LAB REPORT
******On A Dice Game******

3220101111 洪晨辉

****2024.11.20****

1. Problem Description

The objective of this exercise is to design an electronic dice game similar to the traditional game of craps. The game involves two dice, each capable of producing values between 1 and 6, resulting in sums between 2 and 12. Two counters are used to simulate the rolling of the dice. The game progresses based on the sum of the dice rolls according to specific rules, and the design requires creating multiple hardware components and integrating them into a functional circuit.

Game Rules:

On the first roll of the dice:

The player wins if the sum of the dice is 7 or 11.

The player loses if the sum is 2, 3, or 12.

For any other sum, the result becomes the point, and the player must roll again.

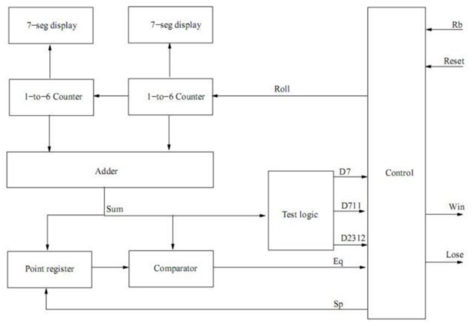
On subsequent rolls:

The player wins if the sum equals the previously stored point.

The player loses if the sum equals 7.

If neither condition is met, the player continues rolling until a win or loss occurs.

2. Design Formulation



The system uses the following key components to implement the dice game:

1-to-6 Counters:

Two counters simulate rolling two dice. The counters increment at high speed when the roll button (Rb) is pressed, making the values unreadable.

When the roll button is released, the counters hold the values, which represent the dice roll results.

Adder:

An adder calculates the sum of the outputs from the two counters, producing a value between 2 and 12.

Point Register:

The point register stores the sum of the dice after the first roll (if it is not a win or loss).

Test Logic:
This block determines specific game conditions:
D7 = 1 if the sum of the dice equals 7.
D711 = 1 if the sum of the dice is 7 or 11.
D2312 = 1 if the sum of the dice equals 2, 3, or 12.

Comparator:
The comparator checks whether the current sum of the dice equals the stored point from the point register, producing the signal Eq.

Control Circuit:
The control circuit manages the game logic based on the signals provided by the test logic and comparator. It:
Detects win/loss conditions (Win, Lose).
Controls the game progression (e.g., storing the point and triggering rolls).

7-Segment Displays:
The outputs of the two counters are displayed on 7-segment displays to show the dice values once the roll button is released.

3. Design Entry

Counters

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Counter is
    Port (
        clk   : in  std_logic; -- Clock signal
        rst   : in  std_logic; -- Reset signal
        roll  : in  std_logic;
        count : out std_logic_vector(2 downto 0);  -- 3-bit output
        cycle : out std_logic
    );
end Counter;

architecture Behavioral of Counter is
    signal count_signal : std_logic_vector(2 downto 0) := "001"; -- Initial value
    signal cycle_signal : std_logic := '0'; -- Signal for cycle output
begin
    process (clk, rst)
        variable count_var : std_logic_vector(2 downto 0) := "001"; -- Internal variable
    begin
        if rst = '1' then
            count_var := "001"; -- Reset to "001" (binary 1)
            cycle_signal <= '0';
        elsif rising_edge(clk) then
            if roll = '1' then
                if count_var = "110" then -- Check if it is "110" (binary 6)
                    count_var := "001"; -- Wrap around to "001" (binary 1)
                    cycle_signal <= '1'; -- New clock to the next counter
                else
                    count_var := std_logic_vector(unsigned(count_var) + 1); -- Increment counter
                    cycle_signal <= '0'; -- Same
                end if;
            end if;
        end if;
        -- Update the signal outputs
        count_signal <= count_var;
    end process;

    -- Assign signals to the output ports
    count <= count_signal;
    cycle <= cycle_signal;

end Behavioral;
```

Adder

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

Entity Adder is port(
  A   : In std_logic;
```

```
 B   : In std_logic;
 CI  : In std_logic;
 Sum : Out std_logic;
 CO  : Out std_logic
 );
End Adder;

Architecture behavioral of Adder Is

Begin

 sum <= A xor B xor CI ;
 CO <= (A and B) or (CI and A) or (CI and B) ;

end behavioral;
```

MultiAdder

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

entity MultiAdder is
     port (
  A    : In  std_logic_vector(2 downto 0);
  B    : In  std_logic_vector(2 downto 0);
  Sum  : Out std_logic_vector(3 downto 0)
 );
end MultiAdder;

Architecture structural of MultiAdder is

component Adder
 port(
  A  : In std_logic;
  B  : In std_logic;
  CI : In std_logic;

  Sum: Out std_logic;
  CO : Out std_logic);
end component;

 signal Carryout: std_logic_vector(1 downto 0);

Begin

 Adder0: Adder port map(A=>A(0), B=>B(0) , CI=>'0'    , Sum=>Sum(0) , CO=>Carryout(0) );
 Adder1: Adder port map(A=>A(1), B=>B(1) , CI=>Carryout(0) , Sum=>Sum(1) , CO=>Carryout(1) );
 Adder2: Adder port map(A=>A(2), B=>B(2) , CI=>Carryout(1) , Sum=>Sum(2) , CO=>Sum(3)    );


end structural;
```

Point_Register

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

entity Point_register is
    generic ( width : integer := 3 );

    Port ( clk       : in  std_logic;     -- Clock input
           rst       : in  std_logic;     -- Synchronous reset
           Sp        : in  std_logic;     -- Load enable signal
           Data_In   : in  std_logic_vector(width downto 0);    -- Data input
           Data_Out  : out std_logic_vector(width downto 0)     -- Data output
    );
end Point_register;

architecture slice of Point_register is
    signal Data : std_logic_vector(width downto 0) := (others => '0'); -- Internal signal
begin
    process(clk)  -- Only clk in the sensitivity list for synchronous logic
    begin
        if rising_edge(clk) then
            if rst = '1' then
                -- Reset the register synchronously
                Data <= (others => '0');
            elsif Sp = '1' then
                -- Load new data when Sp is high
                Data <= Data_In;
            end if;
        end if;
    end process;
```

```vhdl

    -- Assign internal signal to output
    Data_Out <= Data;

end slice;
```

TestLogic

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity TestLogic is
    Port (
        Sum     : in std_logic_vector(3 downto 0); -- Sum of the two dice
        D7      : out STD_LOGIC;
        D711    : out STD_LOGIC;
        D2312   : out STD_LOGIC
    );
end TestLogic;

architecture Behavioral of TestLogic is
begin
    PROCESS(Sum)
    BEGIN
        -- Default outputs
        D7    <= '0';
        D711  <= '0';
        D2312 <= '0';

        CASE Sum IS
            WHEN "0111" =>
                D7    <= '1';
                D711  <= '1';
            WHEN "1011" =>
                D711  <= '1';
            WHEN "0010" | "0011" | "1100" =>
                D2312 <= '1';
            WHEN OTHERS =>
                -- Do nothing, all signals remain '0'
                NULL;
        END CASE;
    END PROCESS;
end Behavioral;
```

Comparator

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comparator is
    generic(
        WIDTH : integer := 3
    );
    Port (
 clk : in STD_LOGIC;
          CompA : in STD_LOGIC_VECTOR (WIDTH   downto 0);
          CompB : in STD_LOGIC_VECTOR (WIDTH   downto 0);
          Eq: out STD_LOGIC);
end comparator;

architecture Behavioral of comparator is
begin
    process(clk) begin
     if (CompA = CompB) then
         Eq <= '1';
     else
          Eq <= '0';
     end if;
    end process;
end Behavioral;
```

Control

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Control is
    Port (
        clk     : in  STD_LOGIC;
        rst     : in  STD_LOGIC;
        Rb      : in  STD_LOGIC; -- Roll button
        D7      : in  STD_LOGIC;
        D711    : in  STD_LOGIC;
        D2312   : in  STD_LOGIC;
        Eq      : in  STD_LOGIC;

        -- Outputs
        Win     : out STD_LOGIC;
        Lose    : out STD_LOGIC;
        Sp      : out STD_LOGIC;
        Roll    : out STD_LOGIC
    );
end Control;

architecture Behavioral of Control is
    -- State Declaration
    TYPE STATE_TYPE IS (Start, Play1, Play2Interval, Play2,
  EndGame);
    SIGNAL state : STATE_TYPE := Start;
    SIGNAL Roll0 : std_logic;

begin
    PROCESS(clk, rst)
    BEGIN

        Roll0 <= Rb;
        Roll <= Roll0;

  IF rst = '1' THEN
            state  <= Start;
            Win    <= '0';
            Lose   <= '0';
            Sp     <= '1';
            Roll   <= '0';

        ELSIF state = EndGame THEN
   Roll <= '0';

        ELSIF rising_edge(clk) THEN
            -- Default outputs

            CASE state IS
                WHEN Start =>
                    IF Rb = '1' THEN
                        state <= Play1;
                    END IF;

    WHEN Play1 =>
     IF Rb = '0' THEN
      Sp <= '0';
      IF D711 = '1' THEN
       Win <= '1';
       state <= EndGame;
      ElSIF D2312 = '1' THEN
       Lose <= '1';
       state <= EndGame;
      ELSE
       state <= Play2Interval;
      END IF;
     END IF;

    When Play2Interval =>
     IF Rb = '1' THEN
      state <= Play2;
     END IF;

                WHEN Play2 =>
                    IF Rb = '0' THEN
                        IF D7 = '1' THEN
                            Lose <= '1';
                            state <= EndGame;
                        ELSIF Eq = '1' THEN
                            Win <= '1';
                            state <= EndGame;
                        ELSE
       state <= Play2Interval;
                        END IF;
                    END IF;

                WHEN EndGame =>
                    -- Wait for reset
                    NULL;
            END CASE;
        END IF;
    END PROCESS;

end Behavioral;
```

Char_7seg

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

entity char_7seg is
    port (
        c : in std_logic_vector(2 downto 0); -- 4-bit inpu
 t character code
        display : out std_logic_vector(7 downto 0) -- 7-se
 gment output
    );
end char_7seg;
```

```

architecture behavior of char_7seg is
begin
    process(c)
    begin
        case c is
            when "000" => display <= "11000000"; -- 0
            when "001" => display <= "11111001"; -- 1
            when "010" => display <= "10100100"; -- 2
            when "011" => display <= "10110000"; -- 3
            when "100" => display <= "10011001"; -- 4
            when "101" => display <= "10010010"; -- 5
            when "110" => display <= "10000010"; -- 6
            when others => display <= "11111111"; -- Default to blank
        end case;
    end process;
end behavior;
```

Dice(Main function)

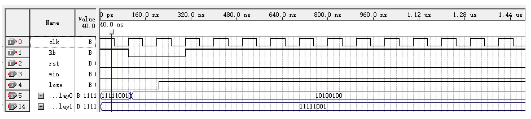
```
library IEEE;
use ieee.std_logic_1164.all;

entity Dice is
generic(
        width : Natural := 3 --
    );
     port (
    clk     : in  std_logic;
    Rb     : in std_logic;
           rst     : in  std_logic;

    win : out std_logic;
    lose : out std_logic;

    display0 : out std_logic_vector(7 downto 0);
    display1 : out std_logic_vector(7 downto 0)

 );
end Dice;

Architecture structural of Dice is
-----------------------------------------------
component MultiAdder
     port (
  A       : In  std_logic_vector(width -1 downto 0);
  B       : In  std_logic_vector(width -1 downto 0);
  Sum     : Out std_logic_vector(width    downto 0)
 );
end component;
-----------------------------------------------
component Counter
     Port ( clk      : in std_logic;
               rst      : in std_logic;
        roll     : in std_logic;
               count    : out std_logic_vector (width -1 downto 0);
               cycle    : out std_logic
 );
end component;
-----------------------------------------------
component Point_register
     Port (
        clk      : in  std_logic;
               rst      : in  std_logic;
        Sp       : in  std_logic;
               Data_In  : in  std_logic_vector (width downto 0);
               Data_Out : out std_logic_vector (width downto 0)
 );
end component;
-----------------------------------------------
component comparator
     Port ( clk      : in  std_logic;
               CompA    : in  STD_LOGIC_VECTOR (width    downto 0);
               CompB    : in  STD_LOGIC_VECTOR (width    downto 0);
               Eq       : out STD_LOGIC
 );
end component;
-----------------------------------------------
component TestLogic
 Port (
        Sum     : in std_logic_vector(3 downto 0); -- Sum of the two dice
        D7      : out STD_LOGIC;
        D711    : out STD_LOGIC;
```

```vhdl
        D2312   : out STD_LOGIC
    );
end component;
-----------------------------------------------
component Control
    Port (
        clk     : in  STD_LOGIC;
        rst     : in  STD_LOGIC;
        Rb      : in  STD_LOGIC; -- Roll button
        D7      : in  STD_LOGIC;
        D711    : in  STD_LOGIC;
        D2312   : in  STD_LOGIC;
        Eq      : in  STD_LOGIC;

        -- Outputs
        Win     : out STD_LOGIC;
        Lose    : out STD_LOGIC;
        Sp      : out STD_LOGIC;
        Roll    : out STD_LOGIC
    );
end component;
-----------------------------------------------
component char_7seg
    Port (
        c : in std_logic_vector(2 downto 0); -- 4-bit input character code
        display : out std_logic_vector(7 downto 0) -- 7-segment output
    );
end component;
-----------------------------------------------
    signal Cnt0   : std_logic_vector(width -1 downto 0):= (others => '0');
    signal Cnt1   : std_logic_vector(width -1 downto 0):= (others => '0');
    signal DatIn : std_logic_vector(width    downto 0):= (others => '0');
    signal DatOut: std_logic_vector(width    downto 0):= (others => '0');
    signal Sum0   : std_logic_vector(width    downto 0);
    signal Sp0 : std_logic;
    signal Eq0 : std_logic;
    signal D70 : std_logic;
    signal D7110 : std_logic;
    signal D23120 : std_logic;
    signal Roll0 : std_logic;
    signal cycle0 : std_logic;
    signal cycle1 : std_logic;


Begin

Counter0:        Counter             port map (clk => clk , rst    => rst   , roll    => Roll0 , count => cnt0, cycle => cycle0);
Counter1:        Counter             port map (clk => cycle0 , rst    => rst   , roll    => Roll0 , count => cnt1, cycle => cycle1 );
MultiAdder0:     MultiAdder          port map (A     => cnt0, B     => cnt1  , Sum    => Sum0 );
Point_register0:    Point_register      port map (clk => clk , rst    => rst   , Sp     => Sp0    , Data_In => Sum0, Data_Out => DatOut);
comparator0:        comparator          port map (clk => clk , CompA => DatOut, CompB => Sum0   , Eq => Eq0 );
TestLogic0:    TestLogic           port map (Sum => Sum0, D7 => D70 , D711 => D7110, D2312 => D23120);
Control0:           Control             port map (clk => clk , rst => rst , Rb => Rb, D7 => D70, D711 => D7110, D2312 => D23120 , Eq => Eq0, Win => win, Lose => lose, Sp => Sp0, Roll => Roll0);
Char_7seg0:           char_7seg           port map (c => cnt0, display => display0 );
Char_7seg1:           char_7seg           port map (c => cnt1, display => display1 );

end structural;

end behavior;
```

4. Simulation and Synthesis Results

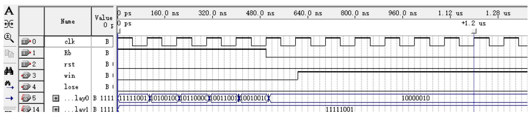
On the first roll of dice:

Loses when D2312 = 1(e.g. 3)



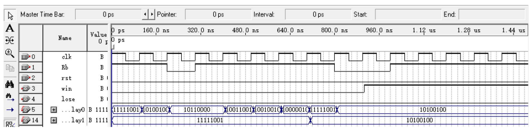
Wins when D711 = 1(e.g. 7)

Else proceeds.

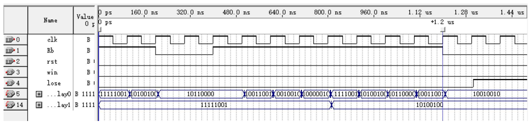


On the second roll of dice:

Wins when Eq = 1



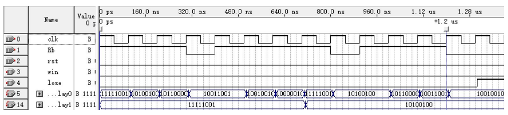
Loses when D7 = 1



Else proceeds.

On any subsequent roll:

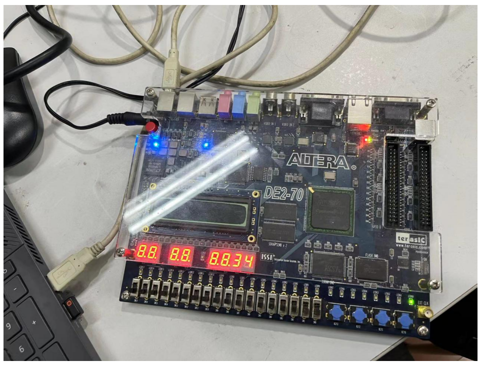
The criteria remains, e.g. 1st Point = 5 and the 3rd roll triggers.



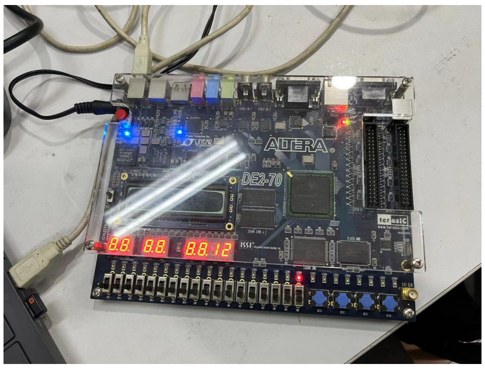
5. Experimental Results

On the first roll of dice:

Wins when D711 = 1 (e.g. 7)



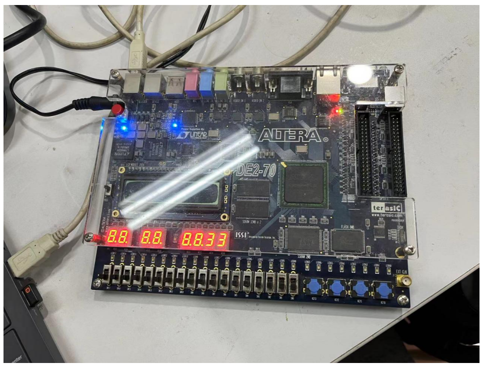
Lose when D2312 = 1 (e.g. 3)



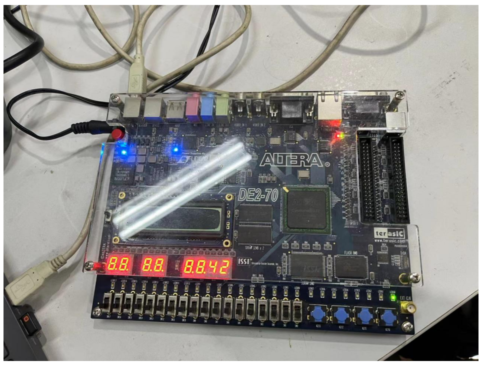
Else proceeds.

On the elaborating roll:

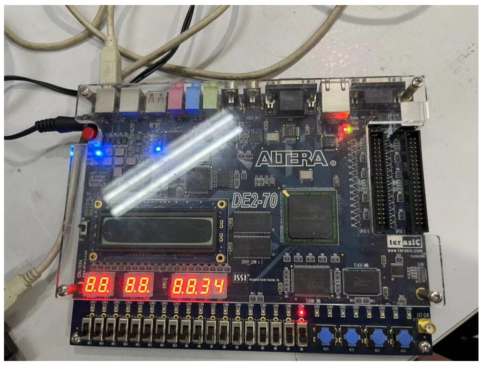
If the first roll reads 6,



...the green light would be ablaze,



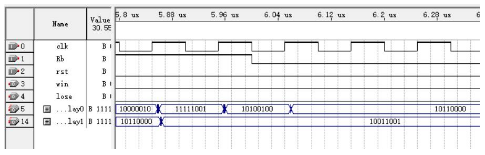
...else if D7 = 1, one loses.



6. Discussion and Conclusion

This humble implementation, though seemingly running smoothly, is a result of sheer dedication and countless trials and errors. One of the most notable glitches encountered during the early stages of development was a peculiar one-cycle lag. This issue would have been painstaking to resolve without abandoning the 'Rb in, Roll out' structure of the Control block and directly plugging the

Roll Button signal into the subsequent stage, which is the first stage of the cascaded counter.



The root cause of this issue lies in the timing sequence. When the Rb signal is turned off, it isn’t processed until the next cycle. This delay turns off the roll signal but prevents the counter from addressing it because both are operating within the same asynchronous structure. Since the counter cannot predict the next state of the roll button, the issue remains unsolvable unless the roll signal is deliberately moved forward. This requires re-routing it directly to the roll button input.