Chess Game Clock designed in FPGA Cyclone V DE1 SoC

Syed Shafi Ahmed
University of Greenwich*)*
Medway, The United Kingdom
sa7830z@gre.ac.uk

***Abstract*—The report consists of a design process and code for a simple chess game clock. Cyclone V DE1 SoC FPGA design board has been used and programmed with VHDL. The project successfully recreates the primary game-mode clock of a chess match. This includes user time configuration, player switching mechanism, countdown mechanism for individual player and time reset. A switching mechanism has been implemented to change the game mode to increment or Fischer game mode. However, the project was not a success in this section. The game mode functions properly but lacks the user time configuration option. Both game mode lacks pausing mechanism and do not point out the winner of the game when time runs out.**

I. Introduction

Chess clocks are placed at the beginning of the game to count down from the agreed-upon time. Only one of the two clocks is active at a moment, with players beginning their opponent's clock (and pausing their own) by pushing a button after each move. This report is about the design and programming process of recreating two chess game mode, classic game and Fischer game, in DE1 SoC FPGA board using VHDL.

II. Designing

A. Requirements

The design should replicate the two game modes of the chess game in a single code.

The first game mode is the classic game mode. Each player has a certain time which is agreed upon both players. Game starts with player 1 as he makes his move after pressing the timer countdown. Player 2 will start his move after player 1 and press his key on the timer to start his countdown. The timer will stop player 1 countdown while player 2 countdown is ongoing. Switching player will not reset the countdown rather will pause indefinitely. Players has to complete their moves or win the game before their time reaches to 0. Timer reaching to 0 will cause the respective player a game lose even though the player is wining.

The second game mode will have a bonus time increment while switching players. The bonus time is agreed upon both players. Unlike game mode 1, game mode 2 will have very short initial time. The players can increase their time by making their move quicker and switch player. The player finishing the move will be given a bonus time to their countdown. The win and lose rules follows the same principle as game mode 1.

With the descriptions of game mode above the FPGA board should be able to:

- Select Game Mode.
- Give players option to set initial time (Game Mode 1) and Bonus time (Game Mode 2)
- Start Count down and maintain the countdown for each player while one of the player is idle
- Reset the game
- Show the countdown on the Display
- Point out the player having no more time left to play

B. State Machine Diagram

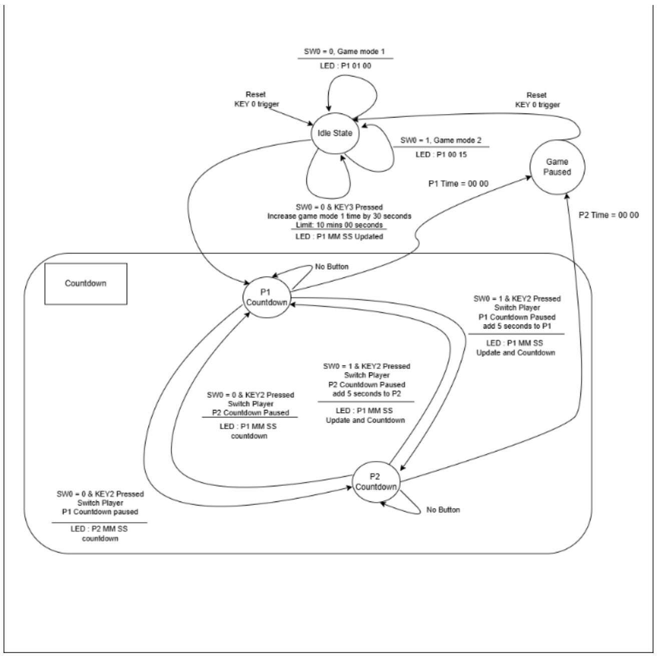
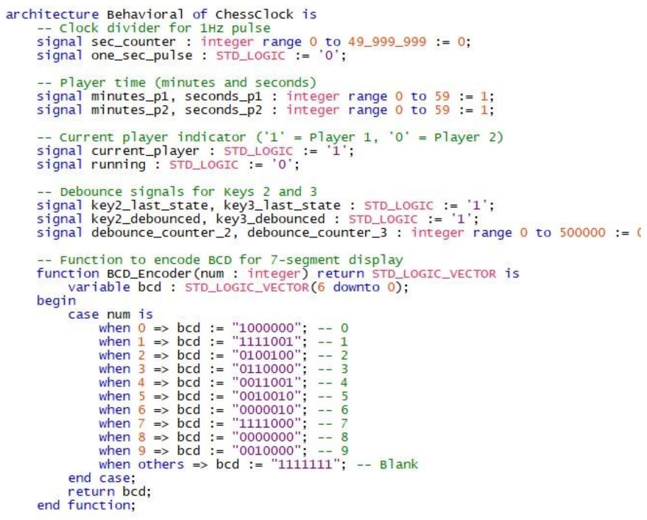


Fig 1: State Machine Diagram of the Chess Game Clock

Fig 1 is the visual representation in the software design that depicts the different states the chess game clock can be in and how it transitions between different states based on the conditions and triggered buttons. The VHDL code will be constructed based on Fig 1. The state machine diagram follow has been constructed using Mealy Machine Diagram structure.

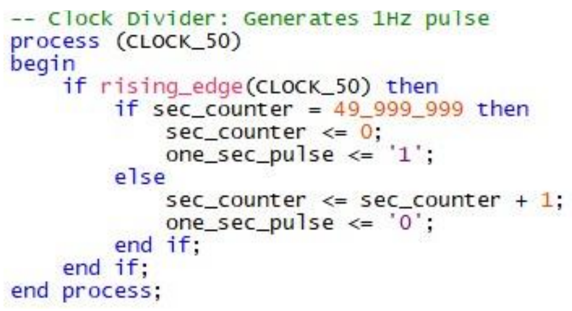
III. VHDL Code

This code implements a chess clock for the DE1-SoC FPGA, managing timers for two players. A clock divider (sec_counter) generates a 1Hz pulse (one_sec_pulse) from the 50MHz FPGA clock (CLOCK_50). Player minutes and seconds are tracked (minutes_p1, seconds_p1, minutes_p2, seconds_p2), starting at 1 minute each. The current player (current_player) determines whose timer decrements. A debounce mechanism ensures stable button presses for KEY2 (pause/reset) and KEY3 (switch player). The BCD_Encoder function converts numbers into 7-segment display format, allowing time visualization on HEX0-HEX5. The running signal controls whether the countdown is active or paused.



```
architecture Behavioral of ChessClock is
    -- Clock divider for 1Hz pulse
    signal sec_counter : integer range 0 to 49_999_999 := 0;
    signal one_sec_pulse : STD_LOGIC := '0';

    -- Player time (minutes and seconds)
    signal minutes_p1, seconds_p1 : integer range 0 to 59 := 1;
    signal minutes_p2, seconds_p2 : integer range 0 to 59 := 1;

    -- Current player indicator ('1' = Player 1, '0' = Player 2)
    signal current_player : STD_LOGIC := '1';
    signal running : STD_LOGIC := '0';

    -- Debounce signals for Keys 2 and 3
    signal key2_last_state, key3_last_state : STD_LOGIC := '1';
    signal key2_debounced, key3_debounced : STD_LOGIC := '1';
    signal debounce_counter_2, debounce_counter_3 : integer range 0 to 500000 :=

    -- Function to encode BCD for 7-segment display
    function BCD_Encoder(num : integer) return STD_LOGIC_VECTOR is
        variable bcd : STD_LOGIC_VECTOR(6 downto 0);
    begin
        case num is
            when 0 => bcd := "1000000"; -- 0
            when 1 => bcd := "1111001"; -- 1
            when 2 => bcd := "0100100"; -- 2
            when 3 => bcd := "0110000"; -- 3
            when 4 => bcd := "0011001"; -- 4
            when 5 => bcd := "0010010"; -- 5
            when 6 => bcd := "0000010"; -- 6
            when 7 => bcd := "1111000"; -- 7
            when 8 => bcd := "0000000"; -- 8
            when 9 => bcd := "0010000"; -- 9
            when others => bcd := "1111111"; -- Blank
        end case;
        return bcd;
    end function;
```

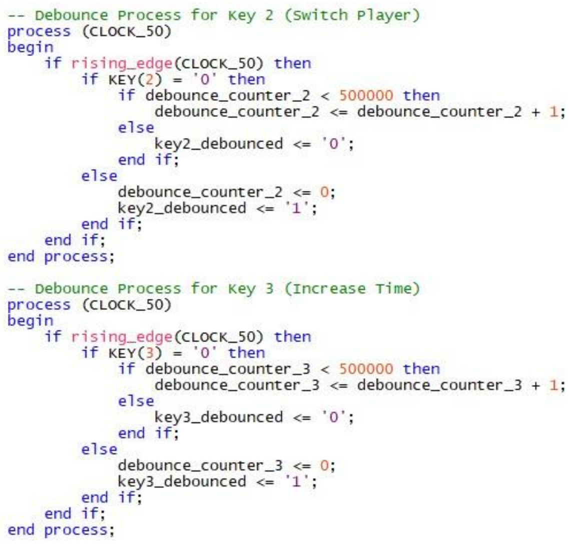
Fig 2: Code part 1



```
-- Clock Divider: Generates 1Hz pulse
process (CLOCK_50)
begin
    if rising_edge(CLOCK_50) then
        if sec_counter = 49_999_999 then
            sec_counter <= 0;
            one_sec_pulse <= '1';
        else
            sec_counter <= sec_counter + 1;
            one_sec_pulse <= '0';
        end if;
    end if;
end process;
```

Fig 3: Clock Divider

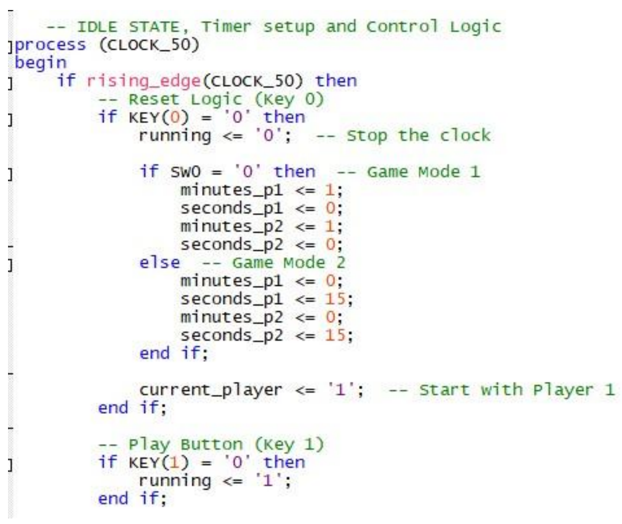
This code clock divider process converts the 50 MHz clock (CLOCK_50) into a 1Hz pulse (one_sec_pulse). It increments sec_counter every clock cycle. When sec_counter reaches 49,999,999 (one second), it resets to 0, and one_sec_pulse is set to '1' for one cycle.



```
-- Debounce Process for Key 2 (Switch Player)
process (CLOCK_50)
begin
    if rising_edge(CLOCK_50) then
        if KEY(2) = '0' then
            if debounce_counter_2 < 500000 then
                debounce_counter_2 <= debounce_counter_2 + 1;
            else
                key2_debounced <= '0';
            end if;
        else
            debounce_counter_2 <= 0;
            key2_debounced <= '1';
        end if;
    end if;
end process;

-- Debounce Process for Key 3 (Increase Time)
process (CLOCK_50)
begin
    if rising_edge(CLOCK_50) then
        if KEY(3) = '0' then
            if debounce_counter_3 < 500000 then
                debounce_counter_3 <= debounce_counter_3 + 1;
            else
                key3_debounced <= '0';
            end if;
        else
            debounce_counter_3 <= 0;
            key3_debounced <= '1';
        end if;
    end if;
end process;
```

Fig 4: Debouncing Key 2 and Key 3

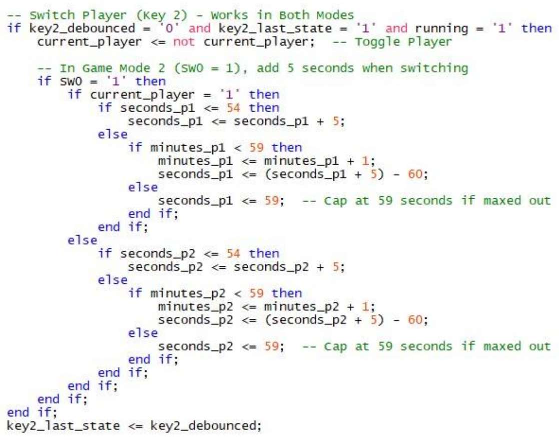
These code debounce processes stabilize button inputs (KEY(2) and KEY(3)) by filtering mechanical bouncing. Each process increments a counter (500,000 cycles) when the button is pressed ('0'). Once stabilized, keyX_debounced is set to '0'. When released, the counter resets, and keyX_debounced returns to '1'.



```
    -- IDLE STATE, Timer setup and Control Logic
process (CLOCK_50)
begin
    if rising_edge(CLOCK_50) then
        -- Reset Logic (Key 0)
        if KEY(0) = '0' then
            running <= '0';  -- Stop the clock

            if SW0 = '0' then  -- Game Mode 1
                minutes_p1 <= 1;
                seconds_p1 <= 0;
                minutes_p2 <= 1;
                seconds_p2 <= 0;
            else  -- Game Mode 2
                minutes_p1 <= 0;
                seconds_p1 <= 15;
                minutes_p2 <= 0;
                seconds_p2 <= 15;
            end if;

            current_player <= '1';  -- Start with Player 1
        end if;

        -- Play Button (Key 1)
        if KEY(1) = '0' then
            running <= '1';
        end if;
```

Fig 5: IDLE State

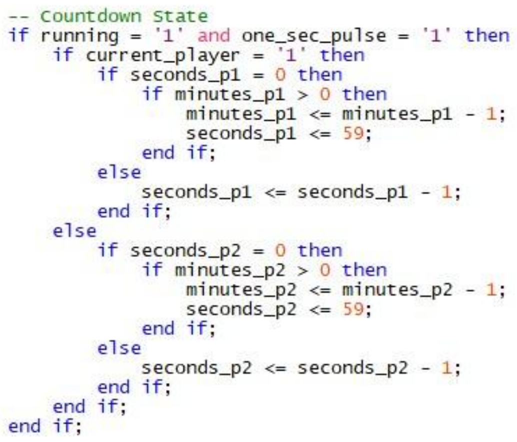
This code process manages timer setup and control. Pressing KEY(0) resets the clock and sets Game Mode based on SW0 (1-minute or 15-second timers). current_player starts as Player 1. Pressing KEY(1) starts the timer (running = '1'). This ensures proper initialization and game flow control.



```
-- Switch Player (Key 2) - Works in Both Modes
if key2_debounced = '0' and key2_last_state = '1' and running = '1' then
    current_player <= not current_player;  -- Toggle Player

    -- In Game Mode 2 (SW0 = 1), add 5 seconds when switching
    if SW0 = '1' then
        if current_player = '1' then
            if seconds_p1 <= 54 then
                seconds_p1 <= seconds_p1 + 5;
            else
                if minutes_p1 < 59 then
                    minutes_p1 <= minutes_p1 + 1;
                    seconds_p1 <= (seconds_p1 + 5) - 60;
                else
                    seconds_p1 <= 59;  -- Cap at 59 seconds if maxed out
                end if;
            end if;
        else
            if seconds_p2 <= 54 then
                seconds_p2 <= seconds_p2 + 5;
            else
                if minutes_p2 < 59 then
                    minutes_p2 <= minutes_p2 + 1;
                    seconds_p2 <= (seconds_p2 + 5) - 60;
                else
                    seconds_p2 <= 59;  -- Cap at 59 seconds if maxed out
                end if;
            end if;
        end if;
    end if;
end if;
key2_last_state <= key2_debounced;
```

Fig 8: Player Switching mode

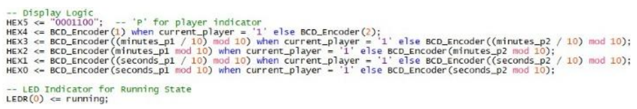
This code switches the active player when key2_debounced = '0' (button press) and running = '1'. The active player (current_player) toggles. In Game Mode 2 (SW0 = '1'), 5 seconds are added to the current player's timer, adjusting for minute overflow, and ensuring no timer exceeds 59 seconds.



```
-- Countdown State
if running = '1' and one_sec_pulse = '1' then
    if current_player = '1' then
        if seconds_p1 = 0 then
            if minutes_p1 > 0 then
                minutes_p1 <= minutes_p1 - 1;
                seconds_p1 <= 59;
            end if;
        else
            seconds_p1 <= seconds_p1 - 1;
        end if;
    else
        if seconds_p2 = 0 then
            if minutes_p2 > 0 then
                minutes_p2 <= minutes_p2 - 1;
                seconds_p2 <= 59;
            end if;
        else
            seconds_p2 <= seconds_p2 - 1;
        end if;
    end if;
end if;
```

Fig 9: Countdown State

This code process manages the countdown for both players when running = '1' and one_sec_pulse = '1'. For Player 1 (current_player = '1'), it decrements seconds and handles minute transitions. The same logic applies for Player 2, ensuring both timers countdown properly.



```
-- Display Logic
HEX5 <= "0001100";  -- 'P' for player indicator
HEX4 <= BCD_Encoder(1) when current_player = '1' else BCD_Encoder(2);
HEX3 <= BCD_Encoder((minutes_p1 / 10) mod 10) when current_player = '1' else BCD_Encoder((minutes_p2 / 10) mod 10);
HEX2 <= BCD_Encoder(minutes_p1 mod 10) when current_player = '1' else BCD_Encoder(minutes_p2 mod 10);
HEX1 <= BCD_Encoder((seconds_p1 / 10) mod 10) when current_player = '1' else BCD_Encoder((seconds_p2 / 10) mod 10);
HEX0 <= BCD_Encoder(seconds_p1 mod 10) when current_player = '1' else BCD_Encoder(seconds_p2 mod 10);

-- LED Indicator for Running State
LEDR(0) <= running;
```

Fig 10: Display

This code controls the 7-segment displays for the chess clock. HEX5 shows the player indicator ('P'). HEX4-HEX0 display the active player's minutes and seconds using the BCD_Encoder function. The LED indicator (LEDR(0)) shows whether the clock is running (running signal).

IV. User Instrusction and Testing

Before staring the game, the system setup is required. ChessClock.sof should be navigated in the output_folder for adding the program to the board. Cyclone V DE1 SoC device should be selected in the programming section before adding the file. Compilation is not required. Pins are to be checked if not working properly. Pin planner has been provided below.

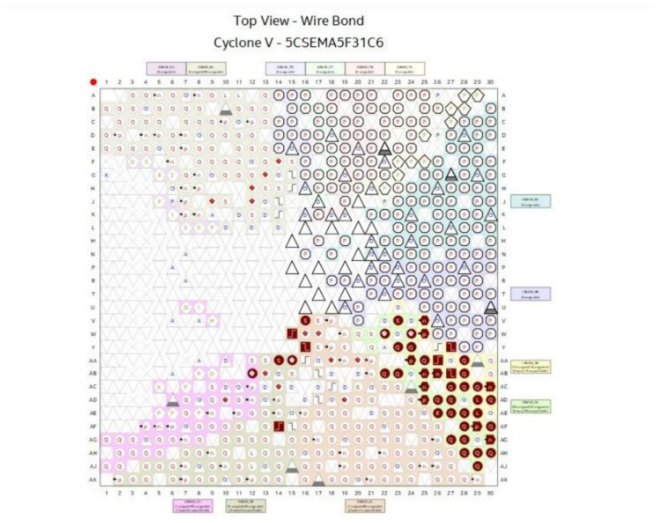


Fig 11: Pin Planner

A. *Gamemode 1*

After setting up the device, Check for the switch labelled SW0. If SW0 is off or in 0 state, the game mode is classic. To play classing mode continue with SW0 as 0 state. An initial minimum game time is provided with player 1 as starting player. Initial time 1 minute.

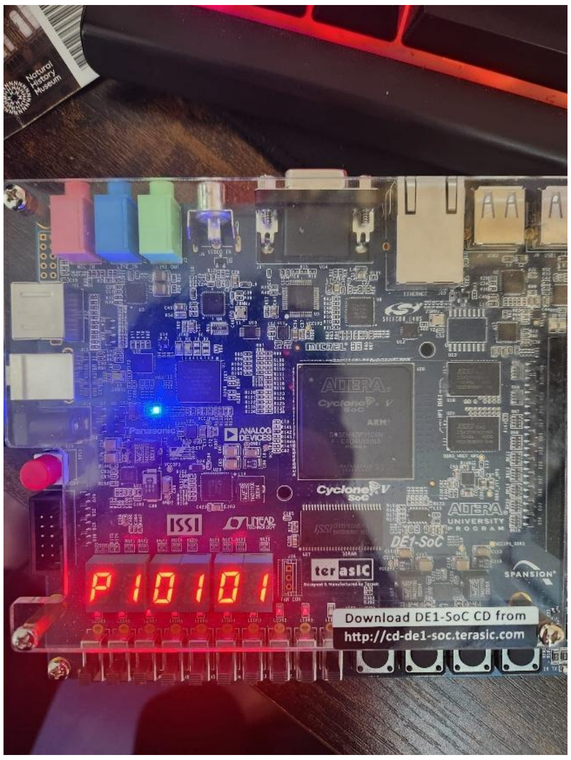


Fig 12: IDLE State

To increase game time, use key labelled Key 3. Pressing Key 3 each time will increase the game time by 30 seconds and can be increased up to 10 minutes. Time cannot be changed during play time. Must press reset, Key 0.

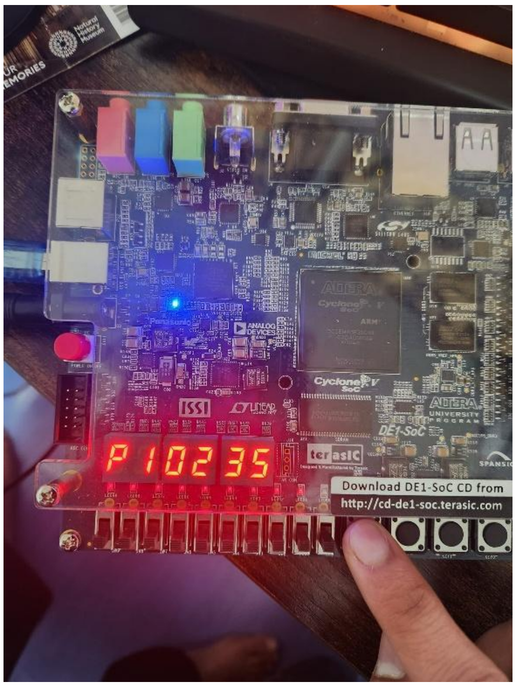


Fig 13: Increase Game time

Press key labelled Key 1, to start the game. The player shown in the display will start playing (Initially Player 1). Before starting the game Player cannot be switched.

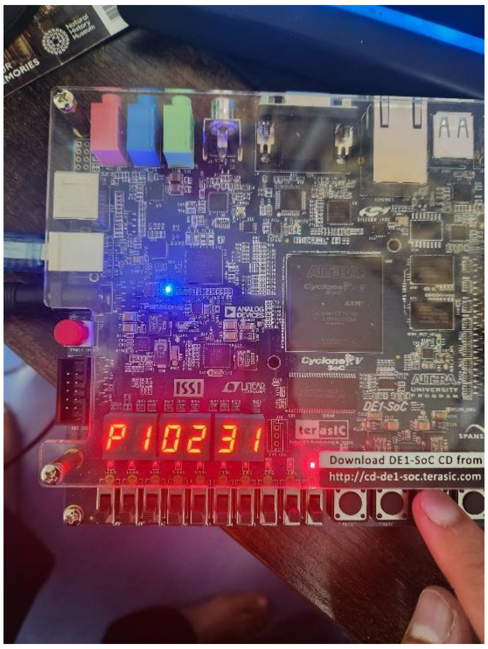


Fig 14: Start Game

Press key labelled Key 2, to switch players. The countdown time will be paused for the player not shown in the display. Player reaching 0 mins and 0 seconds will be declared as losing player.

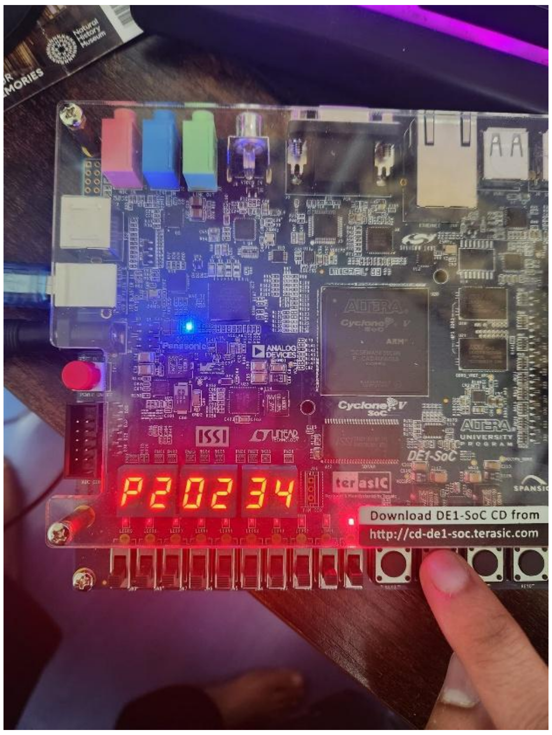


Fig 15: Change Player

Press key labelled Key 0 to Reset the game.

B. *Gamemode 2*

To play game mode 2, toggle SW0 to on or 1 state. Press Key 0 to reset to idle state of game mode 2. The initial time will be set to 15 seconds for both players, which cannot be changed.

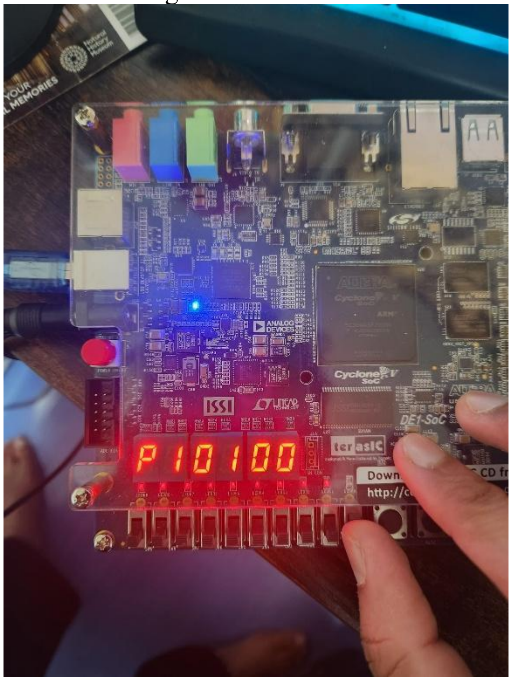


Fig 16: SW0 to 1 to change mode

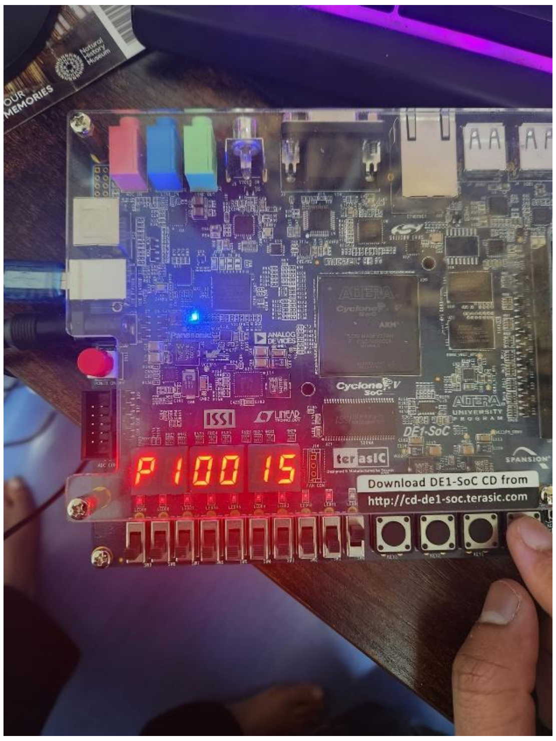


Fig 17: Reset to Set the game mode

To start press Key 1 and to switch player press Key 2, similar principle as game mode 1. Only difference, while switching player, 5 seconds bonus will be added to the player finishing move.

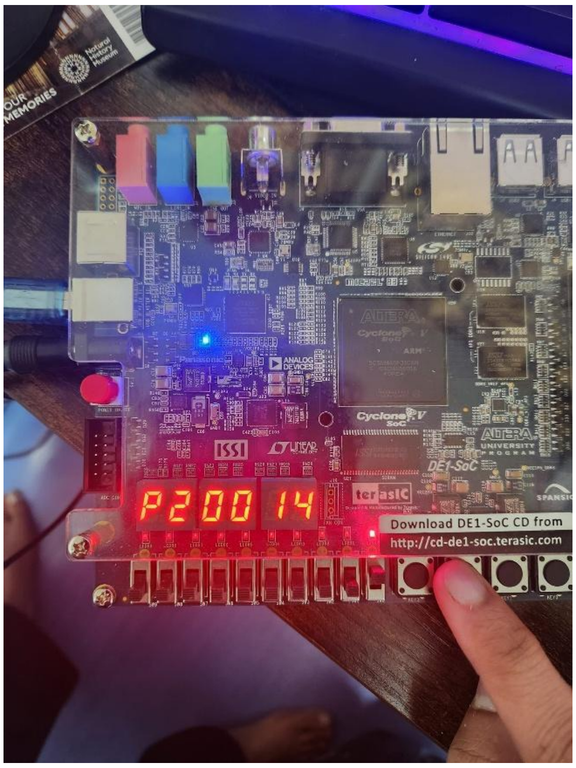


Fig 19: Change player

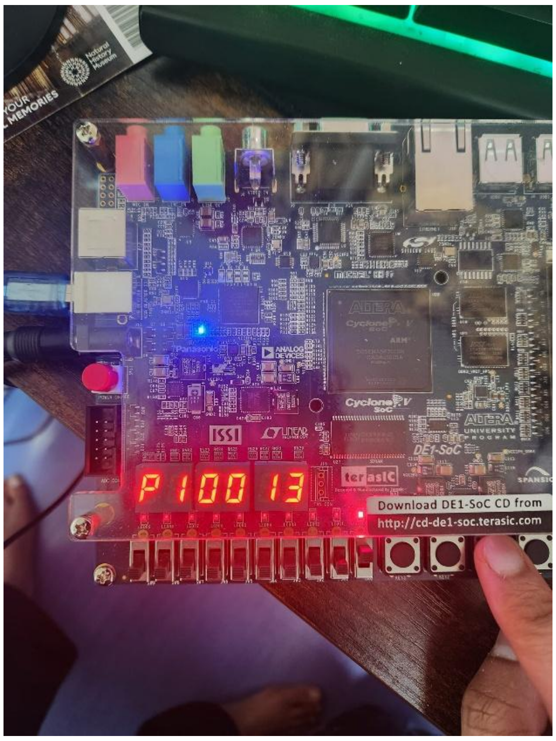


Fig :18 Start the game mode

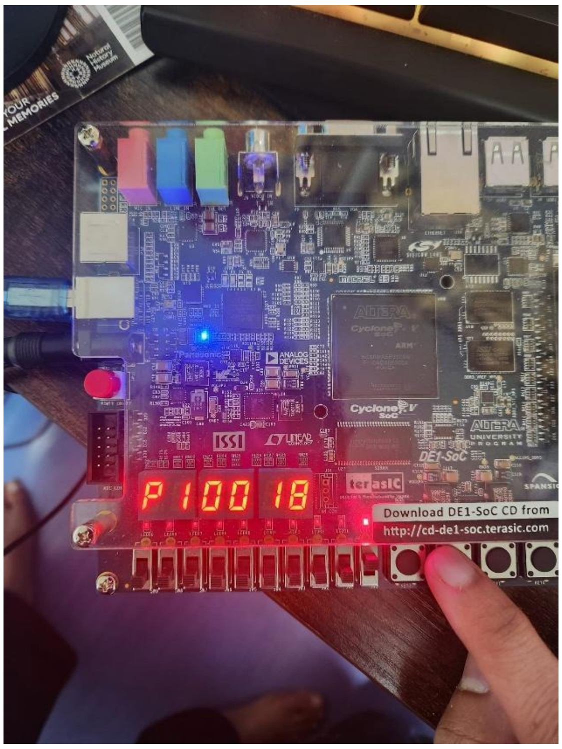


Fig 20: Switching player and 5 second bonus

Press reset to reset the game. To play game mode 1 again, toggle SW0 to off and press Key 0 to set the game as idle state of game mode 1.

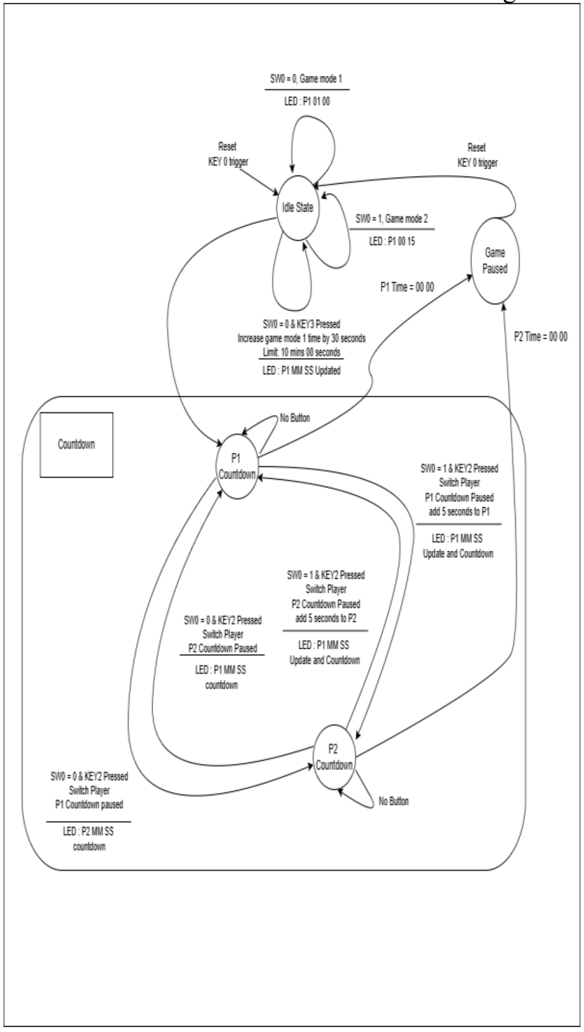
Appendices

A. Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ChessClock is
  Port (
    CLOCK_50 : in STD_LOGIC;
    KEY : in STD_LOGIC_VECTOR(3 downto
0);
    SW0 : in STD_LOGIC;
    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5 :
out STD_LOGIC_VECTOR(6 downto 0);
    LEDR : out STD_LOGIC_VECTOR(0 downto
0)
  );
end ChessClock;

architecture Behavioral of ChessClock is
  -- Clock divider for 1Hz pulse
  signal sec_counter : integer range 0 to
49_999_999 := 0;
  signal one_sec_pulse : STD_LOGIC := '0';

  -- Player time (minutes and seconds)
  signal minutes_p1, seconds_p1 : integer range 0
to 59 := 1;
  signal minutes_p2, seconds_p2 : integer range 0
to 59 := 1;

  -- Current player indicator ('1' = Player 1, '0' =
Player 2)
  signal current_player : STD_LOGIC := '1';
  signal running : STD_LOGIC := '0';

  -- Debounce signals for Keys 2 and 3
  signal key2_last_state, key3_last_state :
STD_LOGIC := '1';
  signal key2_debounced, key3_debounced :
STD_LOGIC := '1';
  signal debounce_counter_2, debounce_counter_3
: integer range 0 to 500000 := 0;

  -- Function to encode BCD for 7-segment display
  function BCD_Encoder(num : integer) return
STD_LOGIC_VECTOR is
    variable bcd : STD_LOGIC_VECTOR(6
downto 0);
  begin
    case num is
      when 0 => bcd := "1000000"; -- 0
      when 1 => bcd := "1111001"; -- 1
      when 2 => bcd := "0100100"; -- 2
      when 3 => bcd := "0110000"; -- 3
      when 4 => bcd := "0011001"; -- 4
      when 5 => bcd := "0010010"; -- 5
      when 6 => bcd := "0000010"; -- 6
      when 7 => bcd := "1111000"; -- 7
      when 8 => bcd := "0000000"; -- 8
      when 9 => bcd := "0010000"; -- 9
      when others => bcd := "1111111"; -- Blank
    end case;
    return bcd;
  end function;

begin
  -- Clock Divider: Generates 1Hz pulse
  process (CLOCK_50)
  begin
    if rising_edge(CLOCK_50) then
      if sec_counter = 49_999_999 then
        sec_counter <= 0;
        one_sec_pulse <= '1';
      else
        sec_counter <= sec_counter + 1;
        one_sec_pulse <= '0';
      end if;
    end if;
  end process;

  -- Debounce Process for Key 2 (Switch Player)
  process (CLOCK_50)
  begin
    if rising_edge(CLOCK_50) then
      if KEY(2) = '0' then
        if debounce_counter_2 < 500000 then
          debounce_counter_2 <=
debounce_counter_2 + 1;
        else
          key2_debounced <= '0';
        end if;
      else
        debounce_counter_2 <= 0;
        key2_debounced <= '1';
      end if;
    end if;
  end process;

  -- Debounce Process for Key 3 (Increase Time)
  process (CLOCK_50)
  begin
    if rising_edge(CLOCK_50) then
      if KEY(3) = '0' then
        if debounce_counter_3 < 500000 then
          debounce_counter_3 <=
debounce_counter_3 + 1;
        else
          key3_debounced <= '0';
        end if;
      else
        debounce_counter_3 <= 0;
        key3_debounced <= '1';
      end if;
    end if;
  end process;

 -- IDLE STATE, Timer setup and Control Logic
process (CLOCK_50)
begin
  if rising_edge(CLOCK_50) then
    -- Reset Logic (Key 0)
```

```
        if KEY(0) = '0' then
            running <= '0';  -- Stop the clock

            if SW0 = '0' then  -- Game Mode 1
                minutes_p1 <= 1;
                seconds_p1 <= 0;
                minutes_p2 <= 1;
                seconds_p2 <= 0;
            else  -- Game Mode 2
                minutes_p1 <= 0;
                seconds_p1 <= 15;
                minutes_p2 <= 0;
                seconds_p2 <= 15;
            end if;

            current_player <= '1';  -- Start with Player 1
        end if;

        -- Play Button (Key 1)
        if KEY(1) = '0' then
            running <= '1';
        end if;



        -- Switch Player (Key 2) - Works in Both
Modes
        if key2_debounced = '0' and key2_last_state =
'1' and running = '1' then
            current_player <= not current_player;   --
Toggle Player

            -- In Game Mode 2 (SW0 = 1), add 5 seconds
when switching
            if SW0 = '1' then
                if current_player = '1' then
                    if seconds_p1 <= 54 then
                        seconds_p1 <= seconds_p1 + 5;
                    else
                        if minutes_p1 < 59 then
                            minutes_p1 <= minutes_p1 + 1;
                            seconds_p1 <= (seconds_p1 + 5) -
60;
                        else
                            seconds_p1 <= 59;  -- Cap at 59
seconds if maxed out
                        end if;
                    end if;
                else
                    if seconds_p2 <= 54 then
                        seconds_p2 <= seconds_p2 + 5;
                    else
                        if minutes_p2 < 59 then
                            minutes_p2 <= minutes_p2 + 1;
                            seconds_p2 <= (seconds_p2 + 5) -
60;
                        else
                            seconds_p2 <= 59;  -- Cap at 59
seconds if maxed out
                        end if;
                    end if;
                end if;
            end if;
        end if;
        key2_last_state <= key2_debounced;



        -- Countdown State
        if running = '1' and one_sec_pulse = '1' then
            if current_player = '1' then
                if seconds_p1 = 0 then
                    if minutes_p1 > 0 then
                        minutes_p1 <= minutes_p1 - 1;
                        seconds_p1 <= 59;
                    end if;
                else
                    seconds_p1 <= seconds_p1 - 1;
                end if;
            else
                if seconds_p2 = 0 then
                    if minutes_p2 > 0 then
                        minutes_p2 <= minutes_p2 - 1;
                        seconds_p2 <= 59;
                    end if;
                else
                    seconds_p2 <= seconds_p2 - 1;
                end if;
            end if;
        end if;
    end if;
end process;


    -- Display Logic
    HEX5 <= "0001100";  -- 'P' for player indicator
    HEX4 <= BCD_Encoder(1) when current_player
= '1' else BCD_Encoder(2);
    HEX3 <= BCD_Encoder((minutes_p1 / 10) mod
10) when current_player = '1' else
BCD_Encoder((minutes_p2 / 10) mod 10);
    HEX2 <= BCD_Encoder(minutes_p1 mod 10)
when current_player = '1' else
BCD_Encoder(minutes_p2 mod 10);
    HEX1 <= BCD_Encoder((seconds_p1 / 10) mod
10) when current_player = '1' else
BCD_Encoder((seconds_p2 / 10) mod 10);
    HEX0 <= BCD_Encoder(seconds_p1 mod 10)
when current_player = '1' else
BCD_Encoder(seconds_p2 mod 10);

    -- LED Indicator for Running State
    LEDR(0) <= running;

end Behavioral;
```

B. State Machine Diagram


SW0 = 0, Game mode 1
LED : P1 01 00
Reset
KEY 0 trigger
Idle State
SW0 = 1, Game mode 2
LED : P1 00 15
Reset
KEY 0 trigger
Game
Paused
P1 Time = 00 00
P2 Time = 00 00
SW0 = 0 & KEY3 Pressed
Increase game mode 1 time by 30 seconds
Limit: 10 mins 00 seconds
LED : P1 MM SS Updated
Countdown
No Button
P1
Countdown
SW0 = 1 & KEY2 Pressed
Switch Player
P1 Countdown Paused
add 5 seconds to P1
LED : P1 MM SS
Update and Countdown
SW0 = 1 & KEY2 Pressed
Switch Player
P2 Countdown Paused
add 5 seconds to P2
LED : P1 MM SS
Update and Countdown
SW0 = 0 & KEY2 Pressed
Switch Player
P2 Countdown Paused
LED : P1 MM SS
countdown
P2
Countdown
No Button
SW0 = 0 & KEY2 Pressed
Switch Player
P1 Countdown paused
LED : P2 MM SS
countdown