ECE 351
Verilog and FPGA Design

Lecture 2_2: Simulating w/ QuestaSim
Exercise 1 – Putting it all together

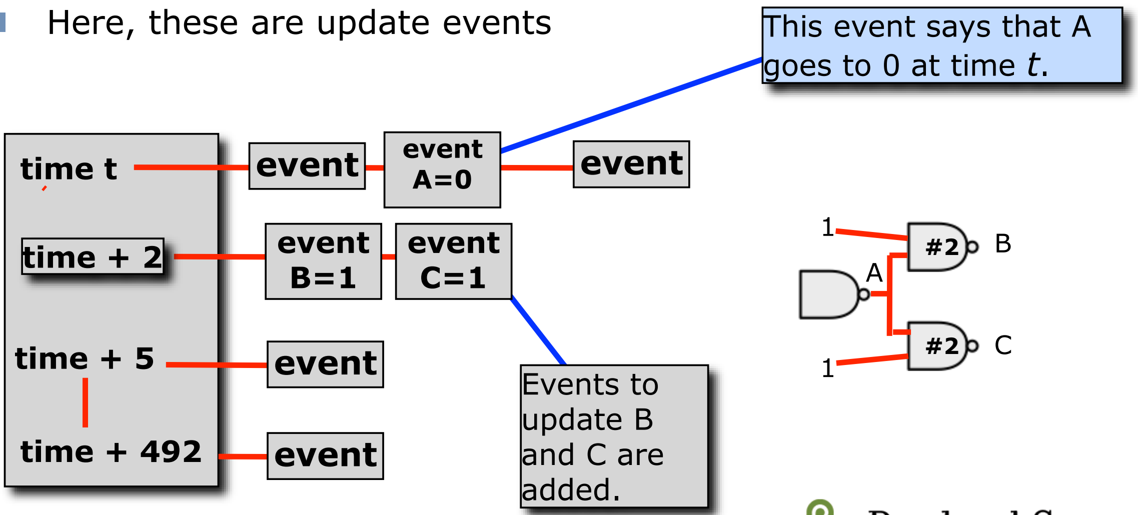
Roy Kravitz
Electrical and Computer Engineering Department
Maseeh College of Engineering and Computer Science

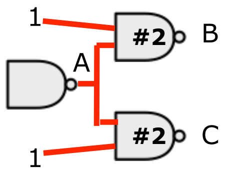
Portland State UNIVERSITY

1

Review: Event list

- A time-ordered list of events is maintained
  - All events for a given time are kept together in an event list ordered by time
  - Events are handled at the given time and possibly schedule their output to change at a later time (a new event)
  - Here, these are update events


This event says that A goes to 0 at time *t*.
time t
time + 2
time + 5
time + 492
event
event A=0
event
event B=1
event C=1
event
event
Events to update B and C are added.



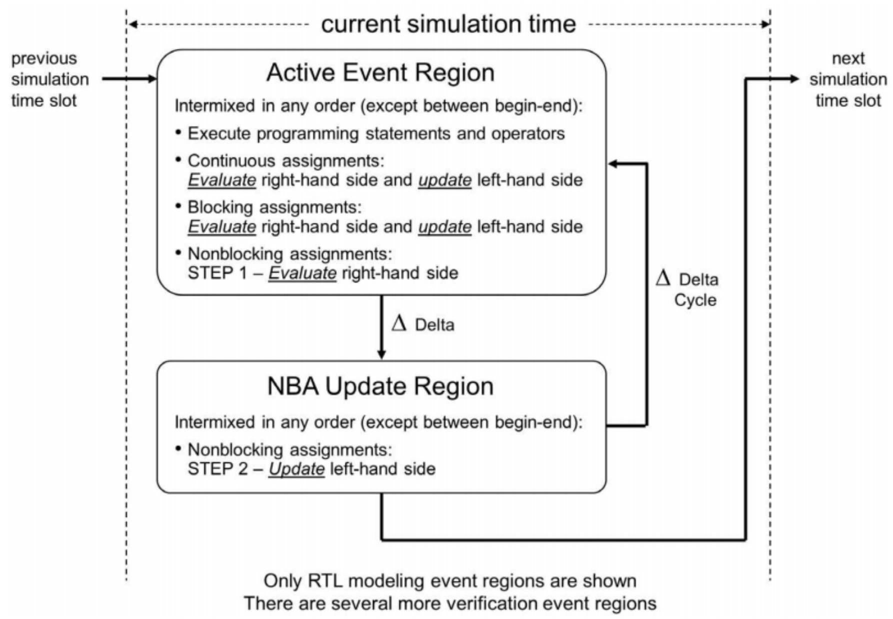
Portland State UNIVERSITY

Pseudo Code: Event-driven simulation

One traversal of the while loop is a *simulation cycle*.

```
put all initial and always blocks in the event list
while something in time-ordered event list {
        advance simulation time to first event's time
        retrieve all events e for this time
        update state from all update-event values

        For each event e in arbitrary order {
             If it's an update event {
                    follow fanout, evaluate gates there
                    If an output changes
                           schedule update event for it
             }
             else // it's an evaluation event
                    evaluate the model
        }
}
```

Event scheduling


current simulation time
previous simulation time slot
next simulation time slot
Active Event Region
Intermixed in any order (except between begin-end):
• Execute programming statements and operators
• Continuous assignments: *Evaluate* right-hand side and *update* left-hand side
• Blocking assignments: *Evaluate* right-hand side and *update* left-hand side
• Nonblocking assignments: STEP 1 – *Evaluate* right-hand side
Δ Delta
Δ Delta Cycle
NBA Update Region
Intermixed in any order (except between begin-end):
• Nonblocking assignments: STEP 2 – *Update* left-hand side
Only RTL modeling event regions are shown
There are several more verification event regions

- Blocking assigns (`assign a = b`) – model combinational logic
- Non-blocking assigns (`a <= b`) – model sequential logic

Portland State UNIVERSITY

Sutherland Figure 1-8

Logic simulation w/ QuestaSim

Example: ..\examples\ripple_carry_counter.pdf

QuestaSim Tutorial: ..\docs\Questa® SIM Tutorial.pdf

Using QuestaSim at PSU: ..\docs\UsingMentorQuestaAtPSU_R2_0.pdf

Some definitions

- **Module** – the basic building block in Verilog
  - Can be an element or a collection of lower-level design blocks
  - Can provide abstraction…hides the details of the implementation (i.e. possible to modify block internals without affecting the rest of the design)
- **Instance** – Module is a template, instance is the actual object
- **Simulation** – The act of applying inputs to and monitoring the outputs from a Verilog model
- There are two distinct components of a simulation
  - **Design Block** – the implementation of the desired functionality
  - **Stimulus Block (Test Bench or Testbench)** – The inputs applied to the design block

Portland State UNIVERSITY

Display tasks

- Verilog supports system tasks for performing I/O, generating random numbers, initializing memory, etc.
  - Modeled after C library calls, but typically less flexible
  - All system tasks have the form `$<keyword>`
- Display tasks:
  - `$display()` – prints information on `stdout` w/ new line char
  - `$write()` – prints information on `stdout` w/o new line char
  - `$strobe()` – like `$display()` except prints at end of time step (e.g. all events have been processed)
  - `$monitor()` – continually monitors all of the arguments and prints whenever any of the signals being monitored changes

Portland State UNIVERSITY

$display()

- `$display()` - displays values of variables or strings or expressions
  - *u*sage: `$display(format_desc, p1, p2, p3,…,pn);`
  - Format description is similar to C (`%d`, `%b`, etc. work)
  - Ex: `$display("ID of the port is %b", port_id);`
- You must explicitly tell Verilog to `$display` something…in other words, the `$display()` must be in a block of executable code
- Common problem: $display() is invoked in a situation where (simulation) time does not advance
  - Ex:

```
for (i = 0, i< 100, i = i+1)
    $display($time, x, y, z);
```

  - Simple fix: `#5 $display(…); // add some delay`

Portland State UNIVERSITY

$monitor()

- `$monitor()` - displays values of variables or strings or expressions whenever their value(s) change
  - *usage*: `$monitor(p1, p2, p3,…,pn);`
  - All the variables in the list are displayed whenever one or more of them change
  - Ex:

```
$monitor($time, “clock = %b, reset = %b”, clock, reset);
```

  - `$monitoron`, `$monitoroff` enable and disable monitoring

Portland State UNIVERSITY

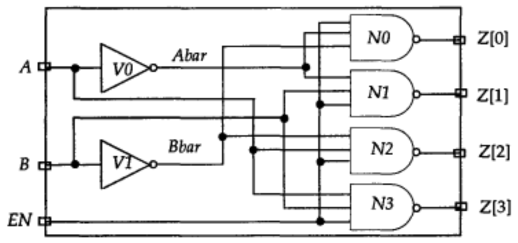
Delays

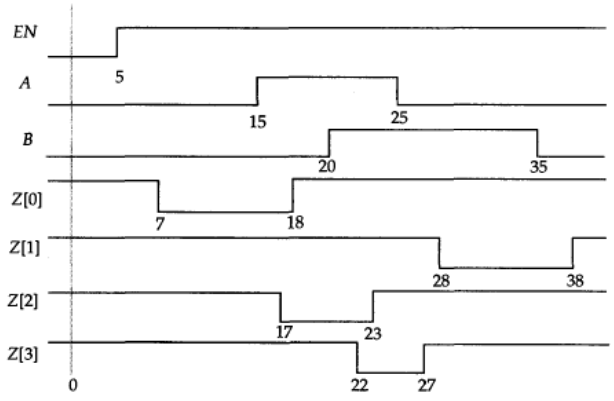
- SystemVerilog simulators model the behavior of a digital system over time and support delays (specified in time units)
  - `assign #2 sum = a ^ b  // #2 refers to time units`
  - Behavior: `sum` is assigned to `a ^ b` two time units after the current time whenever either `a` or `b` or both change
- You can associate time units to "physical" time using the `timeunit and timeprecison statements`
  - `ex: timeunit 1ns; timeprecision 100ps` (or `timeunit 1ns/100ps`)
    - 1 time unit = 1ns, precision is 100ps (e.g., all delays rounded to 0.1ns)
  - Normally put into each module but only useful for simulation…synthesis ignores delays in the code because delays are technology-specific
  - No default defined in the spec…up to the vendor
- There are several ways to assign delays and they are dependent on context…more on this later

Portland State UNIVERSITY

Delays (cont'd)

```
module decoder2x4 (
 input logic A, B, EN,
 output logic [0:3] Z
);

timeunit 1ns/1ns;

logic Abar, Bbar;

assign #1 Abar = ~A;
assign #1 Bbar = ~B

assign #2 Z[0] = ~(Abar & Bbar & EN);
assign #2 Z[1] = ~(Abar & B & EN);
assign #2 Z[2] = ~(A & Bbar & EN);
assign #2 Z[3] = ~(A & B & EN);

endmodule : decoder2x4
```

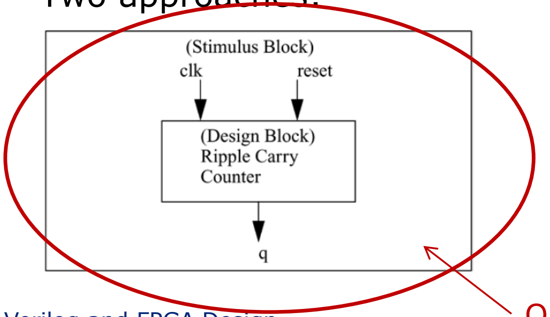


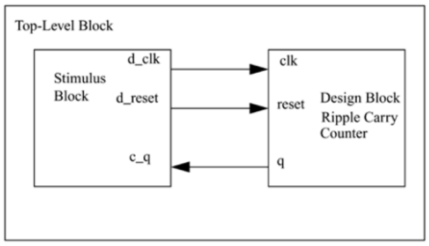


Original example: Bhasker, Verilog HDL Primer, 3rd Edition

Testbenches

- The functionality of a design can be tested by applying stimulus and checking for a correct result
  - The stimulus block is commonly called a Test Bench
  - Test benches can be written in Verilog using the full feature set of the language
  - You can build one or more test benches, each focusing on a specific piece of functionality
  - Obvious…but worth saying – The correctness of the design depends heavily on the effectiveness of the test cases
  - Two approaches:

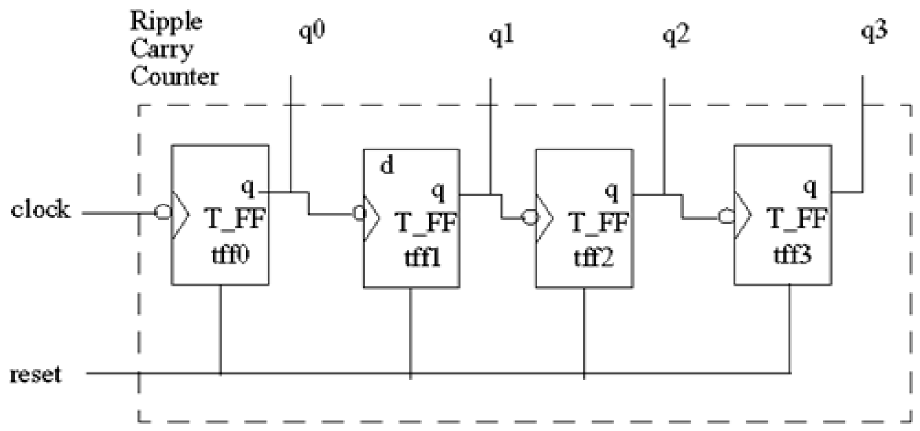


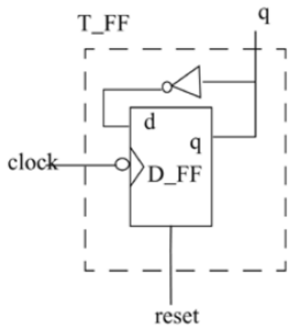


Our Example

Portland State UNIVERSITY

Example: 4-bit Ripple Carry Counter





reset	q_n	q_{n+1}
1	1	0
1	0	0
0	0	1
0	1	0

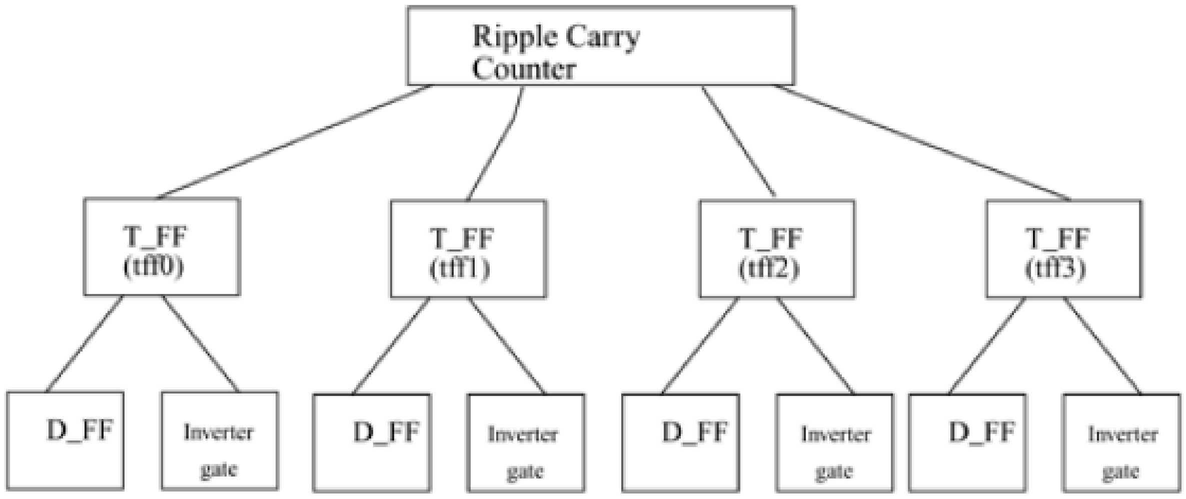
Specification:

Increments count (*q[0:3]*) by 1 on every negative edge of *clock*; count is set 4'b0000 when *reset* is asserted

Portland State UNIVERSITY

Source: Palnitkar, Verilog HDL, 2nd Edition

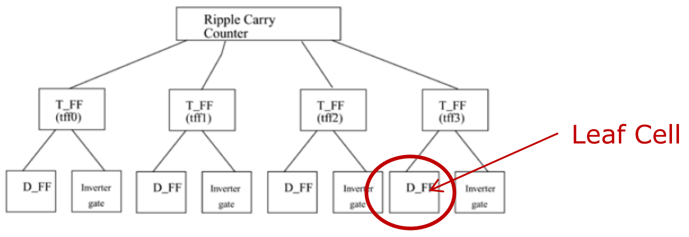
Hierarchy



Source: Palnitkar, Verilog HDL, 2nd Edition

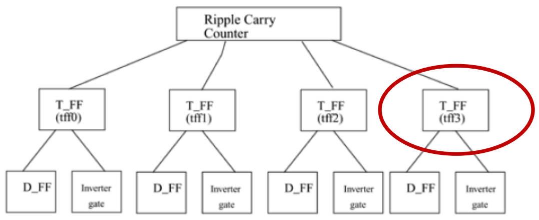
Design block – D flip-flop

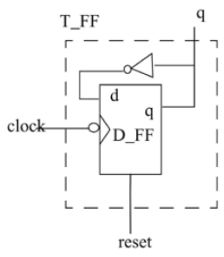
```
// D flip-flop with asynchronous reset – RTL model
module DFF(
  output logic q,
  input logic d, clk, reset
);

always_ff @(negedge clk or posedge reset) begin
  if (reset)
    q <= 1'b0;
  else
    q <= d;
end

endmodule : DFF
```



Design Block – TFF Sub-block

```
// Toggle Flip-flop
module TFF (
  output logic q,
  input logic clk, reset
);

logic d;

// instantiate a D flip-flop
DFF dff0(.q, .d, .clk, .reset);

// create the toggle flip-flop by driving input with inverted output
not n1(d, q); // not is a Verilog provided primitive.

endmodule : TFF
```

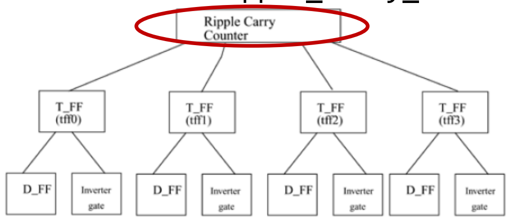


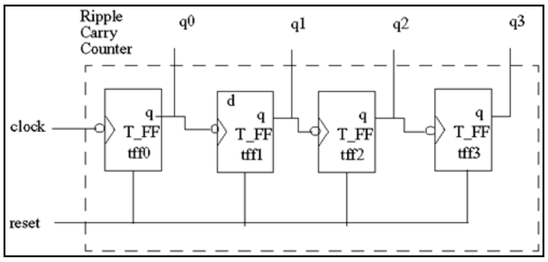


Portland State UNIVERSITY

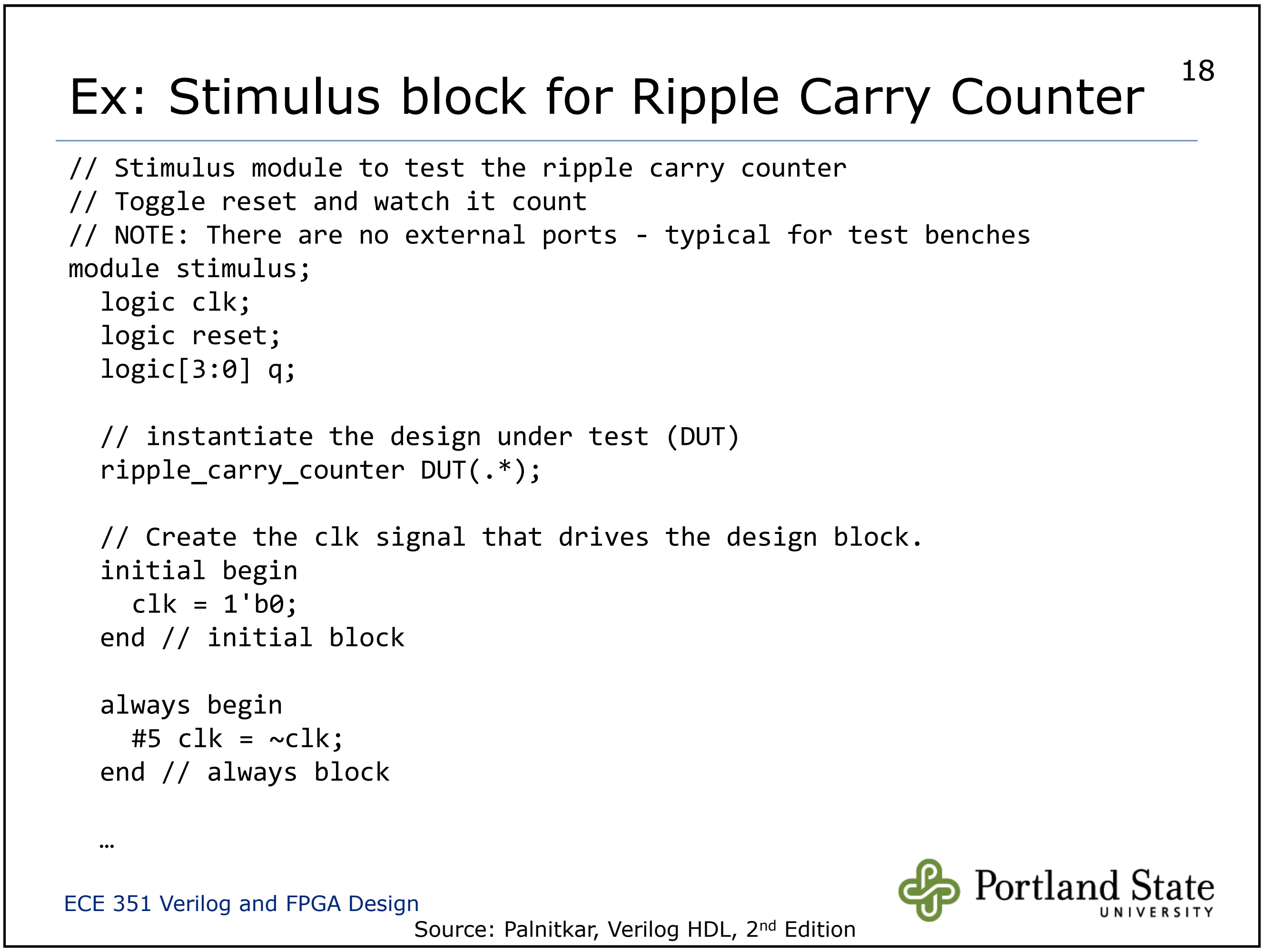
Design block – top-level

```
// Top level module for 4-bit Ripple Carry Counter
module ripple_carry_counter (
  output [3:0] count,
  input clk, reset
);
// internal signals
logic q0, q1, q2, q3;

//instantiate 4 toggle flip-flops
TFF tff0(q0,clk, reset);
TFF tff1(q1,q0, reset);
TFF tff2(q2,q1, reset);
TFF tff3(q3,q2, reset);
assign count = {q3, q2, q1, q0};

endmodule : ripple_carry_counter
```





Source: Palnitkar, Verilog HDL, 2nd Edition



Ex: Stimulus block for Ripple Carry Counter

```
// Stimulus module to test the ripple carry counter
// Toggle reset and watch it count
// NOTE: There are no external ports - typical for test benches
module stimulus;
  logic clk;
  logic reset;
  logic[3:0] q;

  // instantiate the design under test (DUT)
  ripple_carry_counter DUT(.*);

  // Create the clk signal that drives the design block.
  initial begin
    clk = 1'b0;
  end // initial block

  always begin
    #5 clk = ~clk;
  end // always block

  …
```

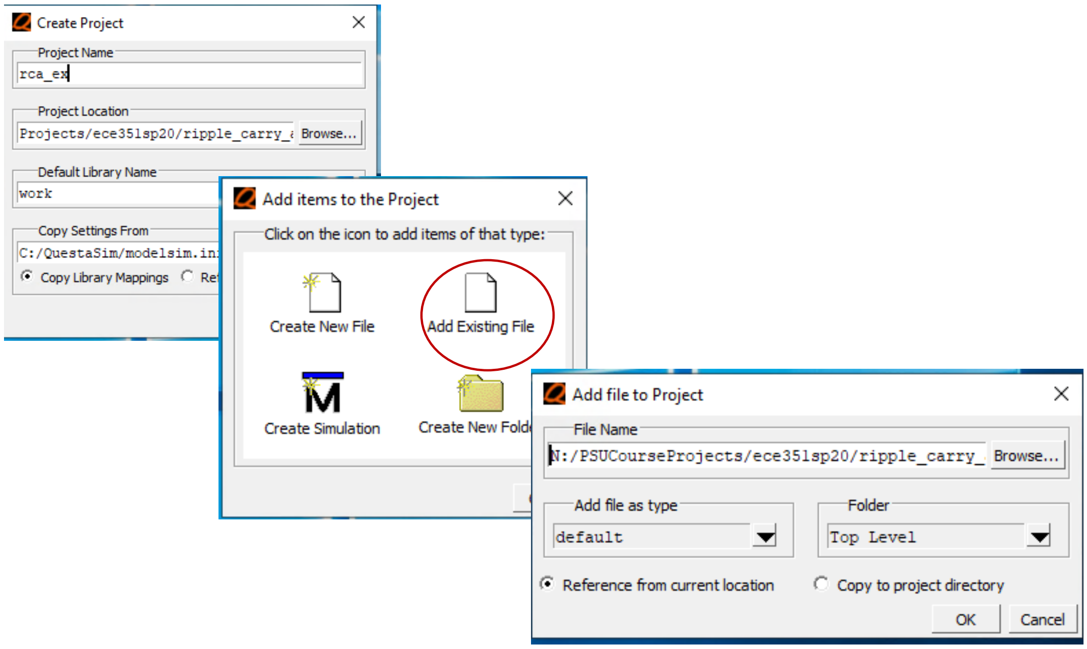
Source: Palnitkar, Verilog HDL, 2nd Edition

Ex: Stimulus Block for Ripple Carry Counter

```
// Monitor the outputs
  initial begin
    $monitor($time, " Output q = %d",  q);
  end

  // Control the reset signal that drives the design block
  initial begin : test_vectors
    #0  reset = 1'b1;
    #15   reset = 1'b0;
    #180reset = 1'b1;
    #10   reset = 1'b0;
    #20   $stop;
  end : test_vectors
endmodule : stimulus
```

Using the QuestaSim GUI

Create a new QuestaSim Project and add the source files
`File/New Project…`



Create Project

Project Name: rca_ex

Project Location: Projects/ece351sp20/ripple_carry_

Default Library Name: work

Copy Settings From: C:/QuestaSim/modelsim.in

(● Copy Library Mappings ○ Re

Add items to the Project

Click on the icon to add items of that type:

Add file to Project

File Name: N:/PSUCourseProjects/ece351sp20/ripple_carry_

Add file as type: default

Folder: Top Level

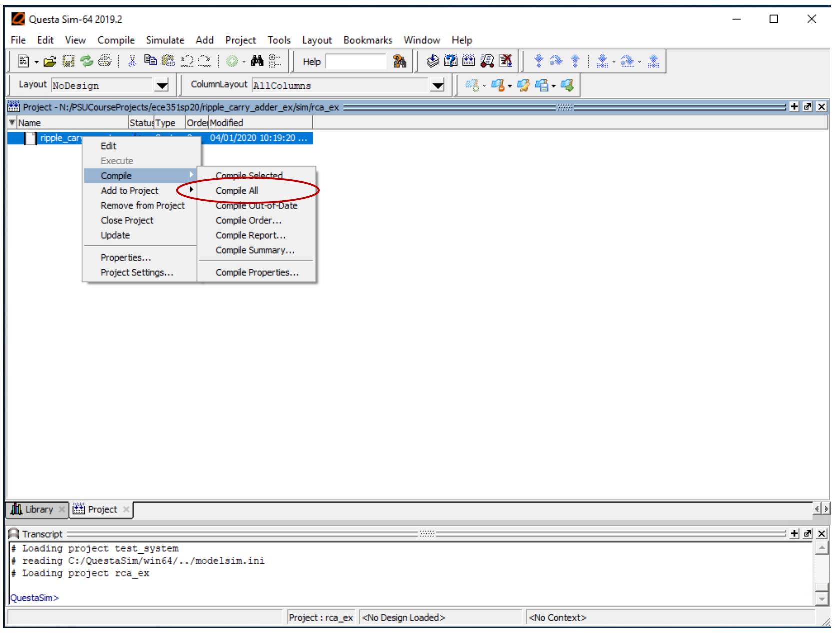
● Reference from current location ○ Copy to project directory

Portland State UNIVERSITY

Using the QuestaSim GUI (cont'd)

Compile the source files

`Compile/Compile All`



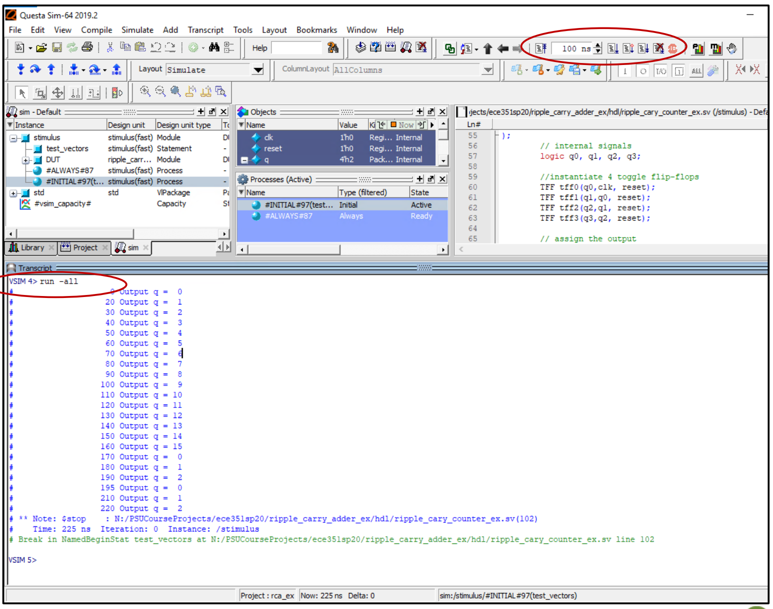
```
# Loading project test_system
# reading C:/QuestaSim/win64/../modelsim.ini
# Loading project rca_ex

QuestaSim>
```

Portland State UNIVERSITY

Using the QuestaSim GUI (cont’d)

Load the simulation model

`Select Library tab, open work directory, select top module, right click and select Simulate`



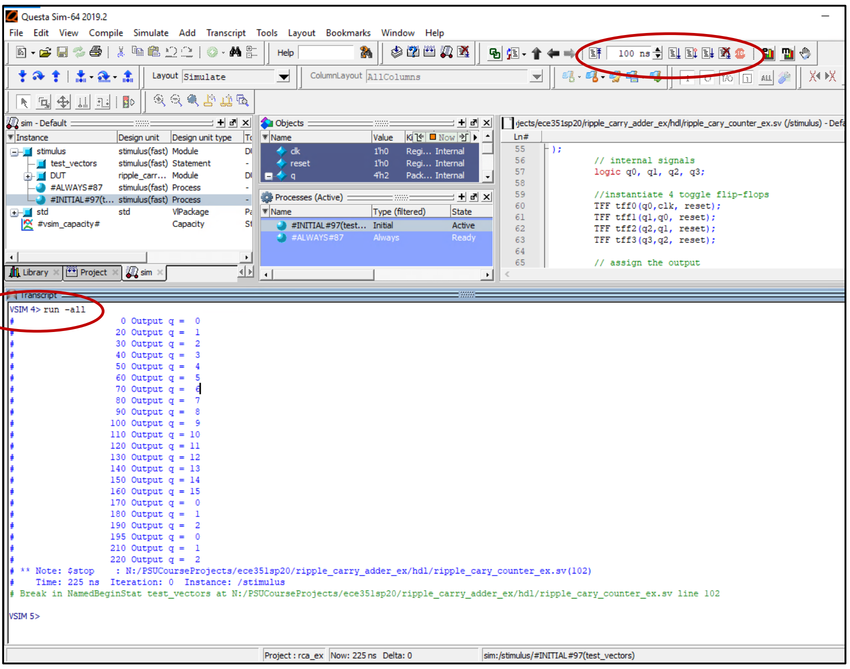
```
    // internal signals
    logic q0, q1, q2, q3;

    //instantiate 4 toggle flip-flops
    TFF tff0(q0,clk, reset);
    TFF tff1(q1,q0, reset);
    TFF tff2(q2,q1, reset);
    TFF tff3(q3,q2, reset);

    // assign the output
```

```
VSIM 4> run -all
#                    0 Output q =  0
#                   20 Output q =  1
#                   30 Output q =  2
#                   40 Output q =  3
#                   50 Output q =  4
#                   60 Output q =  5
#                   70 Output q =  6
#                   80 Output q =  7
#                   90 Output q =  8
#                  100 Output q =  9
#                  110 Output q = 10
#                  120 Output q = 11
#                  130 Output q = 12
#                  140 Output q = 13
#                  150 Output q = 14
#                  160 Output q = 15
#                  170 Output q =  0
#                  180 Output q =  1
#                  190 Output q =  2
#                  195 Output q =  0
#                  210 Output q =  1
#                  220 Output q =  2
# ** Note: $stop    : N:/PSUCourseProjects/ece351sp20/ripple_carry_adder_ex/hdl/ripple_cary_counter_ex.sv(102)
#    Time: 225 ns  Iteration: 0  Instance: /stimulus
# Break in NamedBeginStat test_vectors at N:/PSUCourseProjects/ece351sp20/ripple_carry_adder_ex/hdl/ripple_cary_counter_ex.sv line 102

VSIM 5>
```

Project : rca_ex | Now: 225 ns Delta: 0 | sim:/stimulus/#INITIAL#97(test_vectors)

Using the QuestaSim GUI (cont'd)

Run the simulation



Questa Sim-64 2019.2

Instance	Design unit	Design unit type
stimulus	stimulus(fast)	Module
test_vectors	stimulus(fast)	Statement
DUT	ripple_carr…	Module
#ALWAYS#87	stimulus(fast)	Process
#INITIAL#97(t…	stimulus(fast)	Process
std	std	VlPackage
#vsim_capacity#		Capacity

Name	Value	Kind	
clk	1'h0	Regi…	Internal
reset	1'h0	Regi…	Internal
q	4'h2	Pack…	Internal

Name	Type (filtered)	State
#INITIAL#97(test…	Initial	Active
#ALWAYS#87	Always	Ready

…jects/ece351sp20/ripple_carry_adder_ex/hdl/ripple_cary_counter_ex.sv (/stimulus)

```
);
        // internal signals
        logic q0, q1, q2, q3;

        //instantiate 4 toggle flip-flops
        TFF tff0(q0,clk, reset);
        TFF tff1(q1,q0, reset);
        TFF tff2(q2,q1, reset);
        TFF tff3(q3,q2, reset);

        // assign the output
```

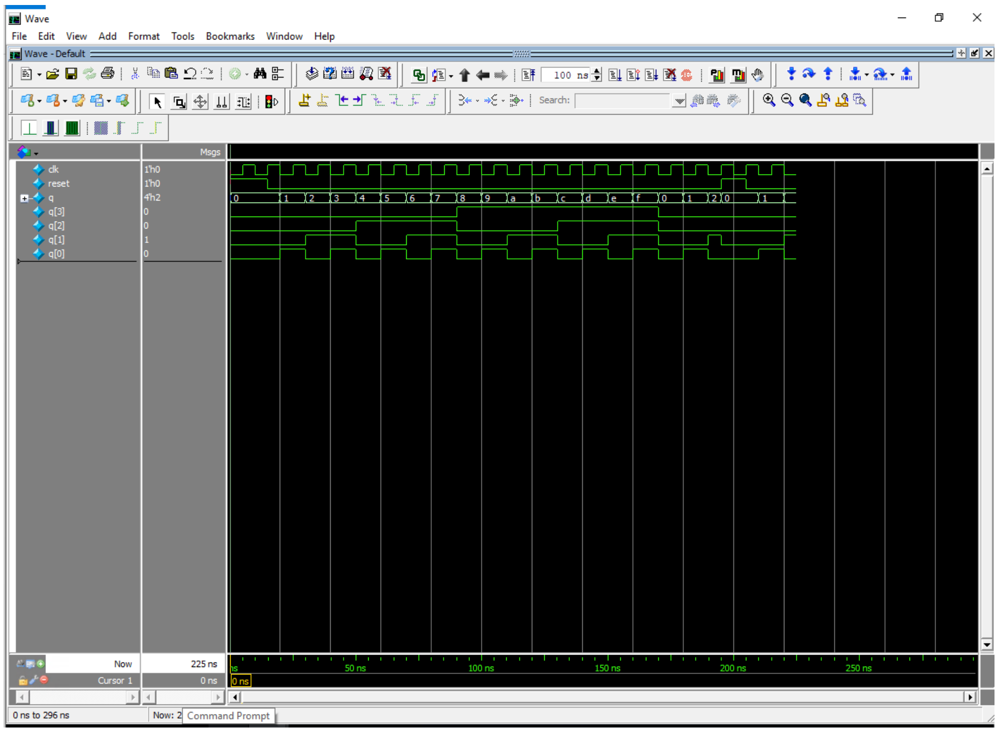
Transcript

```
VSIM 4> run -all
#                  0 Output q =  0
#                 20 Output q =  1
#                 30 Output q =  2
#                 40 Output q =  3
#                 50 Output q =  4
#                 60 Output q =  5
#                 70 Output q =  6
#                 80 Output q =  7
#                 90 Output q =  8
#                100 Output q =  9
#                110 Output q = 10
#                120 Output q = 11
#                130 Output q = 12
#                140 Output q = 13
#                150 Output q = 14
#                160 Output q = 15
#                170 Output q =  0
#                180 Output q =  1
#                190 Output q =  2
#                195 Output q =  0
#                210 Output q =  1
#                220 Output q =  2
# ** Note: $stop    : N:/PSUCourseProjects/ece351sp20/ripple_carry_adder_ex/hdl/ripple_cary_counter_ex.sv(102)
#    Time: 225 ns  Iteration: 0  Instance: /stimulus
# Break in NamedBeginStat test_vectors at N:/PSUCourseProjects/ece351sp20/ripple_carry_adder_ex/hdl/ripple_cary_counter_ex.sv line 102

VSIM 5>
```

Project : rca_ex | Now: 225 ns Delta: 0 | sim:/stimulus/#INITIAL#97(test_vectors)

Portland State UNIVERSITY

Using the QuestaSim GUI (cont'd)



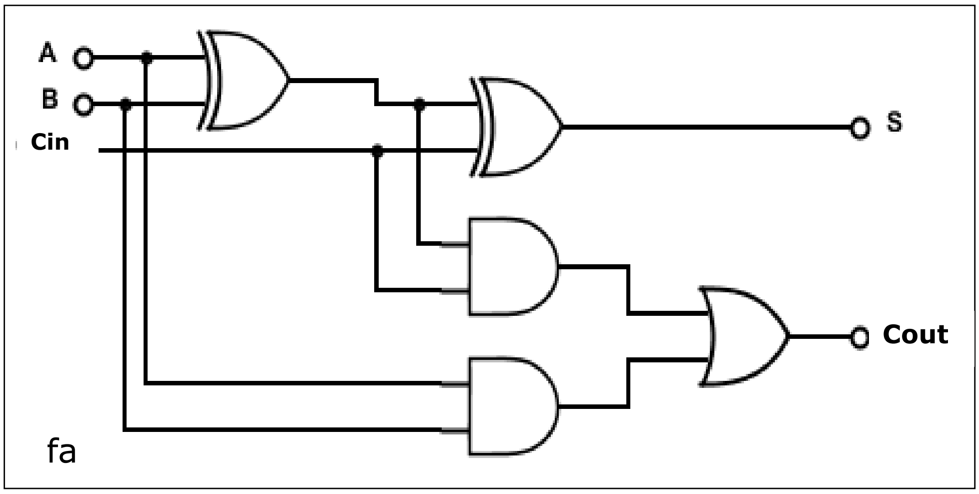
Portland State UNIVERSITY

Exercise 1 - Putting it all together
(implementing an 8-bit ripple-carry adder)

Steps:

- Create a gate-level module for a single-bit full adder
- Create a 4-bit ripple carry adder by instantiating and connecting 4 single-bit full adders
- *Create an 8-bit ripple carry adder DUT by instantiating and connecting two instances of the 4-bit ripple carry adder*
- *Simulate your design with the testbench I provided*
- *Submit your source code and a transcript showing your simulation results to your Exercise 1 dropbox on D2L*
    - *Due to D2L by 5:00 PM on Friday*

Single-bit full adder schematic

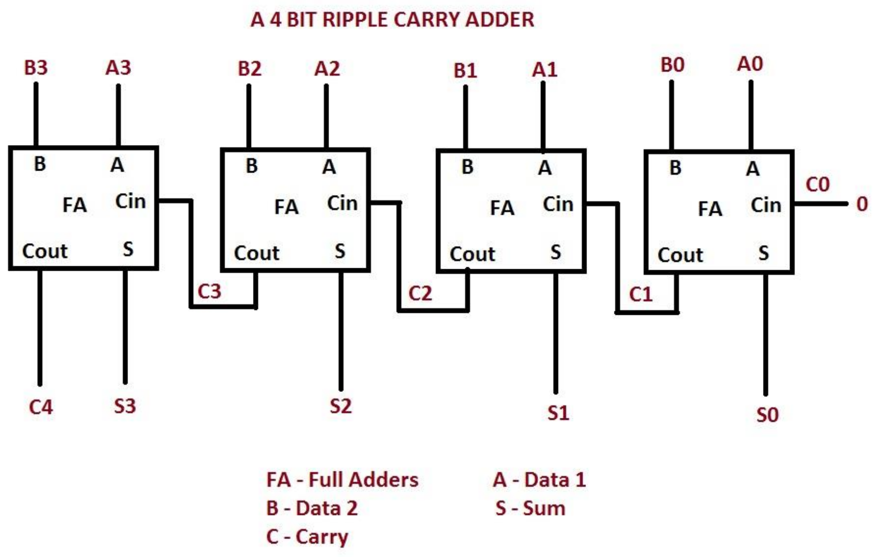


Write a SystemVerilog module (file: `fa.sv`) that implements this schematic using gate-level modeling

Single-bit full adder module starter code

```
/////////////////////////////////////////////
// fa.sv - Single bit full adder
//
// author:  <name> (<email address>)
/////////////////////////////////////////////
module FA (
  input logic A, B, Cin,     // a, b, and carry_in inputs
  output logic S, Cout       // sum and carry out outputs
);

// ADD YOUR CODE HERE

endmodule : FA
```

4-bit Ripple Carry Adder

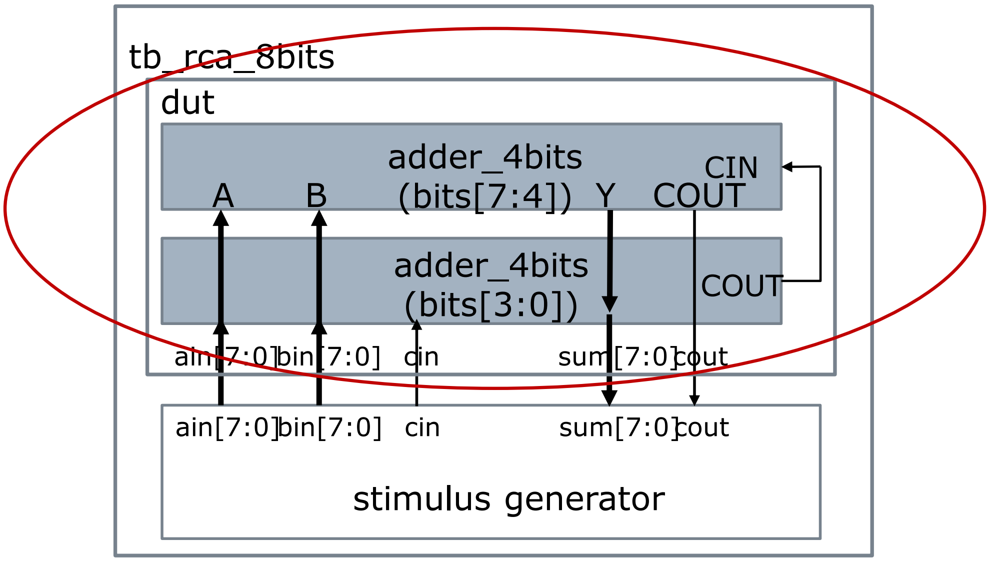


Write a SystemVerilog module (file: `adder_4bits.sv`) that implements this schematic by instantiating 4 instances of FA and connecting the inputs and outputs.

4-bit adder module starter code

```
//////////////////////////////////////////////
// adder_4bits.sv - 4 bit adder
//
// Author:  <name> (<email address>)
//////////////////////////////////////////////
module adder_4bits (
  input logic [3:0] A, B,
  input logic C0,
  output logic [3:0] S,
  output logic C4
);

// internal variables
wire C1, C2, C3;

// instantiate and connect four FA's
// ADD YOUR CODE HERE

endmodule: adder_4bits
```

Top level 8-bit ripple carry adder (the DUT)



Write a SystemVerilog module (file: dut.sv) that instantiates and connects to adder_4bits

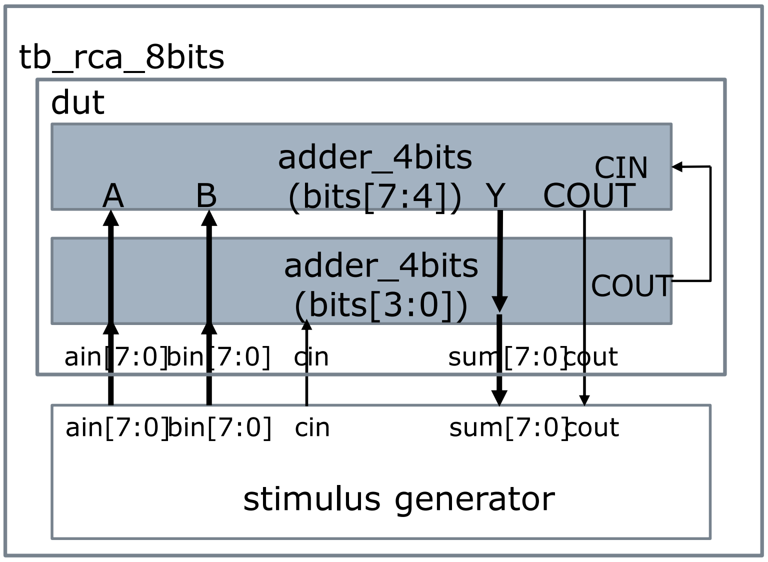
Portland State UNIVERSITY

Top level starter code

```
//////////////////////////////////////////////
// dut.sv - 8-bit Ripple Carry Adder
//
// Author:  <name> (<email address>)
//////////////////////////////////////////////
module dut (
  input logic [7:0] ain, bin,
  input logic cin,
  output logic [7:0] sum,
  output logic cout
);

timeunit 1ns/1ns;

// ADD YOUR CODE HERE

endmodule: dut
```

Portland State UNIVERSITY

Simulate your design



Build a project in QuestaSim and simulate your design. Submit your SystemVerilog source code and a transcript of your successful simulation to D2L by Noon on Friday

Portland State UNIVERSITY

Next Time

- Topics:
  - Exercise 1 solution
  - Packed and unpacked arrays
  - Parameters and constants
- You should:
  - Finish writing, simulate and submit your "in-class" exercise results to D2L (worth up to 20 pts)
    - Include the SystemVerilog code that you wrote and a transcript showing your simulation results.
    - Put all of the files and the transcript in a single .zip or rar file named <yourname>_exercise1 (ex: rkravitz_exercise1.zip)
    - Deadline is 5:00 PM on Friday 09-April
- Homework, projects and quizzes
  - Homework #1 will be assigned Tue, 13-Apr

Portland State UNIVERSITY