DIGITAL LOGIC
RTL & VERILOG
Interview Questions

A Practical Study Guide for Design Engineers

VERILOGCODE.COM

Ordering Information:
Quantity sales. Special discounts are available on quantity purchases by corporations, associations, and others. Orders by U.S. trade bookstores and wholesalers, please visit: www.VerilogCode.com

Printed in the United States of America

First Printing, May 2015

Revision 1.0

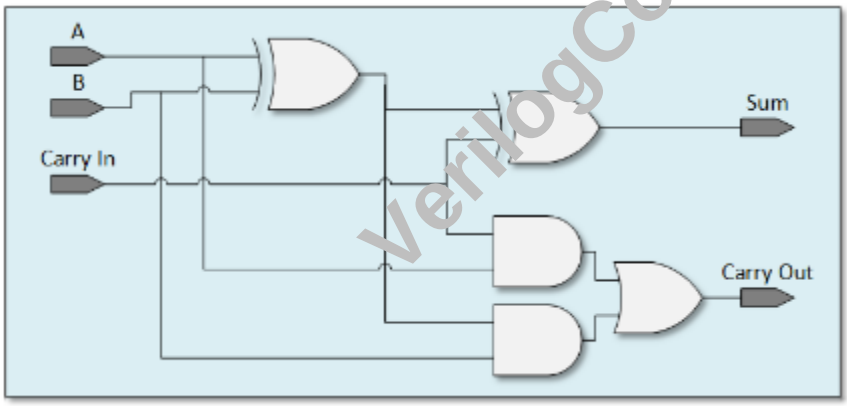
ISBN-13: 978-1512021462

ISBN-10: 1512021466

www.VerilogCode.com

For permission requests, contact VerilogCode.com

DIGITAL LOGIC
RTL & VERILOG
Interview Questions



A Practical Study Guide for Design Engineers

VERILOGCODE.COM

DIGITAL LOGIC

RTL & VERILOG

Interview Questions

About the Author:

Trey Johnson has been designing digital logic circuits and writing RTL code in both Verilog and VHDL languages for almost twenty years.

In the late 1990's, Johnson designed and developed some of the first multimedia hardware components used inside early smartphones, with his primary design focus in video and graphical subsystems, LCD and camera subsystems, and 2D hardware accelerators. He has worked closely and designed hardware components for both ARM and DSP processors and cache subsystems. He also has experience designing I/O peripherals such as resistive touch screen displays, magnetic card readers, PCI Express controllers and SerDes subsystems, and memory controllers.

Johnson has been granted three United States Patents for his digital design solutions. He is the founder of **VerilogCode.com** which is a website dedicated to sharing information about Verilog and RTL design.

Please visit the website for more digital design and job interview questions and to also share your own experiences.

This book is dedicated to Brandi, Tucker, Gunner and Alexa

Thank you for riding with me on life's waves of change,

...and for embracing the journey along the way.

Table of Contents

Introduction…… 17

RTL Verilog Syntax Questions ……..……. 19

RTL Logic Design Questions ……..…… 29

Clock Dividers, Clock Gating, and Reset Questions…. 49

Clock Domain Crossing Questions ……. 59

Power Related Design Questions ……... 65

Digital Logic Questions ...……. 71

Logical Thinking Questions ..……..... 93

Answers to Logical Questions……..... 99

Further Reading and Studying on Your Own…...…..... 103

Personal Interview Notes and Questions……..... 105

Credit and Sources……. 119

List of Questions

RTL Verilog Syntax Questions

1. Explain *blocking* versus *non-blocking* statements
2. Show Verilog code for *bitwise* versus *conditional* operators
3. Verilog code for logic gates: *and, or, nand, nor, xor, xnor*
4. Verilog code for bitwise reduction
5. Verilog code multiplying and dividing by powers of 2
6. Verilog code for sign extension and concatenation
7. Write Verilog Code for *asynchronous* and *synchronous* Flip Flops
8. Verilog coding - what are three ways to code a mux
9. What type of circuit would the synthesis tool create for mux code
10. Verilog code for latch versus flip flop and draw timing diagram

RTL Logic Design Questions

11. Design a circuit to detect if a signal transitions in any direction
12. Design a circuit to detect a 1 cycle high pulse (synchronously)
13. Design a sequence detector circuit to detect 1,0,1,1,0 using FSM
14. Verilog code to detect a pattern 10110 anywhere in last 8 samples
15. For the timing diagram show, write Verilog code to create it
16. Design a debounce circuit to remove input glitches
17. Write Verilog code to convert BCD to gray code
18. Design a synchronous fifo module using dual port RAM
19. Design a circuit to detect if number is divisible by three
20. Design a circuit to calculate Fibonacci sequence
21. Design a circuit to find the maximum and second highest number
22. Design a circuit to output second, minute, and hour from 1ms input
23. Given the timing diagram, write the equivalent Verilog code
24. Draw the structure of a digital FIR filter with 5-taps

Clock Dividers, Clock Gating, and Reset Questions

25. Design a clock divide by 2 circuit
26. Design a clock divide by 3 circuit (with 50 percent duty cycle)
27. Design a clock divider by *N* circuit (with 50 percent duty cycle)
28. Design a glitch free clock gating cell with enable pin
29. How to detect a rising edge of an input signal if clocks are off
30. Design a reset circuit with async assertion and sync deassertion

Clock Domain Crossing Questions

31. What is metastability?
32. Design a circuit from a slow clock domain to a fast clock domain
33. Design a circuit to handle CDC from fast domain to a slow domain
34. How would the circuit change if you need to synchronize a bus?
35. Gray coding techniques to cross clock domains

Power Related Questions

36. Describe two components of power
37. Describe how to reduce static power
38. Describe how to reduce Dynamic power
39. Describe low power RTL coding techniques

Refresher: Digital Logic Questions

40. What is definition of setup and hold time for a flip flop
41. Venn Diagram and Boolean Logic
42. Logic Gate Design - Transistor Level
43. Cross Section of a transistor – understand the process node
44. Karnaugh Maps (clk divide by 3 circuit not 50-50 duty cycle)
45. Half Adder using XOR gates for addition and AND gate for carry
46. Using only 2 input muxes, create a *nand*, *nor*, *inverter*, *or*, *and*, *xor*
47. How to use and XOR gate like a controlled inverter?
48. Design an *inverter*, *and* , *or*, and *xor* gate using just nand gates
49. Create 4:1 mux using 2:1 mux
50. What is the fastest frequency this circuit can run?
51. Convert this decimal value to binary, hex, and octal format

Logical Thinking Questions

52. Four Gallons of Water
53. The Path to Freedom
54. Three Light Switches
55. Multiplication Question
56. Einstein's Riddle

List of Figures

1. Blocking Statements and Equivalent Gates *(Fig. 1)*
2. Non-blocking Statements and Equivalent Gates *(Fig. 2)*
3. Timing Diagram - capture with latch and flip flop *(Fig.3)*
4. Latch vs. Flip Flop timing diagram *(Fig. 4)*
5. Equivalent Gates from RTL code in Ex. 11 (Fig. 5)
6. Timing Diagram for Edge Detection using clocks *(Fig. 6)*
7. Sequence Detector using FSM *(Fig. 7)*
8. Decoder Circuit - Shift Register and Combinatorial Logic *(Fig. 8)*
9. Write the Verilog code to produce the above waveform *(Fig. 9)*
10. Classic Synchronizer Circuit using 2 Flip Flops *(Fig. 10)*
11. Circuit to detect low to high transition *(Fig. 11)*
12. BCD to Gray Code *(Fig. 12)*
13. Circuit to Generate Fibonacci Series *(Fig. 13)*
14. Write Verilog Code to Produce this Timing Diagram *(Fig. 14)*
15. 5-TAP Digital FIR filter *(Fig. 15)*
16. Timing Diagram for Clock Divide by 3 *(Fig. 16)*
17. Timing Diagram for Clock Divide by 3 (50-50 duty cycle) *(Fig.17)*
18. Timing Diagram for Clock Divide by 4 (50-50 duty cycle) *(Fig.18)*
19. Timing Diagram for Clock Divide by 5 (50-50 duty cycle) *(Fig.19)*
20. Latched Based Clock Gating Circuit *(Fig. 20)*
21. Classic 2-Stage Synchronizer *(Fig. 21)*
22. Clock Domain Crossing using Full Handshaking *(Fig. 22)*
23. Synchronize Bus Across Clock Domains Full Handshake *(Fig. 23)*
24. Example of Critical Path in Sequential Circuit *(Fig. 24)*
25. Venn Diagram *(Fig. 25)*
26. Logic Gates for A&(B|C) *(Fig. 26)*
27. Cross-section of CMOS Transistor *(Fig. 27)*
28. 3 State FSM for Clock Divider *(Fig. 28)*
29. Gray coding (with violation from state 11 to state 00) *(Fig. 29)*
30. 2-bit FSM (illegal gray code transition) *(Fig. 30)*
31. Six state FSM with correct gray code values *(Fig. 31)*
32. Equivalent Gates Result and Carry *(Fig. 32)*
33. Full Adder Circuit *(Fig. 33)*
34. Multibit Adder *(Fig. 34)*
35. Inverter *(Fig. 35)*
36. AND Gate Implemented with NAND Gates *(Fig. 36)*
37. OR Gate Implemented with NAND Gates *(Fig. 37)*

38. XOR Gate Implemented with NAND Gates *(Fig. 38)*
39. 4:1 Mux created with 2:1 muxes *(Fig. 39)*
40. 4:1 mux implemented with combinatorial logic *(Fig. 40)*
41. What is maximum frequency this circuit can run? *(Fig. 41)*

List of Verilog Coding Examples

1. Operator: & versus && *(Ex. 1)*
2. Operator: | versus || *(Ex.2)*
3. Operator: ~ versus ! *(Ex. 3)*
4. Bitwise Operators *(Ex. 4)*
5. Bitwise Reduction *(Ex. 5)*
6. Shift Operations *(Ex. 6)*
7. Concatenation Operations *(Ex. 7)*
8. Verilog coding styles for a 4:1 mux *(Ex. 8)*
9. Priority Encoder *(Ex. 9)*
10. Parallel Muxing Scheme using full case statement *(Ex. 10)*
11. Verilog code for Latch and Flip Flop *(Ex. 11)*
12. Verilog Code for Edge Detection *(Ex. 12)*
13. Circuit to detect pulse *(Ex. 13)*
14. Example Verilog Code for *Fig. 9* waveform *(Ex. 14)*
15. Verilog Code for 3-bit Gray Code *(Ex. 15)*
16. Verilog code to convert BCD to Gray Code *(Ex. 16)*
17. FIFO control logic *(Ex. 17)*
18. FSM states for divisible by 3 circuit *(Ex. 18)*
19. Verilog code for Fibonacci Generation *(Ex. 19)*
20. Verilog code to generate one second, minute, and hour *(Ex. 20)*
21. Verilog Code for *Fig. 14* Timing Diagram *(Ex. 21)*
22. Verilog Code for Clock Divide by 2 *(Ex. 22)*
23. Verilog Code for Clock Divide by 3 with 50-50 duty cycle *(Ex.23)*
24. Example Verilog Code for Generic Clock Divide by N *(Ex. 24)*
25. Example Verilog Code for Glitch Free Clock Gate *(Ex. 25)*
26. Asynchronous Edge Detection Circuit with no clocks *(Ex. 26)*
27. Verilog code for Reset synchronization *(Ex. 27)*
28. Start/Stop Conditions for Counters *(Ex. 28)*

List of Tables

1. Asynchronous versus Synchronous Flip Flops *(Table 1)*
2. Inverter Gate, Truth Table, and transistor-level circuit *(Table 2)*
3. NAND Gate, Truth Table, and transistor-level circuit *(Table 3)*
4. NOR Gate, Truth Table, and transistor-level circuit *(Table 4)*
5. AND Gate, Truth Table, and transistor-level circuit *(Table 5)*
6. OR Gate, Truth Table, and transistor-level circuit *(Table 6)*
7. Semiconductor Manufacturing Processing Nodes *(Table 7)*
8. Truth Table for an Adder *(Table 8)*
9. Inverter Implemented with 2:1 mux *(Table 9)*
10. AND Gate Implemented with 2:1 mux *(Table 10)*
11. OR Gate Implemented with 2:1 mux *(Table 11)*
12. NAND Gate Implemented with 2:1 mux and inverter *(Table 12)*
13. NOR Gate Implemented with 2:1 mux and inverter *(Table 13)*
14. XOR Gate Implemented with 2:1 mux and inverter *(Table 14)*
15. XNOR Gate Implemented with 2:1 mux and inverter *(Table 15)*
16. XOR Gate Used as Controlled Inverter *(Table 16)*
17. Path to Freedom Doors *(Table 17)*
18. Multiply Question **(***Table 18)*

Introduction

I'm 30,000 feet above the ground, on a plane headed to San Diego for a job interview. I'm a little anxious, and I take some comfort in looking out of the small window pane next to me. My eyes wander back down to the magazine article sitting on my lap, then I read these words:

"***Transitions. That's all life is, and it's tougher than physics. From school to work to retirement to dead***".

The magazine article is about children who are interviewing for preschool, and the most important characteristic that the administrators look for is how well can the child *adapt* to change with new surroundings and new rules. But this article could have been written about me: an engineer who spent eighteen years working for the same company, and one day was suddenly let go as a result of a corporate downsizing. I am now the one who must adapt to change, new surroundings, and new rules.

This book documents real interview questions that I encountered from my own personal job interviewing experiences with some of the top-tier semiconductor companies in the world. This book also contains fundamental digital design material and practical Verilog code examples that I created based on the themes from the types of questions that I experienced first hand. This book will help prepare you for your own interviewing process. It is by no means the end-all, but rather, consider this book as a great starting point. As you read through some of the questions, I will also share with you some of my personal insight and knowledge in the *Author's Tips* section, which I have acquired through my career as a digital logic professional.

Interviewing for a job is like going on a date; at first you may feel a little nervous or awkward, but after some time and more interviews you soon become more comfortable and confident. Do not get discouraged in

the beginning! The job interviewing experience can be daunting. It will test your mental toughness. I experienced headaches during my first few interviews because of the long hours of mental stress. But the saying is true that *practice makes perfect*. After several more interview attempts, I became more comfortable, developed a sense of calmness, and felt more prepared to answer the questions.

I've encountered many different types of interview questions ranging from real world practical examples, to academic textbook or theoretical questions (usually asked by people with PHDs with not much practical experience), to tricky questions using some obscure circuit (which would never be applicable in the real world), to behavioral questions (usually asked by Human Resource representatives). This book focuses on real world practical examples and it also discusses some of the tricky and obscure questions that are asked. Preparing for behavioral questions is important and is covered on our website.

This book is divided into multiple sections covering the following topics: RTL Verilog coding syntax, RTL Logic Design (including low power RTL design principles), clocking and reset circuits, clock domain crossing questions, digital design fundamentals, and logical thinking questions. Each section is unrelated to the other so you can jump around to any section or question that interests you.

This book is a great starting place for you to begin preparing for your job interview. This book provides you with a broad range of information and covers many topics. By the end of this book, you will have more knowledge and insight into the types of digital design interview questions being asked in the field of semiconductor digital design.

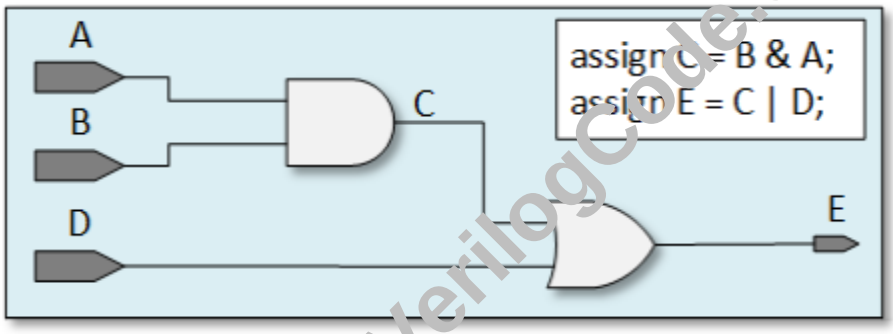
Remember that life will always bring about change, and it's how well you can transition and adapt that is important. Have a strong and positive attitude and you will succeed!

Good luck on your new journey!

RTL Verilog Syntax Questions

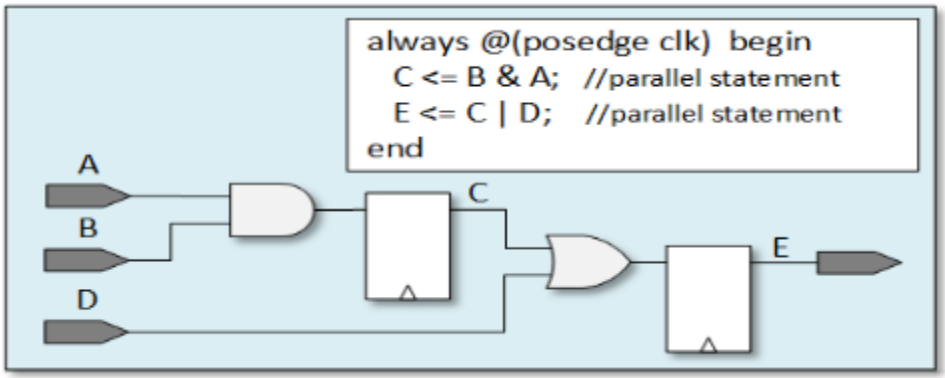
1. What is the difference between blocking and non-block statements, and when are they used?

Blocking statements are coded in Verilog with the = operator, and are used when creating combinatorial logic. This operator *blocks* the simulator from executing subsequent statements until the current evaluation and assignment is done. Consider the following code (E is assigned the immediate new value of C):



Blocking Statements and Equivalent Gates *(Fig. 1)*

Non-blocking statements are coded in Verilog with <= operator, are always used when coding flip flops inside a clocked process. The assignment is postponed until all the subsequent statements are evaluated. This allows for parallel or concurrent execution of statements. In this example, E is assigned the previous value of C (not the immediate):



Non-blocking Statements and Equivalent Gates *(Fig. 2)*

2. Explain the difference between logical and bitwise operators

In Verilog, you should understand the different syntax used for writing conditional if statement operations compared to writing code for creating logical gates with bitwise operations.

For bitwise operations resulting in an *AND* gate, use the & operator
assign C = A & B; //This will create an *and* gate

For conditional *and* if statements, use the && operator
if (cond1 == 1'b1 && cond2 == 1'b1) {...}

Operator: & versus && *(Ex. 1)*

For bitwise operations resulting in an *OR* gate, use the | operator
assign C = A | B; //This will create an *or* gate

For conditional *or* if statements , use the || operator
if (cond1 == 1'b1 || cond2 == 1'b1) {...}

Operator: | versus || *(Ex.2)*

For bitwise operations resulting in *inverter* gate, use the ~ operator
assign A = ~B; //This will create an *inverter*

For conditional *not* if statements, use the !
if (!cond1) { … }

Operator: ~ versus ! *(Ex. 3)*

3. Write code for logic gates: *and, or, xor, nand, nor, xnor*

```
assign C =   (A & B);    //AND gate
assign C =   (A  | B);    //OR gate
assign C =   (A ^  B);   //XOR gate
assign C = ~(A & B);   //NAND gate
assign C = ~(A  | B);   //NOR gate
assign C = ~(A ^  B);   //XNOR gate
```

Bitwise Operators *(Ex. 4)*

4. How can you bitwise reduce a multi bit signal?

```
wire [15:0] databus;
wire all_ones_detected;
wire is_databus_odd;
wire signal_not_zero;

assign all_ones_detected = &databus; // AND all the bits together;
assign is_databus_odd    = ^databus;  // XOR all the bits together;
assign signal_not_zero   = |databus; //   OR all the bits together;
```

Bitwise Reduction *(Ex. 5)*

5. What code multiplies or divides by powers of 2?

```
assign C = A << 2;  // shift left ( multiply by 4)
assign C = A >> 3;  // shift right (division by 8)
```

Shift Operations *(Ex. 6)*

6. How would you perform sign extension?

```
reg [4:0] A;
wire [9:0] C;
assign C = { {5{A[4]}} ,A};  // sign extension
```

Concatenation Operations *(Ex. 7)*

7. What are three ways to code a 4:1 mux in Verilog?

Assume the 4:1 mux inputs are named 'A, B, C, D', you will need a 2-bit *select* line (to choose between 4 inputs) and have a 1-bit *output*:

Coding Style 1:

```
assign output = (select == 2'b00 ) ? A :
                (select == 2'b01) ? B :
                (select ==2'b10) ? C : D;
```

Coding Style 2:

```
always @(*)
if (select ==  2'b00)
    output <= A;
else if (select == 2'b01)
    output <= B;
else if (select == 2'b10)
    output <=C;
else
    output <= D;
```

Coding Style 3:

```
always @(*)
case (select):
  2'b00: output  <= A;
  2'b01: output  <= B;
  2'b10: output  <= C;
  2'b11: output  <= D;
end case;
```

Verilog coding styles for a 4:1 mux *(Ex. 8)*

You could also instantiate a mux cell directly if it exists in your library.

8. What circuit would synthesis create for previous mux coding styles?

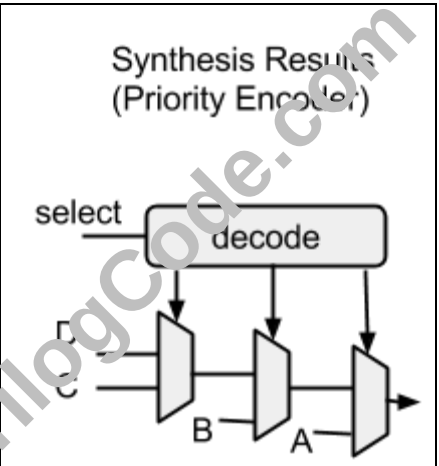
Coding Styles 1 and 2 are equivalent circuits but use different coding styles, and both produce a priority encoder (path A is highest)

Style 1:

```
assign output = (select == 2'b00) ? A :
                (select == 2'b01) ? B :
                (select ==2'b10) ? C :
                D;
```

Style 2:

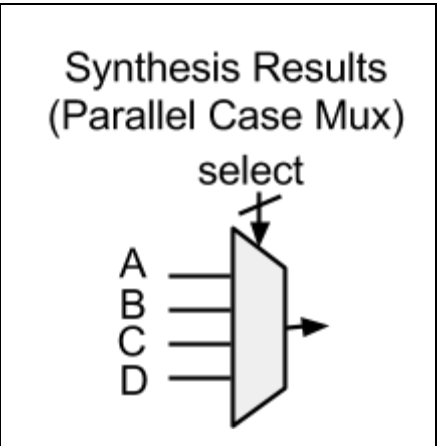
```
always @(*)
if (select == 2'b00)
    output <= A;
else if (select == 2'b01)
    output <= B;
else if (select == 2'b10)
    output <=C;
else
    output <= D;
```



Priority Encoder *(Ex. 9)*

Style 3

```
always @(*)
case (select):
  2'b00: output  <= A;
  2'b01: output  <= B;
  2'b10: output  <= C;
  2'b11: output  <= D;
end case;
```

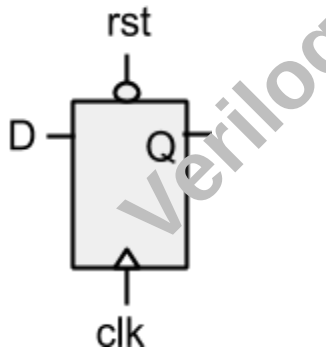
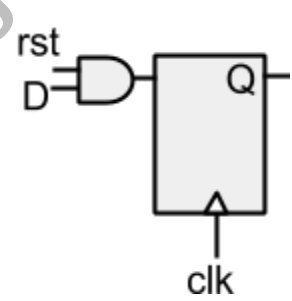


Parallel Muxing Scheme using full case statement *(Ex. 10)*

9. Write Code for Asynchronous/Synchronous Flip Flops and discuss the pros and cons of each

	Asynchronous	Synchronous

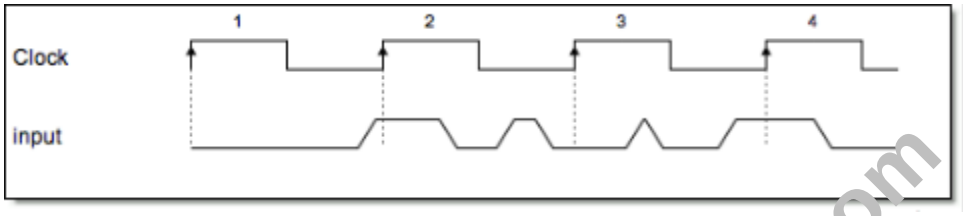
Verilog Code	always @(posedge clk or negedge rst)
	If (!rst)
	Q <= #1 1'b0;
	else
	Q <= #1 D;
		always @(posedge clk)
		If (!rst)
		Q <= #1 1'b0;
		else
		Q <= #1 D;

Schematic		
Pros	No need for clocks to be running	No need to worry about asynchronous timing
Cons	Any glitch on reset signal will reset flops (If there are cross clock domain paths, then you need to synchronize the reset to each clock domain). Need to time the removal of reset	Need to time both the assertion of reset and also deassertion of reset

Asynchronous versus Synchronous Flip Flops *(Table 1)*

Author's Tip: The questions so far has focused on Verilog syntax. Basic syntax questions can easily be looked up on Google (after you have a job), however, on a job interview it's important to be prepared for these questions and knock them out of the park.

10. Write Verilog to capture input below with a latch and a flip flop, and draw the timing outputs of each.

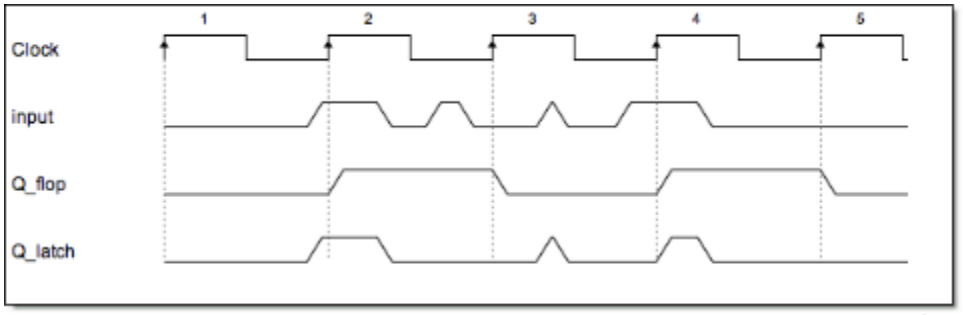


Timing Diagram - capture input into a latch and flip flop *(Fig. 3)*

```
//purposely coding latch
always @(clk or input)
  if (clk == 1'b1)
    Q_latch <= input;

//flip flop coding a Latch
always @(posedge clk)
  Q_flop <= #1 input;
```


Verilog code for Latch and Flip Flop *(Ex. 11)*

The outputs of the latch and flip flop is below in the timing diagram:



Latch vs. Flip Flop timing diagram *(Fig. 4)*

The output of the flip flop only changes on the rising edge of clock, and will equal the value of the input captured at the rising edge of clock. The latch output will follow the input signal while the latch is open (in this case while clock is high). When the latch closes (clock is low), the output holds its previous value.

RTL Logic Design Questions

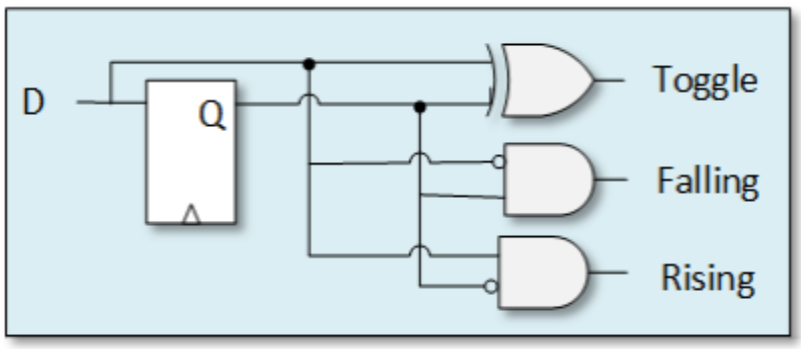
11. Design a circuit that can detect if an input signal transitions in either direction. Draw a timing diagram.

For this answer let's assume there is a clock available, and also the input signal D is on the same clock domain as our circuit. As with all interview questions, you should state your assumptions before answering the questions or confirm with the interviewer so you are both on the same page.

The easiest way to detect if an input signal has changed it simply compare current signal at time T to the previous version of the signal at time t-1. Therefore, a flip flop is used to capture the signal, and then you can compare it to the previous version of the same signal:

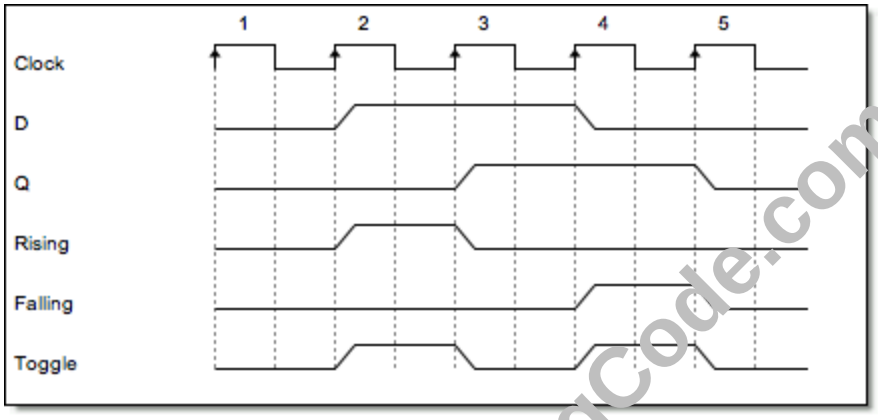

```
//capture the signal to have a delayed version
always @ ( posedge clk or negedge reset)
if (~reset)
  Q <= #1 1'b0;
 else
  Q <= #1 D;

//detect signal transitions
assign Toggle = (Q ^ D) ;  //xor gate
assign Falling = ~D & Q;  / /D is low, but was high
assign Rising  =  D & ~Q; // D is high, but was low
```

Verilog Code for Edge Detection *(Ex. 12)*



Equivalent Gates from RTL code in Ex. 11 *(Fig. 5)*

The timing diagram is shown below. This waveform was drawn with tool called TimeGen, downloaded from XFusionSoftware.com:



Timing Diagram for Edge Detection using clocks *(Fig. 6)*

12. Design a circuit to detect a 1 cycle pulse input

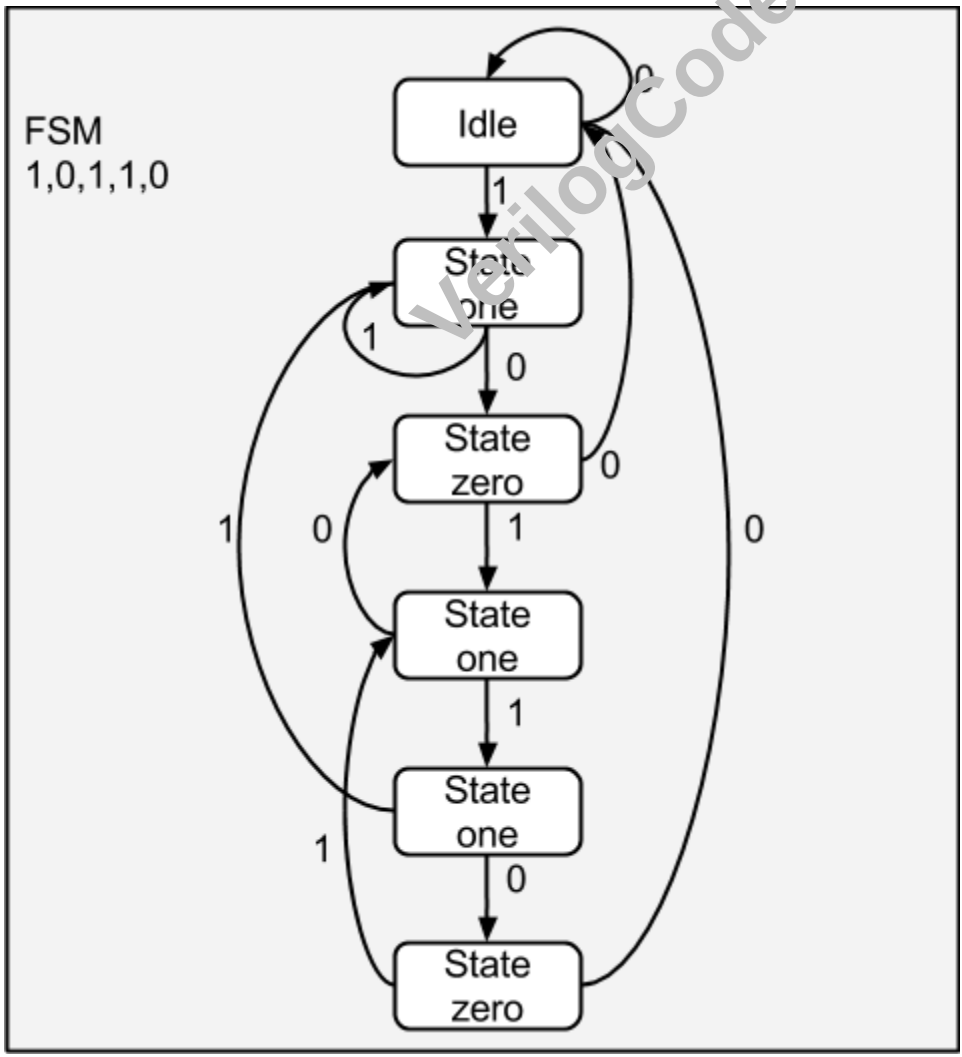
Using delay states to remember the state of the signal for previous two cycles, you can simple check for 0, 1, then 0. You could also use an FSM.

```
//capture the signal and delay 2 cycles

always @ ( posedge clk or negedge reset)
if (~reset) begin
  Q1 <= #1 1'b0;
  Q2 <= #1 1'b0;
end
  else begin
   Q1 <= #1 D;   //delay T-1
   Q2 <= #1 Q1; //delay T-2
   end;

assign pulse_high= ~D & Q1 & ~Q2;
```

Circuit to detect pulse *(Ex. 13)*

13. Design a sequence detector for the pattern: 10110

One method is to design a FSM to detect the sequence. If the FSM reaches state 5, then generate a valid output pulse (all the other states will output low). It's also important to make sure you go back to correct state if the wrong value comes in (you don't always need to go back to idle state). You need to check if some of the sequence has already started and go back to the appropriate state.



Sequence Detector using FSM *(Fig. 7)*

This is the end of the free preview PDF.

If you would like to see all 56 questions, with 41 figures and drawings, and 28 Verilog examples, then please purchase the PDF from our website:

VerilogCode.com