Project Title: Design and Implementation of a Digital Alarm Clock Using Verilog.

Objectives

The main objectives of this project are

- To design and develop a digital alarm clock using Verilog that accurately displays hours, minutes, and seconds.
- To implement a reset functionality that allows the clock to synchronize its time by setting user-defined hour and minute values and resetting seconds to zero upon a reset signal.
- To integrate an alarm feature that triggers an alert when the current time matches the preset alarm time and keeps the alarm active for exactly one minute.
- To write and simulate the Verilog code using testbenches in Cadence software and observe the output waveforms to verify the correct behavior of the clock and alarm functionalities.
- To perform RTL synthesis of the Verilog design in Cadence, optimizing for efficient hardware implementation

Introduction

This project involves the design and implementation of a digital alarm clock using Verilog, intended to display real-time hours, minutes, and seconds and update them accurately with each clock cycle, simulating the behavior of a practical clock. Beyond simple time display, the clock includes two key features: **reset** and **alarm** functionalities.

The **reset feature** allows the clock to synchronize its time upon receiving a high pulse on the reset signal. When activated, the clock resets the second count to zero, while setting the minute and hour outputs to user-defined values provided through reset_min and reset_hour. This ensures precise and consistent timekeeping, allowing the user to set the clock to a specific starting time at any point.

The **alarm feature** enables the clock to compare its current hour and minute outputs with user-defined alarm settings (alarm_hour and alarm_min). When both match, the clock triggers the alarm by setting an output flag high (alarm = 1) for exactly one minute. After this period, as the minute count increments, the alarm flag is automatically reset to zero, turning the alert off.

To implement and validate this design, several key concepts and terminologies are used:

- **Verilog HDL (Hardware Description Language):** Verilog is used to describe the behavior and structure of digital systems. In this project, Verilog code defines the

logic for time counting, reset operations, and alarm triggering.

- **Testbench Simulation:** A testbench is a virtual environment in which the Verilog design is simulated. It provides input stimuli (like clock signals and reset pulses) and observes output waveforms to verify that the design works correctly.

- **Cadence Software:** Cadence provides a suite of electronic design automation (EDA) tools used for simulation, synthesis, and layout of digital designs. In this project, Cadence tools are used for waveform observation, RTL synthesis, and layout design.

- **RTL Synthesis:** RTL synthesis is the process of converting Verilog code into a gate-level netlist that can be physically implemented on hardware. This step optimizes the design for performance, area, and power consumption, preparing it for real-world deployment.

The project follows a structured workflow: writing Verilog code, simulating and verifying the design using Cadence testbenches, and performing RTL synthesis for optimization.

Software Requirements:

- Cadence

Working Procedure:

1) For Verilog Code:

In this project, we developed a Verilog module to replicate the functionality of a digital clock with an integrated alarm system. The module accepts multiple input signals to control time counting, resetting, and alarm activation. Bit-widths for each signal are carefully selected to accommodate their maximum values:

Second, we need to reach 0 to 59. The minimum number of bits required to represent up to 59 is determined by the formula:

$$2^n\text{-}1 \geq \text{maximum value}$$

$$2^n\text{-}1 \geq 59$$

The value is 6 because 2^6-1= 63.
For this reason, we need at least **6 bits** to store minute values ranging from 0 to 59.
Similarly, in the case of seconds in the Alarm Clock.
For Hours, we need to reach 0 to 23. The minimum number of bits required to represent up to 23 is determined by the formula:

$$2^n\text{-}1 \geq \text{maximum value}$$

$$2^n\text{-}1 \geq 23$$

The value is 5 because 2^5-1= 31.
For this reason, we need at least **5 bits** to store minute values ranging from 0 to 32.

1) For Verilog Code:

In this project, we developed a Verilog module to replicate the functionality of a digital clock with an integrated alarm system. The module accepts multiple input signals to control time counting, resetting, and alarm activation. Bit-widths for each signal are carefully selected to accommodate their maximum values: minutes and seconds (ranging up to 60) require 6 bits, while hours (ranging up to 24) require 5 bits, as 2^6-1 = 63 and 2^5-1 = 31.

Input Signals:

1. clk: Clock input signal that drives the counting operation of the clock.
2. rst: Reset input; a high pulse on this line triggers the clock's reset sequence.
3. rst_min: A 6-bit input to set the minute value during reset.
4. rst_hr: A 5-bit input to set the hour value during reset.
5. a_min: A 6-bit input that defines the minute setting for the alarm.
6. a_hr: A 5-bit input that defines the hour setting for the alarm.

Output Signals:

1. sec: A 6-bit output showing the current seconds.
2. min: A 6-bit output showing the current minutes.
3. hr: A 5-bit output showing the current hours.
4. alarm_flag: A 1-bit output flag that is raised when the alarm time condition is satisfied.

Registers for Timekeeping:

- curr_sec: 6-bit register that keeps track of seconds.
- curr_min: 6-bit register that stores the current minutes.
- curr_hr: 5-bit register used for storing the current hour count.

These registers are updated continuously based on the clock pulse to reflect the running time.

1. Reset Operation:

- curr_sec is reset to zero (6'd0).
- curr_min is assigned the value of rst_min.
- curr_hr is assigned the value of rst_hr.

2. Normal Clock Operation:

- The seconds register (curr_sec) increments with each clock pulse.
- When curr_sec reaches 59, it resets back to zero, and the minutes counter (curr_min) is incremented.
- If curr_min also reaches 59, it resets to zero, and the hours counter (curr_hr) increments.
- When curr_hr reaches 23, it resets back to zero, thus maintaining a 24-hour time format.

Alarm Checking Logic:

The module constantly compares the current time with the preset alarm time:

- When curr_min equals a_min and curr_hr equals a_hr, the alarm_flag is set high (1'b1), indicating that the alarm condition has been triggered.
- The flag remains high for exactly one minute while the minute and hour continue matching the alarm time.
- Once the minute increments and the condition are no longer valid, the alarm flag resets automatically to zero.

2) For Testbench Code:

1. Module Declaration and Instantiation:

- The alarm_clock_TB module serves as the testbench designed to verify the behavior of the Verilog module alarm_clock.
- The alarm_clock module has the following ports connected in the testbench:

    - clk: An input signal representing the clock.
    - rst: An input signal used to reset the clock.
    - rst_min: A 6-bit input to set the minute value during reset.
      rst_hr: A 5-bit input to set the hour value during reset.
    - a_min: A 6-bit input representing the alarm minute setting.
    - a_hr: A 5-bit input representing the alarm hour setting.
    - sec: A 6-bit output providing the current seconds count.
    - min: A 6-bit output providing the current minutes count.
    - hr: A 5-bit output providing the current hour count.
    - alarm_flag: A 1-bit output flag that is set high when the alarm condition is triggered.

2. Clock Generation:

- The testbench generates a clock (clk) signal using an always block.
- This clock toggles every 1 time unit (e.g., nanosecond), creating a square wave with a period of 2 time units and a 50% duty cycle.
- In this simulation setup, each tick of the clock corresponds to 1 second of real time in the clock module.

3. Initialization and Simulation:

- The initial block initializes the clock module inputs.
    - rst is set to 0 to keep the clock in normal mode.
    - rst_min and rst_hr are both initialized to 0, so the clock starts counting from 00:00 (midnight).
    - a_min is set to 2 and a_hr to 0, configuring the alarm to go off at 00:02.

- A reset operation is performed:
    - After 5 time units, rst is set high to activate the reset.
    - After another 5 time units, rst is set back to 0 to release the reset and allow normal operation.

- The simulation is allowed to run for 300 time units:
    - This duration is sufficient to let the clock count past 2 minutes and observe the alarm flag going high.

- $shm_open and $shm_probe system tasks are used to enable signal monitoring for waveform viewing.
- The simulation ends after 300 time units using $finish.

Verilog code:

```
module alarm_clock(

   input clk,
   input rst,
   input [5:0] rst_min, // Max: 60
   input [4:0] rst_hr,  // Max: 24
   input [5:0] a_min,
   input [4:0] a_hr,
   output [5:0] sec,
   output [5:0] min,
   output [4:0] hr,
   output alarm_flag);
   // Time registers
   reg [5:0] curr_sec = 0;
   reg [5:0] curr_min = 0;
   reg [4:0] curr_hr = 0;
   always @(posedge clk or posedge rst) begin
      if (rst) begin
         curr_sec <= 6'd0;
         curr_min <= rst_min;
         curr_hr  <= rst_hr;
      end else begin
         if (curr_sec == 6'd59) begin
            curr_sec <= 6'd0;
            if (curr_min == 6'd59) begin
               curr_min <= 6'd0;

               if (curr_hr == 5'd23)
                  curr_hr <= 5'd0;
               else
                  curr_hr <= curr_hr + 1;
            end else begin
               curr_min <= curr_min + 1;
            end
         end else begin
            curr_sec <= curr_sec + 1;
         end
      end
   end
   // Assign current time
   assign sec = curr_sec;
   assign min = curr_min;
   assign hr  = curr_hr;
```

```
   assign alarm_flag = (curr_min == a_min && curr_hr == a_hr) ?
1'b1 : 1'b0;

endmodule
```

Testbench code:

```
module alarm_clock_TB;
   reg clk;
   reg rst;
   reg [5:0] rst_min;
   reg [4:0] rst_hr;
   reg [5:0] a_min;
   reg [4:0] a_hr;
   wire [5:0] sec;
   wire [5:0] min;
   wire [4:0] hr;
   wire alarm_flag;
   alarm_clock dut (
      .clk(clk),
      .rst(rst),
      .rst_min(rst_min),
      .rst_hr(rst_hr),
      .a_min(a_min),
      .a_hr(a_hr),
      .sec(sec),
      .min(min),
      .hr(hr),
      .alarm_flag(alarm_flag)
   );
   initial clk = 0;
   always #1 clk = ~clk;
   initial begin
      $shm_open("shm.db", 1);
      $shm_probe("AS");
      rst = 0;
      rst_min = 6'd0; // Start at 0 minutes
      rst_hr  = 5'd0; // Start at 0 hours
      a_min   = 6'd2; // Alarm at 2 minutes
      a_hr    = 5'd0;
      #5 rst = 1;      // Apply reset
      #5 rst = 0;      // Release reset
      #300 $finish;
   end
endmodule
```

Timing diagrams:

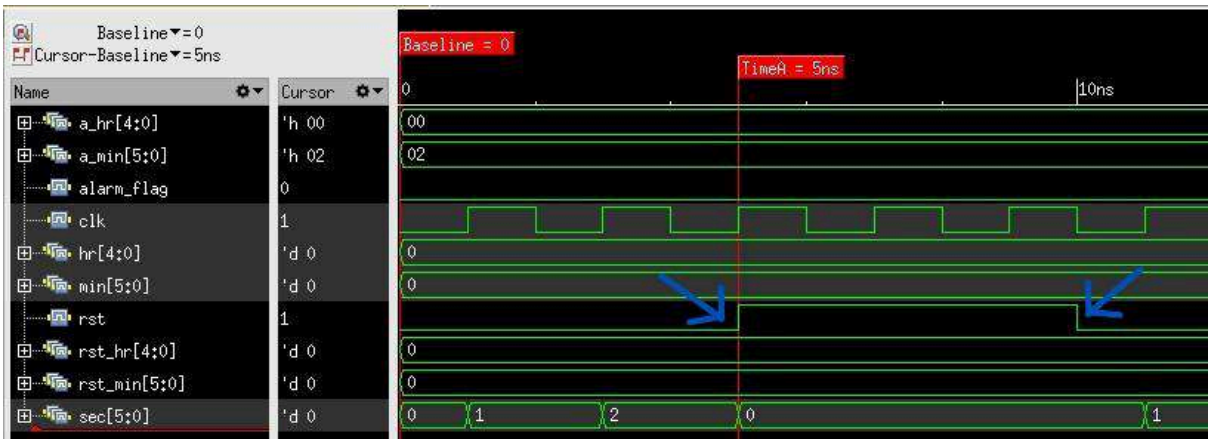


Fig: The timing diagram shows the reset function at 5ns, and again set at 10ns

The timing diagram shows the behavior of the digital alarm clock circuit during a reset condition and its return to normal operation. At 5ns, the reset signal goes high, which activates the reset condition. As a result, the output values for seconds, minutes, and hours are updated according to the reset inputs: seconds are set to 0, minutes to 2 (from reset_min), and hours to 0 (from reset_hour). During this period, the alarm flag remains low, as no condition for alarm activation is met yet. Then, at 10ns, the reset signal is pulled low, which disables the reset state. After this, the clock resumes normal operation, and the time starts incrementing with each clock pulse. This behavior confirms that the reset logic works correctly and that the circuit transitions smoothly from a reset condition to regular timekeeping. The diagram clearly illustrates the effect of reset and helps verify that all counters are properly initialized and controlled based on the timing of the reset signal.

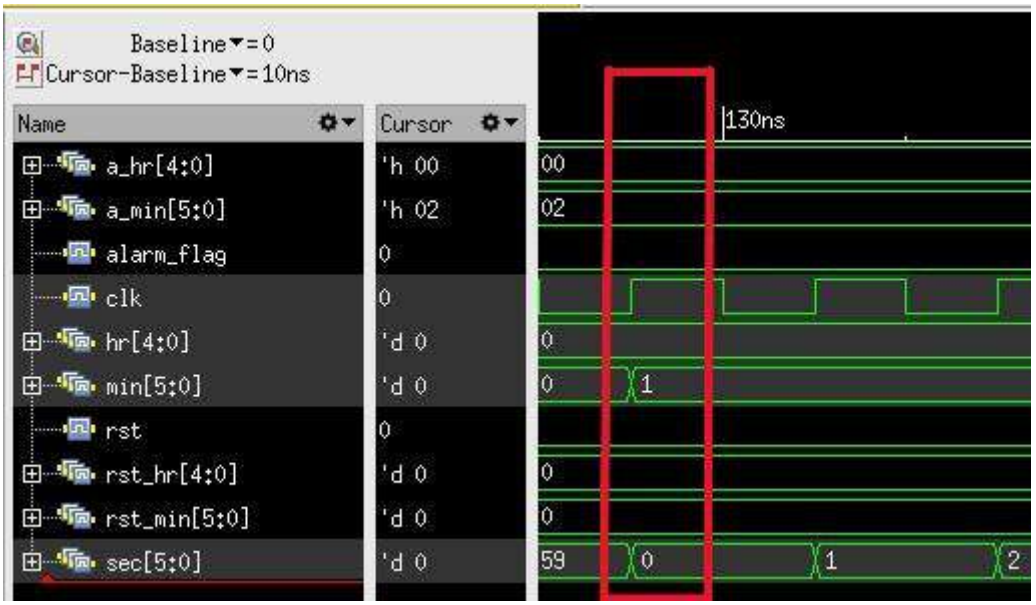


Fig: The timing diagram at 1 minute.

The timing diagram illustrates the transition from 59 seconds to 1 minute, confirming that the digital clock module correctly updates time. Initially, the second counter reaches 59, and at the next positive clock edge, it resets to 0 while the minute counter increments from 0 to 1. This is highlighted in the red box, showing the critical moment when a full minute passes. The hour value remains unchanged since only the minute is affected at this point. This behavior verifies that the second-to-minute rollover logic is functioning properly. The alarm flag remains low, indicating that the set alarm time has not yet been matched. Overall, the diagram confirms correct time progression in response to continuous clock pulses and demonstrates that the clock accurately tracks elapsed time across minutes.

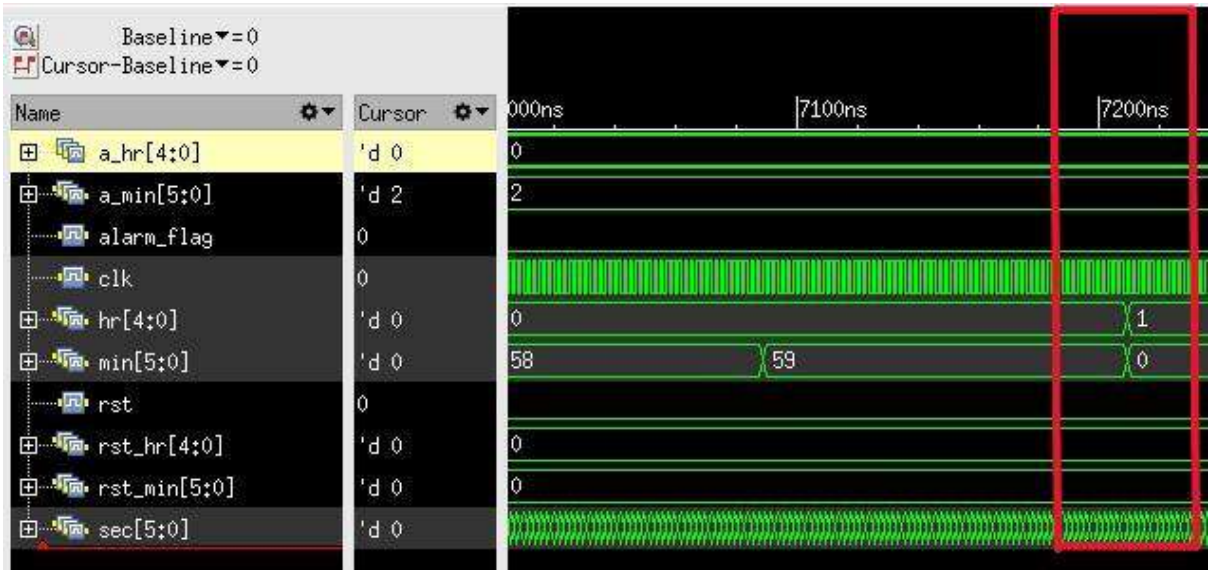


Fig: The timing diagram at 1 hour.

This timing diagram highlights the transition from 59 minutes to 1 hour, which is a critical checkpoint in validating the proper functionality of a digital clock. As shown in the red box, the minute register reaches its maximum value of 59, and at the next positive edge of the clock, it resets to 0. Simultaneously, the hour register increments from 0 to 1. The second register also resets to 0 after reaching 59, maintaining correct synchronization across all three time units. This transition confirms that the cascading update mechanism—where seconds overflow into minutes and minutes overflow into hours—is implemented accurately. The waveform provides clear visual proof that the clock handles boundary conditions correctly, ensuring continuous and consistent timekeeping. The alarm flag remains inactive in this scenario, as the alarm condition is not met. This event further validates the clock's ability to operate reliably across extended periods without error or misalignment.

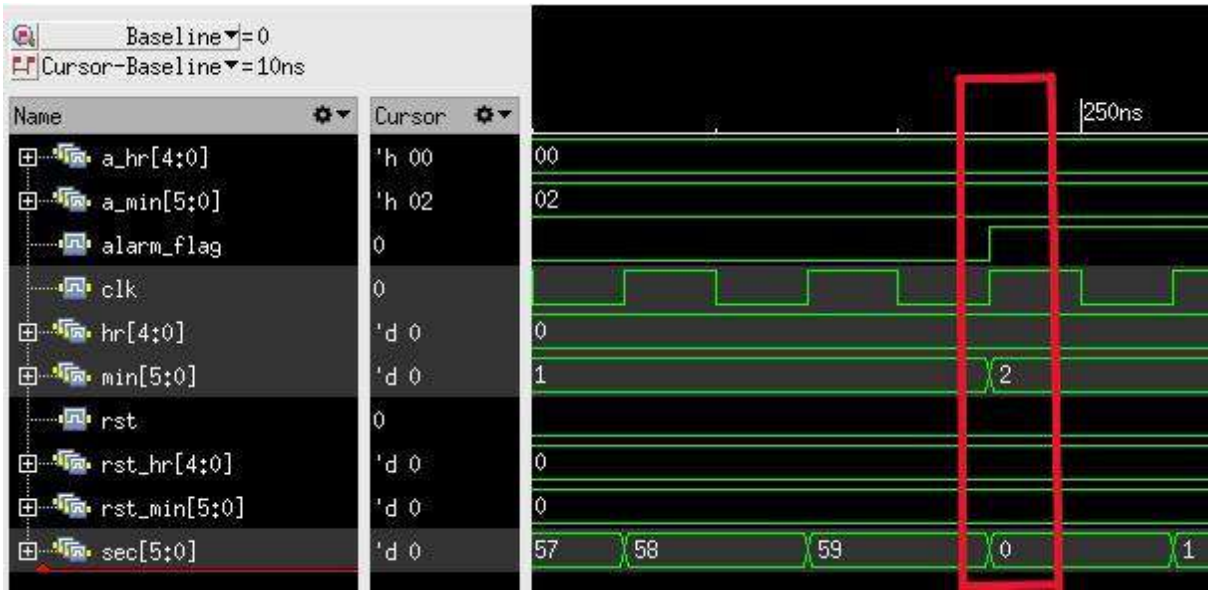


Fig: The timing diagram shows the alarm flag rising at 2 minutes.

This timing diagram demonstrates the triggering of the alarm at the correct time. As shown, the seconds count transitions from 59 to 0, and the minute value increments from 1 to 2, while the hour remains unchanged. According to the design, the alarm is programmed to activate when the current time matches the alarm time, which is set to 2 minutes and 0 hours. At exactly this point, the alarm flag goes high, as highlighted by the red box. This confirms that the alarm detection logic is functioning correctly, comparing the current hour and minute against the predefined alarm time and setting the alarm flag accordingly. The diagram verifies that the alarm signal rises precisely when expected, showcasing accurate synchronization between timekeeping and alarm generation within the system.

RTL Synthesis:
To perform the RTL synthesis of the digital alarm clock, we initially need to create five folders. These are as follows:

- rtl
- synthesis
- constraint
- simulation
- lib

These were made using the following terminal commands:

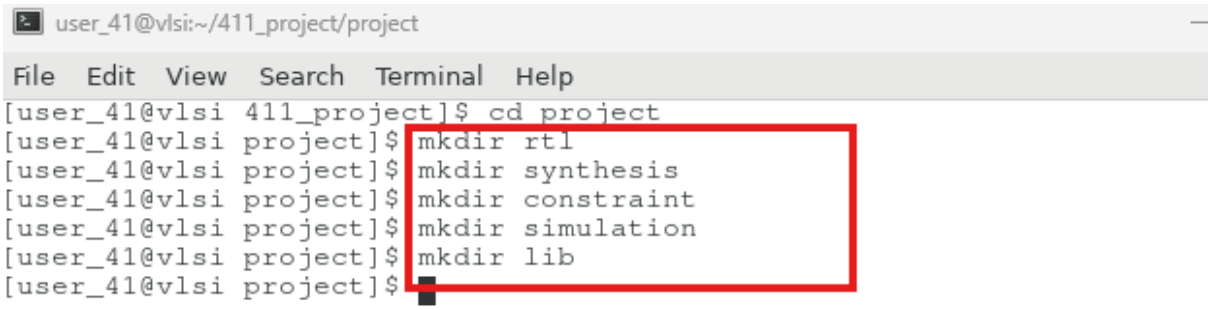


Figure: *Creating the necessary folders for RTL synthesis.*

Then, we copied the fast.lib and slow.lib files using the following commands to the lib directory.

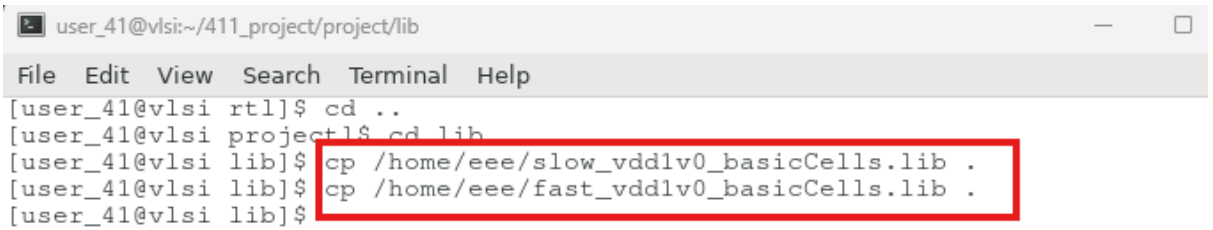


Figure: *Copying the fast and slow libraries.*

Logical synthesis is performed on RTL design with both fast and slow libraries with Synopsys Design Constraints (SDC) to generate a gate-level netlist and power reports.

The constraint file in the constraint folder had to be significantly modified by our Verilog design code.

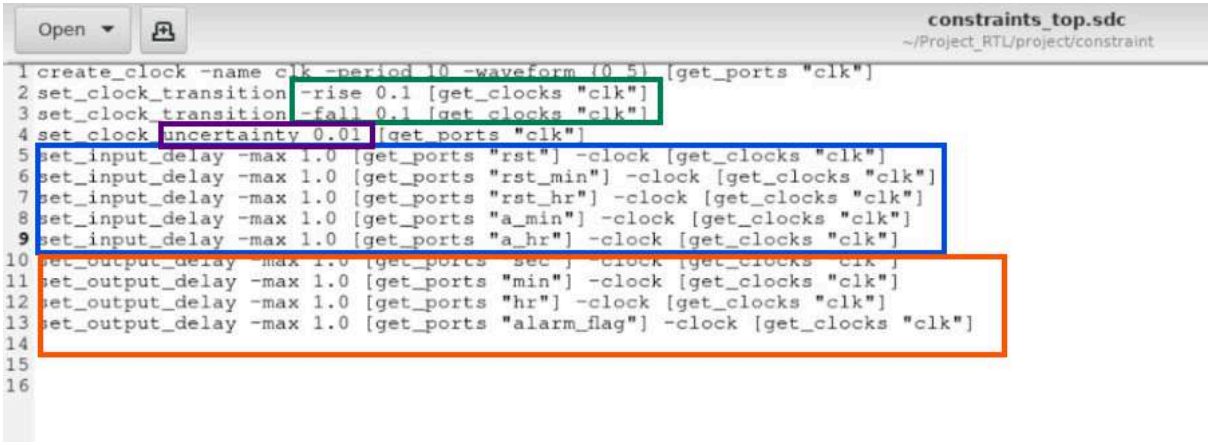


Figure: *Constraints File.*

A clock named clk with a period of 10 units and a waveform of {0 5}. The clock variable was originally named 'clk' and thus did not have to be changed in the constraint file. However, there are 5 inputs in our design, so they had to be typed manually in the constraints_top.sdc.sdc.sdc file (outlined in blue). Similarly, there are 4 outputs (outlined in orange). The rise time and fall time of the clock were set to 0.1 units (outlined in green), and the corresponding uncertainty was set to 0.01 (outlined in purple).

Finally, we had to update the rc_script.TCL file for RTL synthesis for synthesis using the fast and slow libraries.

```
set_db lib_search_path ../lib/
set_db hdl_search_path ../rtl/
set_db library fast_vdd1v0_basicCells.lib
read_hdl alarm_clock.v
elaborate
read_sdc ../constraint/constraints_top.sdc
syn_generic
syn_map

#synthesize -to_mapped -effort medium
write_hdl > alarm_clock_netlist.v
write_sdc > alarm_clock_sdc.sdc
```

Figure: *rc_script.tcl for RTL synthesis using fast library.*

```
set_db lib_search_path ../lib/
set_db hdl_search_path ../rtl/
set_db library slow_vdd1v0_basicCells.lib
read_hdl alarm_clock.v
elaborate
read_sdc ../constraint/constraints_top.sdc
syn_generic
syn_map

#synthesize -to_mapped -effort medium
write_hdl > alarm_clock_netlist.v
write_sdc > alarm_clock_sdc.sdc
```

Figure: *rc_script.tcl for RTL synthesis using fast library.*

Results of RTL Synthesis:

Schematic View:

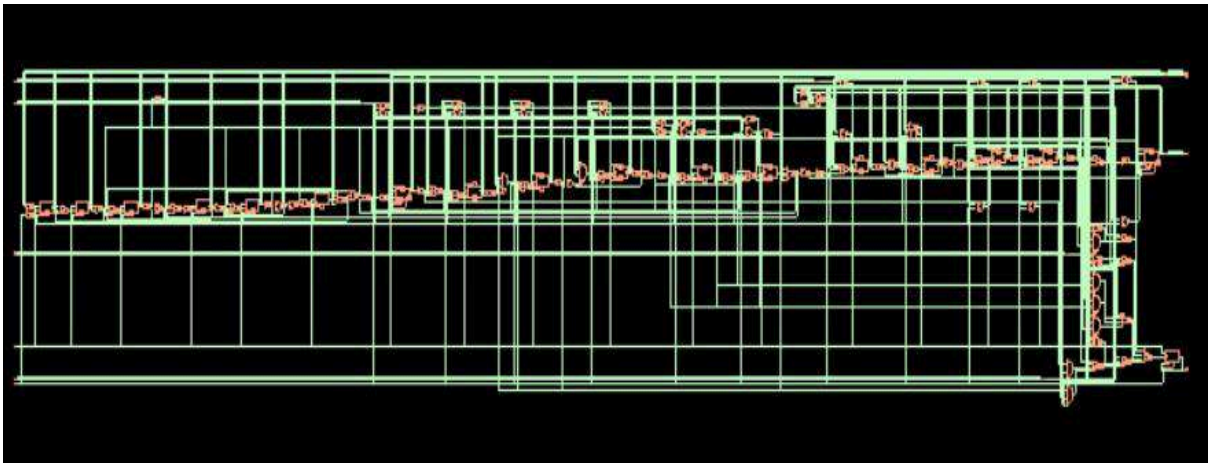


Figure: *Schematic view using the fast library.*

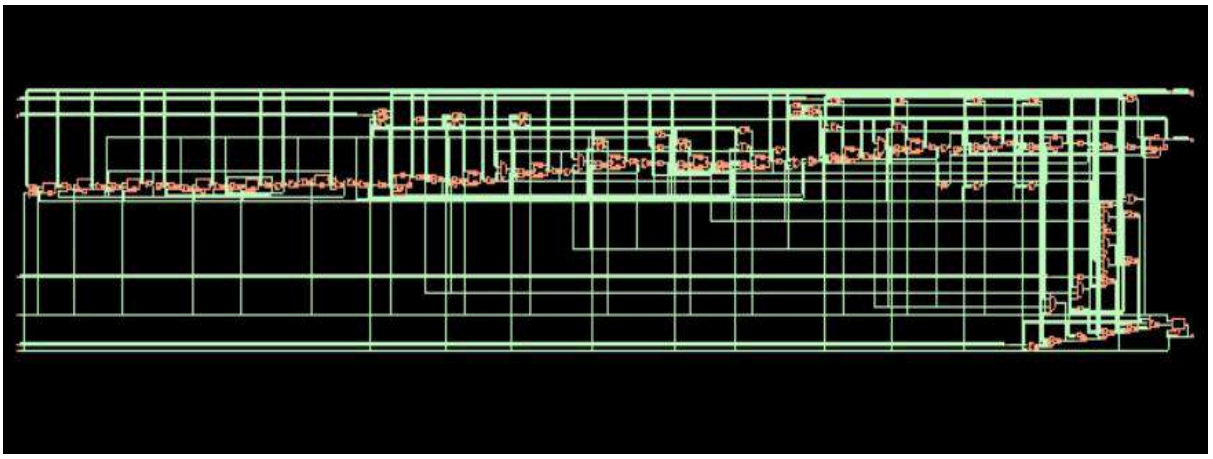


Figure: *Schematic view using the slow library.*

Power Details Report:

Power Details Report

Generated by: Genus(TM) Synthesis Solution 21.18-s082_1 (Jul 18 2023 13:08:41)
Generated on: Apr 25 2025 09:11:50
Module: design:alarm_clock
Technology library: fast_vdd1v0 1.0
Operating conditions: PVT_1P1V_0C (balanced_tree)
Wireload mode: enclosed

Instance	Cells	Leakage (nW)	Internal (nW)	Net (nW)	Switching (nW)
alarm_clock	115	16.223	8971.265	1425.914	10397.179

Figure: *Power Details Report For Fast Library.*

Power Details Report

Generated by: Genus(TM) Synthesis Solution 21.18-s082_1 (Jul 18 2023 13:08:41)
Generated on: Apr 25 2025 09:07:15
Module: design:alarm_clock
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Instance	Cells	Leakage (nW)	Internal (nW)	Net (nW)	Switching (nW)
alarm_clock	118	5.457	5511.758	741.258	6253.015

Figure: *Power Details Report For Slow Library.*

Netlist Statistics:

Netlist Statistics

Generated by: Genus(TM) Synthesis Solution 21.18-s082_1 (J
Generated on: Apr 26 2025 20:29:11
Module: design:alarm_clock
Technology library: fast_vdd1v0 1.0
Operating conditions: PVT_1P1V_0C (balanced_tree)
Wireload mode: enclosed

Type	Instance	Area	Area_%
Sequential	18	168.264	54.365
Logic	77	127.566	41.215
Inverter	20	13.680	4.420
Total	115	309.510	100

Figure: *Netlist Statistics For Fast Library.*

Netlist Statistics

Generated by: Genus(TM) Synthesis Solution 21.18-s082_1 (
Generated on: Apr 26 2025 20:23:15
Module: design:alarm_clock
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Type	Instance	Area	Area_%
Sequential	18	168.264	54.545
Logic	77	124.488	40.355
Inverter	23	15.732	5.100
Total	118	308.484	100

Figure: *Netlist Statistics For Slow Library.*

Gate Count Report:

Gate Count Report

Generated by: Genus(TM) Synthesis Solution 21.18-s082_1 (Jul 18 2023 13:08:41)
Generated on: Apr 26 2025 20:28:43
Module: design:alarm_clock
Technology library: fast_vdd1v0 1.0
Operating conditions: PVT_1P1V_0C (balanced_tree)
Wireload mode: enclosed

Instance	Cells	Cell Area	Net Area	Total Area	Wireload	WL
alarm_clock	115	309.510	0.000	309.510		(S)

Figure: *Gate Count Report For Fast Library.*

Gate Count Report

Generated by: Genus(TM) Synthesis Solution 21.18-s082_1 (Jul 18 2023 13:08:41)
Generated on: Apr 26 2025 20:22:42
Module: design:alarm_clock
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Instance	Cells	Cell Area	Net Area	Total Area	Wireload	W
alarm_clock	118	308.484	0.000	308.484		(S)

Figure: *Gate Count Report For Slow Library.*

Results and Observations:

In this project, we successfully designed and simulated a digital alarm clock using Verilog HDL, incorporating both reset and alarm functionalities. The simulation results were carefully analyzed using waveform diagrams generated in the Cadence environment, providing a clear representation of the system's behavior under different conditions.

To build a clock, first, we had to build a timestamp to establish it as a timing reference. In our simulation, the clk toggles every 1 time unit, resulting in a complete cycle every 2 time units. We assume that each positive edge of clk represents the passing of one second in our clock's logic. As a result, while the simulation runs in time units, we establish our time resolution, with one clk pulse representing one second of clock time. Based on this resolution, the waveform's values for seconds, minutes, and hours are changed. This simplification enables us to simulate the passage of real time in a controlled and visible manner during the simulation.

The configured rst_hr and rst_min values are used to update the current hour and minute values when the reset signal (rst = 1) is asserted. We currently have rst_hr and rst_min set to 0 (00000 and 000000, respectively) in our testbench configuration. Five time units are used to assert the reset signal, and ten time units—or five to ten simulated seconds—are used to deassert it. The internal time of the clock is reset on the positive edge of the clock during this period, clearing the current hour, minute, and second to zero. This guarantees synchronization with the designated initial values and enables the clock to begin from the first.

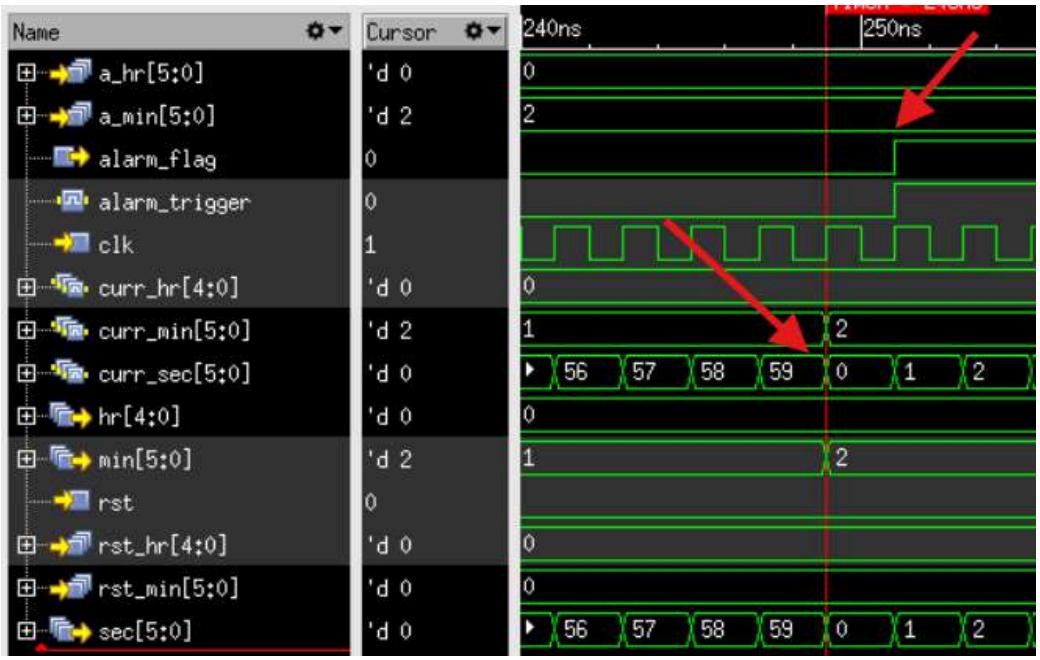
In our digital clock system, when the hour and minute readings match the set alarm time, the alarm flag rises. In our configuration, a_min and a_hr are used to set the alarm time. If a_min is set to 2, and a_hour is set to 0 (we are considering binary values), the alarm flag rises precisely when the clock reaches 2 minutes and 0 hours. Every clock cycle, the system internally checks for this condition. The signal alarm_flag becomes 1, signifying that the alarm is active, if the current time coincides with the alarm time. The alarm is turned off (alarm_flag goes back to 0) if the time no longer matches (for example, after another minute).

The timing diagrams captured during simulation visually confirmed each phase of operation. They showed the precise moment of reset activation and deactivation, the minute transition from 59 seconds to the next minute, and the triggering of the alarm at the correct time. These waveforms matched the expected output behavior based on our Verilog design and confirmed the correctness of both reset and alarm logic.

Overall, the simulation outcomes validated the intended functionality of the digital alarm clock. The clock counted time accurately, reset correctly to specified values, and reliably triggered the alarm when the set time was reached. These observations confirm that our design meets the required specifications and behaves as expected under various operating conditions.

Discussion

During this project, a major problem we came across was when the alarm_flag was being raised 1 clock cycle after the set alarm time as shown below:



This is because the alarm checking logic was inside the always @(posedge clk or posedge rst) block as shown below:

```
always @(posedge clk or posedge rst)
begin

   //code for updating curr_min and curr_hr

   if (curr_min == a_min && curr_hr == a_hr)
      alarm_flag <= 1;
   else
      alarm_flag <= 0;
end
```

Due to the usage of nonblocking assignments (<=) throughout this always block, all the variables are updated at the end of the block. Due to this, any assignments or logic checking use the previous values of those variables. As the code runs line by line, when it checks if the current time is equal to the alarm time, it is comparing the alarm time with the previous value of the current time, not the updated one, as the current time has not been updated yet.

As the code reaches the end of the always @(posedge clk or posedge rst) block, the value of current time updates to the alarm time we set. At the same time, 1 positive edge has passed. At the next positive edge, the always block is run again. Now in the alarm checking logic, the code compares the alarm time to the new current time. As they are equal, the alarm flag is raised.

This can simply be fixed by bringing the alarm checking logic outside the always @(posedge clk or posedge rst) block, as shown below

```
always @(posedge clk or posedge rst) begin

   //code for updating curr_min and curr_hr

end

assign alarm_flag = (curr_min == a_min && curr_hr == a_hr) ? 1'bl : 1'b0;
```

Here, the updated current time is compared with the set alarm time. This, along with the usage of blocking or combinational assignment, any change is reflected instantly, without waiting for the next positive edge. As a resul,t we get the output shown below: