Design and Verification of Digital Clock FSM using SystemVerilog

(by CH.ANIRUDH)

1. Introduction

A digital clock is a **sequential circuit** that keeps track of time using a combination of **counters** and control logic. It is widely used in **digital systems**, **embedded applications**, and real-time clock (RTC) implementations. The digital clock operates by counting seconds, minutes, and hours, displaying time in a 24-hour or 12-hour format. In this project, a Finite State Machine (FSM) is designed to implement a **SystemVerilog**-based digital clock, ensuring accurate state transitions between hours, minutes, and seconds. The FSM approach provides a structured way to control the clock's behavior, avoiding unnecessary glitches and ensuring smooth time progression.

This project focuses on implementing a Digital Clock FSM using SystemVerilog **enum**, ensuring a clear and structured state representation. The design is simulated and verified using a testbench, checking various edge cases such as midnight transition (23:59 → 00:00), seconds rollover (59 → 00), and hour format control. The project also explores synthesis aspects, ensuring that the **FSM-based clock** is optimized for hardware implementation in **FPGA**s or **ASICs**.

2. Digital Clock FSM Overview

The **digital clock** is implemented using a **Finite State Machine** (FSM), where each state represents a specific time event. The clock consists of three main counters: seconds, minutes, and hours. The **FSM** ensures that when the seconds counter reaches 59, it resets to 00 and **increments** the minutes counter. Similarly, when the minutes **counter** reaches 59, it resets and increments the hours counter. If the clock operates in 24-hour mode, it resets at 23:59:59, and in 12-hour mode, it resets at 12:59:59 with an AM/PM toggle.

The FSM operates in a synchronous manner, with state transitions occurring on every positive edge of the **clock signal**. The states are represented using **SystemVerilog** enum, ensuring a readable and structured design. The transitions between states are event-driven, controlled by conditions such as the carry-out from counters, reset signal, and mode selection (12-hour or 24-hour format). The design also includes enable signals for starting, stopping, and resetting the clock.

To ensure efficient operation, the clock is designed with modular counters for **seconds**, **minutes**, and **hours**. Each counter is implemented using a

SystemVerilog always block, which increments the count based on FSM control signals. The FSM transitions ensure that each counter resets at the appropriate limit, **preventing** invalid states.

3. Design Implementation

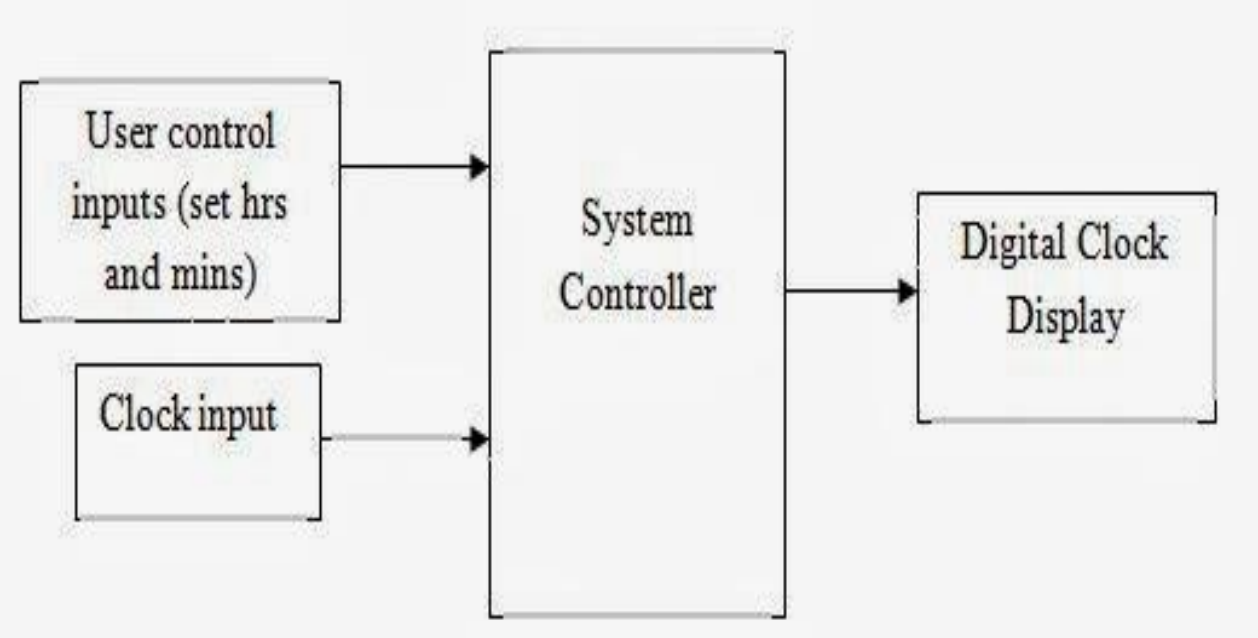
The digital clock FSM is implemented in SystemVerilog, following a structured RTL design approach. The design consists of the following key components:

1. **FSM Controller** – The core **Finite State Machine** manages state transitions between different clock conditions. It ensures smooth rollover from seconds to minutes and hours, resetting when necessary.
2. **Seconds Counter** – A mod-60 counter that increments every second and triggers a carry signal when it reaches 59.
3. **Minutes Counter** – A mod-60 counter that increments when the seconds counter resets to 00 and generates a carry-out signal for hours.
4. **Hours Counter** – A mod-24 counter (or mod-12 for 12-hour mode) that increments when minutes reset to 00. It resets at 23:59:59 or toggles AM/PM in 12-hour mode.
5. **Clock Control Signals** – Enable, reset, and mode-selection signals allow external control over the clock.
6. **SystemVerilog enum Implementation** – The FSM states are represented using SystemVerilog enum, improving readability and avoiding hardcoded state values.

The **FSM states** include:

- **IDLE**: The clock is stopped (waiting for enable signal).
- **RUNNING**: The clock increments time based on second transitions.
- **MIDNIGHT RESET**: The clock resets when reaching **23:59:59** or 12:59:59 (AM/PM mode).
- **HOUR INCREMENT**: Handles hour counter updates when **minutes reach 59**.
- **STOPPED**: The clock is paused but retains the current time.

The FSM logic is implemented using sequential always blocks, ensuring proper synchronization with the system clock. The clock is driven by a one-second pulse, generated using a frequency divider for simulation purposes.



Fig(a): Digital-Clock

4. Operations of Digital Clock FSM

The **Digital Clock FSM** operates as a **sequential state machine**, managing **seconds, minutes, and hours counting**. The **FSM transitions** between different states based on time conditions and updates the clock accordingly. Below is the step-by-step operation of the digital clock:

State Machine Execution

The FSM has **four states**:
Idle – The clock waits and checks conditions for incrementing time.
Increment Seconds – Increments **seconds** every clock cycle.
Increment Minutes – When seconds reach 59, **minutes** are incremented, and seconds reset to 0.
Increment Hours – When minutes reach 59, **hours** are incremented, and minutes reset to 0.

Time Counting and State Transitions

Incrementing Seconds

- Clock starts in the **Idle state**.

- If sec < 59, it moves to the **Increment Seconds** state and increments the **seconds counter**.
- If sec == 59, the FSM moves to the **Increment Minutes** state.

Incrementing Minutes

- If sec == 59, the seconds counter resets to 0, and the minutes counter increments.
- If min < 59, the FSM moves back to **Idle**.
- If min == 59, the FSM moves to **Increment H0urs**.

Incrementing Hours

- If min == 59, the minutes counter resets to 0, and the hours counter increments.
- If hr == 23, the FSM resets the time to 00:00:00.
- Otherwise, it moves back to **Idle**.

5. SystemVerilog Implementation:

RTL Code:

```
module digital_clock_FSM(

    input logic clk,
    input logic reset,
    output logic [5:0] seconds,
    output logic [5:0] minutes,
    output logic [4:0] hours

);
//start 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 = 0,min = 0;
logic [4:0] hr = 0;
//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; // rools over to IDLE
            else
              next_state = IDLE;
            end
        endcase
        end
    //output logics and time updates
    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
```

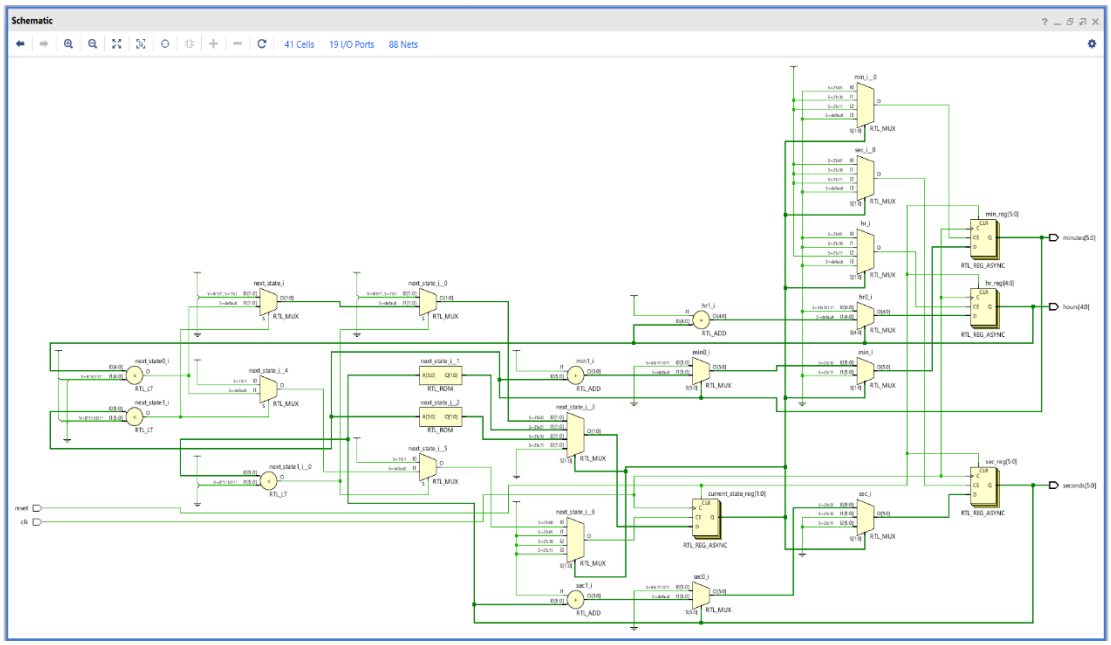
```
        end
      end
    //assign outputs
    assign seconds =  sec;
    assign minutes = min;
    assign hours = hr;

endmodule
```

6. Testbench Simulation:

```
    module digital_clock_FSM_tb;
        logic clk;
        logic reset;
        logic [5:0] seconds,minutes;
        logic [4:0] hours;
     //instantiate the DUT
        digital_clock_FSM uut(
            .clk(clk),
            .reset(reset),
            .seconds(seconds),
            .minutes(minutes),
            .hours(hours)
            );
     //clock generation
      initial begin
        clk = 0;
        forever #5 clk = ~clk; //100MHz clock
        end
     //reset logic
      initial begin
        reset = 1;
```

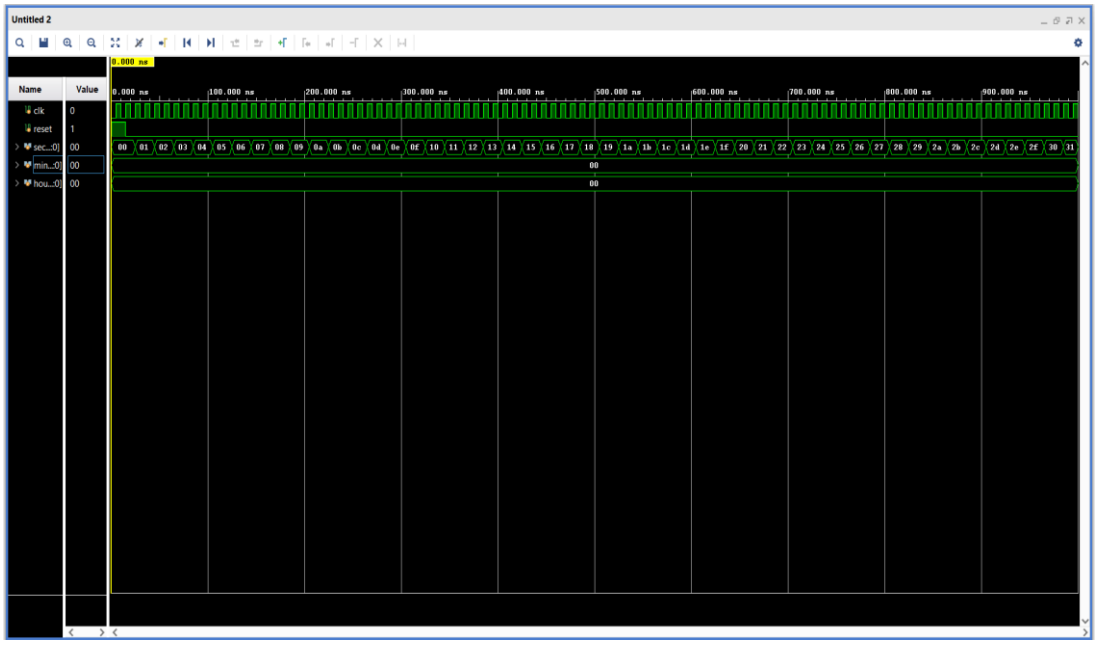
```
    #15 reset = 0;
  end
 //test scenario
  initial begin
     $monitor ("Time -> %0d : %0d :%0d", hours,minutes,seconds);
   //simulate for sometime
   #1000;
   $stop;
   end

endmodule
```

7. Elaborated design:



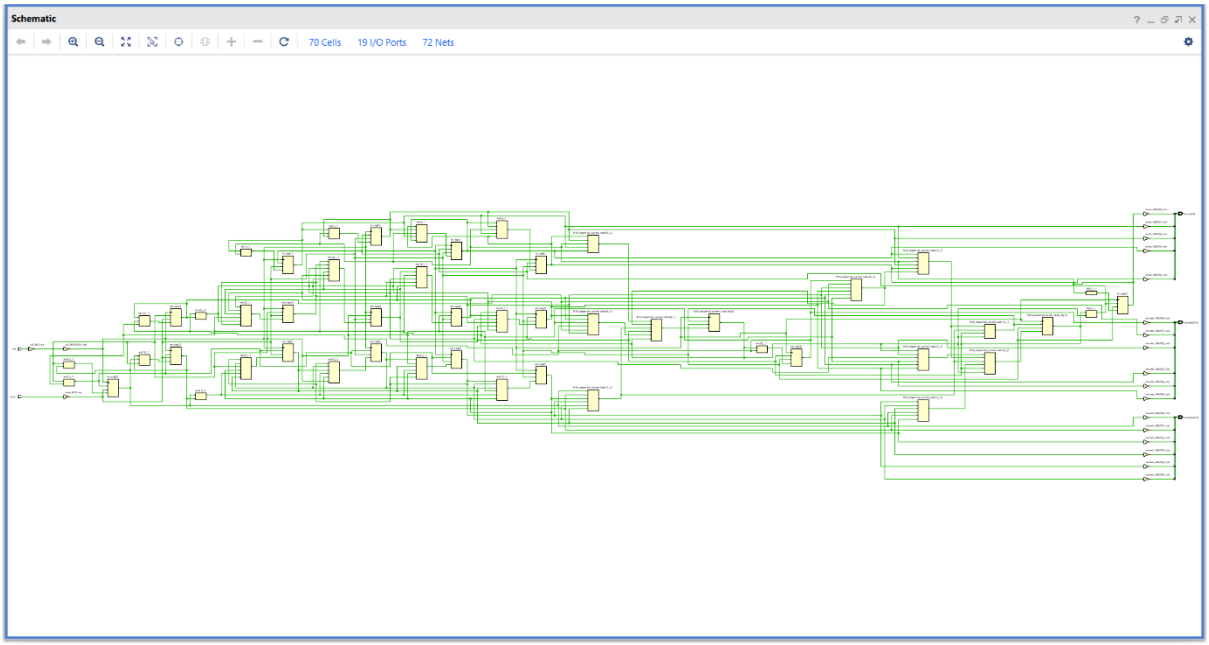
Fig(b): Elaborated Design of Digital-clock FSM

8. Simulation:



Fig(c): Simulation

9.Synthesis:



Fig(d): Schematic

10. Advantages & Disadvanatges:

Advantages:

Using an FSM-based approach for the digital clock provides several benefits:

1. **Structured Design – FSM** ensures well-defined transitions between clock states.
2. **Modular Counters** – The use of independent counters for seconds, minutes, and hours improves **scalability**.
3. **Glitch-Free Operation –** State transitions are controlled, reducing timing **glitches**.
4. **Easy Debugging – SystemVerilog enum** improves readability, making debugging simpler.
5. **Hardware Optimized –** FSM-based designs are efficient for **FPGA** and ASIC implementation, ensuring minimal hardware overhead**.**

Disadvantages:

Despite its advantages, an FSM-based digital clock also has some limitations:

1. **Fixed Functionality** – Unlike software-based clocks, FSMs have predefined states and cannot be **dynamically** modified**.**
2. **Complexity for Additional Features –** Adding features like stopwatch **functionality** or alarms requires additional FSM states, increasing complexity.
3. **Resource Utilization –** While efficient, an FSM-based clock may use slightly more **flip-flops** and **logic gates** compared to a simple counter-based approach.

11.Applications:

FSM-based digital clocks are widely used in various **real-time applications** such as:

Embedded Systems – Used in **real-time clocks (RTC)** for microcontrollers and FPGA-based designs.

FPGA and ASIC Designs – Implemented in **hardware-level digital timers and scheduling circuits**.

Consumer Electronics – Found in **digital watches, microwave ovens, and digital dashboard displays**.

Automotive Applications – Used in **car dashboard clocks and infotainment systems**.

Industrial Automation – Provides accurate timekeeping for **event-triggered automation systems.**

12. Conclusion:

This project successfully implements a **Digital Clock FSM** using **SystemVerilog .** Ensuring a structured approach to state transitions and timekeeping. The **FSM** design effectively manages **seconds, minutes**, and **hours** transitions, ensuring **accurate time** updates and handling rollovers at midnight.

The testbench verifies the design by simulating various scenarios, including midnight reset, second/minute transitions, and **12-hour vs. 24-hour modes.**

---The End---