VHDL Design Tools Tutorial: Autonomous ground drone

Hugo DAGE

November 16, 2022

1 Introduction

For this BE session, we were asked to design the hardware architecture of a drone to make it follow a defined path. To perform it, our drone is equipped with two motorized wheels, two digital infrared sensors, and a Basys 3™ FPGA Board based on Artix®-7 FPGA by Xilinx. The board has various switches, a Four-digit 7-segment display, and several USB ports. We will use VIVADO 2018 to code our design and simulate its signals.

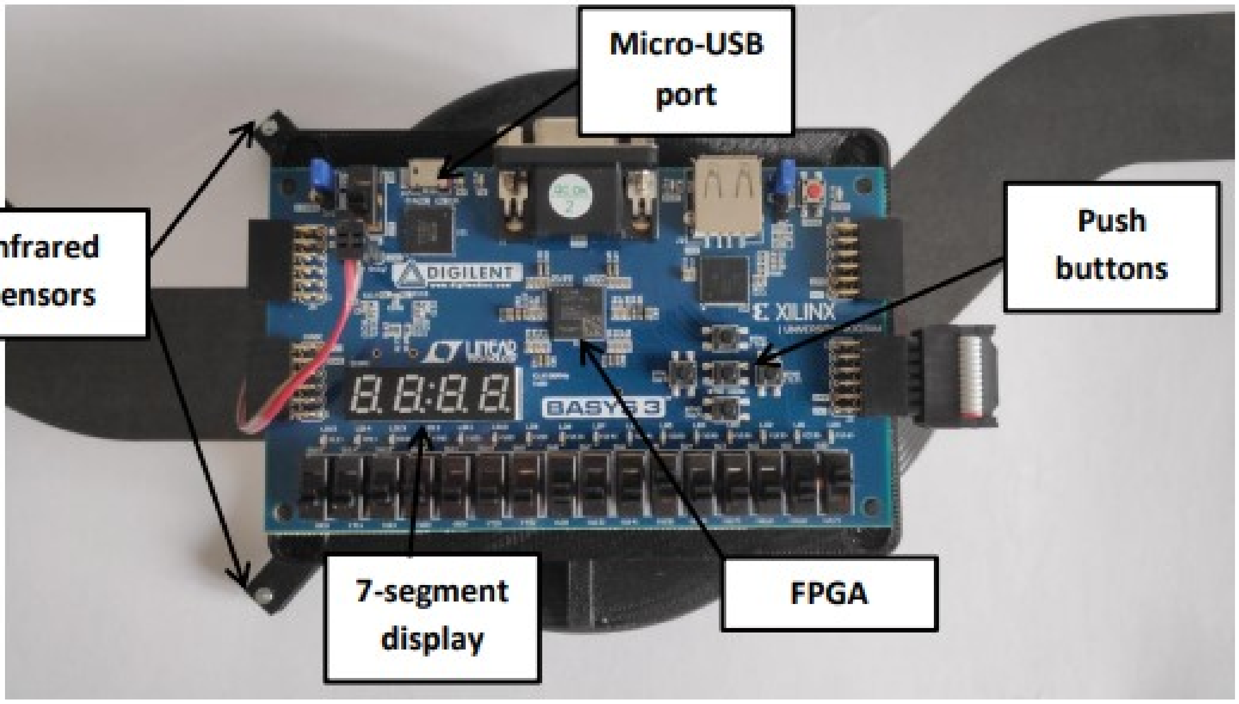


Figure 1: Picture of the drone

2 Specifications

To perform the task of following a defined path, we have to use defined inputs: a 100MHz clock, A start/stop press button, a Reset Press Button, and the two infrared sensors located at the front of the drone, on each side of it. The aim is to control two motors linked to the wheels to give the drone the right direction such that it follows the given black line. The outputs will be the signals we will apply to these motors. For our application, we only need to move forward, we can do it using only two of the outputs signals: `MotorRight_pos` and `MotorLeft_pos`

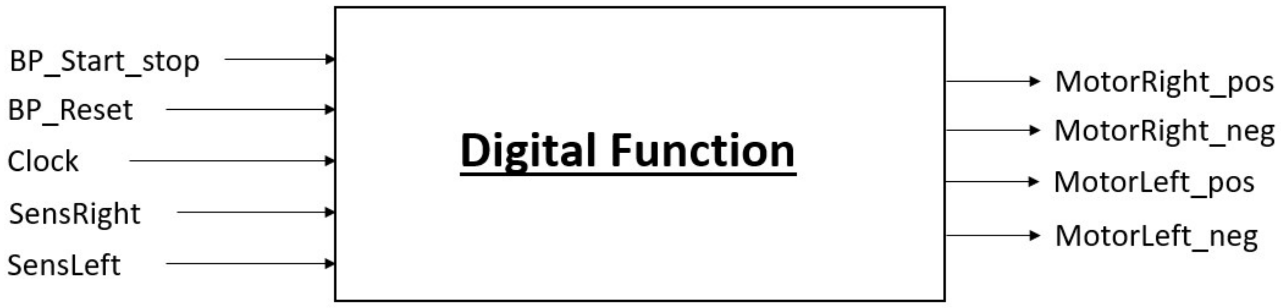


Figure 2: Generic Bloc diagram of the digital function

3 Digital Function Description

To simplify the implementation of our design, we have separated it into three internal blocs with distinct combinatorial logic. There is a control bloc, which enables or disables the movement of the drone if some buttons are pressed; a speed bloc, which generates PWM pulses to send to the motors with different duty cycles; and the direction bloc which, by analyzing data sent by sensors, choose the right signal to send to the engines to give the drone the right direction. Figure 3 is a block diagram of our digital function with those three internal blocs.

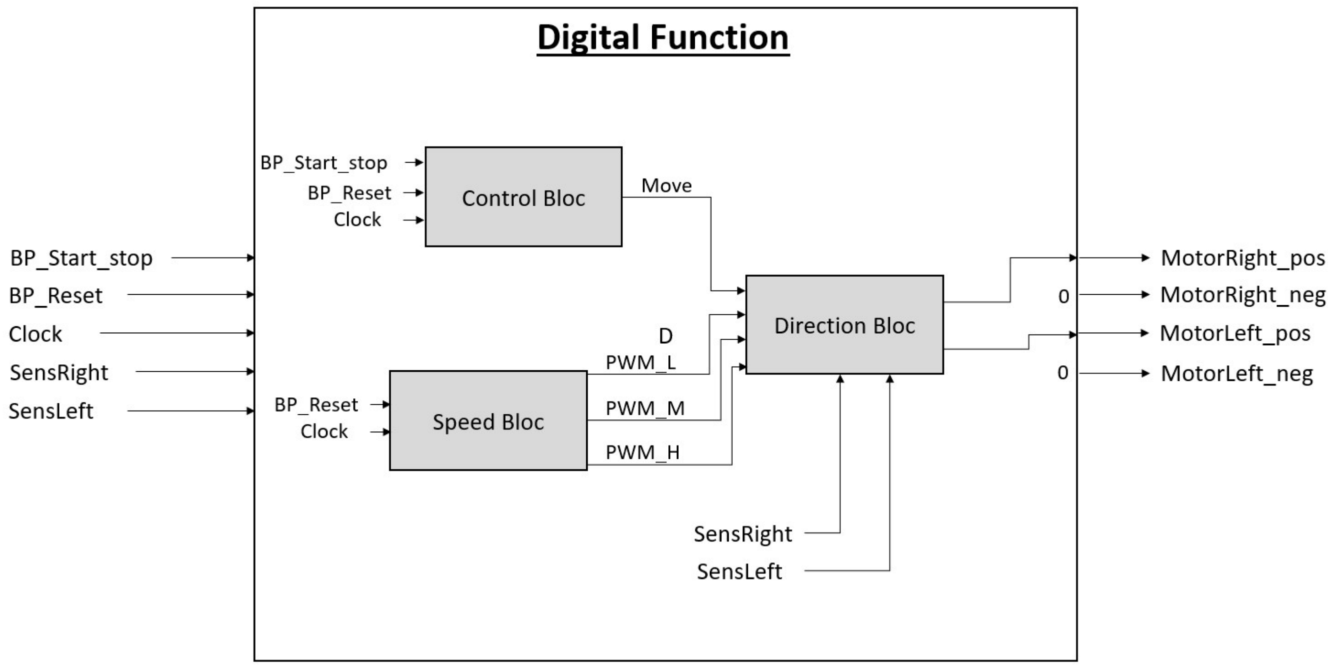


Figure 3: Block diagram of the digital function with 3 internal blocs

3.1 Control Bloc

The control bloc has three inputs: two buttons (`BP_Start_Stop` and `BP_Reset`) and the `Clock`. The aim of the combinatorial logic inside is to detect when a button is pressed to produce an output signal: `Move` which has to be equal to 1 to allow the drone to move and 0 to stop it. This process is synchronous, so with each rising edge of the clock, the move signal will be updated. The period of this clock is 10ns; a human will press the button during several periods, which can lead to unexpected behavior. For that reason, we have implemented a state machine with four states as described: The first state A0 is reached when the drone is not moving and no button is pressed or the button reset has been the last pressed before; The state M0 is reached from A0 when `BP_Start_Stop` is pressed but not yet released; the state M1 occurs when the device is in state A0 and the `BP_Start_Stop` is released;

Finally, the state A1 is reached from M1 when the `BP_Start_Stop` is pressed again. See Figure 4 the graph that illustrates the state machine.

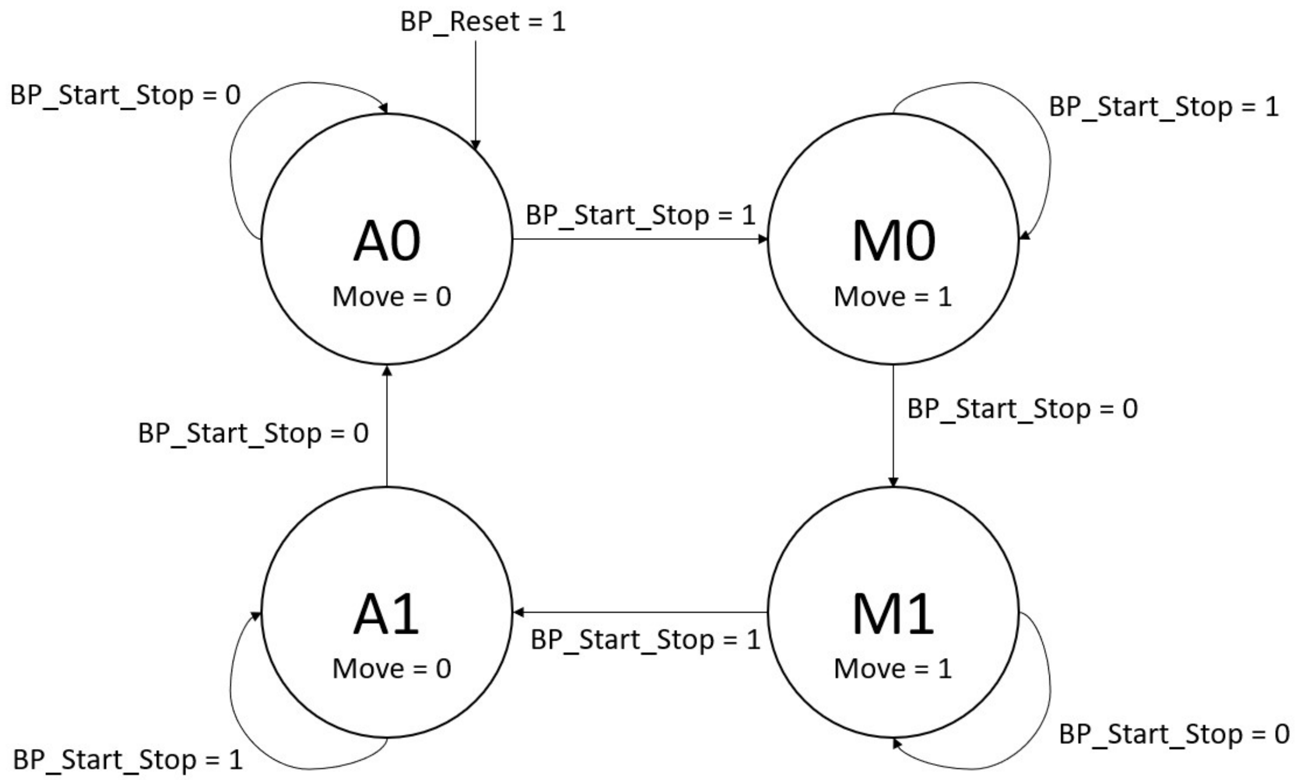


Figure 4: Diagram of the state machine designed for control bloc

From a logical point of view, a Moore state machine itself is compounded by 3 blocs: The next state function which computes the next state depending on the inputs and the current state; The register which actualizes the current state at each rising edge of the clock, and the output function which modify output signal depending on the current state.

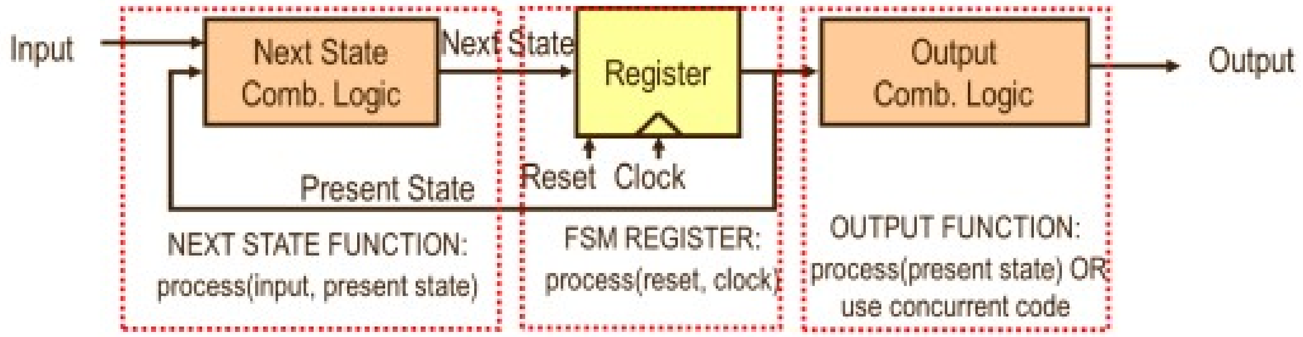

Input
Next State Comb. Logic
Next State
Register
Output Comb. Logic
Output
Present State
Reset
Clock
NEXT STATE FUNCTION: process(input, present state)
FSM REGISTER: process(reset, clock)
OUTPUT FUNCTION: process(present state) OR use concurrent code

Figure 5: Moore State Machine

With this design, our control bloc allows commanding the drone by pressing the buttons. It also prevents undesirable events to occur, such as starting and stopping the drone with only one push on the `BP_Start_Stop` lasting several clock pulses.

3.2 Speed Bloc

The role of the speed bloc is to generate three different PWM signals with a fixed duty cycle and a frequency of 50Hz. The outputs are `PWM_L`;`PWM_M`;`PWM_H` with respectively duty cycles at 15, 50 and 95 percent for low, middle, and high speed.

To do so, we have created a pulse signal with a period 5kHz. Being 100 times bigger than the frequency of the PWM, we will have the possibility to set the duty cycles of our signals with a hundredth resolution. Once we have generated this `5kHz_rhythm` pulse with the clock, we will increment a `counter` at each rising edge of it, from 0 to 99. When this counter's value is 99, 0 will be assigned to it at the next rising edge. At this moment, we put our 3 PWM signals to the value 1. When the counter reaches 15, we put the `PWM_L` at 0 so that his duty cycle is 15 percent. We do the same for the `PWM_M` signal at 50 and for the `PWM_H` at 95. By doing so, the three PWM signals will have the frequency and the duty cycles required.

3.3 Direction Bloc

The direction bloc will assign the PWM signals created before to the motors : `MotorLeft_pos` and `MotorRight_pos` depending on three inputs. These inputs are the sensors values, `SensLeft` and `SensRight`, and the `Move` coming from the control bloc. The structure needed is basically a Multiplexer.

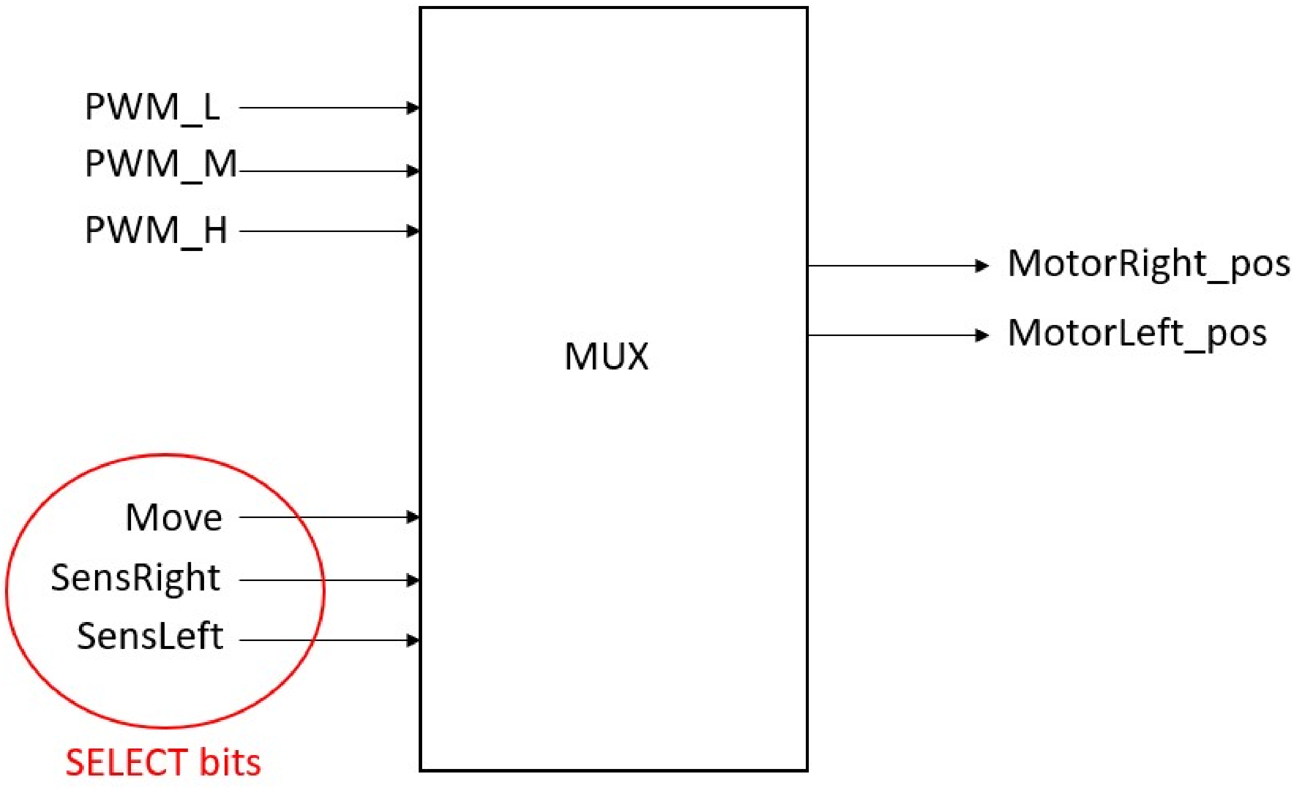


Figure 6: Structure of the Direction bloc

To allow our drone to follow the path, we have to apply the right signal to the motors depending on the sensor's data. When both sensor signals are equal to 0, that means the black line must be between them, so the drone can move purely forward, with the same speed on the two motors. When one of the sensors sees the black line, it sends back a high signal, which means that the drone is exiting the path and has to turn. For example, if the right sensor sends back a high value, the robot will have to turn to the right to correct its trajectory. To do that, the left motor has to be faster than the right one until this trajectory is corrected and the two sensors send back 0 values. We've elaborated a table representing the output depending on the inputs. These choices are not unique: we have decided to go high speed when moving forward and relatively low speed when turning. When the two sensors send back 1s, the drone is lost so we've decided to stop it in this case.

SensRight	SensLeft	MotorRight_pos	MotorLeft_pos
0	0	PWM_H	PWM_H
0	1	PWM_M	PWM_L
1	0	PWM_L	PWM_M
1	1	0	0

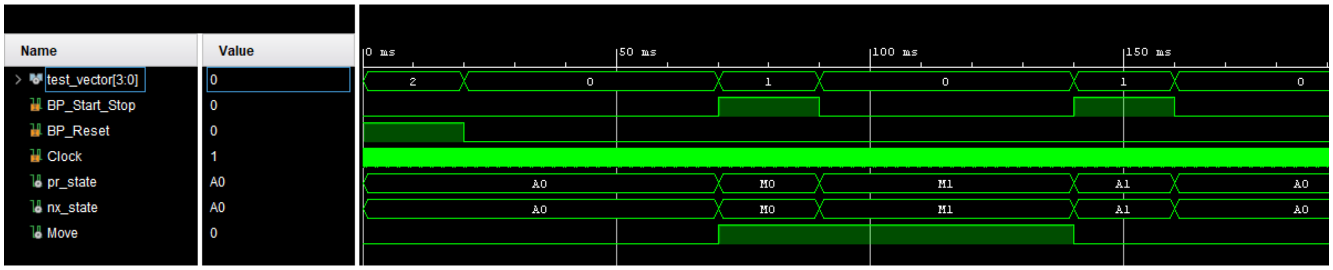
Table 1: Truth table implemented on our multiplexer

4 Simulations

Before synthesizing our design in the FPGA, we simulated our system to find out if the output of each of our blocs were coherent with several possible inputs.

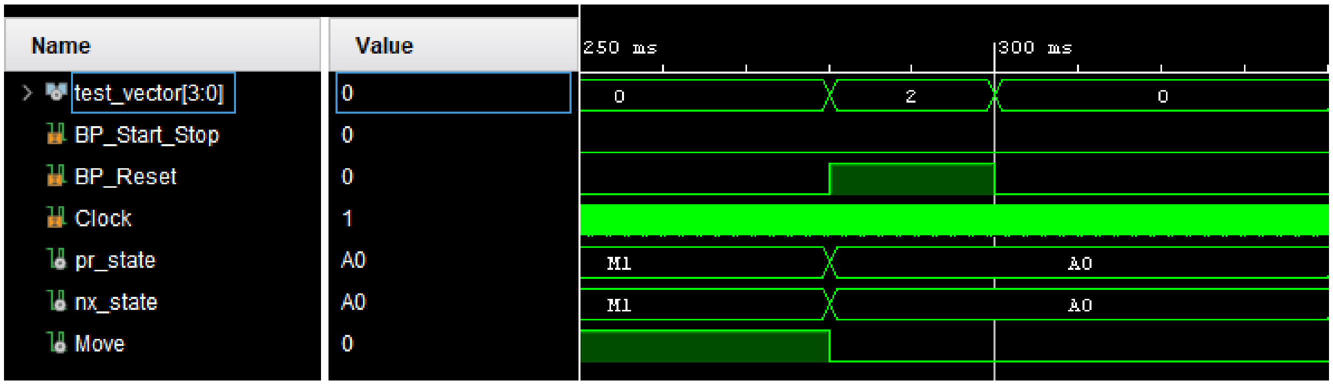
4.1 Simulation of the Control Bloc

First, we have simulated the `Move` signal in functions of different `BP_Start_Stop` and `BP_Reset` which are our inputs signals : Figures 7, 8, 9 are the results of our first simulation. We see that each time the Button Start/stop is pressed or released, the state changes as wished following the sequence $A0 \rightarrow M0 \rightarrow M1 \rightarrow A1 \rightarrow A0$ while this button is pressed and released twice. A push on the Reset button changes the state to A0.



Name	Value
test_vector[3:0]	0
BP_Start_Stop	0
BP_Reset	0
Clock	1
pr_state	A0
nx_state	A0
Move	0

Figure 7: Simulation of the control bloc : usual transition cycle



Name	Value
test_vector[3:0]	0
BP_Start_Stop	0
BP_Reset	0
Clock	1
pr_state	A0
nx_state	A0
Move	0

Figure 8: Simulation of the control bloc : usual transition cycle

If we zoom in, we can see that the current State is delayed by 1 clock pulse from the next state, which is normal behavior for a Moore state Machine.

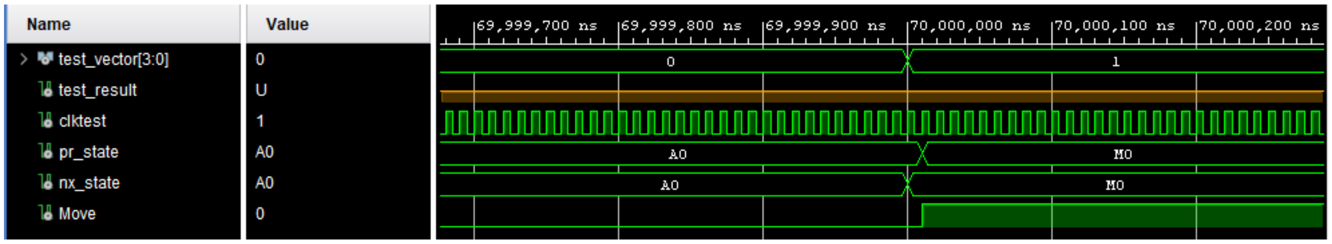


Figure 9: Observation of the delay between Next and Current State

4.2 Simulation of the Speed Bloc

This bloc can be simulated as being linked directly to the inputs, we can simulate it independently of the first bloc. Our goal is to validate if the bloc is able to generate a 5kHz pulse and the 3 PWMs as wished. We see in figure 10 that a pulse lasting one clock period is generated on the `rhythm_5kHz` signal each 0.2us (period of a 5kHz signal).

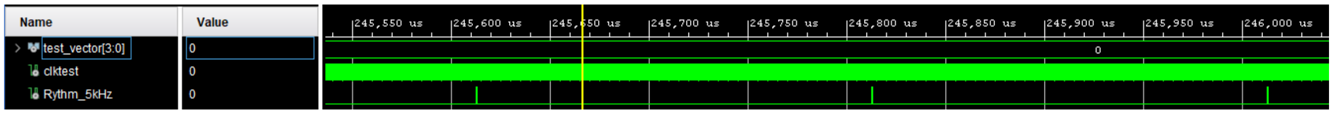


Figure 10: Simulation of 5kHz pulse generated by the Speed Bloc

Based on this signal, as explained before, we generate the 3 PWMs with different duty cycles based on a counter which increments at each pulse of the `rhythm_5kHz`.

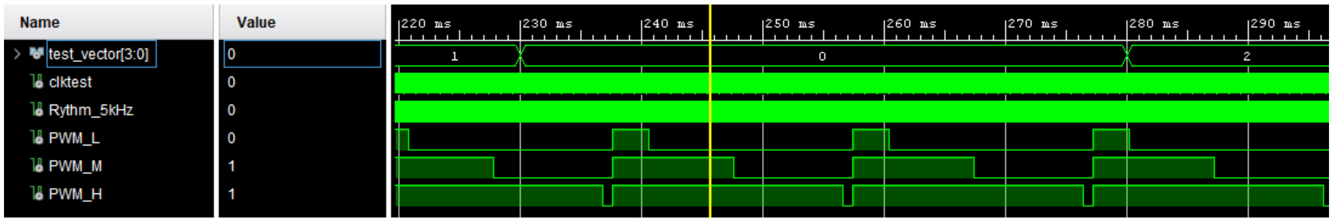


Figure 11: Simulation of 5kHz pulse generated by the Speed Bloc

We can see on Figure 11 that our 3 PWMs are generated with a period of 20ns, and the 3 duty cycles: 15, 50, and 90 percent for `PWM_L`, `PWM_M`, and `PWM_H`.

4.3 Simulation of the Direction Bloc

Now that we have validated the behavior of our two first blocs, we can simulate our last bloc which uses the outputs of the precedent ones as inputs. We see in Figure 13 that the values of the sensor's signals influence the output as we predicted in a precedent section and that a push of buttons leads also to switching the output ON/OFF.

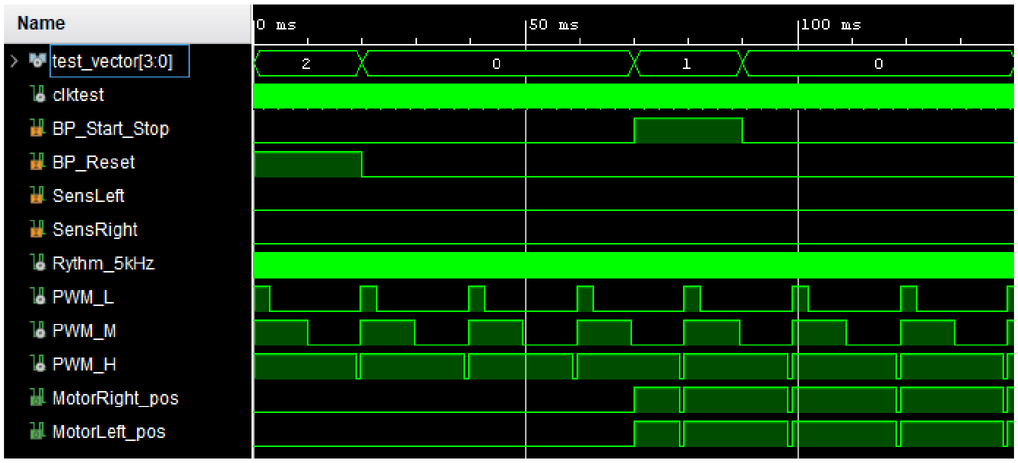


Figure 12: Simulation of the Direction Bloc: Starting Drone

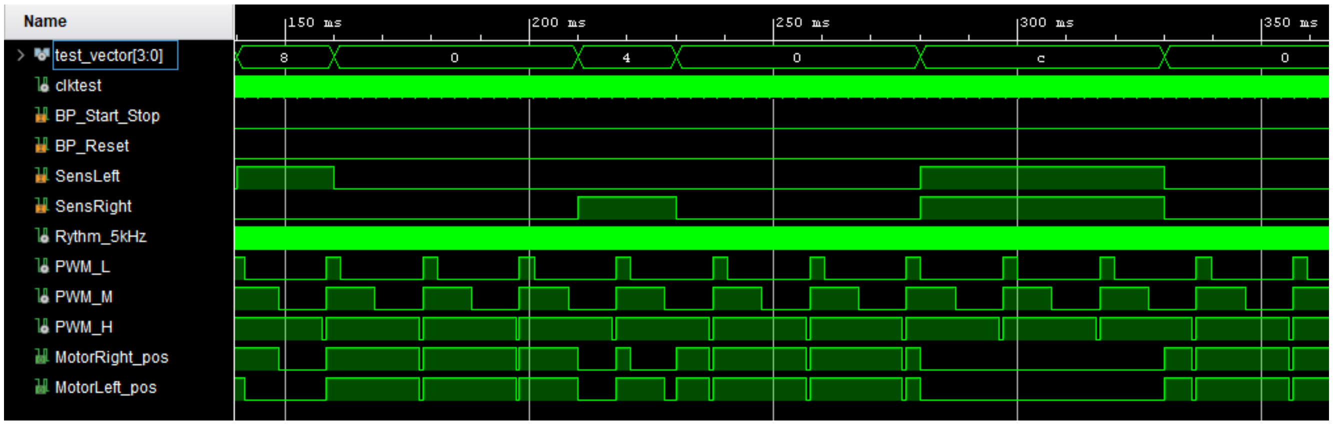


Figure 13: Simulation of the Direction Bloc: Turning

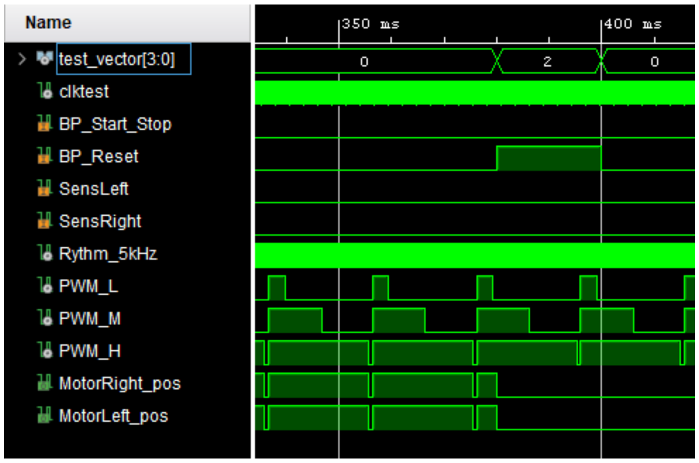


Figure 14: Simulation of the Direction Bloc: Reseting

All of our systems have the wished behavior in simulations. We can now implement it in the FPGA and validate its operation in real conditions.

5 Implementing

After synthesizing our code, we can generate bitstream in order to upload it to the board. Vivado software can display a map of the different links and logic blocs used inside the FPGA. Figure 15 shows the logic mapping for our project. Only a few parts of the Artix®-7 FPGA are used among these available.

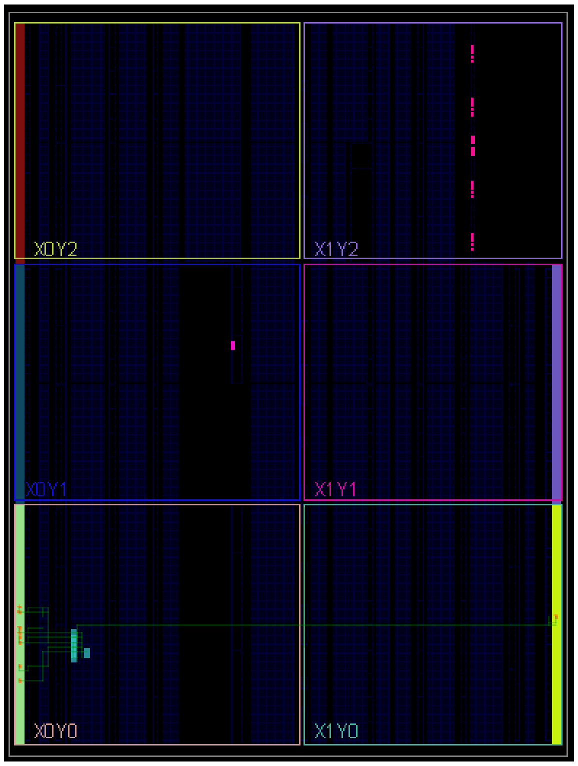


Figure 15: Map of the implementation on the Artix®-7 FPGA

6 Optional part : 7 segments display

The last function we have to implement in our drone is the display of both motors' speeds in the 4 digits 7 segments display of the board.

6.1 Description of the function

According to the datasheet of the board, we have to display the four digits one by one in less than 16ms to have a proper display and not perceive the blinking of the LEDs. Each of the 4 displays has a common Anode (AN0...3) and 8 cathodes (CA...G / DP). To display something, we have to put on low the anode signal corresponding to this digit and the cathode signals of the segments we want to enlighten. We will display digits one by one for 1ms: To do that, we will need a counter which counts at a rhythm of 1kHz from 0 to 3. Depending on this counter value, we will display the digit 0 to 3 by turning the corresponding AN signal to '0' and all the others to '1'. To display the right speed, we must enlighten the cathodes corresponding to the right segments. Figure 16 represent our complete digital function, with this last bloc.

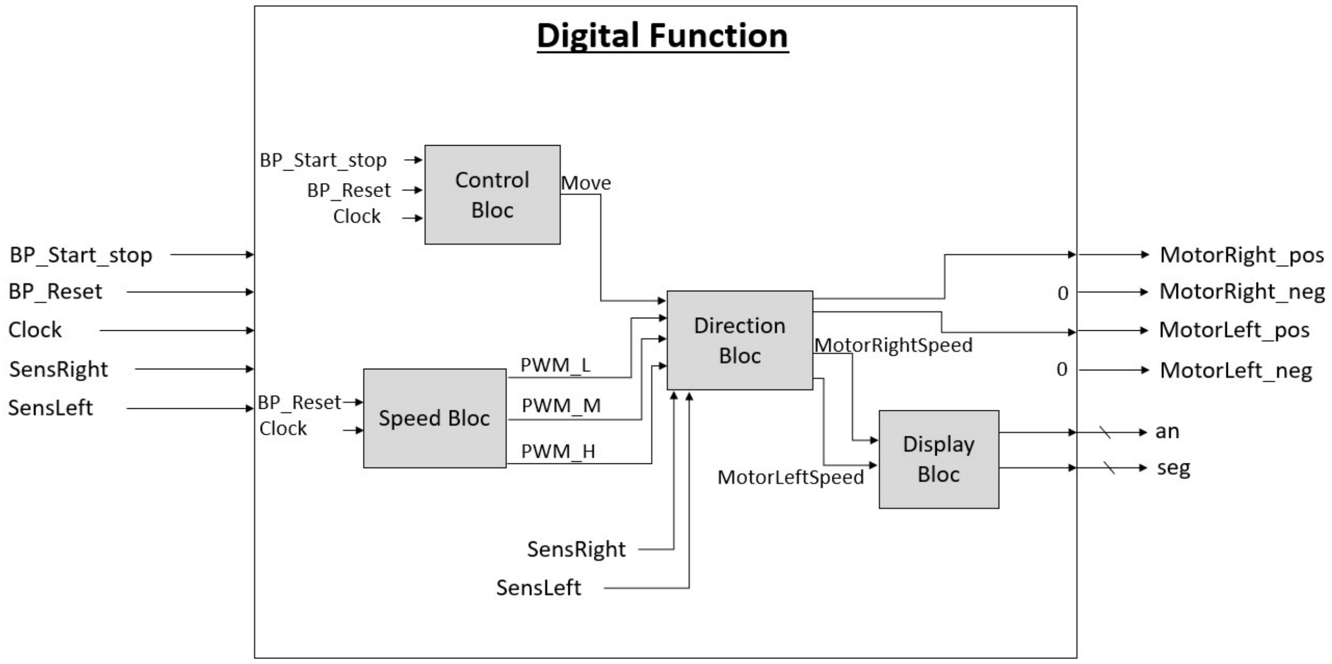


Figure 16: Structure of the Direction bloc

On our application, we don't have to use the dot. We will not use the `DP` signal. On our constraints file, the signal for 4 anodes is `an` which is a vector of size 4 of `std_logic`. The signal for the cathodes is a vector named `seg` of size 6 of `std_logic`. Table 2 represents the different `an` vectors we will apply to display the different digits.

Digit	`an` Binary Vector	`an` Hexadecimal Value
0	'0111'	7
1	'1011'	B
2	'1101'	D
3	'1110'	E

Table 2: `an` Value depending on the digit selected

The speed of both motors can take four values that correspond to the duty cycles : 0, 15, 50, and 95. These figures contain four different numbers: 0,1,5,9. We have to encode these four numbers to be displayed on the screen, selecting the right cathodes. Table 3 list the different `seg` values depending on the number we want to display.

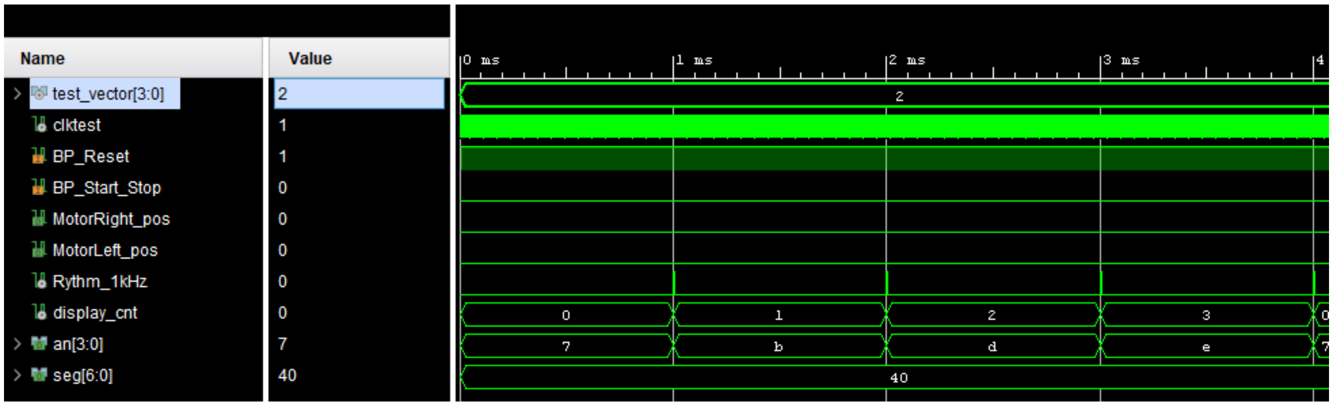
Number	`seg` Binary Vector	`seg` Hexadecimal Value
0	'1000000'	40
1	'1111100'	7C
5	'0010010'	12
9	'0010000'	10

Table 3: `seg` Value depending on the digit selected

To be able to display the values on the screen, we need to access the duty cycle's values of the motors signals in some way. In VHDL, we can't directly access the outputs of our entities. So we have to modify slightly our multiplexer (Direction Bloc) so that when he changes the PWM signals applied on the motors, he changes also inside signals `LeftSpeed` and `LeftSpeed` that can be accessed to display the speeds.

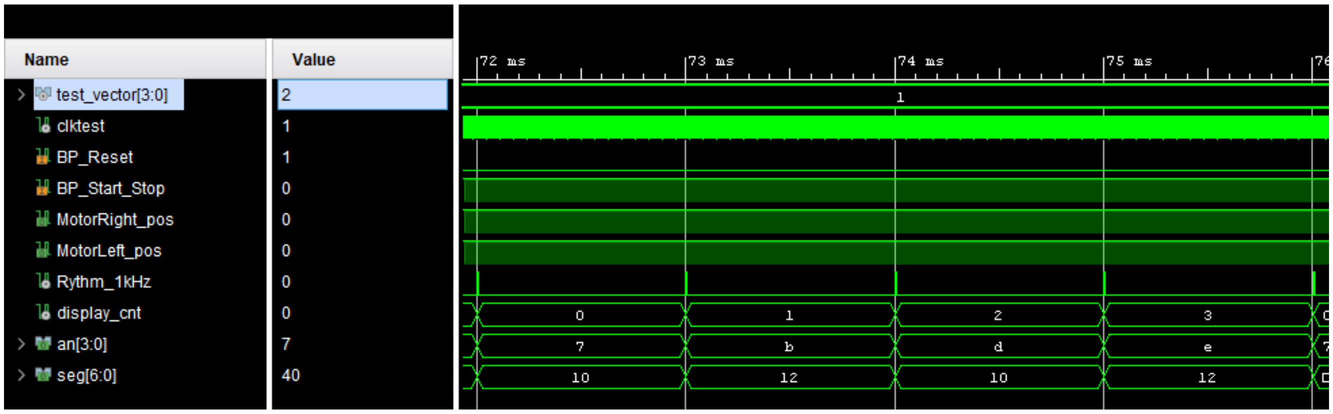
6.2 Simulation

Our simulation allowed us to check several functions: First, we validate that our signal `rhythm1k` has a period equal to 1ms. We can then check that the counter count is from 0 to 3. Depending on this counter, the ‘an’ vector has to change from one value to another which is listed in Table 2. Once we have checked those steps, we know that we succeed to displays the four digits on the 7 segments display as wished. We still have to check if the right speed value is displayed. To do that, we have displayed the outputs `motorRightSpeed_pos` and `MotorLeftSpeed_pos` on our simulations chronogram to check the value of the ‘seg’ signal according to the speed of the motors. The numbers corresponding to the different values of this vector are listed in Table 3. We see in Figure 17 that when both motors are stopped, we have our ‘an’ vector switching values every ms and our ‘seg’ vector that is equal to 40 during the whole time. That means that the display should be ’00 00’ which is accurate. The next figures represent the different combinations of motor speed. We see that in any case, the ‘seg’ signal has the right value and changes each ms, at the same time as the ‘an’ vector, to display another digit.



Name	Value
test_vector[3:0]	2
clktest	1
BP_Reset	1
BP_Start_Stop	0
MotorRight_pos	0
MotorLeft_pos	0
Rythm_1kHz	0
display_cnt	0
an[3:0]	7
seg[6:0]	40

Figure 17: chronogram n°1 of the display simulation



Name	Value
test_vector[3:0]	2
clktest	1
BP_Reset	1
BP_Start_Stop	0
MotorRight_pos	0
MotorLeft_pos	0
Rythm_1kHz	0
display_cnt	0
an[3:0]	7
seg[6:0]	40

Figure 18: chronogram n°2 of the display simulation

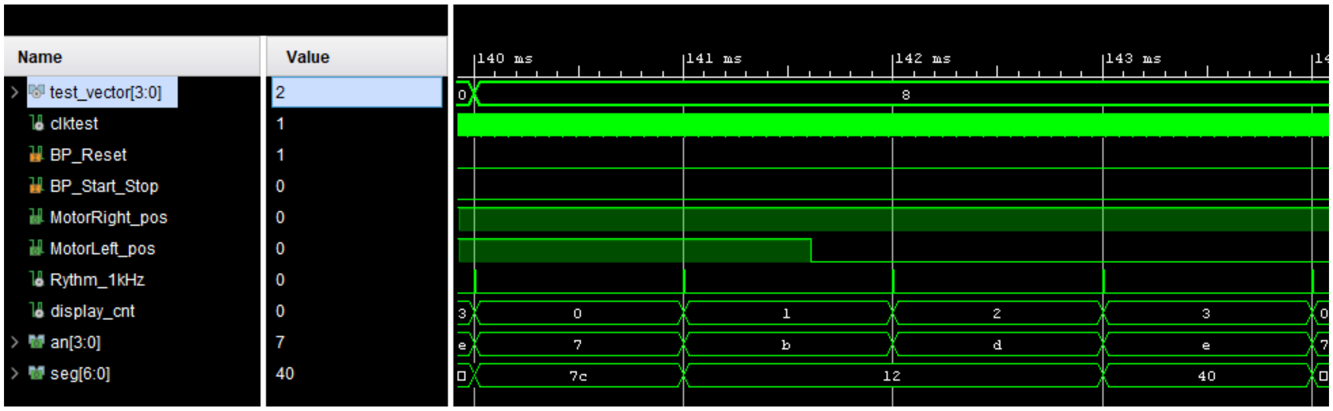


Figure 19: chronogram n°2 of the display simulation

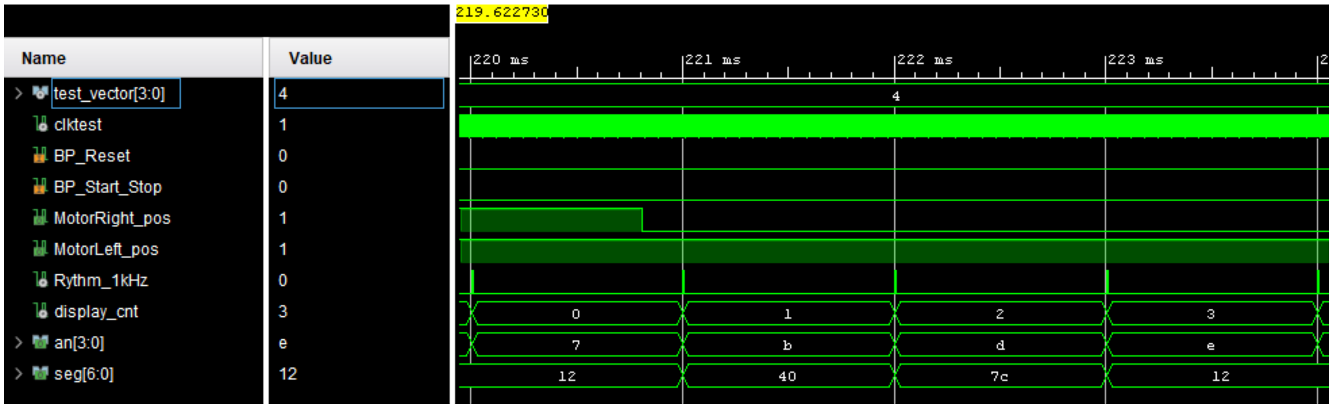


Figure 20: chronogram n°2 of the display simulation