Certainly! Here's a list of 100 SystemVerilog coding questions, one for each day of your #100DaysOfVerification challenge. These questions cover a wide range of SystemVerilog topics and difficulty levels to help you progressively improve your verification skills:

**Days 1-10: SystemVerilog Basics**

1. Write a SystemVerilog module for a 2-input AND gate.
2. Implement a 4-input OR gate using SystemVerilog.
3. Create a SystemVerilog module for a 3-to-8 decoder.
4. Design a testbench for a 2-input XOR gate in SystemVerilog.
5. Write SystemVerilog code for a 4-bit binary adder.
6. Implement a 4-bit ripple-carry adder testbench.
7. Create a SystemVerilog module for a D flip-flop.
8. Write a SystemVerilog module for a 4-bit binary counter.
9. Implement a simple scoreboard in SystemVerilog.
10. Design a testbench for a 4-input multiplexer.

**Days 11-20: Advanced Combinational Logic** 11. Write a SystemVerilog module for a 4-to-16 line decoder using generate statements.

1. Implement a 4-bit BCD to 7-segment decoder in SystemVerilog.
2. Create a SystemVerilog module for a 16-bit barrel shifter.
3. Design a 16-bit parallel adder-subtractor in SystemVerilog.
4. Write SystemVerilog code for a 2:1 priority multiplexer.
5. Implement a 16-bit parallel-to-serial converter in SystemVerilog.
6. Create a SystemVerilog module for a 4-bit ALU.
7. Write a SystemVerilog testbench to verify a 4-bit magnitude comparator.
8. Design a testbench to verify the functionality of a priority encoder module in SystemVerilog.
9. Implement a 4-bit ripple-carry subtractor in SystemVerilog.

**Days 21-30: Sequential Logic and State Machines** 21. Write SystemVerilog code for a T flip-flop.

1. Implement a 3-bit synchronous up-counter in SystemVerilog.
2. Create a SystemVerilog module for a 4-bit shift register.
3. Design a Mealy state machine for a simple traffic light controller in SystemVerilog.
4. Write SystemVerilog code for an edge-triggered D flip-flop with asynchronous reset.
5. Implement a 4-bit Johnson counter in SystemVerilog.
6. Design a Moore state machine for a vending machine controller in SystemVerilog.
7. Create a SystemVerilog module for a 4-bit synchronous down-counter.
8. Write SystemVerilog code for a ring counter.
9. Implement a 4-bit gray code counter in SystemVerilog.

**Days 31-40: Testbench Development** 31. Write a SystemVerilog testbench to verify the functionality of a UART transmitter module.

1. Implement a testbench for a memory controller module in SystemVerilog.
2. Create a SystemVerilog testbench to verify the operation of a FIFO buffer.
3. Design a testbench for a 16-bit floating-point adder in SystemVerilog.
4. Write SystemVerilog code for a constrained random testbench to verify a simple CPU instruction set.
5. Implement a testbench for a SPI (Serial Peripheral Interface) module in SystemVerilog.
6. Create a SystemVerilog testbench to verify a DDR (Double Data Rate) memory interface.
7. Write SystemVerilog code for a scoreboard to track data transactions in a verification environment.
8. Implement a SystemVerilog testbench for a DMA (Direct Memory Access) controller.
9. Design a testbench for a pipelined processor core in SystemVerilog.

**Days 41-50: Advanced Verification Techniques** 41. Write SystemVerilog code to implement a UVM (Universal Verification Methodology) sequence.

1. Implement a verification environment for a custom protocol using UVM.
2. Create a SystemVerilog testbench with functional coverage to analyze the verification progress.
3. Design a SystemVerilog testbench for verifying a cache coherence protocol.
4. Write SystemVerilog code to generate and verify transactions on an AXI (Advanced eXtensible Interface) bus.
5. Implement a testbench for a complex SoC (System-on-Chip) design using UVM.
6. Create a SystemVerilog testbench for a PCIe (Peripheral Component Interconnect Express) interface.
7. Write SystemVerilog code to implement a protocol checker for a custom communication protocol.
8. Design a testbench for verifying power management features in a chip using UVM.
9. Implement a SystemVerilog testbench for verifying a cryptographic module.

**Days 51-60: Coverage and Debugging** 51. Write SystemVerilog code to implement functional coverage for a cache controller.

1. Implement a SystemVerilog assertion to check for proper reset behavior in a design.
2. Create a SystemVerilog testbench to simulate corner-case scenarios in a design.
3. Design a testbench to track and report code coverage metrics for a CPU design.
4. Write SystemVerilog code to implement cross-coverage between two verification components.
5. Implement a SystemVerilog assertion to check for proper clock domain crossings.
6. Create a SystemVerilog testbench for debugging a failing test case.
7. Design a testbench to collect and analyze assertion violations.
8. Write SystemVerilog code to implement a scoreboard for a cache coherence protocol.
9. Implement a SystemVerilog assertion for checking proper data alignment in memory transactions.

**Days 61-70: Interfacing and Communication** 61. Create a SystemVerilog module for a UART receiver.

1. Write SystemVerilog code to simulate SPI communication with a sensor.
2. Implement a SystemVerilog testbench for verifying I2C communication.
3. Design a SystemVerilog module for a wishbone bus interface.
4. Write SystemVerilog code for a simple memory-mapped register interface.
5. Implement a SystemVerilog testbench to verify communication between multiple components using FIFOs.
6. Create a SystemVerilog module for a simple message-passing interface.
7. Design a SystemVerilog testbench for verifying the communication protocol between a CPU and memory.
8. Write SystemVerilog code to implement a custom handshake protocol between two modules.
9. Implement a SystemVerilog testbench for a packet-based network communication protocol.

**Days 71-80: Advanced Topics** 71. Design a SystemVerilog module for a dynamic reconfigurable hardware block.

1. Write SystemVerilog code to implement an instruction set simulator for a custom CPU.
2. Create a SystemVerilog module for a hardware accelerator.
3. Implement a SystemVerilog testbench for verifying a hardware security module.
4. Design a SystemVerilog module for a floating-point unit (FPU).
5. Write SystemVerilog code for a high-level synthesizable module.
6. Implement a SystemVerilog testbench for a complex DSP (Digital Signal Processor) design.
7. Create a SystemVerilog module for a cryptographic co-processor.
8. Design a SystemVerilog testbench for a hardware-based image processing pipeline.
9. Write SystemVerilog code to implement a hardware task scheduler.

**Days 81-90: Real-World Projects** 81. Choose a real-world project idea and outline its verification requirements.

1. Create a verification plan for your chosen project.
2. Start implementing your project's verification environment in SystemVerilog.
3. Write test cases to verify the core functionality of your project.
4. Implement constrained random testing for your project's verification.
5. Design a testbench to verify corner cases and boundary conditions.
6. Write assertions to check critical properties of your project's design.
7. Perform functional coverage analysis for your project's verification.
8. Debug and resolve issues in your project's verification environment.
9. Reflect on your #100DaysOfVerification journey and what you've learned about SystemVerilog and verification.

These questions should help you structure your learning and provide a roadmap for your #100DaysOfVerification challenge. Feel free to adapt them as needed based on your specific interests and goals.