VNU-HCM
HO CHI MINH UNIVERSITY OF TECHNOLOGY

PROJECT REPORT

SUBJECT: LOGIC DESIGN WITH HDL

Lecturer Lab: Nguyễn Thiên Ân
Class: CC16

STT	Student ID	Student's Name
1	2252059	Bùi Gia Bảo
2	2252059	Đặng Đoàn Đức Hoàng
3	2252522	Trần Cẩm Hòa
4	2252505	Nguyễn Tuấn Phong

CONTENTS

I. INTRODUCTION

1) Topic

Chess clocks are set at the start of the game to count down from the agreed time. Just one of the two clocks runs at a time, with players starting their opponent's clock (and pausing their own) by pressing a button after making each move. The first digital chess clock was invented by a student at Cornell University in 1970s.

In this project, we try to simulate a chess clock with the main time unit is minutes and seconds. The clock is equipped some basic functions such as adjusting minutes and seconds, stopping and running buttons, …

2) Scope

The chess clock will work on 4 digit 7 segments controlled by ArtyZ7, a FPGA development board from Diligent. Some buttons are also linked with FPGA as the way to control the clock.

3) Summary

This report includes 5 other main parts:

- Backgrounds and applications: Where we talk about necessary theory and applications of the circuit.
- Design: Present the ideal, block diagram, flowchart of the design.
- Implementation: Includes technical modules and its code.
- Results: Show setups, waveforms, explanations and report timing, resources.
- Conclusion: summary working achievements, advantages, disadvantages and future works.

II. BACKGROUNDS AND APPLICATIONS

1) Backgrounds

a) Structure in 4 digits 7 segments LED (include how to use multiplexing technic to control):
   - A 4-digit 7-segment LED display has 14 pins. 8 of the pins are for the 8 LEDs on each of the 7 segment displays, which includes A-G and DP (decimal point). The other 4 pins (14, 11, 10, 6) represent each of 4 digits from DIG1-DIG4, and the left 2 pins are used for activating 2 vertical dots in the middle of the board.

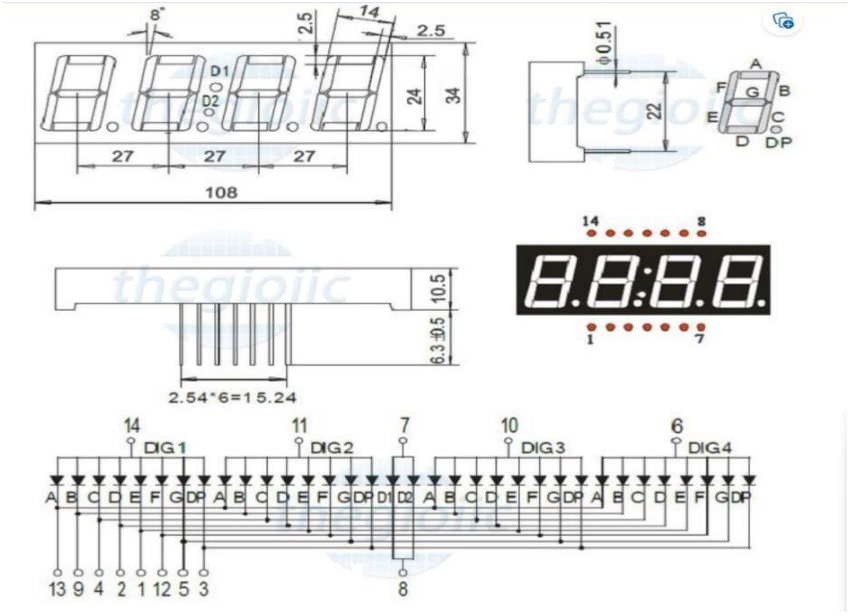


Figure 1: Structure of 4-digit 7-segment LED display

- Each segment in the display module is multiplexed, meaning it shares the same anode connection points. And each of the four digits in the module have their own common cathode connection point. This allows each digit to be turned on or off independently. Also, this multiplexing technique turns the massive amount of microcontroller pins necessary to control a display into just eleven or twelve.
- In order to display four different numbers on this 4-digit display at a time, we use a method called multiplexing. In detail, this method shows one digit at a time on a display unit and switch between display units very fast. Due to persistence of vision, human eye cannot differentiate between which display is ON/OFF. The human eye just visualizes all the 4 display units to be ON all the time.

b) External reference clock on Artyz7 (125Mhz):

- The Arty Z7 board external reference clock input works by allowing an external clock signal to be connected to the board's clock input pins. This clock signal is typically generated by an external oscillator or clock source that provides a stable and precise frequency output. When the external reference clock is connected to the arty z7 board. The board's onboard clocking circuitry can use it as a reference for generating internal clock signals
- The onboard clocking circuitry typically includes one or more phase-locked loops (PLLs) that can multiply or divide the

frequency of the external reference clock to generate clock signals at different frequencies.

c) Arduino/chipKIT Shield Connector on Artyz7:

- The Arty Z7 can be connected to standard Arduino and chipKIT shields to add extended functionality. Special care was taken while designing the Arty Z7 to make sure it is compatible with the majority of Arduino and chipKIT shields on the market. The shield connector has 49 pins connected to the Zynq PL for general purpose Digital I/O on the Arty Z7-20 and 26 on the Arty Z7-10. Due to the flexibility of FPGAs, it is possible to use these pins for just about anything including digital read/write, SPI connections, UART connections, I2C connections, and PWM. Six of these pins (labeled AN0-AN5) can also be used as single-ended analog inputs with an input range of 0V-3.3V, and another six (labeled AN6-11) can be used as differential analog inputs.

Pin Name	Shield Function
IO0-IO13	General purpose I/O pins
IO26-IO41, A (IO42)	Arty Z7-20 General purpose I/O pins

Figure 2: ChipKIT IO Analog Header on Arty Z7

A0-A5	Single-Ended Analog Input
A6-A11	Differential Analog Input

Figure 3: Digital I/O on Arty Z7

d) Debouncing Button:

- When we press a pushbutton or toggle switch or a micro switch, two metal parts come into contact to short the supply. But they do not connect instantly but the metal parts connect and disconnect several times before the actual stable connection is made. The same thing happens while releasing the button. Unpredictably, this results the **false triggering or multiple triggering** like the

button is pressed multiple times (Figure 4).

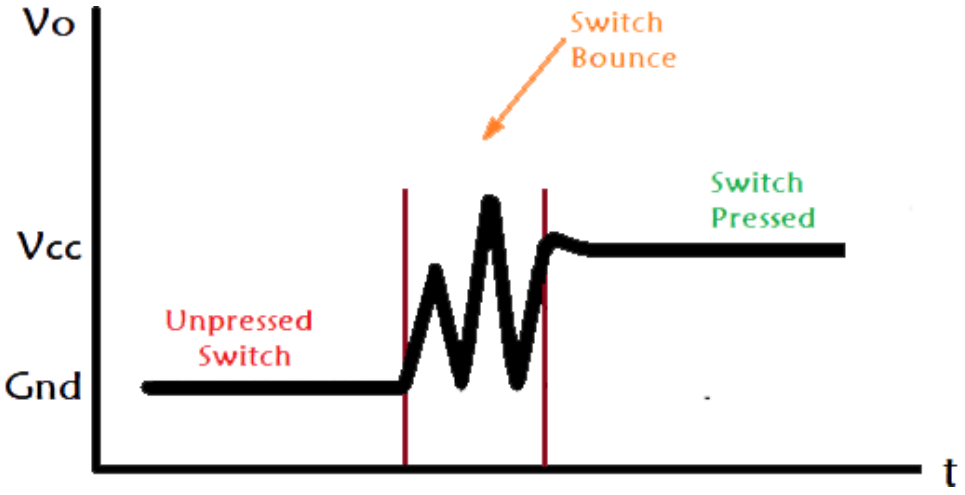


Figure 4: Switch bounce

- Debouncing-switch technic is used for clear all-natural bounce whenever pressing the button. There are many ways to perform this technic with different advantages and disadvantages, in hardware or even software. In this project, we implement a software module to convert the data collecting from a normal switch into "clear" data by connecting two D flip-flops and an AND operator (Figure 5).

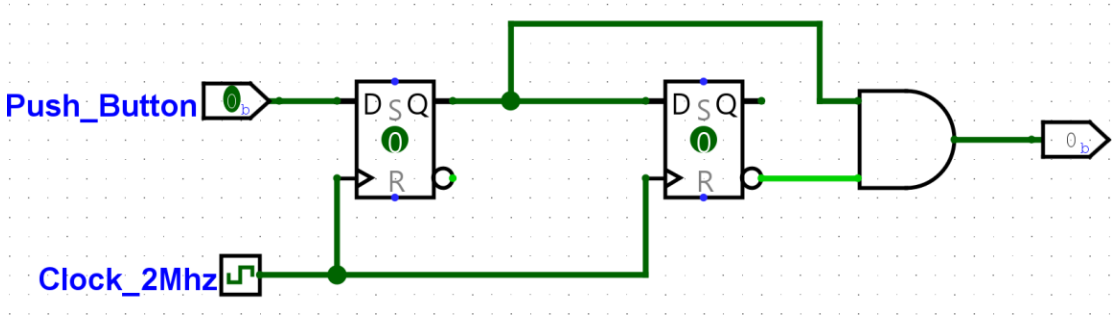


Figure 5: Debouncing Button Circuit

- After being processed by this circuit, the data becomes clear and synchronous with the clock. In addition, the activation duration (the duration of data in HIGH state after pushing the button) equal to a frequency of the clock (2Mhz).

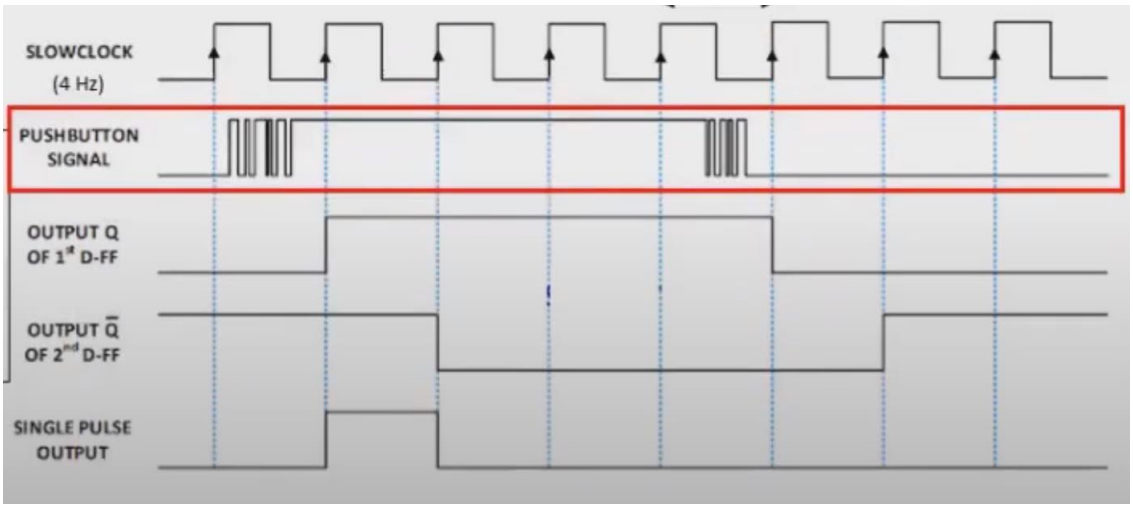


Figure 6: Time diagram of circuit

e) Finite State Machine:

- Finite state machines (FSMs) are at the heart of most digital design. The basic idea of an FSM is to store a sequence of different unique states and transition between them depending on the values of the inputs and the current state of the machine. The FSM can be of two types: Moore (where the output of the state machine is purely dependent on the state variables) and Mealy (where the output can depend on the current state variable values *and* the input values). The general structure of an FSM is shown in Figure 7.

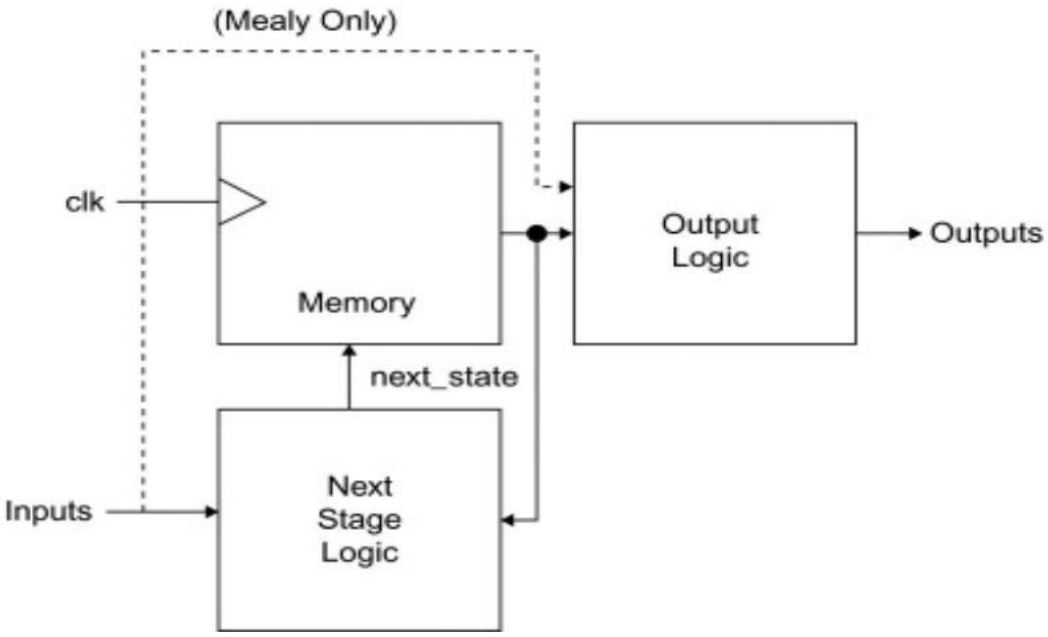


Figure 7: Time diagram of circuit

2) Applications

As its name, chess clock can be used for playing chess, but it can also be used for some other sports because of its counting-down function.

III. DESIGN

1) Apparatus

In this project, we need some hardware apart from ArtyZ7:

- Breadboards
- Resistors 220 Ohm
- 2-leg momentary tactile switches
- 4 digits 7 segments Led Board (each digit is from 0 to 9)
- On Arty z7:
  - ChipKit IO Analog Header
  - 1 switch
  - 2 buttons

2) Block Diagram

Besides aforementioned equipment, those are controlled by linking some essential software modules, which are compiled and performed automatically by the CPU inside Arty z7 FPGA. These software modules are arranged in order as below figure:

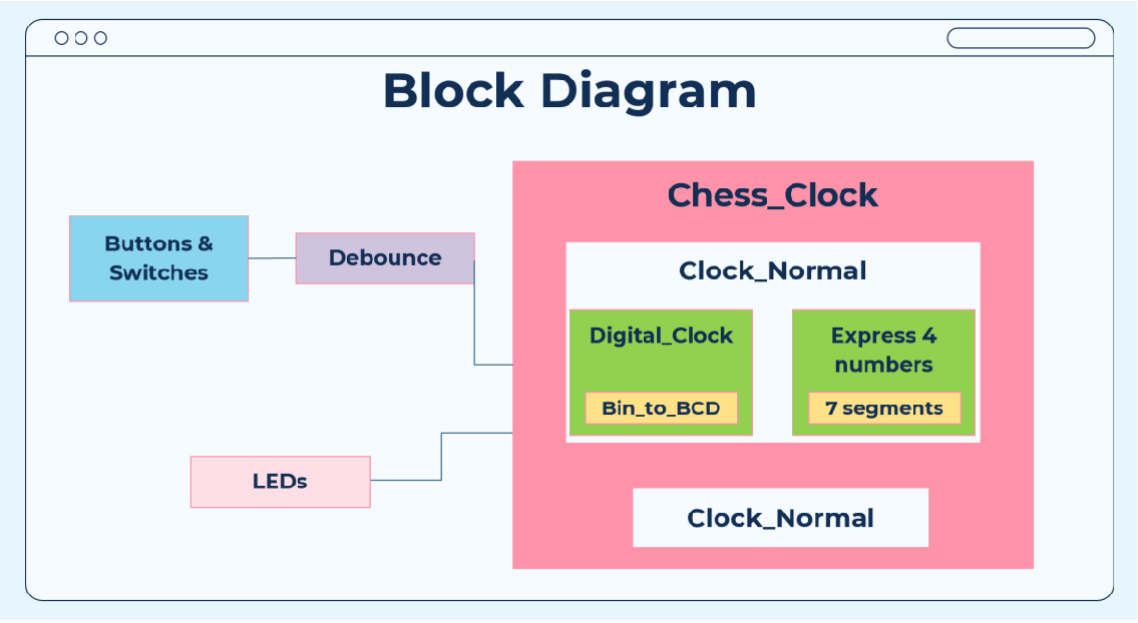


Figure 7: Block Diagram

Bin_to_BCD:

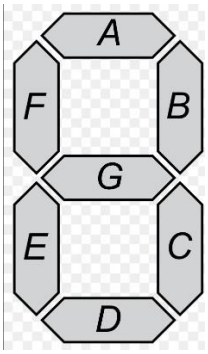
- There are many ways we can express a number in digital systems such as decimal, binary, hexadecimal, etc. BCD (Binary-coded decimal) is system of writing numerals that assigns a 4-digit

binary code to each digit 0 through 9 in decimal number. For example, 12 can be expressed as 1100 in binary, but 0001_0010 in BCD. This module converts a number in binary to BCD.

- Data in BCD is extremely powerful in this project when we have to express each number on 4-digit 7-segment Led Board. Every 2 digits number in decimal or binary cannot be shown on the Led Board. Converting these numbers to BCD format helps us to divide decimal numbers into many parts which has value from 0 to 9 and show it on Led Board.

7 segments:

- In order to express numbers from 0 to 9 on 7 segments Led, we have to set LOW (0V) to appropriate segments, which is marked from A to G as figure:



- This module take input as a 4 bits binary number, which has value from 0 to 9, and convert it to a 7 bits binary number, with each bit of output is corresponding to each segment in order (Segment A is corresponding to MSB bit of output, and G is corresponding LSB bit).
- For example:
    - input: 4'b0111 (7 in decimal) → output: 7'b0001111

Digital_Clock:

- A clock for playing chess have to functions as a normal clock as first, this module provides a clock by taking data from clock on Arty Z7, some buttons which function as minute/second increment, stop, etc. The outputs correspond to numerical values representing temporal units of seconds, minutes, and hours, which are used for "Express 4 numbers" module to perform on Led Board.

Express 4 numbers:

- On Led Board, we cannot present 4 different numbers on 4 digits simultaneously, so multiplexing technic is used in this project as a

way to perform 4 numbers nearly at a time. Although individual digits are still sequentially activated, the process occurs at a rate beyond the temporal resolution of the human visual system. Specifically, the digits are rapidly and iteratively switched on and off, exceeding the perception threshold of the observer.

- This module performs this technic exactly, with taking input as 4 numerical values to be displayed on LED, along with outputs are 4-bit number used for selecting which digit is activated, and 7-bit number operating as the value which is shown on the activated digit.

Clock_Normal:

- This module basically is the combination between “Express 4 numbers” and “Digital_Clock” module, functioning as a count-down clock. By taking input from buttons, switches and clock inside Arty Z7, the module will process the data and respond the corresponding time.

Chess_Clock:

- After having module of a normal clock, we make a chess clock by linked 2 normal clock and control by some buttons. Some functions such as switching between 2 clocks are performed in this module.

Debounce:

- The purpose of this module is to debounce a button press and establish a predetermined activation duration.

IV. IMPLEMENTATION

Firstly, we separate each task into smaller projects in the following diagram, which has already been shown previously. Using our former projects in the lab and theoretical class, we re-implement our clock (specifically exercise 2 in lab 2) and D Flip-Flop using edge detection in Verilog code.

```
module clock(
    input clk, //clock of arty z7
    output reg led
);
    integer count = 0;
    always @(posedge clk)
    begin
        count <= count + 1;
        if (count == 1000000) led <=1;
        else if (count == 2000000)
        begin
            led <= 0;
```

```
            count <=0;
        end
    end
endmodule

module dff(clkclk, d, q, qbar, rst);
//rising clock-edge dff with active high synchronous reset
  input clkclk, rst, d;
  output reg q;
  output qbar;
  assign qbar = ~q;
  always @(posedge clkclk)
    if (rst == 1'b1) q <= 1'b0;
    else q <= d;
endmodule
```

The reason why we choose the number 1000000 in the count to simulate 62.5Hz is just our estimate. If we choose the lower number, it may happen multiple push problems. However, if the number is too high, it would occur a delay (which happens to us)

After that, with some support on the Internet, we successfully recreate a debounce button for our clocks using structural model. All of the information about debounce button and schematic design of it has already explained above.

```
module debounce (
    input clk,
    input but,
    output led
);
    wire clkclk;
    clock slowclk(clk, clkclk);

    wire q1, q2o;
    dff name1(clkclk, but, q1, , );
    dff name2(clkclk, q1, , q2o, );

    assign led = q1 & q2o;
endmodule
```

Next, we turn our attention to code how to change binary numbers into BCD format.

```
module BintoBCD (
    input [11:0] binary,
    output reg [3:0] thou,
    output reg [3:0] hund,
    output reg [3:0] tens,
    output reg [3:0] ones
);
    reg [11:0] bcd_data = 0;
```

```
    always @(binary)
    begin
        bcd_data = binary;
        thou = bcd_data / 1000;
        bcd_data = bcd_data % 1000;

        hund = bcd_data / 100;
        bcd_data = bcd_data % 100;

        tens = bcd_data / 10;
        ones = bcd_data % 10;
    end
endmodule
```

This code is a converter from Binary numbers to BCD code. Because each unit digit can be represented from 0 to 9 so we need at least 4 bits (3 bits are not enough because it can only display from 0 to 2^3 -1=7). Because of that, the BCD data needs to store at least 12 bits for all of circumstances.

After that, we implement the digital clock basing on what we have done to convert binary numbers to BCD code

```
module digital_clock (
    input clk, // clock of arty z7
    input en, // assign it to a switch
    input rst,
    input on,
    input minup,
    input secup,
    input mindown,
    input secdown,
    output [3:0] s1,
    output [3:0] s2,
    output [3:0] m1,    // all these are what we show on the 7 segment
(0-9 : 4 bits)
    output [3:0] m2,
    output [3:0] h1,
    output [3:0] h2
);
    // h2 h1 : m2 m1
    reg [5:0] hour = 0, min = 0, sec = 0; // 60 : 6 bits
    integer clkc = 0;
    always @(posedge clk)
    begin
        if (on == 1'b1)
        begin
            //reset clock
            if (rst==1'b1) {hour, min, sec} <= 0;
            //set clock
            else if (minup == 1)
                if (min == 59) min <= 0;
```

```
                else min <= min + 1;
            else if (secup == 1)
                if (sec == 59) sec <= 0;
                else sec <= sec + 1;
            else if (secdown == 1)
                if (sec == 0) sec <=59;
                else sec <= sec - 1;
            else if (mindown == 1)
                if (min == 0) min <=59;
                else min <= min - 1;
        end
        // count
        if (en == 1)
            if (clkc == 125000000)
            begin
                clkc  <=  0;
                if (sec == 0)
                begin
                    if (min == 0)
                    begin
                        if (hour == 0)
                        begin
                            {hour, min, sec} <= 0;
                        end
                        else
                        begin
                            hour <= hour - 1;
                            min <= 59;
                            sec <= 59;
                        end
                    end
                    else
                    begin
                        min <= min - 1;
                        sec <= 59;
                    end
                end
                else
                    sec <= sec - 1;
            end
            else clkc <= clkc + 1;
    end
    BintoBCD secs(.binary(sec), .thou(), .hund(), .tens(s2), .ones(s1));
    BintoBCD mins(.binary(min), .thou(), .hund(), .tens(m2), .ones(m1));
    BintoBCD hours(.binary(hour), .thou(), .hund(), .tens(h2),
.ones(h1));
endmodule
```

The code explains how we change the real-time into binary numbers, then binary to BCD code. Our code has covered all the cases in that the daily clock

will happen when change (for example, we decrease from 1 hour to 59 minutes and 59 seconds). There are two main functions that our chess clock has: set clock and count down. In setting up, we need to look after the increase and all of the cases that would happen when a clock rises. It is the same problem with counting down, but this needs to cooperate with the initial clock count down in Arty Z7 (the frequency is 125 MHz).

Next, we are looking forward to coding how to display the LED number on the four seven-segment LEDs. We divide the tasks into smaller tasks: coding shows each seven segment LED and then expressing four numbers together. Luckily, we have already created seven segments LED before in the LAB class (Exercise 3 in LAB 2)

```
module Seven_segments (
    input [3:0] inp,
//    input select,
    output reg [6:0] out
);
    always @(inp)
    begin
        case (inp)
            0: out <= 7'b0000001;
            1: out <= 7'b1001111;
            2: out <= 7'b0010010;
            3: out <= 7'b0000110;
            4: out <= 7'b1001100;
            5: out <= 7'b0100100;
            6: out <= 7'b0100000;
            7: out <= 7'b0001111;
            8: out <= 7'b0000000;
            9: out <= 7'b0000100;
            default: out <= 7'b1111111;
        endcase
    end
endmodule
```

The only noticeable caution of this is that we need to remember this is an active low device.

Next, we are heading to code expressing 4 numbers:

```
module express_4nums (
    input clk,
    input clr,
    input [3:0] in1,
    input [3:0] in2,
    input [3:0] in3,
    input  [3:0]  in4,
    output reg [6:0] seg,
    output reg [3:0] select
);
    wire [6:0] seg1, seg2, seg3, seg4;
    localparam N = 13;
```

```
    localparam LEFT = 2'b00, MIDLEFT = 2'b01, MIDRIGHT = 2'b10, RIGHT =
2'b11;
    reg [N-1:0] count; // the 13-bit counter which can allow us to
multiplex at estimated 15MHz
    always @(posedge clk or posedge clr)
    begin
        if (clr) count <= 0;
        else count = count + 1;
    end

    always @(*)
    begin
        case(count[N-1:N-2]) // using only 2 MSB bits of the counter
            LEFT:
            begin
                seg = seg1;
                select = 4'b1000;
            end
            MIDLEFT:
            begin
                seg = seg2;
                select = 4'b0100;
            end
            MIDRIGHT:
            begin
                seg = seg3;
                select = 4'b0010;
            end
            RIGHT:
            begin
                seg = seg4;
                select = 4'b0001;
            end
        endcase
    end

    Seven_segments dis1(in1, seg1);
    Seven_segments dis2(in2, seg2);
    Seven_segments dis3(in3, seg3);
    Seven_segments dis4(in4, seg4);

endmodule
```

In this task, we need to search the Internet and decide to use the localparam line. Localparam prevents the values to be overwritten (directly) from outside the module, or simply a const syntax in Verilog code. We choose a count is nearly 2^{13}-1=8191 to show the LED's scan speed. If we change lower, it may occur inappropriate to display our seven-segment LEDs. But if we change it higher, the delay would happen.

We have all of the necessary functions of a chess clock so this is where we design a normal clock

```
  module Clock_Normal (
    input clk, //clock of arty z7
    input sw, //switch[0] to enable the clock
    input on,
    input btnC, //reset the clock
    input btnU, //min increment
    input btnR, //sec increment
    input btnUd, //hour decrement
    input btnRd, //min decrement
    output [6:0] seg,
    output [3:0] select
);
    wire [3:0] s1, s2, m1, m2, h1, h2;
    reg secup, minup, secdown, mindown;
    wire btnCclr, btnUclr, btnRclr, btnUdclr, btnRdclr;
    reg btnCclr_prev, btnUclr_prev, btnRclr_prev, btnUdclr_prev,
btnRdclr_prev;

    // debounce the button
    debounce dbC(clk, btnC, btnCclr); //clear
    debounce dbU(clk, btnU, btnUclr); //min up
    debounce dbR(clk, btnR, btnRclr); //second up
    debounce dbUd(clk, btnUd, btnUdclr); //min up
    debounce dbRd(clk, btnRd, btnRdclr); //seccond up

    //instantiate seven segments driver
    express_4nums express(clk, 1'b0, m2, m1, s2, s1, seg, select);
    digital_clock dig_clk(clk, sw, btnCclr, on, minup, secup, mindown,
secdown, s1, s2, m1, m2, h1, h2);

    always @(posedge clk)
    begin
        btnUclr_prev <= btnUclr;
        btnRclr_prev <= btnRclr;
        btnUdclr_prev <= btnUdclr;
        btnRdclr_prev <= btnRdclr;
    end
    always @(btnUclr, btnRclr, btnUdclr, btnRdclr)
    begin
         if (btnUdclr_prev == 1'b0 && btnUdclr == 1'b1) mindown = 1'b1;
else mindown = 1'b0;
         if (btnRdclr_prev == 1'b0 && btnRdclr == 1'b1) secdown = 1'b1;
else secdown = 1'b0;
         if (btnUclr_prev == 1'b0 && btnUclr == 1'b1) minup = 1'b1; else
minup = 1'b0;
         if (btnRclr_prev == 1'b0 && btnRclr == 1'b1) secup = 1'b1; else
secup = 1'b0;
```

```
    end
endmodule
```

Initially, we need to debounce all the buttons we are not using on the chip to be error-free as much as possible. Then we instantiate them into express them on 4 number and give them functions to work appropriately by accessing them through digital clock module. The hardest part of this normal clock is to code the button functions correctly by overcoming some situations such as multiple pushes or continuously counting down without pressing. Therefore, we design a previous push imaginary input to solve these problems. In those loops, there are two things we need to discuss previously and conditions to setup. First, we declare four previous buttons simultaneously to mimic the chance that when two buttons press immediately, it would occur error which one should go first. Then, we set the conditions for our clock when it goes up or drops. We add conditions for the previous button to be in 1'b0 so that we can withdraw the holding button issues.

Finally, we code for the chess clocks by using 2 normal clocks and some buttons functioning the switch to each player.

```
module chess_clock (
    input clk, // clock of arty z7
    input btn2, // enable clock 1, unable clock 2
    input btn1, // enable clock 2, unable clock 1
    input stopp,
    input change, // select another clock to set up
    input btnC, //reset the clock
    input btnU, //min increment
    input btnR, //sec increment
    input btnUd, //min decrement
    input btnRd, //sec decrement
    output [6:0] seg, segO,
    output [3:0] select, selectO,
    output reg in_select1, in_select2
);

    // reg [7:0] led, ledO;
    reg en1 = 0, en2 = 0, on1 = 1, on2 = 0, sel = 0, change_clr_pre;
    wire change_clr;

    debounce chg(clk, change, change_clr);
    always @(posedge clk)
    begin
        change_clr_pre <= change_clr;
        if (change_clr_pre == 0 && change_clr == 1) sel <=~sel;
    end
    always @(posedge clk)
    begin
        if (sel == 0)
        begin
            on1 <= 0;
```

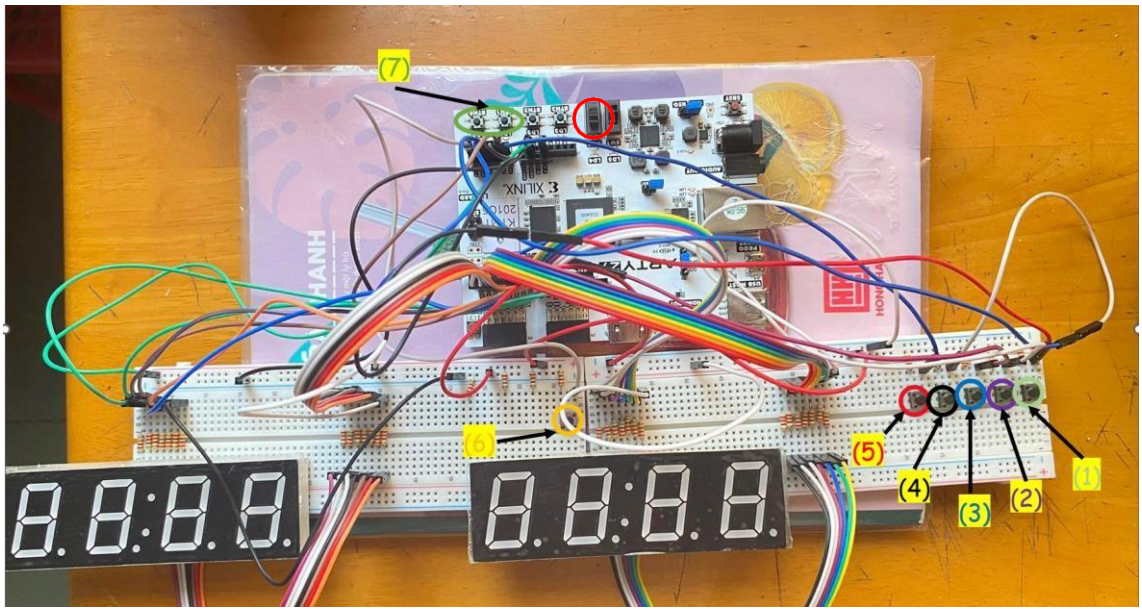
```
            on2 <= 1;
            if (stopp == 0)
            begin
                in_select1 <= 1; in_select2 <= 1;
            end
            else
            begin
                in_select1 <= 1; in_select2 <= 0;
            end
        end
        else
        begin
            on1 <= 1;
            on2 <= 0;
            if (stopp == 0)
            begin
                in_select1 <= 1; in_select2 <= 1;
            end
            else
            begin
                in_select1 <= 0; in_select2 <= 1;
            end
        end

        if (stopp == 1)
        begin
            en1 <= 1'b0;
            en2 <= 1'b0;
        end
        else
        begin
            if (btn1 == 1'b1)
            begin
                en1 <= 1'b0;
                en2 <= 1'b1;
            end
            if (btn2 == 1'b1)
            begin
                en1 <= 1'b1;
                en2 <= 1'b0;
            end
        end
    end
    Clock_Normal clock1(clk, en1, on1, btnC, btnU, btnR, btnUd, btnRd,
seg, select);
    Clock_Normal clock2(clk, en2, on2, btnC, btnU, btnR, btnUd, btnRd,
seg0, select0);

endmodule
```

We make two buttons to function the properties of switching clock when the move is played by each chess player. The sel imaginary input is applying the final state machine knowledge. It will express when which clock happened or not.

V. RESULTS



Totally, there are 8 buttons and 1 switch used for setting up and controlling the chess clock:

- (1): Increase the second of the clock (which is being selected) by 1 unit.
- (2): Decrease the second of the clock (which is being selected) by 1unit.
- (3): Increase the time of the clock (which is being selected) by 1 unit.
- (4): Decrease minute of the clock (which is being selected) by 1 unit.
- (5): Reset the clock (which is being selected) (Set the time of 00:00).
- (6): Select between 2 clocks to set its time.
- (7): These 2 buttons are for 2 players playing chess (button 1 for player 1, and button 2 for player 2). When player 1 pushes the button, his clock will stop and the opponent's clock will run, and vice versa.
- Switch 0 (Sw0) on Arty z7: Turn it on to stop both clocks, and set it off to play the game.

VI. CONCLUSION

1) Working achievements

We have successfully rebuilt a chess clock on Arty Z7 with some functions like counting down, setting the input clock, switching between players or showing the output on the clocks.

2) Advantages

Our chess clock can simulate some basic functions of the chess clocks and we learn a lot during this project like learning more about debounce technique or digging deep down into Verilog code.

3) Disadvantages

Unfortunately, our clock cannot be used in daily matches between chess players. Firstly, it is too difficult for the players to see each other's timing because our setup is quite awkward. Moreover, it is too expensive to use an Arty Z7 chip with just only one chess clock. Besides, it is nearly impossible for the players to press the buttons while playing. The positions of these buttons are hard to reach and too small that it can cause some timing problems for them, leading to underperforming of their chess games.