# Lab Report 4

EEL3701C - Digital Logic & Computer Systems

Benjamin Ruddy
UFID 33609976

## Pre-lab

### Pre-lab Questions + Homework

**Problem One:** Implement Muxian clock (no alarm)

essay3_planning_wst.md | problem1.vhd | problem2.vhd

```
entity problem1 is
    port (
            clk: in std_logic;
            seg_left: out std_logic_vector(7 downto 0) := (others => '0'); -- the number to the left of the ':' on our clock (0-5)
            seg_right: out std_logic_vector(7 downto 0) := (others => '0') -- the number to the right of the ':" on our clock (0-9)
        );
end entity problem1;

architecture bhv of problem1 is

    signal count: std_logic_vector (31 downto 0) := (others => '0');
    signal tmp: std_logic := '0';
    signal clock_hour: std_logic_vector (3 downto 0) := (others => '0');
    signal clock_minute: std_logic_vector (3 downto 0) := (others => '0');
begin

    process(clk)
    begin
        if(rising_edge(clk)) then

            if(unsigned(count) = to_unsigned(2e6, 32)) then -- count = f_base  / (2*f_slowclock) = 4e6/(2*1) = 2e6
                tmp <= not tmp;
                count <= (others => '0');
            else
                count <= count + 1;
            end if;

        end if;
    end process;

    process(tmp)
    begin
        if(rising_edge(tmp)) then

            if (clock_minute = "1001") then
                clock_minute <= (others => '0');
                if(clock_hour = "0101") then
                    clock_hour <= (others => '0');
                else
                    clock_hour <= clock_hour + 1;
                end if;
            else
                clock_minute <= clock_minute + 1;
            end if;

        end if;
    end process;

seg_left <= "11111100" when (clock_hour = "0000") else
            "01100000" when (clock_hour = "0001") else
            "11011010" when (clock_hour = "0010") else
            "11110010" when (clock_hour = "0011") else
            "01100110" when (clock_hour = "0100") else
            "10110110";

seg_right <= "11111100" when (clock_minute = "0000") else
            "01100000" when (clock_minute = "0001") else
            "11011010" when (clock_minute = "0010") else
            "11110010" when (clock_minute = "0011") else
            "01100110" when (clock_minute = "0100") else
            "10110110" when (clock_minute = "0101") else
            "10111110" when (clock_minute = "0110") else
            "11100000" when (clock_minute = "0111") else
            "11111110" when (clock_minute = "1000") else
            "11100110";

end architecture bhv;
```

Clock divider

Clock counters (hour and minute)

Assigning counter values to seven-segment display

Figure 1: VHDL Code for the Muxian clock (compiled, error-free)

**Problem Two:** Implement Muxian clock with an alarm Finite State Machine (FSM)

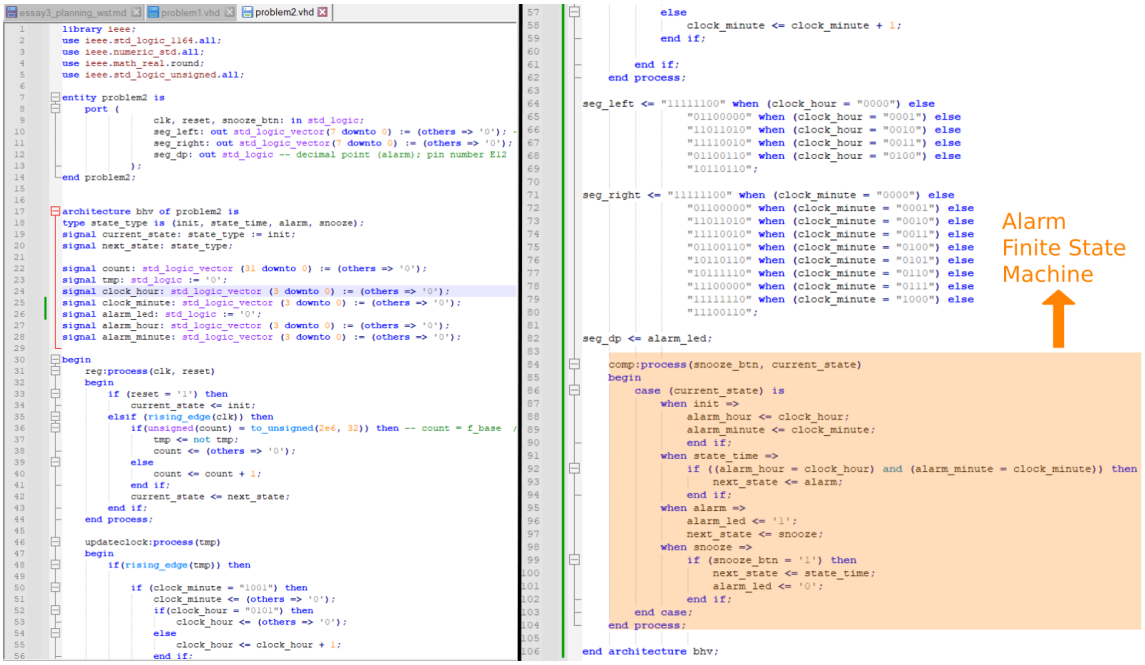



Figure 2: VHDL code for the Muxian clock, now with a finite state machine for the alarm

### Pre-lab Design and Implementation

The pre-lab design and implementation involved the usage of a clock divider, similar to those from past labs, where we reduced the frequency of the clock from 4 MHz to 1 Hz. This is done using VHDL processes, along with helpful library functions such as `rising_edge()`.

The logic of the time-keeping was handled with 4-bit counters, in which the hour counter was of modulo 6, and the minute counter was of modulo 10. These counter values, along with an optional alarm signal, were assigned to the seven-segment displays according to their corresponding pin in the **MXDB Manual**.

Finally, the design and implementation of the finite state machine was handled through a *switch statement* that identified the current state of our alarm machine, and stayed or changed state depending on the conditions, such as whether the alarm switch is turned on by the user.

### Pre-lab Reflection

This pre-lab, most of all, allowed me to gain a practical understanding of finite state machines. With the heavy amounts of terminology and abstract symbols/language, it was easy at first to wonder how this fit into real-world technology. Through the use of state signals in VHDL, along with help from switch statements, I was able to better comprehend this in my implementation of the Muxian alarm clock.

## Post-Lab

### Problem Statement

This lab consisted of:

1. (Problem 1) Synthesizing and programming the Muxian clock from **Pre-lab problem 1** onto the MXDB, testing it, and recording it.
2. (Problem 2) Synthesizing and programming the Muxian clock (with the alarm FSM) from **Pre-lab problem 2** onto the MXDB, testing it, and recording it.

### Design

Since the post-lab only involved implementing the designs from the pre-lab, there is no design done to be shown here. However, the following is a summary of our work until now:

- In VHDL, we first created a clock divider to slow down the frequency of our Muxian clock to 1 Hz.
- Next, we implemented part 1 (the clock with no alarm system) by incrementing two different count variables: one count for the hour (went from 0 to 5), and one for the minute (went from 0 to 9). These were modified on each clock cycle of our slow clock, and were reset to zero when incrementing at their maximum value.
- Using enums in VHDL, we created different states for our new clock, this time with an alarm system. These states changed according to the rules for the alarm *finite state machine* that we created, which involved behavior such as an LED turning on if the clock reached the time that an alarm was set at.

### Implementation

- As per usual, the internal 4 MHz clock of the MXDB (pin H4) was used as our base clock, upon which we then implemented the slow clock using the clock divider formula.
- We used the built-in seven segment display on the MXDB to visually represent our clock. Two vectors of bits were created to control the left display (the hour) and the right display (the minute). Each index in this vector corresponded to a certain segment of the display, which was practically implemented by assigning the indices to different pins, as per the MXDB manual.
- In the case of the alarm implementation (Problem 2), it was handled with the only leftover part of the seven-segment display that we weren't using: the dot at the bottom-right corner. This is convenient, as it simply involved creating a new signal and assigning it to one more pin.

- Finally, the inputs to our alarm clock – RESET and SNOOZE – were incorporated into our final product by using the typical set up of two resistors being hooked up to two switches, which were then fed into pins M12 and M11 on the MXDB.

**Testing**

Testing for this process involved:

1. (Problem 1) Ensuring that the Muxian clock counted from 0:0 all the way up to 5:9, and then starting over again from 0:0.

2. (Problem 2) Ensuring that the Alarm system worked as per the rules of our finite state machine (i.e. the reset button set the alarm to the new time, the alarm turned on when reaching the specified time, and the snooze button turning off the alarm).

The full videos demonstrating the correct behavior for both problems have been attached as separate video files in the Canvas submission to this lab report.

Below are some images of the implementation.

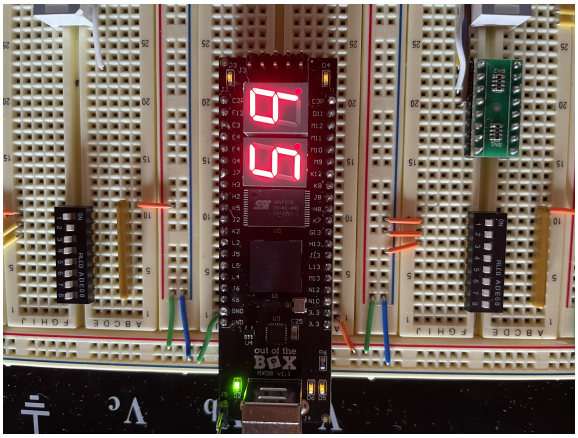


Figure 3: The Muxian clock reaching its maximum value of 5 hours and 9 minutes

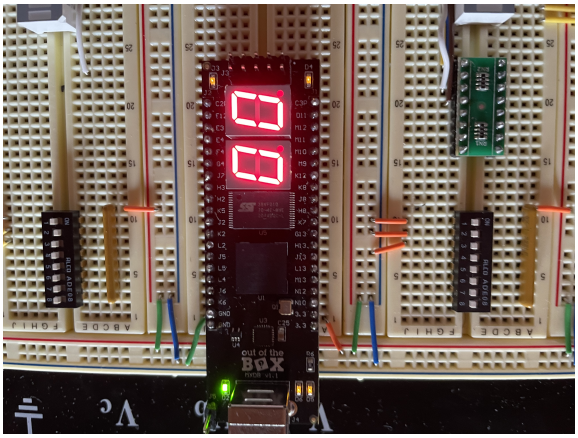


Figure 4: The Muxian clock back at 0:0, after being incremented from 5:9

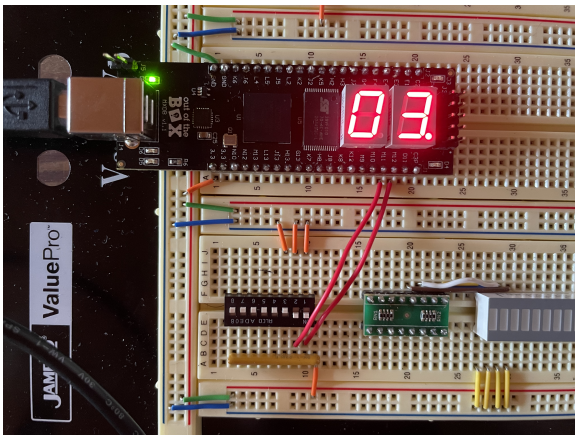
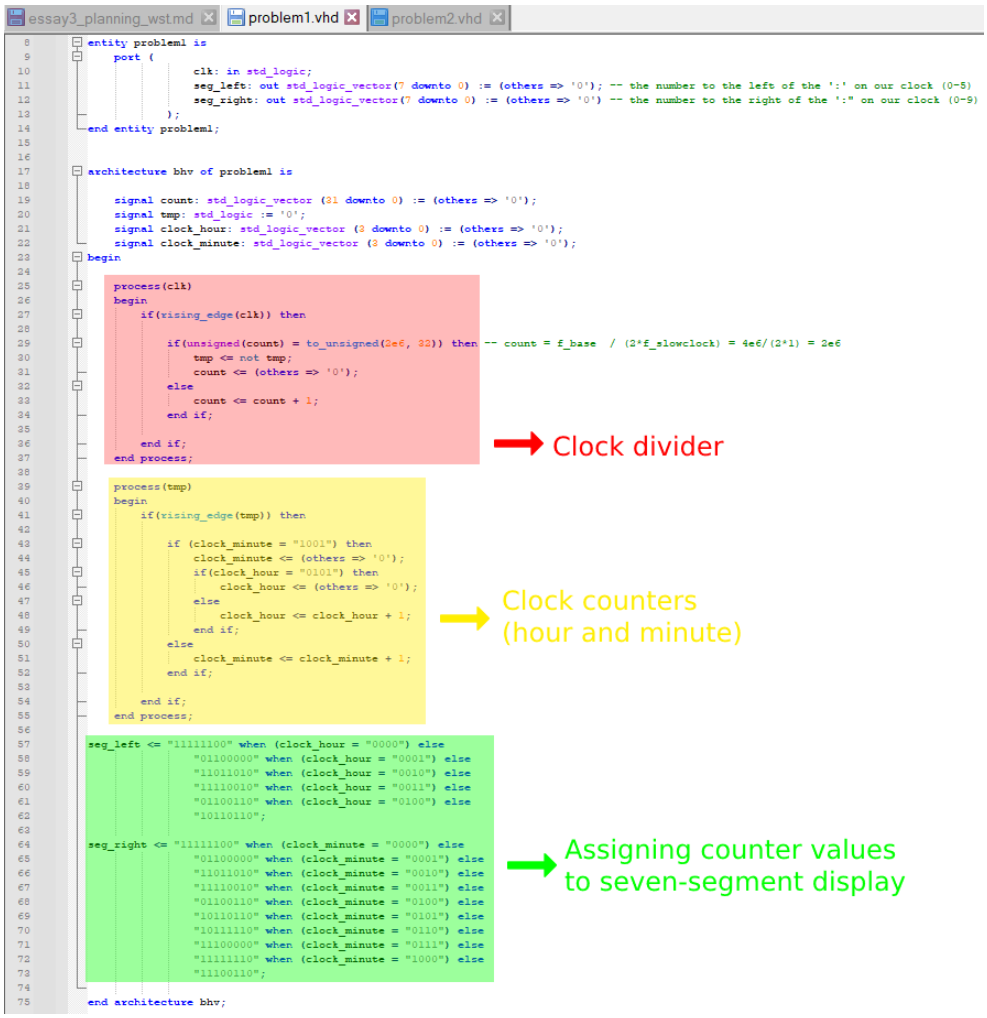



Figure 5: A picture of the Muxian alarm clock, in the ALARM state.
Note that the rightmost switch is SNOOZE, and to its left is the RESET switch.

**Conclusion**

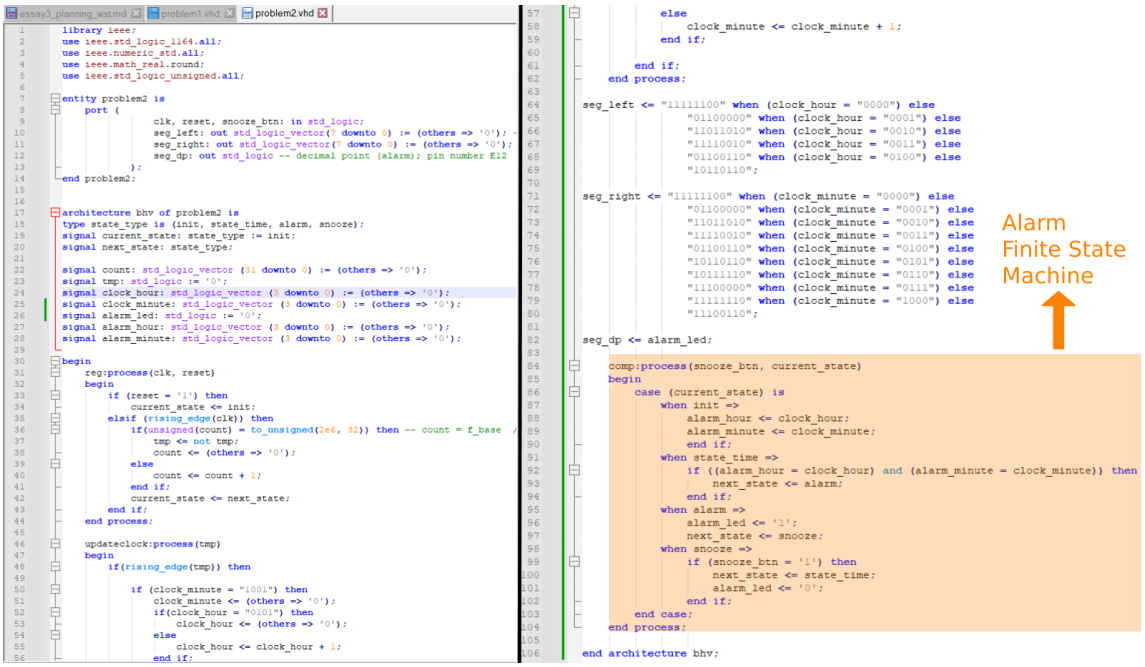
After completing this lab, I achieved an intimate understanding of the applications of finite state machines. While the pre-lab taught me how FSMs could be moved from a theortical representation to a code representation, the post-lab taught me the nuances of its live inner-workings, the difficulties associated with them, and some edge cases that can go unnoticed even though the code is compiled. As an example, I learned that the usage of the word `time` is not checked for by VHDL, despite it being a reserved word that otherwise prevents the functionality of a circuit. I also learned that two processes cannot have the same signal in its sensitivity list. Overall, this lab helped me not only solidify my pre-existing skills in VHDL and breadboarding, but it also gave me valuable insights to keep in mind for future labs.

## Appendix




```
entity problem1 is
    port (
            clk: in std_logic;
            seg_left: out std_logic_vector(7 downto 0) := (others => '0'); -- the number to the left of the ':' on our clock (0-5)
            seg_right: out std_logic_vector(7 downto 0) := (others => '0') -- the number to the right of the ':" on our clock (0-9)
        );
end entity problem1;

architecture bhv of problem1 is

    signal count: std_logic_vector (31 downto 0) := (others => '0');
    signal tmp: std_logic := '0';
    signal clock_hour: std_logic_vector (3 downto 0) := (others => '0');
    signal clock_minute: std_logic_vector (3 downto 0) := (others => '0');
begin

    process(clk)
    begin
        if(rising_edge(clk)) then

            if(unsigned(count) = to_unsigned(2e6, 32)) then -- count = f_base  / (2*f_slowclock) = 4e6/(2*1) = 2e6
                tmp <= not tmp;
                count <= (others => '0');
            else
                count <= count + 1;
            end if;

        end if;
    end process;

    process(tmp)
    begin
        if(rising_edge(tmp)) then

            if (clock_minute = "1001") then
                clock_minute <= (others => '0');
                if(clock_hour = "0101") then
                    clock_hour <= (others => '0');
                else
                    clock_hour <= clock_hour + 1;
                end if;
            else
                clock_minute <= clock_minute + 1;
            end if;

        end if;
    end process;

    seg_left <= "11111100" when (clock_hour = "0000") else
                "01100000" when (clock_hour = "0001") else
                "11011010" when (clock_hour = "0010") else
                "11110010" when (clock_hour = "0011") else
                "01100110" when (clock_hour = "0100") else
                "10110110";

    seg_right <= "11111100" when (clock_minute = "0000") else
                 "01100000" when (clock_minute = "0001") else
                 "11011010" when (clock_minute = "0010") else
                 "11110010" when (clock_minute = "0011") else
                 "01100110" when (clock_minute = "0100") else
                 "10110110" when (clock_minute = "0101") else
                 "10111110" when (clock_minute = "0110") else
                 "11100000" when (clock_minute = "0111") else
                 "11111110" when (clock_minute = "1000") else
                 "11100110";

end architecture bhv;
```

Figure 1: VHDL Code for the Muxian clock (compiled, error-free)




```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.round;
use ieee.std_logic_unsigned.all;

entity problem2 is
    port (
            clk, reset, snooze_btn: in std_logic;
            seg_left: out std_logic_vector(7 downto 0) := (others => '0');
            seg_right: out std_logic_vector(7 downto 0) := (others => '0');
            seg_dp: out std_logic -- decimal point (alarm); pin number E12
        );
end problem2;

architecture bhv of problem2 is
type state_type is (init, state_time, alarm, snooze);
signal current_state: state_type := init;
signal next_state: state_type;

signal count: std_logic_vector (31 downto 0) := (others => '0');
signal tmp: std_logic := '0';
signal clock_hour: std_logic_vector (3 downto 0) := (others => '0');
signal clock_minute: std_logic_vector (3 downto 0) := (others => '0');
signal alarm_led: std_logic := '0';
signal alarm_hour: std_logic_vector (3 downto 0) := (others => '0');
signal alarm_minute: std_logic_vector (3 downto 0) := (others => '0');

begin
    reg:process(clk, reset)
    begin
        if (reset = '1') then
            current_state <= init;
        elsif (rising_edge(clk)) then
            if(unsigned(count) = to_unsigned(2e6, 32)) then -- count = f_base /
                tmp <= not tmp;
                count <= (others => '0');
            else
                count <= count + 1;
            end if;
            current_state <= next_state;
        end if;
    end process;

    updateclock:process(tmp)
    begin
        if(rising_edge(tmp)) then

            if (clock_minute = "1001") then
                clock_minute <= (others => '0');
                if(clock_hour = "0101") then
                    clock_hour <= (others => '0');
                else
                    clock_hour <= clock_hour + 1;
                end if;
            else
                clock_minute <= clock_minute + 1;
            end if;

        end if;
    end process;

seg_left <= "11111100" when (clock_hour = "0000") else
            "01100000" when (clock_hour = "0001") else
            "11011010" when (clock_hour = "0010") else
            "11110010" when (clock_hour = "0011") else
            "01100110" when (clock_hour = "0100") else
            "10110110";

seg_right <= "11111100" when (clock_minute = "0000") else
             "01100000" when (clock_minute = "0001") else
             "11011010" when (clock_minute = "0010") else
             "11110010" when (clock_minute = "0011") else
             "01100110" when (clock_minute = "0100") else
             "10110110" when (clock_minute = "0101") else
             "10111110" when (clock_minute = "0110") else
             "11100000" when (clock_minute = "0111") else
             "11111110" when (clock_minute = "1000") else
             "11100110";

seg_dp <= alarm_led;

    comp:process(snooze_btn, current_state)
    begin
        case (current_state) is
            when init =>
                alarm_hour <= clock_hour;
                alarm_minute <= clock_minute;
                end if;
            when state_time =>
                if ((alarm_hour = clock_hour) and (alarm_minute = clock_minute)) then
                    next_state <= alarm;
                end if;
            when alarm =>
                alarm_led <= '1';
                next_state <= snooze;
            when snooze =>
                if (snooze_btn = '1') then
                    next_state <= state_time;
                    alarm_led <= '0';
                end if;
        end case;
    end process;

end architecture bhv;
```

Figure 2: VHDL code for the Muxian clock, now with a finite state machine for the alarm

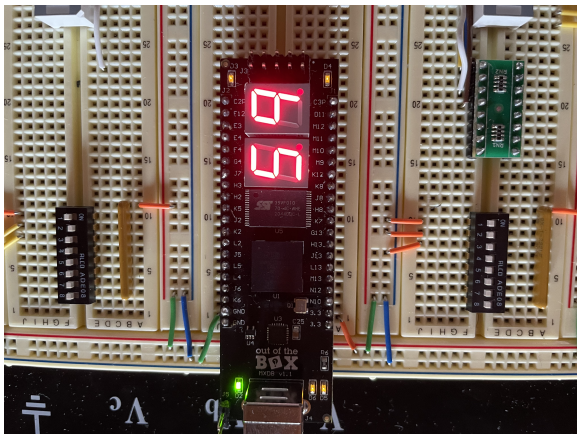


Figure 3: The Muxian clock reaching its maximum value of 5 hours and 9 minutes

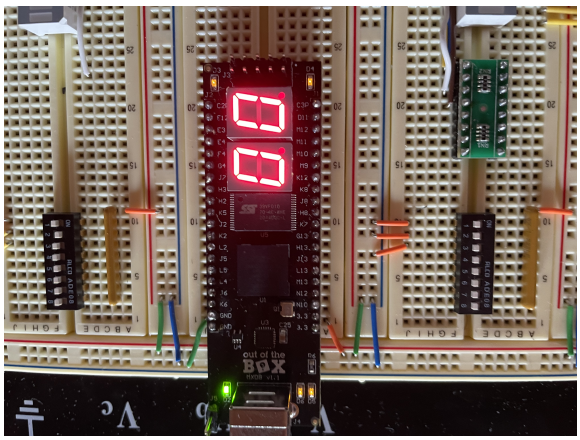


Figure 4: The Muxian clock back at 0:0, after being incremented from 5:9

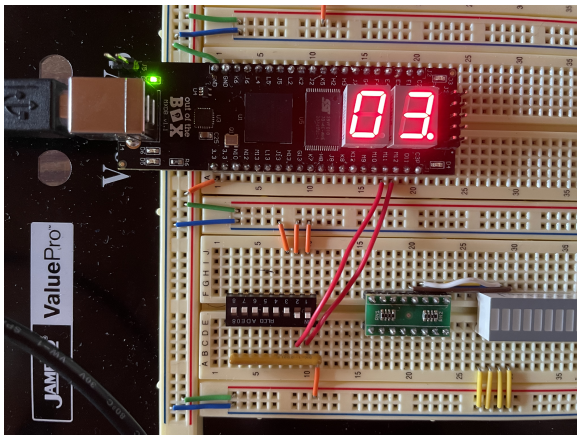



Figure 5: A picture of the Muxian alarm clock, in the ALARM state.