Project 103 : Digital Clock FSM

A Comprehensive Study of Advanced Digital Circuits

By: Nikunj Agrawal , Gati Goyal, Abhishek Sharma , Ayush Jain

Documentation Specialist: Dhruv Patel & Nandini Maheshwari

Contents

1 Introduction

A Digital Clock Finite State Machine (FSM) is a sequential circuit that functions as the core logic for managing the states and operations of a digital clock. Digital clocks, widely used in modern systems, require precise timing mechanisms and efficient state management to ensure accurate timekeeping and smooth transitions between states such as hours, minutes, and seconds. The FSM serves as a reliable design model for implementing such functionality.

An FSM-based design for a digital clock breaks down the complex operation into distinct states, each representing a specific stage of the clock's operation. These states can include second increments, minute updates, hour changes, and mode switching (e.g., time-setting or alarm configuration). The transitions between states are governed by predefined rules based on inputs such as clock pulses, user commands, or external triggers.

The FSM approach provides a structured and modular methodology for implementing digital clocks, enabling designers to achieve high precision and efficient use of hardware resources. By leveraging the deterministic behavior of FSMs, digital clocks can maintain consistency in timing, even in resource-constrained environments.

This document outlines the design and operation of a Digital Clock FSM, including its state transition diagram, logic implementation, and potential applications in embedded systems and consumer electronics.

2 Background

A Finite State Machine (FSM) is a mathematical model of computation widely used in digital design and control systems. FSMs operate by transitioning between a finite number of states based on inputs, enabling deterministic behavior for managing sequential processes. In digital systems, FSMs are often employed to implement control logic due to their simplicity and predictability.

In the context of a digital clock, an FSM provides an efficient framework for managing the clock's timekeeping operations. Digital clocks require precise coordination to handle tasks such as incrementing seconds, updating minutes and hours, managing user interactions, and switching between different modes (e.g., time-setting and alarm configuration). These operations can be naturally modeled as states in an FSM, with well-defined transitions triggered by clock pulses, user inputs, or external signals.

A typical digital clock FSM includes states for incrementing seconds, updating minutes, and transitioning to the next hour. Additional states may be incorporated for auxiliary functions like resetting the clock, setting time, or controlling alarms. The FSM ensures that these operations occur sequentially and accurately, leveraging hardware-efficient designs for implementation in resource-constrained environments.

The FSM-based approach offers several advantages for digital clock design. It simplifies the design process by breaking down complex behavior into smaller, manageable states, and ensures reliable operation under various conditions. This makes FSMs a cornerstone in the design of modern digital clocks, embedded systems, and other time-sensitive applications.

This section provides an overview of the FSM concept and its role in digital clock design, laying the foundation for understanding the subsequent sections that detail the implementation and functionality of a digital clock FSM.

3 Structure and Operation

The Digital Clock Finite State Machine (FSM) is designed to handle the sequential operations required for precise timekeeping in digital clocks. The structure of the FSM consists of key components, including states, transitions, inputs, and outputs, each contributing to its overall functionality.

3.1 Structure

The FSM for a digital clock typically consists of the following elements:

- **States:** Each state represents a specific stage in the clock's operation. Common states include:
  - Incrementing seconds.
  - Updating minutes when seconds reach 60.
  - Advancing hours when minutes reach 60.
  - Special states for time-setting or alarm modes.
- **Inputs:** Inputs to the FSM include:
  - Clock pulses (from an external clock signal) to synchronize timekeeping.
  - User commands for setting time or configuring alarms.
  - Reset signals to initialize or reset the clock.
- **Transitions:** Transitions define how the FSM moves from one state to another based on inputs. For example:
  - Transition from the "Increment Seconds" state to "Update Minutes" state when seconds reach 60.
  - Transition to "Update Hours" state when minutes reach 60.
  - Transition to "Set Time" or "Alarm Config" states upon receiving user commands.
- **Outputs:** Outputs include signals to update the displayed time or to trigger alarms and auxiliary functions.

3.2 Operation

The operation of the Digital Clock FSM can be summarized as follows:

1. **Idle State:** The FSM begins in an idle state, waiting for an external clock pulse or a user command to start operation.
2. **Increment Seconds:** With each clock pulse, the FSM increments the seconds counter. When the counter reaches 60, the FSM transitions to the "Update Minutes" state.
3. **Update Minutes:** In this state, the FSM resets the seconds counter to zero and increments the minutes counter. If the minutes counter reaches 60, the FSM transitions to the "Update Hours" state.
4. **Update Hours:** The FSM resets the minutes counter to zero and increments the hours counter. If the hours counter reaches 24 (in a 24-hour format), it resets to zero.
5. **Auxiliary Operations:** In response to user inputs, the FSM can enter auxiliary states such as "Set Time" or "Alarm Configuration," allowing the user to adjust the clock settings.

By systematically transitioning between these states, the FSM ensures accurate and reliable timekeeping. The modular structure of the FSM enables easy integration of additional features, such as alarms, timers, or different time formats, without compromising the core functionality.

This structured and sequential operation makes the FSM an ideal choice for implementing digital clocks in embedded systems and other time-critical applications.

4 Implementation in System Verilog

Below is an example of a Digital Clock FSM implemented in System Verilog:

Listing 1: Digital Clock FSM

```
      module digital_clock (
    input logic clk,
    input logic reset,
    output logic [5:0] seconds,
    output logic [5:0] minutes,
    output logic [4:0] hours
);
    // State encoding
    typedef enum logic [1:0] {IDLE, INCREMENT_SECONDS,
       INCREMENT_MINUTES, INCREMENT_HOURS} state_t;
    state_t current_state, next_state;

    // Time registers
    logic [5:0] sec, min;
    logic [4:0] hr;

    // State transitions
    always_ff @(posedge clk or posedge reset) begin
        if (reset)
            current_state <= IDLE;
        else
            current_state <= next_state;
    end

    // State machine logic
    always_comb begin
        next_state = current_state;  // Default state remains unchanged
        case (current_state)
            IDLE: begin
                if (sec < 59)
                    next_state = INCREMENT_SECONDS;
                else if (min < 59)
                    next_state = INCREMENT_MINUTES;
                else if (hr < 23)
                    next_state = INCREMENT_HOURS;
            end

            INCREMENT_SECONDS: begin
                if (sec == 59)
                    next_state = INCREMENT_MINUTES;
                else
                    next_state = IDLE;
            end

            INCREMENT_MINUTES: begin
                if (min == 59)
                    next_state = INCREMENT_HOURS;
                else
                    next_state = IDLE;
            end

            INCREMENT_HOURS: begin
                if (hr == 23)
```

```
                    next_state = IDLE; // Rolls over to IDLE
                else
                    next_state = IDLE;
            end
        endcase
    end

    // Output logic and time update
    always_ff @(posedge clk or posedge reset) begin
        if (reset) begin
            sec <= 0;
            min <= 0;
            hr <= 0;
        end else begin
            case (current_state)
                INCREMENT_SECONDS: sec <= (sec == 59) ? 0 : sec + 1;
                INCREMENT_MINUTES: begin
                    sec <= 0;
                    min <= (min == 59) ? 0 : min + 1;
                end
                INCREMENT_HOURS: begin
                    sec <= 0;
                    min <= 0;
                    hr <= (hr == 23) ? 0 : hr + 1;
                end
                default: ; // Do nothing in IDLE
            endcase
        end
    end

    // Assign outputs
    assign seconds = sec;
    assign minutes = min;
    assign hours = hr;
endmodule
```

5 Simulation Results

6 Test Bench

The following test bench verifies the functionality of the Digital Clock FSM :

Listing 2: Digital Clock FSM Testbench

```
  module tb_digital_clock;
    logic clk;
    logic reset;
    logic [5:0] seconds, minutes;
    logic [4:0] hours;

    // Instantiate the DUT
    digital_clock uut (
        .clk(clk),
        .reset(reset),
        .seconds(seconds),
        .minutes(minutes),
        .hours(hours)
```

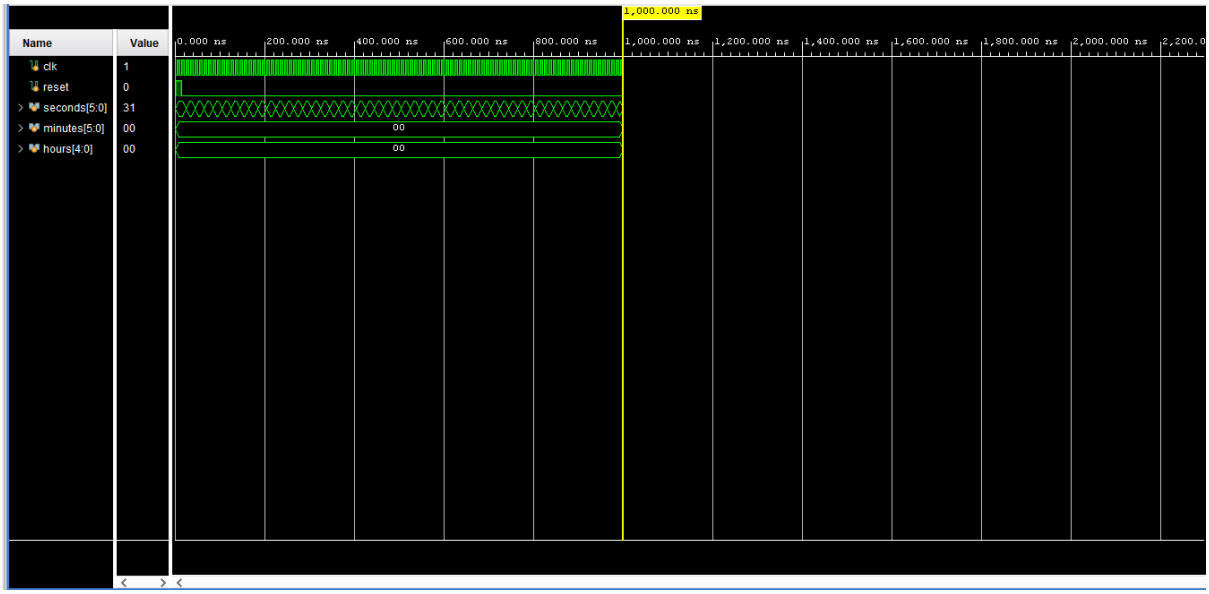

Figure 1: Simulation results of Digital clock FSM

```
    );

    // Clock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 100MHz clock
    end

    // Reset logic
    initial begin
        reset = 1;
        #15 reset = 0;
    end

    // Test scenario
    initial begin
        $monitor("Time -> %0d:%0d:%0d", hours, minutes, seconds);

        // Simulate for some time
        #1000;
        $stop;
    end
endmodule
```

7 Advantages and Disadvantages of Digital Clock FSM

The Finite State Machine (FSM) design for digital clocks offers several advantages, but also presents some challenges. Understanding both the benefits and limitations of this approach is crucial for determining its suitability for specific applications.

7.1 Advantages

- **Simplicity and Modularity:** The FSM model divides the complex operations of a digital clock into simple, discrete states. This modularity simplifies the design, testing, and maintenance of the

clock system.

- **Deterministic Behavior:** FSMs operate based on well-defined transitions, making the behavior of the clock predictable and reliable. This is essential for timekeeping applications where accuracy and consistency are paramount.
- **Efficient Resource Usage:** FSMs are well-suited for hardware implementation due to their efficient use of resources, such as logic gates and flip-flops. The simplicity of the FSM design reduces the need for complex circuitry, making it ideal for embedded systems with limited resources.
- **Scalability:** FSMs allow for easy scalability. Additional features, such as alarms, timers, or custom user interfaces, can be incorporated by adding new states and transitions without disrupting the core timekeeping functionality.
- **Ease of Implementation:** The FSM approach aligns well with digital circuit design, making it easier to implement in hardware description languages (HDLs) such as Verilog or VHDL. This facilitates the synthesis of the design into actual hardware for physical devices.

7.2 Disadvantages

- **Complexity for Large Designs:** As the number of states increases (e.g., with additional features like alarms, countdown timers, or multiple time formats), the FSM design can become more complex and harder to manage. This can lead to increased design time and potential errors.
- **State Explosion:** In large systems, the number of states can grow rapidly, leading to what is known as "state explosion." This makes the FSM difficult to scale and can increase the size of the circuit, consuming more resources.
- **Limited Flexibility:** Once the FSM design is implemented, making changes or adding new functionality may require significant modifications to the state transitions and logic. This lack of flexibility can be a drawback in systems where frequent updates or changes are needed.
- **Difficulty in Handling Concurrent Operations:** FSMs are inherently sequential, and handling concurrent tasks (e.g., multiple alarms or simultaneous time and date displays) may require complex state management or the introduction of additional FSMs, complicating the design.
- **Scalability Concerns in Time-Dependent Systems:** While FSMs are great for smaller, simpler systems, as the complexity increases (e.g., managing multiple time zones or advanced user settings), FSMs can become cumbersome, requiring complex hierarchies of states and transitions.

In summary, the FSM approach to digital clock design is advantageous for its simplicity, predictability, and efficient resource usage, making it ideal for embedded systems and applications requiring accurate timekeeping. However, as system complexity increases, managing an FSM can become more challenging, especially when additional features or concurrent operations are needed.

8 Conclusion

In conclusion, the Digital Clock Finite State Machine (FSM) offers a structured, efficient, and reliable approach to designing timekeeping systems. By breaking down the clock's operations into discrete states, the FSM simplifies the design, testing, and maintenance of digital clocks. Its deterministic nature ensures that transitions between states occur predictably, making it ideal for applications requiring consistent timekeeping.

The FSM approach provides several advantages, including simplicity, scalability, and efficient resource usage, particularly in embedded systems with limited hardware resources. The modular design of FSMs makes it easy to incorporate additional features such as alarms, timers, or multiple time formats without disrupting the core functionality of the clock.

However, as with any design methodology, there are limitations to consider. As the complexity of the system increases, managing a large number of states and transitions can become challenging. Furthermore, FSMs may struggle to efficiently handle concurrent operations, which could require additional complexity in the design.

Overall, the Digital Clock FSM remains a highly effective choice for designing timekeeping systems, particularly for smaller, embedded applications where simplicity, reliability, and resource efficiency are key priorities. For more complex applications, further design techniques, such as hierarchical FSMs or alternative control models, may be required to manage increasing functionality and system requirements.

Future work in this area may focus on optimizing the FSM for handling more advanced features or improving its scalability for larger systems. Despite its challenges, the FSM approach provides a solid foundation for digital clock design and continues to be a valuable tool in embedded and digital system design.

9 Schematic

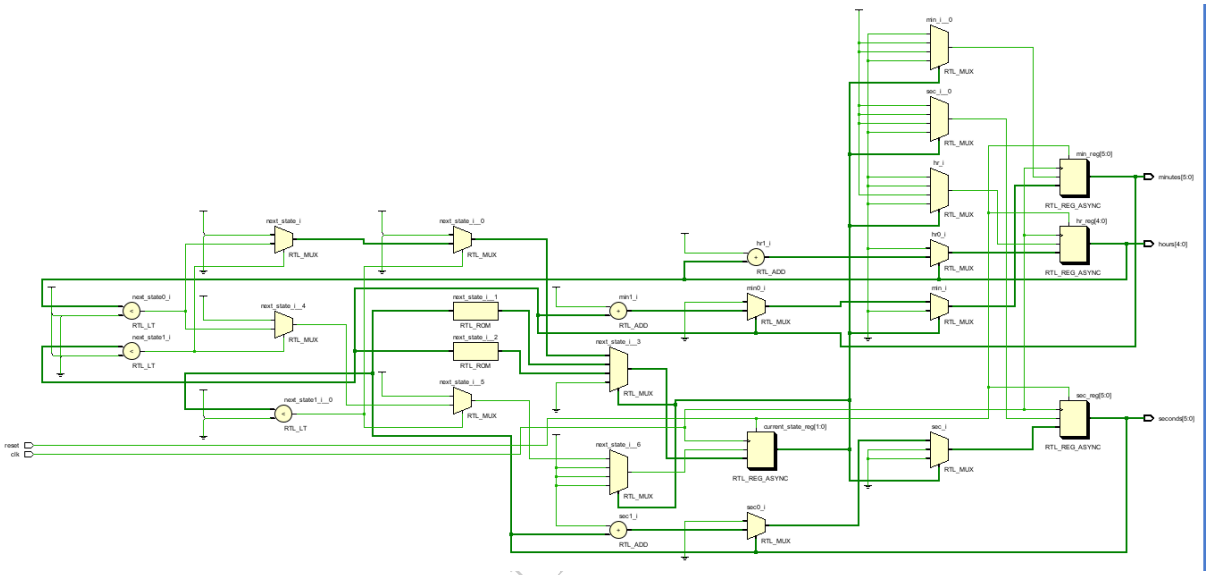

Figure 2: Schematic of Digital Clock FSM

10 Synthesis Design

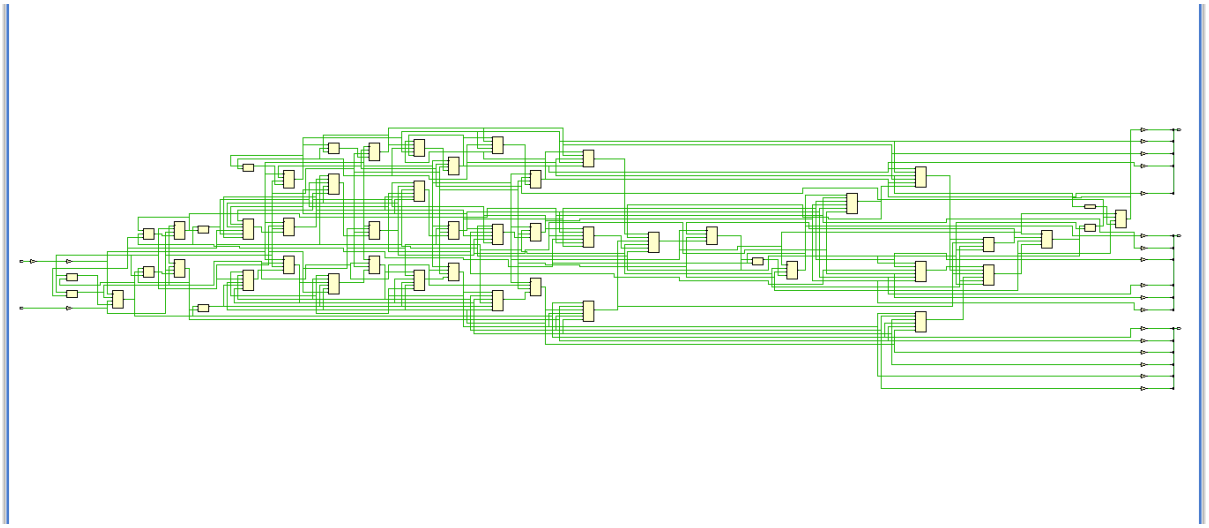

Figure 3: Synthesis of Digital Clock FSM

11 Frequently Asked Questions (FAQ)

11.1 What is a Finite State Machine (FSM)?

A Finite State Machine (FSM) is a computational model that consists of a finite number of states. It operates by transitioning between these states based on inputs, making it useful for modeling sequential logic systems. In a digital clock, the FSM controls the transitions between states such as incrementing seconds, updating minutes, or advancing hours.

11.2 Why use an FSM in digital clock design?

An FSM is ideal for digital clock design because it provides a structured and predictable way to manage the sequential operations of the clock. Each operation, such as incrementing seconds or updating minutes, can be modeled as a state in the FSM, making the system easy to design, test, and maintain. The FSM's deterministic nature ensures that transitions occur accurately and reliably.

11.3 How does the FSM handle time increments in a digital clock?

The FSM for a digital clock handles time increments by transitioning between states. Each clock pulse triggers the FSM to increment the seconds, and once the seconds reach 60, the FSM transitions to the "update minutes" state. Similarly, when minutes reach 60, the FSM moves to the "update hours" state. This systematic approach ensures accurate timekeeping.

11.4 Can the FSM design be used for both 12-hour and 24-hour clocks?

Yes, the FSM design can be adapted for both 12-hour and 24-hour clocks. For a 12-hour clock, the FSM would include logic to reset the hour count after 12, while for a 24-hour clock, it would reset the hour count after 24. The state transitions and outputs can be modified accordingly to handle different time formats.

11.5 What happens when the user sets the time or adjusts the clock?

When the user sets the time or adjusts the clock, the FSM can enter a special state, such as "Set Time," where the user inputs the desired time. In this state, the FSM allows the user to increment the hours, minutes, or seconds directly. Once the user is done, the FSM transitions back to the normal timekeeping states.

11.6 What are the limitations of using an FSM for digital clock design?

While FSMs are efficient and easy to implement, they can become complex as the system grows. For instance, adding features like alarms, timers, or different time zones can increase the number of states, leading to more complex designs. Additionally, FSMs are inherently sequential, which can make it difficult to handle concurrent operations without introducing additional complexity.

11.7 How can the FSM design be improved for larger systems?

For larger systems, one approach is to use hierarchical FSMs, where high-level states are broken down into smaller sub-states. This can help manage complexity and reduce the number of transitions. Alternatively, other control models, such as dataflow-based designs or microprocessor-based systems, may be considered for systems requiring more flexibility or handling multiple operations simultaneously.