Bilkent University Electrical and Electronics Department
EEE102-01 Term Project Report: Basketball Scoreboard

Mehmet Eralp Bulut
22303468

EEE102-1

30 December 2024

Youtube Video: https://youtu.be/cJPCy2p5xHM

1. Introduction

This project aims to design a basketball scoreboard with using BASYS3, VHDL coding, and external devices. External devices are magnetic field and motion sensor.

2. Methodology

Using the BASYS3 FPGA board, VHDL coding, and external sensors including a motion sensor and a magnetic field sensor, the project aims to design and build a basketball scoreboard. The scoreboard is a seven-segment, four-digit display (Figure 1.1) that shows:

- **Most Significant Two Digits (MSD):** Display a countdown timer starting at 60 seconds. This countdown decreases at a 1-second interval until it reaches zero.
- **Least Significant Two Digits (LSD):** Display the basketball player's score, incrementing by either 1 or 5 based on the input from sensors:
    - **Motion Sensor:** Detects the movement of a non-magnetic plastic basketball and increments the score by 1.
    - **Magnetic Field Sensor:** Detects the magnetic field from a magnetized basketball, and increments the score by 5.

When the countdown timer reaches zero, the scoreboard freezes, and all LEDs on the BASYS3 board illuminate until the middle pushbutton is pressed to reset the game. (LEDs' and pushbuttons are indicated by number 6 and 7 in Figure 1.2)

3. Design Specifications

The main components and their functions are:

Active IR Sensor: This motion sensor makes use of a receiver and an infrared emitter (LED). The receiver picks up the reflected signal from objects after the emitter puts out infrared light. This sensor is commonly used for object detection.

KY-003: Magnetic field sensor uses Hall Effect principle to measure magnetic field. It normally outputs 0 from the digital output pin but when it is close to an object that creates a magnetic field it outputs 1.

Four-Digit Seven-Segment Display (Figure 1.1): Four different digits are controlled by the anode selection signals (an_out). The refresh clock ensures smooth transitions between digits.

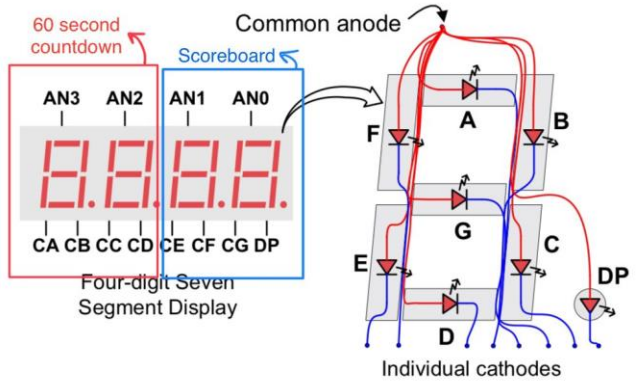


Figure 18. Common anode circuit node.

Figure 1.1 (Four-digit 7 Segment Display)

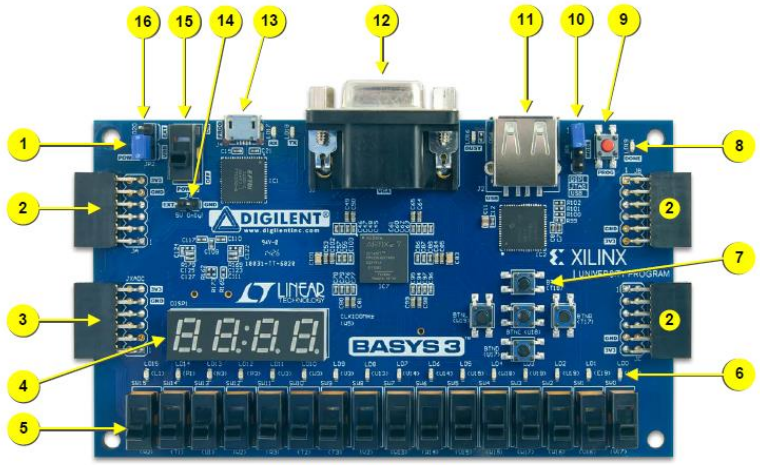


Figure 1. Basys3 FPGA board with callouts.

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod connector(s)	10	Programming mode jumper
3	Analog signal Pmod connector (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

Table 1. Basys3 Callouts and component descriptions.

Figure 1.2 from BASYS 3 FPGA Board Reference Manual

Mehmet Eralp Bulut
22303468

The key components of my VHDL design:

Inputs:

- CLK (100 MHz clock signal)
- RESET
- HALL_SENSOR (Input from the Magnetic Field Sensor which uses Hall Effect Principle)
- MOTION_SENSOR

Outputs:

- an_out (Controls the active anode of the seven-segment display)
- LED_out (Controls the segments of the seven-segment display)
- LIGHTS_OUT (Controls BASYS's LEDs)

Internal Signals:

- seven_segment_value
- countdown_value (Tracks the 60 second countdown timer)
- refresh_clk
- one_second_clk
- freeze
- hall_filtered, motion_filtered (Debounce signals)

Clock Division:

To manage the multiplexing of the seven-segment display, a 1 kHz signal (refresh_clk) is generated from a 100 MHz input clock. Moreover, a 1 Hz clock (one_second_clock) is produced for the countdown timer.

```
-- 1 khz Generate Refresh Clock
process(CLK)
begin
    if rising_edge(CLK) then
        if refresh_div = 99999 then
            refresh_div <= 0;
            refresh_clk <= not refresh_clk;
        else
            refresh_div <= refresh_div + 1;
        end if;
    end if;
end process;

--1 Hz Clock for Countdown
process(CLK)
begin
    if rising_edge(CLK) then
        if second_div = 49999999 then
            second_div <= 0;
            one_second_clk <= not one_second_clk;
        else
            second_div <= second_div + 1;
        end if;
    end if;
end process;
```

Figure 2.1 (Clock Division Process)

Debouncing Logic:

I implemented a debouncing logic because the magnetic field sensor caused the scoreboard to increase unexpectedly. Debouncing logic eliminates the noise from the magnetic field sensor signal, ensuring stable input.

```
--Hall Effect Sensor Input
process(CLK)
begin
    if rising_edge(CLK) then
        if RESET = '1' then
            hall_sensor_sync <= (others => '0');
            debounce_counter <= 0;
            hall_filtered <= '0';
        else
            -- Synchronize input
            hall_sensor_sync(0) <= HALL_SENSOR;
            hall_sensor_sync(1) <= hall_sensor_sync(0);

            -- Debounce logic
            if hall_sensor_sync(1) = hall_filtered then
                if debounce_counter < 100000 then
                    debounce_counter <= debounce_counter + 1;
                end if;
            else
                debounce_counter <= 0;
            end if;

            if debounce_counter = 100000 then
                hall_filtered <= hall_sensor_sync(1);
            end if;
        end if;
    end if;
end process;
```

Figure 3.1 (Debouncing Logic for Magnetic Field Sensor)

I also implemented this debouncing logic to the motion sensor in order to ensure accurate results on the scoreboard.

```
--Motion Sensor Input
process(CLK)
begin
    if rising_edge(CLK) then
        if RESET = '1' then
            motion_sensor_sync <= (others => '0');
            debounce_counter_m <= 0;
            motion_filtered <= '0';
        else
            motion_sensor_sync(0) <= MOTION_SENSOR;
            motion_sensor_sync(1) <= motion_sensor_sync(0);

            if motion_sensor_sync(1) = motion_filtered then
                if debounce_counter_m < 100000 then
                    debounce_counter_m <= debounce_counter_m + 1;
                end if;
            else
                debounce_counter_m <= 0;
            end if;

            if debounce_counter_m = 100000 then
                motion_filtered <= motion_sensor_sync(1);
            end if;
        end if;
    end if;
end process;
```

Figure 3.2 (Debouncing Logic for Motion Sensor)

Scoreboard Counting:

This process checks for the rising edges of the debounced magnetic field and motion sensor signal. When a signal is detected, the score increases by 1 or 5 depending on the type of ball. When a magnetized basketball scores, the magnetic field sensor increments by 4, and the motion sensor increments by 1, for a total of 5. I chose this logic because I encountered synchronization problems when I tried to increment by 5 if both sensors were activated at the same time. This problems occurs because of the asynchronous nature of inputs. (Both sensors operate independently.)

```
--Hall Sensor
if (hall_filtered = '1') and (hall_prev = '0') then
    if seven_segment_value + 4 > 99 then
        seven_segment_value <= 0;
    else
        seven_segment_value <= seven_segment_value + 4;
    end if;
end if;
hall_prev <= hall_filtered;

--Motion Sensor
if (motion_filtered = '1') and (motion_prev = '0') then
    if seven_segment_value + 1 > 99 then
        seven_segment_value <= 0;
    else
        seven_segment_value <= seven_segment_value + 1;
    end if;
end if;
motion_prev <= motion_filtered;
```

Figure 4.1 (Scoreboard Counting Process)

Countdown Timer:

I developed the countdown timer and seven segment codes with using our applications in Lab 5 Seven-Segment Display.

```
-- Countdown Timer Logic
process(one_second_clk, RESET)
begin
    if RESET = '1' then
        countdown_value <= 60;
        freeze <= '0';
        LIGHTS_OUT <= "0000000000000000";
    elsif rising_edge(one_second_clk) then
        if freeze = '0' then
            if countdown_value > 0 then
                countdown_value <= countdown_value - 1;
                LIGHTS_OUT <= "0000000000000000";
            else
                freeze <= '1'; -- Stop the counter when countdown reaches 0
                LIGHTS_OUT <= "1111111111111111";
            end if;
        end if;
    end if;
end process;
```

Figure 5.1 (Countdown Timer Process)

4. Results

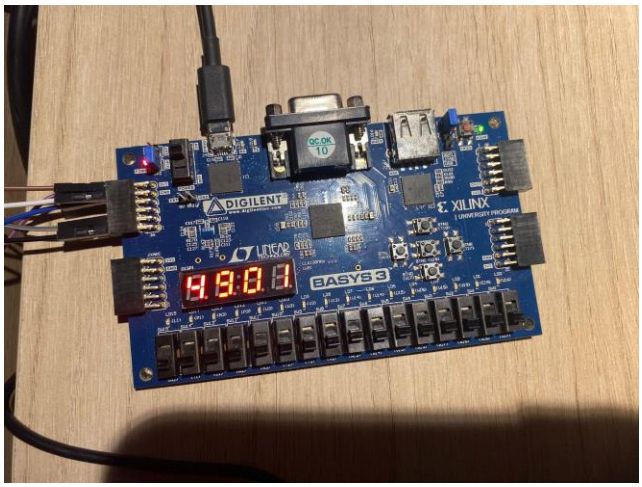


Figure 6.1 (Non-magnetic basketball scores and increments by 1 point)

Mehmet Eralp Bulut
22303468

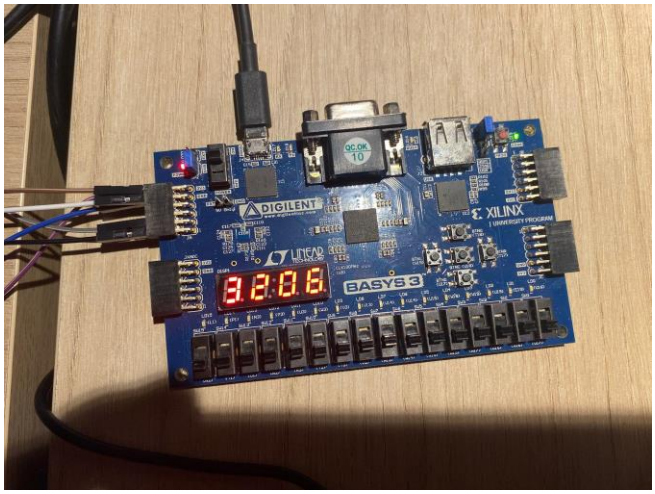


Figure 6.2 (Magnetic basketball scores and increments by 5 points)

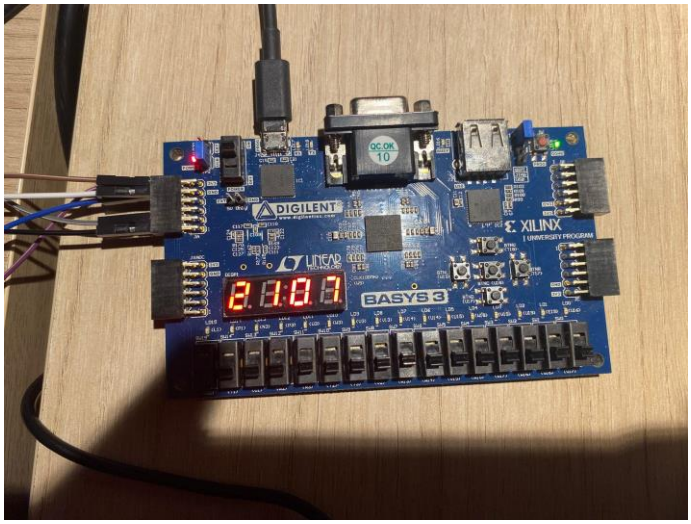
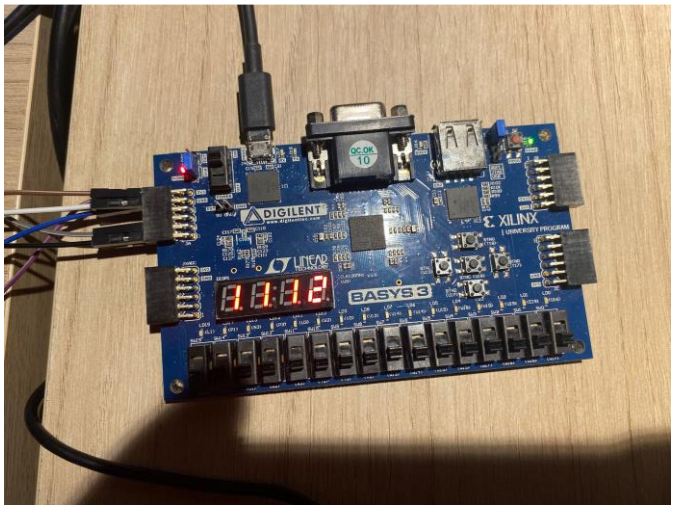


Figure 6.3 (Non-magnetic basketball scores and increments by 1 point)



Mehmet Eralp Bulut
22303468

Figure 6.4 (Magnetic basketball scores and increments by 5 points)

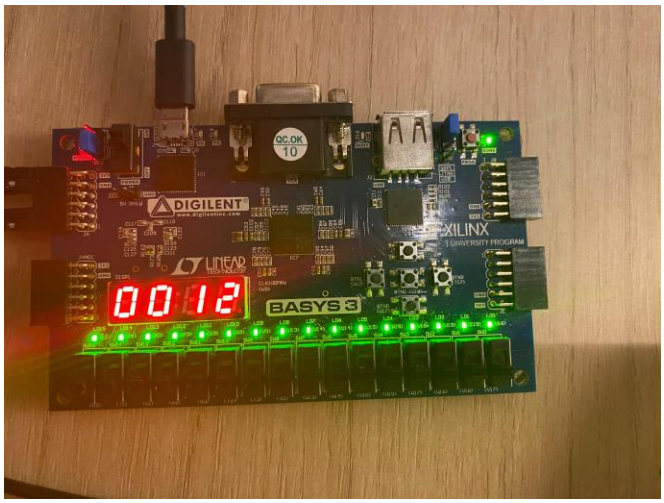


Figure 6.5 (60 second is up, and all the LEDs light up)

5. Conclusion

Using the BASYS3 FPGA board and VHDL, this project successfully created a basketball scoreboard. The BASYS board tracked the score and displayed it on a four-digit, seven-segment display, alongside a countdown timer. This was achieved by combining a motion sensor with a magnetic field sensor.

Managing the noise from the magnetic field sensor was one of the most challenging tasks. The sensor's signal was sensitive to fluctuations, which led to inaccurate scoring. To resolve this issue, I implemented debouncing logic, which helped stabilize the signal and filter out the noise. This approach enhanced the accuracy and reliability of the system by ensuring that the sensors tracked only legitimate inputs.

Overall, this project showed how to design and build a working digital system that handles real-time inputs. In the future, features like tracking multiple players' scores could make it even more useful.

6. References

[1] *Basys3™ FPGA Board Reference Manual*. (2014). www.digilentinc.com

7. Appendices

main.vhd

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

entity BasketballCounter is
    Port (
        CLK      : in  STD_LOGIC;              -- 100 MHz clock
        RESET    : in  STD_LOGIC;              -- Reset signal
        HALL_SENSOR : in  STD_LOGIC;           -- Hall effect sensor input
        MOTION_SENSOR : in  STD_LOGIC;         -- Motion sensor input
        an_out   : out STD_LOGIC_VECTOR (3 downto 0); -- Seven-segment anodes
        LED_out  : out STD_LOGIC_VECTOR (6 downto 0);  -- Seven-segment segments
        LIGHTS_OUT   : out STD_LOGIC_VECTOR (15 downto 0)
    );
end BasketballCounter;

architecture Behavioral of BasketballCounter is

    signal seven_segment_value : integer range 0 to 99 := 0;  -- Display Value(0-99)
    signal an_out_r           : STD_LOGIC_VECTOR(3 downto 0) := "1110";  -- Active anode
    signal selected_digit     : integer range 0 to 9 := 0;  -- Current digit to display
    signal refresh_counter    : integer range 0 to 3 := 0;  -- Counter for display refresh
    signal LED_value          : STD_LOGIC_VECTOR(6 downto 0);  -- Segments LED
    signal refresh_clk        : STD_LOGIC := '0';
    signal refresh_div        : integer range 0 to 99999 := 0; -- Clock divider counter
    signal countdown_value    : integer range 0 to 60 := 60; -- Countdown timer
    signal one_second_clk     : STD_LOGIC := '0';  -- 1 Hz clock for countdown
    signal second_div         : integer range 0 to 49999999 := 0; -- 1-second clock divider
    signal freeze          : STD_LOGIC := '0';

    -- Debouncing Signals for Hall Sensor
    signal hall_sensor_sync   : STD_LOGIC_VECTOR(1 downto 0) := (others => '0');
    signal debounce_counter   : integer range 0 to 100000 := 0;
    signal hall_filtered      : STD_LOGIC := '0';
    signal hall_prev          : STD_LOGIC := '0';

    -- Debouncing Signals for Motion Sensor
    signal motion_sensor_sync  : STD_LOGIC_VECTOR(1 downto 0) := (others => '0');
    signal debounce_counter_m  : integer range 0 to 100000 := 0;
    signal motion_filtered     : STD_LOGIC := '0';
    signal motion_prev         : STD_LOGIC := '0';

begin

    -- 1 khz Generate Refresh Clock
    process(CLK)
```

```
begin
    if rising_edge(CLK) then
        if refresh_div = 99999 then
            refresh_div <= 0;
            refresh_clk <= not refresh_clk;
        else
            refresh_div <= refresh_div + 1;
        end if;
    end if;
end process;

--1 Hz Clock for Countdown
process(CLK)
begin
    if rising_edge(CLK) then
        if second_div = 49999999 then
            second_div <= 0;
            one_second_clk <= not one_second_clk;
        else
            second_div <= second_div + 1;
        end if;
    end if;
end process;

--Hall Effect Sensor Input
process(CLK)
begin
    if rising_edge(CLK) then
        if RESET = '1' then
            hall_sensor_sync <= (others => '0');
            debounce_counter <= 0;
            hall_filtered <= '0';
        else
            -- Synchronize input
            hall_sensor_sync(0) <= HALL_SENSOR;
            hall_sensor_sync(1) <= hall_sensor_sync(0);

            -- Debounce logic
            if hall_sensor_sync(1) = hall_filtered then
                if debounce_counter < 100000 then
                    debounce_counter <= debounce_counter + 1;
                end if;
            else
                debounce_counter <= 0;
            end if;

            if debounce_counter = 100000 then
                hall_filtered <= hall_sensor_sync(1);
            end if;
        end if;
    end if;
end process;

--Motion Sensor Input
process(CLK)
```

```
begin
    if rising_edge(CLK) then
        if RESET = '1' then
            motion_sensor_sync <= (others => '0');
            debounce_counter_m <= 0;
            motion_filtered <= '0';
        else
            motion_sensor_sync(0) <= MOTION_SENSOR;
            motion_sensor_sync(1) <= motion_sensor_sync(0);

            if motion_sensor_sync(1) = motion_filtered then
                if debounce_counter_m < 100000 then
                    debounce_counter_m <= debounce_counter_m + 1;
                end if;
            else
                debounce_counter_m <= 0;
            end if;

            if debounce_counter_m = 100000 then
                motion_filtered <= motion_sensor_sync(1);
            end if;
        end if;
    end if;
end process;

-- Scoreboard Counting Logic
process(CLK)
begin
    if rising_edge(CLK) then
        if RESET = '1' then
            seven_segment_value <= 0;
            hall_prev <= '0';
            motion_prev <= '0';
        elsif freeze = '0' then
            --Hall Sensor
            if (hall_filtered = '1') and (hall_prev = '0') then
                if seven_segment_value + 4 > 99 then
                    seven_segment_value <= 0;
                else
                    seven_segment_value <= seven_segment_value + 4;
                end if;
            end if;
            hall_prev <= hall_filtered;

            --Motion Sensor
            if (motion_filtered = '1') and (motion_prev = '0') then
                if seven_segment_value + 1 > 99 then
                    seven_segment_value <= 0;
                else
                    seven_segment_value <= seven_segment_value + 1;
                end if;
            end if;
            motion_prev <= motion_filtered;
        end if;
    end if;
```

```
    end process;

    -- Countdown Timer Logic
    process(one_second_clk, RESET)
    begin
        if RESET = '1' then
            countdown_value <= 60;
            freeze <= '0';
            LIGHTS_OUT <= "0000000000000000";
        elsif rising_edge(one_second_clk) then
            if freeze = '0' then
                if countdown_value > 0 then
                    countdown_value <= countdown_value - 1;
                    LIGHTS_OUT <= "0000000000000000";
                else
                    freeze <= '1'; -- Stop the counter when countdown reaches 0
                    LIGHTS_OUT <= "1111111111111111";
                end if;
            end if;
        end if;
    end process;

    -- Refresh Counter and Anode Control
    process(refresh_clk)
    begin
        if rising_edge(refresh_clk) then
            refresh_counter <= refresh_counter + 1;
            if refresh_counter = 3 then
                refresh_counter <= 0;
            end if;

            case refresh_counter is
                when 0 =>
                    an_out_r <= "1110";
                    selected_digit <= seven_segment_value mod 10;  -- Ones
                when 1 =>
                    an_out_r <= "1101";
                    selected_digit <= seven_segment_value / 10;  -- Tens
                when 2 =>
                    an_out_r <= "1011";
                    selected_digit <= countdown_value mod 10;  -- countdown
                when 3 =>
                    an_out_r <= "0111";
                    selected_digit <= countdown_value / 10;  -- countdown
                when others =>
                    an_out_r <= "1111";  -- All digits off
                    selected_digit <= 0;
            end case;
        end if;
    end process;

    -- Seven-Segment Decoder
    process(selected_digit)
    begin
        case selected_digit is
```

```
            when 0 => LED_value <= "0000001"; -- 0
            when 1 => LED_value <= "1001111"; -- 1
            when 2 => LED_value <= "0010010"; -- 2
            when 3 => LED_value <= "0000110"; -- 3
            when 4 => LED_value <= "1001100"; -- 4
            when 5 => LED_value <= "0100100"; -- 5
            when 6 => LED_value <= "0100000"; -- 6
            when 7 => LED_value <= "0001111"; -- 7
            when 8 => LED_value <= "0000000"; -- 8
            when 9 => LED_value <= "0000100"; -- 9
            when others => LED_value <= "1111111"; -- Blank (off)
        end case;
    end process;



    -- Outputs
    an_out <= an_out_r;      -- Assign anode
    LED_out <= LED_value;    -- Assign decoded segments

end Behavioral;
```

assign_led.xdc

```
## Clock Signal
set_property PACKAGE_PIN W5 [get_ports {CLK}]
set_property IOSTANDARD LVCMOS33 [get_ports {CLK}]

## Reset Button
set_property PACKAGE_PIN U18 [get_ports {RESET}]
set_property IOSTANDARD LVCMOS33 [get_ports {RESET}]

## Hall Effect Sensor Input
set_property PACKAGE_PIN J1 [get_ports {HALL_SENSOR}]
set_property IOSTANDARD LVCMOS33 [get_ports {HALL_SENSOR}]

## Motion Sensor Input
set_property PACKAGE_PIN L2 [get_ports {MOTION_SENSOR}]
set_property IOSTANDARD LVCMOS33 [get_ports {MOTION_SENSOR}]


## Seven-Segment Display Anodes
set_property PACKAGE_PIN U2 [get_ports {an_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an_out[0]}]

set_property PACKAGE_PIN U4 [get_ports {an_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an_out[1]}]

set_property PACKAGE_PIN V4 [get_ports {an_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an_out[2]}]

set_property PACKAGE_PIN W4 [get_ports {an_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an_out[3]}]
```

```
## Seven-Segment Display Segments
set_property PACKAGE_PIN W7 [get_ports {LED_out[6]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {LED_out[6]}]
set_property PACKAGE_PIN W6 [get_ports {LED_out[5]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {LED_out[5]}]
set_property PACKAGE_PIN U8 [get_ports {LED_out[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {LED_out[4]}]
set_property PACKAGE_PIN V8 [get_ports {LED_out[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {LED_out[3]}]
set_property PACKAGE_PIN U5 [get_ports {LED_out[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {LED_out[2]}]
set_property PACKAGE_PIN V5 [get_ports {LED_out[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {LED_out[1]}]
set_property PACKAGE_PIN U7 [get_ports {LED_out[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {LED_out[0]}]

# Onboard LEDs
set_property PACKAGE_PIN U16 [get_ports {LIGHTS_OUT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[0]}]

set_property PACKAGE_PIN E19 [get_ports {LIGHTS_OUT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[1]}]

set_property PACKAGE_PIN U19 [get_ports {LIGHTS_OUT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[2]}]

set_property PACKAGE_PIN V19 [get_ports {LIGHTS_OUT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[3]}]

set_property PACKAGE_PIN W18 [get_ports {LIGHTS_OUT[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[4]}]

set_property PACKAGE_PIN U15 [get_ports {LIGHTS_OUT[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[5]}]

set_property PACKAGE_PIN U14 [get_ports {LIGHTS_OUT[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[6]}]

set_property PACKAGE_PIN V14 [get_ports {LIGHTS_OUT[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[7]}]

set_property PACKAGE_PIN V13 [get_ports {LIGHTS_OUT[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[8]}]

set_property PACKAGE_PIN V3 [get_ports {LIGHTS_OUT[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[9]}]

set_property PACKAGE_PIN W3 [get_ports {LIGHTS_OUT[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[10]}]

set_property PACKAGE_PIN U3 [get_ports {LIGHTS_OUT[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[11]}]

set_property PACKAGE_PIN P3 [get_ports {LIGHTS_OUT[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[12]}]
```

```
set_property PACKAGE_PIN N3 [get_ports {LIGHTS_OUT[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[13]}]

set_property PACKAGE_PIN P1 [get_ports {LIGHTS_OUT[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[14]}]

set_property PACKAGE_PIN L1 [get_ports {LIGHTS_OUT[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LIGHTS_OUT[15]}]
```