ECE 332 – Embedded Systems Lab

Lab 3: RLE Compression using Verilog and Verification using Functional Simulation

Objectives

- Learn to write Verilog for a custom design
- Understand how to verify your design using functional simulation
- Learn to write Verilog test bench for your design

Run Length Encoding

- Takes data stream and records how many bits are the same

- Output the value of the bit and the number of iterations of that value
  - ID Value
  - Count Value

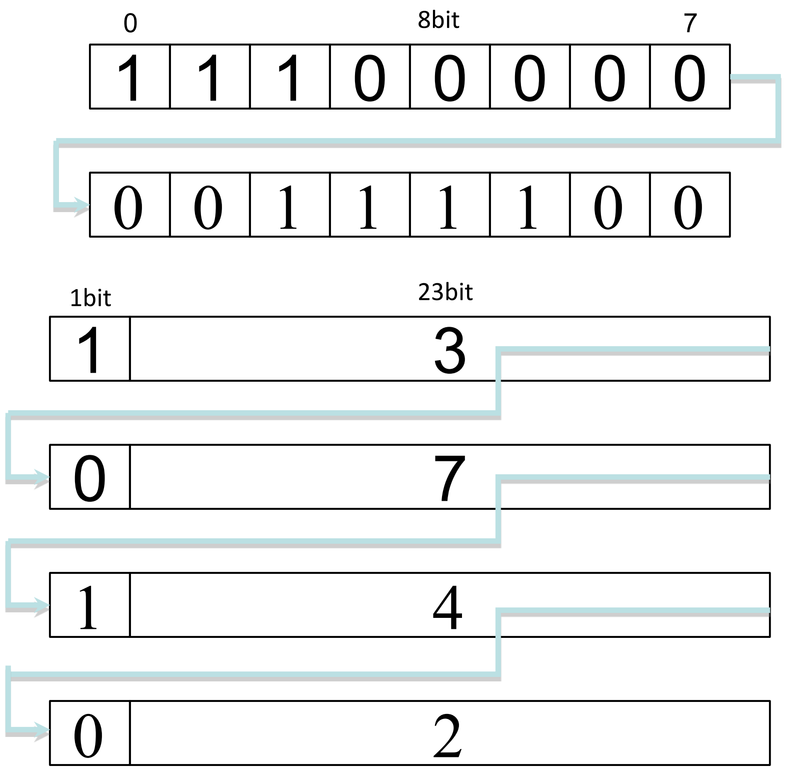
Run Length Encoding (Contd..)

- Input – 8 Bit data stream segments
- Output – 24 Bit data
- How does it work?
  - First bit – ID Bit (0 or 1)
  - Count value - 23 bits (Number of 0s or 1s)
- More of same value in sequence, more data is saved
- Worst Case - Interchanging data (01010101.....)

Illustration

Original Data Stream Segment

Encoded Data Stream Segment



RLE Implementation

- Implemented using a state machine written in Verilog (provided)
  - Understand the state diagram (appendix)
  - State transitions provided
  - Fill in the commented sections

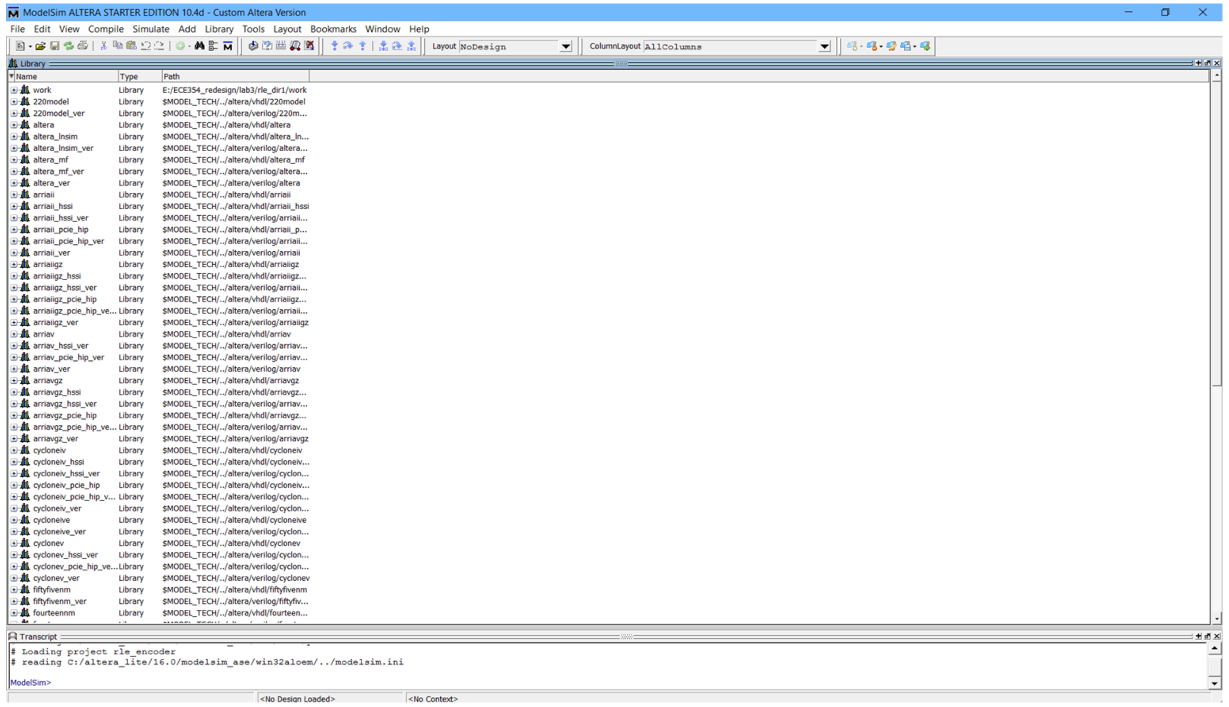
```
always @(posedge clk) begin
        if(rst) state <= INIT;
        else state <= next_state;

        case(state)
        INIT: begin
                //Initialize registers
        end
        REQUEST_INPUT: begin
                //Assert rd_req signal to FIFO by setting rd_reg
                //FIFO takes rd_req signal at next clock
        end
        WAIT_INPUT: begin
                //De-assert rd_req by setting rd_reg
```

Functional Simulation

- Technique to verify the functionality of a hardware design (Design verification)
- Standard procedure followed in the industry before implementing in a target device
- Provide the inputs to your design and check the outputs to confirm functionality
- ModelSim from Mentor Graphics is a well known tool used in the industry

ModelSim – Functional simulation tool

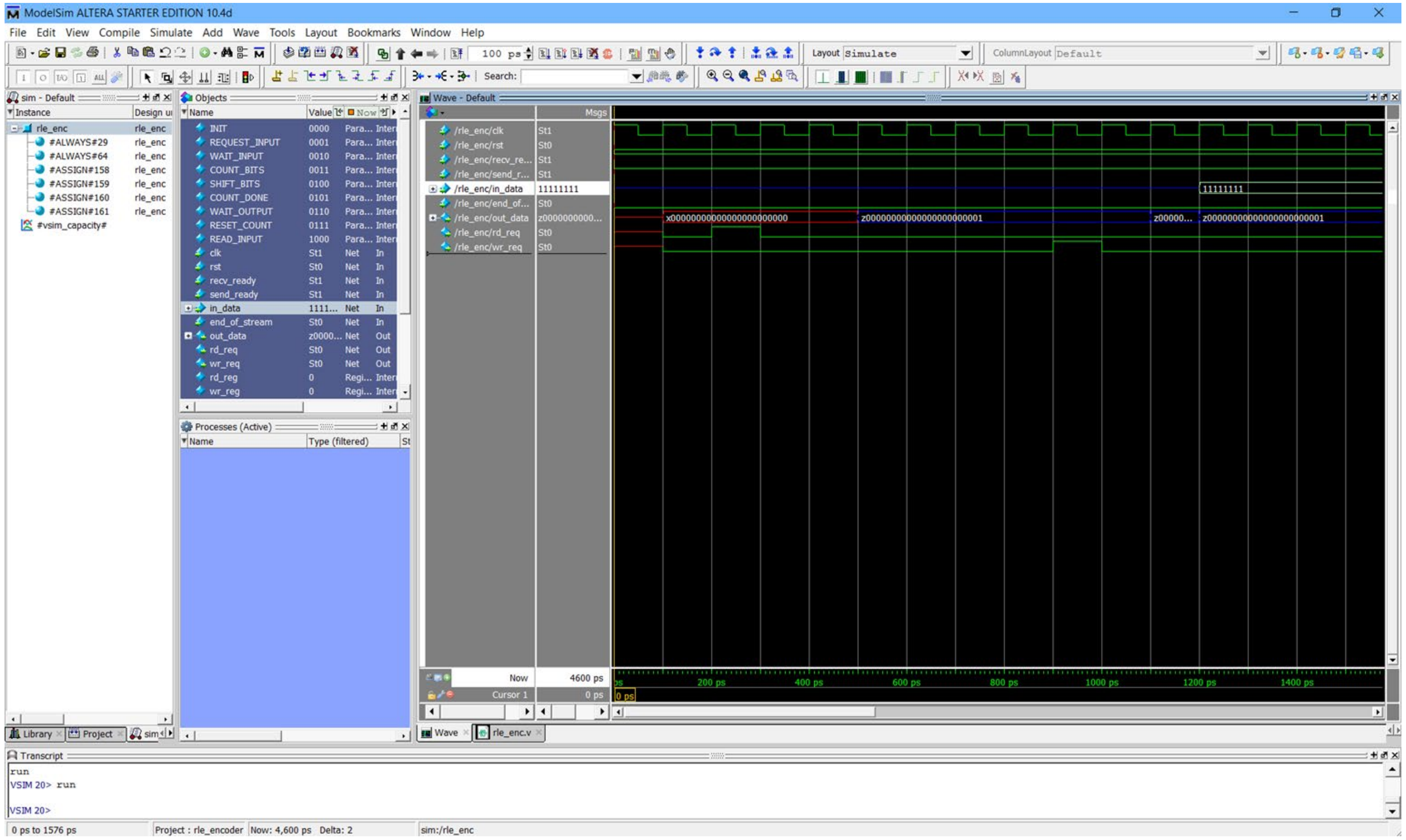
- Two ways to perform the design verification
  - Set the inputs through wave editor and observe the outputs
  - Design a test bench for your design


ModelSim ALTERA STARTER EDITION 10.4d - Custom Altera Version
File Edit View Compile Simulate Add Library Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Library

Name	Type	Path
work	Library	E:/ECE354_redesign/lab3/rle_dir1/work
220model	Library	$MODEL_TECH/../altera/vhdl/220model
220model_ver	Library	$MODEL_TECH/../altera/verilog/220m...
altera	Library	$MODEL_TECH/../altera/vhdl/altera
altera_lnsim	Library	$MODEL_TECH/../altera/vhdl/altera_ln...
altera_lnsim_ver	Library	$MODEL_TECH/../altera/verilog/altera...
altera_mf	Library	$MODEL_TECH/../altera/vhdl/altera_mf
altera_mf_ver	Library	$MODEL_TECH/../altera/verilog/altera...
altera_ver	Library	$MODEL_TECH/../altera/verilog/altera
arriaii	Library	$MODEL_TECH/../altera/vhdl/arriaii
arriaii_hssi	Library	$MODEL_TECH/../altera/vhdl/arriaii_hssi
arriaii_hssi_ver	Library	$MODEL_TECH/../altera/verilog/arriaii...
arriaii_pcie_hip	Library	$MODEL_TECH/../altera/vhdl/arriaii_p...
arriaii_pcie_hip_ver	Library	$MODEL_TECH/../altera/verilog/arriaii...
arriaii_ver	Library	$MODEL_TECH/../altera/verilog/arriaii
arriaiigz	Library	$MODEL_TECH/../altera/vhdl/arriaiigz
arriaiigz_hssi	Library	$MODEL_TECH/../altera/vhdl/arriaiigz...
arriaiigz_hssi_ver	Library	$MODEL_TECH/../altera/verilog/arriaii...
arriaiigz_pcie_hip	Library	$MODEL_TECH/../altera/vhdl/arriaiigz...
arriaiigz_pcie_hip_ve...	Library	$MODEL_TECH/../altera/verilog/arriaii...
arriaiigz_ver	Library	$MODEL_TECH/../altera/verilog/arriaiigz
arriav	Library	$MODEL_TECH/../altera/vhdl/arriav
arriav_hssi_ver	Library	$MODEL_TECH/../altera/verilog/arriav...
arriav_pcie_hip_ver	Library	$MODEL_TECH/../altera/verilog/arriav...
arriav_ver	Library	$MODEL_TECH/../altera/verilog/arriav
arriavgz	Library	$MODEL_TECH/../altera/vhdl/arriavgz
arriavgz_hssi	Library	$MODEL_TECH/../altera/vhdl/arriavgz...
arriavgz_hssi_ver	Library	$MODEL_TECH/../altera/verilog/arriav...
arriavgz_pcie_hip	Library	$MODEL_TECH/../altera/vhdl/arriavgz...
arriavgz_pcie_hip_ve...	Library	$MODEL_TECH/../altera/verilog/arriav...
arriavgz_ver	Library	$MODEL_TECH/../altera/verilog/arriavgz
cycloneiv	Library	$MODEL_TECH/../altera/vhdl/cycloneiv
cycloneiv_hssi	Library	$MODEL_TECH/../altera/vhdl/cycloneiv...
cycloneiv_hssi_ver	Library	$MODEL_TECH/../altera/verilog/cyclon...
cycloneiv_pcie_hip	Library	$MODEL_TECH/../altera/vhdl/cycloneiv...
cycloneiv_pcie_hip_v...	Library	$MODEL_TECH/../altera/verilog/cyclon...
cycloneiv_ver	Library	$MODEL_TECH/../altera/verilog/cyclon...
cycloneive	Library	$MODEL_TECH/../altera/vhdl/cycloneive
cycloneive_ver	Library	$MODEL_TECH/../altera/verilog/cyclon...
cyclonev	Library	$MODEL_TECH/../altera/vhdl/cyclonev
cyclonev_hssi_ver	Library	$MODEL_TECH/../altera/verilog/cyclon...
cyclonev_pcie_hip_ve...	Library	$MODEL_TECH/../altera/verilog/cyclon...
cyclonev_ver	Library	$MODEL_TECH/../altera/verilog/cyclonev
fiftyfivenm	Library	$MODEL_TECH/../altera/vhdl/fiftyfivenm
fiftyfivenm_ver	Library	$MODEL_TECH/../altera/verilog/fiftyfiv...
fourteennm	Library	$MODEL_TECH/../altera/vhdl/fourteen...

```
# Loading project rle_encoder
# reading C:/altera_lite/16.0/modelsim_ase/win32aloem/../modelsim.ini

ModelSim>
```

Wave Editor in ModelSim


ModelSim ALTERA STARTER EDITION 10.4d
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout Default
sim - Default
Objects
Processes (Active)
Wave - Default
Transcript

Instance	Design unit
rle_enc	rle_enc
#ALWAYS#29	rle_enc
#ALWAYS#64	rle_enc
#ASSIGN#158	rle_enc
#ASSIGN#159	rle_enc
#ASSIGN#160	rle_enc
#ASSIGN#161	rle_enc
#vsim_capacity#	

Name	Value		
INIT	0000	Para...	Inter
REQUEST_INPUT	0001	Para...	Inter
WAIT_INPUT	0010	Para...	Inter
COUNT_BITS	0011	Para...	Inter
SHIFT_BITS	0100	Para...	Inter
COUNT_DONE	0101	Para...	Inter
WAIT_OUTPUT	0110	Para...	Inter
RESET_COUNT	0111	Para...	Inter
READ_INPUT	1000	Para...	Inter
clk	St1	Net	In
rst	St0	Net	In
recv_ready	St1	Net	In
send_ready	St1	Net	In
in_data	1111...	Net	In
end_of_stream	St0	Net	In
out_data	z0000...	Net	Out
rd_req	St0	Net	Out
wr_req	St0	Net	Out
rd_reg	0	Regi...	Inter
wr_reg	0	Regi...	Inter

	Msgs
/rle_enc/clk	St1
/rle_enc/rst	St0
/rle_enc/recv_re...	St1
/rle_enc/send_r...	St1
/rle_enc/in_data	11111111
/rle_enc/end_of...	St0
/rle_enc/out_data	z0000000000...
/rle_enc/rd_req	St0
/rle_enc/wr_req	St0
Now	4600 ps
Cursor 1	0 ps

x000000000000000000000000
z000000000000000000000001
z00000...
z000000000000000000000001
11111111
200 ps
400 ps
600 ps
800 ps
1000 ps
1200 ps
1400 ps
0 ps

```
run
VSIM 20> run

VSIM 20>
```

```
0 ps to 1576 ps    Project : rle_encoder  Now: 4,600 ps  Delta: 2    sim:/rle_enc
```

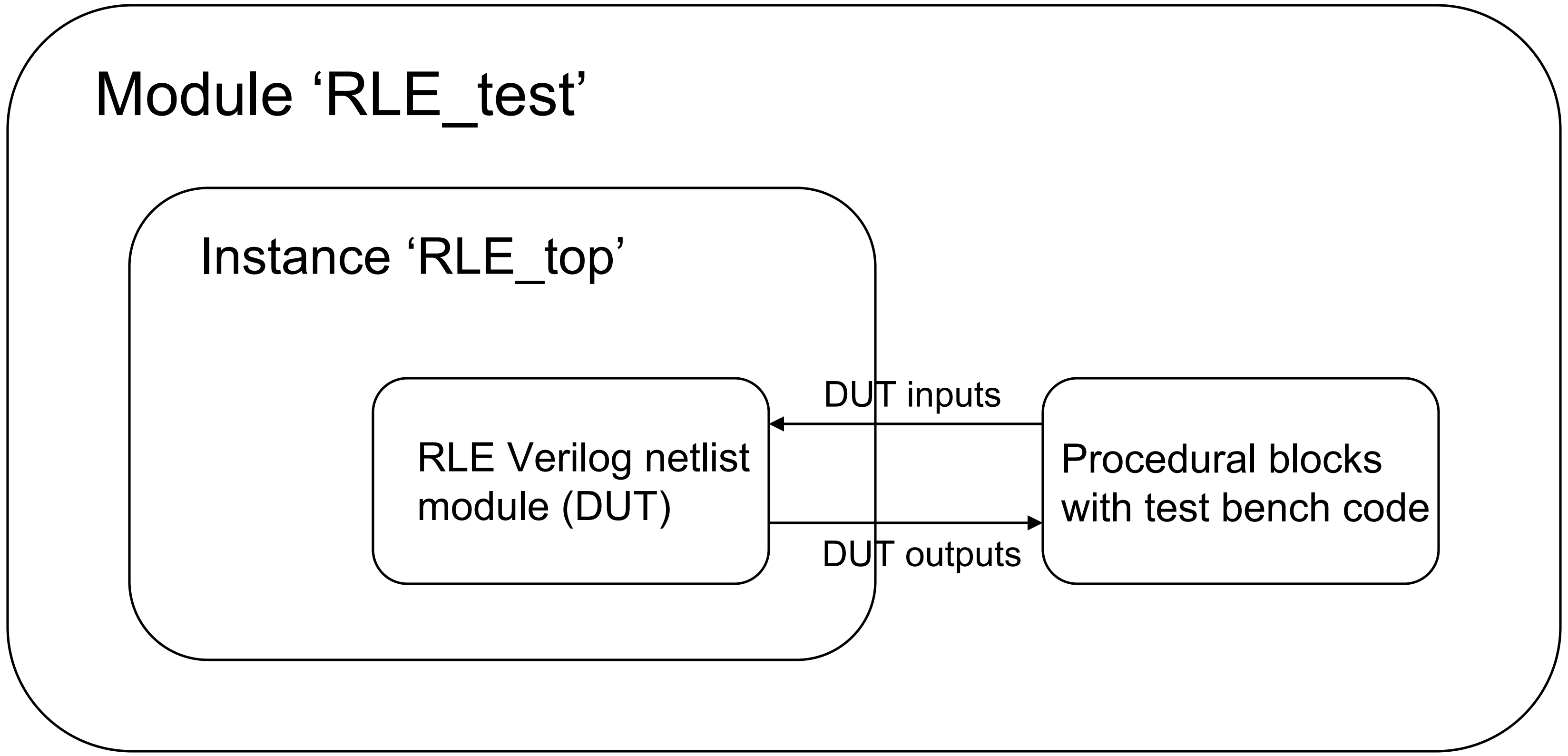
Test Bench

- Verilog wrapper module which does the following
  - Invokes the Design Under Test (DUT) - RLE
  - Provides the inputs to the DUT
  - Formats the outputs suitable for viewing

- Verilog provides constructs such as procedural blocks and timing controls

Good source: https://people.ece.cornell.edu/land/courses/ece5760/Verilog/LatticeTestbenchPrimer.pdf

Test Bench Template


Module 'RLE_test'
Instance 'RLE_top'
RLE Verilog netlist module (DUT)
DUT inputs
DUT outputs
Procedural blocks with test bench code

Summary

- To implement hardware for RLE
  - Encoding is done in Hardware (Verilog)

- Functional simulation
  - Verify the design by providing the input stimulus and observe the output using Verilog test bench

Appendix
Signals and their Description

RLE Internal Signals

- reg rd_reg
  - connected to rd_req;
- reg wr_reg
  - Connected to wr_req;
- reg [22:0] bit_count
  - Stores number of same consecutive bit in bit stream
  - value_type represent bit ID
- reg value_type
  - Bit ID

RLE Internal Signals

- reg [7:0] shift_buf
  - Store 8 bit segment of bit stream comes from input side FIFO
  - Will be shifted out to calculate number of bits
- reg [3:0] shift_count
  - Current shift amount of shift_buf
- reg new_bitstream
  - Indicate new bit sequence is starting
  - Current encoded data segment is passed to output side FIFO

RLE Internal Signals

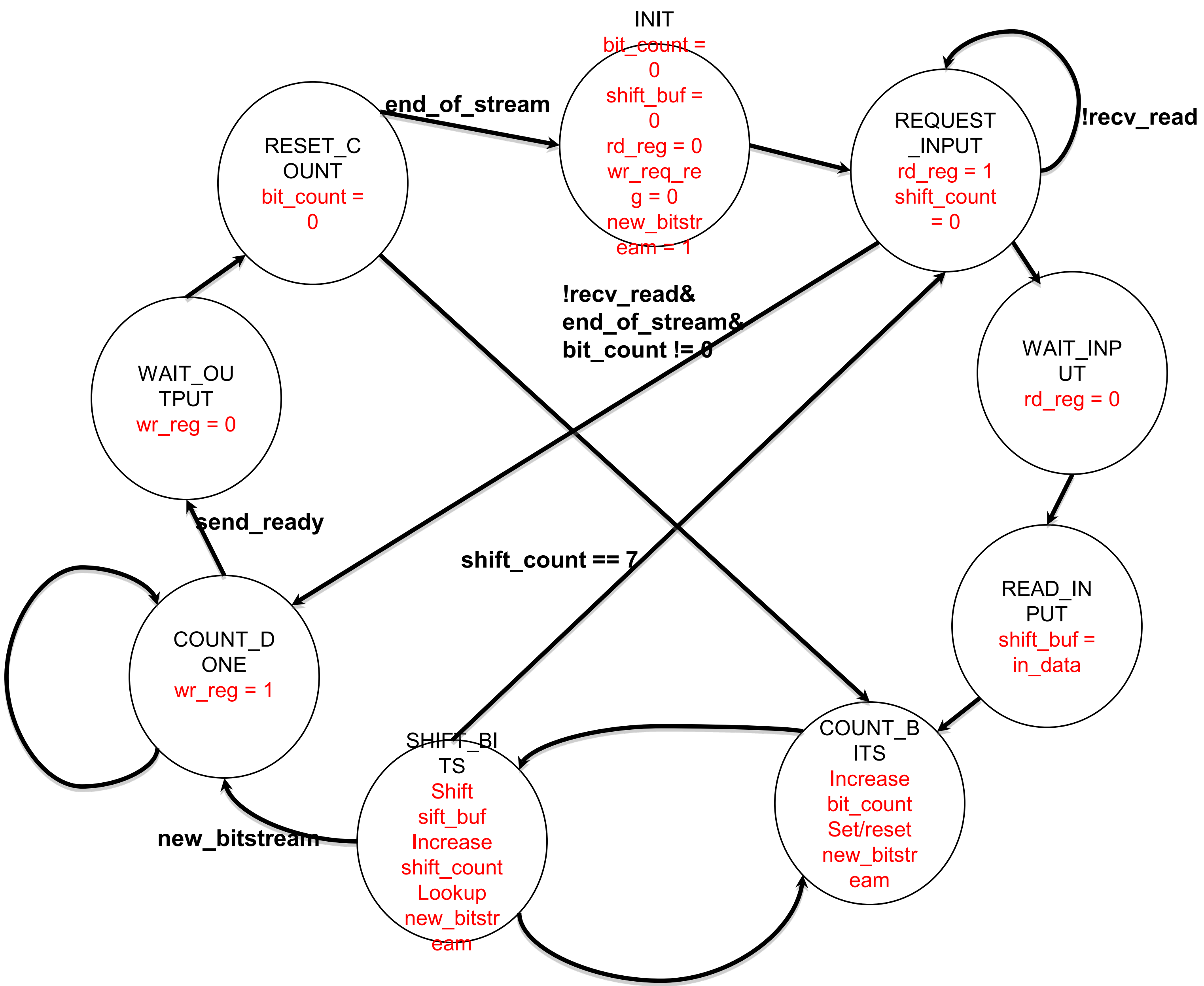
- reg [3:0] state
  - Represent State.
  - There are 9 states in total
- reg [3:0] next_state
  - Represent Next state

RLE Interface - Input

- input clk,rst
  - clk and reset
- input recv_ready
  - Connected with (!fifo_empty) of input side FIFO
- input send_ready
  - Connected with (!fifo_full) of output side FIFO
- input [7:0] in_data
  - Input data from input side FIFO
  - 8 bit segment of original bit stream
- input end_of_stream
  - Indicate the end of bit stream
  - Request flushing out the last segment

RLE Interface - Output

- output [23:0] out_data
  - Output data to output side FIFO
  - [23] has bit ID, [22:0] has bit counting value
- output rd_req
  - Read request for input side FIFO
- output wr_req
  - Write request for output side FIFO


INIT
bit_count = 0
shift_buf = 0
rd_reg = 0
wr_req_reg = 0
new_bitstream = 1
REQUEST_INPUT
rd_reg = 1
shift_count = 0
!recv_read
end_of_stream
RESET_COUNT
bit_count = 0
!recv_read&
end_of_stream&
bit_count != 0
WAIT_INPUT
rd_reg = 0
WAIT_OUTPUT
wr_reg = 0
send_ready
shift_count == 7
READ_INPUT
shift_buf = in_data
COUNT_DONE
wr_reg = 1
SHIFT_BITS
Shift
sift_buf
Increase
shift_count
Lookup
new_bitstream
COUNT_BITS
Increase
bit_count
Set/reset
new_bitstream
new_bitstream

RLE State in detail

- INIT
  - Initialize registers
- REQUEST_INPUT
  - Assert rd_req signal to FIFO by setting rd_reg
  - FIFO takes rd_req signal at next clock
- WAIT_INPUT
  - 1 cycle stall is needed
  - De-assert rd_req by setting rd_reg
- READ_INPUT
  - FIFO provides valid data after taking rd_req
  - shift_buf stores 8 bit input data

RLE State in detail

- COUNT_BITS
  - Count number of consecutive bits in shift_buf
  - If new type of bit starts, store bit ID in value_type register
  - If current value_type and shift_buf[0] is not matched, notify current encoding is completed and new encoding will be started
- SHIFT_BITS
  - Right shift the shift_buf
  - Increase shift_count
  - Look up new_bitstream
- COUNT_DONE
  - Assert wr_req by setting wr_reg
  - FIFO will take wr_req signal in next clock cycle

RLE State in detail

- WAIT_OUTPUT
  - 1 cycle stall is needed
  - De-assert wr_req by setting wr_reg
- RESET_COUNT
  - Reset bit counting register after passing encoded data to output side FIFO

Race Condition

An Example

```
always
@(posedge clk)
   a = b;
always
@(posedge clk)
   b = c;
```

Which statement will be executed first?

- Produces different simulation result from real hardware
- Keeping verilog design guideline is important

Important Design Guideline for LAB 3

- Do not mix BA and NBA in the same always block
- Use BA for combinational circuit in given design
  - State transition
- Use NBA for sequential circuit in given design
  - Updating registers