Digital/FPGA/ASIC Hardware Design Interview Guide

Justin Mikhail
University of Alberta, 2025
BSc. Computer Engineering, Spec. in Nanoscale System Design

Purpose

This guide has been created to help you prepare for digital hardware interviews (ASIC/FPGA) at a level expected from a recent graduate of a Computer Engineering undergraduate program. It covers combinational/sequential logic, HDL (Verilog/SystemVerilog), microarchitecture, timing analysis, and more. Interview rounds, especially for full-time positions at larger companies, may cover each or all of these topics, depending on the role. From my experience, expect earlier rounds to be a technical focus with verbal explanation and coding exercises, and more design whiteboarding and resume questions as the rounds go on.

Disclaimer: Some questions adapted from Harris & Harris [1]. This resource is purely for educational and learning purposes.

1 Preparation Insights

- **Know Your Resume:** Be prepared to discuss everything listed such as projects, tools, and responsibilities, at both high and detailed levels, including drawing a block diagram of your project while talking through it.
- **Cross-Functional Readiness:** Expect comprehensive questions that may span verification, design, and software/firmware interaction. Interviewers often assess system-level understanding beyond RTL and understanding of good design practices.
- **Problem Solving Focus:** Interviewers value your approach and reasoning more than just getting the right answer. Communicate your thought process clearly and consider trade-offs. Practice explaining your thought process during problems, and don't be afraid to ask for a moment to think or ask to clarify assumptions. This demonstrates critical thinking and ensures you fully understand the problem before solving it.
- **Master the Fundamentals:** Ensure a strong understanding of core digital design concepts such as pipelining, FSMs, combinational and sequential logic, and memory addressing. Often an interviewer will ask you to explain a concept before you do a design exercise with it.
- **PPA and Timing** Be ready to discuss Power-Performance-Area (PPA) tradeoffs, Clock Domain Crossing (CDC), timing, and low-power strategies. Be well-versed in setup and hold times, clock skew, on-chip variations, power consumption, metastability, and timing paths. Once again, relate thought processes and design choices to Power, Performance, and Area.
- **RTL Design Topics:** Practice implementing sequence detectors, understand the difference between Mealy and Moore machines, and be able to write clear, synthesizable RTL. Know that an RTL testbench is just as important as the module itself. Understanding the simulation tool and reading waveforms is essential.
- **Leverage Your Experience:** Be able to explain your past roles, design challenges, and key decisions. Reflect on what you learned and how you grew from those experiences. Show humility and a willingness to learn from mistakes. Do your best to be charismatic and personable, as interviewers are also evaluating if they want to work with you.
- **Don't Rely on AI:** Study for interviews as you would for an exam, as you will not have access to AI tools during the interview. While AI can be a helpful tool for learning, it should not replace your own understanding and problem-solving skills.

2 Digital Design Fundamentals

2.1 Combinational Logic

1. What is DeMorgan's theorem?
2. How can you build an OR gate using only NAND gates? Show the Boolean algebra and the circuit diagram.
3. Draw a two-input XOR gate using only NAND gates. What is the minimum number of gates required?
4. What is the difference between a combinational and a sequential circuit?
5. What is a tristate buffer? How and why is it used?
6. Are NAND gates universal? Why?
7. What is the boolean equation of a multiplexer?
8. Explain the concept of fan-out in combinational circuits. How does it impact circuit design?
9. What is the significance of propagation delay in combinational circuits? How can it be minimized?
10. Design a circuit that determines whether a month has 31 days based on a 4-bit input representing the month (1-12). Use a truth table and K-map.
11. Design a priority encoder for a 4-bit input. Explain its functionality.

2.2 Sequential Logic

1. What is setup time and hold time?
2. What determines the maximum clock frequency? How could you change a circuit if the clock frequency needed to be increased?
3. Explain the difference between a Mealy and a Moore state machine. Which has more states? Which may be more prone to glitches?
4. What is pipelining, and why is it useful in digital systems?
5. Design a D flip-flop using a JK flip-flop.
6. Design a 2-bit counter using JK flip-flops.
7. Create an FSM that detects the input sequence `10X1`. (X = Don't Care)
8. Design an edge detector circuit that outputs HIGH on a rising edge (0 to 1 transition).
9. What is metastability in flip-flops, and how can it be mitigated?
10. Explain the concept of clock domain crossing and why it is important. What are 3 common methods to handle it?
11. Draw a state transition diagram for a `10110` sequence detector, with and without detection overlap.

3 Verilog / SystemVerilog

3.1 Language Basics

1. How does Verilog differ from VHDL?
2. What is the difference between synchronous and asynchronous resets? Provide Verilog examples and discuss their advantages and disadvantages in terms of synthesis and timing.
3. When should `always_comb` be used instead of `always_ff` in SystemVerilog?
4. Compare blocking (`=`) and non-blocking (`<=`) assignments in Verilog. When should each be used?
5. Design an FSM that detects the sequence "011" and write the Verilog code. Draw the state (transition) diagram before writing any code. Bonus: Compare both Mealy and Moore module implementations.
6. How do you swap two variables in Verilog with and without a temporary register? Provide a Verilog example for each.
7. How do you define parameterized modules in Verilog?
8. What is the purpose of the `generate` statement in SystemVerilog?
9. How does `if-else` differ from `case` statements in SystemVerilog? When would you use one over the other? How may they differ in terms of synthesis?
10. What is the purpose of the `initial` block in Verilog? How does it differ from the `always` block?
11. Describe the differences between `case`, `casez`, and `casex`.
12. Below are two Verilog snippets. Identify the issues in each causing a latch to be inferred.

```
// ---------------- Example A ----------------
// Q: Which of these will infer a latch?
// A: Because there is no default: the case-statement does not cover {s1,s0}=2'b11,
//    so 'out' must hold its previous value in that condition -> latch inferred.

always @(s1 or s0 or i0 or i1 or i2 or i3) begin
  case ({s1, s0})
    2'd0: out = i0;
    2'd1: out = i1;
    2'd2: out = i2;
    // <---- Missing default: if {s1,s0} == 2'b11, 'out' is never assigned here.
  endcase
end


// ---------------- Example B ----------------
// Q: Which of these infers a latch?
// A: 'z' is used on the RHS but not in the sensitivity list.
//    On simulation, out only updates when x or y changes, so if z toggles, 'out' holds
//    its old value -> latch inferred. (If you wrote always @(*), this would be fully combinational.)

always @(x or y) begin
  out = x & y & z;  // 'z' is missing from the sensitivity list -> latch inferred
end
```

13. Given the code and delays below, what are the logic values of the signals at t = 5?

```
module tb;
  reg a, b, c, q;

  initial begin
    $monitor("[%0t] a=%0b b=%0b c=%0b q=%0b", $time, a, b, c, q);

    // Initialize all signals to 0 at time 0
    a <= 0;
    b <= 0;
    c <= 0;
    q <= 0;

    // Inter-assignment delay: Wait for #5 time units
    // and then assign a and c to 1. Note that 'a' and 'c'
    // get updated at the end of the current time step
    #5 a <= 1;
       c <= 1;

    // Intra-assignment delay: First execute the statement
    // then wait for 5 time units and then assign the evaluated
    // value to q
    q <= #5 a & b | c;

    #20;
  end
endmodule
```

14. What is a glitch in digital circuits, and how would you eliminate the glitch shown in this waveform by changing the code?

```
module count(
input clk,
output reg equal3,
output reg [1:0] counter);

always @(posedge clk) begin
  counter <= counter + 1;
end

always @* begin
  if (counter == 3)
    equal3 = 1;
  else
    equal3 = 0;
end
endmodule
```

```
`timescale 1 ns/1 ns
module clktb();
  reg clk_s;
  wire [1:0] counter_s;
  wire equal3_s;

  count
    t1(.clk(clk_s), .counter(counter_s), .equal3(equal3_s));

  always begin
    clk_s <= 0; #10;
    clk_s <= 1; #10;
```

```
  end
endmodule
```

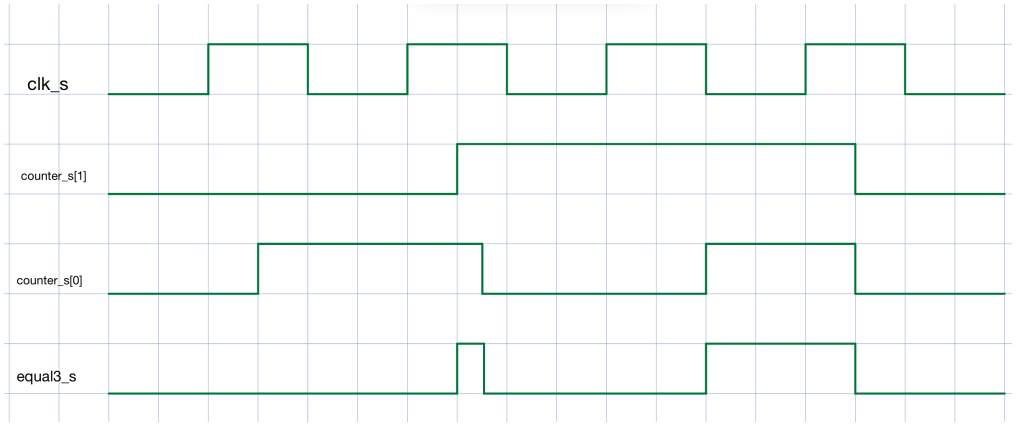


Figure 1: Glitch Waveform Example.

15. Write a Verilog module for the following circuit given in the diagram by first writing a module for a D flip-flop and using 3 instances of it in the design. Bonus: Write a testbench for the module.

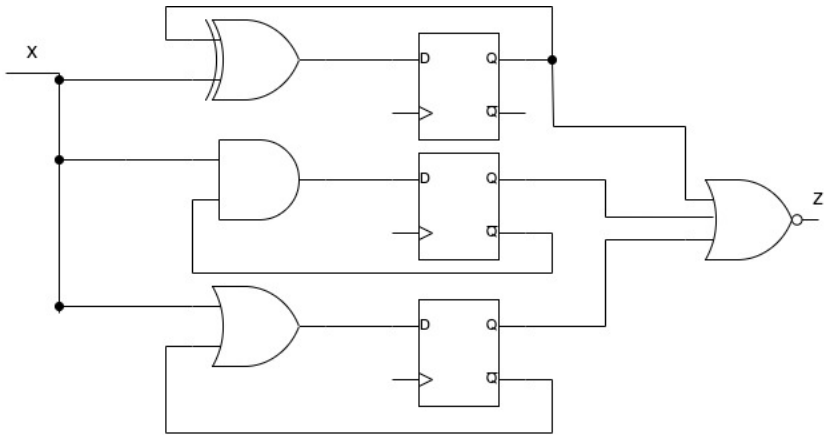


Figure 2: Circuit Diagram for Verilog Module.

16. Write a Verilog module to infer a Block RAM (BRAM) in synthesis with a read and a write (R/W) port at different R/W clock frequencies. Include a testbench that demonstrates reading and writing to the BRAM. Bonus: Parameterize the BRAM depth and width.

17. Create a 4-bit ripple-carry full adder in Verilog. Write a testbench that tests the adder with various inputs and checks the outputs. Include assertions to verify correctness. Bonus: Use a `generate` statement to instantiate a full adder for each bit.

3.2 Testbenches & Verification

1. What is the goal of a good testbench in RTL?
2. What is the impact of delay statements on simulation vs. synthesis?
3. What is a time step in a simulation tool?
4. What is the difference between inter-delay and intra-delay statements in Verilog?
5. How do you write a testbench for a module that includes clock and reset signals? Provide a simple example.

6. What is the purpose of assertions in SystemVerilog, and how do you use them in testbenches?
7. For a given Verilog exercise in the previous section, write a comprehensive testbench with a clock, reset, and randomized stimulus. Verify the output using assertions and visual inspection of the waveform.

4 Signal Processing & DSP

1. What is sampling in the context of digital signal processing?
2. What is the Nyquist theorem, and how does it relate to sampling?
3. How is an analog signal converted to digital?
4. Explain the affects of aliasing in both time and frequency domains.
5. What is a Fourier Transform, and how is it used in signal processing? Explain the meaning of a Fourier Transform in both time and frequency domains.
6. What is a Fast Fourier Transform (FFT), and how does it differ from a Discrete Fourier Transform (DFT)?
7. How many bits are needed to represent a signal with N distinct values?
8. Compare FIR and IIR filters. What are the benefits, limitations, and phase characteristics?
9. How do you derive the transfer function of an IIR filter given a schematic/diagram?
10. What is the difference between a low-pass and high-pass filter?

5 Timing

1. Explain why a circuit's contamination delay might be less than its propagation delay.
2. Explain timing constraints for logic between two registers.
3. What are the equations for setup and hold time? Explain the variables and meanings.
4. If a buffer is added to the clock input of the second flip-flop, how does that affect setup time requirements?
5. List a few ways to fix a setup time violation.
6. List a few ways to fix a hold time violation.
7. What is metastability, and how do synchronizers mitigate its effects?
8. What is clock skew, and how does it impact setup and hold timing?
9. Define clock jitter and its impact on performance.
10. What is a Phase-Locked Loop (PLL)?
11. Identify and explain potential timing violations in the following waveform.

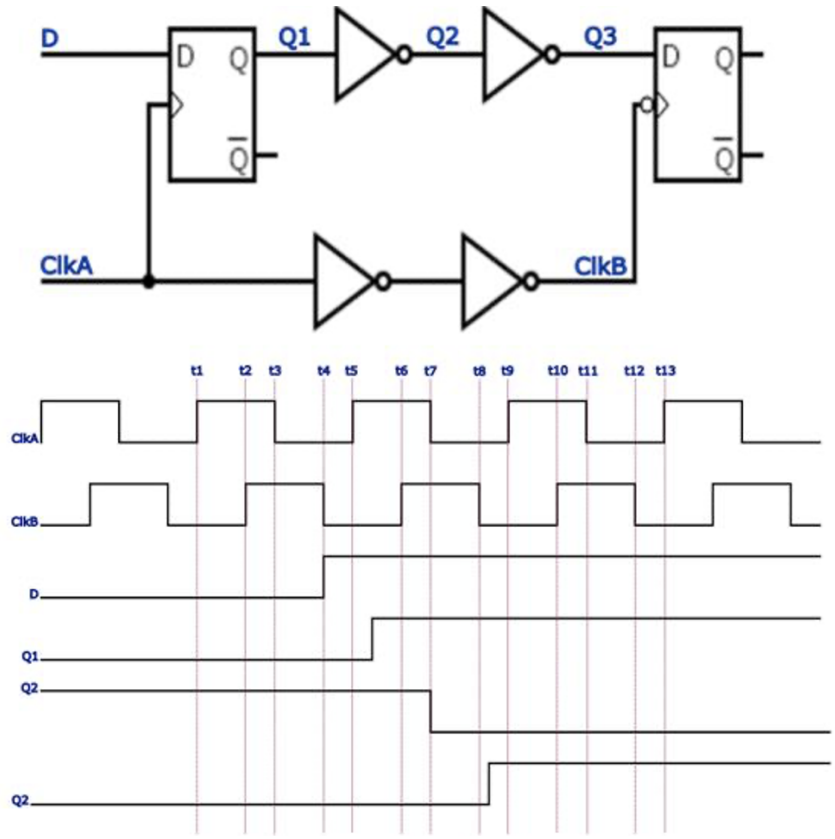
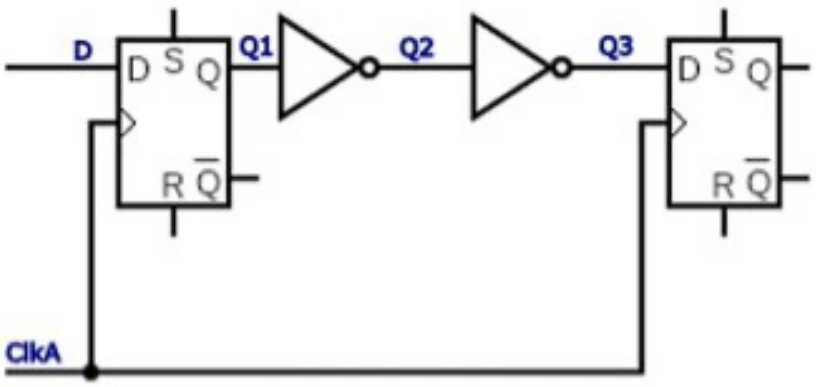


Figure 3: Timing Waveform.

12. Using the given timing values, calculate the minimum viable clock period.



i. $T_{Q1-Q3} = 5ns$
ii. $T_{ClkA-Q} = 1ns$
iii. $T_{Setup\ (Both\ FF)} = 2ns$

Figure 4: Timing Calculation Example.

6 Physical Design & FPGA/ASIC Flow

1. Describe the difference between synthesis in FPGA vs ASIC design. What is the meaning of a netlist?
2. How did you simulate your design? What tools did you use?

3. How may power (VDD) and ground (GND) nets be typically routed on a chip?
4. Explain the process and goals of clock tree synthesis. What considerations should be made for clock routing?
5. What are the some challenges during place and route/layout?
6. What is timing closure? Have you used a static timing analysis tool?
7. Describe how an FPGA functions, and specifically, what is a Look-Up Table (LUT)?
8. What is High-Level Synthesis (HLS)?
9. What is a constraints file in the context of FPGA design?
10. Walk through a typical ASIC design flow and project life cycle.
11. What is the difference between DRC and LVS checks? Why are each important?
12. What are the advantages and disadvantages of using low-voltage transistors (LVT) in ASIC design? Relate your answer to PPA (Power, Performance, Area) trade-offs.
13. What are the effects of changing drive strength of a transistor in an ASIC design? How does it affect PPA?
14. What are static and dynamic power? How can you reduce each in an ASIC design? What is the equation for dynamic power?

7 Architecture & Miscellaneous

1. What is an ALU?
2. What is a stack data structure? What is queue data structure?
3. What is a FIFO buffer? What happens if the write frequency is greater than the read frequency? How is the depth/size determined?
4. What is heap memory, and how does it differ from stack memory?
5. What is a smart pointer in C++? How does it differ from a pointer in C?
6. How is C code eventually ran by a CPU? Describe the process from C code to machine code.
7. What is a program counter, and how does it work in a CPU? How do jump or branch instructions affect it?
8. How do you reverse a linked list in C?
9. What is spatial and temporal locality in the context of cache memory?
10. Compare clock gating and power gating. When is each used?
11. How can multiplication be implemented efficiently in digital circuits?
12. What is the maximum possible result from multiplying two (N)-bit unsigned numbers? How does this differ for signed numbers?
13. Compare cache organizations: direct-mapped, set-associative, and fully-associative.
14. What are pipeline hazards, and how can they be resolved? Why don't modern CPUs use extremely deep pipelines (e.g., 100 stages)?
15. Discuss the trade-offs involved in different implementations of FSM encodings on an FPGA such as one-hot, binary, and gray code.

16. What is a JTAG interface, and how is it used in hardware debugging?
17. What are DACs and ADCs, and how are they applied in systems?
18. What is the resolution of a DAC or ADC, and how does it relate to bit depth and voltage levels?
19. What is SPI and what signals are needed to implement it?
20. What is I2C and what signals are needed to implement it?
21. What is UART and what signals are needed to implement it?

8 Recommended Resources & Further Tips

- Download Xilinx Vivado and practice writing and verifying designs. Get familiar with the design flow from source creation to bitstream generation. Customize the waveform views to ensure you understand the simulation. Understand the outputs of the synthesis and implementation tools, and how to read the timing reports. Bonus: Automate a project build using Tcl scripts in Vivado.
- Use online resources such as nandland.com, EDA Playground, and HDLBits to practice and learn Verilog/SystemVerilog in a browser.

References

[1] D. A. Harris and S. L. Harris, *Computer Architecture & Digital Design*. Burlington, MA: Morgan Kaufmann, risc-v ed., 2021.