Digital Clock

In Verilog

<u>**Lessons Learned**</u>

Nima Bargestan

1. Modular Design

The principle of modular design is a methodology that breaks down a large system into small, independent, reusable components. It makes the separation of concerns in design *more debuggable, extensible, and maintainable*. Each module will be developed and tested separately, then integrated to form the complete system.

Example in the Project:

- *ClockDivider* **Module**:
  - Handles the generation of a pulse signal at a specified frequency, independent of the main clock logic.
  - Reusable in other projects that require clock frequency reduction.
- *DigitalClock* **Module:**
  - Responsible for counting seconds, minutes, and hours, updating time based on the pulse signal from the ClockDivider.
- By isolating the clock division and timekeeping into separate modules, you made the design more modular and reusable.

2. Clock Division

A great deal of comprehension was needed to understand this key concept. Clock division is a method of obtaining a desired lower frequency from an input clock frequency by counting the clock cycles. This is quite useful in applications where slower timing signals are required, like generating a 1 Hz pulse from a high-frequency clock.

Example in the Project:

- The *ClockDivider* module:
  - Uses a counter (**count**) to track the number of clock cycles.
  - Produces a pulse (**pulse_out**) when the counter reaches the **divide_by** value (derived from **clock_frequency**).
  - This pulse acts as a 1 Hz clock for the *DigitalClock* module, ensuring consistent timing for seconds, minutes, and hours.

3. Sequential Logic

Sequential logic involves storing and updating states (e.g., counters) on clock edges. It ensures predictable behavior based on clock signals and allows the design to maintain its state across clock cycles.

Example in the Project:

    - The *DigitalClock* module:
        - Updates the seconds, minutes, and hours counters on the rising edge of the **one_sec_pulse**.
        - Each counter is implemented using registers, ensuring that their values persist and are updated synchronously.

4. Binary Coded Decimal (BCD)

BCD is a representation of decimal numbers where each digit is encoded using 4 binary bits. It simplifies interfacing with digital displays (e.g., seven-segment displays) and provides human-readable values.

Example in the Project:

- Seconds, minutes, and hours counters are converted to BCD format:
    - The ones and tens places are calculated using modulo (**% 10**) and division (**/ 10**) operations.
    - Example:
        - For a counter value of 37, **sec_tens = 3 (37 / 10)** and **sec_ones = 7 (37 % 10)**.
- This makes the outputs directly usable for display purposes.

5. Parameterization

Parameterization allows the use of constants that can be modified without changing the core logic of the design. This makes the design flexible and scalable.

Example in the Project:

- Parameters like **MAX_SECONDS**, **MAX_MINUTES**, and **MAX_HOURS** define the maximum values for counters.
    - Example: **MAX_SECONDS = 59** ensures the seconds counter rolls over after 59.
- The *ClockDivider* module uses the **divide_by** parameter to configure the number of cycles for pulse generation. This makes it adaptable to different input clock frequencies.

6. Reset Logic

Reset logic ensures that the design initializes to a known state upon reset. This is crucial for proper operation and predictable behavior after power-up or during reset conditions.

Example in the Project:

- The *reset* signal:
    - Clears all counters (seconds, minutes, hours) in the *DigitalClock* module.
    - Resets the clock cycle counter in the *ClockDivider* module, ensuring a fresh start for pulse generation.
- Example Behavior:
    - If the reset is asserted, the time is set to *00:00:00*, and all internal counters are cleared.

7. Testbench Development

A testbench simulates the behavior of the design in a controlled environment, verifying its functionality under various scenarios. It generates input signals, observes outputs, and ensures the design meets its specifications.

Example in the Project:

- The *clock_testbench*:
    - Generates a *1.2 MHz* clock signal by toggling the **clk signal** every 416 ns.
    - Simulates reset conditions to verify proper initialization.
    - Monitors the outputs of the *DigitalClock* module (hours, minutes, seconds) based on the **one_sec_pulse**.
    - Logs time updates to the console using **$display**.

Summary

This project reflects my understanding of foundational digital design concepts and their practical applications. As a student, I have successfully combined key techniques like modular design, clock division, sequential logic, and testbench development to create a functional digital clock system. The use of parameterization, reset logic, and timing simulation demonstrates my growing familiarity with real-world design practices.

This project also shows my ability to apply what I've learned in my course, such as Verilog basics, while exploring new concepts like modularization and reusable components. It's a strong step forward in building my skills and confidence in digital system design. There's always room to grow, but this project is a great example of putting theory into practice effectively.

References

Modular Design:

- *Verilog (Wikipedia)*: Offers an overview of Verilog, including examples of modular design practices.
  Wikipedia

Clock Division:

- *Verilog code for Clock divider on FPGA*: Provides a comprehensive guide on implementing clock dividers in Verilog, complete with code examples and testbenches.
  FPGA4Student
- *Designing A Clock Divider In Verilog For Efficient Time Management*: Discusses the implementation of clock dividers in Verilog for efficient time management.
  PeerDH

Sequential Logic:

- *Sequential Logic with always block in Verilog*: Explores the use of the 'always' block in Verilog for designing sequential logic circuits.
  Piembsystech
- *Verilog HDL – Sequential Logic*: Provides insights into sequential logic design using Verilog, including state machine design principles.
  UT Dallas Personal Page

Binary Coded Decimal (BCD):

- *Verilog (Wikipedia)*: Includes examples of BCD representation and manipulation in Verilog.
  Wikipedia

Parameterization:

- *ASIC Design Flow in Verilog Programming Language*: Discusses the role of parameterization in the ASIC design flow using Verilog.
  Piembsystech

Reset Logic:

- *Implementing a Clock Boundary Synchronizer in Verilog*: Covers the implementation of reset logic in clock boundary synchronizers.
  Digi-Key Forum

Testbench Development:

- *Verilog-Codes-Sequential-Circuits*: A collection of Verilog codes for sequential circuits, including testbenches for practice.
  GitHub
-

END