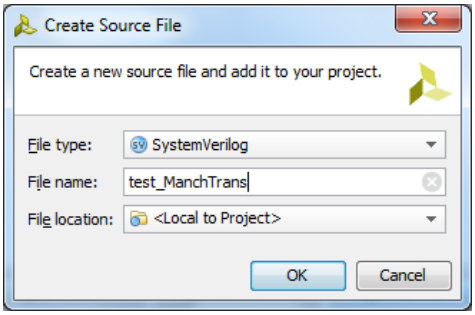
Task 2 – Behavioural Simulation of the ‘ManchTrans’ block

Appendix A contains a listing for a test-module to be used to verify the ‘ManchTrans’ module. Create a new simulation source file named ‘Test_ManchTrans.sv’ using the procedure described below and copy the text from Appendix A into the file, reformatting it as required. Add the source file as a ‘simulation source’ to the Vivado® project created earlier. Ensure all of the SystemVerilog design source files you have created in task 1 are also added to the project as design files. The following screenshot shows how the properties of any source file can be viewed and changed if required:



Click on the ‘Add Sources’ tool button in the Project Manager group and then select ‘Add or Create Simulation Sources’ from the ‘Add Sources’ window.

Click on ‘Create File’ and enter/select the fields as shown below:



Click ‘OK’, then ‘Finish’ then ‘OK’ again to skip the ‘Define Module’ step. An almost empty SV test-module source file will be added to the ‘Simulation Sources’ group. Open this file and copy/paste the test-module source from the Appendix into it. It may be necessary to edit the text to tidy up the formatting.

Run a behavioural simulation of the design and capture the waveform results, including all relevant signals (it may be necessary to set the ‘test_ManchTrans’ module as the top-level simulation module, if other test-modules are already present in the project)

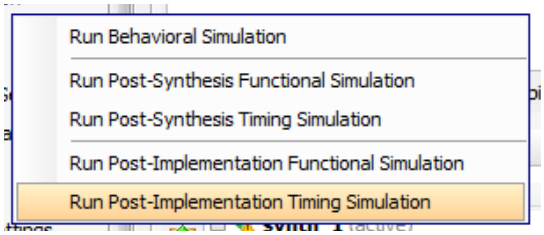
The initial simulation sets the error control inputs, ‘Error_on’ and ‘ErrRate’ to zero. Try changing these to: ‘Error_on = 1’b1’ and ‘ErrRate = 5’. Rerun the simulation (Run – Relaunch Simulation) and record/explain the simulation results (refer to the Bit-Error Rate table provided in Task 1).

Copy and paste clear images of the simulation results, showing a variety of views (full and zoomed-in) to clearly confirm the operation of the ‘ManchTrans’ module. Write down brief comments to explain the waveforms presented.

[10 marks]

Task 3 – Synthesis and Implementation of the 'ManchTrans' block.

Briefly explain the different kinds of simulation that can be performed using the Vivado Simulator, as shown in the following image.



In addition, explain why it is important to perform simulation after implementation of the design, either as individual modules or the complete design.

[10 marks]

You are now going to Synthesise, Implement and run post-implementation simulation on the 'ManchTrans' module. These processes will verify the correctness of the 'ManchTrans' block before proceeding with the rest of the design.

Without assigning any pin numbers to the inputs and outputs (i.e. there are no constraints in the project), run the Synthesis process on the top-level design module 'ManchTrans.sv' (select the design source and 'Set as Top' if is not already the top-level design module).

There may be a number of warnings and these may be ignored, however if any errors occur that prevent successful synthesis, attempt to correct them and rerun the process until no errors or warnings are reported.

Once Synthesis completes successfully, open the synthesised design.

Capture a screen image of the Project Summary window (you may need to detach this from the main Vivado window to see all of the panels), making sure the Utilization panel is clearly visible, with the resource usage values shown in tabular form.
Paste the image into your assignment report to confirm successful synthesis and briefly comment on the information provided in the Utilization panel.

[5 marks]

From within the Flow Navigator panel at the left-hand side of the Vivado window, select 'Synthesis → Synthezised Design → Schematic', to view the top-level schematic diagram of the synthesized logic circuit. Capture an image of the schematic and add it to your assignment report along with brief comments.

[5 marks]

Task 4 – Post-Implementation Timing Simulation of 'ManchTrans' block.

Close the 'Synthesized Design' window by clicking on the 'X' at the top-right-hand corner of the window (not the one in the TRHC of the main Vivado window, as this will close the entire program).

Click on, 'Implementation → Run Implementation' in the Flow Navigator, to invoke the implementation process.

Once implementation has completed successfully it will be possible to run a Post-Implementation Timing Simulation.

Run the simulation and copy/paste clear images of the simulation results into your assignment report, showing a variety of views (full and zoomed-in) with, and without, errors turned on. Include

only the top-level test-module signals. Compare the post-implementation timing simulation results with those obtained in task 2 and write down comments on any differences.

(10 marks)

Task 5 - Create SV source descriptions for the Manchester Receiver (ManchRecv) block

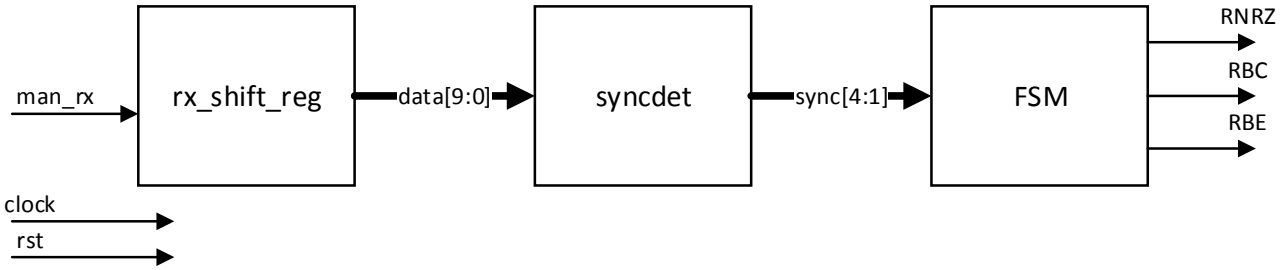
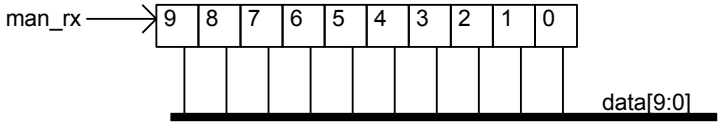


Figure 3a – Internal structure of Manchester Receiver Block

The Manchester receiver block consists of three sub-blocks, as shown in figure 3a above. The digital data stream coming out of the receiver side of the communication system enters a shift register at the 'man_rx' input. The main system 100MHz clock samples and shifts the incoming bits into the shift register most significant bit. The contents of the shift register are shifted to the right on each positive-edge, the resulting parallel 10-bit value is present on the 'data' output port. This process is illustrated by the diagram below:



Create a SV source description for the 'rx_shift_register' sub-block, shown in figure 3a, observing the following guidelines:

i. The length of the shift register (and therefore 'data') is set by a parameter 'N' having a default value of 10 (this must be equal to the 'divclk' module parameter 'DIVIDER')
ii. The shift register is asynchronously cleared to all zeros when 'rst' is taken to logic-1.

[10 marks]

The middle sub-block in figure 3a, 'syncdet', is used to detect specific 10-bit patterns appearing on the output of the shift register, these are listed below along with the output that is activated when the pattern is present:

data	sync[1]	sync[2]	sync[3]	sync[4]
0000011111_2	1	0	0	0
1111100000_2	0	1	0	0
1111111111_2	0	0	1	0
0000000000_2	0	0	0	1

Making use of continuous assignment statements, create a SV source description for the combinational logic sub-block 'syncdet', described by the above table, saving it in a file named 'syncdet.sv'.

[5 marks]

Figure 3b shows an incomplete timing diagram for a set of reciever waveforms. Assuming the reciever shift register, 'rx_shift_register', is initially reset, copy the diagram of Figure 3b and add waveforms for the 4 'sync' pulses, given the waveform for the 'man_rx' input.

The waveforms can be added to the diagram by hand, and the completed figure scanned and inserted as a picture into your assignment document.

[5 marks]

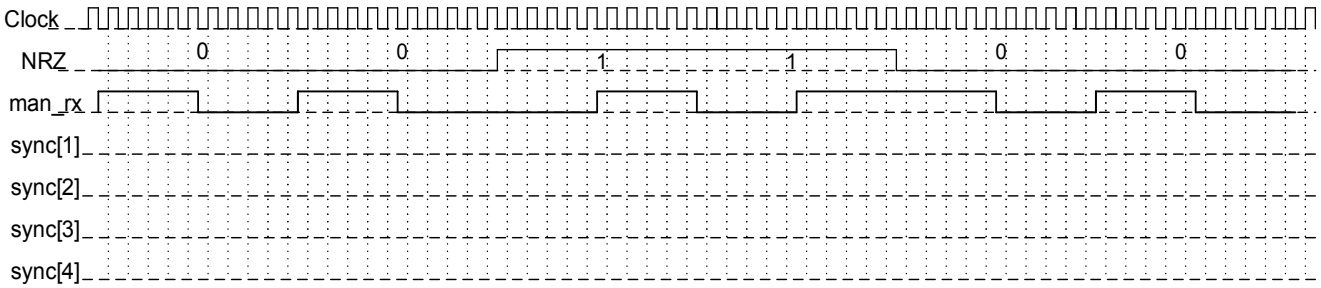


Figure 3b – Incomplete reciever waveforms

Having completed the waveforms above, make notes in your assignment report concerning the significance of the pulses appearing on the 'sync[4:1]' output pulses.

[5 marks]

The remaining part of the Manchester receiver block diagram of figure 3a is a Finite State Machine (FSM), the purpose of which is to detect the sequence of 'sync' pulses coming from the 'syncdet' block and use these to synchronise to the 'man_rx' data stream and extract the following three signals:

a. RNRZ – Recovered Non-Return-to-Zero data bits
b. RBC – Recovered Bit Clock (10MHz, synchronised to RNRZ data)
c. RBE – Recovered Bit Enable, this is a single clock pulse length pulse occurring at the mid-point of each RNRZ data bit.

The ASM diagram shown in figure 3b describes the FSM sub-block of the 'ManchRecv' block. The following guidelines apply to the diagram of figure 3b:

i. The transfers shown in each state are to take place on the positive-edge of the clock signal.
ii. The machine is reset asynchronously by asserting the 'rst' input.
iii. The state values are to be represented by an enumeration type having a base-type of **logic** vector.
iv. The module should include an output named 'RBE', however this can be left unconnected or forced to logic-0 in this version (logic to drive RBE will be added later).

Create a SV source description for the FSM, based upon the ASM diagram of figure 3b, save this module source in a file named 'FSM.sv'. Provide a listing of the source file in your assignment document.

[15 marks]

With reference to figure 3a above, instantiate the three SV modules created above ('rx_shift_reg', 'syncdet' and 'FSM') into a module named 'ManchRecv' and save the SV source file as 'ManchRecv.sv'. Provide a listing of the source file in your assignment document.

Add all of the new SV source files created in this task to the Vivado project (as design files).

[5 marks]

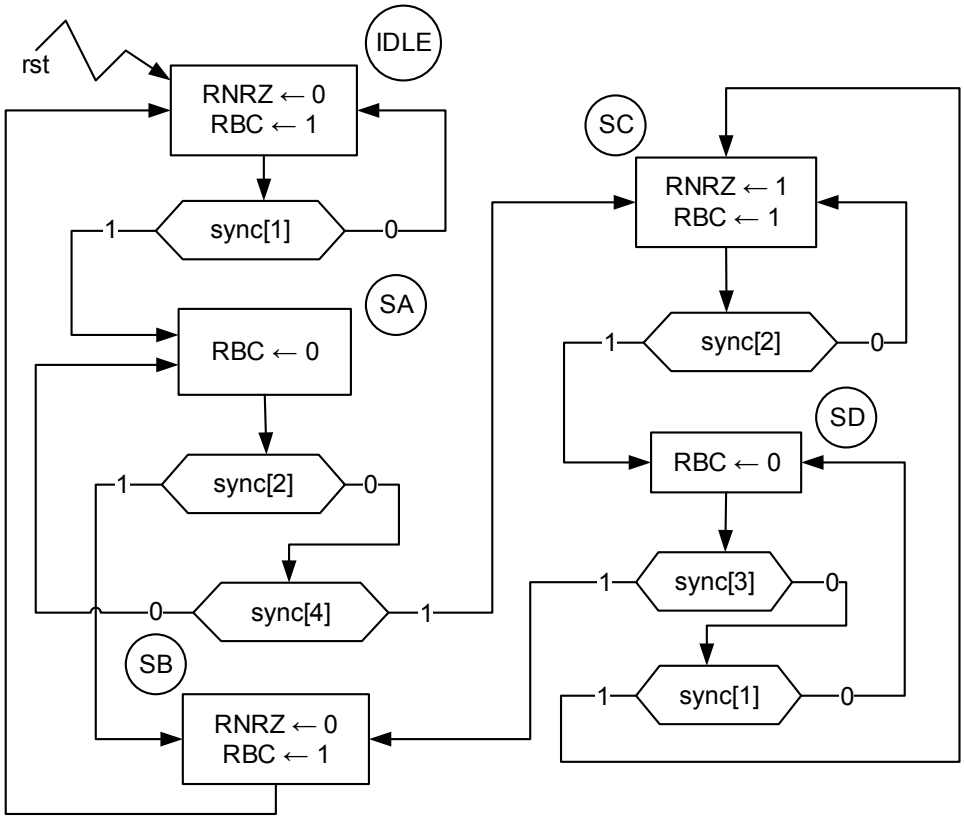


Figure 3b – ManchRecv FSM ASM Diagram

Task 6 – Simulation of the Manchester Transmitter and Reciever modules.

Open the SV source file for the 'test_ManchTrans' module provided in appendix A and save it under the new filename 'test_ManchTrans_Recv.sv'.

Modify the 'test_ManchTrans_Recv.sv' test-module by adding an instance of the Manchester receiver module developed in the previous task along with an instance of the 'TransDelay' module, also provided in the appendix, setting the transmission delay to be 33.0e-9 seconds. The new test-module is to have the structure shown in figure 3c below, make use of signal names that allow automated connection to be used wherever possible. Include a listing of the 'test_ManchTrans_Recv.sv' source file in your report.

[10 marks]

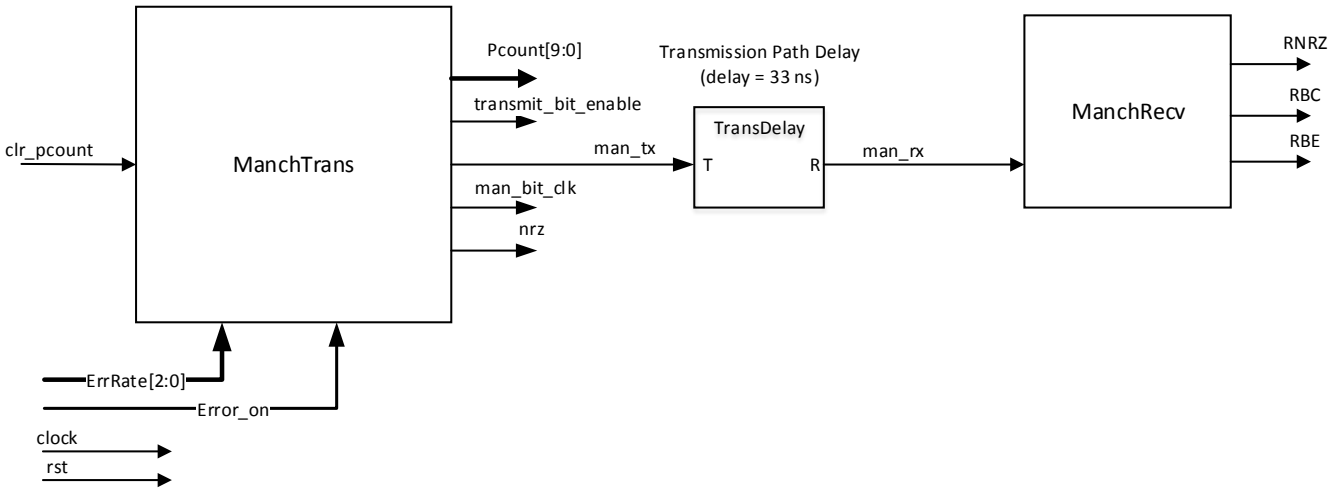


Figure 3c – Test module for Manchester transmitter and reciever

Add the new simulation module ‘test_ManchTrans_Recv.sv’ to the project as a simulation source and set it to be the top-level module in the ‘Simulation Sources’ hierarchy tree in the sources window (Right-click on source, choose ‘Set as Top’). Correct any syntax errors that may be reported at this point.

Launch the Behavioural Simulation of the ‘test_ManchTrans_Recv’ module. Further syntax may be reported at this point causing the simulation to fail, correct these and repeat the process until the simulation starts correctly.

Add the following ‘ManchRecv’ internal signals to the wave window:

i. FSM state
ii. 10-bit output of shift register ‘data’, displayed in HEX format.
iii. 4-bit output of ‘syncdet’, ‘sync’ (expand these to 4 individual traces).

Arrange the waveforms in the waveform window vertically to reflect the logical flow of the signals from transmitter to receiver in figure 3c. Change the colours of the waveform to more clearly highlight the important signals and restart and run the entire simulation (until the $stop command). Capture and record the simulation results and write down brief explanatory comments on their meaning.

Measure the delay between signals ‘man_tx’ and ‘man_rx’, confirm that it is the correct value and capture an image of the measurement.

[15 marks]

Capture images of the simulation results for the following time ranges:

i. 0 ns to 600 ns
ii. 900 ns to 1200 ns

In the second time range above, add markers to measure the delay between the ‘nrz’ transmitter data source and the recovered non-return-to-zero data ‘RNRZ’. Comment on this delay and the factors contributing to it. Try setting the transmission delay to 0.0 seconds and rerun the simulation, in order to measure the inherent delay of the encoding and decoding logic circuits within the transmitter and receiver sub-systems.

[5 marks]

Figure 3d shows the logic circuit required to extract the receive bit enable (RBE) signal from the reciever bit clock (RBC). Note that the SET and CLR flip-flop inputs are not required.
Add SV source statements to the ‘FSM’ module to describe the logic shown in figure 3d, using the SV statements shown below as a guide.

```
logic [4:0] qr;       //5-bit register

always_ff @(posedge clock)
       qr <= ……;    //add missing expression

assign RBE = ……;    //add missing expression
```

Re-invoke (Run – Relaunch Simulation) the simulation in order to verify the correct behaviour of the RBE signal (The RBE pulses should occur just to the left of the mid-point of the RNRZ data bits). Provide a listing of the additional SV statements required to describe the logic of figure 3d and capture the simulation results with appropriate comments.

[5 marks]

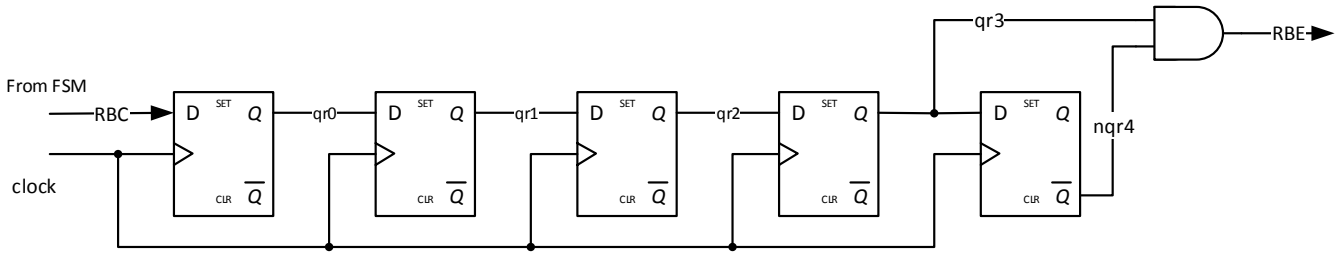


Figure 3d – RBE equivalent circuit (added to the FSM module).

Task 7 –Explanation of the Digital Delay block

The ‘Delay’ and ‘Compare’ blocks shown in figure 1a are implemented as a single block named ‘DelayComp’ in the BERT system of figure 1c.

This block consists of two parts, (i) digital delays to delay the transmitted NRZ data stream and ‘Transmit Bit Enable’ (TBE) signals by a number of clock periods equal to the value applied the 8-bit ‘Delay’ input port (sliding switches SW0-SW7), and (ii) a Binary Coded Decimal 4-digit counter that increments when the delayed version of ‘TBE’ coincides with the ‘Received Bit Enable’ (RBE) and the value of the delayed version of ‘NRZ’ (NRZD) matches the recovered non-return-to-zero data value, ‘RNRZ’, this is known as the ‘match counter’. The combination of these two parts allows the system to count the number of matching bits, when the delay of the channel is equalised with the delay introduced by the ‘dig_delay’ blocks, thereby giving a measure of the Bit Error Rate.

The output of the above counter, ‘m_count’, indicates the number of matches occurring between the received data stream and the transmitted data stream, after the transmitted data has been ‘synchronised’ with the received data. Synchronisation is indicated by a value of ‘m_count’ that exceeds the threshold parameter set up within the ‘SyncControl’ block, when this occurs the ‘SyncLED’ is illuminated.

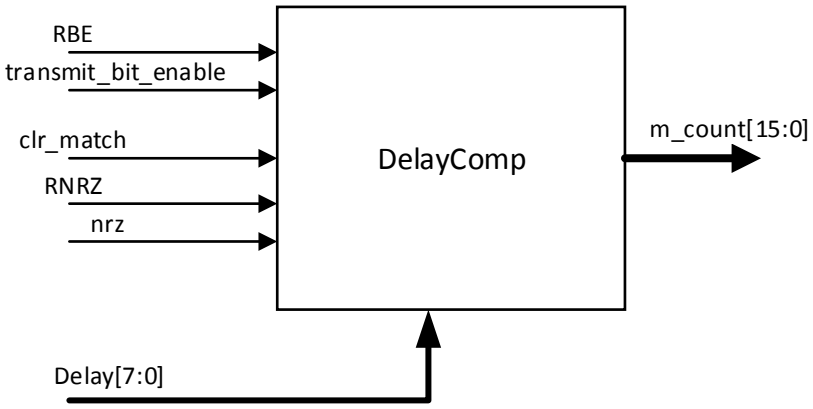


Figure 4a – Block symbol for ‘DelayComp’

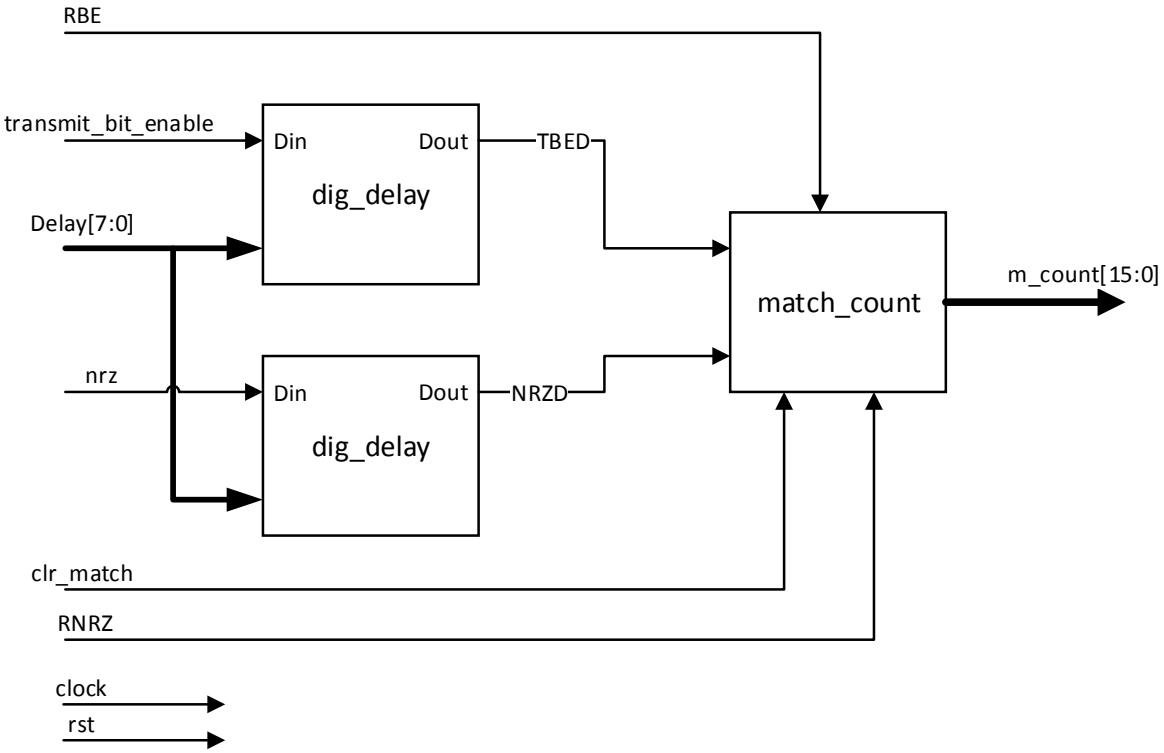


Figure 4b –Internal structure of the 'DelayComp' block

The SystemVerilog source description for the digital delay part (dig_delay) of the 'DelayComp' is provided in Appendix A, this source file is named 'dig_delay.sv'.
Study the source description and sketch an equivalent logic circuit, assuming the parameter 'N' is set to 3. Use the logic circuit to explain how the module implements a delay between the input signal 'Din' and the output 'Dout'. Deduce a general expression for the delay in terms of the clock period, T_{clock}, and the value of 'Delay', in addition to the minimum and maximum possible delays corresponding to a parameter value of D = 8 and the minimum/maximum values of 'Delay'.

[10 marks]

Task 8 – Create the SystemVerilog source description for the match counter and complete the 'DelayComp' block.

The operation of the 'match_count' module was described earlier. A match occurs when the delayed version of 'TBE' ('TBED') coincides with the 'Received Bit Enable' (RBE) and, during the same clock-cycle, the value of the delayed version of 'NRZ', 'NRZD' matches the recovered non-return-to-zero data value, 'RNRZ'.

A match causes the Binary Coded Decimal count to increment by 1. A partial SV source description for the match_count module is given below:

```
module match_count (input logic clock, rst, clr_match,
              input logic TBED, RBE, RNRZ, NRZD,
              output logic [15:0] m_count);  //count output is Binary Coded Decimal

logic [15:0] q;

always_ff @(posedge clock, posedge rst)
begin
       if (rst)
              q <= 0;
       else if (clr_match == 1'b1)
              q <= 0;
       else if (……)  //insert match condition here
       ……   //add statements to increment BCD count here
       ……
end

assign m_count = q;
endmodule : match_count
```

Add SV statement to the above incomplete source description and save the file under the name 'match_count.sv'. The register named 'q' stores a 4-digit BCD number, where each group of 4-bits represents a decimal digit, i.e. <thousands, hundreds, tens, units>.

Add the design file to the Vivado project, correcting any syntax errors that may be flagged in the source. Include a listing of the completed 'match_count' module source in your report.

[10 marks]

Create a new design source file named 'DelayComp.sv'. Using the block diagram of figure 4b as a guide, create a net-list description containing instances of the 'dig_delay' and 'match_count' modules, for the 'DelayComp' module. The module header should include a **parameter** 'D' (having a default value 8) that determines the value of the **parameter** 'N' of both instances of the 'dig_delay' modules shown in figure 4b.

Add the design file to the Vivado project, correcting any syntax errors that may be flagged in the source. Provide a listing of the 'DelayComp' module source in your report.

[5 marks]

Task 9 – Combine the ManchTrans, ManchRecv and DelayComp modules into a test-module and perform behavioural simulations.

The three modules developed above are to be combined in a test-module in order to verify correct operation of the 'DelayComp' module. Figure 5a shows the structure of the test-module source description 'test_ManchTrans_Recv_DelayComp.sv'.

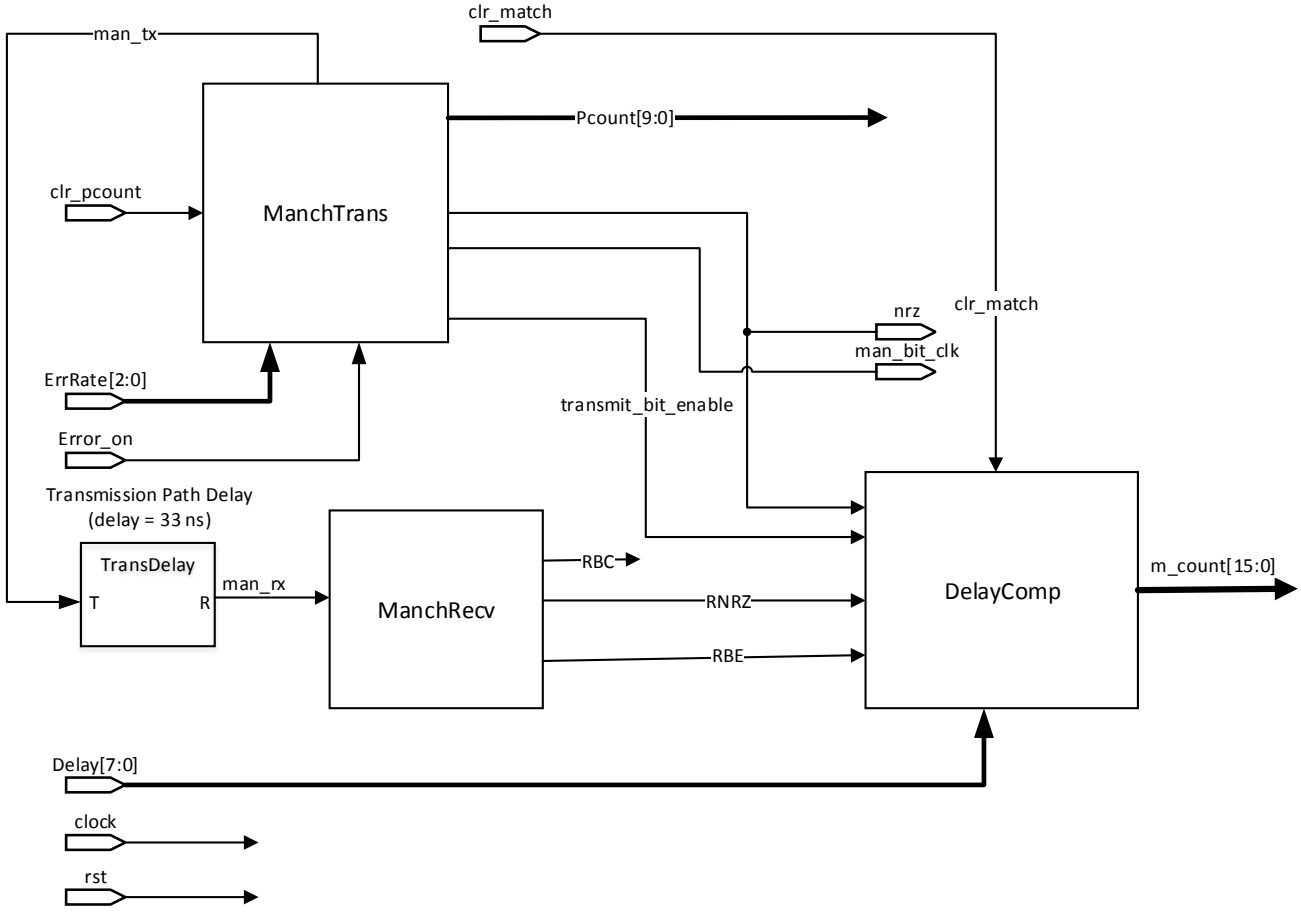


Figure 5a – test-module for Transmitter, Receiver and DelayComp blocks

Open the last test-module you created (test_ManchTrans_Recv.sv) and copy all of the source text into a new file, saving it under the name 'test_ManchTrans_Recv_DelayComp.sv'. Edit this new test-module source so that it describes the test-module shown in figure 5a. This will mainly involve adding an instance of the 'DelayComp' module and associated signals.

At this point the contents of the second **initial** block can be left unchanged apart from adding statements to initialise any new signals to zero. (eg. 'Delay' and 'clr_match') and changing the following statement:

```
repeat (205) @(negedge clock);
```

to:

```
repeat (2600) @(negedge clock);
```

Also, set the transmission delay to '0.0e-9' initially.

Change the width of the local signal ‘Pcount’ to 8-bits and modify the parameters of the ‘ManchTrans’ instance accordingly (i.e. N = 8 and ‘TapMask’ must be set to a bit-pattern that results in a maximal length sequence of 255 bits).

Include a listing of the test-module in your assignment report.

[10 marks]

Add the new test-module to the Vivado project as a simulation source, and set it to be the top-level simulation source.

Select the new test-module and run a behavioural simulation, at this point you may receive error messages if any of the source files contain syntax errors. These must be corrected before re-launching the simulator.

Once the simulation has started successfully, go to the ‘Scopes’ panel and click on the instance of the ‘DelayComp’ module. Select the internal signals ‘NRZD’ and ‘TBED’ and add them to the waveform window.
Select the signal named ‘m_count’ and change the radix to Hexadecimal. Set the radix for ‘Pcount’ and ‘Delay’ to unsigned decimal.

Restart (Run – Restart…) and run (Run – Run All...) to update the waveforms.

Capture images of the simulation waveforms and record them in your assignment report. (Full view and zoomed views as appropriate)

Write comments in your assignment report concerning the results of the simulation, in particular, explain the behaviour of the following waveforms:

NRZD, TBED, RNRZ, RBE and m_count

[5 marks]

Use the waveform measurement Cursor and Markers to determine the value of the ‘Delay’ signal required to synchronise the receiver signals to the transmitter signals. Modify the test-module source accordingly and re-invoke the simulation.

Comment in your report on the effect on the above signals of synchronising the receiver signals to the transmitter signals, write down the value of ‘Delay’ required to achieve this. Compare ‘Pcount’ and ‘m_count’, explain what you observe?

[5 marks]

Open the source text of the test-module ‘test_ManchTrans_Recv_DelayComp’ and set the transmission delay parameter to 374 ns and the 8-bit ‘Delay’ signal to 53_{10}.

Relaunch the simulation and record the waveform results in your report. Explain the effect of changing the transmission delay to 374 ns from 0 ns, in particular comment on the value of ‘m_count’ corresponding to the point at which ‘Pcount’ reaches 255_{10} with the ‘Delay’ signal set to 53_{10}.

[5 marks]

Change the value of the ‘Delay’ signal to a value required to resynchronise the transmitted and received signals (i.e one that results in the ‘m_count’ value incrementing one <u>every</u> NRZ data received).
Relaunch the simulation and record the waveform results in your report.

[5 marks]

Run additional simulations with the value of 'Delay' reduced or increased by a factor of 10, what do you observe in terms of the value of 'm_count' corresponding to the point at which 'Pcount' reaches 255_{10}?

[5 marks]

Task 10 – Addition the 'Sync_control' block and simulation of the complete BERT system.

The synchronisation control block monitors the number of pseudo-random bits that have been transmitted (Pcount) and, when this reaches a value set by the module parameter 'NumPatt' (number of patterns or bits transmitted), transfers the 4-digit BCD value of 'm_count' (match count) to the display driver circuit. This results in the display being updated with the number of correctly received bits after every burst of 'NumPatt' bits transmitted. Another parameter, 'SyncThresh' (synchronisation threshold) is compared with 'm_count', such that if the match count exceeds the synchronisation threshold, the system is synchronised and the 'SyncLED' is illuminated. The values of the 'Sync_control' parameters are summarised below:

P – decimal value that sets the width of the 'Pcount' input port and is the same as ManchTrans parameter 'N'.

SyncThresh – 4-digit hexadecimal value representing a Binary Coded Decimal threshold match count required for synchronisdation.

NumPatt – Number of data bits transmitted in each burst. A decimal value having a range determined by P, normally set to 1000 for implementation.

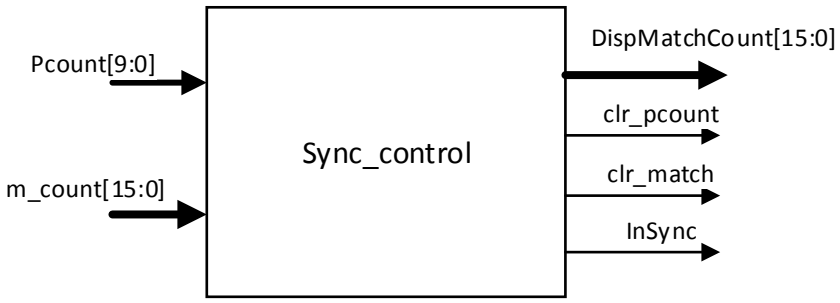


Figure 6a – Symbol for 'Sync_control' block

Figure 6b shows the ASM diagram for the synchronisation control module.

Given the simulation results obtained in Task 9, suggest a suitable value for parameter 'SyncThresh' and explain your choice.

[5 marks]

The module header for the synchronisation control module is provided below:

```
module sync_control_v3 #(parameter P = 10,
                SyncThresh = 16'h0000, //must be BCD format
                NumPatt = 1000)
          (input logic clock, rst,
              input logic [P-1:0] Pcount,
              input logic [15:0] m_count,
              output logic clr_match,
              output logic clr_pcount,  //change to clr_pcount
              output logic InSync,
              output logic [15:0] DispMatchCount);
```

Using figures 6a and 6b as a guide, create a SystemVerilog source description for the synchronisation control module (saved in a file 'sync_control.sv').

[20 marks]

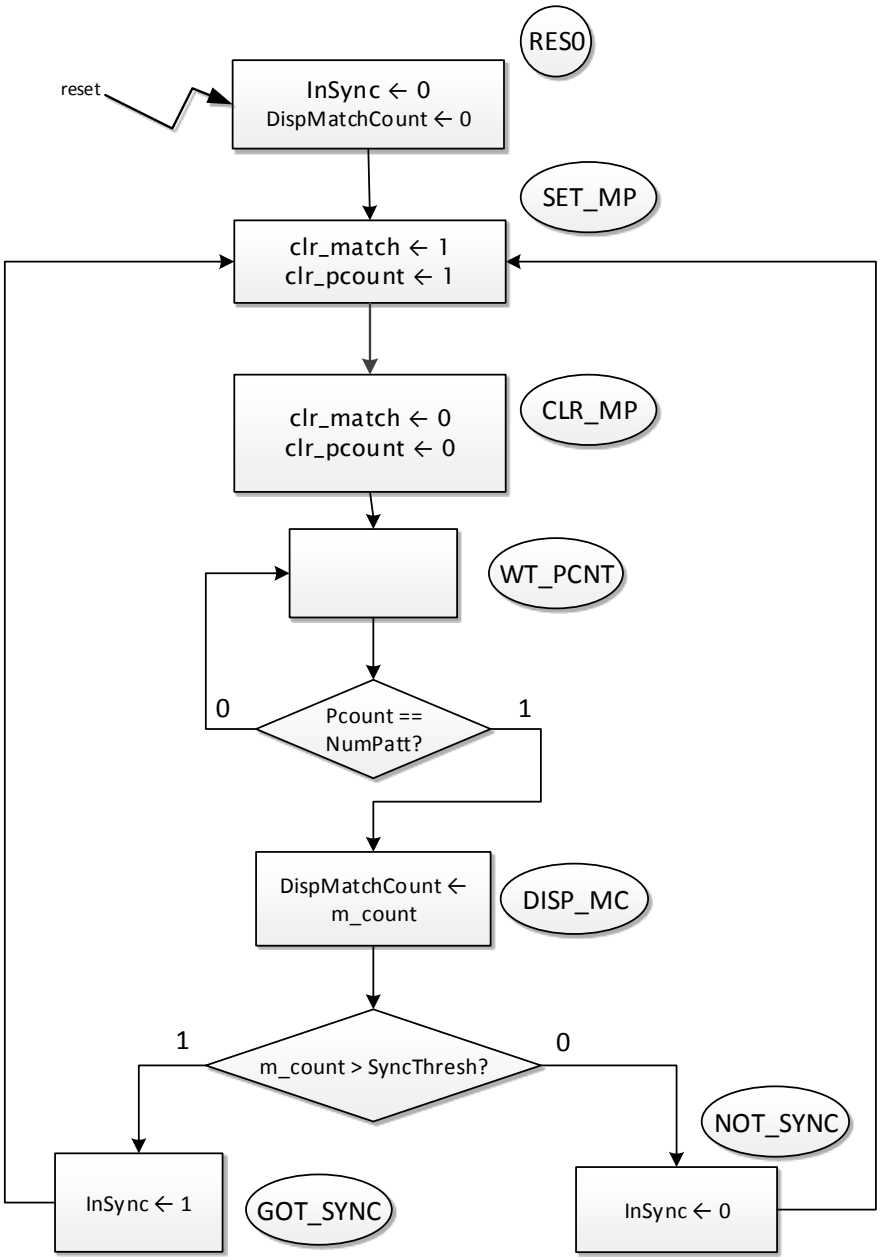


Figure 6b – ASM Diagram for 'Sync_control' module

Provide a short explanation of the operation of the 'Sync_control' module, based upon the ASM diagram of figure 6b. Explain how this block works with the other blocks in the system of figure 1c. [5 marks]

Task 11 – Simulation of the complete BERT system

Using figure 1c as a guide, create a SystemVerilog source description for the top-level Bit Error Rate Tester system (saved in a file named 'BERT.sv'), making use of consistent interconnection names to exploit automated connection wherever possible.
Note that this version of the top-level design does not require the inclusion of the 'Display Driver' block, this will be added prior to implementation.

When instantiating the sub-modules within the top-level 'BERT' module, set the parameters as follows:

ManchTrans: N = 10, TapMask = 10-bit feedback mask to give maximal length sequence

DelayComp: D = 8

Sync_control: P = 10, SyncThresh = 16'h0055, NumPatt = 101

[10 marks]

Perform behavioural simulations of the complete BERT system (excluding the display driver logic), making use of a test-module including the transmission delay block, with the system in the so-called 'loop-back' configuration, where the transmitted data output (man_tx) is connected to the received data input (man_rx), via the channel delay model ('TransDelay'). Initially, set the delay to 0 ns and disable the insertion of errors into the transmitted data. Capture all simulation waveforms and insert them into your report with comments.

Re-invoke the simulation with different channel delay times and demonstrate that the system can achieve synchronisation by adjusting the 'Delay' input value.
Document all simulation results with appropriate comments and annotations.

Finally, turn on the error insertion logic and verify that the system performs correctly, i.e. show how the 'DispMatchCount' depends on the error rate and the relationship between this and the 'Sync_LED' output.

[15 marks]

Task 12 – Implementation and loop-back testing of the complete BERT system

Go to eLearning and obtain the display driver SV module source files, these can be found in the following archive:

Course Documents – Workshop - Basys3 XADC with Display and RTL Controller - XADC_RTL_Display_R2RDAC.zip

Open the above file and extract the following files:

'DispMux.sv', 'DispCntr.sv' and 'bcd2seg.sv'.

Add the above files to the project as design sources and instantiate the modules into the top-level BERT module to create the display driver circuit, the connections and FPGA pin numbers are identical to those used in the 'XADC Display RTL.pdf' document located in the same item as the above zip file.

Set the parameters of the 'SyncControl' block for implementation, this will involve changing 'SynchThresh' and 'NumPatt', the latter being 1001_{10}.

The instance of the display counter, 'dispcntr', should be configured to refresh the 4-digit display at a suitable rate.

The master constraint file, 'Basys3_Master.xdc', must be edited and save under the name 'BERT.xdc', in order to assign FPGA pins to the top-level input/outputs of the design.
For example, the 'rst' (system reset) input signal is assigned to FPGA pin 'U18', which is in turn connected to the centre push-button 'btnC' as follows:

```
##Buttons

set_property PACKAGE_PIN U18 [get_ports rst]

set_property IOSTANDARD LVCMOS33 [get_ports rst]
```

In the above, 'rst' replaces 'btnC'.

Put a listing of the constraint file ‘BERT.xdc’ in an appendix of your assignment report.

Add the constraint file to the Vivado project and run the synthesis and implementation processes on the top-level design module. Finally, generate the configuration file (‘BERT.bit’) and use the Hardware Manager to configure the Artix-7 FPGA on the Basys3 board.
With a wire link connecting pins 1 and 4 of PMOD connector ‘JB’, demonstrate the BERT system in ‘loop-back’ configuration, with and without error insertion.

[20 marks]

File ‘En0720_KD7020_assignment_2017_18.docx’, September 2017.

Appendix A – SystemVerilog Source Listings

Source file 'TransmissionDelay.sv'

```
`timescale 1 ns/ 1 ps

//SV model of a pure time delay

module TransDelay #(parameter real delay = 1.5e-6) //time delay in seconds
        (input logic T,          //transmit signal
        output logic R);         //received signal

always @(T)                //delay is calculated in nanoseconds
  R <= #(1.0e9*delay) T;  //transport delay in ns

endmodule : TransDelay
```

Source file 'Test_ManchTrans.sv'

```
//SV test-module for Manchester Transmitter block 'ManchTrans'

`timescale 1ns / 1ps

module test_ManchTrans();

//inputs
logic clock, rst, clr_pcount;

logic Error_on;

logic [2:0] ErrRate;

//outputs
logic man_tx, man_bit_clk, nrz, transmit_bit_enable;

logic [3:0] Pcount;

ManchTrans #(.N(4), .TapMask(4'b1100)) DUT(.*);

//100MHz clock
initial begin
   clock = 1'b0;
   forever
      #5 clock = ~clock;
end

initial begin
   rst = 1'b1;
   clr_pcount = 1'b0;
   Error_on = 1'b0;
   ErrRate = 0;
   repeat (5) @(negedge clock);
   rst = 1'b0;
   repeat (205) @(negedge clock);
   clr_pcount = 1'b1;
   repeat (1) @(negedge clock);
   clr_pcount = 1'b0;
   repeat (200) @(negedge clock);
   $stop;
end

endmodule : test_ManchTrans
```

Source file ‘dig_delay.sv’

```
module dig_delay #(parameter N = 8)
       (input logic clock, rst, Din,
       input logic [N-1:0] Delay,
       output logic Dout);

logic [2**N – 1 : 0] q;

always_ff @(posedge clock, posedge rst)
begin
       if (rst == 1'b1)
              q <= 0;
       else
              q <= {q[2**N - 2 : 0], Din};
end

assign Dout = q[Delay];

endmodule : dig_delay
```