VHDL Design Tools Tutorial – Lab 1

Autonomous ground drone

Students: Wendi Ding, Tatiana Torrico Claure

1. Introduction

The objective of this project is to design an autonomous ground vehicle that has to move parts all around a factory, it has to follow a black line on the ground. The core of this drone is a xc7a35tcpg236-1 cpg236 Xilinx Artix-7 family FPGA which is configured through a micro-USB port. The clock frequency of this system is 50MHz.

To develop this project it is necessary to separate the systems in 4 different subsystems, which are:

- Control Block
- Direction Block
- Motor Speed Block
- Display Block

The input and output signals as well as interconnections between the blocks are shown in Fig.1.

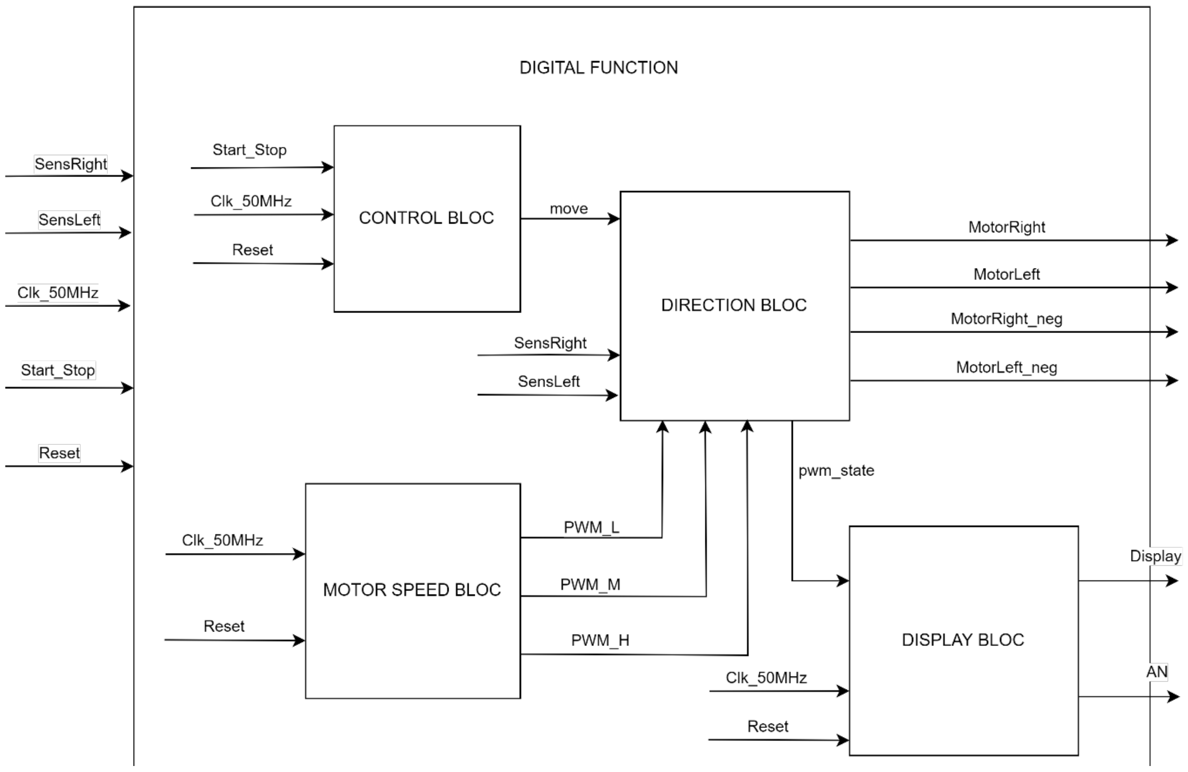


Figure 1: Block Diagram of Digital Functions

2. Inputs and Outputs

2.1. Inputs

There are 4 inputs:

- **SensRight and SensLeft**: The value of the sensor is 1 when it detects black, and 0 when not.
- **Start_Stop and Reset**: The value is 1 when the user presses the button, else it is 0.

- **Clk_50MHz**: A clock with a frequency of 50 MHz.

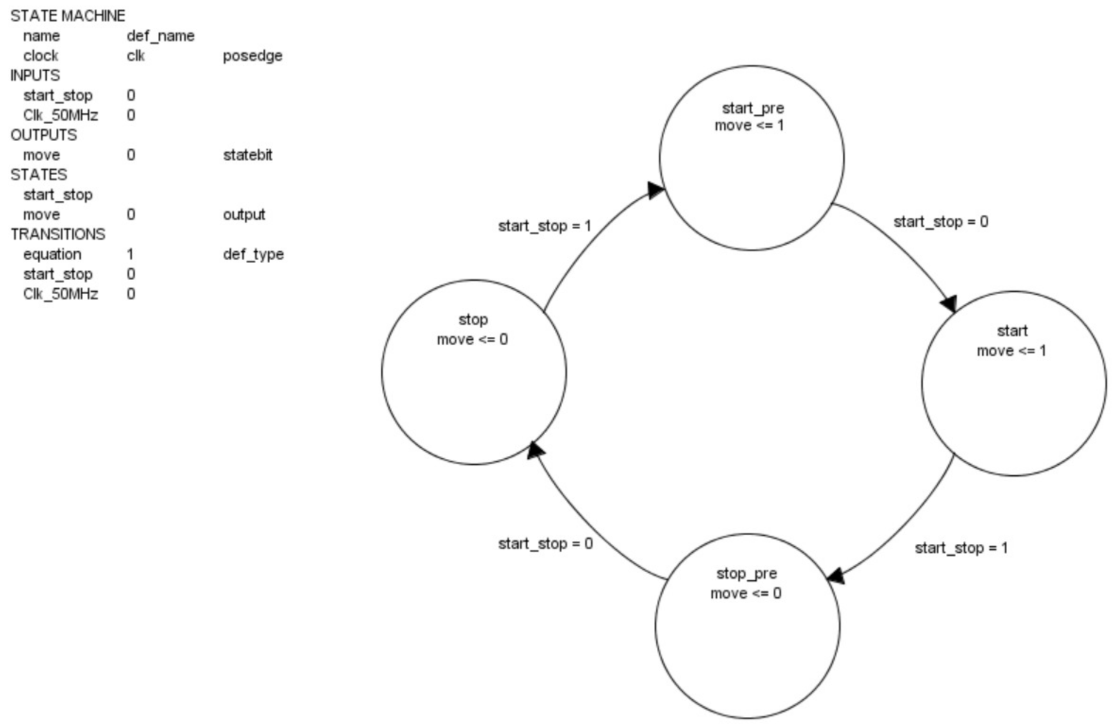
2.2. Outputs

There are 15 outputs:

- **MotorRight and MotorLeft**: Each motor is controlled by a 1-bit signal. The control signal should be at 1 to run the motor.
- **MotorRight_neg and MotorLeft_neg**: Negative output for the motor control signal, which should always be zero throughout the experiment.
- **Display(6 downto 0)**: Seven signals for the 7-segment display. One display has 7 leds, each led controlled by a bit connected onto the anodes of the leds.
- **AN(3 downto 0)**: Four signals to select the 7-segment display.

3. Behavior of the drone

3.1. State machine for Start and Stop



```
STATE MACHINE
  name          def_name
  clock         clk           posedge
INPUTS
  start_stop    0
  Clk_50MHz     0
OUTPUTS
  move          0             statebit
STATES
  start_stop
  move          0             output
TRANSITIONS
  equation      1             def_type
  start_stop    0
  Clk_50MHz     0
```

Figure 2: State Machine-Start and Stop

On this block **Start_Stop** button is implemented. The drone should start when the user presses the Start and Stop push button. When the user releases the button, the drone should continue to move. The drone should stop when the user presses a second time the push button.

Since there is only one button to control 2 behaviors, and the button has 2 states, we can know that for the value Start_Stop=1, the drone moves or stops depending on its previous behavior. This function can be realized with a state machine. With one button (with 2 positions), we have 4 states for the start_stop behavior, which is shown in Fig.2.

The state machine is configured in VHDL following the strategy of Moore's Finite State Machine, as is shown in Fig.3. The FSM is divided into three blocks: Next State Function, FSM Register and Output

Function. Figure 4 shows the codes of Moore's FSM in VHDL which consists of three processes each corresponding to one block. Four states are defined as stop_pre, stop, start_pre and start. The Control Block sends the **move** signal to the Direction Block. The drone moves when its present state is start_pre or start, i.e. move='1', it and stops when its present state is stop_pre or stop, i.e. move='0'.

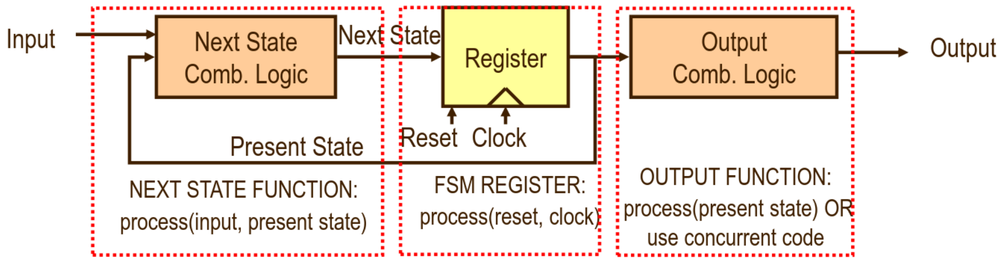

Input
Next State Comb. Logic
Next State
Register
Output Comb. Logic
Output
Present State
Reset
Clock
NEXT STATE FUNCTION: process(input, present state)
FSM REGISTER: process(reset, clock)
OUTPUT FUNCTION: process(present state) OR use concurrent code

Figure 3: Coding Templates for Moore FSM

```
architecture Behavioral of Control_block is
    type ss_state is (stop_pre, stop, start_pre, start); --define 4 states of FSM
    signal pr_state, nx_state: ss_state := stop;

begin

-- section 1: fsm register
    process (Reset, Clk_50MHz)
    begin
        if Reset='1' then
            pr_state <= stop;
        elsif Clk_50MHz'event and Clk_50MHz='1' then
            pr_state <= nx_state;
        end if;
    end process;
-- section 2: next state function
-- when Start_Stop='1', the next state is decided depending on the present state
    process (Start_Stop, pr_state)
    begin
        case pr_state is
            when stop =>
                if Start_Stop='1' then
                    nx_state <= start_pre;
                else
                    nx_state <= stop;
                end if;
            when start_pre =>
                if  Start_Stop='0' then
                    nx_state <= start;
                else
                    nx_state <= start_pre;
                end if;
            when start =>
                if  Start_Stop='1' then
                    nx_state <= stop_pre;
                else
                    nx_state <= start;
                end if;
            when stop_pre =>
                if  Start_Stop='0' then
                    nx_state <= stop;
                else
                    nx_state <= stop_pre;
                end if;
        end case;
    end process;
 -- section 3: output function
 -- The drone moves when its present state is start_pre or start
 -- The drone stops when its present state is stop_pre or stop
    move <= '1' when pr_state=start_pre or pr_state=start else '0';
```

Figure 4: VHDL codes for the CONTROL BLOC

3.2. Speed of the motors

The speed of the motor is controlled by the duty cycle of a PWM (Pulse Width Modulation) signal. Here we choose 50Hz as the frequency of the PWM duty cycle. 3 different levels of speed are used on the drone:

- High speed: The PWM signal is high for 95% duty cycle and low for 5% duty cycle.
- Low speed: The PWM signal is high for 15% duty cycle and low for 85% duty cycle.
- Middle speed: The PWM signal is high for 50% duty cycle and low for 50% duty cycle.

In order to create the PWM, we need to create a counter called signal PWMcnt, that counts from 0 to 99 in 20 ms, see figure 5. The PWMcnt counts +1 every 200µs. So we first create a counter to divide the 50 MHz clock to create a 200µs signal, that counts +1 every 20 ns and counts from 0-9999 every 200µs. Then, by comparing the required value of the duty cycle to the counter PWMcnt, we can generate the appropriate control signal to the motor.

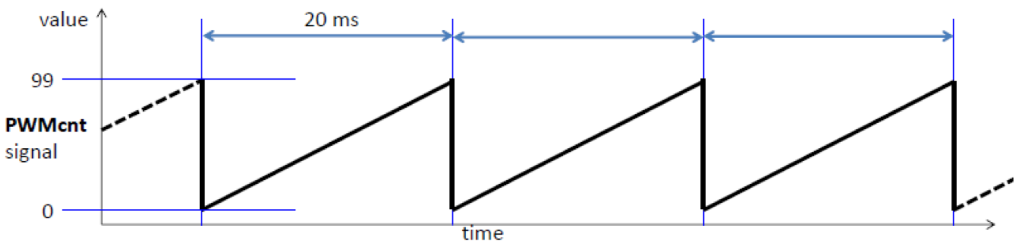
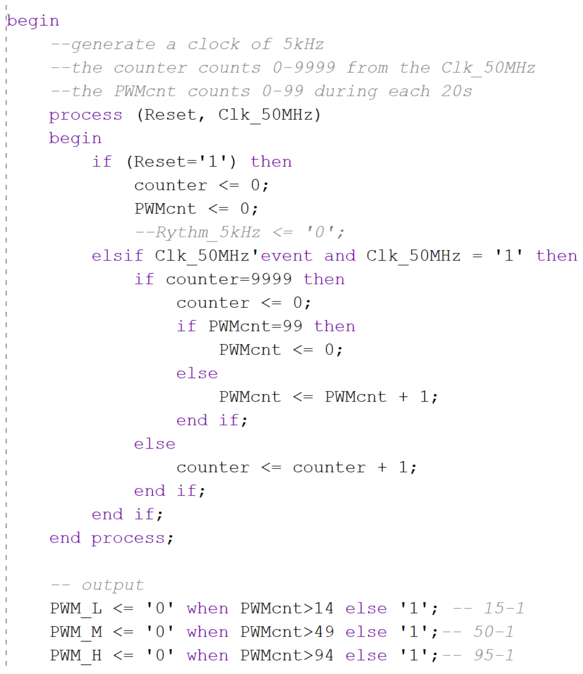


Figure 5: PWM counter from 0 to 99

Figure 6 shows the VHDL codes in which the motor speed is programmed based on the time of each duty cycle. First we configure the clock with the counter signal, then the PWMcnt signal is used to control the three levels of output PWM speeds of the Speed Control Block, which is set according the percentage of the duty cycle. Three output signals (PWM_L, PWM_M, PWM_H) are sent from the Speed Control Bloc to the Direction Block.



```
begin
    --generate a clock of 5kHz
    --the counter counts 0-9999 from the Clk_50MHz
    --the PWMcnt counts 0-99 during each 20s
    process (Reset, Clk_50MHz)
    begin
        if (Reset='1') then
            counter <= 0;
            PWMcnt <= 0;
            --Rythm_5kHz <= '0';
        elsif Clk_50MHz'event and Clk_50MHz = '1' then
            if counter=9999 then
                counter <= 0;
                if PWMcnt=99 then
                    PWMcnt <= 0;
                else
                    PWMcnt <= PWMcnt + 1;
                end if;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    -- output
    PWM_L <= '0' when PWMcnt>14 else '1'; -- 15-1
    PWM_M <= '0' when PWMcnt>49 else '1';-- 50-1
    PWM_H <= '0' when PWMcnt>94 else '1';-- 95-1
```

Figure 6: Coding Motor Speed Block

3.3. Direction Control Bloc

The Direction Control Bloc is used to track the border of the black line. There are no steerable wheels, so the direction is controlled by speed difference between the 2 motors.

Requirements of the Direction are stated as follows:

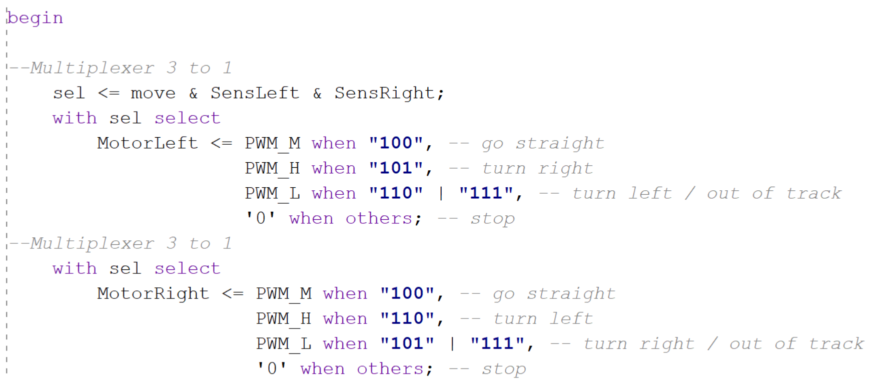
- The direction control depends on the states of the sensors.
- If the right sensor detects the black line (SensRight='1' and SensLeft='0'), the car needs to turn right.
- If the left sensor detects the black line (SensRight='0' and SensLeft='1'), the car needs to turn left.
- If both sensors don't detect the black line (SensRight='0' and SensLeft='0') the car goes straight.
- The turns are made by stopping or decreasing the velocity on one motor and incrementing the velocity of the other motor.

For the Direction block it was used 3 multiplexers, in each one all the possible inputs of the sensors have been configured as the selector signal **sel**. The signal is a standard logic vector (2 downto 0). The first number states if the **move** signal from the Control Bloc is '1' or '0', the second number is the reading of input signal **SensLeft** and the last number corresponds to the input signal **SensRight**.

The logic of turning conditions is configured as follows:

- **Go Straight**: If SensLeft='0' and SensRight='0', we can expect that the sensors are on either side of the line, then the drone should move straight forward with both motor speeds at middle level, i.e. MotorLeft=PWM_M, MotorRight=PWM_M.
- **Turn Right**: If SensLeft='0' and SensRight='1', it means that the right sensor sees the black line, so the drone should turn right to find the border of the black line. The speed difference of the wheels should be used to turn the drone to right. So the output is high speed for the left motor and low speed for the right motor, i.e. MotorLeft=PWM_H, MotorRight=PWM_L.
- **Turn Left**: If SensLeft='1' and SensRight='0', it means that the left sensor sees the black line, so the drone should turn left to find the border of the black line. The speed difference of the wheels should be used to turn the drone to left. So the output is high speed for the right motor and low speed for the left motor, i.e. MotorLeft=PWM_L, MotorRight=PWM_H.
- **Out of Track**: If SensLeft='1' and SensRight='1', then both sensors are looking on the black line. The drone is lost. So the speed of both wheels should decrease until one the sensor detect the border of the line. In this case, we set both motors to low speed, i.e. MotorLeft=PWM_L, MotorRight=PWM_L.

The VHDL codes of two multiplexers for MotorLeft and MotorRight signals are shown in Fig.7 and the logic circuit diagram is shown in Fig.8.



```
begin

--Multiplexer 3 to 1
    sel <= move & SensLeft & SensRight;
    with sel select
        MotorLeft <= PWM_M when "100", -- go straight
                     PWM_H when "101", -- turn right
                     PWM_L when "110" | "111", -- turn left / out of track
                     '0' when others; -- stop
--Multiplexer 3 to 1
    with sel select
        MotorRight <= PWM_M when "100", -- go straight
                      PWM_H when "110", -- turn left
                      PWM_L when "101" | "111", -- turn right / out of track
                      '0' when others; -- stop
```

Figure 7: Coding of Direction Block

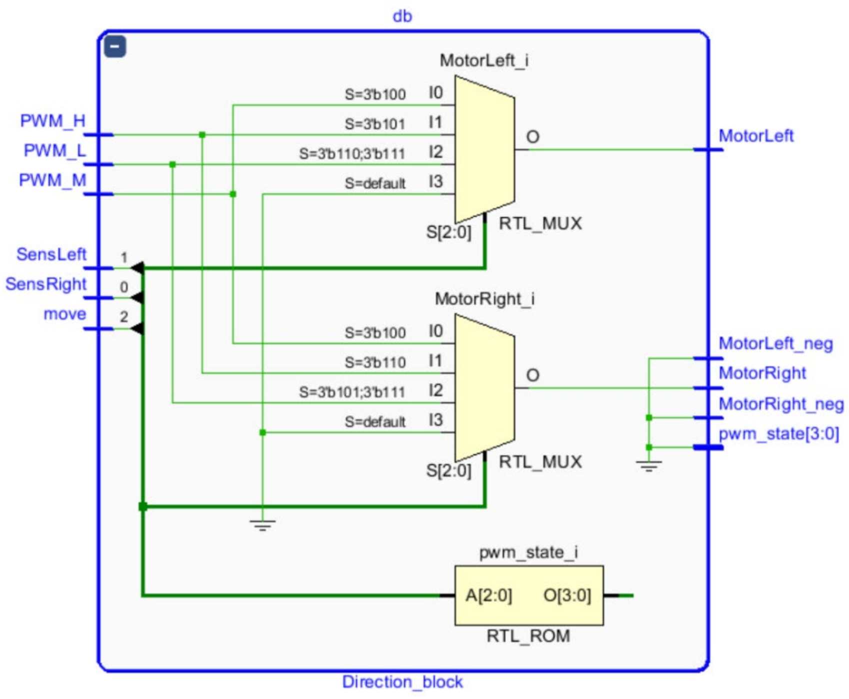


Figure 8: The logic circuit diagram of Direction Block.

3.4. Display Bloc

The duty cycle of the 2 motors is displayed on the 4 7-segment displays. Each motor's duty cycle is displayed with 2 digits, i.e. 2 7-segments.

A total of 4 anode signals, i.e. **AN(3 downto 0)**, are used for the 4 7-segment display. The cathodes remain separated for 1 7-segment, but are common for the 4 displays and are represented by the signal **Display(6 downto 0)**.

In order to display 4 values on the 4 7-segment, the signal Display should circle the 4 values alternatively. The duration for displaying each digit is 1ms and the period of the display circle is 4ms. The signal **AN** should select the proper 7-segment display for each value of Display, as shown in Fig.9.

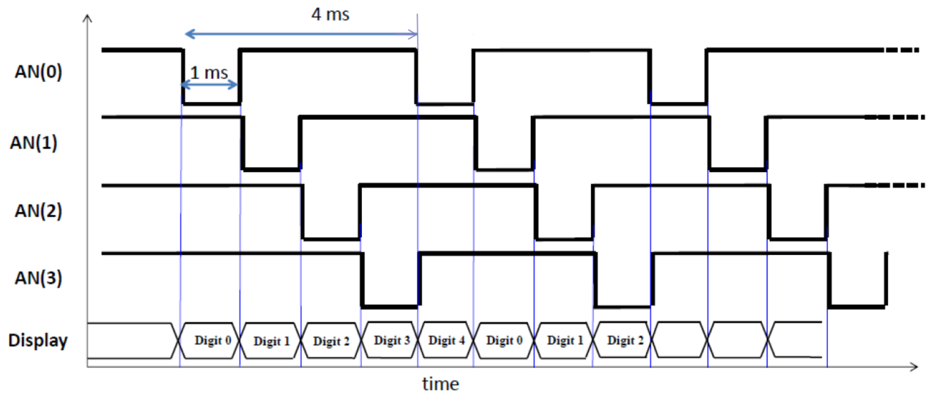


Figure 3.3: Chronogram of Display and AN signals

Figure 9: Chronogram of Display and AN signals

To configure the Display Block, we need a signal named **pwm_state(3 downto 0)** from the Direction Bloc. This signal indicates the current state of the left and right motors, with the first two digits for the left motor and last two digits for the right motor. For example, if pwm_state=”0000”, it means that both motors are stopped. If pwm_state=”1010”, it means that both motors are at PWM_M. If pwm_state=”1101”, it means that the left motor is at PWM_H while the right motor is at PWM_L, which means that the drone is turning right.

To configure the 7-segment display, we need to transfer the integer values to the 7-segment display signals. Fig. 10 shows the multiplexer which determines the Display signal based on required value to be displayed.

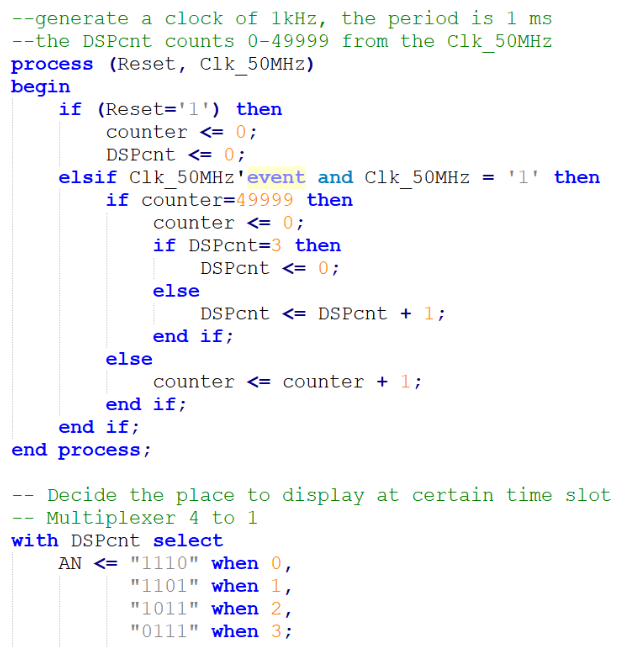
```
entity Display_block is
  Port( Clk_50MHz : in STD_LOGIC;
        Reset : in STD_LOGIC;
        pwm_state : in STD_LOGIC_VECTOR (3 downto 0);
        Display : out STD_LOGIC_VECTOR (6 downto 0);
        AN : out STD_LOGIC_VECTOR (3 downto 0));
end Display_block;

architecture Behavioral of Display_block is

    signal counter: integer range 0 to 49999 := 0; -- counter to divide the clock Clk_50MHz
    signal DSPcnt: integer range 0 to 3 := 0; --count from 0 to 3 in 4 ms
    signal value: integer range 0 to 9 := 0; --count from 0 to 3 in 4 ms
 -- Transfer the integer value to the Seven-Segment Display
 with value select
     Display <= "0000001" when 0,
                "1001111" when 1,
                "0010010" when 2,
                "0000110" when 3,
                "1001100" when 4,
                "0100100" when 5,
                "0100000" when 6,
                "0001111" when 7,
                "0000000" when 8,
                "0000100" when 9;
```

Figure 10: VHDL codes for the Display signal

In order to circle between the anode signals every 1ms, we need to generate a clock of 1 kHz. To do this, we create a counter that divides the 50 MHz clock and counts from 0 to 49999 every 1 ms. Another counter named **DSPcnt** is aimed at counting from 0 to 3 every 4 ms (Fig. 11), so that we can determine which anode to select and which value to display at each time point.

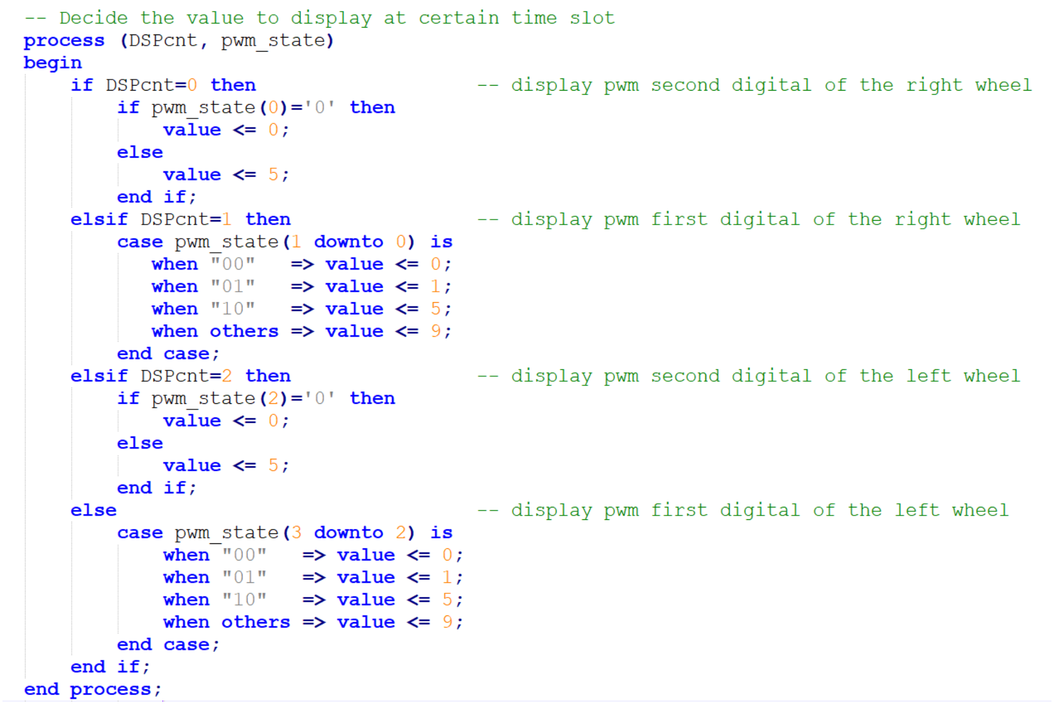


```
--generate a clock of 1kHz, the period is 1 ms
--the DSPcnt counts 0-49999 from the Clk_50MHz
process (Reset, Clk_50MHz)
begin
    if (Reset='1') then
        counter <= 0;
        DSPcnt  <= 0;
    elsif Clk_50MHz'event and Clk_50MHz = '1' then
        if counter=49999 then
            counter <= 0;
            if DSPcnt=3 then
                DSPcnt <= 0;
            else
                DSPcnt <= DSPcnt + 1;
            end if;
        else
            counter <= counter + 1;
        end if;
    end if;
end process;

-- Decide the place to display at certain time slot
-- Multiplexer 4 to 1
with DSPcnt select
    AN <= "1110" when 0,
          "1101" when 1,
          "1011" when 2,
          "0111" when 3;
```

Figure 11: VHDL codes for DSPcnt and AN signals

At last, we can decide which value to display at a certain time slot based on the values of **DSPcnt** and **pwm_state** with the following VHDL codes in Fig.12.

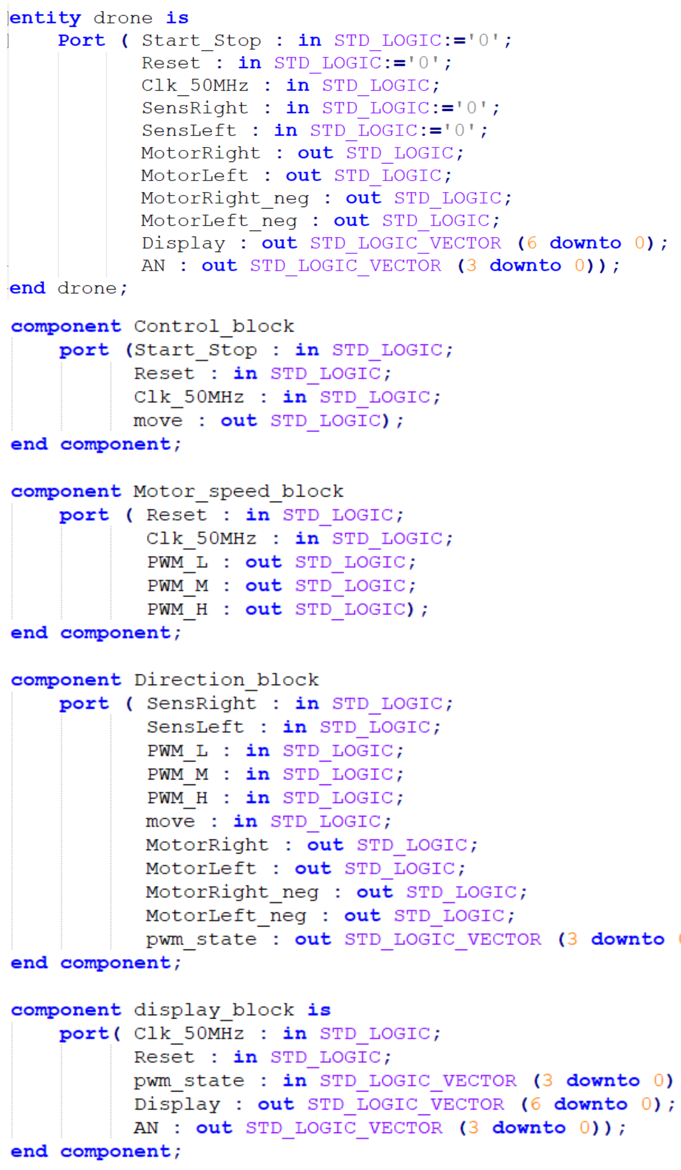


```
-- Decide the value to display at certain time slot
process (DSPcnt, pwm_state)
begin
    if DSPcnt=0 then                    -- display pwm second digital of the right wheel
        if pwm_state(0)='0' then
            value <= 0;
        else
            value <= 5;
        end if;
    elsif DSPcnt=1 then                 -- display pwm first digital of the right wheel
        case pwm_state(1 downto 0) is
           when "00"    => value <= 0;
           when "01"    => value <= 1;
           when "10"    => value <= 5;
           when others => value <= 9;
        end case;
    elsif DSPcnt=2 then                 -- display pwm second digital of the left wheel
        if pwm_state(2)='0' then
            value <= 0;
        else
            value <= 5;
        end if;
    else                                -- display pwm first digital of the left wheel
        case pwm_state(3 downto 2) is
            when "00"    => value <= 0;
            when "01"    => value <= 1;
            when "10"    => value <= 5;
            when others => value <= 9;
        end case;
    end if;
end process;
```

Figure 12: VHDL codes to decide which value to display at a certain time slot

3.5. Drone Overall Configuration

The drone is composed of all the blocks stated before, as shown in Fig. 13. It takes Start_Stop, Reset, Clk_50MHz, SensRight, SensLeft as input signals and takes MotorRight, MotorLeft, MotorRight_neg, MotorLeft_neg, Display and AN as output signal. Figure 14 shows the logic circuit block diagram of the drone generated in Vivado.



```vhdl
entity drone is
    Port ( Start_Stop : in STD_LOGIC:='0';
           Reset : in STD_LOGIC:='0';
           Clk_50MHz : in STD_LOGIC;
           SensRight : in STD_LOGIC:='0';
           SensLeft : in STD_LOGIC:='0';
           MotorRight : out STD_LOGIC;
           MotorLeft : out STD_LOGIC;
           MotorRight_neg : out STD_LOGIC;
           MotorLeft_neg : out STD_LOGIC;
           Display : out STD_LOGIC_VECTOR (6 downto 0);
           AN : out STD_LOGIC_VECTOR (3 downto 0));
end drone;

component Control_block
    port (Start_Stop : in STD_LOGIC;
          Reset : in STD_LOGIC;
          Clk_50MHz : in STD_LOGIC;
          move : out STD_LOGIC);
end component;

component Motor_speed_block
    port ( Reset : in STD_LOGIC;
           Clk_50MHz : in STD_LOGIC;
           PWM_L : out STD_LOGIC;
           PWM_M : out STD_LOGIC;
           PWM_H : out STD_LOGIC);
end component;

component Direction_block
    port ( SensRight : in STD_LOGIC;
           SensLeft : in STD_LOGIC;
           PWM_L : in STD_LOGIC;
           PWM_M : in STD_LOGIC;
           PWM_H : in STD_LOGIC;
           move : in STD_LOGIC;
           MotorRight : out STD_LOGIC;
           MotorLeft : out STD_LOGIC;
           MotorRight_neg : out STD_LOGIC;
           MotorLeft_neg : out STD_LOGIC;
           pwm_state : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component display_block is
    port( Clk_50MHz : in STD_LOGIC;
          Reset : in STD_LOGIC;
          pwm_state : in STD_LOGIC_VECTOR (3 downto 0);
          Display : out STD_LOGIC_VECTOR (6 downto 0);
          AN : out STD_LOGIC_VECTOR (3 downto 0));
end component;
```

Figure 13: VHDL codes for the integrated drone

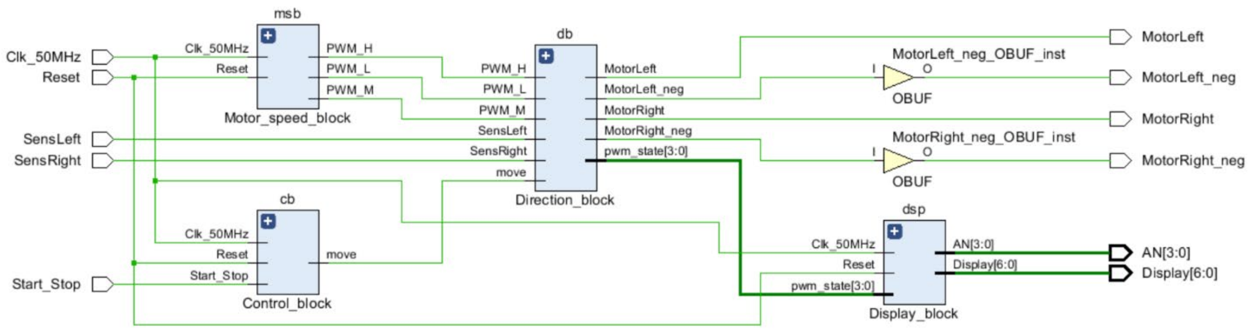


Figure 14: Logic circuit block diagram of the drone generated in Vivado

4. Testbench Configuration and Simulation

4.1. Testbench Configuration

The Testbench is configured in order to simulate the behavior of the drone in different conditions. It is built using the function port map and the input signals are defined in Fig. 15 for the simulation.

```
    component Drone
        port ( Start_Stop : in STD_LOGIC;
               Reset : in STD_LOGIC;
               Clk_50MHz : in STD_LOGIC;
               SensRight : in STD_LOGIC;
               SensLeft : in STD_LOGIC;
               MotorRight : out STD_LOGIC;
               MotorLeft : out STD_LOGIC;
               MotorRight_neg : out STD_LOGIC;
               MotorLeft_neg : out STD_LOGIC;
               Display : out STD_LOGIC_VECTOR (6 downto 0);
               AN : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

begin
     dr: Drone
     port map( Start_Stop => Start_Stop,
               Reset => Reset,
               Clk_50MHz => Clk_50MHz,
               SensRight => SensRight,
               SensLeft => SensLeft,
               MotorRight => MotorRight,
               MotorLeft => MotorLeft,
               MotorRight_neg => MotorRight_neg,
               MotorLeft_neg => MotorLeft_neg,
               Display => Display,
               AN => AN);
        process is
        begin
            wait for 10 ms;
            Start_Stop <= '1';  -- press the start_stop button for 10 ms and release
            wait for 10 ms;
            Start_Stop <= '0';
            SensRight <= '1';  -- turn right
            wait for 50 ms;
            SensLeft <= '1'; -- out of track
            wait for 50 ms;
            SensRight <= '0'; -- turn left
            wait for 50 ms;
            SensLeft <= '0'; -- go straight
            wait for 50 ms;
            Start_Stop <= '1';  -- press the start_stop button for 10 ms and release
            wait for 10 ms;
            Start_Stop <= '0';
            wait for 50 ms;
            Start_Stop <= '1';  -- press the start_stop button for 10 ms and release
            wait for 10 ms;
            Start_Stop <= '0';
            wait for 50 ms;
            Reset <= '1';   -- press the reset button 10 ms and release
            wait for 10 ms;
            Reset <= '0';
            wait;
        end process;
```

Figure 15: VHDL codes for the Testbench

4.2. Control Block Simulation

We first simulated the Control Block and it can be seen in Fig.16 that the output signal move changes each time the **Stat_Stop** button is pressed (Stat_Stop changes from 0 to 1). And when the **Reset** button is pressed, the **move** signal become zero, which is to our expectation.

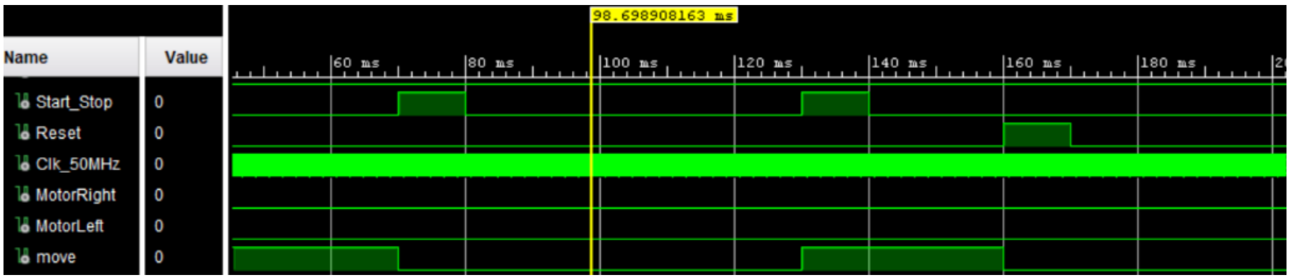
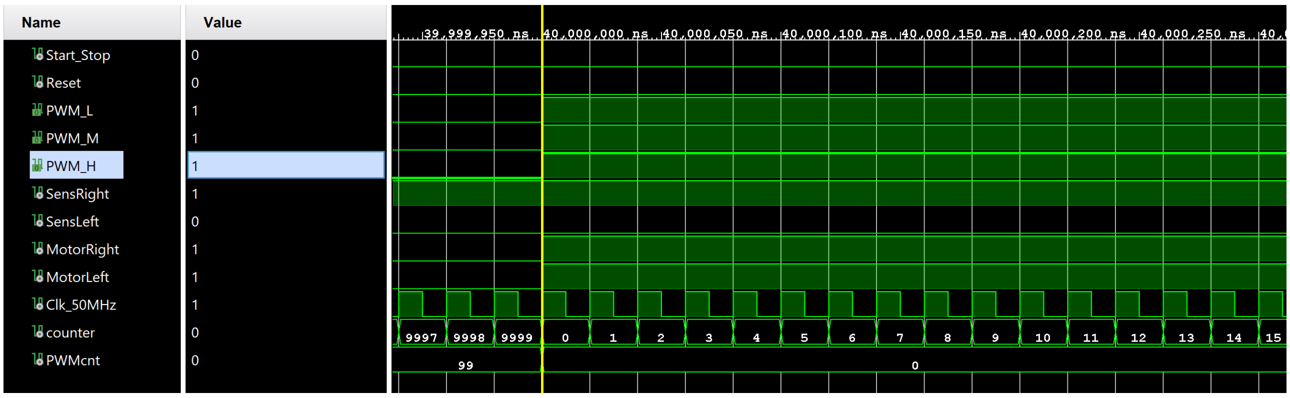


Figure 16: Simulation of the Control Block

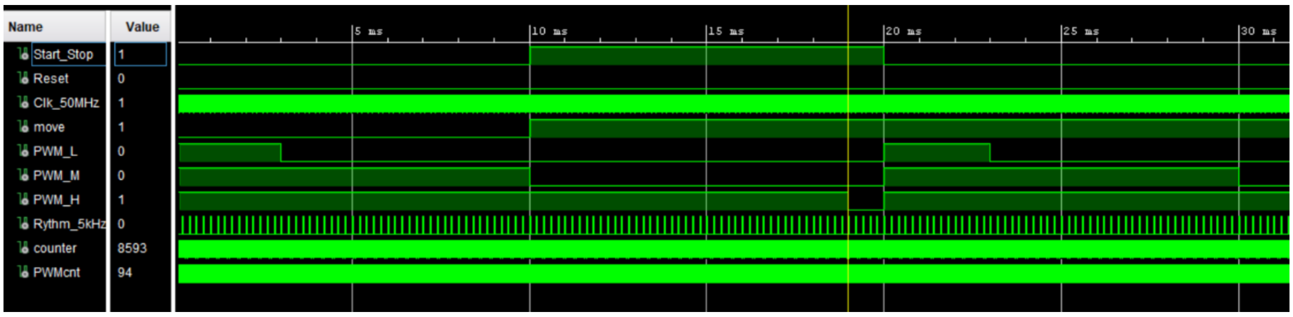
4.3. Motor Speed Block Simulation

On the following simulation results, it is possible to see one of the requirements in nano seconds. The **counter** signal counts +1 on each **Clk_50MHz** clock cycle and goes from 0-9999. When the counter is reset to 0, the **PWMcnt** signal counts +1, and it counts from 0 to 99 in 20 ms. In the next figure it is shown the behavior of the system on the interval of 20 ms.



Name	Value
Start_Stop	0
Reset	0
PWM_L	1
PWM_M	1
PWM_H	1
SensRight	1
SensLeft	0
MotorRight	1
MotorLeft	1
Clk_50MHz	1
counter	0
PWMcnt	0

Figure 17: Motor Speed Block Simulation in ns.

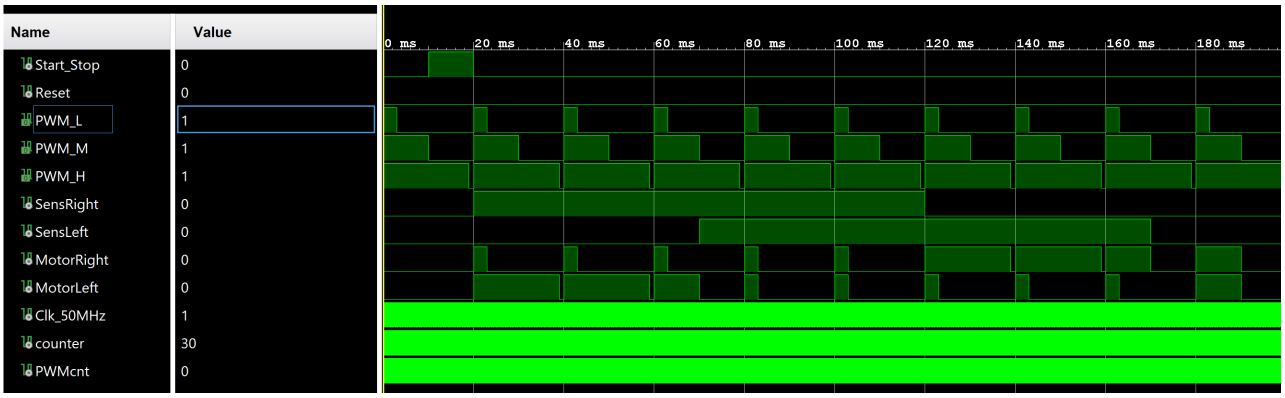


Name	Value
Start_Stop	1
Reset	0
Clk_50MHz	1
move	1
PWM_L	0
PWM_M	0
PWM_H	1
Rythm_5kHz	0
counter	8593
PWMcnt	94

Figure 18: Motor Speed Block Simulation in ms.

4.4. Direction Block Simulation

In the next simulation it is possible to check the behavior of the whole system. It is shown that between 120 ms to 160 ms when the sensor right it is on 0, the turn has to be done to the right, and in that moment the right motor receives the signal for high speed and follows the pattern of the **PWM_H** signal, while the left motor receives the signal for low speed and follows the pattern of the **PWM_L** signal. After 180 ms both sensors don't register the line so that the drone continue straight with a medium speed for both motors, which is to our expectation.



Name	Value
Start_Stop	0
Reset	0
PWM_L	1
PWM_M	1
PWM_H	1
SensRight	0
SensLeft	0
MotorRight	0
MotorLeft	0
Clk_50MHz	1
counter	30
PWMcnt	0

Figure 19: Simulation to check the behavior of the whole system.

4.5. Display Block Simulation

The following simulation shows the correct behavior of the display block. In Fig.20, we can see that the **DSPcnt** signal counts +1 every 1ms and counts from 0 to 3 every 4ms. Correspondingly, the **AN** signal switches the value of '0' between the digits in each display circle, and the value signal to be displayed also changes according to the time slot. For example, from 10ms to 20ms after pressing the **Start_Stop** button, both sensors have '0' values and the drone goes straight forward. Thus the **value** signal switches between '0' '5' '0' '5' and the '0' switches from **AN[0]** to **AN[3]** in each 4ms, and thus the 4 displays will show together "5050" which indicates the **PWM_M** speed level of both motors. Similarly, from 20ms on, **SensRight**='1' and the drone should turn right. Thus the value signal switches between '5' '1' '5' '9' and the '0' switches from **AN[0]** to **AN[3]** in each 4ms, and thus the 4 displays will show together "9515", which indicates the **PWM_H** speed level for **MotorLeft** and **PWM_L** speed level for **MotorRight**. From the simulation, we conclude that the behavior of the displays is to our expection.

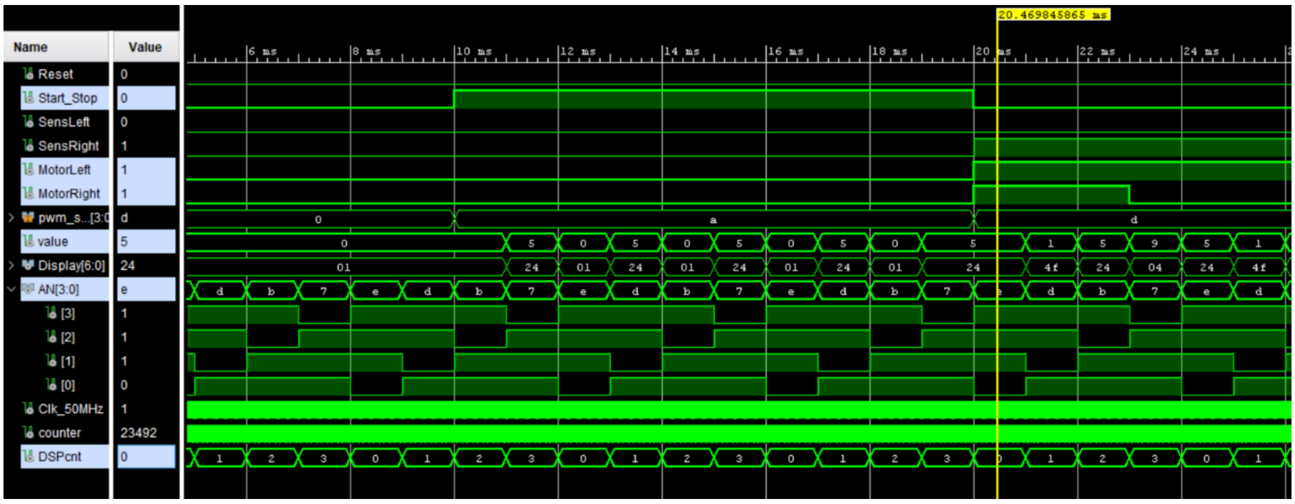
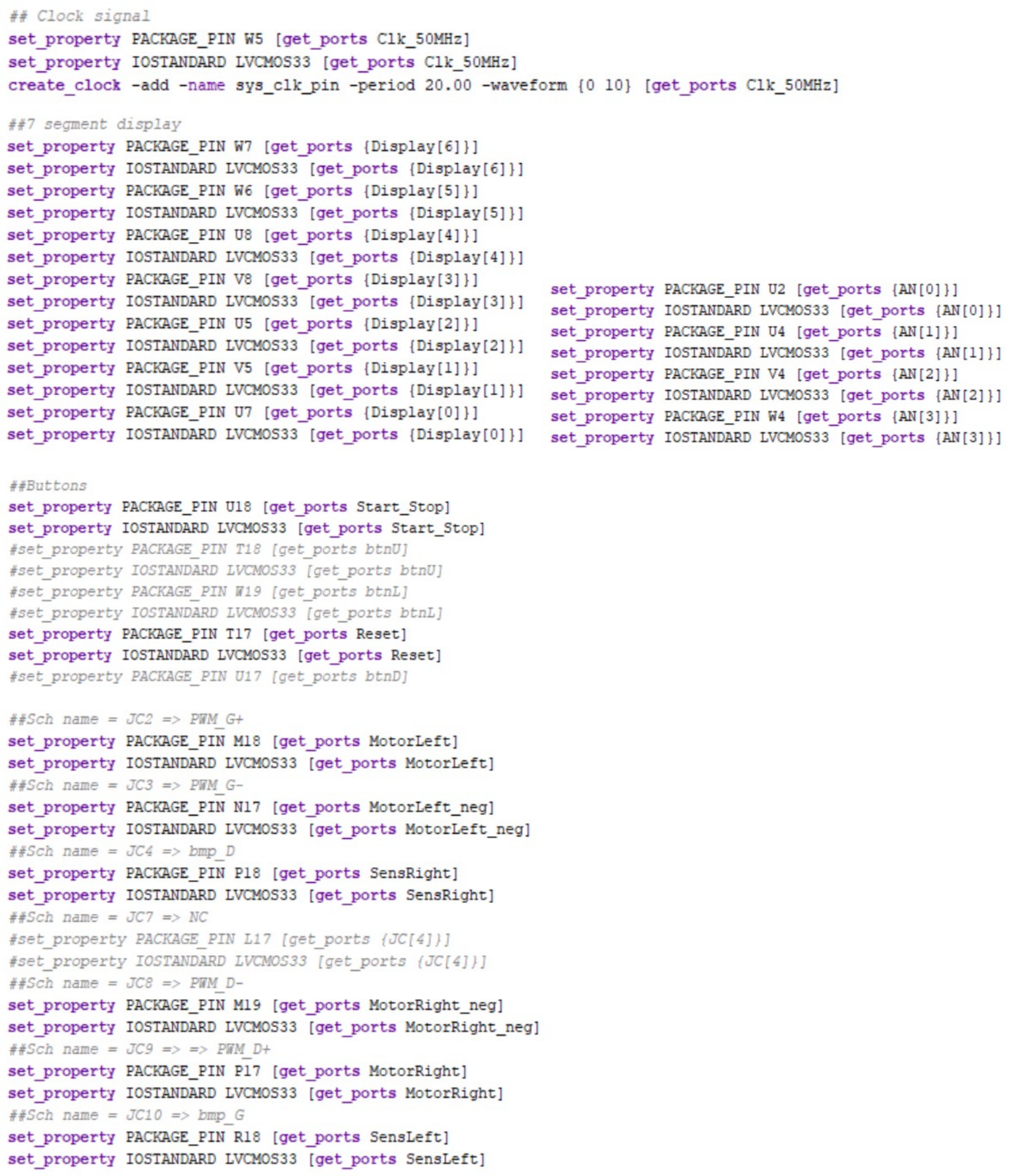


Figure 20: Display Block Simulation

5. Constraint File

Before making the synthesis and implementation in Vivado, it is necessary to generate a constraint file. This file is mandatory to specify which physical pins will be used for the signals and specifies the period of the clock for the Xilinx tool. This constraint is used to place and route the logic elements inside the FPGA while granting that the propagation delay is below the clock period.

The constraint file *drone.xdc* used in our project is defined as follows. As can be seen in Fig.21, we define the clock period as 20ns using the command create_clock, and the waveform is '1' from 0 to 10ns and '0' from 10 to 20ns. The 7 segment display ports are set to the Display and AN signals. The pin U8 is defined as the Start_Stop button and T17 as the Reset button. The signals of the sensors and motors are also configured to the corresponding pins.



```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports Clk_50MHz]
set_property IOSTANDARD LVCMOS33 [get_ports Clk_50MHz]
create_clock -add -name sys_clk_pin -period 20.00 -waveform {0 10} [get_ports Clk_50MHz]

##7 segment display
set_property PACKAGE_PIN W7 [get_ports {Display[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[6]}]
set_property PACKAGE_PIN W6 [get_ports {Display[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[5]}]
set_property PACKAGE_PIN U8 [get_ports {Display[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[4]}]
set_property PACKAGE_PIN V8 [get_ports {Display[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[3]}]
set_property PACKAGE_PIN U5 [get_ports {Display[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[2]}]
set_property PACKAGE_PIN V5 [get_ports {Display[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[1]}]
set_property PACKAGE_PIN U7 [get_ports {Display[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Display[0]}]

set_property PACKAGE_PIN U2 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
set_property PACKAGE_PIN U4 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
set_property PACKAGE_PIN V4 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
set_property PACKAGE_PIN W4 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]

##Buttons
set_property PACKAGE_PIN U18 [get_ports Start_Stop]
set_property IOSTANDARD LVCMOS33 [get_ports Start_Stop]
#set_property PACKAGE_PIN T18 [get_ports btnU]
#set_property IOSTANDARD LVCMOS33 [get_ports btnU]
#set_property PACKAGE_PIN W19 [get_ports btnL]
#set_property IOSTANDARD LVCMOS33 [get_ports btnL]
set_property PACKAGE_PIN T17 [get_ports Reset]
set_property IOSTANDARD LVCMOS33 [get_ports Reset]
#set_property PACKAGE_PIN U17 [get_ports btnD]

##Sch name = JC2 => PWM_G+
set_property PACKAGE_PIN M18 [get_ports MotorLeft]
set_property IOSTANDARD LVCMOS33 [get_ports MotorLeft]
##Sch name = JC3 => PWM_G-
set_property PACKAGE_PIN N17 [get_ports MotorLeft_neg]
set_property IOSTANDARD LVCMOS33 [get_ports MotorLeft_neg]
##Sch name = JC4 => bmp_D
set_property PACKAGE_PIN P18 [get_ports SensRight]
set_property IOSTANDARD LVCMOS33 [get_ports SensRight]
##Sch name = JC7 => NC
#set_property PACKAGE_PIN L17 [get_ports {JC[4]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {JC[4]}]
##Sch name = JC8 => PWM_D-
set_property PACKAGE_PIN M19 [get_ports MotorRight_neg]
set_property IOSTANDARD LVCMOS33 [get_ports MotorRight_neg]
##Sch name = JC9 => => PWM_D+
set_property PACKAGE_PIN P17 [get_ports MotorRight]
set_property IOSTANDARD LVCMOS33 [get_ports MotorRight]
##Sch name = JC10 => bmp_G
set_property PACKAGE_PIN R18 [get_ports SensLeft]
set_property IOSTANDARD LVCMOS33 [get_ports SensLeft]
```

Figure 21: The constraint file *drone.xdc*

6. Test

After configuring the FPGA with the programming files, we did the appropriate tests for validation. On the real time test made on the laboratory, it was possible to corroborate the functioning of the design with the start and stop, reset, turn left and right, track following, and the display function. The video of the real-time testing together with the VHDL codes can be found through the link: https://nextcloud.isae.fr/index.php/s/KoCLTLjzdqKXKwr, and the VHDL codes for the whole project can be found through the link: https://nextcloud.isae.fr/index.php/f/426345918.

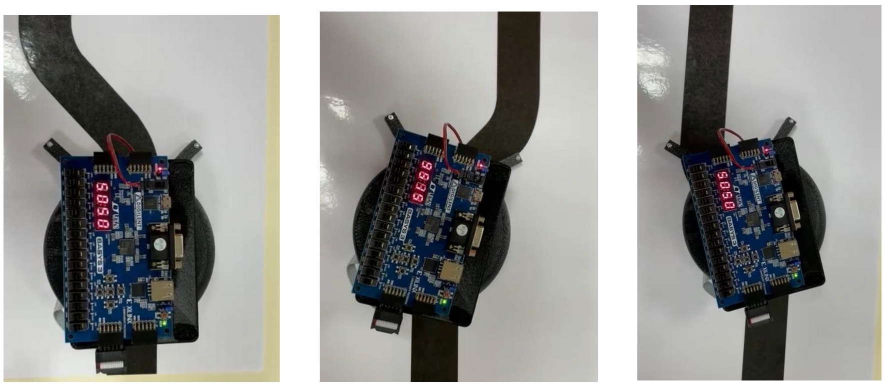


Figure 22: Real-time Testing