Alarm Clock in VHDL

Author: Tobias Thorgren
1FA326 Digital Electronics Design with VHDL

2025-05-16

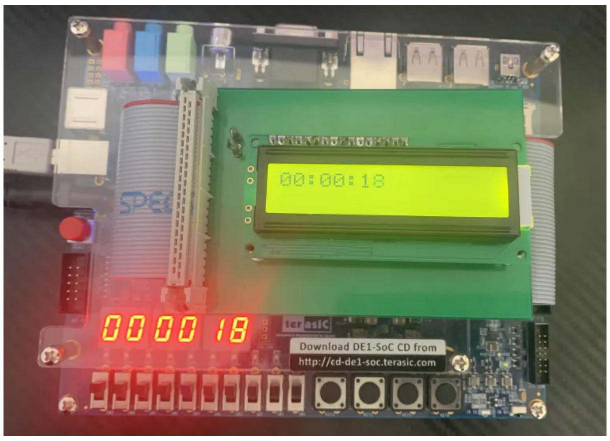


Figure 1: Final implementation, working alarm clock on DE1-SoC

Abstract

The project task was to design an alarm-clock. Some features include: showing the current time, changing the current time, setting an alarm, turning alarm on, turning alarm off and resetting all previous setting and the time back to 00:00:00.

The final design contains all of the features stated above along with some nice to haves, as visual indication of current mode and both indication of time on a LCD screen but also on the development kit, on the six, seven segment displays.

Contents

1 Introduction **3**

2 Project Description **3**

2.1 Tools & Software 3

2.2 Design 3

3 Theory **4**

3.1 DE1-SoC 4

3.2 VHDL 4

3.3 HD44780 5

3.4 Debouncing 5

3.5 ASCII 5

3.6 State Machine 6

4 Implementation **6**

4.1 Clk 1Hz 7

4.2 Mode Selector 7

4.3 Clock Counter 9

4.4 Alarm Control 10

4.5 Clock Merge 11

4.6 Clock to LCD 13

4.7 Two-Digit Alarm Display 14

4.8 Complete Design 15

5 Tests & Results **17**

5.1 Signaltap 17

5.2 Simulations 18

6 Conclusions and Evaluation **19**

1 Introduction

Although the alarm clock has seen a decline of use since the development of mobile phone, by having a dedicated alarm one has an easy excuse to why one did not wake up in time when the alarm malfunctions.

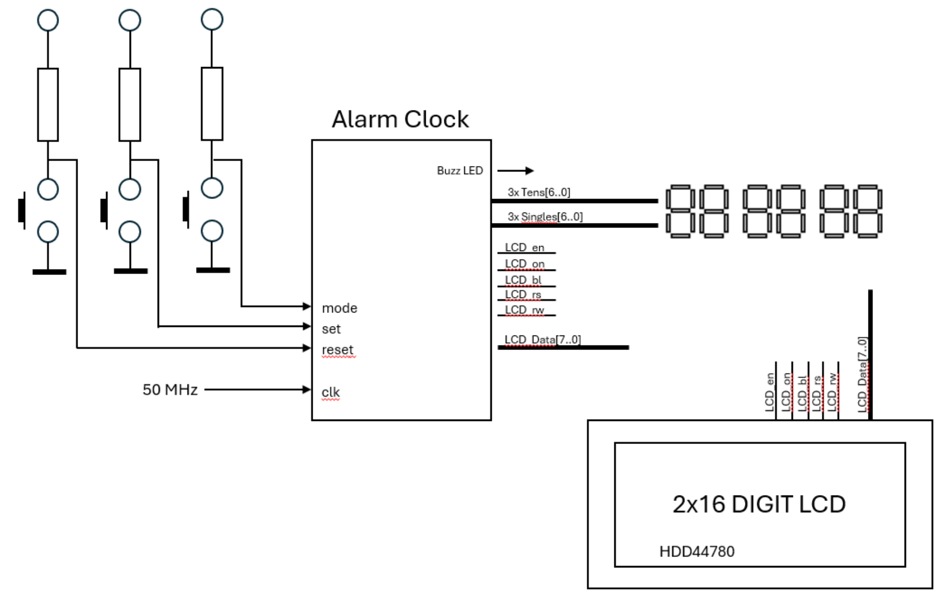


Figure 2: Overview of the Alarm Clock design

Along with being on time. This project gave some valuable knowledge as completing a given task with minimal outside help. I had to start from an vague idea of how how the clock could work and slowly progress by completing smaller milestones on at the time. I would say I enhanced my fluency in the VHDL language through this hands on project.

2 Project Description

2.1 Tools & Software

For compilation and simulation the project was constructed using Intel's FPGA development tool, *Quartus Prime Lite Edition* software. The project was later implemented on a DE1-SoC Development Kit along with a HDD44780 2x16 LCD display.

2.2 Design

The overview of the design works by acquiring 4 inputs whom of which are a 50 megahertz clock signal, a reset button, mode button and set button. The outputs consists of 4 LED's, 3 for showing current mode and 1 for indication when the alarm goes off. Then 6 output buses are used for the 7 segment displays as well as 13 pinouts for the LCD display.

The design consists of 7 different component parts/blocks. Each part has a unique operation.

The components include the following:

- **Clk 1Hz**, this unit scales down 50 megahertz clock down to a 1Hz signal with 50% duty cycle.
- **Mode Selector**, this unit reads the inputs from the user and selects the right mode from a state machine and send out the mode to coordinate with the other blocks.
- **Clock Counter**, this unit retrieves the 50 megahertz clock and the 1Hz clock and depending on the mode, it can ether continue a regular clock counting at 1 second per second or change the current time.
- **Alarm Control**, this unit is used when the alarm is turned on. By changing time when the alarm is on, one can change the time of the alarm goes off.
- **Clock Merge**, this unit combines the regular clock time and the alarm time. Depending on what mode is active the right outgoing signals are sent to the different displays.
- **Clock to LCD**, this unit finalizes the time given by clock merge and handles the LCD display's setup instructions and converting the time to ASCII for a good looking time indication.
- **Two Digit Alarm Display**, this unit was used for the development and converts the final time given by clock merge.

The complexity of each unit varies.

3 Theory

3.1 DE1-SoC

The development kit used in this project is the DE1-SoC, a robust hardware platform featuring an Altera (Intel) System-on-chip (SoC) FPGA. This design combines programmable logic with a Hard processor System (HPS), offering both flexible hardware and powerful processing capabilities. The platform's key feature is its dual-core ARM Cortex-A9 HPS, which enable advanced processing task while working parallel with the FPGA's logic. This architecture allow for efficient division of tasks between software processing (handled by the ARM cores) and hardware acceleration (in the FPGA). For the alarm clock, i primarily used the FPGA for custom digital circuits for timekeeping, display control and alarm indication. [1]

3.2 VHDL

Very High Speed Integrated Circuit Program Hardware Description Language (VHDL) is a hardware description language used for structuring and modelling digital system at multiple levels of abstraction. Levels include: logic gates, for design entry, documentation and, verification. [2]

3.3 HD44780

The HD44780 is a LCD controller. For this project the controller and LCD screen assembly was used together in a package for displaying the current time of the clock. The controller has several standard configurations. This project utilized the 8-bit mode of the controller and the Character Generator ROM with predefined alphanumeric characters. Each character has a corresponding 8-bit ASCII code. [3]

Pin	**Function**
RS	Selects registers: • 0: Instruction register (write) • 1: Data register (write and read)
RW	Selects read or write: • 0: Write • 1: Read
EN	Enable pulse; starts data read/write on falling edge.
ON	On signal 1: ON 0: OFF
BL	Backlight: • 1: ON • 0: OFF
DB0-DB7	8-bit bidirectional data bus (can operate in 4-bit mode, ignoring DB0-DB3).

Table 1: Pin Functions of the HD44780 LCD Controller[4]

3.4 Debouncing

When switching a, button or switch may momentarily bounce off the internal contact before making a stable connection. When sampling with a high speed, this will cause unwanted behaviour as multiple electric signals is sent to the software.
There are two multiple ways to solve this issue. One is hardware, implementing an RC circuit to the switch to discharge the bounce over time. The other way is in the software. By introducing a delay and comparing the last state of the switch and seeing if continues to stay in the same state after some time. Although the buttons on the DE1-SoC are debouncing buttons through hardware when sampling with a relative high frequency of 50 Mega Hertz problems may still occur. Therefore an easy falling edge detection on the button was used, although it not a debounce tactic at all. [5]

3.5 ASCII

American Standard Code for Information Interchange (ASCII) is a character encoding standard that allows communication of letters and symbols over a data bus. The ASCII language works by having a unique 8 bit code for each symbol. [6]

Character	ASCII Code (8-bit Binary)
0	00110000
1	00110001
2	00110010
3	00110011
4	00110100
5	00110101
6	00110110
7	00110111
8	00111000
9	00111001
:	00111010
␣	00100000

Table 2: 8-bit binary ASCII codes used in the project.

3.6 State Machine

A state machine is an abstraction used to design algorithms. Depending on inputs, the state machine transitions to a different state based on those inputs and the current state. A state can be described as the status of a system waiting for a certain input to trigger a transition. When the transition condition is fulfilled, the state changes accordingly.[7]

The state machine can be represented in a state diagram, where circles represent the possible states and arrows between states represent transitions. Each arrow is labelled with a transition condition that must be satisfied for the system to move from one state to another.

4 Implementation

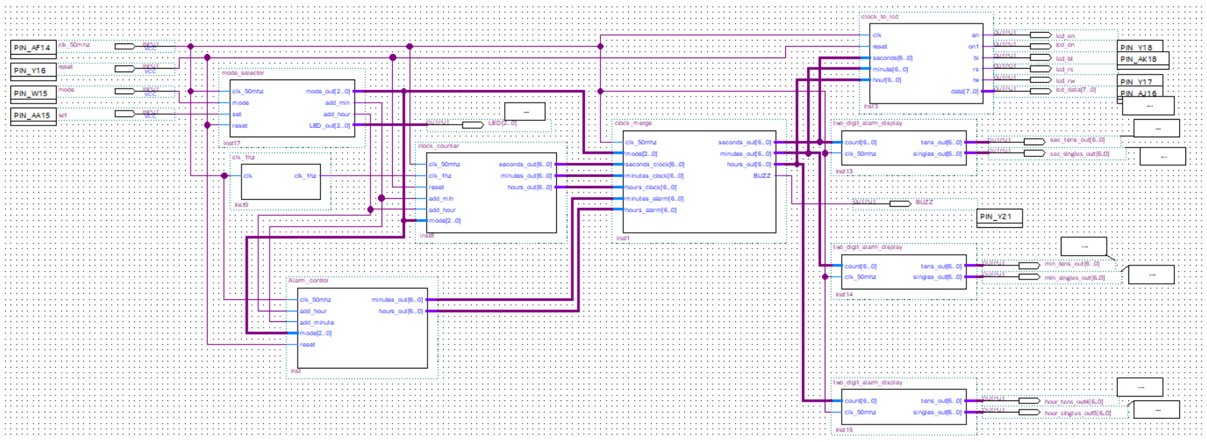


Figure 3: All units assembled in a Block Diagram/Schematic File

4.1 Clk 1Hz

The clk 1Hz component scales down the 50 Mega Hertz clock downto 1Hz. This is done by checking when a rising edge is detected at the 50 Mega Hertz clock and counting upwards. Each time the counting reaches 24 999 999 the output of the component is inverted and the counting is reset. This results in a 1Hz clock with a 50% duty cycle.

Inputs	Outputs
50 MHz clock	1 Hz clock

Table 3: IO for clk 1hz block

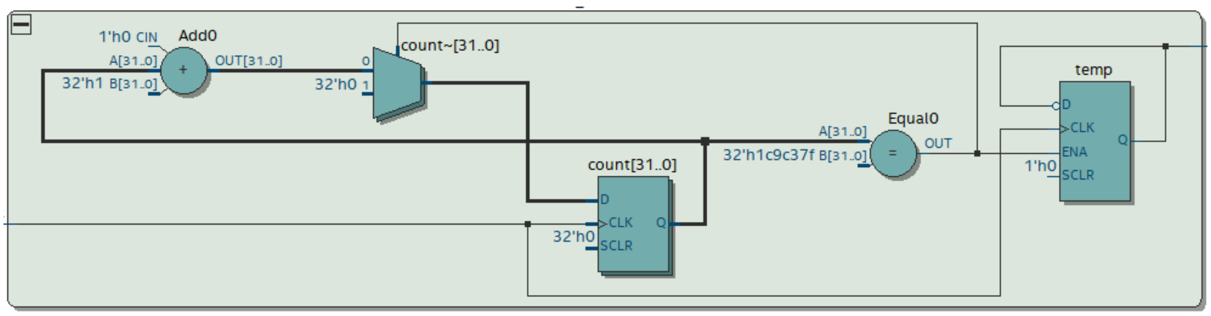


Figure 4: The RTL view of clk 1hz

4.2 Mode Selector

The `mode_selector` component handles user interaction and determines the system's current operating mode. It consists of two main processes:

- **Button Debouncing:** Ensures clean detection of button presses by monitoring current and previous values to detect falling edges.
- **Finite State Machine (FSM):** Implements the logic for mode transitions based on the current state and button inputs.

Interface Overview

Signal	Description
Inputs	
`clk_50mhz`	50 MHz clock signal for system synchronization
`reset`	Active-low reset signal
`mode`	Mode button input
`set`	Set button input
Outputs	
`mode_out`	3-bit signal indicating current mode to other blocks
`add_min`	Pulsed signal indicating minute increment (manual time/ alarm)
`add_hour`	Pulsed signal indicating hour increment (manual time/ alarm)
`LED_out`	Signal to indicate whether alarm is active

Table 4: I/O signals of the `mode_selector` module

State Machine Behavior

The FSM transitions between seven named states based on button inputs. Each state has an associated 3-bit `mode_out` code used to synchronize operation across all modules.

- **IDLE** (`"000"`)
  Default operating mode: current time is displayed and alarm is off.
  - `mode` → `SELECT_TIME_MIN`
  - `set` → `ALARM_ON`
- **SELECT_TIME_MIN** (`"100"`)
  Time-setting mode for minutes. Pressing `set` increments the minutes.
  - `mode` → `SELECT_TIME_HOUR`
- **SELECT_TIME_HOUR** (`"101"`)
  Time-setting mode for hours. Pressing `set` increments the hours.
  - `mode` → `IDLE`
- **ALARM_ON** (`"001"`)
  Alarm is active.
  - `set` → returns to `IDLE` (alarm off)
  - `mode` → `CHANGE_ALARM_MIN`
- **CHANGE_ALARM_MIN** (`"010"`)
  Alarm-setting mode for minutes. Pressing `set` increments the alarm minute value.
  - `mode` → `CHANGE_ALARM_HOUR`
- **CHANGE_ALARM_HOUR** (`"011"`)
  Alarm-setting mode for hours. Pressing `set` increments the alarm hour value.

– `mode` → `ALARM_ON`

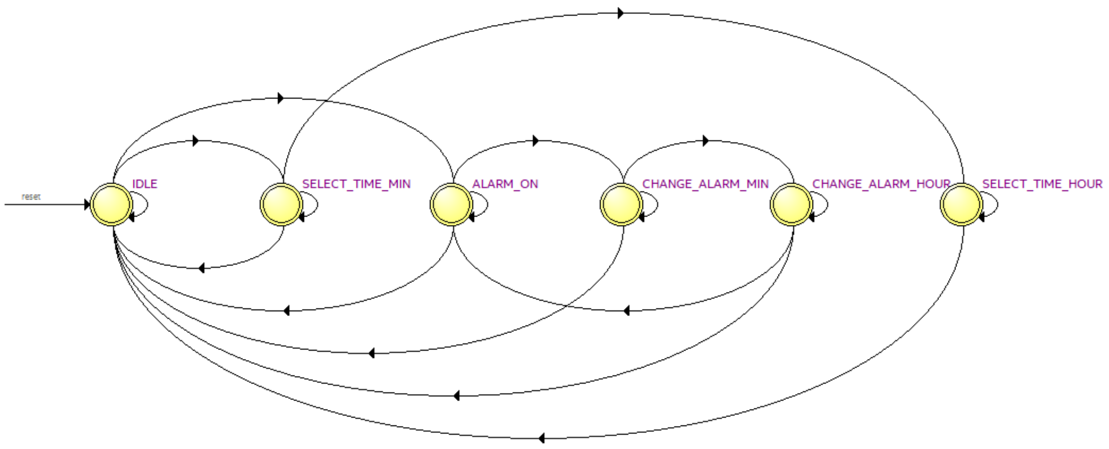


Figure 5: State diagram for the `Mode Selector` FSM

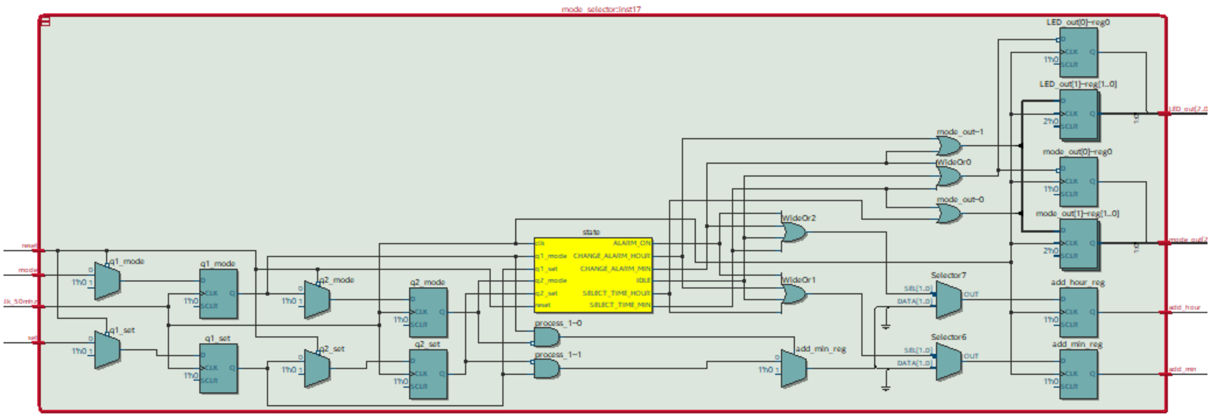


Figure 6: RTL schematic view of the `Mode Selector` component

4.3 Clock Counter

The `clock_counter` entity implements a digital clock with functionality to track and manually adjust hours, minutes, and seconds. It also handles the roll over of time as the time reaches the highest number of that digit, incrementing the next digit by one. The interface includes:

Signal	Description
Inputs	
`clk_50mhz`	50 MHz clock signal used for general synchronization
`clk_1hz`	1 Hz clock signal used to increment time once per second
`reset`	Active-low reset to initialize the time registers
`add_min`	Pulse signal to manually increment minutes
`add_hour`	Pulse signal to manually increment hours
`mode`	3-bit signal determining current operating mode
Outputs	
`seconds_out`	Current second value as a 7-bit binary signal
`minutes_out`	Current minute value as a 7-bit binary signal
`hours_out`	Current hour value as a 7-bit binary signal

Table 5: I/O signals of the `clock_counter` module

This block starts off by transitioning the 1 Hertz clock into a one tick pulse when a rising edge is detected. This is later used to increment the time one second per second. Then the reset button is handles as when pressed, all register and setting are to be reset. Next the mode determines what time operation is used.

- **Mode = (”100”)**
  When inside `SELECT_TIME_MIN`, when a rising edge of `add_min` is detected, the internal signal `min_change` is incremented by one.
- **Mode = (”101”)**
  When inside `SELECT_TIME_HOUR`, when a rising edge of `add_hour` is detected, the interan signal `hour_change` is incremented by one.
- **Else**
  Else the clock can increment one second per second without being interrupted.

At last internal signals are transitioned to the outgoing signals for correct display of time.

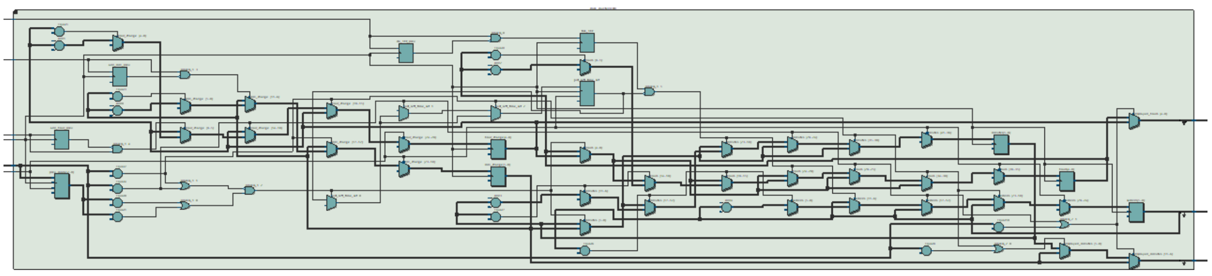


Figure 7: RTL schematic view of the `Clock Counter` component

4.4 Alarm Control

The Alarm Control handles the alarm by storing and transferring the saved alarm.

Signal	Description
Inputs	
`clk_50mhz`	50 MHz clock signal used to synchronize operations
`add_hour`	Pulse signal used to increment the alarm hour (mode dependent)
`add_minute`	Pulse signal used to increment the alarm minute (mode dependent)
`mode`	3-bit signal determining current mode of operation
`reset`	Active-low signal to reset alarm time registers
Outputs	
`minutes_out`	Current alarm minute value as a 7-bit binary signal
`hours_out`	Current alarm hour value as a 7-bit binary signal

Table 6: I/O signals of the `Alarm_control` module

This block starts off by handling the reset button as when pressed the alarm is reset to 00:00:00. Then depending on what is the current mode the following is executed.

- **Mode = ("010")**
  When inside `CHANGE_ALARM_MIN`, as a rising edge of `add_minute` is detected the alarm minute time is incremented by one.

- **Mode = ("101")**
  When inside `CHANGE_ALARM_HOUR`, as a rising edge of `add_hour` is detected the alarm hour time is incremented by one.

At the end internal signals are transferred as the right data type to outgoing signals

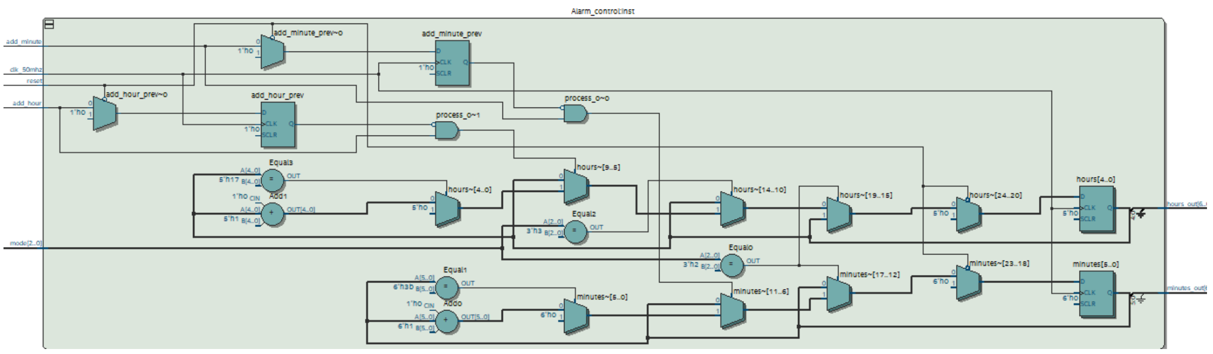


Figure 8: RTL schematic view of the `Alarm Control` component

4.5 Clock Merge

This block handles all signals and depending on the mode, letting the right signal go through as well as it enables the half a second blinking when changing the time.

Signal	Description
Inputs	
`clk_50mhz`	50 MHz clock signal used for synchronization and blinking logic
`mode`	3-bit signal determining the current operating or editing mode
`seconds_clock`	Current second value from the clock logic
`minutes_clock`	Current minute value from the clock logic
`hours_clock`	Current hour value from the clock logic
`minutes_alarm`	Alarm minute value to be merged into display output
`hours_alarm`	Alarm hour value to be merged into display output
Outputs	
`seconds_out`	Final seconds value to be shown on the display
`minutes_out`	Final minutes value to be shown on the display (with blinking in edit modes)
`hours_out`	Final hours value to be shown on the display (with blinking in edit modes)
`BUZZ`	Signal activated when current time matches the alarm time

Table 7: I/O signals of the `clock_merge` module

The a case statement is used to link the incoming mode signal to the right case. Blinking works by sending a higher number then 59 to the display blocks. This is interpreted as a blink state of the number is off and showing a blank space.

By default is the regular time is always let trough until overwritten by current mode.

- **Mode = (”010”)**
  When `CHANGE_ALARM_MIN` is active the alarm time are let trough and blinking is enabled on the minute digits.
- **Mode = (”011”)**
  When `CHANGE_ALARM_HOUR` is active the alarm time is let trough and blinking is enabled on the hour digits.
- **Mode = (”100”)**
  When `SELECT_TIME_MIN` is active the current time is let trough and blinking is enabled on the minute digits.
- **Mode = (”101”)**
  When `SELECT_TIME_hour` is active the current time is let trough and blinking is enabled on the hour digits.
- **Mode = (”001”)**
  When `ALARM_ON` is active the current time is let trough and when the alarm time and current time align the blink is instead directed to the `BUZZ` signal.
- **Mode = (”000”)**
  When `IDLE`, the `BUZZ` is forced low.

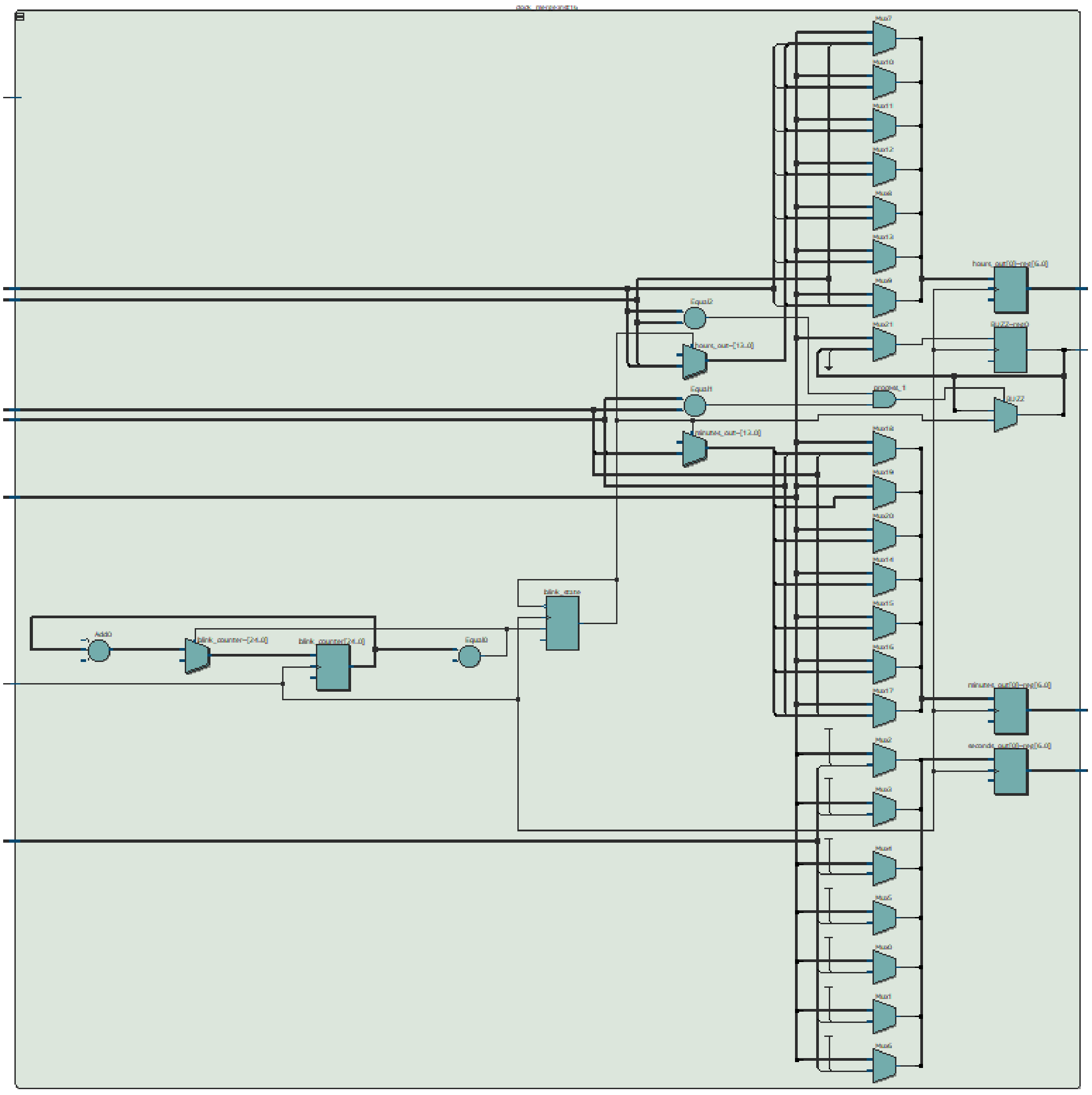


Figure 9: RTL schematic view of the `Clock Merge` component

4.6 Clock to LCD

The `clock_to_lcd` entity serves as an interface between the internal clock logic and a character-based LCD display. Its primary function is to format and present the current time in `hh:mm:ss` format on the LCD. The component handles ASCII conversion, checking inputs, and sequential data transmission in accordance with the LCD's timing protocol.

Signal	Description
Inputs	
`clk_50mhz`	System clock signal for timing and state machine control
`reset`	Active-low reset signal for reinitializing the state machine
`seconds`	7-bit input representing the seconds value in BCD or binary format
`minute`	7-bit input representing the minutes value in BCD or binary format
`hour`	7-bit input representing the hours value in BCD or binary format
Outputs	
`en`	Enable signal for the LCD interface (pulsed periodically)
`on1`	Turns on the LCD (always high)
`bl`	Controls the LCD backlight (always high)
`rs`	Register select (0 for command, 1 for data)
`rw`	Read/Write control signal (always 0, write mode)
`data`	8-bit data bus used for commands and ASCII characters

Table 8: I/O signals of the `clock_to_lcd` module

Internally, the component performs the following steps:

- Converts each incoming vector of the hour, minute, and second values to its corresponding 4-bit representation, separating to 2 digits.
- Each 4-bit digit is mapped to an ASCII character via the internal `bin_to_ascii` function, which is defined using a `case` statement.
- Sanity checks ensure time values are within valid bounds; if not, blank characters are displayed instead.
- A state machine controls the LCD initialization and the display of the time. It cycles through 12 states:
  - Four **setup states** configure the LCD in 8-bit mode, clear the display, turn on the display, and return the cursor home.
  - Eight **data states** write the hour, colon, minute, colon, and second values to the LCD.
- A clock counter ensures appropriate delays between commands by toggling the `en` signal at fixed intervals.

Special care is taken in the timing mechanism. The `clk_count` variable increments until it matches the predefined `clk_goal` value. Two intermediate thresholds, `enable_clk_goal1` and `enable_clk_goal2`, control the pulsing of the `en` signal required by the LCD protocol.

4.7 Two-Digit Alarm Display

The `two_digit_alarm_display` entity is responsible for converting a 7-bit binary input value into a two-digit decimal representation suitable for display on 7-segment displays. It outputs the segment codes for the tens and singles digits. The component is designed to handle values in the range 0–59, and shows blank digits for out-of-range values. The I/O ports are detailed below:

Signal	**Description**
Inputs	
`clk_50mhz`	50 MHz clock signal used to synchronize the logic
`count`	7-bit binary input representing a decimal value (expected 0–59)
Outputs	
`tens_out`	7-bit segment code for the tens digit
`singles_out`	7-bit segment code for the singles digit

Table 9: I/O signals of the `two_digit_alarm_display` module

Internally, the design works as follows:

- The input value `count` is converted from a 7-bit unsigned binary to an integer.
- If the value exceeds 59 (i.e., greater than `‘‘111011"`), both digits are set to blank by assigning the output `‘‘1111111"`.
- Otherwise, the value is split into tens and singles digits using integer division and modulo operations.
- Both digits are converted to 7-segment codes using two separate `case` statements. Each digit is mapped to its corresponding active-low 7-segment representation:
  - `0` → `1000000`
  - `1` → `1111001`
  - ...
  - `9` → `0010000`
  - Invalid values default to blank (`1111111`)

This component provides a straightforward method to visualize numeric alarm values such as hours, minutes, or user-set thresholds, and can be reused in different display configurations.

4.8 Complete Design

The complete design integrates all the individual functional blocks into a functional alarm clock system. The blocks are interconnected using internal signals and synchronized through a shared clock domain. The top-level design is structured in a Block Diagram File (BDF), enabling clear visual placement and wiring of all components.

Signal	Description
Global Inputs	
`clk_50mhz`	50 MHz system clock input
`reset`	Asynchronous reset signal
`mode`	Mode selection button
`set`	Set button for hour/minute adjustments
Global Outputs	
`LED_out[2..0], BUZZ`	LED indicators (3 for mode, 1 for alarm status)
`seg_display`	7-bit outputs to six 7-segment displays
`lcd_data`	8-bit LCD data bus
`lcd_rs, lcd_rw, lcd_en`	LCD control signals
`lcd_on, lcd_bl`	LCD power and backlight control

Table 10: Top-level input/output signals for the full alarm clock system

Signal Flow

The signal flow begins with the 50 MHz clock entering the `clk_1hz` module, which generates a stable 1 Hz pulse used across timing-related modules. Both the 50 MHz and 1 Hz clocks are routed to the `clock_counter`, which keeps track of the current time. The `mode_selector` interprets button inputs and defines system mode, generating control signals that influence whether time is incremented, alarm time is updated, or alarm mode is toggled.

The `alarm_control` module stores and compares the current time with the alarm time, triggering an active alarm output when matched. The `clock_merge` unit decides whether to forward the system time or the alarm time depending on the selected mode, effectively determining what is displayed.

Two display subsystems visualize the output:

- `clock_to_lcd` handles ASCII conversion and initialization sequences for the HD44780 LCD, displaying time in human-readable format.
- `two_digit_alarm_display` converts the binary time data into BCD and segment control signals for the six 7-segment displays on the DE1-SoC board.

Reset and Alarm Behavior

The reset signal asynchronously initializes all modules to their default state, clearing any previous time, alarm configuration, and display data. When the alarm goes off, the corresponding LED illuminates. The user may either cycle to a different mode or press the set button again to disable it, transitioning back to idle.

Synchronization and Timing

All time-sensitive modules (e.g., time counting, alarm matching, display updates) are synchronized through the 1 Hz clock. Control modules like the mode selector operate at the higher 50 MHz resolution to accurately detect user button inputs.

Summary of Operation

In summary, the complete design behaves as follows:

1. Displays the current time on both LCD and 7-segment displays.
2. Allows user to change time and alarm settings via mode and set buttons.
3. Activates alarm output at the pre-set time and provides clear LED indication.
4. Supports full system reset for testing or daily reuse.

All modules are modular and reusable, making the design extensible for further enhancements. So by extending the amount of modes functionalities such as snooze functionality or 24/12-hour could be implemented.

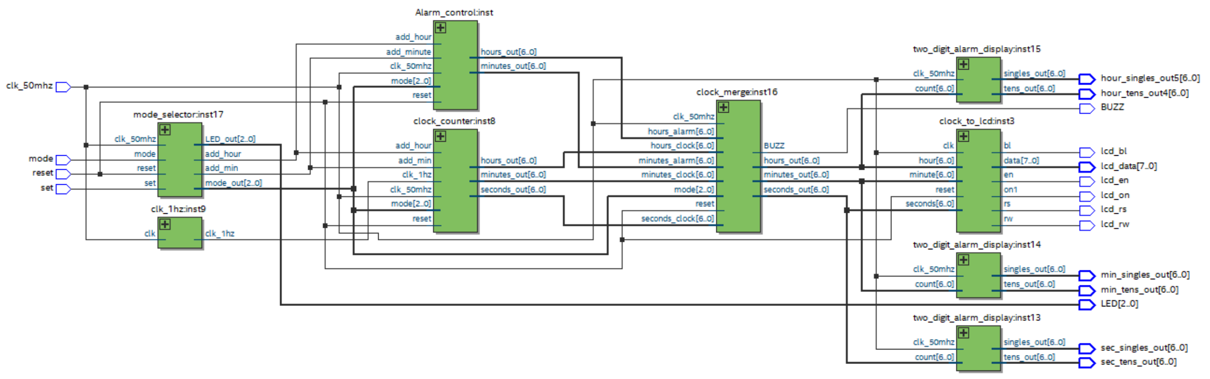


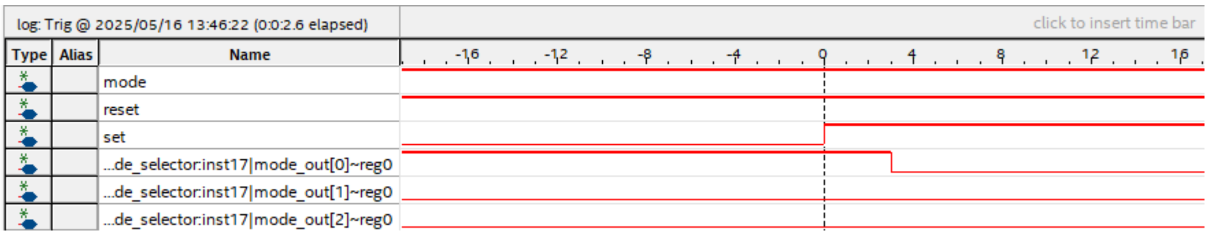
Figure 10: The complete design as shown in the RTL view.

5 Tests & Results

Some test where implemented to confirm some functions of the software. Signaltap was used for test of certain functions and modelsim was used to simulate the Mode Selector block. Both was done after the whole design was finished.

5.1 Signaltap

First of the change of modes from "001" to "000" is tested. As seen in figure 11 shortly after the set button is pressed the mode changes to "000".


log: Trig @ 2025/05/16 13:46:22 (0:0:2.6 elapsed)

Type	Alias	Name
		mode
		reset
		set
		...de_selector:inst17|mode_out[0]~reg0
		...de_selector:inst17|mode_out[1]~reg0
		...de_selector:inst17|mode_out[2]~reg0

Figure 11: Change of mode: ("001") → ("000") confirmation.

In figure 12 the change of mode is seen as mode button is pressed changing mode from "000" regular time to "100" `SELECT_TIME_MIN`.

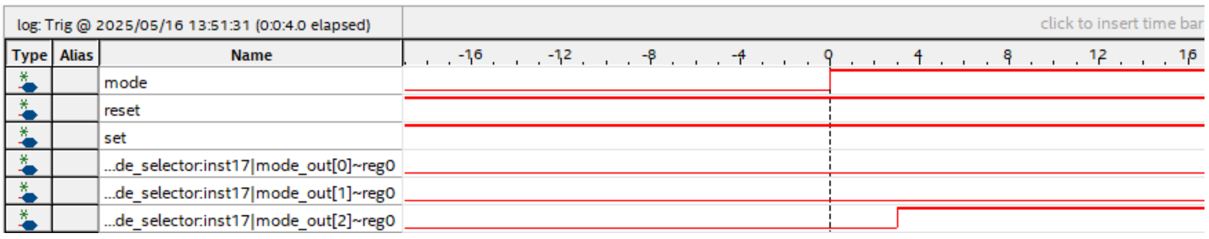

log: Trig @ 2025/05/16 13:51:31 (0:0:4.0 elapsed)
click to insert time bar
Type
Alias
Name
-16
-12
-8
-4
0
4
8
12
16
mode
reset
set
...de_selector:inst17|mode_out[0]~reg0
...de_selector:inst17|mode_out[1]~reg0
...de_selector:inst17|mode_out[2]~reg0

Figure 12: Change of mode: ("000") → ("100")

The selection of times is tested in figure 13. As seen when in the appropriate mode, as pressing the set button, the time is increased by one on `minutes_out`. However, as seen in figure 14 the same result should be accomplished but a binary 127 is sent out on `hour_out` instead due to the blinking mechanic.

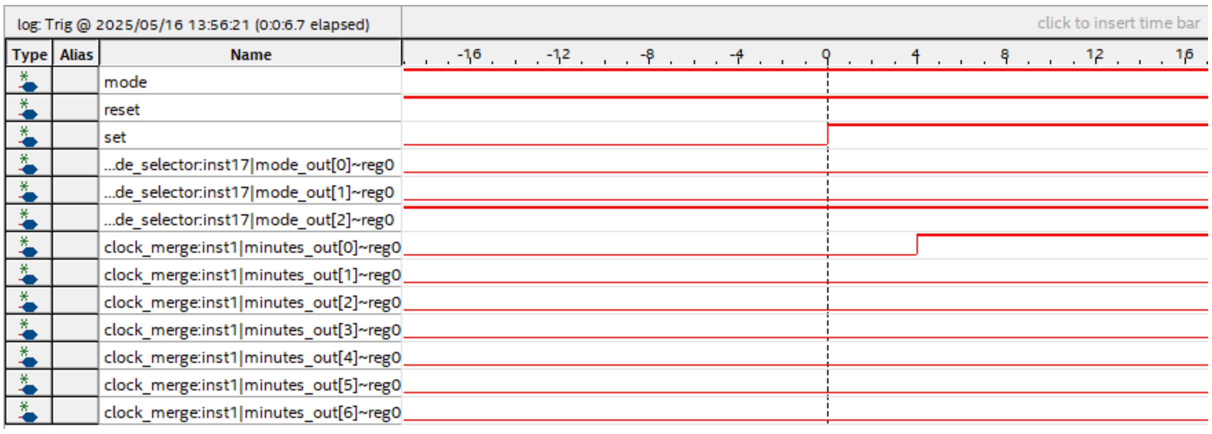

log: Trig @ 2025/05/16 13:56:21 (0:0:6.7 elapsed)
click to insert time bar
Type
Alias
Name
-16
-12
-8
-4
0
4
8
12
16
mode
reset
set
...de_selector:inst17|mode_out[0]~reg0
...de_selector:inst17|mode_out[1]~reg0
...de_selector:inst17|mode_out[2]~reg0
clock_merge:inst1|minutes_out[0]~reg0
clock_merge:inst1|minutes_out[1]~reg0
clock_merge:inst1|minutes_out[2]~reg0
clock_merge:inst1|minutes_out[3]~reg0
clock_merge:inst1|minutes_out[4]~reg0
clock_merge:inst1|minutes_out[5]~reg0
clock_merge:inst1|minutes_out[6]~reg0

Figure 13: Time incremented by one in mode "100"

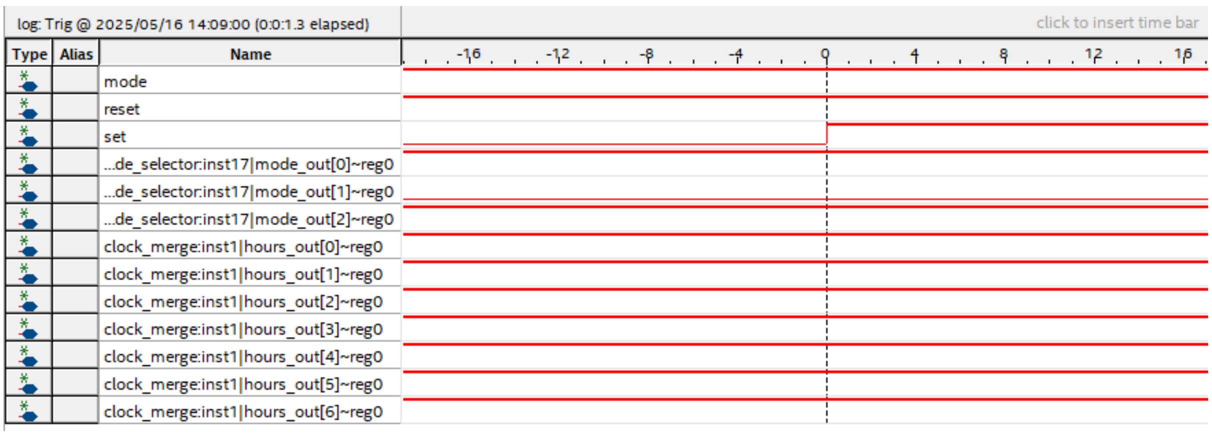

log: Trig @ 2025/05/16 14:09:00 (0:0:1.3 elapsed)
click to insert time bar
Type
Alias
Name
-16
-12
-8
-4
0
4
8
12
16
mode
reset
set
...de_selector:inst17|mode_out[0]~reg0
...de_selector:inst17|mode_out[1]~reg0
...de_selector:inst17|mode_out[2]~reg0
clock_merge:inst1|hours_out[0]~reg0
clock_merge:inst1|hours_out[1]~reg0
clock_merge:inst1|hours_out[2]~reg0
clock_merge:inst1|hours_out[3]~reg0
clock_merge:inst1|hours_out[4]~reg0
clock_merge:inst1|hours_out[5]~reg0
clock_merge:inst1|hours_out[6]~reg0

Figure 14: Time incremented by one in mode "101"

5.2 Simulations

Simulation of Clock Merge was executed to see if the block works like intended as it is all signal go trough this block.

The result of an execution is as the mode changes so does the output. When in mode `CHANGE_ALARM_MIN` or ”010” the seconds are turned off, the minutes are changed to `minutes_alarm` and hours are changed to `hours_alarm`.

The same goes for when entering `SELECT_TIME_MIN`, the outgoing times is changed back to `hours_out`, `minutes_out`, and `seconds_out`.

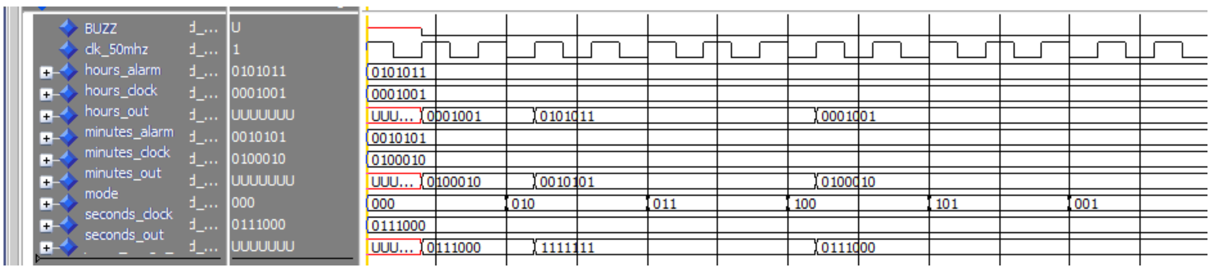


Figure 15

6 Conclusions and Evaluation

This project successfully demonstrates the implementation of a fully functional alarm clock system on the DE1-SoC development board using VHDL. All intended features as timekeeping, alarm setting, and visual display of time. Build up by seven different components the design is quite easy to understand and can be scaled with more components for extra features.

Beyond functionality, this project served as a valuable learning experience in VHDL and Quartus Prime. It deepened the understanding of synchronous logic design, state machines, timing considerations, and practical interfacing with an external display such as the HD44780 LCD. Although working through the syntax of VHDL and solving bugs, the greatest challenge was to get familiar with the development tool. A majority of time was spent on getting simulation tools and the programmer to cooperate.

The final implementation does not only cover the bear minimum of requirements but also some system feedback as the mode indication of current mode, alarm on/off, and a second display of time. However it should be said that the 7 segment displays were not removed just to ensure to not ruin the project but it was a nice to have when developing the system.

Overall, the project highlights the strengths of hardware design through VHDL and demonstrates the importance of a though out base for development, simulation, and testing in digital system design. If the system had been more complex and difficult to keep track of, the simulation would have been more important during development.

References

[1] Intel Corporation, "DE1-SoC Development and Education Board," 2024. Accessed: Jun. 10, 2024. Available at: `https://www.intel.com/content/www/us/en/partner/showcase/offering/a5b3b0000004cbaAAA/de1soc-board.html`.

[2] Wikipedia contributors, "Vhdl — wikipedia, the free encyclopedia," 2025. Accessed: 2025-05-15. Available at: `https://en.wikipedia.org/wiki/VHDL`.

[3] Wikipedia contributors, "Hitachi HD44780 LCD controller — Wikipedia, the free encyclopedia," 2024. Accessed: 2025-05-15. Available at: `https://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller#Mode_selection`.

[4] Hitachi, Ltd., *HD44780U (LCD-II) Dot Matrix Liquid Crystal Display Controller/-Driver*. Hitachi, Ltd., Aug 1999. Document Revision: 1.0. Available at: `https://cdn.sparkfun.com/assets/9/5/f/7/b/HD44780.pdf`.

[5] Pico Technology, "What is switch bounce and how to implement debounce," 2023. Accessed: 2025-05-14. Available at: `https://www.picotech.com/library/articles/blog/what-is-switch-bounce-how-to-implement-debounce`.

[6] Wikipedia contributors, "Ascii — wikipedia, the free encyclopedia," 2024. Accessed: 2025-05-14. Available at: `https://en.wikipedia.org/wiki/ASCII`.

[7] MDN contributors, "State machine - mdn web docs," 2024. Accessed: 2025-05-14. Available at: `https://developer.mozilla.org/en-US/docs/Glossary/State_machine`.

Appendix A. Source Code

Connections between all components can be seen in the block diagram in figure 3.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use ieee.numeric_std.all;
entity clk_1hz is

  port (
    clk : in std_logic ;
    clk_1hz : out std_logic
    );

end clk_1hz;

architecture Behavioral of clk_1hz is

signal temp : std_logic := '0';
signal count : integer := 0;

begin
  process(clk)
    begin
      if rising_edge(clk) then
        count <= count + 1;
        if (count = 24_999_999) then
          temp <= not temp;
          count <= 0;
        end if;
      end if;
      clk_1hz <= temp;
  end process;

end Behavioral;
```

Listing 1: Clk 1Hz

```vhdl
Library ieee;
Use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mode_selector is
    port (
        clk_50mhz : in std_logic;
        reset : in std_logic;
        mode : in std_logic;
        set : in std_logic;
        mode_out : out std_logic_vector(2 downto 0);
        add_min : out std_logic;
        add_hour : out std_logic;
        LED_out : out std_logic_vector(2 downto 0)
    );
end mode_selector;

architecture Behavioral of mode_selector is
    type state_type is (IDLE, SELECT_TIME_MIN, SELECT_TIME_HOUR,
       ALARM_ON, CHANGE_ALARM_MIN, CHANGE_ALARM_HOUR);
    signal state : state_type := IDLE;
    signal q1_mode, q2_mode, q1_set, q2_set : std_logic := '0';
    signal add_min_reg, add_hour_reg : std_logic := '0';
begin
    -- Button debouncing and edge detection
    process(clk_50mhz)
    begin
        if rising_edge(clk_50mhz) then
            if reset = '0' then
                q1_mode <= '0';
                q2_mode <= '0';
                q1_set <= '0';
                q2_set <= '0';

            else
                -- Simple debounce mechanism
                if (q1_set /= q2_set) or (q1_mode /= q2_mode)
                   then
                    q2_mode <= q1_mode;
                    q1_mode <= mode;
                    q2_set <= q1_set;
                    q1_set <= set;
                else
                    q2_mode <= q1_mode;
                    q1_mode <= mode;
                    q2_set <= q1_set;
                    q1_set <= set;
                end if;
            end if;
        end if;
    end process;

    -- State machine process
    process(clk_50mhz)
    begin
        if rising_edge(clk_50mhz) then
                -- Clear the pulse signals after one clock cycle
                add_min_reg <= '0';
                add_hour_reg <= '0';
           if reset = '0' then
             state <= IDLE;
           end if;
```

```

                case state is
                    when IDLE =>
                        LED_out <= "000";
                        mode_out <= "000";
                        if (q1_set = '1' and q2_set = '0') then
                            state <= ALARM_ON;
                        elsif (q1_mode = '1' and q2_mode = '0')
                           then
                            state <= SELECT_TIME_MIN;
                        end if;

                    when SELECT_TIME_MIN =>
                        LED_out <= "100";
                        mode_out <= "100";
                        if (q1_mode = '1' and q2_mode = '0') then
                            state <= SELECT_TIME_HOUR;
                        elsif (q1_set = '1' and q2_set = '0')
                           then
                            add_min_reg <= '1';
                        end if;

                    when SELECT_TIME_HOUR =>
                        LED_out <= "101";
                        mode_out <= "101";
                        if (q1_mode = '1' and q2_mode = '0') then
                            state <= IDLE;
                        elsif (q1_set = '1' and q2_set = '0')
                           then
                            add_hour_reg <= '1';
                        end if;

                    when ALARM_ON =>
                        LED_out <= "001";
                        mode_out <= "001";
                        if (q1_set = '1' and q2_set = '0') then
                            state <= IDLE;
                        elsif (q1_mode = '1' and q2_mode = '0')
                           then
                            state <= CHANGE_ALARM_MIN;
                        end if;

                    when CHANGE_ALARM_MIN =>
                        LED_out <= "010";
                        mode_out <= "010";
                        if (q1_mode = '1' and q2_mode = '0') then
                            state <= CHANGE_ALARM_HOUR;
                        elsif (q1_set = '1' and q2_set = '0')
                           then
                            add_min_reg <= '1';
                        end if;

                    when CHANGE_ALARM_HOUR =>
                        LED_out <= "011";
                        mode_out <= "011";
                        if (q1_mode = '1' and q2_mode = '0') then
                            state <= ALARM_ON;
                        elsif (q1_set = '1' and q2_set = '0')
                           then
                            add_hour_reg <= '1';
                        end if;
                end case;
        end if;
    end process;

    -- Assign output signals
    add_min <= add_min_reg;
    add_hour <= add_hour_reg;
end Behavioral;
```

Listing 2: Mode Selector

```
Library ieee;
Use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Alarm_control is
  port (
    clk_50mhz : in std_logic;
    add_hour : in std_logic;
    add_minute : in std_logic;
    mode : in std_logic_vector(2 downto 0);
    reset : in std_logic;

    minutes_out : out std_logic_vector(6 downto 0);
    hours_out : out std_logic_vector(6 downto 0)
      );
end Alarm_control;


architecture Behavioral of Alarm_control is
-- Internal signals
signal minutes : integer range 0 to 59;
signal hours : integer range 0 to 23;
signal add_hour_prev : std_logic := '0';
signal add_minute_prev : std_logic := '0';

begin
  process(clk_50mhz)
  begin
    if rising_edge(clk_50mhz) then
      if reset = '0' then  -- Active low reset
            minutes <= 0;
            hours <= 0;
            add_hour_prev <= '0';
            add_minute_prev <= '0';
        else
      -- Edge detection for button presses
      add_hour_prev <= add_hour;
      add_minute_prev <= add_minute;

      -- Update alarm time when in alarm setting modes (010 for
         minutes, 011 for hours)
        if mode = "010" then  -- CHANGE_ALARM_MIN mode
          if add_minute = '1' and add_minute_prev = '0' then
            if minutes = 59 then
              minutes <= 0;
            else
              minutes <= minutes + 1;
            end if;
          end if;
        elsif mode = "011" then  -- CHANGE_ALARM_HOUR mode
          if add_hour = '1' and add_hour_prev = '0' then
            if hours = 23 then
              hours <= 0;
            else
              hours <= hours + 1;
            end if;
          end if;
        end if;
      end if;
    end if;
  end process;

  -- Convert integers to std_logic_vector for output
  minutes_out <= std_logic_vector(to_unsigned(minutes, 7));
  hours_out <= std_logic_vector(to_unsigned(hours, 7));

end Behavioral;
```

Listing 3: Alarm Control

```
Library ieee;
Use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity clock_merge is
  port (
    clk_50mhz : in std_logic;
    mode : in std_logic_vector(2 downto 0);
    seconds_clock : in std_logic_vector(6 downto 0);
    minutes_clock : in std_logic_vector(6 downto 0);
    hours_clock : in std_logic_vector(6 downto 0);
    minutes_alarm : in std_logic_vector(6 downto 0);
    hours_alarm : in std_logic_vector(6 downto 0);

    seconds_out : out std_logic_vector(6 downto 0);
    minutes_out : out std_logic_vector(6 downto 0);
    hours_out : out std_logic_vector(6 downto 0);
    BUZZ : out std_logic
      );
end clock_merge;


architecture Behavioral of clock_merge is
-- Blinking signal for editing modes
signal blink_counter : integer range 0 to 25000000 := 0;
signal blink_state : std_logic := '1';

begin
  -- Blinking process for editing indicators
  process(clk_50mhz)
  begin
    if rising_edge(clk_50mhz) then
      if blink_counter = 25000000 then  -- 0.5 second blink rate
        blink_state <= not blink_state;
        blink_counter <= 0;
      else
        blink_counter <= blink_counter + 1;
      end if;
    end if;
  end process;

  -- Multiplexing process based on mode
  process(clk_50mhz)
  begin
    if rising_edge(clk_50mhz) then
      -- Always forward the clock values directly in all modes
      -- This ensures that the time values set in SELECT_TIME modes
      -- are visible as soon as the mode changes back to IDLE
      seconds_out <= seconds_clock;
      minutes_out <= minutes_clock;
      hours_out <= hours_clock;

      -- Apply blinking effects for edit modes only
      case mode is
        when "010" => -- CHANGE_ALARM_MIN: Show alarm time with minutes blinking
          seconds_out <= (others => '1');  -- Turn off seconds display during alarm setting
          hours_out <= hours_alarm;

          -- Blink minutes when setting them
          if blink_state = '1' then
            minutes_out <= minutes_alarm;
          else
            minutes_out <= (others => '1');  -- Blank display during blink off
          end if;

        when "011" => -- CHANGE_ALARM_HOUR: Show alarm time with hours blinking
          seconds_out <= (others => '1');  -- Turn off seconds display during alarm setting
          minutes_out <= minutes_alarm;

          -- Blink hours when setting them
          if blink_state = '1' then
            hours_out <= hours_alarm;
          else
            hours_out <= (others => '1');  -- Blank display during blink off
          end if;

        when "100" => -- SELECT_TIME_MIN: Set clock minutes (blinking)
          -- Blink minutes when setting them
          if blink_state = '1' then
            minutes_out <= minutes_clock;
          else
            minutes_out <= (others => '1');  -- Blank display during blink off
          end if;

        when "101" => -- SELECT_TIME_HOUR: Set clock hours (blinking)
          -- Blink hours when setting them
          if blink_state = '1' then
            hours_out <= hours_clock;
          else
            hours_out <= (others => '1');  -- Blank display during blink off
          end if;

        when "001" =>
          if (minutes_clock = minutes_alarm and hours_clock = hours_alarm) then
            if blink_state = '1' then
              BUZZ <= '1';
            else
              BUZZ <= '0';
            end if;
          end if;

        when "000" =>
          BUZZ <= '0';
        when others =>
      end case;
    end if;
  end process;
end Behavioral;
```

Listing 4: Clock Merge

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity clock_to_lcd is
port(

      clk : in std_logic;
      reset : in std_logic;
      seconds : in std_logic_vector(6 downto 0);
      minute : in std_logic_vector(6 downto 0);
      hour : in std_logic_vector(6 downto 0);

      en : out std_logic;
      on1 : out std_logic;
      bl : out std_logic;
      rs : out std_logic;
      rw : out std_logic;
      data : out std_logic_vector(7 downto 0)

);
end entity;

architecture behavior of clock_to_lcd is

signal second1 : std_logic_vector(3 downto 0);
signal second2 : std_logic_vector(3 downto 0);
signal minute1 : std_logic_vector(3 downto 0);
signal minute2 : std_logic_vector(3 downto 0);
signal hour1 : std_logic_vector(3 downto 0);
signal hour2 : std_logic_vector(3 downto 0);

signal seconds_int : integer range 0 to 127;
signal minute_int : integer range 0 to 127;
signal hour_int : integer range 0 to 127;

constant clk_goal : integer := 820000;
constant enable_clk_goal1: integer :=  200000;
constant enable_clk_goal2: integer :=  600000;
signal clk_count : integer := 0;


function bin_to_ascii (bin: std_logic_vector(3 downto 0)) return
```

```
   std_logic_vector is
    begin
        case bin is
            when "0000" => return "00110000";
            when "0001" => return "00110001";
            when "0010" => return "00110010";
            when "0011" => return "00110011";
            when "0100" => return "00110100";
            when "0101" => return "00110101";
            when "0110" => return "00110110";
            when "0111" => return "00110111";
            when "1000" => return "00111000";
            when "1001" => return "00111001";
        when "1010" => return "00100000";
            when others => return "00111111";
        end case;
    end function;

type state_type is (setup1,setup2,setup3,setup4,enter_data1,
   enter_data2,enter_data3,enter_data4,enter_data5,enter_data6,
   enter_data7,enter_data8);
signal state : state_type;

begin

-- The backlight and on signals are obviously always on
    bl <= '1';
    on1 <= '1';


   process(clk,reset)
    begin
    -- Reset logic
    if reset = '0' then
    clk_count <= 0;
    rs <= '0';
      rw <= '0';
      en <= '0';
    state <= setup1;
    data <= "00000000";
    elsif rising_edge(clk) then

      seconds_int <= to_integer(unsigned(seconds));
      minute_int <= to_integer(unsigned(minute));
      hour_int <= to_integer(unsigned(hour));


      if seconds_int > 59 then
         second1 <= "1010"; -- Space
         second2 <= "1010";
      else
```

```
           second1 <= std_logic_vector(to_unsigned(seconds_int / 10, 4));
           second2 <= std_logic_vector(to_unsigned(seconds_int mod 10, 4));
        end if;

        if minute_int > 59 then
           minute1 <= "1010"; -- Space
           minute2 <= "1010";
        else
           minute1 <= std_logic_vector(to_unsigned(minute_int / 10, 4));
           minute2 <= std_logic_vector(to_unsigned(minute_int mod 10, 4));
        end if;

        if hour_int > 23 then
           hour1 <= "1010"; -- Space
           hour2 <= "1010";
        else
           hour1 <= std_logic_vector(to_unsigned(hour_int / 10, 4));
           hour2 <= std_logic_vector(to_unsigned(hour_int mod 10, 4));
        end if;


        if seconds > "111011" then
          second1 <= "1010";
          second2 <= "1010";
        elsif minute > "111011" then
          minute1 <= "1010";
          minute2 <= "1010";
        elsif hour > "11000" then
          hour1 <= "1010";
          hour2 <= "1010";
        end if;

        if clk_count = enable_clk_goal1 then
        en <= '1';
        clk_count <= clk_count + 1;

        elsif clk_count = enable_clk_goal2 then
        en <= '0';
        clk_count <= clk_count + 1;


        elsif clk_count = clk_goal - 1 then
        clk_count <= 0;
          case state is

```

```
            when setup1 => --8-bit mode
              rs <= '0';
              rw <= '0';
              data <= "00111000";
              state <= setup2;

            when setup2 => -- Clear display
              rs <= '0';
              rw <= '0';
              data <= "00000001";
              state <= setup3;

            when setup3 => -- Display on (no cursor)
              rs <= '0';
              rw <= '0';
              data <= "00001100";
              state <= setup4;

            when setup4 =>  -- Return home
              rs <= '0';
              rw <= '0';
              data <= "00000010";
              state <= enter_data1;

            when enter_data1 =>
              rs <= '1';
              rw <= '0';
              data <= bin_to_ascii(hour1);
              state <= enter_data2;

            when enter_data2 =>
              rs <= '1';
              rw <= '0';
              data <= bin_to_ascii(hour2);
              state <= enter_data3;

            when enter_data3 =>
              rs <= '1';
              rw <= '0';
              data <= "00111010";
              state <= enter_data4;

            when enter_data4 =>
              rs <= '1';
              rw <= '0';
              data <= bin_to_ascii(minute1);
              state <= enter_data5;

            when enter_data5 =>
              rs <= '1';
              rw <= '0';
              data <= bin_to_ascii(minute2);
              state <= enter_data6;

            when enter_data6 =>
              rs <= '1';
              rw <= '0';
              data <= "00111010";
              state <= enter_data7;

            when enter_data7 =>
              rs <= '1';
              rw <= '0';
              data <= bin_to_ascii(second1);
              state <= enter_data8;

            when enter_data8 =>
              rs <= '1';
              rw <= '0';
              data <= bin_to_ascii(second2);
              state <= setup4;

          end case;
        else
          clk_count <= clk_count + 1;
        end if;
      end if;
    end process;

  end architecture;
```

Listing 5: Clock to LCD

```
Library ieee;
Use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity two_digit_alarm_display is
  port (
    count : in std_logic_vector(6 downto 0);
    clk_50mhz : in std_logic;
    tens_out : out std_logic_vector(6 downto 0);
    singles_out : out std_logic_vector(6 downto 0)
      );
end two_digit_alarm_display;

architecture rtl of two_digit_alarm_display is
  signal tens, singles, number : integer range 0 to 64;
  begin


    process(clk_50mhz)
      begin
      if count > "111011" then
        singles_out <= "1111111";
        tens_out <= "1111111";
      else
        number <= to_integer(unsigned(count));

        tens <= number / 10;
        singles <= number mod 10;

        case singles is
          when 0 => singles_out <= "1000000"; -- 0
          when 1 => singles_out <= "1111001"; -- 1
          when 2 => singles_out <= "0100100"; -- 2
          when 3 => singles_out <= "0110000"; -- 3
          when 4 => singles_out <= "0011001"; -- 4
          when 5 => singles_out <= "0010010"; -- 5
          when 6 => singles_out <= "0000010"; -- 6
          when 7 => singles_out <= "1111000"; -- 7
          when 8 => singles_out <= "0000000"; -- 8
          when 9 => singles_out <= "0010000"; -- 9
          when others => singles_out <= "1111111";
        end case;

        case tens is
          when 0 => tens_out <= "1000000"; -- 0
          when 1 => tens_out <= "1111001"; -- 1
          when 2 => tens_out <= "0100100"; -- 2
          when 3 => tens_out <= "0110000"; -- 3
          when 4 => tens_out <= "0011001"; -- 4
          when 5 => tens_out <= "0010010"; -- 5
          when 6 => tens_out <= "0000010"; -- 6
          when 7 => tens_out <= "1111000"; -- 7
          when 8 => tens_out <= "0000000"; -- 8
          when 9 => tens_out <= "0010000"; -- 9
          when others => tens_out <= "1111111";
        end case;
      end if;
    end process;
end rtl;
```

Listing 6: Two Digit Alarm Display