CENG3430 Project Report

The Chinese University of Hong Kong

May 2019

Catch the Thief on FPGA

members:
Mo Yu Hin (SID:1155116319)

II. Content Page

III. Summary

This report document is the development of a classic board game - **Hare & Hounds**. The project is implemented based on the Xilinx Digilent ZedBoard Zynq-7000 ARM/FPGA Soc Development Board and a VGA monitor. This report covers the system requirements, specifications and testing methodologies of our application in detail, including the problems encountered and how to tackle them.

IV. Introduction

The concept of the Hare & Hounds originated in 19th century France and became popular with French military officers during the Franco-Prussian War of 1870-1871. Later in 1963, an article by Martin Gardner in the journal Scientific American generated further interest in the game in 1963, and it has become popular with computer programmers due to the ease of implementing its simple rules. Last year, a very famous game development company called Nintendo published a game call - Clubhouse Games: 51 Worldwide Classics and made Hare & Hounds become popular to the new generation gamers.

During the design stage, I used to make hare and hounds to become the chess logo. However, it is very complicated to develop such a user interface with VHDL. In order to strengthen the feeling of opposition between the two sides, I chose to use blue and red but not the representative color of hare and hounds, which is white and brown. To maintain the characteristics of the chess pieces, I changed the name of the game to **Catch the Thief**, as blue is the representative color of cops and red represents danger, which is more fit to the user interface of the project.

This project aimed to implement this game using a field- programmable gate array (FPGA). An FPGA consists of logic blocks which are programmable, reconfigurable interconnects and input/output pads2. The logic blocks used in an FPGA could consist of memory elements such as flip- flops or memory blocks. The logic blocks are capable of performing simple to complex computational functions. An FPGA can support several thousand gates and can be reprogrammed, unlike application- specific integrated circuits which are manufactured for specific tasks. This feature helps in tailoring the capabilities of microprocessors to meet specific individual needs and designing specialized integrated circuits. FPGAs have the advantage of a more predictable life cycle due to removal of wafer capabilities, potential respins, faster time to market compared to other options and simple design cycle. They are used in a wide range of applications, and in markets such as aerospace, defense, data centers, automotive, medical and wireless communications.

V. Requirements and Design Specifications

1. Functional Requirements
    a. State changing

    Develop a final state machine to control the state of the system. In the FSM - Phase Control, there are 7 phases to complete the whole system → (p0, chk1, p1, p2, chk2, p3, p4).

    Phase 0: The initial state of the whole system which would help the system initial all the counters and pass to the phase check 1.

    Phase check 1: there are 3 parts → the first part will initiate the counter need to use for the blue player turn, second part is a function to check the winning condition of the red team, and the third part is a function for checking the winning condition of the blue team.

    Phase check 2: same as Phase check 1, initiate the counter need to use for the red player instead.

    Phase 5: It is the state when the red team has won. By pressing the btnC button return the system to Phase 0 to initiate the system and start the game.

    Phase 6: It is the state when the blue team has won. All the functions are the same as Phase 5 but the output is different.

    b. node movement control

    Phase 1: This phase has two parts → the first part will check if the node belongs to the blue team and decide which node is the default node for moving, and the second part is helping the player to select the node the want to move by btnU and btnD button.

    Phase 2: This phase has three parts → the first part will generate an array with all elements has initial value = 11 and fill in all the possible nodes for move's id into the array list, second part is a for loop for checking the array, if any element of the array is not equal to 11, the function will set that node as the default node will move to and break the loop. If all the elements in the array equal to 11, the function returns a counter and draws the state back to phase 1. The third part is a function to help the player to choose the node they want to

move to by pressing the button according to the direction from the node chosen in Phase 1.

Phase 3: Function similar to Phase 1, as the red team only has one node, this phase does not require a select node function.

Phase 4: Function and structure similar to Phase2, there are also 3 parts. The only difference is that Phase 4 does need a invalid select check as it has already checked in the Phase check 2.

c. 7 segment display countdown

The countdown required two check rates. A fast 100hz clock rate controls the SSD pmod to display both digits with two signals → 1. Single digit and 2. Ten digit. A slow 4hz clock rate is to control the count down value from the phase change. The reason for adding a 4hz clock for this system is to prevent the bounce back from the button input.

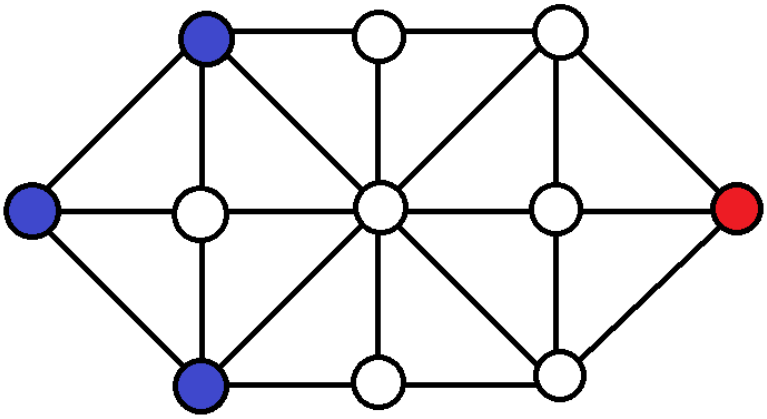
d. VGA screen output

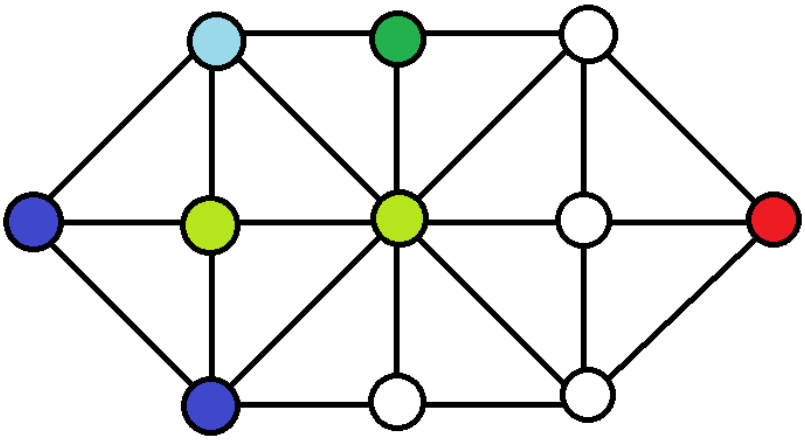
A TFT monitor was used to display the game. A TFT monitor was used as it is one of the most widely used display monitors and which was most readily available to us. This was controlled by a three- row 15- pin DE- 15 VGA connector. A VGA cable was used to connect the display to the 12- bit colour VGA port on the board (pins: VGA- R1 to VGA- VS).

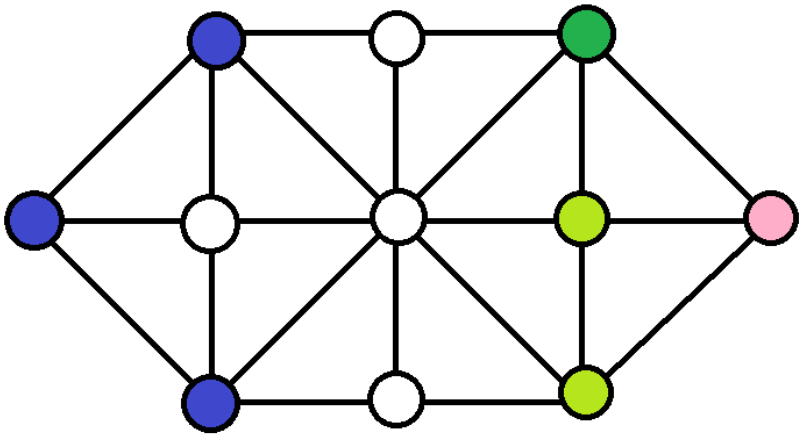
e. Pmod SSD

The Digilent Pmod SSD (Revision A) is a 2 digit seven- segment display commonly used to display a counter or timer. We used a 3.3V logic power supply for the Pmod and the signals will conform to LVCMOS 3.3V or LVTTL 3.3V logic conventions. The Pmod is connected to the board using the JA1 - JA4, JB1 – JB4 GND and UCC ports

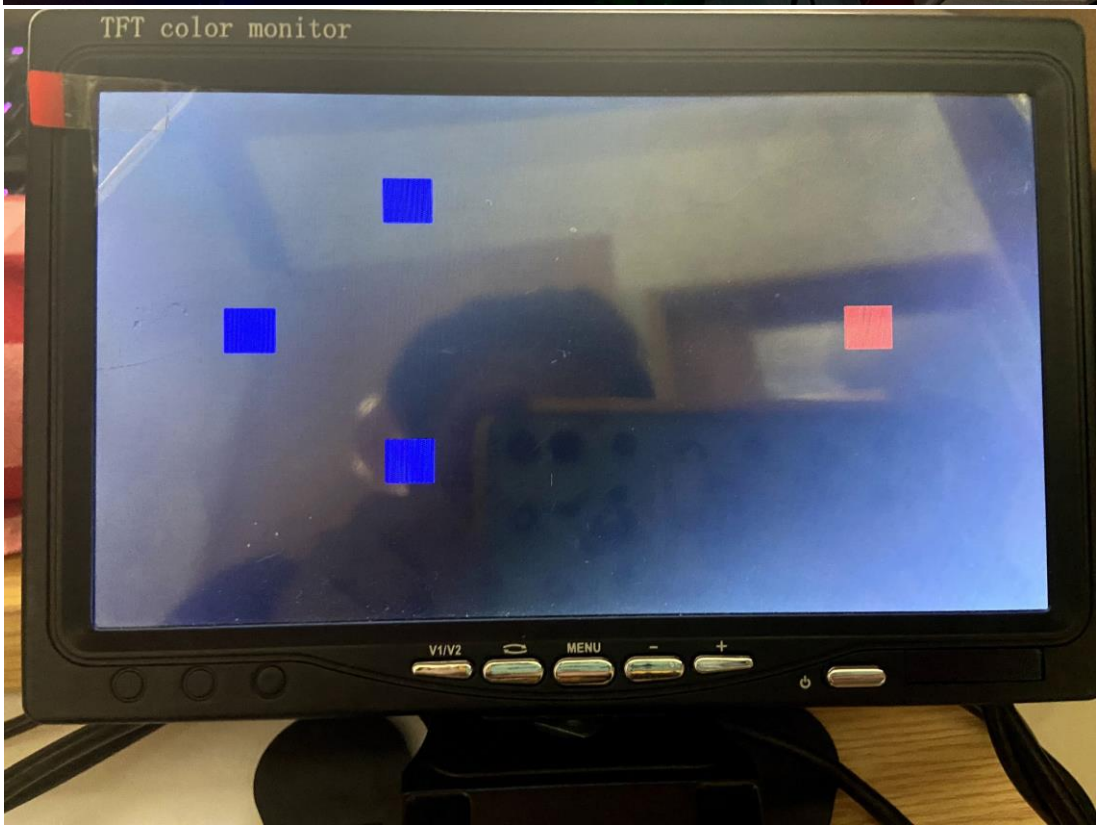
2. Non-Functioning Requirements
    a. User Interface (UI) Design

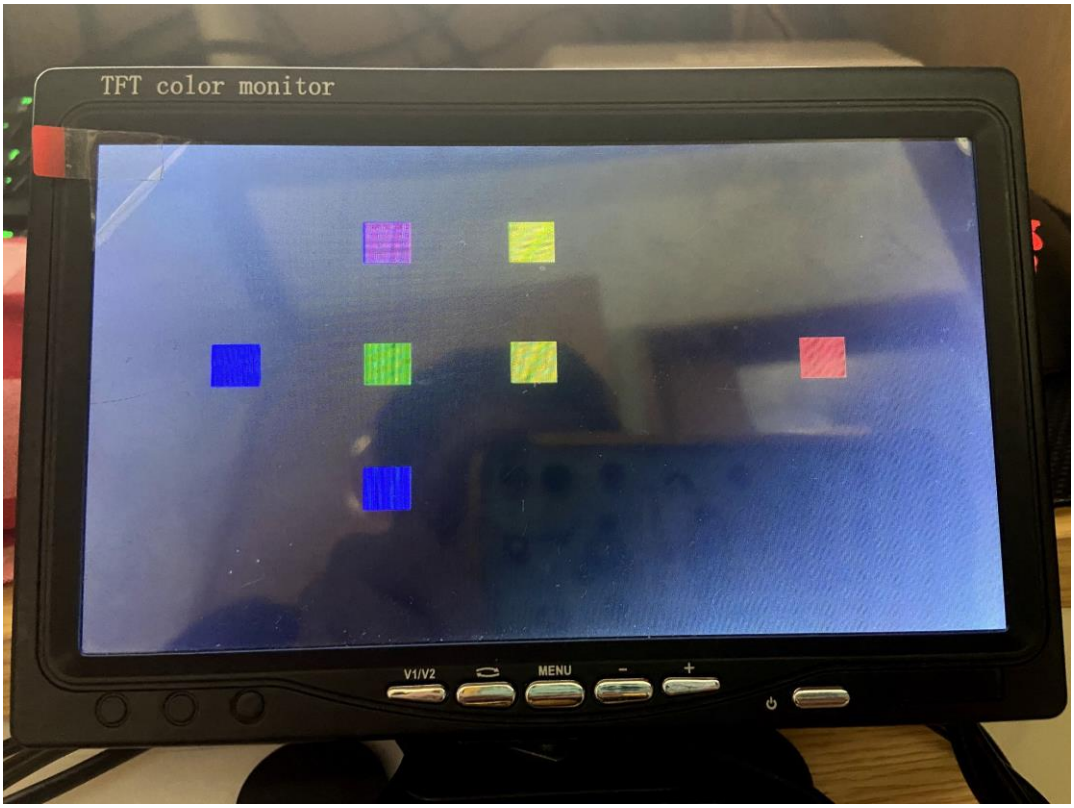


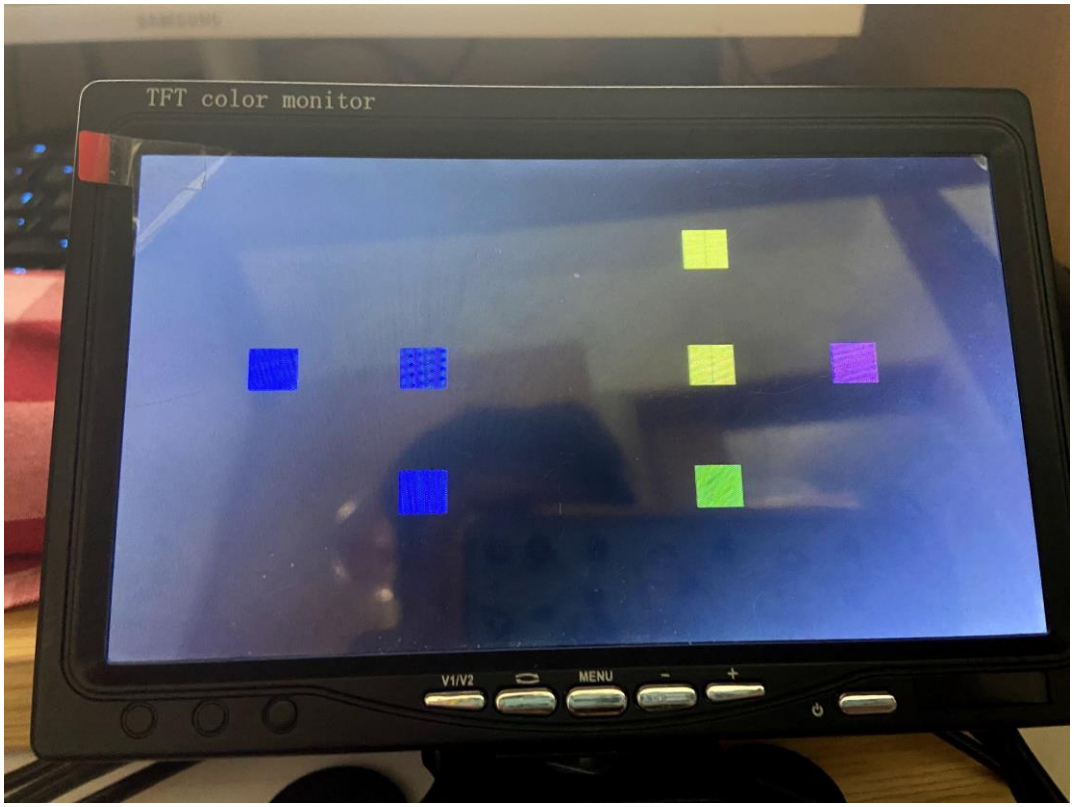




b. User Interface have Implemented







The reason why the user interface is downgraded will be explained in the “Challenge” part.

VI. Evaluation

1. Challenge and Problem

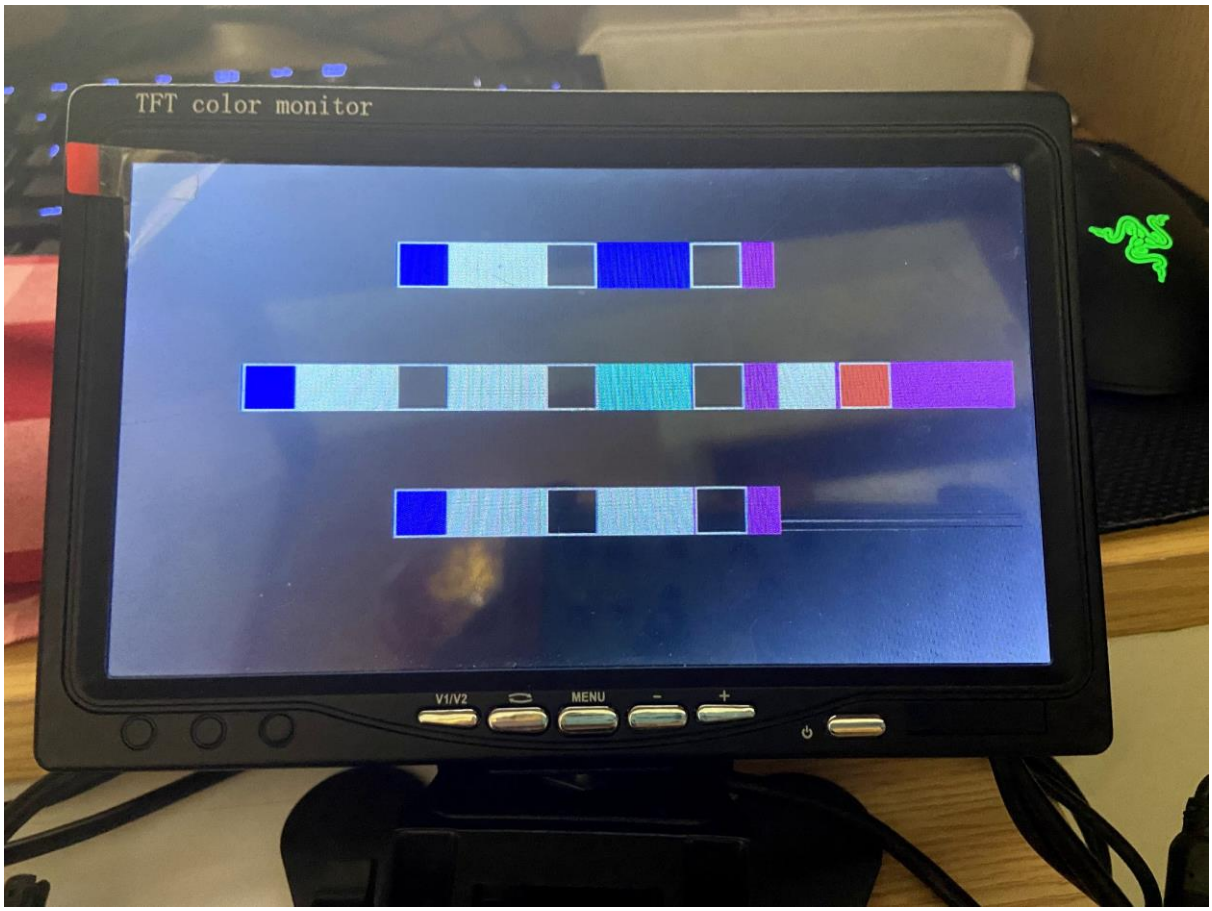
    a. expectation

    The Design of this project required 3 main regions of function. The first part is the countdown phase check of the system which helps the game run the game in steps. The second part is the player vs player mode which provides for two people playing together. And the third part is a player vs computer mode which provides computer opponents with a simple strategy for the player against.

    The first two parts are completely implemented to the device but the third part fails to develop.

    b. User Interface is hard to develop

    When I try to implement some user interface code, the screen somehow crashes out. In order to make the screen clean for the player, I only implemented the most simple interface for running the game.



code:

1st method try: develop a outer border by reate a bigger block

```
if (hcount >= 449 and hcount <= 1150) then
 -- node 0
if (hcount >= 450 and hcount < 509) then
if ( (hcount >= 451 and hcount <= 507) and (vcount >= 272 and vcount <= 328))then
   if ( (hcount >= 454 and hcount <= 504) and (vcount >= 275 and vcount <= 325))then
   if n0 = '1' then
   if (b1 = 0 or b2 = 0 or b3 = 0) and not (bwm = 0) then
      blue <= "1111";
      red <= "0000";
      green <= "0000";
   end if; -- end of b_team count
   if r = 0 and not (rwm = 0) then
      red <= "1111";
      blue<= "0000";
      green <= "0000";
   end if; --end of r_team count
   if bwm = 0 or rwm = 0 then
      red <= "1111";
      blue <= "1111";
      green <= "0000";
   end if; -- end of will move node color
   else
   if bwt = 0 or rwt = 0 then
      green <= "1111";
      red <= "0000";
      blue<= "0000";
   else -- end of will move to nodes color
   if (a(1)=0 or a(2)=0 or a(3)=0 or a(4)=0 or a(5)=0 or a(6)=0 or a(7)=0 or a(8)=0)
   or (c(1)=0 or c(2)=0 or c(3)=0 or c(4)=0 or c(5)=0 or c(6)=0 or c(7)=0 or c(8)=0)
       then
  red <= "1111";
  blue<= "0000";
  green <= "1111";
  else
  red <= "0000";
  blue<= "0000";
  green <= "0000";
  end if; -- end of able nodd color
  end if; -- end of will move to nodes color
  end if; -- end of n2 color state
  else
  red <= "1111";
  blue<= "1111";
  green <= "1111";
  end if; -- end of box content
end if;
```

2nd method try : develop the border one by one

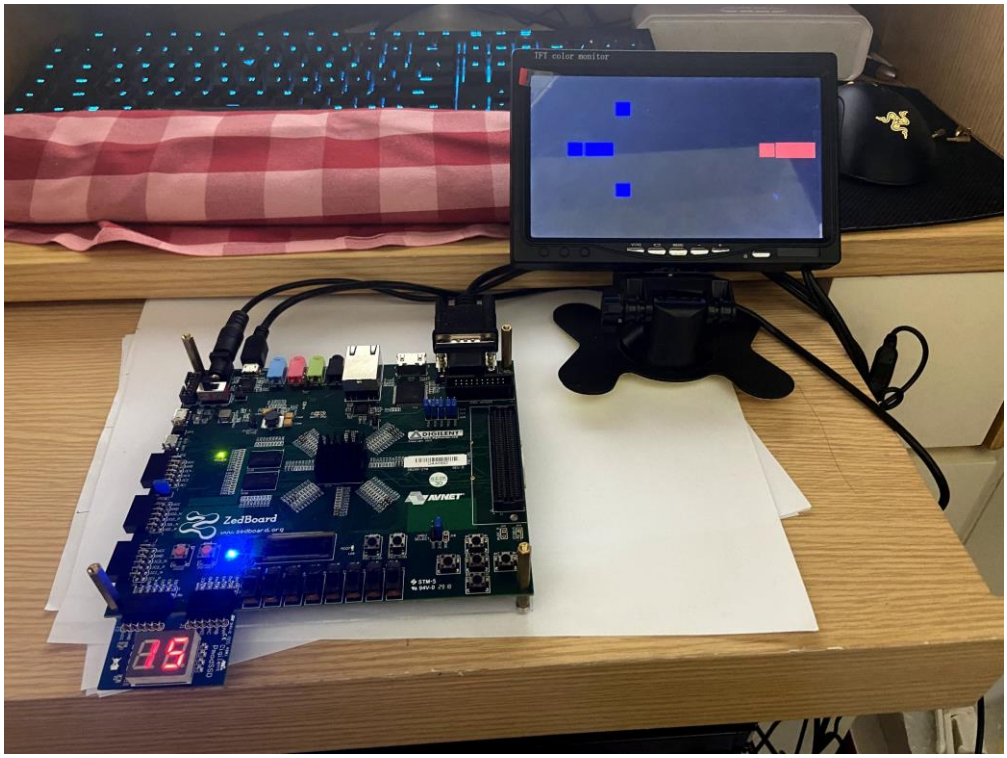
```
data_output_proc: process(hcount, vcount)
begin
    if( (hcount>=H_START and hcount<H_END) and (vcount >= V_START and vcount<V_TOTAL) ) then

        -- node 0
           -- node 0 border
               -- h.border
               if ( (hcount >= 431 and hcount <= 489) and (vcount >= 327 and vcount <= 329)) then
                   red <= "1111";
                   blue <= "1111";
                   green <= "1111";
                end if;
               if ( (hcount >= 431 and hcount <= 489) and (vcount >= 271 and vcount <= 273)) then
                    red <= "1111";
                    blue <= "1111";
                    green <= "1111";
                end if;
                -- v.count
                if ( (hcount >= 431 and hcount <= 433) and (vcount >= 274 and vcount <= 326)) then
                     red <= "1111";
                     blue <= "1111";
                     green <= "1111";
                end if;
                if ( (hcount >= 487 and hcount <= 489) and (vcount >= 274 and vcount <= 326)) then
                     red <= "1111";
                     blue <= "1111";
                     green <= "1111";
                end if;
```

Both of the above methods still cause a color crash.

c. Bug occurs that doesn't understand the reason.
While I am testing and implementing different code to the machine, sometimes some bugs occur for unknown reasons. Such as when I want to take my demo video and want to set the countdown ssd initial display to 15 turns for a faster game process. With only changing the value of the SSD ten digit, and result of the user interface crash. However, those crash disappear after I reset the value to 35.

Original:

Turn to 15:



d. Difficult to develop a good computer opponent with my knowledge
As I don't have experience and knowledge to develop AI. I can only try to develop a hard coded computer opponent for the game. I first think that developing a blue team computer player is easy as the strategy of the blue team is much simpler than the red team. But I found it is very hard to

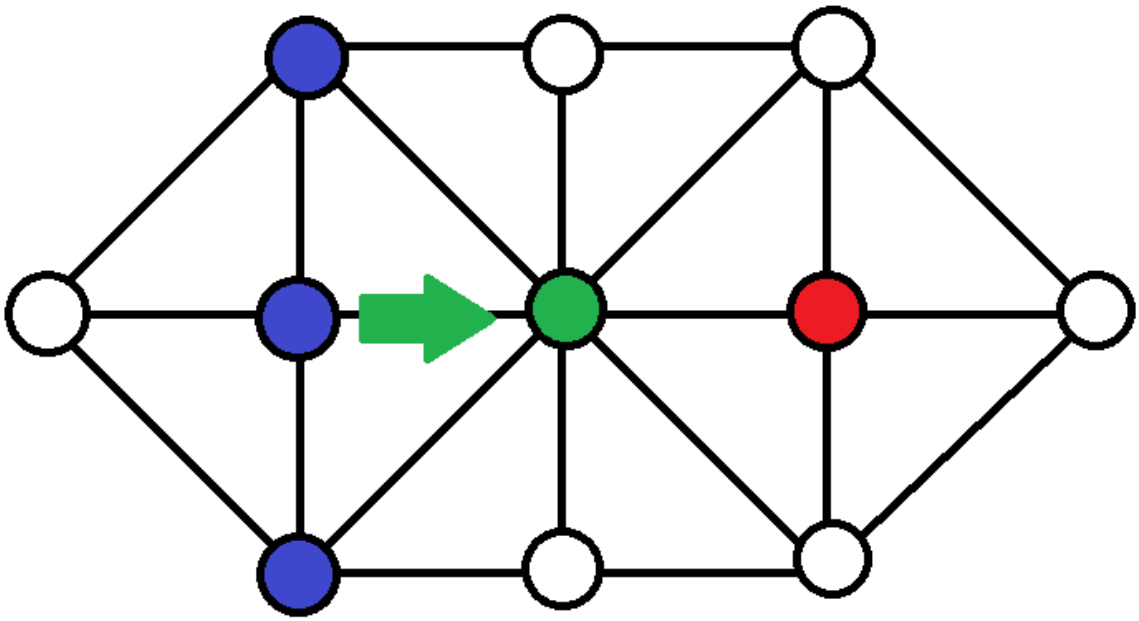
implement a balanced CPU for the game. If I hard-coded for every possible move, the real player is very hard to win the game without a leveling select. But If I just develop a simple CPU with random moves, the game becomes too simple for the player which may end the game in two moves.

VII. Extra Code Have Developed but not Implement to the System

a. Computer Component

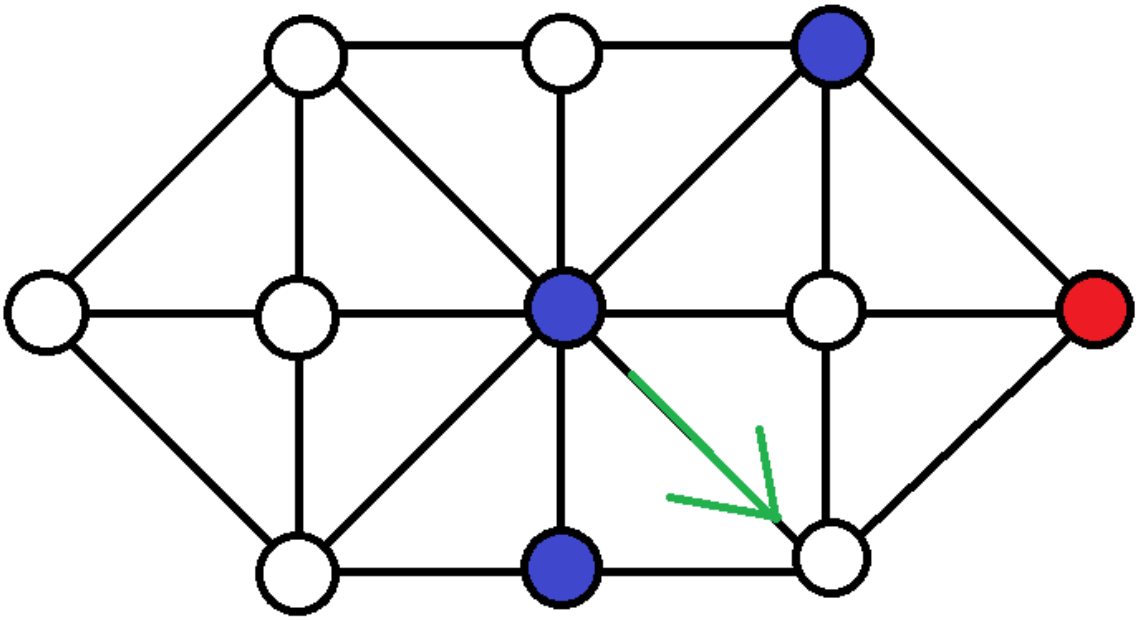
I have some math strategies to think by myself and have finished developing. However, I gave up on adding PVE mode to the system. So these codes are not implemented.

1. 1st move keep blue team on pattern



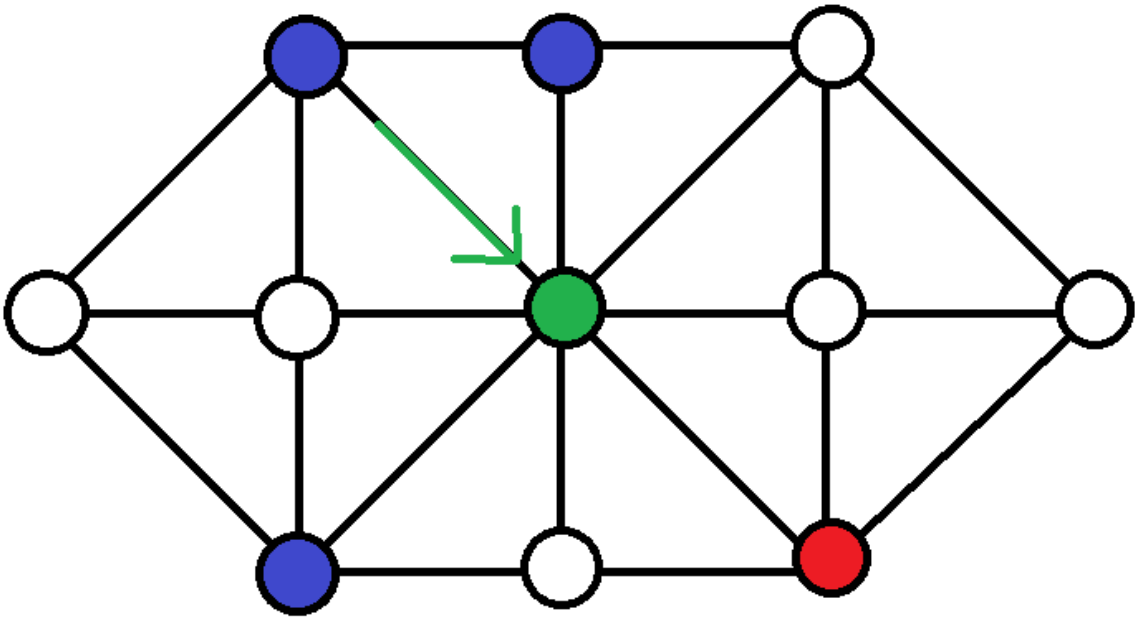
```
if n1 = '1' and n2 = '1' and n3 = '1' then
    if r = 8 then
       if b1 = 2 then
          b1 <= 5;
       else
        if b2 = 2 then
           b2 <= 5;
        else
            if b3 = 2 then
               b3 <= 2;
            end if;
        end if;
    end if;
    end if;
end if;
```

2. 2nd move keep blue team on pattern



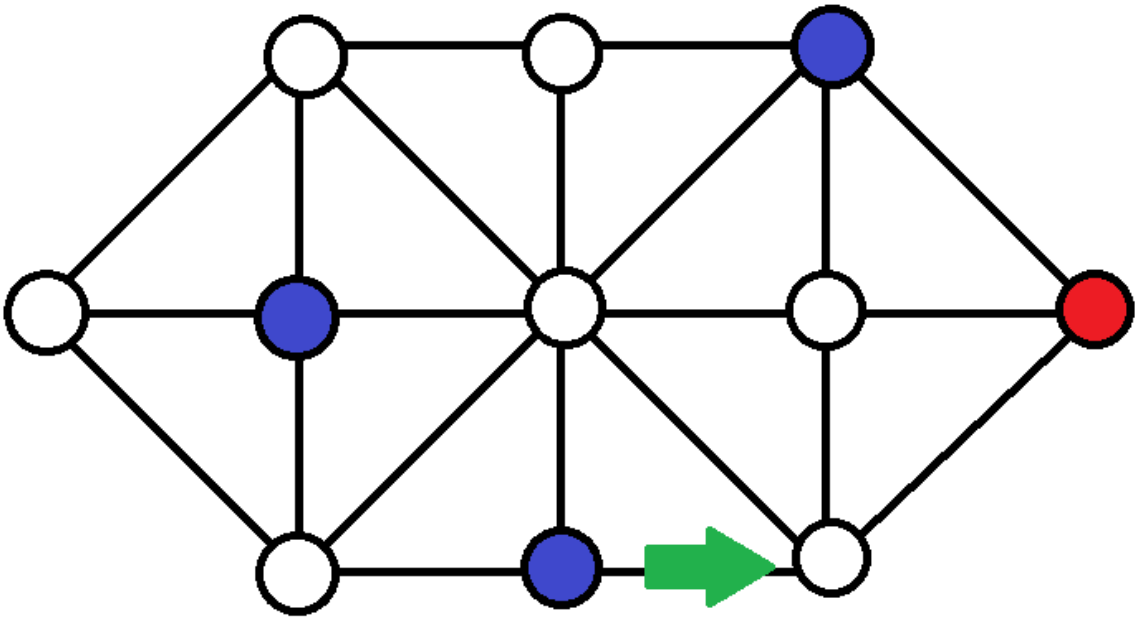
```
if n7 = '1' and n5 = '1' and n6 = '1' then
    if r = 10 then
       if b1 = 5 then
        b1 <= 9;
        else
        if b2 = 5 then
            b2 <= 9;
            else
            if b3 = 5 then
                b3 <= 9;
            end if;
        end if;
        end if;
    end if;
end if;
```

3. 3rd move keep blue team on pattern



```
if n1 = '1' and n3 = '1' and n4 = '1' then
if r = 9 then
     if b1 = 1 then
     b1 <= 5;
     else
     if b2 = 1 then
         b2 <= 5;
         else
         if b3 = 1 then
             b3 <= 5;
         end if;
     end if;
     end if;
    end if;
end if;
```

4. The Winning Move



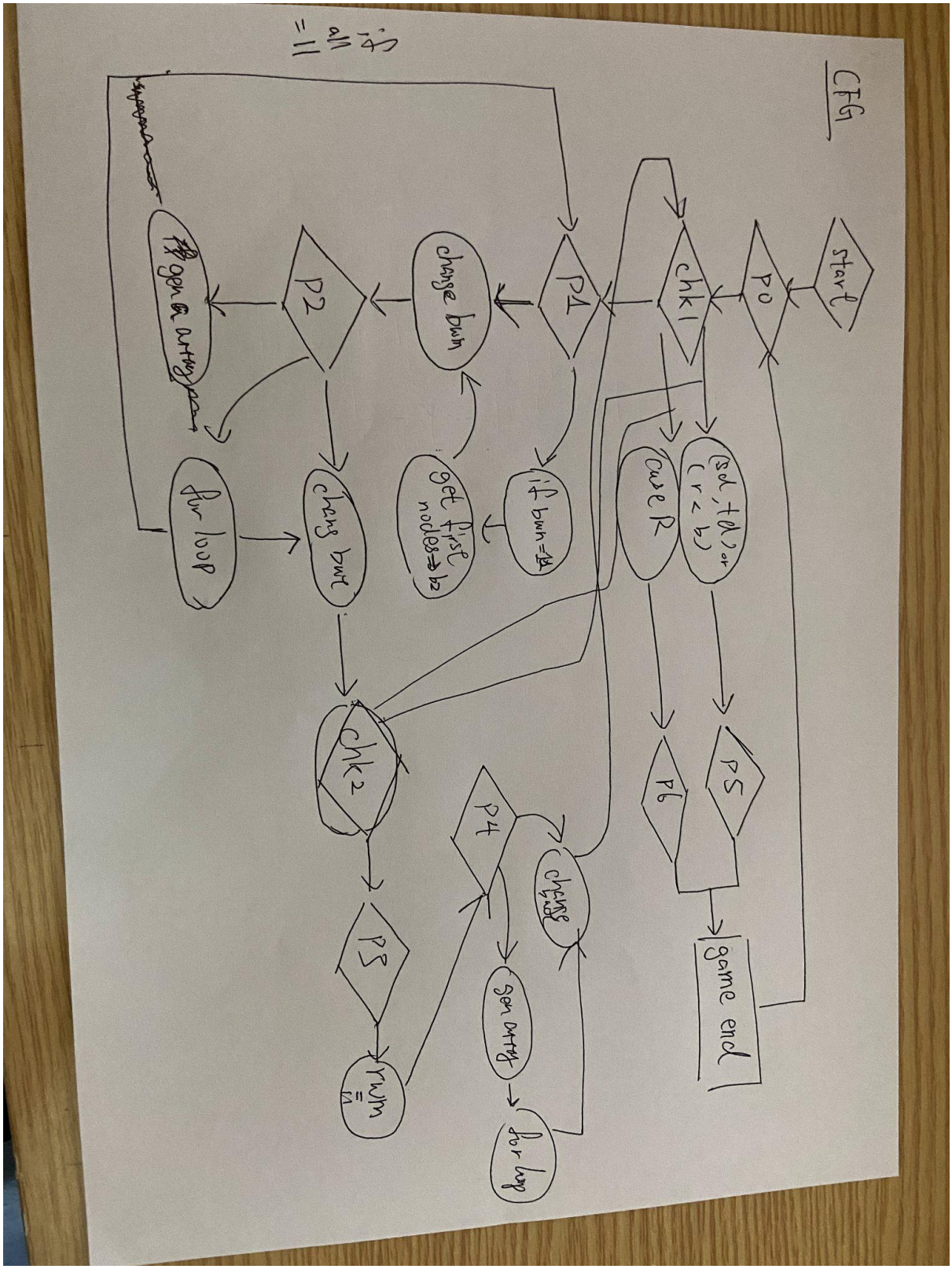
```
if n7 = '1' and n2 = '1' and n6 = '1' then
    if r = 10 then
     if b1 = 6 then
     b1 <= 9;
     else
     if b2 = 6 then
         b2 <= 9;
         else
         if b3 = 6 then
             b3 <= 9;
         end if;
     end if;
     end if;
    end if;
end if;
```

VIII. Conclusion

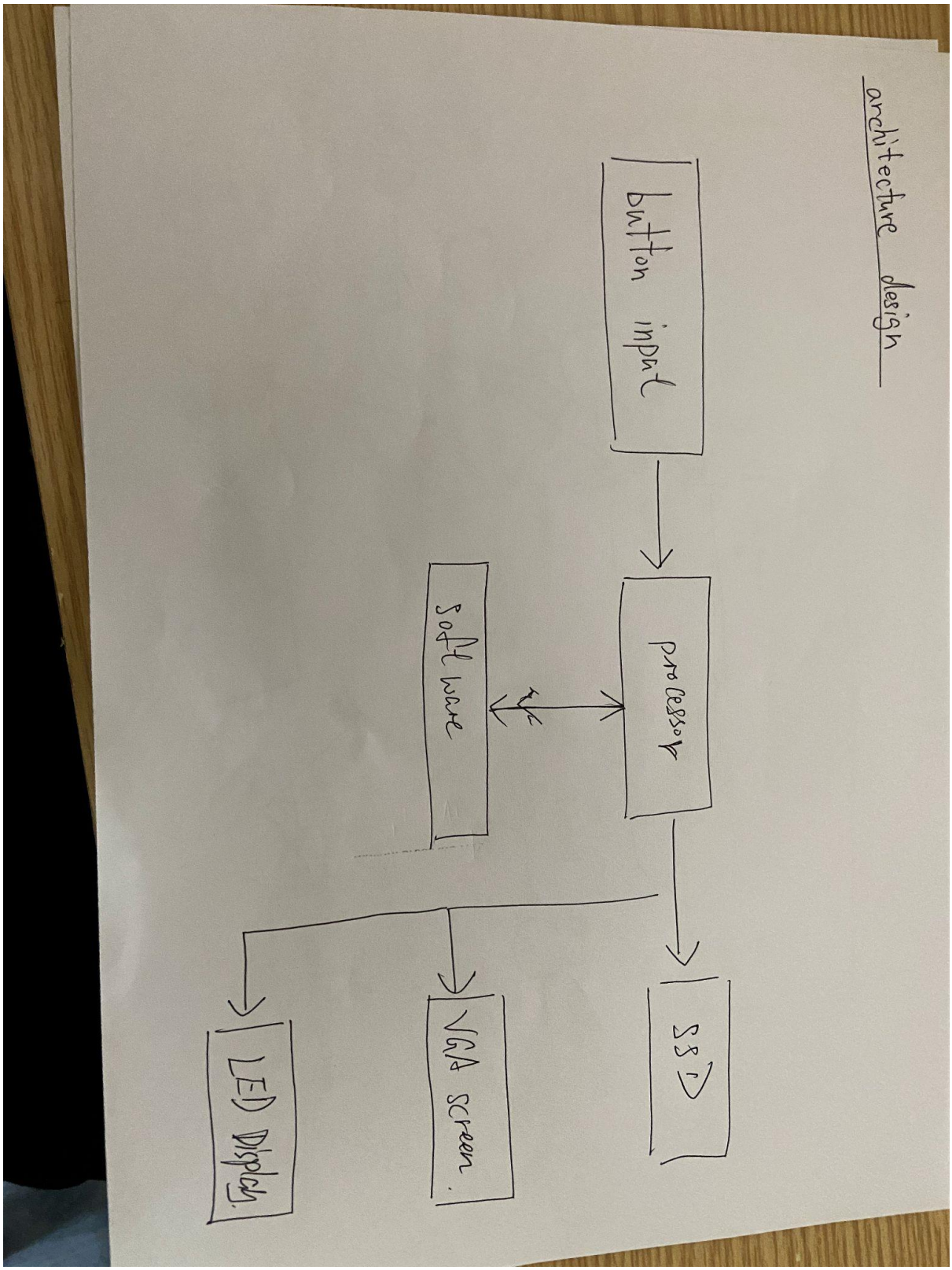
In this project, I am able to successfully create a similar version of the Hare & Hounds using the commonly used Xilinx Digilent ZedBoard Zynq- 7000 ARM/FPGA SoC Development Board.

IX. Diagram

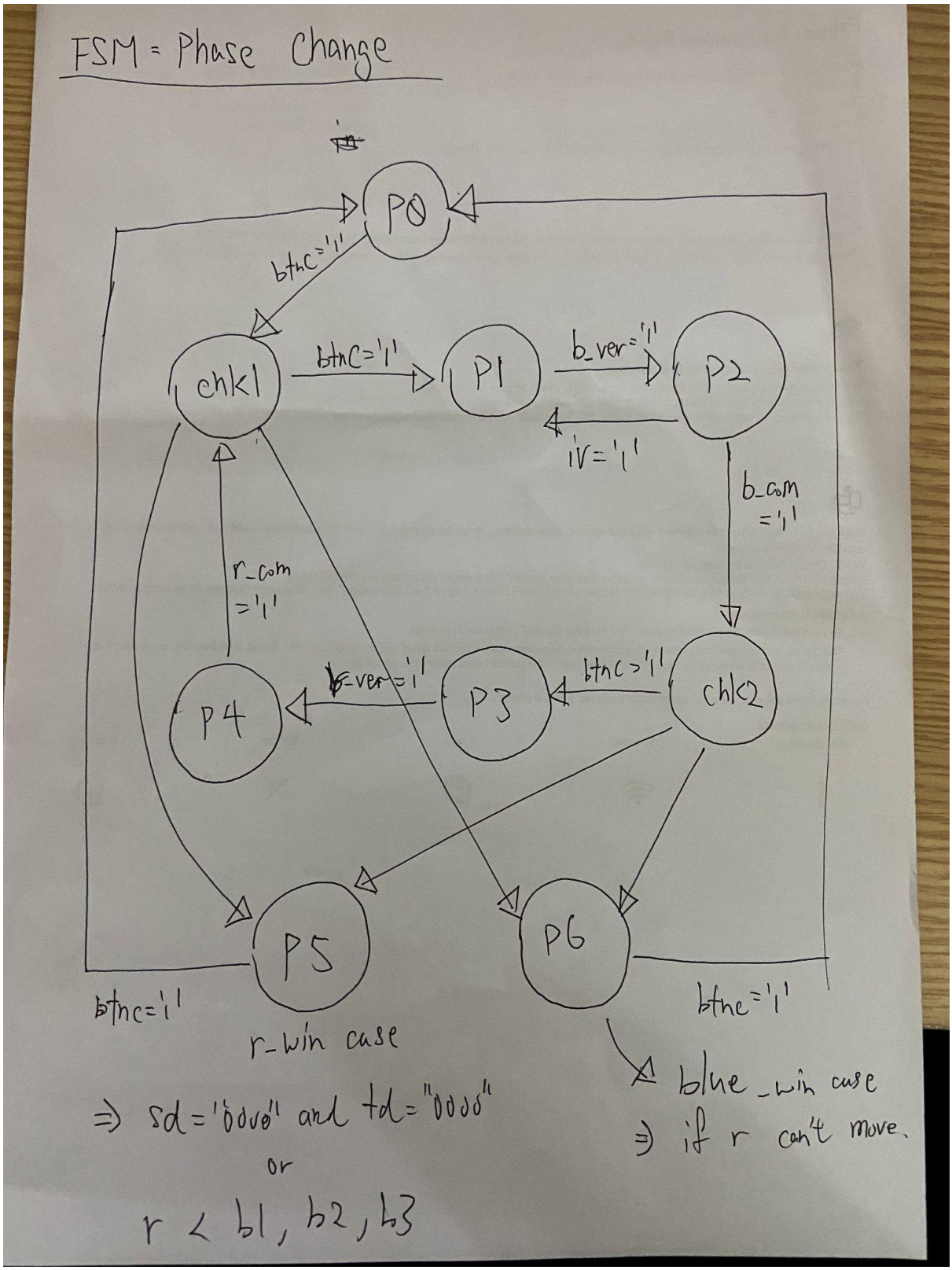
CFG:



arch Desgin:


architecture design
button input
processor
software
SSD
VGA screen
LED Display

FSM:


FSM = Phase Change
P0
btnC='1'
chk1
btnC='1'
P1
b_ver='1'
P2
iv='1'
b_com
='1'
r_com
='1'
P4
b_ver='1'
P3
btnC='1'
chk2
P5
P6
btnc='1'
btnc='1'
r_win case
blue_win case
⇒ if r can't move.
⇒ sd="0000" and td="0000"
or
r < b1, b2, b3

Petri net: