Faculty of Automation and Computer Science

DICE GAME

Supervisor:Fleger Dan

Students:Feier Cătălin Vasile

Butaș Rafael Dorian

Group: 30414

02.06.2023

TABLE OF CONTENTS

1.Summary

Gambling (also known as **betting** or **gaming**) is the wagering of something of value ("the stakes") on a random event with the intent of winning something else of value, where instances of strategy are discounted. Gambling thus requires three elements to be present: consideration (an amount wagered), risk (chance), and a prize. The outcome of the wager is often immediate, such as a single roll of dice, a spin of a roulette wheel, or a horse crossing the finish line, but longer time frames are also common, allowing wagers on the outcome of a future sports contest or even an entire sports season.

Our project consists of a dice game in which the dice faces are categorized in the following way:1,2,3-DOWN and 4,5,6-UP.

Since it's a two-player game, each player needs to introduce a sequence of 3 0/1 digits, where 0 stands for DOWN and 1 stands for UP. After that, all the dice outcomes are verified with the players sequences, and the first player whose sequence appears in that exact order wins the game.

Key elements of the Dice Game Project include:

Nexys 4 FPGA Board for running the game

Random Number Generator for the dice outcomes

Seven Segment Display for the design of the game

Frequency divider from 100 MHz (the base clock) to 10 MHz (0,1 sec, the speed of the numbers used to create an animation for the roll of the dice)

Debouncer for ensuring that we have a single button press

2.Theoretical Foundation

The theoretical foundation for the Dice Game project consists of several principles from digital design, VHDL, and FPGA technologies. This foundation serves as a basis for understanding the project's design choices and implementation strategies.

Digital logic design: The basis of electronic systems, such as computers and cell phones. Digital logic is rooted in binary code, which renders information through zeroes and ones, giving each number in the binary code an opposite value. This system facilitates the design of electronic circuits that convey information, including logic gates with functions that include AND, OR, and NOT commands. The value system translates input signals into specific outputs. These functions ease computing, robotics, and other electronic applications.

VHDL: A hardware description language (HDL) used to design electronic systems at the component, board and system level. VHDL allows models to be developed at a very high level of abstraction. Initially conceived as a documentation language only, most of the language can today be used for simulation and logic synthesis.

Electronics: Electronics is the branch of science that deals with the study of flow and control of electrons (*electricity*) and the study of their behavior and effects in vacuums, gases, and semiconductors, and with devices using such electrons. Understanding the basics of electronics is key for comprehending the project's functionalities.

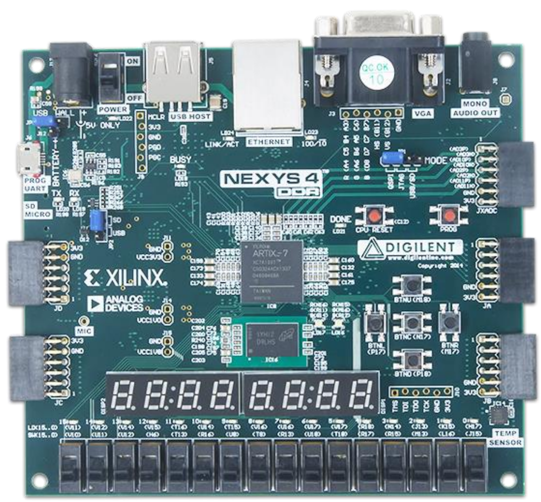
FPGA: Field Programmable Gate Arrays (FPGAs) are integrated circuits often sold off-the-shelf. They're referred to as 'field programmable' because they provide customers with the ability to reconfigure the hardware to meet specific use case requirements after the manufacturing process. This allows for feature upgrades and bug fixes to be performed in situ, which is especially useful for remote deployments.

FPGAs have configurable logic blocks (CLBs) and a set of programmable interconnects that allow the designer to connect blocks and configure them to perform everything from simple logic gates to complex functions. Full SoC designs holding multiple processes can be put onto a single FPGA device.

3.Design and Implementation

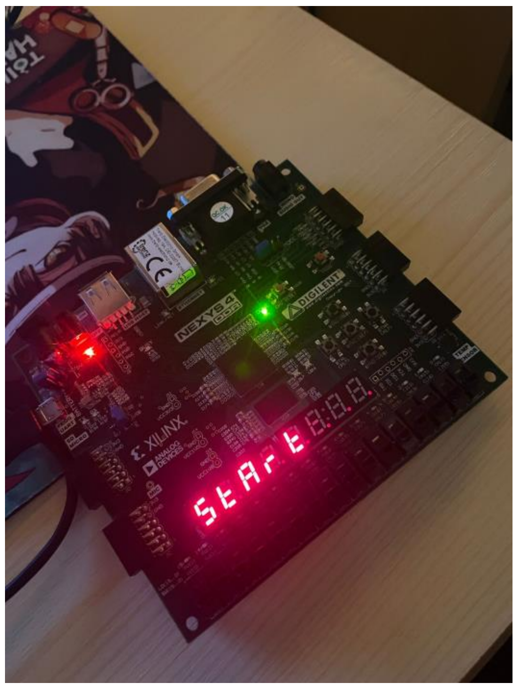
3.1 The Nexys 4 Board functionalities

The Nexys 4 DDR board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7 Field Programmable Gate Array (FPGA) from Xilinx. With its large, high-capacity FPGA (Xilinx part number XC7A100T-1CSG324C), generous external memories, and collection of USB, Ethernet, and other ports, the Nexys4 DDR can host designs ranging from introductory combinational circuits to powerful embedded processors. Several built-in peripherals, including an accelerometer, temperature sensor, MEMs digital microphone, a speaker amplifier, and several I/O devices allow the Nexys4 DDR to be used for a wide range of designs without needing any other components.



3.2 The design idea of the project

At the beginning of the game, or when you press the Reset Button, you will have displayed on the SSD the word "Start" like in the following image:

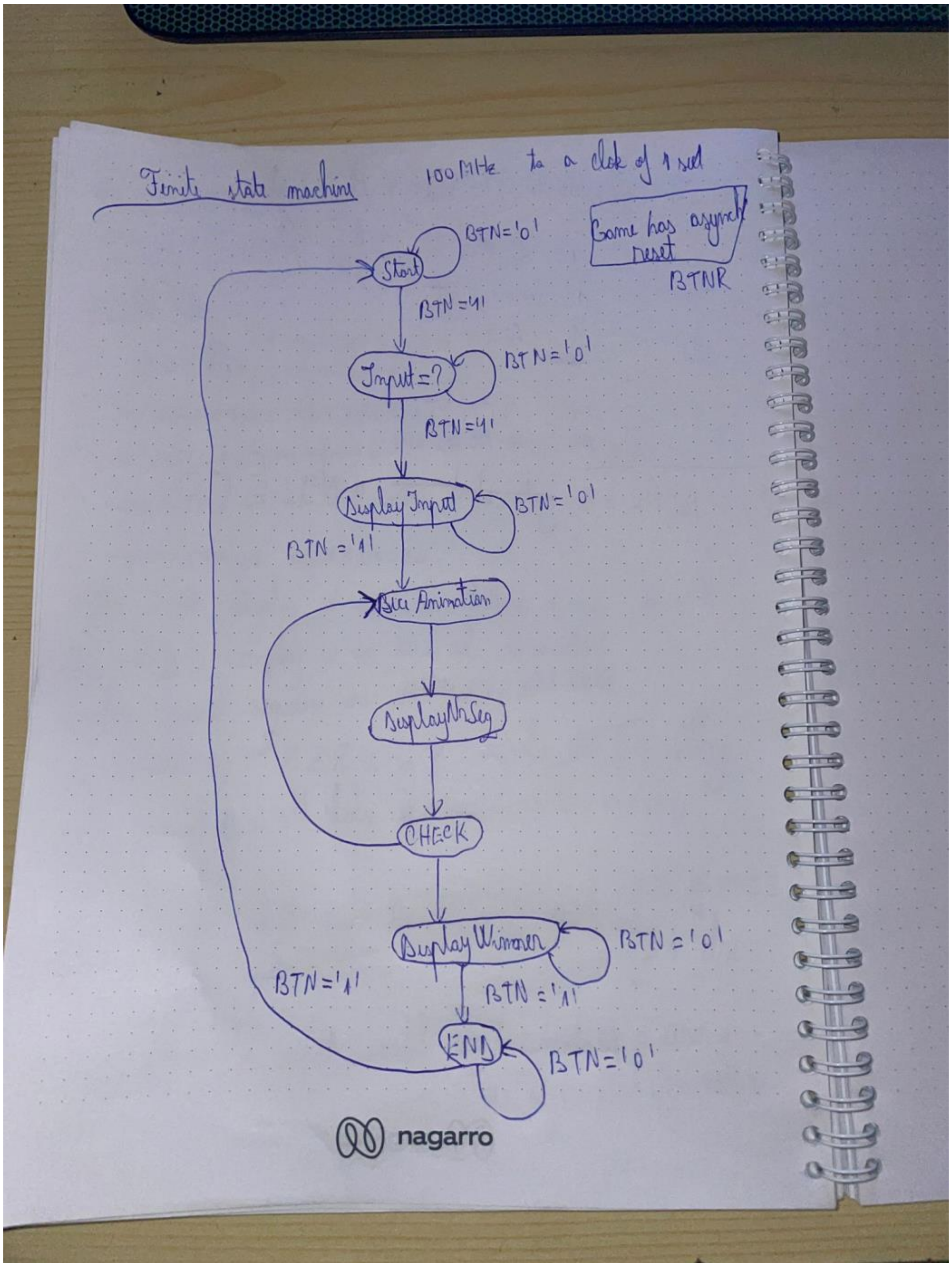


Then, the switches 15,14,13 in this order will represent the user’s 1 sequence (switch OFF is 0 or DOWN, switch ON is 1 or UP), and the same thing for the second user, but with switches 2,1 and 0. The default state of the sequence is 000.

In the next phase, the two sequences will be displayed on the SSD and the dice will start rolling (numbers will be displayed really fast on the last anode and on the first anode the shape of a dice will rotate to create the animation), until it stops and it gives the outcome which will be on the last anode as well.

At each result, near the number (on the sixth anode) it will be displayed U from UP or D from DOWN depending of the value. Also, after each roll the sequence will be right shifted and seq (0) will be equal to 0 if value<3 or 1 if value>3.This sequence will be displayed on the first four anodes (last number of the previous sequence will be also there to see the shifting better). This state will be repeated until a sequence from one of the users is matching the current one.

The following schematic will help you understanding better the functionality and the states of the project:


Finite state machine
100 MHz to a clock of 1 sec
Game has async Reset
BTNR
Start
BTN='0'
BTN='1'
Input=?
BTN='0'
BTN='1'
Display Input
BTN='0'
BTN='1'
Seq Animation
DisplayNrSeq
CHECK
Display Winner
BTN='0'
BTN='1'
BTN='1'
END
BTN='0'
nagarro

3.3 Hardware implementation

Our project is split into smaller subsystems to enhance the modularity and the readability of the code.

1.The MPG (monopulse generator or debouncer) module that filters the pulse of the buttons used:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Debouncer is
Port (
 clk: in std_logic; btn: in std_logic; en: out std_logic);
end Debouncer;
architecture Behavioral of Debouncer is
signal counter: std_logic_vector(16 downto 0):=(others=>'0');
signal Q1, Q2, Q3: std_logic;
Begin
 en <= not(Q3) and Q2;
process(clk)
 begin
```

```
if rising_edge(clk) then
counter<= counter+1;
if counter ="1111111111111111" then
Q1<=btn;
end if;
Q2<=Q1;
Q3<=Q2;
end if;
end process;
end Behavioral;
```

2.The SSD(Seven Segment Display) Module that displays, in each state, the right data.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY SSD IS
  PORT(   CLK: IN    STD_LOGIC;              -- STANDARD CLOCK OF 100 MHZ
        -- ON AFIS WILL BE STORED WHAT WILL BE DISPLAYED ON EACH ANODE
        AFIS0:  IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        AFIS1:  IN STD_LOGIC_VECTOR (4 DOWNTO 0);
```

```
        AFIS2:  IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        AFIS3:  IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        AFIS4:  IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        AFIS5:  IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        AFIS6:  IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        AFIS7:  IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        AN:     OUT STD_LOGIC_VECTOR    (7 DOWNTO 0);
        CAT:    OUT STD_LOGIC_VECTOR    (6 DOWNTO 0)
       );
END SSD;

ARCHITECTURE BHV OF SSD IS

   -- SIGNAL DECLARATION
   SIGNAL COUNT: STD_LOGIC_VECTOR(16 DOWNTO 0) := (16 DOWNTO 0 => '0');
   SIGNAL INPUT_DECODER: STD_LOGIC_VECTOR  (4 DOWNTO 0);

BEGIN

   -- THE COUNTER
   PROCESS(CLK)
   BEGIN
```

```
IF  CLK='1' AND CLK'EVENT THEN
   COUNT<=COUNT+1;
END IF;
END PROCESS;


-- SELECTING THE ANODES
-- AND WHAT NEEDS TO BE DISLPAYED ON EACH OF THEM CONSEQUENTIELLY
PROCESS(COUNT)
BEGIN
CASE(COUNT (16 DOWNTO 14)) IS
   WHEN "000" => AN<="11111110";
             INPUT_DECODER<=AFIS0;   -- SELECT ANODE 0
   WHEN "001" => AN<="11111101";
             INPUT_DECODER<=AFIS1;   -- SELECT ANODE 1
   WHEN "010" => AN<="11111011";
             INPUT_DECODER<=AFIS2;    -- SELECT ANODE 2
   WHEN "011" => AN<="11110111";
             INPUT_DECODER<=AFIS3;   -- SELECT ANODE 3
   WHEN "100" => AN<="11101111";
             INPUT_DECODER<=AFIS4;   -- SELECT ANODE 4
   WHEN "101" => AN<="11011111";
             INPUT_DECODER<=AFIS5;   -- SELECT ANODE 5
```

```
   WHEN "110" => AN<="10111111";
              INPUT_DECODER<=AFIS6;   -- SELECT ANODE 6
   WHEN OTHERS => AN<="01111111";
               INPUT_DECODER<=AFIS7;   -- SELECT ANODE 7
END CASE;
END PROCESS;

-- WHAT WILL THE CATODES BE FOR EVERY CASE, THE NUMBER OF DISPLAYS I HAVE!!
-- they are encoded in this order each abcdefg
PROCESS(INPUT_DECODER)
BEGIN
   CASE    INPUT_DECODER IS
      when "00000" =>  cat<="0100100"; -- S           1
      when "00001" =>  cat<="1110000"; -- t           2
      when "00010" =>  cat<="0001000"; -- A           3
      when "00011" =>  cat<="1111010"; -- r           4
      when "00100" =>  cat<="1001111"; -- I           5
      when "00101" =>  cat<="1101010"; -- n           6
      when "00110" =>  cat<="0011000"; -- P           7
      when "00111" =>  cat<="1000001"; -- U           8
      when "01000" =>  cat<="1110110"; -- =           9
      when "01001" =>  cat<="0011010"; -- ?           10
```

```
        when "01010" =>  cat<="1000010"; -- d                 11
        when "01011" =>  cat<="1001111"; -- 1                 12
        when "01100" =>  cat<="0010010"; -- 2                 13
        when "01101" =>  cat<="0000110"; -- 3                 14
        when "01110" =>  cat<="1001100"; -- 4                 15
        when "01111" =>  cat<="0100100"; -- 5                 16
        when "10000" =>  cat<="0100000"; -- 6                 17
        when "10001" =>  cat<="1100010"; -- full dice          18
        when "10010" =>  cat<="1110010"; -- c                 19
        when "10011" =>  cat<="1101010"; -- u intors           20
        when "10100" =>  cat<="1100110"; -- ]                 21
        when "10101" =>  cat<="0110000"; -- E                 22
        when others =>   cat<="1111111"; -- display nothing     23

      END CASE;
   END PROCESS;

END BHV;
```

3.The Random Number Generator Module which gives us the outcomes of each roll of the dice

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

```
use ieee.std_logic_unsigned.all;


entity randomNr is
   Port ( clk   : in std_logic;
          reset : in std_logic;
          ranNr : out std_logic_vector (2 downto 0)
        );
end randomNr;


architecture Behavioral of randomNr is
   signal nr    : integer :=129;
   constant x : integer := 1664525;
   constant y : integer := 2147483647; -- 2^31-1
   constant z : integer := 1442243407;
begin
   process(clk, reset)
      variable tmp : integer;
   begin
      if reset = '1' then
         nr <= 129;
      elsif rising_edge(clk) then
         tmp := (x * nr + z) mod y;
```

```
        nr <= tmp;
      end if;
  end process;
  ranNr<=std_logic_vector(to_unsigned(( nr mod 6) + 1,3));

end Behavioral;
```

4.The Frequency Divider Module to obtain a 0.1 sec(10 hz)clock for displaying numbers fast while the dice is rolling

```
-- Frequency Divider From 100MHz to 10Hz=0.1sec
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity FDiv01s is
Port ( clk: in std_logic;
      rst: in std_logic;
      new_clk: out std_logic
);
end FDiv01s;
architecture Behavioral of FDiv01s is
  signal clk_div: std_logic:='0';
  signal count: INTEGER:=0;
begin
```

```
  process(clk)
    begin
      if rst='1' then
        clk_div <='0';
        elsif rising_edge(clk) then
          if count<4_999_999 then
            count<=count+1;
          else
            count<=0;
            clk_div<=not(clk_div);
          end if;
      end if;
   end process;
   new_clk<=clk_div;
end Behavioral;
```

4.CONCLUSIONS

In conclusion, this project is a successful dice game that you can play with you friends at any time. Why? Because it combines a decent design and animation done with the help of the seven segment display and the frequency divider, with a fair chance created by the random number generator and a good functionality of the buttons provided by the debouncer module.

5.REFERENCES

-Materials provided by our supervisor

-Tutorials and articles for better understanding how each module work

-Articles for improving definitions of the components used in the project like the following:

www.javatpoint.com

www.wikipedia.com

https://learn.org/

6.MAIN CODE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.numeric_std.all;


entity MAIN is
```

```
 Port (    btnc: in std_logic;                          -- button to go forward
          rst:  in std_logic;                           -- master reset button
          clk:  in std_logic;                           -- clock of 100Mhz
          in_p1:  in    std_logic_vector(2 downto 0);     -- player 1 input
          in_p2:  in    std_logic_vector(2 downto 0);     -- player 2 input
          an:   out std_logic_vector(7 downto 0);         -- anode
          cat:  out std_logic_vector(6 downto 0)          -- cathode
           );
end MAIN;

architecture Behavioral of MAIN is

   -- The new clocks of 1sec and 01sec
   signal      clk1s: std_logic;
   signal      clk01s: std_logic;

   --The input to be saved from the USERS
   signal  p1: std_logic_vector(2 downto 0):="000";
   signal  p2: std_logic_vector(2 downto 0):="000";
```

```
--The initial sequence!
signal seq: std_logic_vector(3 downto 0):="ZZZZ";

-- USED COUNTERS FOR THE FSM
signal  CNT1: INTEGER:=0;   -- counter transition
signal  CNT2: INTEGER:=0;   -- counter animation
--The randomNr!
signal randNr: std_logic_vector(2 downto 0); -- It is constantly generated at every 10 nanosecs
signal rand_nr: std_logic_vector(2 downto 0); -- It is saved at a state

--Send signals for each afis in SSD
 signal send0: std_logic_vector(4 downto 0);
 signal send1: std_logic_vector(4 downto 0);
 signal send2: std_logic_vector(4 downto 0);
 signal send3: std_logic_vector(4 downto 0);
 signal send4: std_logic_vector(4 downto 0);
 signal send5: std_logic_vector(4 downto 0);
 signal send6: std_logic_vector(4 downto 0);
 signal send7: std_logic_vector(4 downto 0);
```

```
 -- Signals to check who is the winner!
 signal ok1: std_logic:='0'; -- For player1
 signal ok2: std_logic:='0'; -- For player2

 --Declaring the STATEs of the project
 type states is(start, input, displayIn, dice, displaynr, check, displayWin, ed);
 signal current_state: states:= start; -- Initialize the states
 signal next_state:    states:= start;

 --The debounced buttons for BTNC and RST
 signal dbtnc: std_logic;
 signal drst:  std_logic;
begin
 -- Getting the debounced button BTNC and RST
 deb1: entity WORK.debouncer port map
 (
   clk=>clk,
   btn=>btnc,
   en=>dbtnc
```

```
);
deb2: entity WORK.debouncer port map
(
   clk=>clk,
   btn=>rst,
   en=>drst
 );

 --Getting the 2 new CLOCKS!! 1s and 0.1s
 fdiv1: entity WORK.FDiv1s port map
 (
   clk=>clk,
   rst=>drst,
   new_clk=>clk1s
 );
 fdiv01: entity WORK.FDiv01s port map
 (
   clk=>clk,
   rst=>drst,
   new_clk=>clk01s
```

```
 );
 -- GETTING THE RANDOM NUMBER, VIA THE PSEUDO-RANDOM-NR-GENERATOR
rannr: entity WORK.randomNr port map
(
   clk=>clk,                    -- USE the 10 NS clock so we get the random nr more random...
   reset=>rst,
   ranNr=>randNr
);
-- MAKING THE CONNECTION TO THE SSD BLACK BOX :)
ses: entity WORK.SSD port map
(
   clk=>clk,
   AFIS0=>SEND0,
   AFIS1=>SEND1,
   AFIS2=>SEND2,
   AFIS3=>SEND3,
   AFIS4=>SEND4,
   AFIS5=>SEND5,
   AFIS6=>SEND6,
   AFIS7=>SEND7,
```

```
   AN=>AN,
   CAT=>CAT
);

-- THE CHANGING STATES PROCESS
process(clk,drst)
   begin
   if drst='1' then
      current_state<=start;
   elsif rising_edge(clk) then
      current_state<=next_state;
      end if;
end process;

-- The process of transition of states, using the state diagram
process(current_state,dbtnc,ok1,ok2,cnt1)
begin
   case current_state is
   when start =>
      if dbtnc='1' then next_state<=input;
```

```
              else next_state<=start; end if;
    when input =>
       if dbtnc='1' then next_state<=displayIn;
              else next_state<=input; end if;
    when displayIn =>
       if dbtnc='1' then next_state<=dice;
              else next_state<=displayin; end if;
   when dice =>
       if cnt1 > 4 then next_state<=displaynr;
              else next_state<=dice;    end if;
   when displaynr => if dbtnc='1' then next_state<=check;
              else next_state<=displaynr; end if;
   when check => if ok1='1' or ok2='1' then next_state<= displaywin;
             else   next_state<=dice; end if;
   when displaywin => if dbtnc='1' then next_state<=ed;
               else next_state<=displaywin; end if;
   when ed => if dbtnc='1' then next_state<=start;
            else next_state<=ed; end if;
   when others => next_state<= start;
 end case;
```

```
 end process;


 -- The counter which helps to make the transition between dice and displayNr
 process(clk1s) is
 begin
    if rising_edge(clk1s) then
      if current_state=DICE then
        if cnt1>5 then
        cnt1<=0;
        else
        cnt1<=cnt1+1;
     end if;
    end if;
    end if;
end process;


-- The counter that will make the numbers and dice animation stay on 0,1 sec each so it gives a fast impression
process(clk01s)
begin
```

```
    if rising_edge(clk01s) then
      if  current_state=DICE then
         if cnt2>6 then
            cnt2<=0;
          else
          cnt2<=cnt2+1;
          end if;
      end if;
   end if;
 end process;
--The process of what to do in each state!
process(current_state)is
begin
   case current_state is
   WHEN START=>
            SEND0<="11111";
            SEND1<="11111";
            SEND2<="11111";
            SEND3<="00001";
            SEND4<="00011";
```

```
        SEND5<="00010";
        SEND6<="00001";
        SEND7<="00000";
WHEN INPUT=>
        SEND0<="11111"; -- nothING
        SEND1<="01001"; -- ?
        SEND2<="01000"; -- =
        SEND3<="00001"; -- T
        SEND4<="00111"; -- U
        SEND5<="00110"; -- P
        SEND6<="00101"; -- N
        SEND7<="00100"; -- I
WHEN DISPLAYIN=>
    if p2(0)='0' then
          SEND0<="01010";
        else SEND0<="00111"; end if;
    if p2(1)='0' then
          SEND1<="01010";
        else
          send1<="00111"; end if;
```

```
    if  p2(2)='0' then
         send2<="01010";
      else send2<="00111"; end if;


      SEND3<="11111";
      SEND4<="11111";


    if  p1(0)='0' then
      SEND5<="01010";
     else SEND6<="00111"; end if;
    if p1(1)='0' then
      SEND6<="01010";
     else SEND6<="00111"; end if;
    if p1(2)='0' then
      SEND7<="01010";
      else SEND7<="00111"; end if;
WHEN DICE=>
      case cnt2 is
      when 0 => send7<="01011";     send0<="10001";
      when 1 => send7<="01100";     send0<="00111";
```

```
        when 2 => send7<="01101";      send0<="00111";
        when 3 => send7<="01110";      send0<="10100";
        when 4 => send7<="01111";      send0<="10011";
        when 5 => send7<="10000";      send0<="10010";
        when others => send7<="10000"; send0<="10001";
        end case;

        SEND1<="11111";
        SEND2<="11111";
        SEND3<="11111";
        SEND4<="11111";
        SEND5<="11111";
        SEND6<="11111";

WHEN DISPLAYNR=>
         case rand_nr is
         when "001" => send7<="01011"; send6<="01010";
         when "010" => send7<="01100"; send6<="01010";
         when "011" => send7<="01101"; send6<="01010";
         when "100" => send7<="01110"; send6<="00111";
```

```
when "101" => send7<="01111"; send6<="00111";
when "110" => send7<="10000"; send6<="00111";
when others => send7<="11111"; send6<="11111";
end case;
send5<="11111";
send4<="11111";
if seq(3) = '1' then
   send3<="00111";
   elsif seq(3)='0' then
   send3<="01010";
   else send3<="11111";
   end if;
if seq(2) = '1' then
   send2<="00111";
   elsif seq(2)='0' then
   send2<="01010";
   else send2<="11111";
   end if;
if seq(1) = '1' then
   send1<="00111";
```

```
      elsif seq(1)='0' then
      send1<="01010";
      else send1<="11111";
      end if;
    if seq(0) = '1' then
      send0<="00111";
      elsif seq(0)='0' then
      send0<="01010";
      else send0<="11111";
      end if;
WHEN CHECK=>
    if( seq(2 downto 0)=p1) then
      ok1<='1';
      ok2<='0';
     end if;
    if(seq(2 downto 0)= p2) then
      ok2<='1';
      ok1<='0';
    end if;
WHEN DISPLAYWIN=>
```

```
        if ok1='1' then
           send7<="01011";
        elsif ok2='1' then
           send7<="01100";
        end if;
        send6<="00111";
        send5<="00111";
        send4<="00100";
        send3<="00101";
        send2<="00101";
        send1<="10101";
        send0<="00011";
WHEN ED=>
        send7<="10101";
        send6<="00101";
        send5<="01010";
        SEND0<="11111";
        SEND1<="11111";
        SEND2<="11111";
        SEND3<="11111";
```

```
            SEND4<="11111";
      WHEN OTHERS=>
            SEND0<="11111";
            SEND1<="11111";
            SEND2<="11111";
            SEND3<="11111";
            SEND4<="11111";
            SEND5<="11111";
            SEND6<="11111";
            SEND7<="11111";
      END CASE;
END PROCESS;

   --Getting the player inputs
   process(clk)
   begin
      if rising_edge(clk) then
         if current_state<=input then
            p1<=in_p1;
            p2<=in_p2;
```

```
      end if;
   end if;
end process;

-- Getting the random NR at the DICE state;
process(clk)
   begin
   if rising_edge(clk) and current_state=dice then
      if cnt1=5 then
         rand_nr<=randnr;
      end if;
   end if;
end process;

-- Modifying the current sequence!
process(clk)
   begin
   if rising_edge(clk) and current_state=dice then
      if cnt1=5 then
         seq(3 downto 1)<=seq(2 downto 0);
```

```
            if(rand_nr="001" or rand_nr="010" or rand_nr="011") then
               seq(0)<='0';
            elsif(rand_nr="100" or rand_nr="101" or rand_nr="110") then
               seq(0)<='1';
            else
               seq(0)<='U';
            end if;
        end if;
    end if;
   end process;
end Behavioral;
```