1. Basics of Digital design   
    VIDEOS :   
   Neso academy digital electronics  
    BOOK:   
   Digital design by Morris Mano  
   Solve DigiQS and previous year gate Digital questions (Don’t leave anything)  
     
   2.) Verilog :   
   Install Vivado for Verilog , and start hands-on in parallel.  
   VIDEOS :   
   Design Lectures of Prof.Indranil Sengupta on Hardware Modelling using Verilog  
   BOOK:  
   Verilog HDL by Samir Palnitkar  
     
   3.) STA   
   Solve as many problems as possible, lot of good problems available online.   
   References :  
   You can go through Synopsis guide which are available for free online.   
   VIDEO:   
   If at all you want to watch video lectures then,  
   Udemy Course on Static Timing Analysis by Kunal Ghosh  
     
   4.) Computer Architecture   
   VIDEO :   
   Neso Academy Computer Organisation &Architecture  
   BOOK:  
   Computer Architecture by Patterson &   
   Hennessy   
     
   5.) CMOS   
   Videos : Lectures of Prof.Janakiraman on Digital IC design (first watch only first 20 videos)   
   Watch this carefully   
   Voltage Transfer Characteristics  
   Noise Margin  
   Leakage, short ckt & Dynamic Power  
   Short Channel effects  
   Books :   
    Digital Integrated Circuits by Rabaey  
    OR  
   CMOS Digital Integrated Circuits by Kang  
      
   6.) C programming basics   
   Video:   
   Neso Academy C playlist.   
   Solve geeksforgeeks questions.  
      
   In Digital Domain you’ll have one more choice to make Fronted or Backend.  
     
   First complete the things I mentioned for digital then,  
     
   If you are good at/ Interested more in   
   > Verilog, C and Architecture then Frontend !   
   > STA, CMOS & power related topics then Backend!

In Digital frontend you have :   
  
>Rtl design   
>Design Verification   
>Design for Test  
  
In Digital backend you have :   
> Sythensis   
> Sta   
>Physical Design   
> Physical Verification   
> Validation   
  
Now I recommend that you prepare for all Frontend roles or Backend roles.   
  
In frontend at least prepare for both design and Design verification roles:   
  
For Rtl design & Verification:   
  
1.)Design Lectures of Prof.Indranil Sengupta on Hardware Modelling using Verilog  
  
Here lecture no: 14 & 28 is very very important, one should know what exactly the rtl code maps to on hardware and vice versa.  
  
Once you complete this lecture, take any rtl code and try designing its hardware on a paper.   
  
After this you should be master in :  
  
> Knowing what writing style of hardware maps to what hardware. ( One change in the RTL can make a difference between a latch or a flop)  
  
> Knowing if the code is synthesizable or not!  
  
> blocking, non-blocking (solve questions on this)  
  
> Converting any problem statement to its fsm diagram and writing fsm based rtl code for the same.  
  
BOOK:  
Verilog HDL by Samir Palnitkar  
  
  
2.)Cover STA (Basics is enough)   
References :  
You can go through Synopsis guide which is available for free online.   
  
3.) Fifo   
  
> Depth calculation   
> Synchronous vs Asynchronous  
> Write RTL code for it and then also make a testbench to test it!  
  
Reference: available on yt (just whatever is available for free on yt is enough no need to take any paid courses for this)  
  
> Vlsi Point fifo playlist   
> Rahul from quicksilicon Rtl Design and Verification of fifo   
  
4.) Clock Domain Crossing   
  
> Synchronous vs Asynchronous  
> Metastability  
> MTBF   
  
reference: (youtube)  
Electronicspedia Clock Domain Crossing   
  
5.) OOP programming required for System Verilog   
  
Reference:   
SystemVerilog Classes by Cadence Design system (available on yt)  
OOP concepts discussed are what you exactly need!  
  
6.) System Verilog  
  
Main topics to look out for :  
> Task vs Functions   
> Fork join statements   
> Inheritance   
> Polymorphism   
>Randomisation  
>Assertions  
  
Reference:   
SystemVerilog by Openlogic (youtube)  
  
7.) UVM using System Verilog   
Reference YouTube:   
> uvm basics by synopsis (beginner level)  
> Asic lab uvm basics   
  
Opensource tools   
Verilog : Vivado   
Sv : Eda Playground

If you are a VLSI aspirant (fresher/experienced) precisely in Design/Verification role, then you must have the understanding in following areas, providing the resource link for each :  
  
⛳ Digital logic design  
From Number system to Sequential Circuits/state machines/memories, you should have the basics in your subconscious. You should be able to draw Muxes/Flip-flops/K-maps/FSM's.   
📍 https://lnkd.in/gW4twZ8i : Udemy course on digital basics  
📍 Digital Design book by Morris Mano.  
📍 Video lecture by Neso academy on YouTube.  
  
⛳ Hardware Description Languages (Verilog/System Verilog)  
For Design Verilog and for Verification System Verilog are the hardware languages you need to have command in. System verilog is almost same as verilog just some updates and extra features top of it.  
📍 NPTEL Verilog lectures by Prof. Indranil Sen Gupta, till lecture 25 it is sufficient , https://lnkd.in/g-WX-23G  
📍 SV Udemy course : https://lnkd.in/gE2Xkt7x  
📍 Verilog HDL: A Guide to Digital Design and Synthesis Book by Samir Palnitkar.  
  
⛳ Protocols (On Chip protocols/Communication protocols)  
There are two types of protocols : on chip protocols like AXI Interconnect and peripheral/communication protocols like PCIE, DDR, APB, SPI, I2C,USB.  
📍 https://lnkd.in/gHHKz72t : Udemy course for protocols  
📍 https://lnkd.in/g8sh8Tem : Brief intro on SoC protocols by Semi Design  
  
⛳ Computer Architecture  
Process Architecture, caches and memory systems are the topics that should be prepared.  
📍 https://lnkd.in/ga5hidHw : Neso academy lectures on you tube  
📍 Computer Organization and Design by David Patterson.  
  
⛳ Programming  
Basics, OOPS, UNIX/Linux, C/C++.  
📍 Solve C/C++ MCQ questions from geeksforgeeks website.  
📍 For UNIX go through tutorials point website  
  
⛳ Scripting Languages   
Perl, Python are the widely used scripting languages that can be the cheery on top if you have command in either of these.  
📍 For Perl/python, https://lnkd.in/g4T-P-M4  
  
⛳ Verification Methodologies   
UVM/OVM, Formal, Power(UPF), Coverage, Assertions are the methodologies being used in projects in the industry.  
📍 https://lnkd.in/gDfJmQYF  
📍 <https://lnkd.in/dSkpJp5K>

`  
Other Important areas :   
✏ Complete ASIC Design flow understanding  
✏ Good understanding of STA (static timing analysis), solve questions on Setup/Hold time violations.  
✏ Questions on calculation of FIFO Depth.  
✏ Go over the projects related to VLSI done in college or in internships.  
✏ Interview Questions for practice : <https://lnkd.in/gNvY2dTn>

I am sharing some of the most frequently asked digital design questions in interviews & written tests.  
  
50 Basic Questions to Prepare in Sequential Circuits:  
  
1. What is a latch? Explain using NOT, NAND, and NOR gates.  
2. Why do we use NAND gates more when compared with NOR gates?  
3. Difference between Latch and Flip-flop?  
4. Why flip-flops are preferred when compared with latches?  
5. What is a transparent latch?  
6. What is race condition?  
7. What is master-slave configuration?  
8. What is a glitch?  
9. Which is more sensitive to glitches (latch or flip-flop)?  
10. Draw & explain D-flip flop vs D-latch timing diagrams.  
11. Draw a D-latch using 2:1 MUX.  
12. Draw a D-flip flop using 2:1 MUX.  
13. Which flip flop is most widely used?  
14. Application of flip flops?  
15. Explain Clock to Q (C2Q) delay.  
16. What is the setup time & hold time?  
17. What is metastability?  
18. How to avoid setup time & hold time violations?  
19. What is clock skew?  
20. Draw the excitation table for the SR flip-flop.  
21. Design a JK flip-flop using a D flip-flop?  
22. Design a T flip-flop using a D flip-flop?  
23. Design a 4-bit up-down counter?  
24. Design a frequency divider by 3 circuit.  
25. Design & explain the MOD-5 counter.  
26. How many flip-flops are required to design a MOD-535 counter?  
27. Draw a 4-bit counter using one D flip-flop & one adder circuit.  
28. Difference between synchronous and asynchronous counters?  
29. What is asynchronous reset & how is it different from synchronous reset?  
30. How many bits a flip-flop can store?  
31. What is a register & how many bits it can store?  
32. What is a shift register & the types of shift registers?  
33. Draw a 4-bit Johnson counter?  
34. Design a PISO Shift register?  
35. Draw & Explain the 4-bit Linear Feedback Shift Register.  
36. Draw & Explain a 4-bit Barrel Shifter using MUX.  
37. What is a Finite State Machine?  
38. Explain Mealy Vs Moore FSM.  
39. Which is preferred (Mealy or Moore)?  
40. Draw a state diagram for the 101 sequence? (both Mealy & Moore non-overlap)  
41. Draw a state diagram for the JK flip-flop.  
42. Design a sequence detector that detects both 1010 & 0101(both mealy & moore overlap).  
43. Design a sequence detector that detects 1X1X (moore overlap).  
44. What is a Clock Domain Crossing (CDC)?  
45. How to overcome CDC?  
46. What is a 2-stage synchronizer?  
47. What is MTBF?  
48. What is FIFO? Explain Synchronous vs Asynchronous FIFO.  
49. Explain Dual port RAM?  
50. What are full & empty conditions in FIFO?

1. What is a combinational circuit?   
2. Difference between combinational & sequential circuits  
3. What are universal gates?  
4. Which logic gate is used more in real-life applications?  
5. What are the applications of Buffer?  
6. Realize the equation Y=AB+C using only 2 input NAND gates  
7. Realize XOR gate using only 2 input NOR gates  
8. Implement a NOT gate using the XOR gate  
9. Explain the difference between NAND and negative input AND gate  
10. Explain the differences between MOSFETs and BJTs, and why we use MOSFET in digital design.  
11. Draw 2 input NAND & NOR gates using CMOS logic  
12. What are pull-up & pull-down networks?  
13. Explain the operation of the CMOS Inverter  
14. Draw the VTC curve of the CMOS Inverter  
15. Explain about half adder  
16. Design an OR gate using half adders  
17. Design a Full adder using only NAND gates  
18. In how many ways, a full adder can be implemented using logic gates?  
19. Draw the Full subtractor truth table & circuit  
20. Difference between Ripple Carry Adder and Carry Look Ahead Adder  
21. Design a logical circuit that performs both addition and subtraction  
22. What is a comparator?  
23. Differences between Encoder and Decoder  
24. Applications of Encoder & Decoder  
25. Design 4 to 16 decoder using 2 to 4 decoders  
26. What is a priority encoder?  
27. What is a Multiplexer? Applications of MUX  
28. Why MUX is called a universal circuit?  
29. Design 4:1 MUX using 2:1 MUX  
30. Design logic gates (AND, OR, NOT, NAND, NOR, XOR) using 2:1 MUX  
31. How many 2:1 MUX are required to design 64:1 MUX?  
32. Realize 2 input AND gate using 4:1 MUX  
33. Implement 2 input NOR gate using 1:2 DEMUX  
34. Implement a full adder using 4:1 Muxes  
35. Explain tri-state buffers (notif1, notif0, bufif1, bufif0)  
36. Implement basic gates (AND, OR, NOT) using tri-state buffers  
37. Design a 4:1 MUX using tri-state buffers  
38. Design a 2:1 MUX using decoder & tri-state buffers  
39. Design a combinational circuit that converts a 4-bit gray code number to a 4-bit binary number using only XOR gates  
40. Design a combinational circuit, that doubles the input frequency. (frequency multiplier by 2)

Digital Electronics Interview Questions (theoretical & MCQs) to test your preparation:

PART 1: <https://lnkd.in/dcFHuviK>  
  
PART 2: <https://lnkd.in/dRrYWt4d>  
  
PART 3: <https://lnkd.in/dwrSPWpt>  
  
PART 4: <https://lnkd.in/dWmKWu48>  
  
PART 5: <https://lnkd.in/daYsS4Px>  
  
PART 6: <https://lnkd.in/dRAE3ubv>  
  
PART 7: <https://lnkd.in/dG8vHyPc>

Sharing useful links for VLSI interview preparation:  
  
1. Useful Verilog Examples : <https://lnkd.in/g--Ep22h>  
  
2. Verilog Interview Questions : <https://lnkd.in/gSFQH9_W>  
  
3. Digital Design Interview Questions : <https://lnkd.in/gBTGeYyr> and  
<https://lnkd.in/gNPfNbmW>  
  
4. FPGA Interview Questions : <https://lnkd.in/g6vBe4fZ>  
  
5. Important VLSI Commands (Linux Commands and GVIM/VI Commands) : <https://lnkd.in/grKsaffB>  
  
6. System Verilog Questions : <https://lnkd.in/gz4EpHdR>  
  
7. UVM Questions : <https://lnkd.in/gFTQnZRr>  
  
8. DLD & COA Questions : <https://lnlkd.in/g_qQN4rZ>  
  
9. Bash Scripting Tutorial & examples : <https://lnkd.in/gg2kRZF9>

Important topics on Verilog in Digital Design useful for professionals who want to enter the field of VLSI:  
  
✓ Verilog stratified event queue understanding and the execution of each block. Difference between preponed and postponed region.  
  
**✓ Difference between net and reg. Default values for wire and reg. Strength levels of different drivers.**

**Strength levels of different drivers:**

In Verilog, different drivers can drive values onto nets or registers with different strengths. The strength levels include:

1. **supply0 and supply1:** These strengths are used to drive a strong 0 or a strong 1, respectively. They are typically used in continuous assignment statements.

verilogCopy code

assign net1 = 1'b1; // supply1 strength

1. **strong0 and strong1:** Similar to **supply0** and **supply1**, these strengths are used to drive a strong 0 or a strong 1. They are often used in procedural assignments.

verilogCopy code

always\_ff @(posedge clk) reg1 <= 1'b0; // strong0 strength

1. **weak0 and weak1:** These strengths are used to drive a weak 0 or a weak 1. Weak drivers can be overridden by strong drivers.

verilogCopy code

initial $monitor("net2=%b", net2); initial net2 = 1'bz; // weak0 strength

1. **highz0 and highz1:** These strengths are used to set the net to a high-impedance state (Z) for 0 or 1, respectively.

verilogCopy code

initial net3 = 1'bz; // highz0 strength

These strength levels are important for modeling different types of driving and loading scenarios in digital designs. The appropriate use of strengths helps ensure that the simulation results accurately reflect the intended behavior of the hardware.

Top of Form

**✓ Difference between Blocking and Non-blocking statements. Coding guidelines for both of them. Which one to use where. Can we mix both the statements.**

\*\*Blocking Statements:\*\*

- \*\*Definition:\*\* Blocking statements are executed sequentially, and the control does not move to the next statement until the current one is completed. They use the `=` assignment operator.

- \*\*Example:\*\*

```verilog

always @(posedge clk) begin

a = b + c; // Blocking assignment

d = a - e; // Control waits until this assignment is completed

end

```

- \*\*Coding Guidelines:\*\*

- Use blocking assignments for synthesizable sequential logic.

- Ensure that the order of execution matches the intended behavior of the hardware.

\*\*Non-blocking Statements:\*\*

- \*\*Definition:\*\* Non-blocking statements allow for concurrent execution. The control immediately moves to the next statement without waiting for the current one to complete. They use the `<=` assignment operator.

- \*\*Example:\*\*

```verilog

always @(posedge clk) begin

a <= b + c; // Non-blocking assignment

d <= a - e; // Control moves to the next statement without waiting

end

```

- \*\*Coding Guidelines:\*\*

- Use non-blocking assignments for modeling synchronous hardware behavior.

- Non-blocking assignments are commonly used in always blocks sensitive to clock edges.

- Helps in avoiding race conditions in simulation.

\*\*Which One to Use Where:\*\*

- Use blocking assignments for combinational logic or tasks where the order of execution matters.

- Use non-blocking assignments for sequential logic, particularly inside clocked always blocks, to model flip-flop behavior and avoid race conditions in simulation.

- Be consistent in your coding style within a module.

\*\*Mixing Blocking and Non-blocking:\*\*

- \*\*Sequential Blocks:\*\* In general, use non-blocking assignments inside sequential blocks like always @(posedge clk).

```verilog

always @(posedge clk) begin

a <= b + c; // Non-blocking assignment

d <= a - e; // Non-blocking assignment

end

```

- \*\*Combinational Blocks:\*\* Use blocking assignments for combinational logic or in procedural blocks where the order of execution matters.

```verilog

initial begin

a = b + c; // Blocking assignment

d = a - e; // Blocking assignment

end

```

\*\*Caution:\*\*

- Avoid mixing blocking and non-blocking assignments within the same always block or procedural block to prevent unintended behavior.

- Mixing them can lead to simulation mismatches and difficulties in understanding the code.

In summary, understanding the differences between blocking and non-blocking assignments is crucial for writing correct and understandable Verilog code. Use blocking assignments for combinational logic and non-blocking assignments for sequential logic, and be consistent in your coding style. Mixing them within the same block can lead to simulation mismatches and should be avoided.

<https://chat.openai.com/share/0a1548c4-ace2-4c6f-a4c7-66f0558313ce>

**✓ Difference between Inter and Intra assignment delays. Execution flow of both the delays using blocking and non-blocking.**  
  
**✓ Difference between local param and defparam statements. What are specparam statements.**

**✓ Difference between Task and Function. Can we call a task inside a function and vice versa. If so how.**  
  
**✓ Difference between casex and casez. Difference between Verilog full case and parallel case and why are they required.**

**✓ Coding in verilog for all the combinational and Sequential circuit of digital design.**  
**✓ Different types of coding modelling style in verilog and which one to use where.**

**✓ How to generate a random number in verilog and what is the need of seed. Difference between rand and randc constructs.**

**✓ Understanding on the synthesizable and nonsynthesizable constructs and what they mean.**

**✓ Coding for sequence detector in verilog. Coding for Mealy and Moore FSM for a particular sequence.**  
  
**✓ Difference between $stop and $finish. Difference between $monitor and $display. Different types of system task understanding,Z.**

**✓ Understanding on the usefulness of different types of compiler directives and UDP primitives.**

**✓ Different types of data types and port connectivity rules in verilog.**

**✓ RTL coding guidelines in verilog - Different execution strategies.**

**✓ Initial, always and generate block understanding. Execution flow of fork join statements.**

**✓ Difference between assign/deassign and force/release statements.**  
**✓ Design a out of order execution processor core and a floating point unit.**  
**✓ Cordic algorithm in verilog and how to calculate the twiddle factors.**  
**✓ Synchronous and Asynchronous Fifo design in Verilog.**  
  
**✓ Verilog code for bidirectional data bus, traffic light controller, SP RAM, DP RAM.**  
**✓ Race around condition in verilog and how to avoid them.**  
**✓ Different ways of clock generation techniques in verilog.**  
**✓ Difference between Verilog and SV and what is the need to migrate from Verilog to SV.  
  
✓ SISO and PISO implementation in Verilog.**  
**✓ Difference between the delay modelling in VHDL and Verilog.**

**✓ How to open a file, writing, reading, append, etc operations. Use of $strobe. Need of VCD in verilog.**  
**✓ Difference between distributed and lumped delay. Use of specify block in verilog. Use of $setup, $hold and $width task in verilog.**  
**✓ How to model a testbench in verilog. How the DUT and TB is going to interact with each other**

Verilog tasks and functions are both used to group code together and make it more reusable. However, there are some key differences between the two.

**Functions**

* Functions are meant to perform some processing on the input and return a single value.
* Functions cannot have time-controlling statements or delays.
* Functions cannot enable other tasks or functions.
* Functions can have one or more input arguments, but they cannot have output or inout arguments.
* Functions must return a single value.

**Tasks**

* Tasks are more general and can calculate multiple result values and return them using output and inout type arguments.
* Tasks can contain time-controlling statements or delays.
* Tasks can enable other tasks or functions.
* Tasks can have zero or more arguments of any type.
* Tasks cannot return a value, but they can achieve the same effect using output arguments.

**Here is a table that summarizes the key differences between functions and tasks in Verilog:**

|  |  |  |
| --- | --- | --- |
| Characteristic | Function | Task |
| Can perform time-consuming operations | No | Yes |
| Can enable other tasks or functions | No | Yes |
| Can have output or inout arguments | No | Yes |
| Must return a value | Yes | No |

**Examples**

Here is an example of a Verilog function:

Verilog

function [31:0] add (

input [31:0] a,

input [31:0] b

)

return (a + b);

endfunction

Use code with caution. [Learn more](https://bard.google.com/faq#coding)

content\_copy

This function adds two 32-bit numbers and returns the result.

Here is an example of a Verilog task:

Verilog

task delay (

input [31:0] time

)

#time;

endtask

This task waits for the specified amount of time before returning.

**When to use functions and tasks**

Functions are typically used to perform simple operations, such as arithmetic calculations or logical operations. Tasks are typically used to perform more complex operations, such as communication or control.

**Conclusion**

Verilog tasks and functions are both powerful tools for organizing and reusing code. By understanding the key differences between the two, you can choose the right tool for the job.

How to start learning Verilog from Scratch: A complete stepwise approach  
  
STEP 1:  
  
Prerequisite:  
  
[1] Good understanding of all the Digital Design Concepts (Gates, FSM, Combinational, Sequential Circuits, Boolean Algebra, Kmap, SOP and POS concepts, Implicants, Hazards, etc) because the concepts and the circuit realization needs to be applied while writing codes in verilog.  
  
List of books:  
  
[a] “Fundamentals of digital circuits by A. Anand Kumar”.  
  
[b] “Digital Design by M. Morris Mano”  
  
[c] “Digital Design by John F. Wakerly”.The concepts which are uncovered in digital design will get covered here.  
  
[d] “Fundamentals of logic design by Charles H Roth”. Refer the FSM from this book.  
  
[e] NPTEL videos (like time borrowing concepts, latch, etc)  
  
[2] Good understanding of the timing analysis concepts like Setup, hold, metastability, clock skew, slew, Maximum frequency and worst delay calculation, etc.  
  
STEP 2:  
  
Learning Verilog:  
  
[3] Once the Digital Design concepts are thoroughly mastered and practiced, one can start referring a book in verilog like Verilog by Sameer Palnitkar and J. Bhasker. Both the books are equally good.  
  
[4] Although all the concepts are important but one has to provide more focus on the concepts like a) wire and Reg b) blocking and non blocking statements, c) delta delay, d) drive strength, e) Compiler directives, f) data types g) Inter and Intra Assignment delay, h) Verilog Event Queue, i) Task and Function, j) Different modelling style of writing code, k) UDP Primitive and Spec parameter constant, l) defparam and local param, m) Generate statement, n) fork and join, o) initial and always, p) loops r) assign and deassign, force and release.  
  
[5] Once all the above concepts are learned, differentiate between the synthesizable and non synthesizable construct because the RTL only uses the construct which are synthesizable.  
  
[6] Download the IEEE Verilog Language Reference Manual (LRM) and keep it as a golden guide.  
  
[7] Start with codes like 2:1 Mux, counter, synchronous and asynchronous Flip flops, priority encoder, decoder and adder circuits.  
  
[8] Once this exercise is over one can move on to State machine coding in Mealy or Moore and Sequence detector in overlapping or non overlapping mode.  
  
[9] After running the code, verify the code through Simulation waveform and also observe the synthesizable representation of the circuit. Also someone can build up a basic Testbench to provide the stimulus.  
  
[10] Some of the other sources of learning are Sunburst Design Papers which are of immense help. NPTEL video lectures of verilog are also of very good help.

module updown(data\_in,clk,reset,load,control,out\_data);

Input clk,reset,load;

Input [3:0]data\_in;

Output reg out\_data;

always@(posedge cllk)

begin

if(reset)

out\_data<=0

else if (load)

out\_data<=data\_in

else if (control)

count<=count+1;

else

count<=count-1;

endmodule

end

numbers not specified are taken as decimal by default.

4’b1111- binary

4’habc1-hexadecimal

4’d2568-decimal

4’bo1234-octal

Negative no’s can be represented by

-7’d4 (stored as 2’s complement of 7)

We can write

7’b1111111

As 7’b1111\_111 //underscores are valid but just not before the number

? can be used in place of z(high impedance)

“Hello world” -always a single line supported by double quotes.

reg value;

reg-keyword

value -identifier;

identifier should start with alphabet or underscore but not with numbers.numbers can elsewhere in the identifier name.

value set in Verilog

0,1,x,z

Reg-x-unknown

Wire-z-high impedance

Data types in Verilog

Wire,reg,integer,real,time

Integer can take signed numbers whereas reg is only for unsigned numbers.

Parameters-used for defining the constants

$monitor($time,”value of signal clock=%b reset=%b”,clock,reset);

'define

The 'define directive is used to define text macros in Verilog (see Example 3-6). This is similar to the #define construct in C. The defined constants or text macros are used in the Verilog code by preceding them with a ' (back tick).

For defining ,

‘define word\_size 32

When we want to use it, write as ‘word\_size

'include

The ' include directive allows you to include entire contents of a Verilog source file in another Verilog file during compilation.

‘include header.v

Module fa(a,b,c1,s,c);

Input a,b,cin;

output s,c;

s=a^b^c;

c=a&&b+ b&&c + c&&a;

module fa\_4(a,b,cin,s,c);

input [3:0]a,b;

input cin;

output [3:0]s;

output c;

wire c1,c2,c3;

fa fa1(a[0],b[0],1’b0,s[0],c1);

fa fa2(a[1],b[1],c1,s[1],c2);

fa fa3(a[2],b[2],c2,s[2],c3);

fa fa4(a[3],b[3],c3,s[3],c);

endmodule

testbench

module tb();

reg [3:0]a,b;

reg cin;

wire[3:0]s;

wire c;

fa \_4 fa7 uut(.a(a),.b(b),.cin(cin),.s(s),.c(c));

$dumpvars;

$dumpfile(:dump.vcd);

Initial

begin

$monitor($time,”A=%b,b=%b,cin=%b,s=%b,cout=%b\n”,a,b,cin,s,c);

end

initial

begin

A = 4'dO; B = 4'dO; C-IN = l'b0;

#10

A = 4'd1; B = 4'd1; C-IN = l'b0;

//so on

end

endmodule

There are three types of delays from the inputs to the output of a primitive gate.

Rise-(0 to 1), Fall(1 to 0) and Turn-off Delays(any state to z).

Min value

The min value is the minimum delay value that the designer expects the gate to have.

Typ val

The typ value is the typical delay value that the designer expects the gate to have.

Max value

The max value is the maximum delay value that the designer expects the gate to have.

Continuous assignment can be of implicit and explicit

explicit

wire out;

assign out=a&b;

implicit

wire out=a&b;

Difference between logical and bit wise operators:  
logical operators: ||,&&,~ ; always result in 0,1or x;

Bit-wise operators: yield a bit-by bit value.

Conditional operators:  
assign c=sel?a:b;

There are two structured procedure statements in Verilog: always and initial. These statements are the two most basic statements in behavioral modeling.

The statements always and initial cannot be nested.

. If there are multiple initial blocks, each block starts to execute concurrently at time 0.

Initial block statements are executed sequentially.

Timing controls provide a way to specify the simulation time at which procedural statements will execute. There are three methods of timing control: delay-based timing control, event-based timing control, and level-sensitive timing control.

//intra assignment delays

initial

begin

X=0; z=0;

y = $5 X + z; //Take value of X and z at the time=0, evaluate //X + z and then wait 5 time units to assign value //to y. end

//Equivalent method with temporary variables and regular delay control

Initial

Begin

x=o; z=o;

temp\_xz = X + z;

$5 y = temp\_xz; //Take value of X + z at the current time and //store it in a temporary variable. Even though X and z //might change between 0 and 5, //the value assigned to y at time 5 is unaffected. end

event received-data;

//Define an event called received-data

always @(posedge clock) //check at each positive clock edge

begin

if(last-datapacket) //If this is the last data packet

->received-data; //trigger the event received-data

end

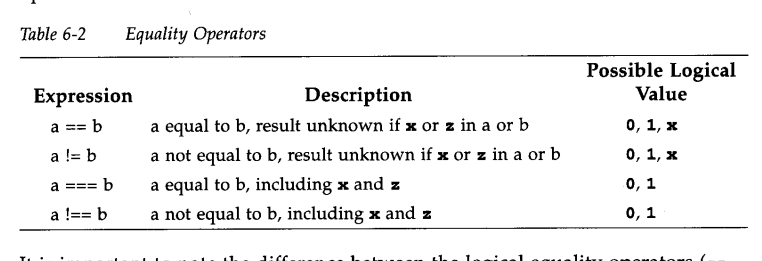
always @(received-data) //Await triggering of event received-data //When event is triggered, store all four

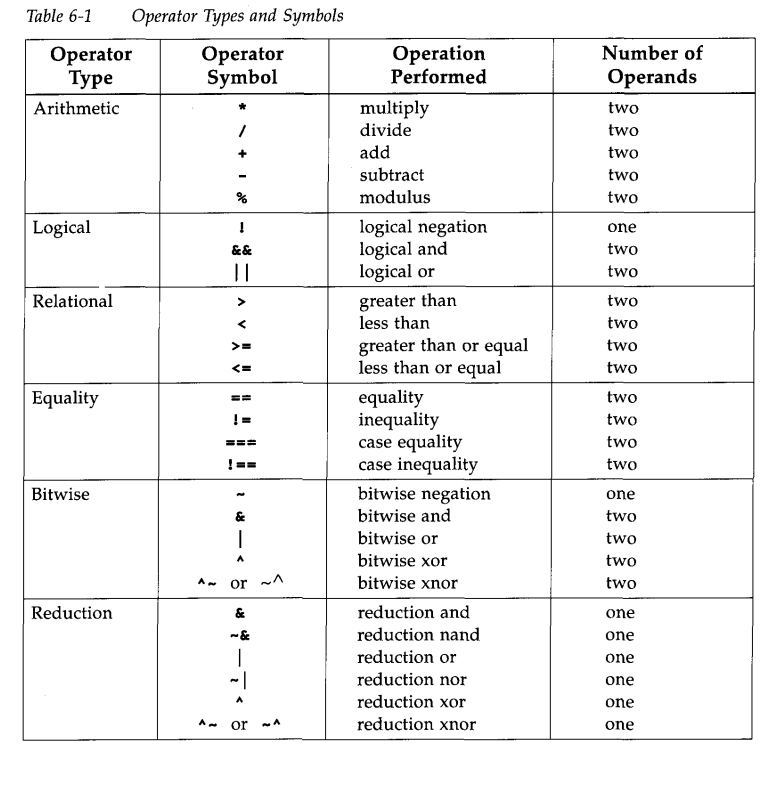
//packets of received data in data buffer //use concatenation operator { }

data-buf = {dataqkt [O] , dataqkt [l], dataqkt [2], dataqkt [3] } ;

always wait (count-enable) #20 count = count + 1;

In the above example, the value of count-enable is monitored continuously. If count-enable is 0, the statement is not entered. If it is logical I, the statement count = count + 1 is executed after 20-time units. If count-enable stays at 1, count will be incremented every 20-time units.





integer count;

initial

begin

count=0;

end

while(count<128)

begin

$display (“count=%d”, count);

count=count+1;

end

sequential blocks:  
executed sequentially.

Start with begin and stops with end .

Parallel blocks:  
specified by key words fork and join.

They are executed concurrently.

Delay is relative to the time it entered the block.

reg x,y;

reg [1:0]z,w;

initial

fork

# x=1’b1; t=0

#5 y=x; t=5

#10 z={x,y}; t=10

#20 w={y,x}; t=20

join

features of blocks:  
Nested blocks,named blocks and disabling of named blocks.

The keyword disable provides a way to terminate the execution of a block. disable can be used to get out of loops, handle error conditions, or control execution of pieces of code, based on a control signal. Disabling a block causes the execution control to be passed to the statement immediately succeeding the block. For C programmers, this is very similar to the break statement used to exit a loop. The difference is that a break statement can break the current loop only, whereas the keyword disable allows disabling of any named block in the design.

module counter4(clock,clear,q);

input clock,clear;

output reg[3:0]q;

always@(negedge clk,posedge clear)

begin

if (clear)

q<=4’d0;

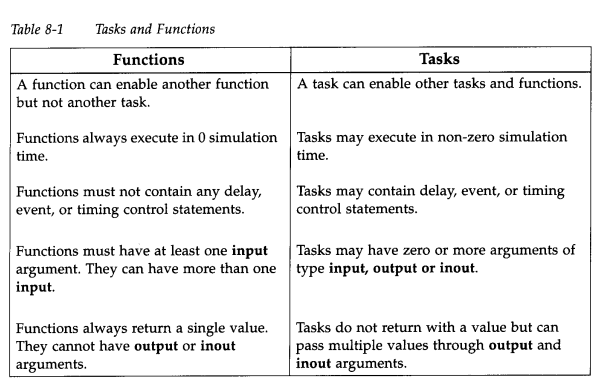
else

q<=(q+1)%16;

end

endmodule

Functions and Tasks



Tasks and functions contain behavioral statements only. Tasks and functions do not contain always or initial statements but are called from always blocks, initial blocks, or other tasks and functions.

Tasks are declared with the keywords task and endtask. Tasks must be used if any one of the following conditions is true for the procedure.

There are delay, timing, or event control constructs in the procedure.

The procedure has zero or more than one output arguments.

The procedure has no input arguments.

//Define a module called operation that contains the task bitwise\_oper module operation;

module operation;

……..

parameter delay=10;

reg[15:0]a,b;

reg[15:0]ab\_and,ab\_or,ab\_xor;

always@(posedge clk)

begin

bitwise\_oper(ab\_and,ab\_or,ab\_xor,a,b); // task or function is called from always block

end

//define task bitwise\_oper

task bitwise\_oper;

output[15:0]ab\_and,ab\_or,ab\_xor;

input [15:0]a,b;

begin

#delay ab\_and=a&b;

ab\_or=a|b;

ab\_xor=a^b;

ab=bitwise\_oper;

end

endtask

endmodule

function example

//Define a module that contains the function calc\_parity ;

module parity (

input [31:0] addr,

output reg parity

);

always @(addr)

begin

parity = calc\_parity(addr);

end

function bit calc\_parity;

input [31:0] addr;

begin

calc\_parity = ^addr;

end

endfunction

endmodule

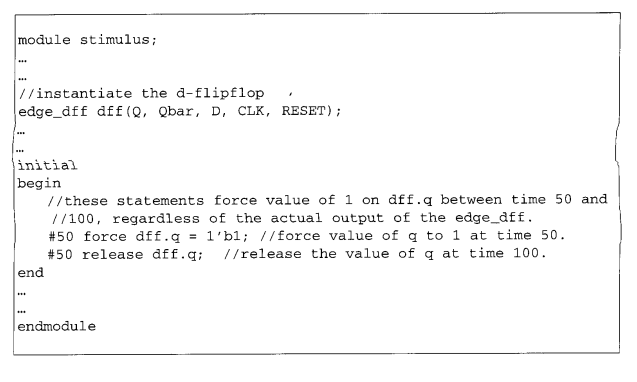
assign,deassign-continuous assignment

. The register variables retain the continuously assigned value after the deassign until they are changed by a future procedural assignment.

force and release

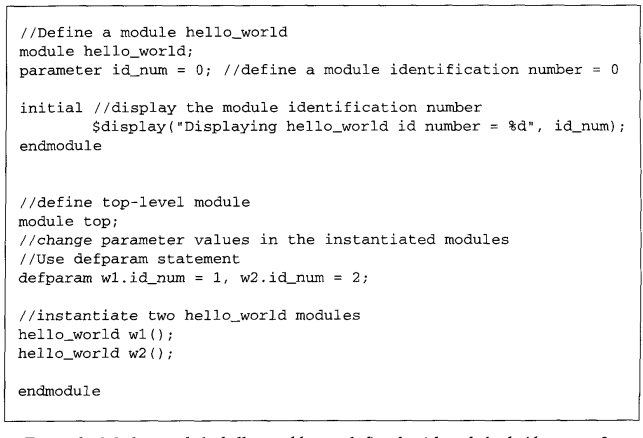
continuous assignment

They can be used to override assignments on both registers and nets. force and release statements are typically used in **the interactive debugging process, where certain registers or nets are forced to a value and the effect on other registers and nets is noted. It is recommended that force and release statements not be used inside design blocks. They should appear only in stimulus or as debug statements.**



**defparam:**

Parameter values can be changed in any module instance in the design with the keyword defparam.



// Code your design here

module ExampleModule #(parameter WIDTH = 4);

reg [WIDTH-1:0] data;

initial begin

$display("Module Instance with WIDTH=%0d", WIDTH);

end

endmodule

module Testbench;

// Without defparam

ExampleModule instance1(); // Uses default WIDTH value from the module definition

// With defparam

ExampleModule instance2();

defparam instance2.WIDTH = 8; // Set WIDTH to 8 for this instance

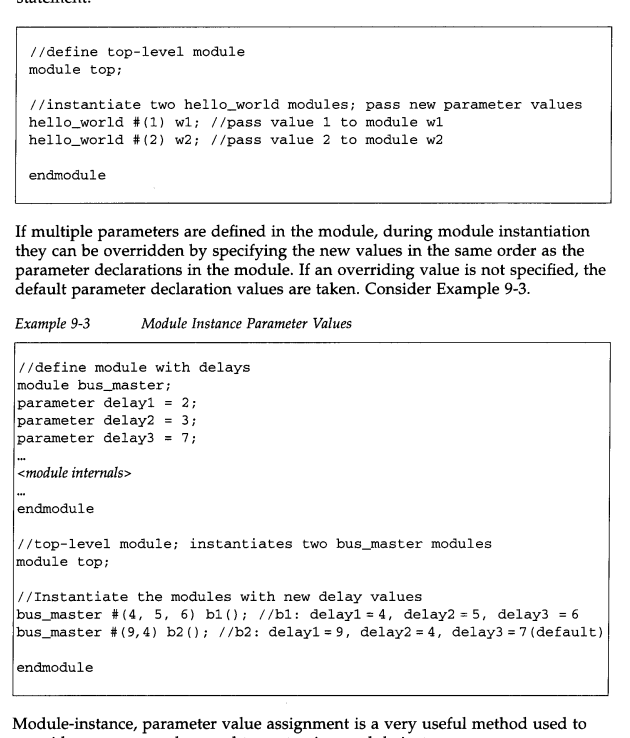
initial begin

#10 $finish; // Finish simulation after a delay

end

endmodule

**Without using defparam also we can change the parameter value as shown below.**



**Timescales:**Verilog HDL allows the reference time unit for modules to be specified with the ' timescale compiler directive

'timescale 100 ns / 1 ns

Which means time unit is 100 ns and a precision of 1 ns.

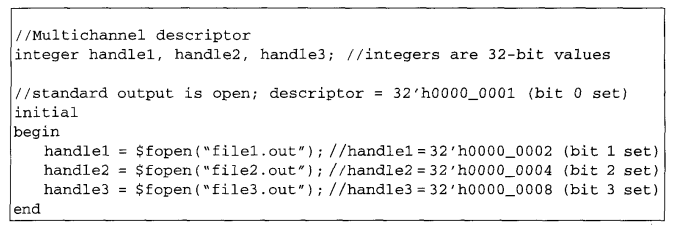
Multiple time scales can be given inside a code for different modules.

Opening a file:

Output from Verilog normally goes to the standard output and the file verilog.log. It is possible to redirect the output of Verilog to a chosen file.

A file can be opened with the system task $fopen.

The task $fopen returns a 32-bit value called a multichannel descriptor. Only one bit is set in a multichannel descriptor. The standard output has a multichannel descriptor with the least significant bit (bit 0) set. Standard output is also called channel 0. The standard output is always open. Each successive call to $fopen opens a new channel and returns a 32-bit descriptor with bit 1 set, bit 2 set, and so on, up to bit 31 set. The channel number corresponds to the individual bit set in the multichannel descriptor. Example 9-7 illustrates the use of file descriptors.



Writing to files:

The system tasks $fdisplay, $fmonitor, $fwrite, and $f strobe are used to write to files. Note that these tasks are the similar in syntax to regular system tasks $display, $monitor, etc., but they provide the additional capability of writing to files.

Displaying Hierarchy:

**$display**:-displays the output after every execution.

**$strobe:**Strobing is done with the system task keyword $strobe. This task is very similar to the $display task except for a slight difference**. If many other statements are executed in the same time unit as the $display task, the order in which the statements and the $display task are executed is nondeterministic. If $strobe is used, it is always executed after all other assignment statements in the same time unit have executed.** Thus**, $strobe provides a synchronization mechanism** to ensure that data is displayed only after all other assignment statements, which change the data in that time step, have executed.

Priority= write>display>strobe>monitor

**Random number generation:**

**Random number generation is used for generating a random set of test vectors.They are used for finding the hidden bugs in the design.They are used in performance analysis of chip architectures.** Usage: $random; $random(<seed>);

The value of is optional and is used to ensure the same random number sequence each time the test is run. The task $random returns a 32-bit random number. All bits, bit-selects, or part-selects of the 32-bit random number can be used

Initializing memory from file

$readmemb(“<file\_name>”,<memeory\_name>);

$readmemb(“,file\_name>”,<memeory\_name>,<start\_addr>);

$readmemb(“,file\_name>”,<memeory\_name>,<start\_addr>,<finish\_addr>);

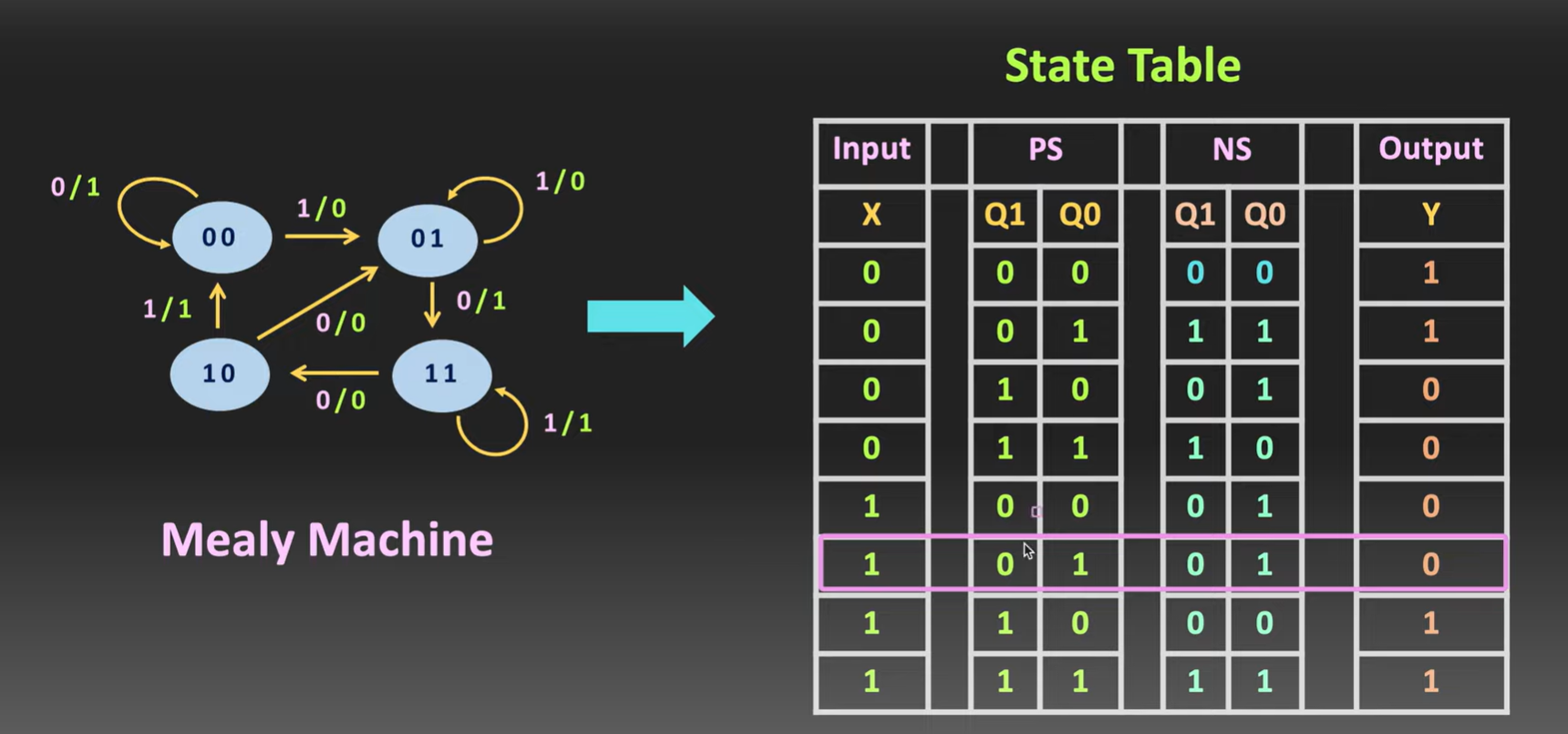
<https://www.linkedin.com/pulse/vlsi-fresher-interview-questions-answers-digital-raju-prasad/?utm_source=share&utm_medium=member_android&utm_campaign=share_via>

$display-print after very execution; terminates with new line

$strobe-print at the end of the current timestep

$write- same as display but doesn’t terminate with new line.

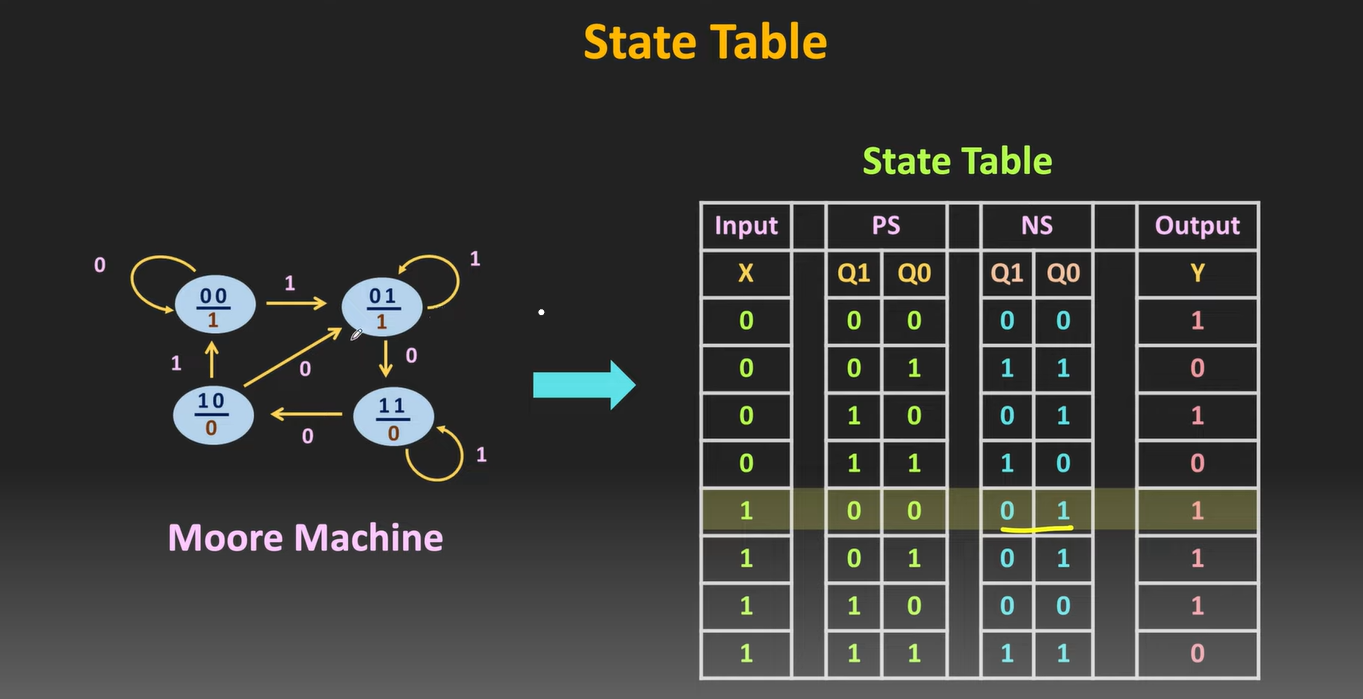
$monitor-print after every execution that involves a change in value measured.



**Here, in Mealy machine, let us consider an example.**

**PS-00**

**If we give 0 as input it gives the NS as 00 and the output would be 1. It is represented as 0/1; which means upon the input as 0 ,we get output as 1.**

****

In moore machine,The output depends only on the present state.

For example,for PS as 00,if we give input as 0,we go the same state 00 and the output is 1.

FIFO-First In First Out

It can be either Synchronous or Asynchronous one.

Synchronous-write and read operations at same frequency.

Asynchronous-write and read operations at different frequency.

Assume device A operating5 at 200 MHz frequency and device B operating at 100 MHz frequency.

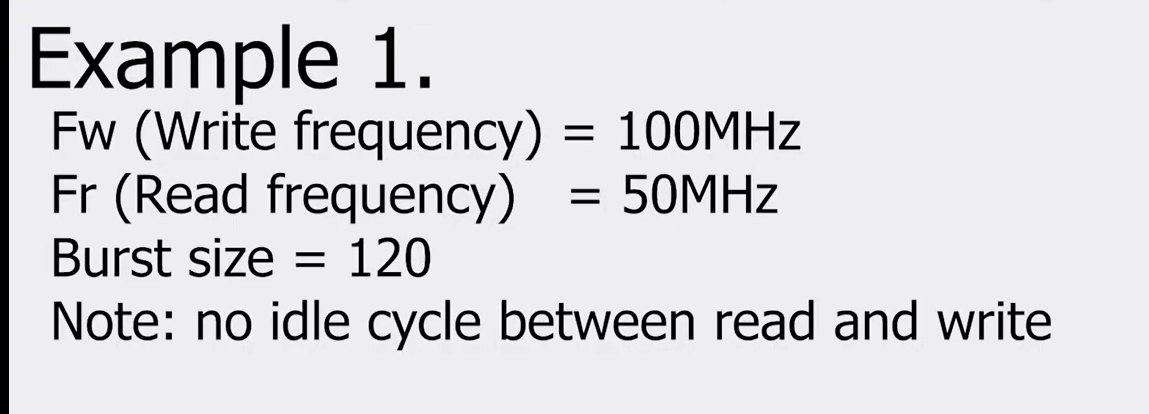
There will be a loss of information if the data transfer occurs between A and B.

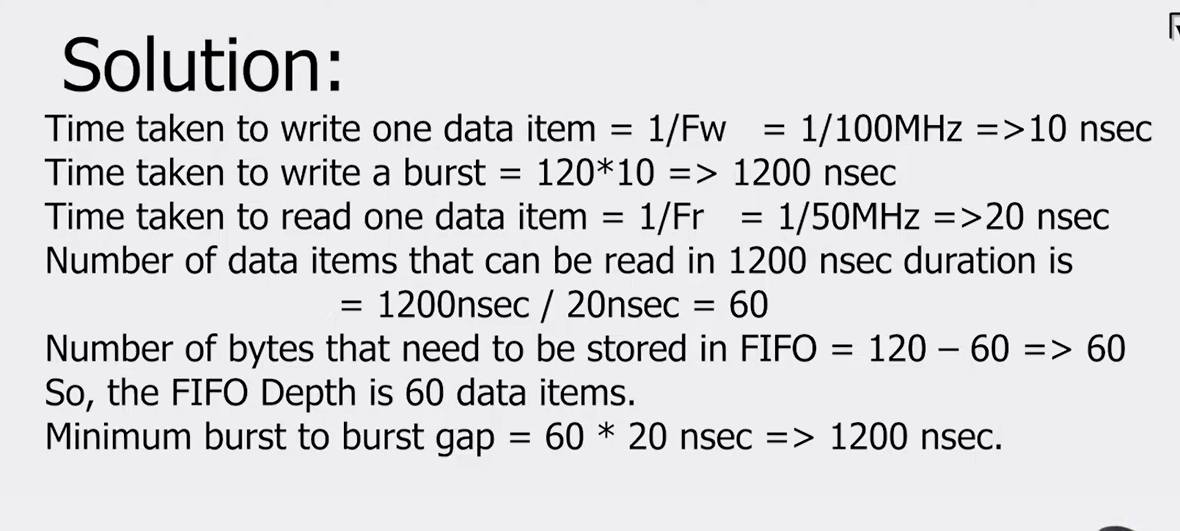
Therefore, we use Asynchronous FIFO that would be used to avoid the loss of data. It acts like a Synchronizer.

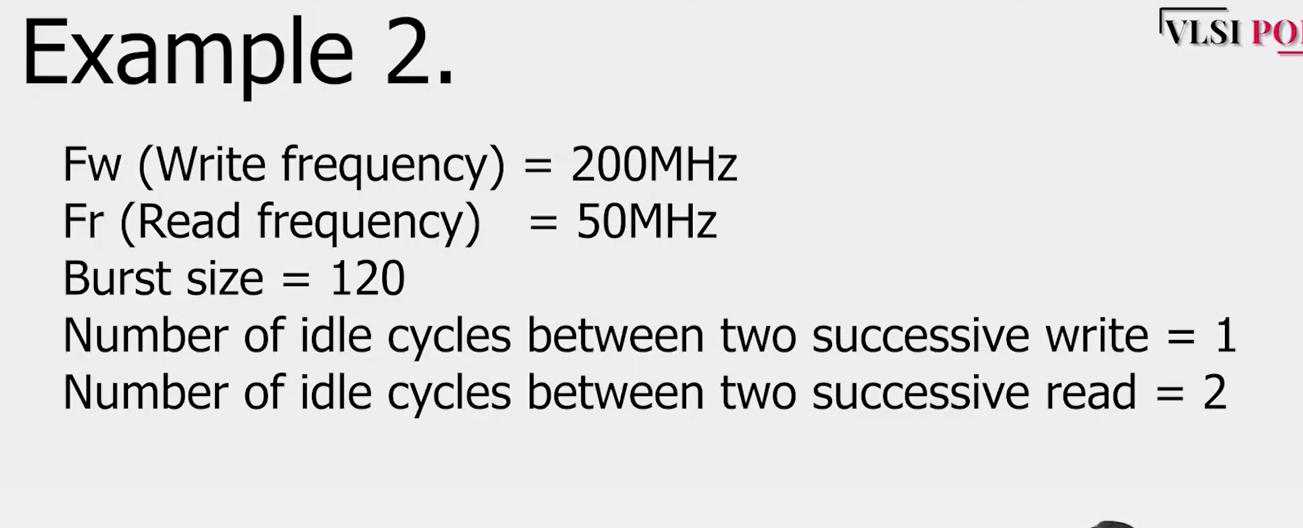
It is important to understand that depth of FIFO should not be less than required, since it involves loss of information.

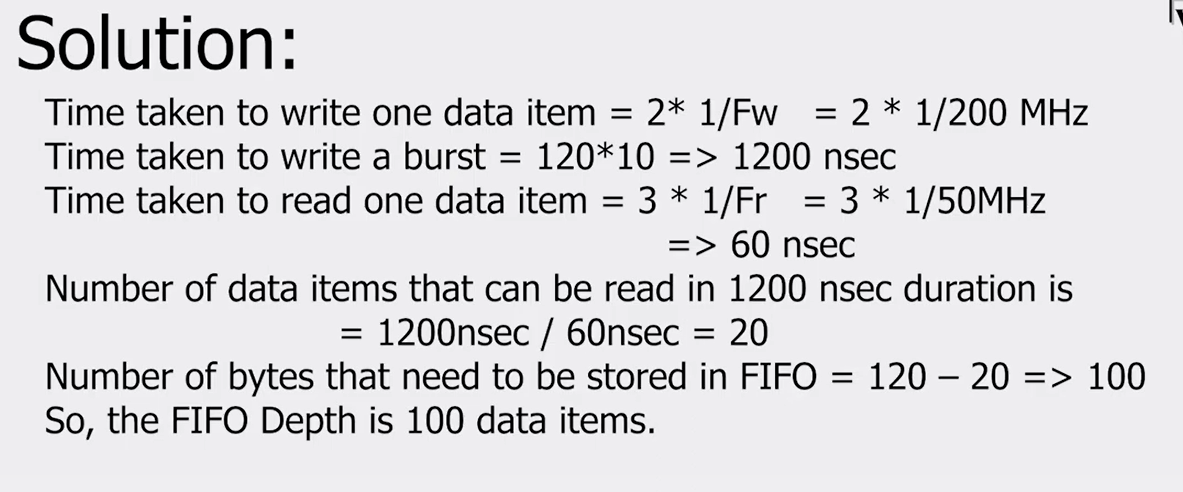
In the case stated,100 MHz should be the depth of FIFO to avoid loss of information. If it is less, we will lose the data and if it is more, it is not economical.

The data to the FIFO should be sent in the form of Burst rather than continuous format so that we can understand when the FIFO gets full.





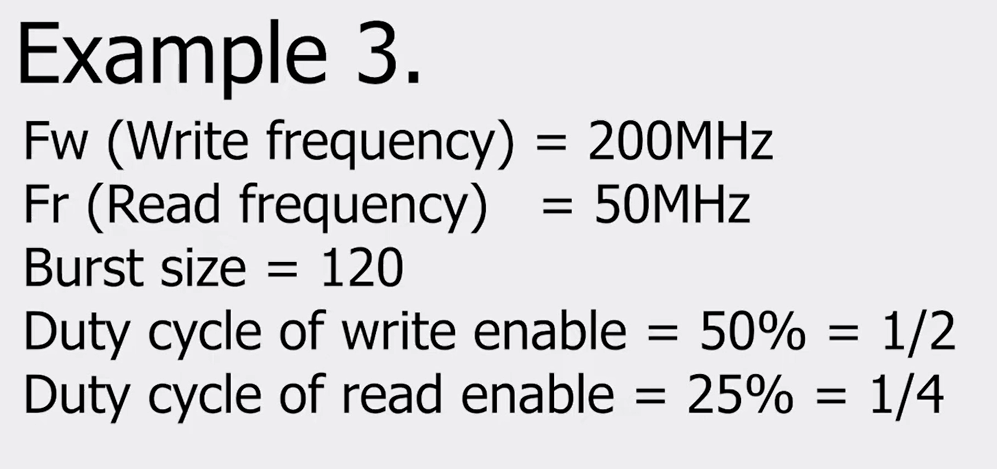




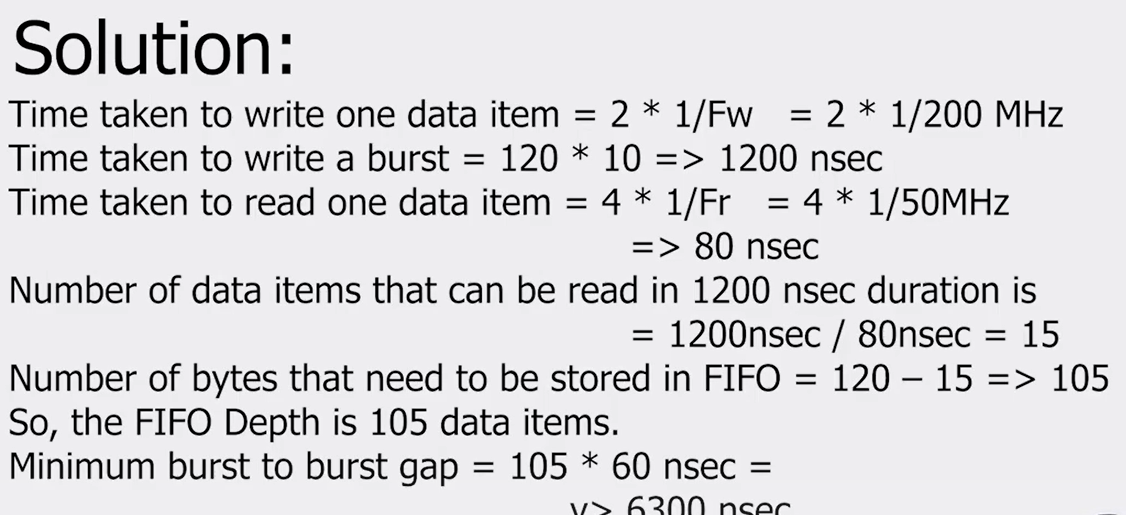
When we say 1 idle cycle between 2 successive write operations,

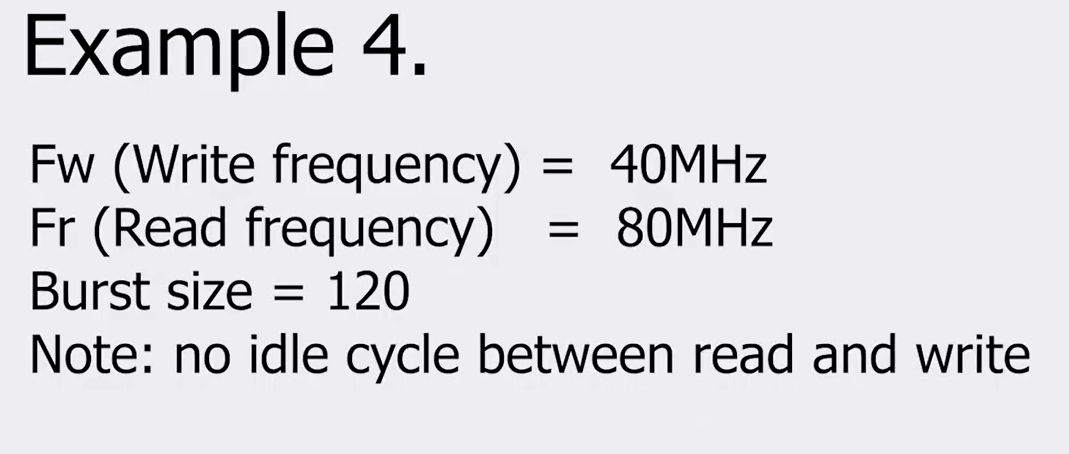
It means that once a write operation is performed,an idle cycle exists and then another write operation takes place. So,it is basically 2 \*1/f

So,in general it is (idle cycle+1)\*1/f

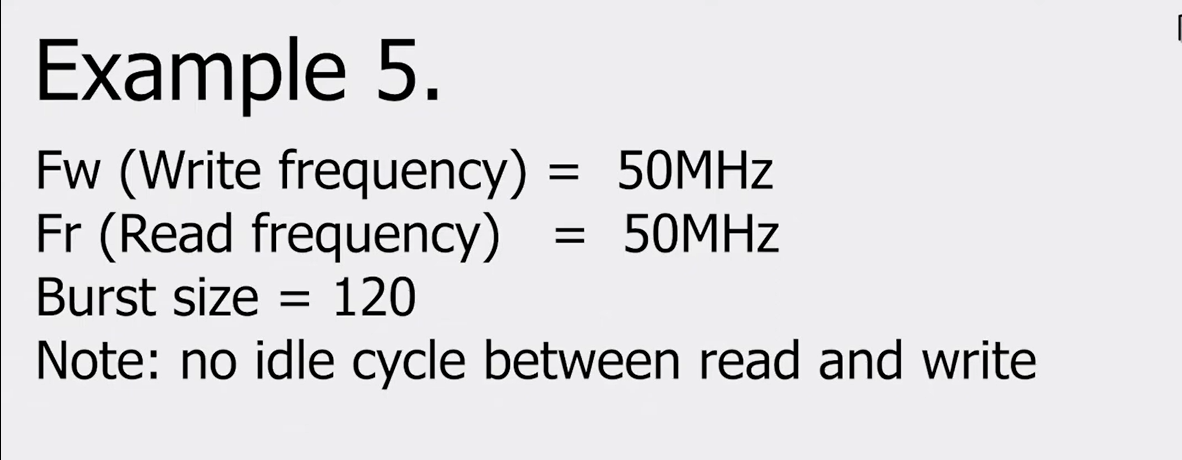


Duty cycle means cycles between 2 successive read or write operations.





Here the read frequency is greater than write frequency,so there is a need of FIFO of depth 1.



Since read frequency is same as write frequency, there is no need of FIFO provided they have same phase.

$fopen(“\_filename”\_);

//opens file in read mode

$fopen(“filename”,”w”);

//opens file in write mode.

module randomno;

integer random\_no;

initial

begin

random\_no= $random%100;

$display(random\_no);

end

endmodule

//just change the code for random no between 40 &50;

integer random\_no;

initial

begin

repeat(5)

begin

#1

random\_no=40+ $random(50-40);

$display(“random no generated is %d”,random\_no);

end

end

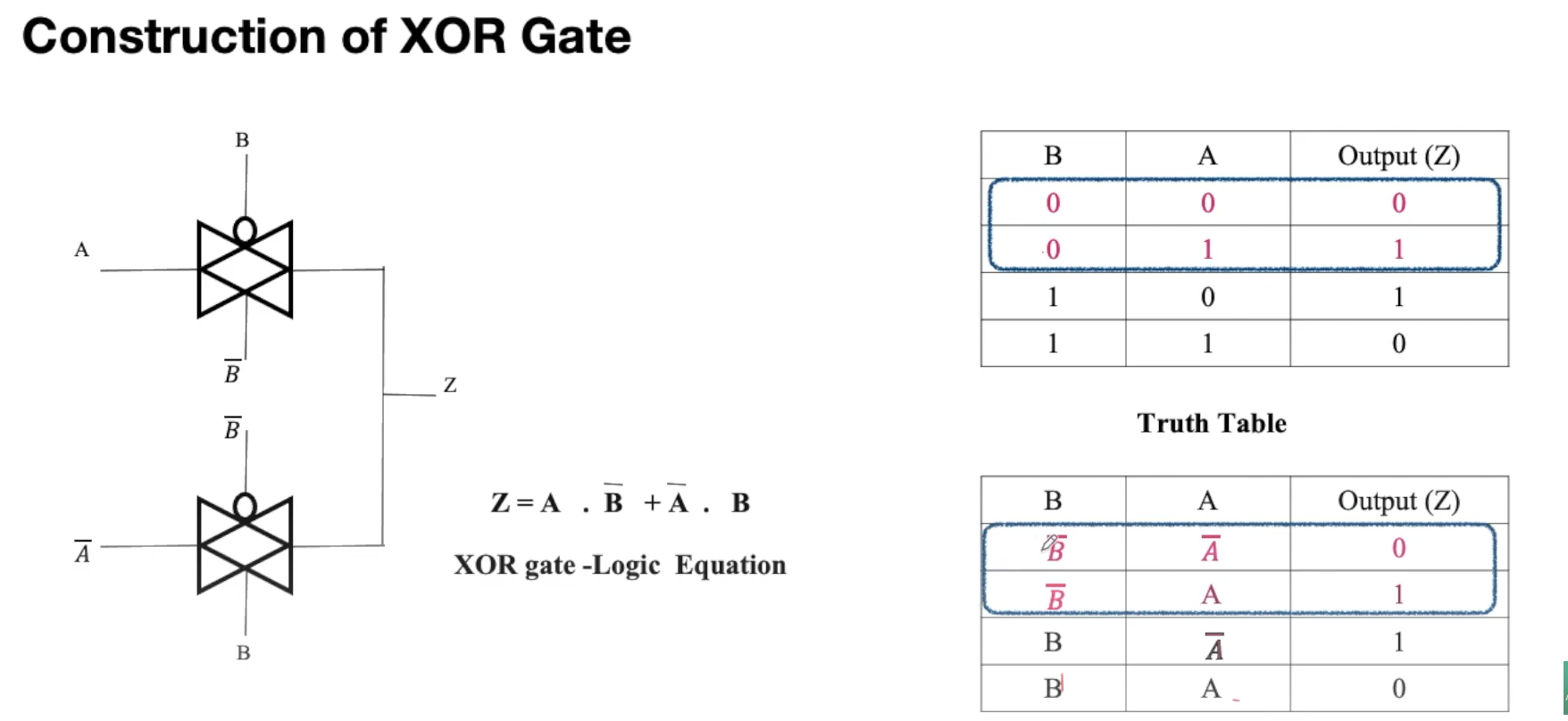
endmodule

synchronous vs asynchronous reset

synchronous reset occurs across the clock edge; which means that it occurs only during the rising or falling edge of the clock and it helps in avoiding glitches.

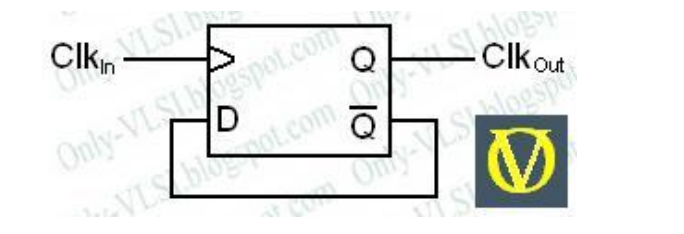
It is important where timing is an important factor.

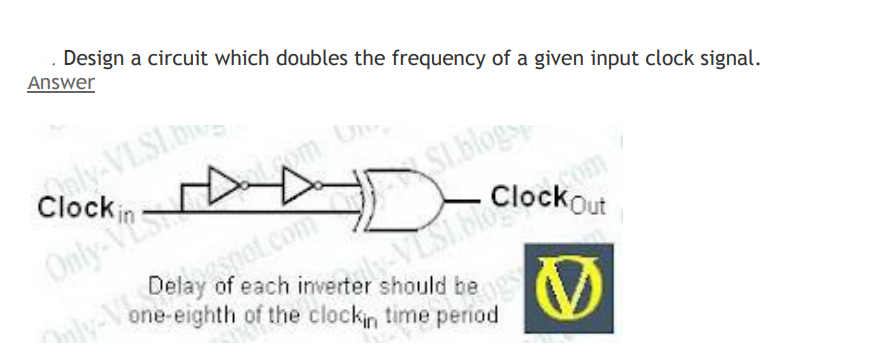
Asynchronous reset -can occur at anytime independent of the clock edge.It leads to metastability issues.Can be used where simplicity and speed matters rather than timing control.



Frequency divider circuit

Frequency divided by 2;





Arrange The Following In The Increasing Order Of Their Complexity: Fpga,pla,cpld,buddy?

Increasing order of complexity: PLA, PAL, CPLD, FPGA.

<https://chat.openai.com/share/98d13920-eb1b-440a-8c65-cfafdf656dc4>

STA,VLSI midha,projects gurinchi, Verilog meedha parallel cases,synthesis vs normal,implementation

Rtl design,simulation and synthesis

Just 15 minutes

Digital design,rtl,sta



CMOS is favored for low power dissipation, while ECL is known for high-speed performance, TTL is reliable and versatile, and DTL is an older logic family with less widespread use in modern applications.

Top of Form

Certainly! Here's a simple comparison table:

| Logic Family | Power Dissipation | Speed | Common Applications |

|--------------|-------------------|-------|----------------------|

| CMOS | Low | Moderate | General-purpose, low-power applications |

| ECL | High | High | High-speed communication systems |

| TTL | Moderate to High | Moderate | Microcontrollers, digital integrated circuits |

| DTL | Moderate to High | Low | Older, less commonly used in modern applications |

This table provides a quick overview of their power dissipation, speed, and common applications, making it easier to memorize the key characteristics of each logic family.

If input frequency is given,and if asked for output frequency,we need to divide the i/p frequency by 2^n where n represent no of flip flops required.

o/p frequency= i/p frequency/(2^n)

In the context of VLSI (Very Large Scale Integration), the terms "bi-stable multivibrator," "astable multivibrator," and "monostable multivibrator" refer to different types of digital circuits, specifically flip-flops and timers, which are essential building blocks in digital electronics.

1. \*\*Bi-stable Multivibrator:\*\*

- \*\*Definition:\*\* A bi-stable multivibrator is a digital circuit with two stable states. It is commonly known as a flip-flop.

- \*\*Function in VLSI:\*\* In VLSI, flip-flops are crucial for storing a single bit of information. They are used as memory elements in sequential circuits, allowing the storage of state information.

- \*\*Applications:\*\* Flip-flops are extensively used for storing and transferring data within digital systems. They play a fundamental role in the design of sequential circuits, including registers and memory cells.

2. \*\*Astable Multivibrator:\*\*

- \*\*Definition:\*\* An astable multivibrator is a digital circuit that oscillates continuously between its two unstable states without external input. It generates a square wave output.

- \*\*Function in VLSI:\*\* Astable multivibrators are often used as clock generators in digital systems. They provide a continuous square wave signal, serving as a clock source for synchronous digital circuits.

- \*\*Applications:\*\* Clock generators are essential for synchronizing various components within a digital system. Astable multivibrators are commonly used for this purpose, providing a stable and periodic clock signal.

3. \*\*Monostable Multivibrator:\*\*

- \*\*Definition:\*\* A monostable multivibrator is a digital circuit with one stable state and one unstable state. It is triggered externally to enter the unstable state and then returns to the stable state after a predefined time.

- \*\*Function in VLSI:\*\* Monostable multivibrators are used for generating pulse-width-modulated signals or for generating single-pulse events in response to external triggers.

- \*\*Applications:\*\* In VLSI, monostable multivibrators are used in applications where a single, controlled pulse needs to be generated in response to a specific input trigger. This can be useful in various scenarios, such as signal conditioning or event synchronization.

In summary, these three types of multivibrators (bi-stable, astable, and monostable) serve different functions in VLSI, contributing to the design and operation of digital circuits.

Universal shift registers- shifting (both right and left )and parallel load only.

Data change from spatial to temporal and vice versa is achieved using shift registers.

Hello everyone!  
Note: Because of character limits, I've only included topics and my experience here. You can find topic wise detailed information in the comment section.  
I've been getting a lot of questions about the important subjects and topics for interviews. Here's the answer specifically for hardware profiles. Keep in mind that for software profiles, there might be additional and different topics, such as coding questions and computer network inquiries.   
  
The crucial thing is to build a good connection and vibe with them. Try to make it feel like you're there for a group discussion about technical matters.Naturally it began with introduction and in that I made sure to highlight my strengths right from the start, especially focusing on my 'XYZ' project. This project was significant to me as I personally worked on it and applied my skills practically. It caught their attention, leading to a 20-minute discussion. Successfully grabbing their attention meant I was familiar with about 40% of the interview questions beforehand. This is a strategy to guide the direction of the interview.  
Here are the topics you need to know:  
  
1. Project Discussion  
2. Digital Electronics  
3. Static Timing Analysis (STA)  
4. COA (Computer Organization and Architecture)  
5. Basics of VLSI  
  
I want to highlight that it's okay not to know something. In such cases, it's important to be honest and admit if you haven't encountered a topic before. I did this during the interview, stating clearly that I hadn't touched on that topic before. I offered to try solving it with their guidance. They then explained the importance of buffer insertion and taught me how to determine the number of buffers needed for transmission.  
So, always remember to be honest and avoid pretending to know everything.  
Towards the end, I asked for feedback on areas for improvement and what they thought I should learn. I also inquired about any incorrect answers I provided. I made it clear that if I didn't know the answer, I usually admitted it. They mentioned that I answered all the questions correctly, but they noticed I was often in a hurry to respond. They emphasized that as an engineer, providing an effective solution should be the priority. For instance, if I design a circuit using a multiplexer, I should prioritize using the minimum number of multiplexers, whether specified in the problem statement or not.  
  
I've shared as much as I can recall here. I've spent valuable time creating this content. If you find it helpful, please share it with someone who needs it. As my mother says, sharing knowledge with those who need it helps increase our own knowledge. ALL THE BEST!

1. Project Discussion:  
During the interview, we discussed one of my solo projects. They asked me to explain the project and had several questions to confirm if I completed it independently. Then, they inquired about the challenges I faced during the project and how I overcame them.

2. Digital Electronics: As you are aware, digital electronics forms the foundation of hardware. It is crucial not to overlook even the simplest topics. In my case, they presented a problem statement and requested me to design a circuit using only a multiplexer (mux). After successfully solving that question, they challenged me to redesign the same circuit using the minimum number of mux units. Following this, they presented a different problem statement and asked me to design a circuit incorporating the elements from the previous circuit.  
The second question involved designing a state machine capable of detecting two sequences simultaneously from a single input string.

3. Static Timing Analysis (STA): They give you a circuit and ask you to figure out certain timing details, like setup time, Hold time, Setup margin, maximum frequency, etc. Then, they ask questions about how you can make the circuit better, for example how we can improve setup and hold margin. They might ask more questions related to that specific circuit.

4. COA (Computer Organization and Architecture):  
In this part of the interview, they questioned me about cache replacement policies. They gave me some data and asked me to explain, step by step, how the data should be stored under different policies. They inquired about two different policies.  
The second question was about memory mapping and how one method differs from another. They also asked me about the advantage of one method over the other in terms of speed.

5. Basics of VLSI:  
This section covers what parameters are essential for a good circuit, along with questions about buffer insertion and worst-case scenarios. I was given a problem statement where I had

to find the minimum number of buffers needed for lossless transmission in a circuit.

In C, operator precedence determines the order in which operators are evaluated in an expression. Operators with higher precedence are evaluated before operators with lower precedence. If operators have the same precedence, the associativity of the operators determines the order of evaluation (left-to-right or right-to-left).

Here's a summary of some common operators in C, listed in decreasing order of precedence (operators at the top have higher precedence):

1. \*\*Postfix operators:\*\* `()`, `[]`, `->`, `.` (left to right)

2. \*\*Unary operators:\*\* `+`, `-`, `++`, `--`, `!`, `~`, `sizeof`, `(type)`, `&`, `\*`

3. \*\*Multiplicative operators:\*\* `\*`, `/`, `%` (left to right)

4. \*\*Additive operators:\*\* `+`, `-` (left to right)

5. \*\*Shift operators:\*\* `<<`, `>>` (left to right)

6. \*\*Relational operators:\*\* `<`, `<=`, `>`, `>=` (left to right)

7. \*\*Equality operators:\*\* `==`, `!=` (left to right)

8. \*\*Bitwise AND operator:\*\* `&` (left to right)

9. \*\*Bitwise XOR operator:\*\* `^` (left to right)

10. \*\*Bitwise OR operator:\*\* `|` (left to right)

11. \*\*Logical AND operator:\*\* `&&` (left to right)

12. \*\*Logical OR operator:\*\* `||` (left to right)

13. \*\*Conditional operator (ternary):\*\* `? :` (right to left)

14. \*\*Assignment operators:\*\* `=`, `+=`, `-=`, `\*=`, `/=`, `%=` etc. (right to left)

15. \*\*Comma operator:\*\* `,` (left to right)

Parentheses `()` can be used to override the default precedence and explicitly specify the order of evaluation. Expressions enclosed in parentheses are evaluated first.

For example:

```c

int result = 2 + 3 \* 4; // result = 14 (multiplication has higher precedence than addition)

int another = (2 + 3) \* 4; // another = 20 (parentheses force addition to be done first)

```

Understanding operator precedence is crucial for writing correct and predictable expressions in C. Always consult the C language specification or a reliable reference for the complete and detailed list of operator precedences.

ASCII values A-Z : 65-90;

a-z:97-122;

\n=10;

1. *main()*

*{*

*printf("\nab");*

*printf("\bsi");*

*printf("\rha");*

*}*

Answer: hai

Explanation:

In new line, ab gets printed.

\b’-backspace,it moves cursor one step back;

Initially cursor position (^) is after ab

ab^

one step back word; cursor is at position b;

overriding with si

asi;

now using \r – linefeed/carriage return character moves cursor to the first position in same line;

asi;

Overriding with ha;

hai -final answer.

Certainly! Below are Verilog code snippets demonstrating the usage of `localparam`, `defparam`, and `specparam`.

1. \*\*localparam\*\*: `localparam` is used to define parameters local to a module. These parameters cannot be overridden or modified outside the module.

```verilog

module example\_module(

input logic [7:0] data\_in,

output logic [7:0] data\_out;

localparam WIDTH = 8; // Local parameter declaration

// Define logic based on localparam WIDTH

assign data\_out = data\_in + WIDTH;

endmodule

```

2. \*\*defparam\*\*: `defparam` is used to set parameter values in an instance of a module.

```verilog

module top\_module;

// Module instance

example\_module inst1(

.data\_in(data\_in),

.data\_out(data\_out)

);

logic [7:0] data\_in;

logic [7:0] data\_out;

// Using defparam to set parameter value for instance

defparam inst1.WIDTH = 16;

endmodule

```

3. \*\*specparam\*\*: `specparam` is used to define a special parameter that can change its value based on the context in which it is used.

```verilog

module specparam\_example(

input logic [7:0] data\_in,

output logic [7:0] data\_out

);

// Define specparam

specparam DELAY = 10;

// Using specparam to create a delay

always @(posedge clk) begin

#(DELAY) data\_out <= data\_in;

end

endmodule

```

These examples illustrate the usage of `localparam`, `defparam`, and `specparam` in Verilog. Each serves a different purpose and can be handy in different design scenarios.

**for Binary counter toggle power would be  
(2×((2^n) -1) – n) P  
for Grey counter toggle power would be  
((2^n) -1)P**

<https://www.linkedin.com/pulse/how-prepare-vlsi-profile-interview-vivek-arya/>

module mealy\_fsm(input clk,reset,x,output z);

parameter A=1;

parameter B=2;

parameter C=3;

parameter D=4;

reg state,next\_state;

always@(posedge clk,negedge reset)

begin

if (!reset)

begin

state<=A;

end

else

state<=next\_state;

end

always @(state or x)

begin

case(state)

A:  
 begin

if(x==0)

next\_state<=A;

else

next\_state<=B;

B:

begin

if (x==0)

next\_state<=C;

end

else

next\_state<=B;

so on

default: next\_state<=A;

end

endcase

end

assign z=(state==D)&&(x==0)?1:0;

endmodule

The code would be the same for Moore FSM except for the fact that there exists one more state than that of Mealy FSM.

In general,Moore FSM has more states than that of Mealy FSM.

Module fifo(input clk,reset,wr,re,[31:0]data\_in,output empty,full,[31:0]data\_out);

reg [4:0]wr\_pt,re\_pt;

reg [31:0]mem[15:0];

integer i;

always@(posedge clk,negedge reset)

begin

if (reset)

begin

wr\_pt<=0;

re\_pt<=0;

for(i=0;i<16;i=i+1)

begin

mem[i]<=0;

end

end

else if(wr==1 &&full==0)

begin

mem[wr\_pt]<=data\_in;

wr\_pt<=wr\_pt+1;

end

else if (re==1 and empty==0)

begin

data\_out<=mem[re\_pt];

re\_pt<=re\_pt+1;

end

end

end

assign full=(wr\_pt==5’b11111)?1:0;

assign empty=(re\_pt==5’b11111)?1:0;

endmodule

module bi\_directional(input clk,reset, inout [7:0]data,direction);

reg [7:0]data\_buffer;

reg direction;