Pig game

Lab 4

Matteo Golinelli, Nicolò Vantini, Marcus Gregory

10 December 2025

1 Introduction

The goal of this project is to implement a digital representation of the so-called ”pig game” with an FPGA board. The rules of the game are simple: each player can roll a dice as many times as they want, each time adding the result to their partial score. After each roll, they decide whether to roll the dice again or to end their turn. In the second case, their partial score is added to their total score. If a player rolls a 1, their partial score is vanished and their turn ends immediately. The first player that reaches 100 points wins.

2 Hardware Components and ports

- FPGA development board (Basys 3)
  - Switch 0; to reset the board (see Callout 5 of Fig.1)
  - Push-buttons; to roll the dice, change the player turn and start a new game (see Callout 7 of Fig.1)
  - LEDs; to display the value of the rolled dice (LEDs 15 to 13) and the current player (LEDs 0 and 1) (see Callout 6 of Fig.1)
  - 7-segment display; to display total scores of both player (see Callout 4 of Fig.1)
- Vivado Design Suite; for VHDL design, simulation, implementation, and bitstream generation

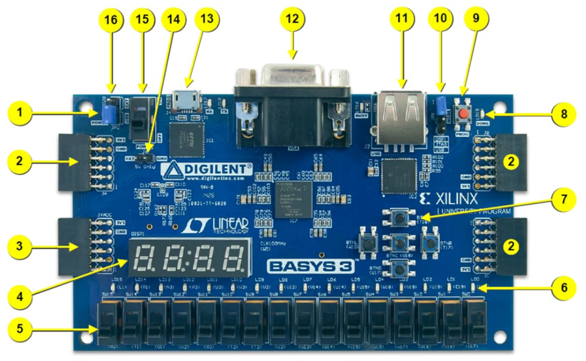

Figure 1. Basys 3 FPGA board with callouts.

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Figure 1: Basys 3 Layout

3 Components

The different components used were mainly provided by professor Velha.

3.1 Control unit

The control unit is the central unit of the entire project. It's a finite state machine which manages the phases of the game, the inputs and the outputs. The rolling of the dice is managed by the fast clock of the FPGA board. In fact, an internal signal cycles between the dice numbers while the *ROLL* button is pressed, and it stops when the button is released. So, a random dice value is obtained every time. After each valid roll (not a 1), the FSM waits for another *ROLL* or the *HOLD* input, which changes the current player and checks if there is a winner for the current game. The FSM is responsible for enabling the registers which store the scores, managing the player turns and changing the current player when a 1 is rolled. When the match ends, the FSM waits for a *NEWGAME* input.

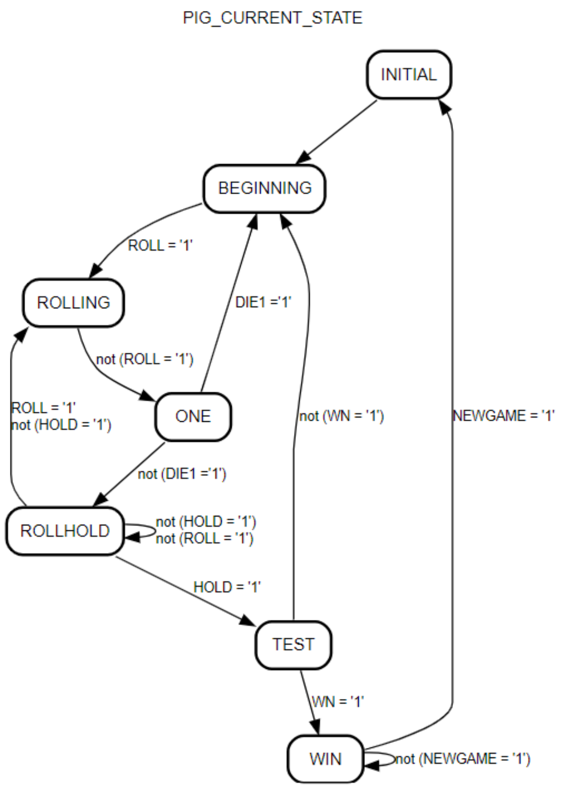

Figure 2: Control unit FSM

3.2 Data path

The data path is a purely combinatory entity which sets various registers based on the inputs given by the control unit. The data path contains registers which store the different scores and the rolled value. It communicates with the control unit about the numbers to be displayed on the 7 segment display, the winner, and if the dice rolled a 1.

3.3 Debouncer

The debouncer is necessary because analog buttons always introduce small oscillations in the signal they produce when pressed or released. A debouncer takes a noisy button input, *BOUNCY*, and converts it to a stable signal. If between each clock cycle the input remains stable a counter is decremented, otherwise the counter is reset to all 1's. When the counter reaches zero, the *DEBOUNCER* signal is set to the input stable signal, and the *PULSE* signal is set to *HIGH* for one clock cycle to notify that the input signal has been stabilized and can be read.

3.4 Seven segment display

The seven segment display needs to be driven by an internal logic because of its implementation on the board. The four digit display can be controlled one digit at a time by four negative active anodes, one per digit. So, the driver shifts between the four anodes and light the correct cathodes for each anode. The shifts are performed in fast sequence (about 100 times per second), such that the human eye is not able to distinguish the different anode changes. In fact, if the display is recorded through a camera, this trick does not work anymore and it is impossible to read the values on the display.
Before the display driver, a BCD component is used on both the player scores to convert binary values contained in the registers into decimal values to be displayed.

3.5 Main (Entity: *piggame*)

The *piggame* entity connects the user inputs (buttons and switches), the clock source, and the output peripherals (LEDs and seven-segment displays) with the internal control and datapath units that implement the game logic. This module manages all board-level interactions. The push buttons are processed through dedicated debouncer blocks for user actions such as *ROLL*, *HOLD*, and *NEWGAME*. These signals are then interpreted by the control unit to drive the evolution of the game. Timing counters are also included to generate slower internal clocks used for display and multiplexing.
Aside from component binding, the main entity contains two processes. The first one is used to light the LED corresponding to the current player, while the second one is used to blink the LEDs when a player wins.

4 Functional Changes

Much of the Pig Game code was already supplied, only requiring a few tweaks. The following is the collection of bug fixes and changes made by the team.

- Corrected check for DIE==1 by checking if DIE==6 due to DIE+=1 increment of the signal that happens at the end of the process
- Imported correct libraries
- Declared signal ports for the `main_pig_game`
- Combined entities in `main_pig_game`
- Added player LED indicator
- Set pins with the .xcd file

5 VHDL Code

5.1 Main Pig Game

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

entity main_pig_game is
    port (
        CLK : in std_logic; --! system clock
        BTN : in std_logic_vector(4 downto 0); --! input buttons
        LED : out std_logic_vector(15 downto 0); --! output to 7-segment display
        ↪  segments
        SSEG_AN : out std_logic_vector(3 downto 0);  --! output to 7-segment display
        ↪  anodes
        SSEG_CA : out std_logic_vector(7 downto 0);  --! output to 7-segment display
        ↪  cathodes
        SW : in std_logic_vector(15 downto 0)  --! input from switches
    );
end entity main_pig_game;

architecture behavioral of main_pig_game is

    -- Components declaration
    component controlunit is
        port(
            clock  : in std_logic; --! Clock
            reset  : in std_logic; --! Reset
            ROLL   : in std_logic; --! button for the roll
            HOLD   : in std_logic; --! button for hold
            NEWGAME: in std_logic; --! button for new game
            ENADIE : out std_logic; --! Enable Die to increment
            LDSU   : out std_logic; --! Add DIE to SUR register
            LDT1   : out std_logic; --! Add SUR to TR1 register
            LDT2   : out std_logic; --! Add SUR to TR2 register
            RSSU   : out std_logic; --! Reset SUR register
            RST1   : out std_logic; --! Reset TR1 register
            RST2   : out std_logic; --! Reset TR2 register
            BP1    : out std_logic; --! enables blinking
            CP     : inout std_logic; --! current player (register outside)
            FP     : inout std_logic; --! First player (register outside)
            DIE1   : in std_logic; --! signal that the die is at one
            WN     : in std_logic --! WIN has been achieved by a player
        );
    end component controlunit;

    component datapath is
        port(
            clock  : in std_logic; --! Clock
            reset  : in std_logic; --! Reset
            ENADIE : in std_logic; --! Enable Die to increment
            LDSU   : in std_logic; --! Add DIE to SUR register
```

```vhdl
        LDT1   : in std_logic; --! Add SUR to TR1 register
        LDT2   : in std_logic; --! Add SUR to TR2 register
        RSSU   : in std_logic; --! Reset SUR register
        RST1   : in std_logic; --! Reset TR1 register
        RST2   : in std_logic; --! Reset TR2 register
        CP     : inout std_logic; --! current player (register outside)
        FP     : inout std_logic; --! First player (register outside)
        DIGIT0 : out std_logic_vector( 3 downto 0 ); --! digit to the right
        DIGIT1 : out std_logic_vector( 3 downto 0 ); --! 2nd digit to the left
        DIGIT2 : out std_logic_vector( 3 downto 0 ); --! 3rd digit to the left
        DIGIT3 : out std_logic_vector( 3 downto 0 ); --! digit to the left
        LEDDIE : out std_logic_vector(2 downto 0); --! LEDs to display the die
        ↪  value
        DIE1   : out std_logic; --! signal that a one has been obtained
        WN     : out std_logic --! WIN has been achieved by a player
    );
end component datapath;

component seven_segment_driver is
    generic (
    size : integer := 20 --! size of the counter 2^20 max
    );
    Port (
    clock  : in std_logic; --! Clock
    reset  : in std_logic; --! Reset
    digit0 : in std_logic_vector( 3 downto 0 ); --! digit to the left
    digit1 : in std_logic_vector( 3 downto 0 ); --! digit number 2 from left
    digit2 : in std_logic_vector( 3 downto 0 ); --! digit number 3 from left
    digit3 : in std_logic_vector( 3 downto 0 ); --! digit uttermost right
    CA     : out std_logic_vector(7 downto 0); --! Cathodes
    AN     : out std_logic_vector( 3 downto 0 ) --! Anodes
    );
end component seven_segment_driver;

component debouncer is
    generic (
        counter_size : integer := 12
    );
    port (
        clock, reset : in std_logic; --! clock and reset
        bouncy       : in std_logic; --! input that can bounce even in less than
        ↪  one clock cycle (the debouncer can be connected to a slow clock)
        pulse    : out std_logic; --! send a pulse as soon as the stable state
        ↪  of the button touch is verified
        debounced: out std_logic --! provide an out that is the stable version
        );
end component debouncer;

-- Signals declaration
signal ENADIE : std_logic; --! Enable Die to increment
signal LDSU   : std_logic; --! Add DIE to SUR register
signal LDT1   : std_logic; --! Add SUR to TR1 register
signal LDT2   : std_logic; --! Add SUR to TR2 register
```

```
    signal RSSU   : std_logic; --! Reset SUR register
    signal RST1   : std_logic; --! Reset TR1 register
    signal RST2   : std_logic; --! Reset TR2 register
    signal BP1    : std_logic; --! enables blinking
    signal CP     : std_logic := '0'; --! current player (register outside)
    signal FP     : std_logic := '0'; --! First player (register outside)
    signal DIE1   : std_logic; --! signal that the die is at one
    signal WN     : std_logic; --! WIN has been achieved by a player
    signal DIGIT0 : std_logic_vector( 3 downto 0 ); --! digit to the right
    signal DIGIT1 : std_logic_vector( 3 downto 0 ); --! 2nd digit to the left
    signal DIGIT2 : std_logic_vector( 3 downto 0 ); --! 3nd digit to the left
    signal DIGIT3 : std_logic_vector( 3 downto 0 ); --! digit to the left
    signal BTN_0_DEBOUNCED : std_logic; --! debounced version of BTN(0)
    signal BTN_1_DEBOUNCED : std_logic; --! debounced version of BTN(1)
    signal BTN_2_DEBOUNCED : std_logic; --! debounced version of BTN(2)
    signal blink_scaler : unsigned(23 downto 0) := (others => '0'); --! scaler for
    ↪   blinking

begin

    -- Components bindings
    control_unit : controlunit
        port map (
            clock   => CLK,
            reset   => SW(0),
            ROLL    => BTN_0_DEBOUNCED,
            HOLD    => BTN_1_DEBOUNCED,
            NEWGAME => BTN_2_DEBOUNCED,
            ENADIE  => ENADIE,
            LDSU    => LDSU,
            LDT1    => LDT1,
            LDT2    => LDT2,
            RSSU    => RSSU,
            RST1    => RST1,
            RST2    => RST2,
            BP1     => BP1,
            CP      => CP,
            FP      => FP,
            DIE1    => DIE1,
            WN      => WN
        );

    data_path : datapath
        port map (
            clock   => CLK,
            reset   => SW(0),
            ENADIE  => ENADIE,
            LDSU    => LDSU,
            LDT1    => LDT1,
            LDT2    => LDT2,
            RSSU    => RSSU,
            RST1    => RST1,
            RST2    => RST2,
            CP      => CP,
            FP      => FP,
            DIGIT0  => DIGIT0,
            DIGIT1  => DIGIT1,
            DIGIT2  => DIGIT2,
            DIGIT3  => DIGIT3,
            LEDDIE  => LED(15 downto 13),
            DIE1    => DIE1,
            WN      => WN
        );

    sseg_driver : seven_segment_driver
        generic map (
            size => 20
        )
        port map (
            clock   => CLK,
            reset   => SW(0),
            digit0  => DIGIT1,
            digit1  => DIGIT0,
            digit2  => DIGIT3,
            digit3  => DIGIT2,
            CA      => SSEG_CA,
            AN      => SSEG_AN
        );

    roll_debouncer : debouncer
        generic map (
            counter_size => 23
        )
        port map (
            clock    => CLK,
            reset    => SW(0),
            bouncy   => BTN(0),
            pulse    => open,
            debounced=> BTN_0_DEBOUNCED
        );

    hold_debouncer : debouncer
        generic map (
            counter_size => 23
        )
        port map (
            clock    => CLK,
            reset    => SW(0),
            bouncy   => BTN(1),
            pulse    => open,
            debounced=> BTN_1_DEBOUNCED
        );

    newgame_debouncer : debouncer
        generic map (
            counter_size => 23
```

```vhdl
        )
        port map (
            clock    => CLK,
            reset    => SW(0),
            bouncy   => BTN(2),
            pulse    => open,
            debounced=> BTN_2_DEBOUNCED
        );

    -- Process that handles the LED indication of the current player
    player : process(CP)
    begin
        LED(0) <= '0';
        LED(1) <= '0';

        if CP = '0' then
            LED(1) <= '1';
        else
            LED(0) <= '1';
        end if;
    end process player;

    -- Process that handles the blinking of the LEDs when a player wins
    blinking : process(CLK)
    begin
        blink_scaler <= blink_scaler + 1;

        if rising_edge(CLK) and (WN = '1') then
            if blink_scaler(23) = '0' then
                LED(12 downto 2) <= (others => '0');
            else
                LED(12 downto 2) <= (others => '1');
            end if;
        end if;
    end process blinking;

end architecture behavioral;
```

5.2 Control Unit

```vhdl
----------------------------------------------------------------------------------
-- Company: University of Trento
-- Engineer: Philippe Velha
--
-- Create Date: 14/12/2023 09:11:40 AM
-- Design Name: datapath
-- Module Name:
-- Project Name: Pig Game
-- Target Devices: Basys 3
-- Tool Versions:
-- Description: implement driver for 7-segment display
--
-- Dependencies: none
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.std_logic_unsigned.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity controlunit is

    port(
        clock  : in std_logic; --! Clock
        reset  : in std_logic; --! Reset
        ROLL   : in std_logic; --! button for the roll
        HOLD   : in std_logic; --! button for hold
        NEWGAME: in std_logic; --! button for new game
        ENADIE : out std_logic; --! Enable Die to increment
        LDSU   : out std_logic; --! Add DIE to SUR register
        LDT1   : out std_logic; --! Add SUR to TR1 register
        LDT2   : out std_logic; --! Add SUR to TR2 register
        RSSU   : out std_logic; --! Reset SUR register
        RST1   : out std_logic; --! Reset TR1 register
        RST2   : out std_logic; --! Reset TR2 register
        BP1    : out std_logic; --! enables blinking
        CP     : inout std_logic; --! current player (register outside)
        FP     : inout std_logic; --! First player (register outside)
        DIE1   : in std_logic; --! signal that the die is at one
        WN     : in std_logic --! WIN has been achieved by a player
    );
end entity controlunit;
```

```
architecture rtl of controlunit is
    --type definition
    type PIG_STATE is (INITIAL, BEGINNING, ROLLING, ONE, ROLLHOLD, TEST, WIN);
    --Signal definition
    signal PIG_CURRENT_STATE : PIG_STATE;

begin
    FSM : process(clock)
    begin
        if rising_edge(clock) then
            if reset = '1' then --chosen this one but could be anything to reset
                -- reset
                PIG_CURRENT_STATE <= INITIAL;
                FP <= '0';
            else
            --default values
                ENADIE <= '0';
                LDSU  <= '0';
                LDT1  <= '0';
                LDT2  <= '0';
                RSSU  <= '0';
                RST1  <= '0';
                RST2  <= '0';
                BP1  <= '0';

            --type PIG_STATE is (INIT, BEGINNING, ROLLING, ONE, ROLLHOLD, TEST,
            ↪   WIN);

                case PIG_CURRENT_STATE is
                when INITIAL =>

                    PIG_CURRENT_STATE <= BEGINNING;

                    RST1 <= '1'; --! reset TR1
                    RST2 <= '1'; --! reset TR2
                    CP <= FP; --! behavioural implmentation


                when BEGINNING =>
                RSSU <= '1';

                    if ROLL = '1' then --! Press the button to roll the dice
                        PIG_CURRENT_STATE <= ROLLING;
                    end if;

                when ROLLING =>

                    if ROLL = '1' then
                        ENADIE <= '1'; --! enables die increment

                    else
                        PIG_CURRENT_STATE <= ONE;
                    end if;

                when ONE =>
                    if DIE1 ='1' then
                        CP <= not CP;
                        PIG_CURRENT_STATE <= BEGINNING;
                    else
                        PIG_CURRENT_STATE <= ROLLHOLD;
                        LDSU <= '1';
                    end if;

                when ROLLHOLD =>

                    if HOLD = '1' then
                        PIG_CURRENT_STATE <= TEST;
                        CP <= not CP;
                        if CP = '1' then
                            LDT1 <= '1';
                        else
                            LDT2 <= '1';
                        end if;
                    else
                        if ROLL = '1' then
                            PIG_CURRENT_STATE <= ROLLING;
                        else
                            PIG_CURRENT_STATE <= ROLLHOLD;
                        end if;
                    end if;

                when TEST =>


                    if WN = '1' then
                        PIG_CURRENT_STATE <= WIN;
                    else
                        PIG_CURRENT_STATE <= BEGINNING;
                    end if;
                when WIN =>

                    BP1 <= '1';

                    if NEWGAME = '1' then
                        FP <= not FP;
                        PIG_CURRENT_STATE <= INITIAL;
                    else
                        PIG_CURRENT_STATE <= WIN;
                    end if;

                end case;

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

5.3 Datapath

```
----------------------------------------------------------------------------------
-- Company: University of Trento
-- Engineer: Philippe Velha
--
-- Create Date: 14/12/2023 09:11:40 AM
-- Design Name: datapath
-- Module Name:
-- Project Name: Pig Game
-- Target Devices: Basys 3
-- Tool Versions:
-- Description: implement driver for 7-segment display
--
-- Dependencies: none
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.std_logic_unsigned.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity datapath is

    port(
        clock  : in std_logic; --! Clock
        reset  : in std_logic; --! Reset
        ENADIE : in std_logic; --! Enable Die to increment
        LDSU   : in std_logic; --! Add DIE to SUR register
        LDT1   : in std_logic; --! Add SUR to TR1 register
        LDT2   : in std_logic; --! Add SUR to TR2 register
        RSSU   : in std_logic; --! Reset SUR register
        RST1   : in std_logic; --! Reset TR1 register
        RST2   : in std_logic; --! Reset TR2 register
        CP     : inout std_logic; --! current player (register outside)
        FP     : inout std_logic; --! First player (register outside)
        DIGIT0 : out std_logic_vector( 3 downto 0 ); --! digit to the right
        DIGIT1 : out std_logic_vector( 3 downto 0 ); --! 2nd digit to the left
        DIGIT2 : out std_logic_vector( 3 downto 0 ); --! 3rd digit to the left
        DIGIT3 : out std_logic_vector( 3 downto 0 ); --! digit to the left
        LEDDIE : out std_logic_vector(2 downto 0); --! LEDs to display the die value
        DIE1   : out std_logic; --! signal that a one has been obtained
        WN     : out std_logic --! WIN has been achieved by a player
    );
```

```
end entity datapath;

architecture rtl of datapath is
    -- definition of constants
    constant frontbits : std_logic_vector(3 downto 0) := (others => '0'); --! bits
    ↪  to be added in front of DIE register for transfer into SUR register
    -- definition of the signals
    signal TR1 : std_logic_vector(6 downto 0) := (others =>'0'); --! Register TR1
    signal TR2 : std_logic_vector(6 downto 0) := (others =>'0'); --! Register TR2
    signal SUR : std_logic_vector(6 downto 0) := (others =>'0'); --! Register SUR
    signal DIE : std_logic_vector(2 downto 0); --! Register DIE register
    signal D   : std_logic_vector(6 downto 0) := (others =>'0'); --! Register D
    signal bcd1 : std_logic_vector(3 downto 0):= (others =>'0'); --! result of
    ↪  conversion of TR1 in bcd
    signal bcd2 : std_logic_vector(3 downto 0):= (others =>'0'); --! result of
    ↪  conversion of TR1 in bcd
    signal bcd3 : std_logic_vector(3 downto 0):= (others =>'0'); --! result of
    ↪  conversion of TR2 in bcd
    signal bcd4 : std_logic_vector(3 downto 0):= (others =>'0'); --! result of
    ↪  conversion of TR2 in bcd
    ----- component definition
    component binbcd
    Port (
        clock  : in std_logic; --! clock
        reset  : in std_logic; --! reset
        bin    : in std_logic_vector(6 downto 0); --! binary 7 bit
        digit0 : out std_logic_vector( 3 downto 0 ); --! digit units
        digit1 : out std_logic_vector( 3 downto 0 ) --! digit ten
    );
    end component;

begin
--------------------------------
    -- instantiate the binary to BCD converter for both player 1 and player 2
    inst_bin2BCD1 : binbcd
        PORT map(
            clock  => clock,
            reset  => reset,
            bin    => TR1,
            digit0 => bcd1, --resulting BCD number- for TR1
            digit1 => bcd2);

    inst_bin2BCD2 : binbcd
        PORT map(
            clock  => clock,
            reset  => reset,
            bin    => TR2,
            digit0 => bcd3, --resulting BCD number- for TR2
            digit1 => bcd4);
    --------------------------------
    Main_process : process(clock,reset) begin
        if reset = '1' then
            -- reset
            DIE <= (others => '0'); --! asynchronous reset
        else
            if rising_edge(clock) then
                if RST1 = '1' then
                    TR1 <= (others => '0');
                end if;
                if RST2 = '1' then
                    TR2 <= (others => '0');
                end if;
                if RSSU = '1' then
                    SUR <= (others => '0');
                end if;
                if LDT1 = '1' then
                    TR1 <= TR1 + SUR;
                end if;
                if LDT2 = '1' then
                    TR2 <= TR2 + SUR;
                end if;
                if ENADIE = '1' then
                    case DIE is
                        when "110" => DIE <= "001";
                        when others => DIE <= DIE +1;
                    end case;
                end if;
                if DIE ="110" then -- Because the process set the signals at the
                 ↪  end, we need to tcheck the previous value of die, which is 6
                    DIE1 <= '1';
                else
                    DIE1 <= '0';
                end if;
                if LDSU = '1' then
                    SUR <= SUR + (frontbits & DIE);
                end if;
                if CP ='1' then
                    D <= TR2;
                else
                    D <= TR1;
                end if;

                if (D > "1100011") then
                    WN <= '1';
                else
                    WN <= '0';
                end if;

            end if;
        end if;

        --! connection to displays
        LEDDIE <= DIE;
        DIGIT0 <= bcd1;
        DIGIT1 <= bcd2;
        DIGIT2 <= bcd3;
```

```
        DIGIT3 <= bcd4;
    end process;

end architecture rtl;
```