verilog code for accumulator Latest Edition

Verilog code for accumulator is a fundamental design pattern in digital systems, especially in applications involving signal processing, digital filtering, and data aggregation. An accumulator in hardware is a register or circuit that sums input values over time, maintaining a running total that can be used for various computational purposes. Writing efficient and reliable Verilog code for an accumulator is essential for engineers working on FPGA or ASIC designs. This article provides a comprehensive overview of Verilog code for an accumulator, exploring its structure, features, and best practices to help you implement effective accumulator modules in your digital designs.

Understanding the Basics of an Accumulator in Verilog

What is an Accumulator?

An accumulator is a circuit that continuously adds incoming data to a stored total, updating its value with each clock cycle or trigger event. It is widely used in digital signal processing for summing samples, implementing filters, or counting events.

Key Features of a Verilog Accumulator

- Input Data: The value to be added.
- Clock Signal: Synchronizes the updates.
- Reset Signal: Clears the accumulator to an initial state.
- Enable Signal: Controls when the accumulator updates.
- Saturation Logic (Optional): Prevents overflow by capping the maximum/minimum value.

Basic Verilog Code for a Simple Accumulator

Before diving into more complex features, here is a simple example of Verilog code for an accumulator:

```verilog

module simple_accumulator (

input wire clk,

input wire reset,
```

```
input wire enable,

input wire [7:0] data_in,

output reg [15:0] sum

);

always @(posedge clk or posedge reset) begin

if (reset) begin

sum
end else if (enable) begin

sum
end

end

endmodule
```

This code defines a basic accumulator that sums 8-bit data inputs, maintaining a 16-bit sum to prevent overflow.

Design Considerations for an Effective Verilog Accumulator

Data Width and Overflow Handling

Choosing appropriate data widths for input and accumulated sum is crucial. If the sum exceeds the maximum value representable, overflow occurs, potentially leading to incorrect results.

- **Data Widths:** Match or exceed expected maximum sum to prevent overflow.
- **Saturation Logic:** Implement logic to clamp the sum at maximum/minimum values if overflow is undesirable.

Synchronization and Timing

Ensure that the accumulator updates are synchronized with the system clock and that signals such as reset and enable are properly timed to avoid metastability.

Reset and Initialization

Use asynchronous or synchronous reset depending on your application. Asynchronous resets are faster but may cause glitches; synchronous resets are safer for timing.

Efficiency and Resource Optimization

Design your accumulator to minimize hardware resource usage while maintaining performance.

Advanced Verilog Accumulator Features

Saturation Logic for Overflow Prevention

Implement saturation logic to prevent the accumulator from overflowing:

```verilog

module saturated_accumulator (

input wire clk,

input wire reset,

input wire enable,

input wire [7:0] data_in,

output reg [15:0] sum

);

parameter MAX_VALUE = 16'hFFFF;

always @(posedge clk or posedge reset) begin

if (reset) begin

sum
```

end else if (enable) begin

if (sum + data_in > MAX_VALUE) begin

sum
end else begin

sum
end

end

end

endmodule

```

This implementation ensures the sum does not overflow the 16-bit register.

## Signed vs Unsigned Accumulators

Depending on your application, your accumulator may need to handle signed data.

- Unsigned Accumulator: Simple addition, no sign consideration.
- Signed Accumulator: Uses two?s complement representation, requiring signed addition.

Example of a signed accumulator:

```verilog

module signed_accumulator (

input wire clk,

input wire reset,

input wire enable,

input wire signed [7:0] data_in,

```
output reg signed [15:0] sum

);

always @(posedge clk or posedge reset) begin

if (reset) begin

sum
end else if (enable) begin

sum
end

end

endmodule
```

Best Practices for Writing Verilog Code for Accumulators

Modular Design

- Write reusable modules with clear interfaces.
- Use parameters for data widths and maximum values.

Simulation and Testing

- Verify accumulator behavior under various input sequences.
- Test boundary conditions such as maximum input values and reset operation.

Documentation and Comments

- Clearly comment your code, explaining logic and parameters.
- Maintain readability for future modifications.

Application Examples of Verilog Accumulators

Digital Filters

Accumulators are core components in FIR filters, where they sum weighted input samples.

Data Counters and Event Summation

Count occurrences of events or sum data streams in real-time systems.

Signal Processing and DSP

Implement integral parts of digital signal processing pipelines.

Conclusion

Designing an efficient and reliable accumulator in Verilog requires understanding of fundamental digital design principles, careful data width management, and appropriate handling of overflow and sign considerations. Whether you're implementing simple summing circuits or complex digital filters, leveraging best practices in Verilog coding ensures your accumulator performs accurately and efficiently. By following the examples and guidelines presented in this article, you can develop robust Verilog modules tailored to your specific application needs.

For further learning, explore advanced topics like pipelined accumulators, multi-channel aggregation, and integration with other digital system components to enhance your digital design expertise.

Verilog code for accumulator: An In-Depth Exploration of Digital Summation Hardware

In the realm of digital design and embedded systems, accumulators serve as fundamental building blocks for a multitude of applications?from digital signal processing (DSP) and control systems to arithmetic logic units and programmable logic devices. At their core, accumulators are specialized registers that continuously sum input values over time, enabling computations such as running totals, iterative calculations, and complex mathematical operations. When translating these concepts into hardware, Verilog?a hardware description language (HDL)?becomes an invaluable tool. This article offers a comprehensive analysis of Verilog code for accumulators, exploring their architecture, implementation techniques, and best practices for designing efficient, reliable hardware modules.

Understanding the Role of an Accumulator in Digital Systems

Definition and Functionality

An accumulator is essentially a register that retains a sum of input values over successive clock cycles. Each cycle, the accumulator adds a new input to its stored total and updates its stored value accordingly. Its primary function is to perform iterative addition, making it indispensable in applications like digital filters, counters, and mathematical computation units.

Key characteristics of an accumulator include:

- Sequential operation: The addition occurs synchronously with the clock signal.
- State retention: The accumulator maintains its sum across multiple clock cycles until reset or cleared.
- Input variability: Inputs can be dynamic, enabling real-time calculations.

Applications of Accumulators

Understanding the significance of accumulators requires examining their diverse applications:

- Digital Signal Processing (DSP): Used in filters, Fourier transforms, and convolution operations where continuous summation of signal samples is needed.
- Control Systems: Employed in integral components of PID controllers to compute accumulated error.
- Statistics and Data Analysis: Calculations of running totals, averages, or variance.
- Arithmetic Units: Building blocks for more complex arithmetic operations like multiplication and division.

Design Considerations for a Verilog Accumulator

Creating an effective accumulator module involves multiple considerations, including data width, timing, reset behavior, and resource utilization.

Key Design Parameters

- Bit-width: Determines the maximum value the accumulator can store without overflow. For example, an 8-bit accumulator can handle sums up to 255 (unsigned).
- Input Data Width: Should be compatible with the accumulator's capacity to prevent overflow or data loss.
- Overflow Handling: Strategies include saturation, wrap-around, or signaling an overflow condition.
- Reset Behavior: Defines how the accumulator is cleared?either asynchronously or

synchronously.
- Pipelining: For high-speed applications, pipelining may be implemented to improve throughput.

Trade-offs in Design

Designers must balance resource usage, speed, and accuracy:

- Increasing bit-width improves range but consumes more hardware.
- Adding overflow detection adds complexity but enhances reliability.
- Synchronous resets simplify design but may introduce latency.

Verilog Implementation of an Accumulator

A typical Verilog code for an accumulator encapsulates the above considerations into a hardware module. Below, we explore a basic implementation, its components, and enhancements.

Basic Verilog Code for a Simple Accumulator

```verilog

module accumulator (

input wire clk,

input wire reset,

input wire [DATA_WIDTH-1:0] data_in,

output reg [ACC_WIDTH-1:0] sum

);

parameter DATA_WIDTH = 8; // Width of input data

parameter ACC_WIDTH = 16; // Width of accumulator to prevent overflow

always @(posedge clk) begin

if (reset) begin
```

```
sum
end else begin

sum
end

end

endmodule
```

Explanation:

- Module Ports:
- `clk`: The clock signal for synchronization.
- `reset`: Resets the accumulator to zero.
- `data_in`: Input data to be added.
- `sum`: Output holding the accumulated total.

- Parameters:
- Allows flexibility in defining data and accumulator widths.

- Behavior:
- On each rising edge of the clock, if reset is active, the sum clears.
- Otherwise, the sum updates by adding the current input.

Features and Enhancements

While the above code provides a foundational accumulator, practical applications often require additional features:

- Overflow Detection:

```verilog
reg overflow_flag;
```

always @(posedge clk) begin

if (reset) begin

sum
overflow_flag
end else begin

{overflow_flag, sum}
end

end

```

- Saturation Arithmetic:

To prevent wrap-around, the accumulator can be designed to saturate at maximum value:

```verilog

reg [ACC_WIDTH-1:0] max_value = {ACC_WIDTH{1'b1}};

always @(posedge clk) begin

if (reset) begin

sum
end else begin

if (sum + data_in > max_value) begin

sum
end else begin

sum
end

end

end

```

- Pipelined Accumulator:

For high-speed systems, pipeline registers can be inserted to break combinational paths, improving throughput.

Advanced Topics in Verilog Accumulator Design

Handling Signed and Unsigned Data

Designs must distinguish between signed and unsigned numbers, affecting addition and overflow behavior.

```verilog

// Signed accumulator example

reg signed [ACC_WIDTH-1:0] sum_signed;

always @(posedge clk) begin

if (reset) begin

sum_signed
end else begin

sum_signed
end

end

```

Multi-Input Accumulators

Some applications require summing multiple inputs simultaneously or over different channels. This can be achieved by extending the module:
```

- Parallel Accumulators: Multiple accumulators operating concurrently.
- Weighted Accumulation: Incorporating coefficients for each input.

Memory Considerations and Efficient Hardware Mapping

- Resource Utilization: Larger bit-widths demand more flip-flops and logic.
- Optimization: Use of carry-lookahead adders or embedded DSP slices in FPGA fabric for efficient summation.
- Power Consumption: Pipelining and clock gating can reduce power.

Testing and Verification of Verilog Accumulator Modules

Thorough testing is crucial to ensure the reliability of the accumulator.

Common Verification Steps:

- Simulation:
- Test for correct sum accumulation over multiple cycles.
- Check reset functionality.
- Verify overflow and saturation logic.
- Simulate signed and unsigned inputs.

- Formal Verification:
- Use assertions to guarantee the sum does not exceed expected limits.
- Validate overflow detection mechanisms.

- Hardware Testing:
- Implement on FPGA or ASIC prototypes.
- Use signal analyzers to monitor correctness in real-time.

Conclusion: Best Practices and Design Tips

Designing a robust Verilog accumulator requires careful planning:

- Choose appropriate bit-widths to balance range and resource constraints.
- Implement overflow detection to prevent silent errors.

- Incorporate reset logic for predictable startup behavior.
- Optimize for speed or area based on application needs?pipelining for high throughput, resource sharing for minimal footprint.
- Test extensively with diverse scenarios to ensure reliability.

Accumulators are a cornerstone in digital design, and their effective implementation in Verilog empowers engineers to develop complex, high-performance systems. As FPGA and ASIC technologies evolve, so do the opportunities for innovative accumulator architectures?ranging from simple, low-power modules to sophisticated, high-speed units with adaptive features. Mastery of Verilog coding for accumulators not only enhances the designer?s toolkit but also paves the way for advancing digital computation in myriad applications.

References

- Roth, C. H., & Kinney, L. (2004). Fundamentals of Logic Design. Thomson.
- Harris, D., & Harris, S. (2012). Digital Design and Computer Architecture. Morgan Kaufmann.
- Xilinx and Intel FPGA documentation on DSP slices and optimized adder circuits.
- OpenCores.org HDL modules and community resources for accumulator designs.

Author?s Note: Whether you are developing a signal processor, a control algorithm, or a custom arithmetic unit, understanding the nuances of Verilog accumulator design is essential. This guide aims to provide a solid foundation, inspiring further exploration and innovation in digital hardware design.

QuestionAnswer What is a Verilog accumulator module and how is it typically used? A Verilog accumulator module sums a sequence of input values over time, often used in digital signal processing, counters, or integration tasks. It stores the running total in a register and updates it with each clock cycle. Can you provide a simple Verilog code example for an 8-bit accumulator? Yes, here's a basic example: ```verilog module accumulator (input wire clk, input wire reset, input wire [7:0] data_in, output reg [15:0] sum); always @(posedge clk or posedge reset) begin if (reset) sum = MAX_VALUE) begin sum

Related keywords: Verilog, accumulator, digital design, HDL, FPGA, ASIC, counter, register, sequential logic, hardware description language

VERILOG MULTIPLE CHOICE QUESTIONS WITH ANSWERS

Verilog Multiple Choice Questions with Answers

Verilog is a hardware description language (HDL) widely used for designing and simulating digital systems. Whether you're a student preparing for exams, a professional verifying designs, or an enthusiast enhancing your knowledge, practicing multiple-choice questions (MCQs) on Verilog can significantly improve your understanding. This comprehensive guide presents a collection of Verilog MCQs with answers, structured to cover fundamental concepts, syntax, simulation, and advanced topics related to Verilog. Dive in to test your knowledge and reinforce your learning.

Introduction to Verilog MCQs

Understanding Verilog through MCQs helps in quick assessment of knowledge, identifying weak areas, and preparing effectively for interviews or exams. The questions included range from basic syntax to complex design constructs, ensuring a well-rounded grasp of the language.

Basic Concepts of Verilog

1. What is Verilog primarily used for?

- A) Software development
- B) Hardware description and simulation
- C) Web development
- D) Database management

Answer: B) Hardware description and simulation

2. Which of the following is a valid Verilog module declaration?

- A) module MyModule;
- B) module MyModule();
- C) module MyModule(input a, b);
- D) All of the above

Answer: D) All of the above

3. In Verilog, what is the primary purpose of the 'initial' block?

- A) To define combinational logic
- B) To specify the start-up conditions and testbench stimuli
- C) To declare variables

- D) To synthesize hardware

Answer: B) To specify the start-up conditions and testbench stimuli

Data Types and Variables in Verilog

4. Which data type is used for storing a 4-bit binary number in Verilog?

- A) reg [3:0]
- B) wire [3:0]
- C) bit4
- D) Both A and B

Answer: D) Both A and B

5. What is the default data type for a variable declared without a type in Verilog?

- A) reg
- B) wire
- C) integer
- D) reg or wire depending on context

Answer: D) reg or wire depending on context

Verilog Operators and Expressions

6. Which operator is used for logical AND in Verilog?

- A) &&
- B) &
- C) and
- D) both A and C

Answer: D) both A and C

7. What is the result of the expression 3'b101 & 3'b110?

- A) 3'b100
- B) 3'b111
- C) 3'b100
- D) 3'b110

Answer: A) 3'b100

Design Constructs and Behavioral Modeling

8. Which Verilog construct is used to model sequential logic?

- A) always @()
- B) initial
- C) always @(posedge clk)
- D) assign

Answer: C) always @(posedge clk)

9. Which statement is used to assign values in a procedural block?

- A) assign
- B)
- C) =
- D) Both B and C

Answer: D) Both B and C

Simulation and Testbenches

10. What is the purpose of a testbench in Verilog?

- A) To synthesize hardware
- B) To verify and simulate the design without hardware
- C) To generate reports
- D) To compile Verilog code

Answer: B) To verify and simulate the design without hardware

11. Which of the following is a common way to initialize signals in a testbench?

- A) Using 'initial' block
- B) Using 'always' block
- C) Using 'assign' statement
- D) Using 'task'

Answer: A) Using 'initial' block

Advanced Topics in Verilog

12. What is the difference between 'blocking' (=) and 'non-blocking' (

- **A) Blocking assignments execute immediately, non-blocking are scheduled**
- **B) Blocking are used for combinational logic, non-blocking for sequential logic**
- **C) Both A and B**
- **D) None of the above**

Answer: C) Both A and B

13. Which Verilog construct is used for parameterized modules?

- **A) `parameter**
- **B) `define**
- **C) `localparam**
- **D) All of the above**

Answer: D) All of the above

14. In Verilog, what is a 'generate' block used for?

- **A) To generate repetitive hardware structures**
- **B) To create conditional code**
- **C) To instantiate multiple modules**
- **D) All of the above**

Answer: D) All of the above

Common Mistakes and Tips for Verilog MCQs

- Carefully read the question to understand whether it asks about syntax, semantics, or best practices.
- Remember that 'reg' and 'wire' serve different purposes; 'reg' can store values in procedural blocks, while 'wire' is used for continuous assignments.
- Understand the difference between blocking (=) and non-blocking (
- Practice writing small Verilog modules and testbenches to strengthen conceptual understanding.
- Use simulation tools like ModelSim or Vivado to verify your answers practically.

Conclusion

Mastering Verilog multiple choice questions with answers enhances your comprehension of digital design principles and Verilog syntax. Regular practice with MCQs helps you familiarize yourself with common interview questions, exam patterns, and real-world design challenges. Remember, understanding the concepts behind each question is crucial, so use this guide not just for rote memorization but for building a solid foundation in Verilog HDL.

Whether you're a beginner or an experienced engineer, continuously challenging yourself with MCQs is an effective way to keep your skills sharp and stay updated with best practices in hardware description language coding. Keep practicing, explore more advanced topics, and leverage simulation tools to complement your theoretical knowledge.

Verilog Multiple Choice Questions with Answers: An Expert Review

In the realm of digital design and hardware description languages (HDLs), Verilog stands out as one of the most widely adopted tools for modeling, simulating, and synthesizing digital circuits. As students, engineers, and designers navigate the complexities of Verilog, mastering its concepts through practice questions becomes an essential step toward proficiency. This article provides an in-depth exploration of Verilog multiple choice questions (MCQs) with answers, serving as a comprehensive guide for learners aiming to strengthen their understanding and assessment readiness.

Understanding the Significance of Verilog MCQs

Multiple choice questions serve as an efficient method to evaluate knowledge across a broad spectrum of topics within Verilog. They are particularly effective because they:

- Test Conceptual Clarity: MCQs often focus on fundamental principles, encouraging learners to understand core ideas rather than rote memorization.
- Facilitate Self-Assessment: Students can quickly gauge their comprehension and identify areas needing further study.
- Prepare for Certification and Exams: Many industry certifications and academic evaluations include MCQ formats, making practice essential.
- Encourage Critical Thinking: Well-designed MCQs challenge students to analyze choices, fostering deeper understanding.

Core Topics Covered in Verilog MCQs

To structure effective MCQs, it?s crucial to identify key Verilog topics that often appear in assessments. These include:

- Basic Syntax and Data Types
- Behavioral and Structural Modeling
- Modules and Hierarchy
- Dataflow and Behavioral Descriptions
- Simulation and Testbenches
- Timing and Delays
- Synthesis Limitations and Guidelines
- Verilog Constructs and Reserved Keywords

Each topic area forms the foundation of Verilog knowledge, and MCQs are designed to test understanding in these domains.

Sample Verilog Multiple Choice Questions with Answers

Below, we explore a curated set of MCQs, explaining each question?s intent and providing detailed answers. These questions are representative of what learners might encounter in exams or practical assessments.

1. Which of the following best describes a 'module' in Verilog?

A) A block of code used for generating test signals

B) The fundamental building block used to model hardware components

C) A syntax error that occurs during compilation

D) A special type of variable used for storing data

Answer: B) The fundamental building block used to model hardware components

Explanation:

In Verilog, a module is a core construct representing a hardware component or system. It encapsulates a piece of hardware design, including inputs, outputs, internal logic, and submodules. Modules are hierarchical, enabling complex designs to be built from simpler submodules. Unlike options A, C, and D, which are either incorrect or unrelated, the correct answer emphasizes the modular and hierarchical nature of Verilog design.

2. What is the primary purpose of the 'always' block in Verilog?

A) To define combinational logic that executes once during simulation

B) To specify sequential logic that executes repeatedly based on a sensitivity list

C) To declare variables that retain their value across simulation cycles

D) To initialize variables at the start of simulation only

Answer: B) To specify sequential logic that executes repeatedly based on a sensitivity list

Explanation:

The 'always' block in Verilog is used to describe both combinational and sequential logic, depending on how it is written. When used with a sensitivity list (e.g., `always @(posedge clk)`), it models sequential logic that triggers on clock edges. Without a sensitivity list, it can model combinational logic. The key aspect here is its role in describing logic that executes repeatedly based on specific signals, making option B the best choice.

3. Which data type is used to declare a 4-bit register in Verilog?

A) reg [3:0] my_reg;

B) wire [3:0] my_wire;

C) bit [4] my_bit;

D) reg my_reg[3:0];

Answer: A) reg [3:0] my_reg;

Explanation:

To declare a 4-bit register, Verilog uses the `reg` keyword with a range specifying the bit width. The syntax `reg [3:0] my_reg;` creates a 4-bit register, with bits numbered from 3 down to 0. Option B declares a wire, which is used for combinational signals, not registers. Option C uses an incorrect data type `bit`, which is not standard in Verilog-2001. Option D suggests an array of regs, which is not the typical way to declare a 4-bit register.

4. In Verilog, what does the 'assign' statement do?

A) Declares a new variable

B) Describes combinational logic by assigning values continuously

C) Initiates sequential logic triggered on clock edges

D) Instantiates a module

Answer: B) Describes combinational logic by assigning values continuously

Explanation:

The 'assign' statement in Verilog is used for continuous assignment, which models combinational logic. It updates the assigned signal immediately whenever the right-hand side expression changes. This is different from procedural assignments within 'always' blocks, which are triggered by events. Options A, C, and D do not accurately describe the function of 'assign'.

5. Which of the following is NOT a valid Verilog keyword?

A) module

B) always

C) process

D) reg

Answer: C) process

Explanation:

In Verilog, `module`, `always`, and `reg` are all valid keywords used for defining modules, behavior, and data types, respectively. However, `process` is not a Verilog keyword; it is more common in VHDL. Recognizing valid keywords is crucial for correct syntax and avoiding compilation errors.

Tips for Using Verilog MCQs Effectively

While practicing MCQs is beneficial, maximizing their effectiveness requires strategic approaches:

- **Understand the Explanation: Always review the explanation given with each answer to grasp the underlying concept.**
- **Identify Patterned Questions: Notice recurring question types or themes to focus your study on weak areas.**
- **Simulate Real Exam Conditions: Practice MCQs under timed conditions to improve efficiency.**
- **Use Multiple Resources: Incorporate textbooks, online quizzes, and simulation tools for comprehensive understanding.**
- **Create Your Own Questions: Developing questions helps reinforce learning and identify gaps in knowledge.**

Leveraging Online Resources and Practice Sets

Numerous online platforms offer extensive collections of Verilog MCQs with answers, often categorized by difficulty level or topic. These resources can be invaluable for:

- **Self-Assessment: Track progress over time.**
- **Exam Preparation: Focus on high-yield topics.**
- **Community Engagement: Join forums or study groups for discussion and clarification.**

Some prominent platforms include:

- **EEVblog Forums**
- **Electronics Tutorials Websites**
- **Online Coding and Hardware Simulation Platforms**
- **Official Certification Practice Tests**

Conclusion: Mastering Verilog MCQs for Success

Verilog multiple choice questions with answers are indispensable tools for anyone seeking mastery in digital design and hardware description languages. They serve as both learning aids and assessment instruments, enabling learners to test their understanding, reinforce concepts, and prepare effectively for academic or industry exams.

By focusing on core topics, understanding question rationales, and leveraging diverse resources, students and professionals can enhance their proficiency in Verilog. Remember, consistent practice with well-designed MCQs not only boosts confidence but also cultivates the critical thinking skills necessary for robust digital system design.

Final thought: Embrace practice as a continuous journey?each question answered brings you closer to becoming a proficient Verilog designer and a valuable contributor to the digital hardware community.

QuestionAnswer What is the primary purpose of Verilog in digital design? Verilog is used for modeling, simulating, and synthesizing digital circuits and systems. Which of the following is a valid Verilog data type? reg and wire are valid Verilog data types. In Verilog, what does the keyword 'always' signify? 'always' indicates a block of code that executes repeatedly based on specified sensitivity lists or conditions. Which Verilog construct is used to describe combinational logic? The 'assign' statement and 'always @' blocks are used for modeling combinational logic. What is the difference between 'reg' and 'wire' in Verilog? 'reg' can hold a value and is used in procedural blocks, while 'wire' is used to connect different modules and is driven by continuous assignments. Which Verilog construct is used for parameterized modules? The 'parameter' keyword allows for defining modules with configurable parameters. What is the role of the 'initial' block in Verilog? The 'initial' block is used to specify code that executes only once at simulation start, typically for setting initial conditions.

Related keywords: Verilog quiz, Verilog MCQs, hardware description language questions, digital design tests, Verilog interview questions, FPGA programming quiz, Verilog fundamentals, digital logic questions, Verilog syntax quiz, Verilog exam preparation

Read the full article online: Verilog multiple choice questions with answers