Project 2: Alarm Clock

Kevin Lui, Justin Chu, Leon Tran
luik@oregonstate.edu
chuju@oregonstate.edu
trane3@oregonstate.edu
ECE 271
12/02/24

Table of Contents

Alarm Clock:

The alarm clock uses a 50 MHz core clock to track time, allowing users to set an alarm in hours and minutes. When enabled, it outputs a signal to drive a speaker with a melody when the alarm time is reached.

Overview of Alarm Clock Project:

The purpose of the project is to design and validate a modular, reliable clock system with an integrated alarm functionality. This project implements an alarm clock system driven by a 50 MHz core clock to maintain accurate timekeeping. The design utilizes modular SystemVerilog components, including counters for seconds, minutes, and hours, to track and display the current time on a seven segment display. As shown in **Figure 1**, it illustrates the system, detailing the primary inputs for time setting, alarm configuration, and control signals, as well as outputs for user display and speaker activation. Separate counters track seconds, minutes, and hours, while independent alarm counters allow users to set and save an alarm time without interfering with the running clock. Comparators detect when the running clock matches the saved alarm time, triggering a signal output for a speaker. The design supports waveform modulation to produce a melody, with all functionality implemented through modular, reusable components.

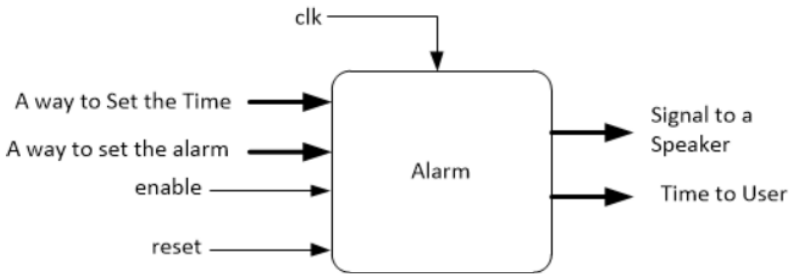


Figure 1. Block diagram of alarm clock system showing inputs for clock signal, time and alarm settings, enable, and reset, with outputs for speaker signal generation and time display.

Reflection on Project:

Initially, we implemented the sound system alarm using LEDs, as we didn't have a speaker available at the time. While the SystemVerilog coding wasn't particularly challenging, we encountered significant difficulties with the hardware setup due to our unfamiliarity with the DE10-Lite board. The DE10-Lite manual proved to be an invaluable resource, particularly in understanding the 2x20 GPIO connections, including the 3.3V power and ground pins. We also relied on YouTube tutorials to learn about speaker setups and wiring techniques.

Our first prototype used a breadboard and a large 8-ohm resistor speaker. However, this setup requires additional components like alligator clips and a 100-ohm resistor to prevent overpowering the speaker, which made the process unnecessarily complicated. After further consideration, we decided to switch to a smaller 32-ohm speaker from TekBots, which simplified the setup and eliminated the need for extra components.

On the SystemVerilog side, we adopted an incremental approach to testing. We started with basic LED blinking to confirm that the square wave signal was being generated correctly. Once that was working, we moved to playing two notes using LEDs to simulate a melody. Finally, we connected the speaker and tested the system by playing a single note, followed by two notes, and then a three-note melody for the final implementation. This step-by-step process allowed us to gradually refine both the hardware and software aspects of the project.

Setting up the alarm was fairly straightforward as it was essentially the clock circuit, but we replaced the clock input with a button input. Initially, we also were not sure about how we wanted to have the alarm increment, and we felt that just incrementing with seconds was too slow. By using switches alongside the button and an and gate, we were able to increment the seconds, minutes, and hours separately or at the same time, which made it easier to set different times outside of a couple of minutes.

Before deciding on a switch and button input to change the alarm time values, we went through a number of iterations, starting with just a button to increment seconds. Although it was a proof of concept, this design was too hard to use when the user wanted to set the time to be more than a few minutes. The next iteration used the switches flipping to increment the seconds, minutes, or hours. This design was easier to set, but constantly flipping switches was still fairly tedious. That was when we settled on our final design which used a combination of switches and a button.

System Architecture and Design:

The system integrates three core components, clock, alarm, and audio generation, working together to provide timekeeping and alarm functionality. The clock provides time outputs, which are compared to the alarm's stored time by comparators. On a match, the comparators trigger an audio module to generate a waveform, activating the alarm.

Clock Section:

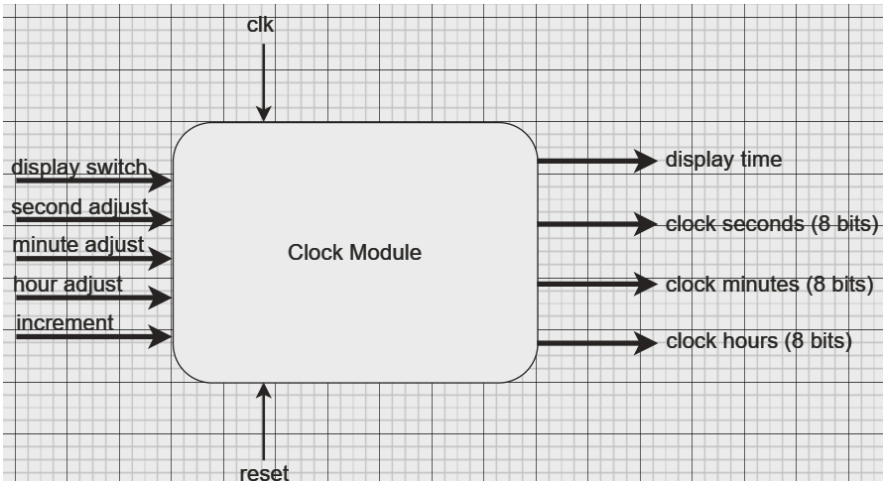


Figure 2. Block diagram for clock module. This module generates time outputs (seconds, minutes, hours), supports time adjustments via input switches, and displays current time.

The clock is heavily based on the Lab 5 clock, with the only difference being the seven segment driver. Users can set the clock time using adjustment inputs for seconds, minutes, and hours, along with an increment button. The clock's outputs go to a BUSMUX which takes the output of the clock display and alarm display and displays one or the other based on a switch input.

Alarm Section:

Setting the alarm uses a similar design to the clock to increment and display the time, as shown in **Figure 3**. By flipping the leftmost switch, the display changes from the clock to the alarm. From there, the three rightmost switches can be flipped to allow the user to adjust the seconds, minutes, and hours, with the rightmost switch corresponding to seconds, the middle switch corresponding to minutes, and the left switch corresponding to hours. When one or more of the three switches are flipped, pressing the lower button will cause whichever corresponds to the switch(es) to increment by 1. Once the desired time has been input, flipping the second leftmost switch turns the alarm "on". Once the clock reaches the time set on the alarm, the alarm will be triggered.

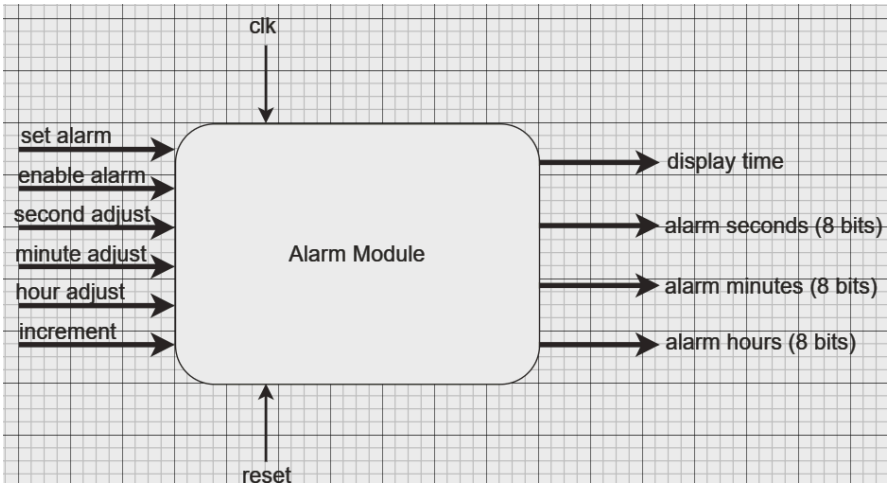


Figure 3. Block diagram for alarm module illustrating inputs including clock, reset, time adjustments switches, and alarm enable. Outputs are the time display and alarm trigger signal.

This trigger, as shown in **Figure 4**, uses comparators to compare the 12 values representing time, with 6 being the clock's values and 6 being the alarm's values. When the comparators see that all the times are equal, it sends a single pulse for that cycle. This pulse is sent to a DFF which only updates whenever the times are equal. The Q value of the DFF then switches from a constant low to a constant high, which is then sent to the sound generator.

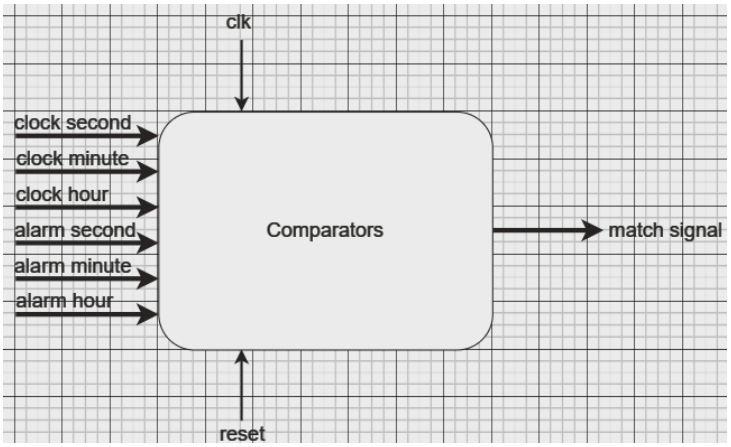


Figure 4. Block diagram for the comparators. The comparators compare the current clock time (seconds, minutes, hours) with the saved alarm time. When inputs match, it generates a single-cycle match pulse.

Audio Section:

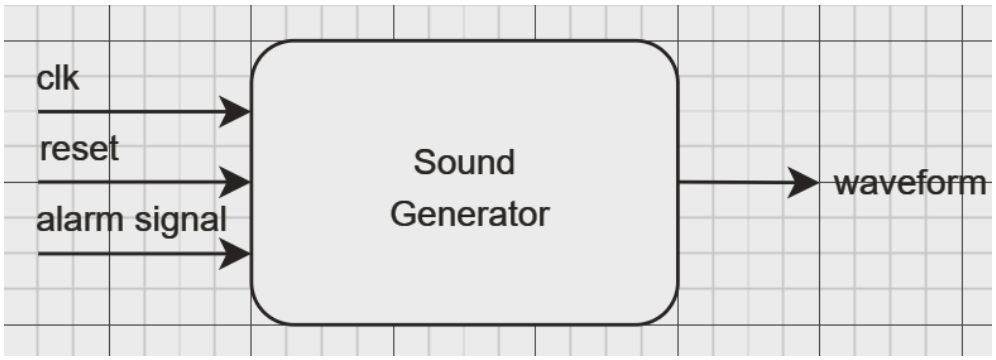


Figure 5. Block diagram of the sound generator module that generates the waveform output based on the clock input, reset signal, and alarm activation signal.

To generate the audio, the 50Mhz clock is inputted into the sound generator module as shown in **Figure 5**. Since 50 Mhz is too fast for audio frequencies, the module uses frequency division technique to produce square wave signals corresponding to musical notes.

To generate audible tones, the module uses frequency division technique. Each note G4, A4, or C5 has an associated DIVIDE_VALUE, which determines the clock division ratio.

Table: Frequency Division Values and Melody Notes

Note	Frequency (Hz)	Divide Value
G4	~392 Hz	63,775
A4	~440 Hz	56,818
C5	~523 Hz	47,778

The module plays a simple 3-note melody: G4 → A4 → C5. The notes are stored in an array called song. Each note is played for a second before moving to the next note. After the last note C5 the melody loops back to the note G4.

<u>Speaker Connection</u>

- Connect square_wave output to GPIO pin and speaker positive input.
    - Signal Name: GPIO_26
    - FPGA Pin No: PIN_AA7
    - I/O Standard: 3.3-V LVTTL
- Connect the negative input of the speaker to a GND pin on the FPGA board.

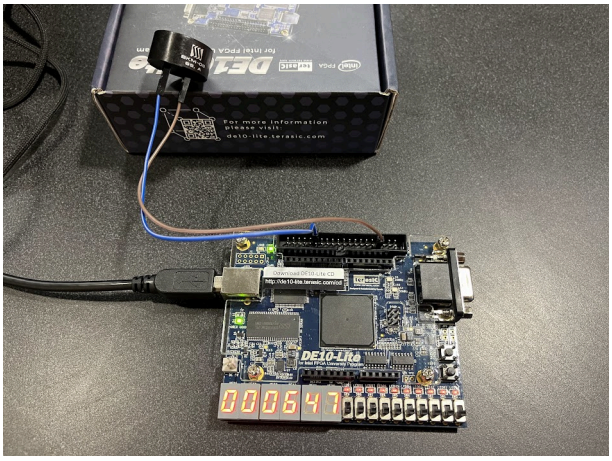


Figure 6. Complete alarm setup with speaker connection

Validation through Simulation:

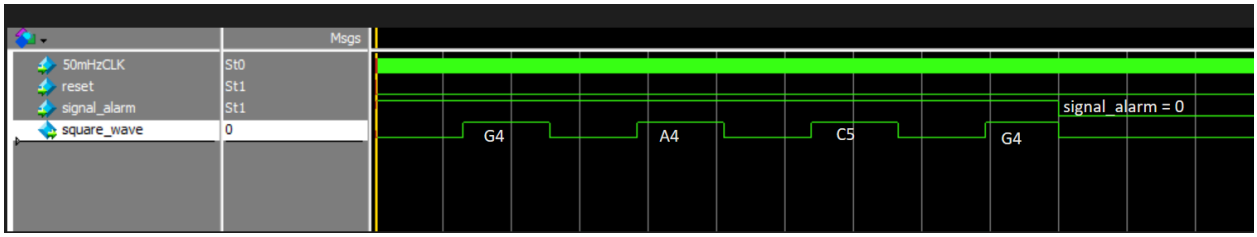
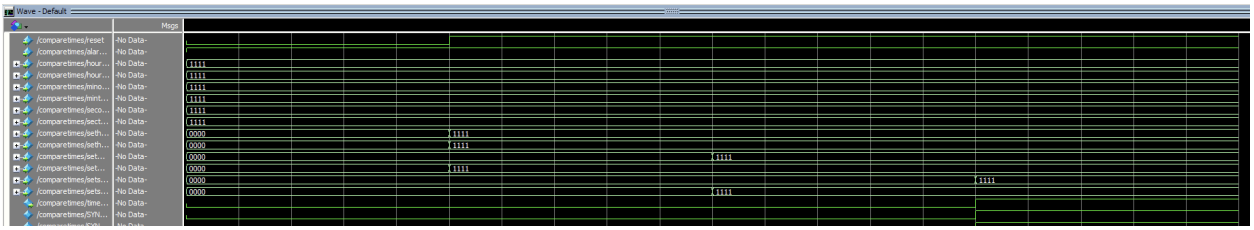


Figure 7. Simulation results for 15 ms showcasing note toggle, reset and signal alarm functionality

Time	Behavior
0 - 1 ms	No sound is generated; square wave = 0
1 - 4 ms	First note, G4 is played
4 - 6 ms	Second note, A4 is played
6 - 9 ms	Third note, C5 is played
9 - 10 ms	G4 note is looped back
10 - 15 ms	Sound stops. Signal_alarm = 0, square_wave = 0

The simulation demonstrates the behavior of the SoundGenerator module over 15 ms, focusing on note transitions, looping, and the effect of disabling the signal_alarm signal. The module is driven by a 50 MHz clock and produces a 3-note melody: G4 (~392 Hz), A4 (~440 Hz), and C5 (~523 Hz).



Time	Changes in Clock Values	Changes in Alarm Signal
0µs	Hours, minutes, and seconds for clock are 1111	None
10µs	Hours and ones for minutes for alarm are set to 1111	None
20µs	Tens for minutes and seconds for alarm are set to 1111	None
30µs	Ones for seconds for alarm are set to 1111	Raises to a value of 1

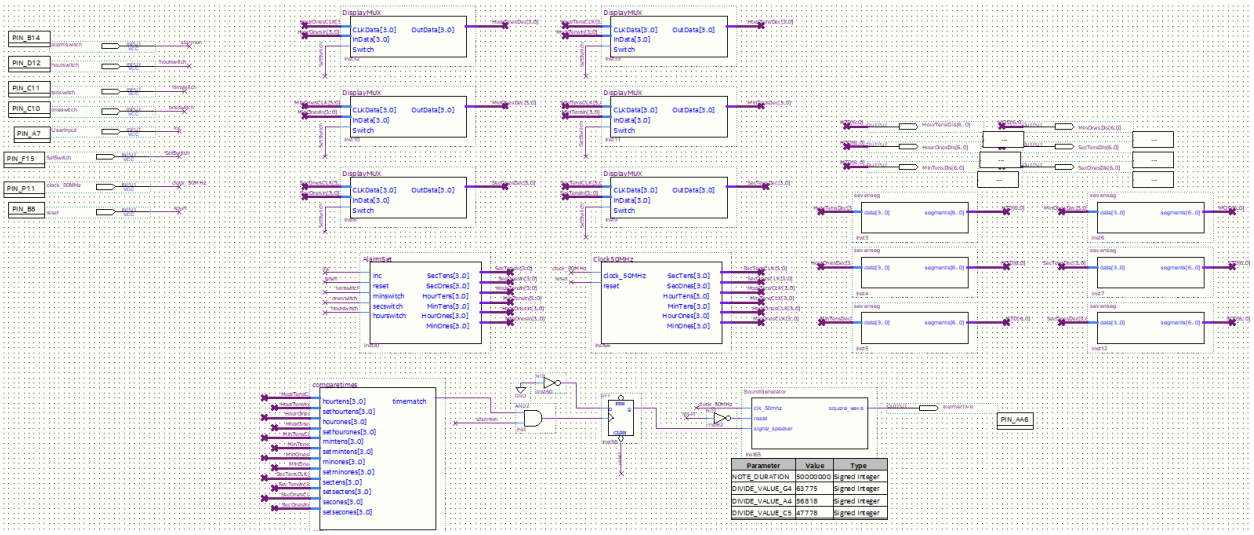
The simulation above shows that the signal to start the alarm, shown by the second from the bottom line, will not be active until all the times have the same values (1111 in this simulation).

Demonstration Video:

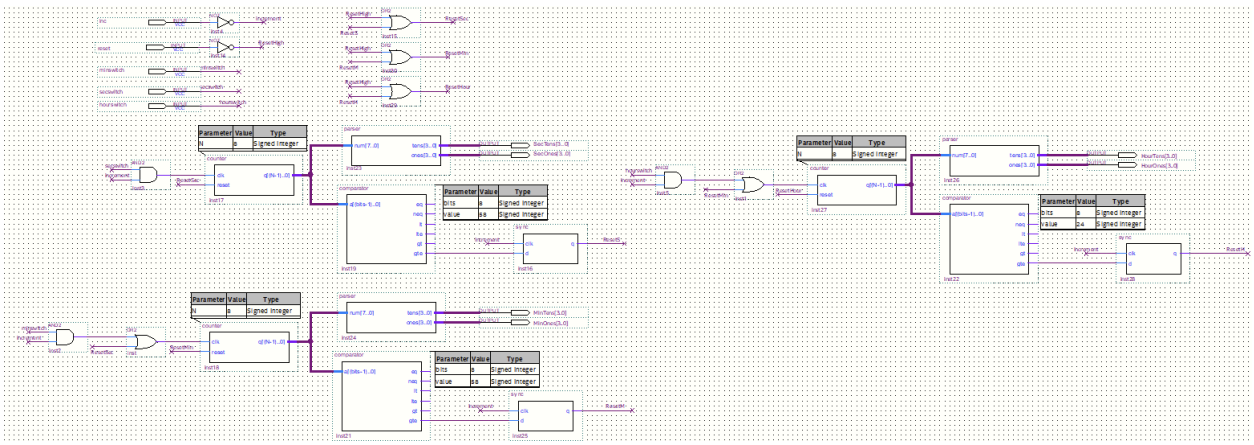
https://media.oregonstate.edu/media/t/1_h6xfm340

Appendix: Complete File Design:

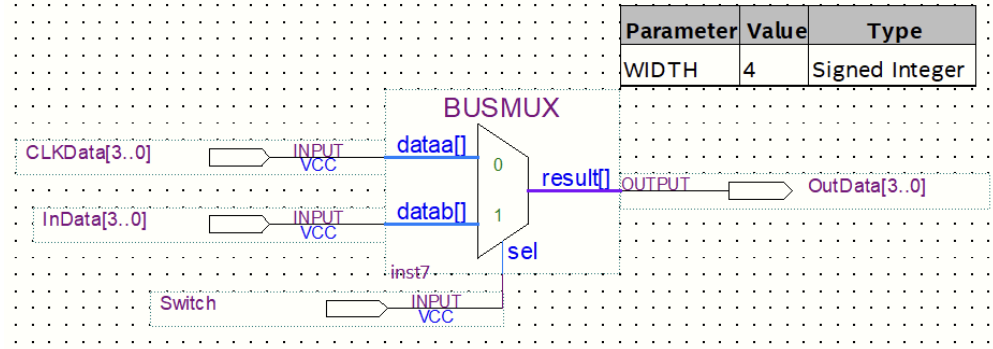
Alarm Clock Schematic



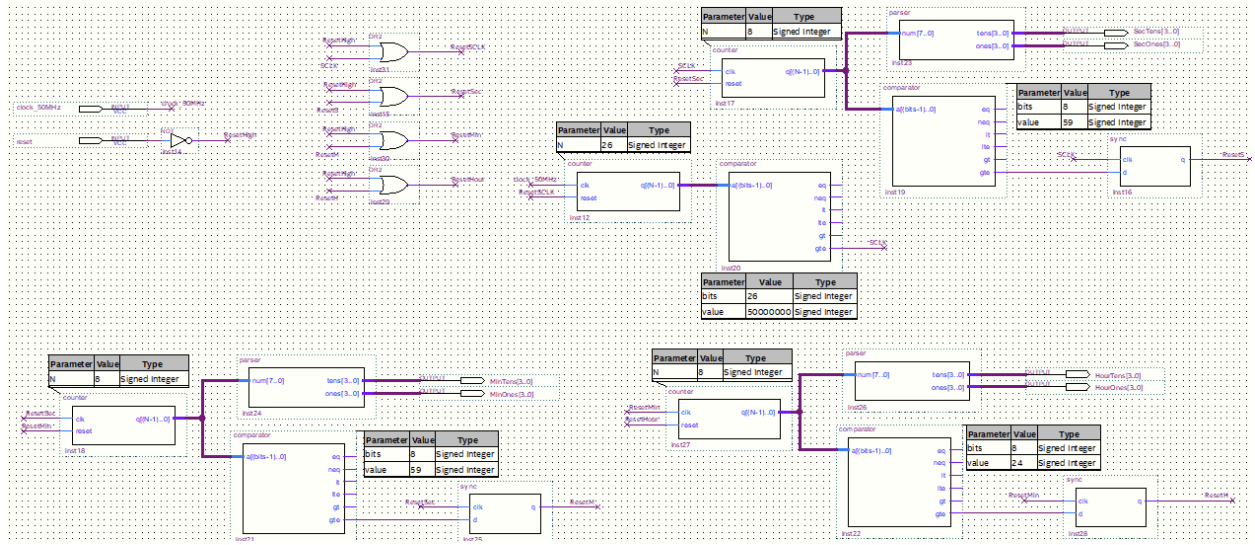
Alarm Clock Set Module



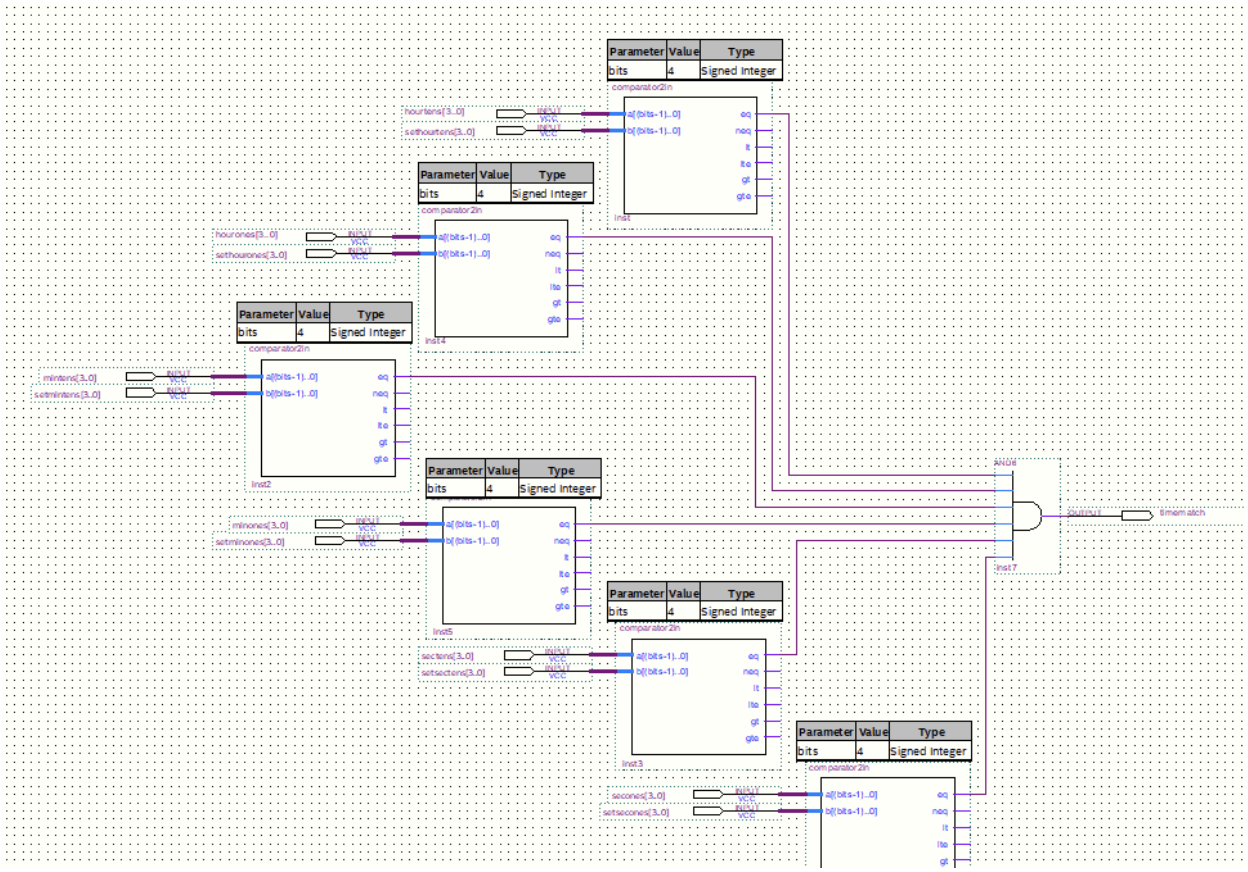
Display MUX Module



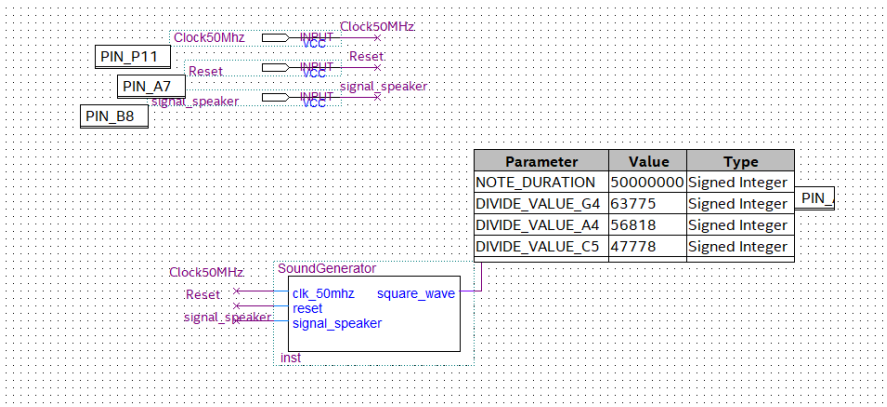
Clock Module



Time Comparison Module



Sound Generator Module:




```
module SoundGenerator(
    input clk_50mhz,
    input reset,
    input signal_speaker,     // enable signal to play the melody
    output reg square_wave    // output square wave for sound
);

    reg [24:0] counter;       // counter for frequency division
    reg [31:0] note_timer;    // timer for note duration
    reg [1:0] current_note;   // index of the current note
```

```
    reg [24:0] divide_value;  // frequency divider for the current note
    parameter NOTE_DURATION = 50_000_000; // duration of each note

    // frequency divider values for notes
    parameter DIVIDE_VALUE_G4 = 63775; // G4 (392 Hz)
    parameter DIVIDE_VALUE_A4 = 56818; // A4 (440 Hz)
    parameter DIVIDE_VALUE_C5 = 47778; // C5 (523 Hz)

    reg [24:0] song [0:2]; // melody with 3 notes
    initial begin
        song[0] = DIVIDE_VALUE_G4;
        song[1] = DIVIDE_VALUE_A4;
        song[2] = DIVIDE_VALUE_C5;
    end

    always @(*) begin
        divide_value = song[current_note]; // set divide value for current note
    end

    always @(posedge clk_50mhz or posedge reset) begin
        if (reset) begin
            counter <= 0;
            square_wave <= 0;
            note_timer <= 0;
            current_note <= 0; // reset to the first note
        end else if (signal_speaker) begin
            if (counter == divide_value - 1) begin
                counter <= 0;
                square_wave <= ~square_wave; // toggle square wave
            end else begin
                counter <= counter + 1;
            end

            if (note_timer == NOTE_DURATION - 1) begin
                note_timer <= 0;
                current_note <= current_note + 1; // move to the next note
                if (current_note == 2)
                    current_note <= 0; // loop back to the first note
            end else begin
                note_timer <= note_timer + 1;
            end
        end else begin
            counter <= 0;
            square_wave <= 0;
            note_timer <= 0;
            current_note <= 0;
        end
    end

endmodule
```

.do File:

```
vsim -gui work.SoundGenerator
add wave clk_50mhz reset signal_speaker square_wave counter
force clk_50mhz 0 0ns , 1 10ns -r 20ns
force reset 1 0ns , 0 20ns
force signal_speaker 1 0ns , 0 10ms
run 15ms
```