EM433 - Architecture, Design and Synthesis of Hardware Systems

Autonomous ground drone

Author :
Nicola Musacchio
Luis Fernando Jasso-Aceves

Professor for the course :
Pr. Arnaud Dion

November 28, 2021

Contents

List of Figures

1 Presentation and specification of the system

The goal of the lab is to design an autonomous ground vehicle that has to move parts all around a factory by following a black line on the ground.
The drone is made of two sensors, one on the left and the other the right side, which allow to detect a black line. A push button is used to start and stop the drone. In addition there is a reset button. The direction is controlled through the difference of velocity between a left and right motor.

1.1 Inputs and Outputs

The system has 5 inputs:

- **SensRight and SensLeft**: 2 infrared sensors are connected on the 2 pins of the FPGA and the value of their control signals is 1 if a black line is detected, else it is 0;
- **Start_Stop** : a push button is connected to 1 pin of the FPGA. Its value is 1 when the user presses the button, else it is 0;
- **Clk_50MHz**: a 50 MHz clock;
- **Reset**: a reset button whose value is 1 in case it is pressed.

The outputs are listed below:

- **MotorRight**:the right motor is run when its control signal takes the value 1;
- **MotorLeft**: the left motor is run when its control signal takes the value 1.

2 Design of the system's blocks

In order to better understand the behavior of the system, the following 3 subsystem's block have been designed separately and then merged for simulating the main block: Start and Stop, Direction and Speed.

2.1 Start and Stop

There is only one output of the Start and Stop block, this output enables the movement of the motors when it's value is 1, and stops them otherwise. There is only one button to start and stop the ground drone, depending on the current state of the drone, pressing the button may set the output to 1 or 0. In order to describe this behaviour a state machine is used, this is presented in figure 1.

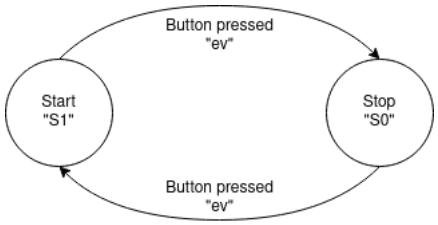

Figure 1: State Machine for starting and stopping

Its important to mention that the signal that triggers a transition is not directly the button, but rather its rising edge, otherwise the state would change with every clock cycle while the user is pressing the button, since the frequency of the clock is 5 MHz, its expected that a human user will press the button over several clock cycles.

For this reason its required to implement a rising edge detector. Additionally, considering that the input is asynchronous it was decided to include a step for synchronizing the signal. This implementation consists in using two flip-flops, the first one ensures that the second one receives a synchronous input, and the second one limits the signal to one clock cycle in duration (with the help of an external logic gate). The diagram of the rising edge detector implemented is presented in figure 2

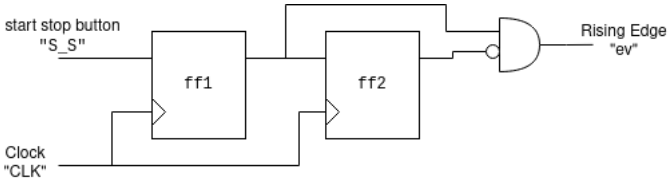

Figure 2: State Machine for starting and stopping

Please note that even though the input is synchronous, we are not addressing the the asynchronous detection of the button signal nor guarding against meta-stability, there are two reasons for this. The first is that as previously mentioned, we expect that the time it takes a human user to press the start-stop button will be much greater than one cycle, given the high frequency of the system. Second reason is simplicity, because additional flip-flops would be required to detect the asynchronous signal by using the signal from the button as a clock.

2.2 Speed

The ground drone does not have any command in order to perform a turn, for this reason the only way to control the direction is the difference of speed between the two motors. As already explained before, each motor is controlled by a 1-bit signal and its value is 1 in case the respective motor is running and 0 in case it is stopped. The speed of the motor will then be controlled by the duty cycle of a 50 Hz Pulse Width Modulation signal. Thus, if the motor signal is 1 for all the PWM duty cycle, then the motor will be at 100%. For this application, 3 different levels of speed are considered:

- **High**: 95% of duty cycle;
- **Medium**: 50% of duty cycle;
- **Low**: 15% of duty cycle.

The only input of the speed block is the 50 MHz clock of the system, and the outputs are the 3 PWM signals corresponding to the 3 different speed levels. In order to create a PWM signal, it is necessary to implement a counter which updates its value at each clock cycle. As the frequency of the PWM signals is 6 orders of magnitude less than the one of the clock, a 5 kHz signal will be used as clock for the PWM counter (Rythm_5kHz signal).

The Rythm signal is created by implementing a counter (Ryhtm_cnt), from 0 to 9999. At each clock rising edge, the Ryhtm counter is updated and the value 0 is assigned to the Rythm signal. Then, at the value 4999 (half of the cycle), the Ryhtm signal's value will be switched to 1. This operation has been done with a synchronous process which takes only the 50 MHz clock in its sensitivity list. The process is executed at each changing state of the signals declared in the sensitivity list, so in this case, every time that a clock rising edge is met.

The same strategy has been adopted for the implementation of the PWM signals. The PWM coutner is an integer in the range 0 to 99 and it updates at each Rythm clock rising

edge. Thus, in this second process, the sensitivity list is made of only the Rythm clock. When the counter is 0, all the 3 PWM signals are set to 1. Then, when the counter is 14, the PWM Low speed signal is set to 0. The same happens for the PWM medium and the PWM high when the counter is respectively 49 and 94. In the figure below, the 3 PWM output signals are displayed.

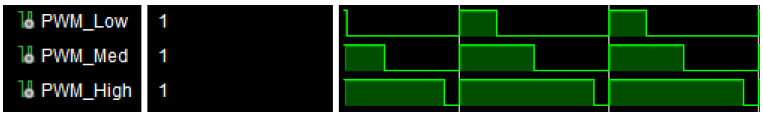

Figure 3: PWM output signals

The 3 PWM output signals are now used as input for the direction block signal.

2.3 Direction

The Direction module is responsible of managing the speed of the two motors according to the sensors detection. The inputs of the block are: the 3 PWM signal generated by the speed block, the right and left sensor signals, and the Start_Stop button signal.
The outputs of the block are the left and right motor.

In the architecture of the block, before starting any process, an internal signal *sens* is declared. It is a *std_logic_vector(2 downto 0)* vector, and its values are respectively the Start_Stop signal, and the left sensor and the right sensor. This choice is useful for easily distinguish the different behaviors of the drone. In this case two processes are used, one for the left motor and the other for the right motor. The sensitivity list of each process is made of the *sens* vector and of the 3 PWM signals. At each changing state of the signals declared in the sensitivity list, the process is executed. A *case...end* structure has been used for assigning the speed to the motors, according to the state of the button and to the sensors detection.
It is possible to distinguish 5 different state:

- **Straight**:when $sens = 100$, both the sensors are not detecting any black line, thus the velocity of the motors is at 95% and the drone will go straight;
- **Turn left**: when $sens = 101$ the right sensor is detecting a black line and a right turn has to be performed. The left motor will consequently run at high speed (95%) while the right motor will run at low speed (15%);
- **Turn right**: when $sens = 110$ the left sensor is detecting a black line and a left turn has to be performed. The right motor will consequently run at high speed (95%) while the left motor will run at low speed (15%);;

- **Lost**: $sens = 111$ both the sensors are detecting a black line, thus the drone will be in a lost state and both the motors will run at a 50% velocity;
- **Stop**: in all the other cases, so when the Start_Stop button is set to 0, the motors will stop running.

3 Simulation Results

for the simulation, a test bench was used for each block, and finally another test bench for the entire system. The scenarios used are described and results are presented in this section.

3.1 Start and Stop

The simulation scenario consists in a periodic input for the start-stop button, and activation of the reset in both states of the states machine. The results are presented in figure 4.

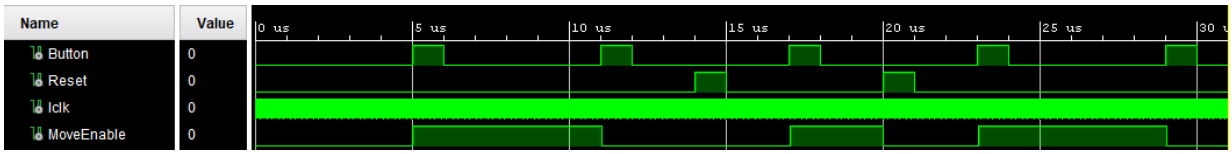

Figure 4: Signals from the start-stop state machine.

3.2 Speed

For the test-bench of the speed block, it is sufficient to provide the clock of the system, the results are presented in figure 5.

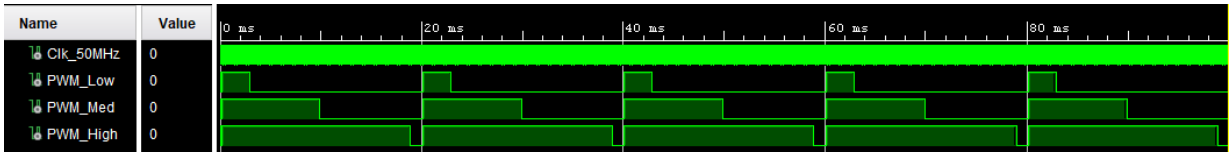

Figure 5: Signals from PWM generation block.

3.3 Direction

The simulation scenario starts with the value for both sensors as 0, in this case both wheels are at high speed, then the sensor of each side takes the value 1 to simulate that the drone is getting out of the good path and should turn, when both sensors have 1, it means that the drone is lost and both wheels are set to medium speed. Finally the Enable signal is set to 0 to show that both wheels stop completely, as can be seen in figure 6.

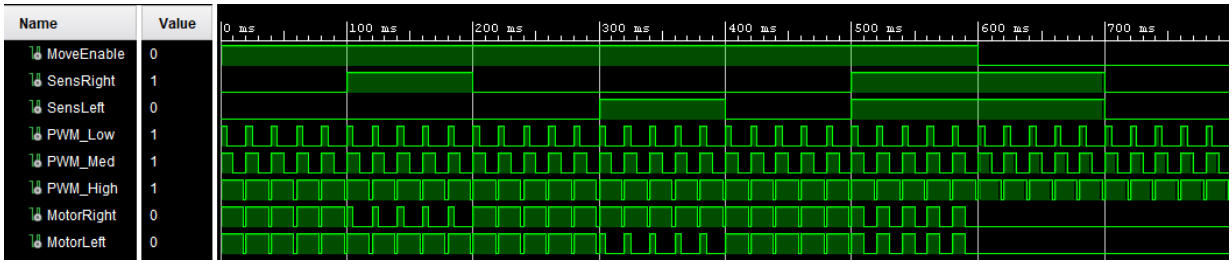

Figure 6: Signals present in the direction test-bench.

3.4 Full System

In this test-bench the start-stop signal is present momentarily to enable the movement of the motors, the sensors of each side are set to 1 for a determined time, and then both sensors together, finally the reset signal is set to 1, this sets the signals for both motors to 0.

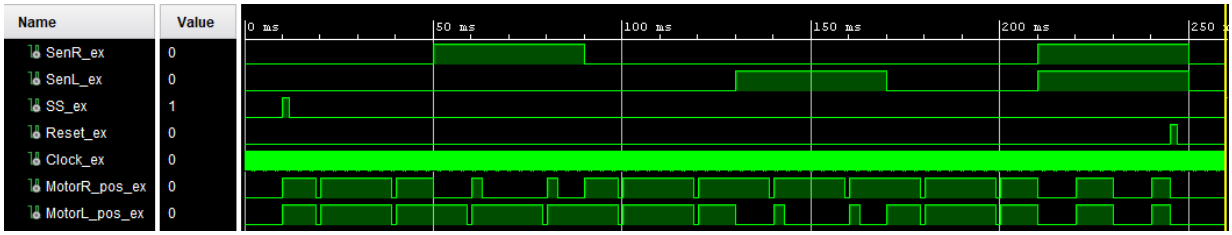

Figure 7: Signals from the test-bench of the complete system

The behaviour presented in figure 7, is compliant with the specification of the ground drone, the simulation of the complete system does not include as many possible scenarios as the previous simulations, given that each block was previously simulated.

4 Experimental Results

The Drone has been tested in different scenarios. Here, it is shown the drone running on a track and performing both right and left turns.

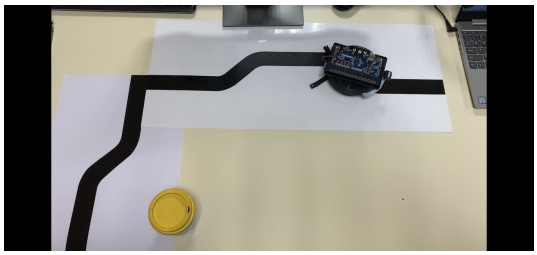

Figure 8: Right turn

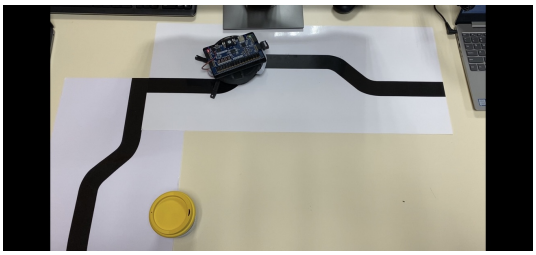

Figure 9: Left turn

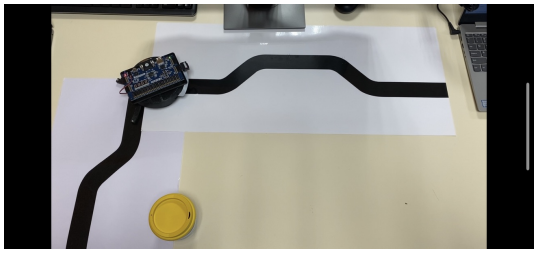

Figure 10: 90 degree turn

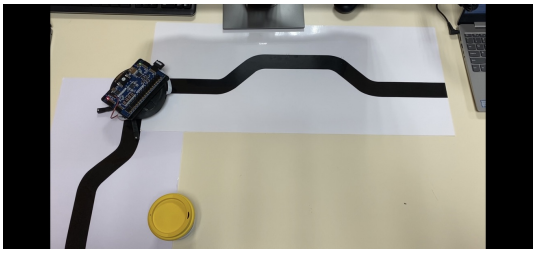

Figure 11: End of 90 degree turn

5 Code

5.1 Ground Drone

```
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity Drone is
35     Port ( SensRight : in STD_LOGIC;
36            SensLeft : in STD_LOGIC;
37            BP_Start_Stop : in STD_LOGIC;
38            BP_Reset : in STD_LOGIC;
39            Clock : in STD_LOGIC;
40            MotorRight_pos : out STD_LOGIC;
41            MotorRight_neg : out STD_LOGIC;
42            MotorLeft_pos : out STD_LOGIC;
43            MotorLeft_neg : out STD_LOGIC--;
44            --Display : out STD_LOGIC_VECTOR(6 dwonto 0);
45            --AN : out STD_LOGIC_VECTOR(6 dwonto 0)
46            );
47 end Drone;
48
49 architecture Behavioral of Drone is
50     signal MoveEnable : STD_LOGIC;
51     signal PWM_Low : STD_LOGIC;
52     signal PWM_Med : STD_LOGIC;
53     signal PWM_High : STD_LOGIC;
54
55     component Start_StateMachine is
56     Port ( S_S : in STD_LOGIC;
57            RST : in STD_LOGIC;
```

```
58              CLK : in STD_LOGIC;
59              ENABLE_OUT : out STD_LOGIC);
60      end component;
61
62      component Direction is
63      Port ( ENABLE_IN : in STD_LOGIC;
64              SEN_RIGHT : in STD_LOGIC;
65              SEN_LEFT : in STD_LOGIC;
66              PWM_L_IN : in STD_LOGIC;
67              PWM_M_IN : in STD_LOGIC;
68              PWM_H_IN : in STD_LOGIC;
69              MOT_RIGHT : out STD_LOGIC;
70              MOT_LEFT : out STD_LOGIC);
71      end component;
72
73      component Speed is
74      Port ( CLK : in STD_LOGIC;
75              PWM_L_OUT : out STD_LOGIC;
76              PWM_M_OUT : out STD_LOGIC;
77              PWM_H_OUT : out STD_LOGIC);
78      end component;
79
80  begin
81      MotorRight_neg <= '0';
82      MotorLeft_neg <= '0';
83      CONTROL_BLOCK: Start_StateMachine PORT MAP ( S_S => BP_Start_Stop,
84                                                   RST => BP_Reset,
85                                                   CLK => Clock,
86                                                   ENABLE_OUT => MoveEnable);
87
88      DIRECTION_BLOCK : Direction PORT MAP ( ENABLE_IN => MoveEnable,
89                                             SEN_RIGHT => SensRight,
90                                             SEN_LEFT => SensLeft,
91                                             PWM_L_IN => PWM_Low,
92                                             PWM_M_IN => PWM_Med,
93                                             PWM_H_IN => PWM_High,
94                                             MOT_RIGHT => MotorRight_pos,
95                                             MOT_LEFT => MotorLeft_pos);
96
97      SPEED_BLOCK : Speed PORT MAP ( CLK => Clock,
```

```
                                    PWM_L_OUT => PWM_Low,
                                    PWM_M_OUT => PWM_Med,
                                    PWM_H_OUT => PWM_High);
end Behavioral;
```

5.2 Start-stop block

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Start_StateMachine is
    Port ( S_S : in STD_LOGIC;
           RST : in STD_LOGIC;
           CLK : in STD_LOGIC;
           ENABLE_OUT : out STD_LOGIC);
end Start_StateMachine;

architecture Behavioral of Start_StateMachine is
    type state is (S0, S1);
    signal pr_state, nx_state: state;
    signal ff1 : STD_LOGIC := '0';
    signal ff2 : STD_LOGIC := '0';
    signal ev : STD_LOGIC := '0';

begin
-- section 0: Event detector
ev <= ff1 and not ff2;
process (CLK)
begin
```

```
53      if (CLK'event and CLK='1') then
54      ff1 <= S_S; -- Synchronize input
55      ff2 <= ff1; -- Detect event
56      end if;
57  end process;
58  -- section 1: fsm register
59  process (RST,CLK)
60  begin
61      if (RST='1') then
62      pr_state <= s0; -- choose reset state
63      elsif (CLK'event and CLK='1') then
64      pr_state <= nx_state;
65      end if;
66  end process;
67  -- section 2: next state function
68  process (ev, pr_state)
69  begin
70  case pr_state is
71      when S0 =>
72          if (ev = '1') then
73          nx_state <= S1;
74          else
75          nx_state <= S0;
76          end if;
77      when S1 =>
78          if (ev = '1') then
79          nx_state <= S0;
80          else
81          nx_state <= S1;
82          end if;
83  end case;
84  end process;
85  -- section 3: output function
86      ENABLE_OUT <= '1' when pr_state = S1 else '0';
87  end Behavioral;
```

5.3 Speed block

```
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity Speed is
35     Port ( CLK : in STD_LOGIC;
36            PWM_L_OUT : out STD_LOGIC;
37            PWM_M_OUT : out STD_LOGIC;
38            PWM_H_OUT : out STD_LOGIC);
39 end Speed;
40
41 architecture Behavioral of Speed is
42     signal PWMcnt: integer range 0 to 99 := 0;
43     signal Rythm_cnt: integer range 0 to 9999 := 0;
44     signal Rythm_CLK : STD_LOGIC := '0';
45
46 begin
47 --create the rythm signal 5kHz
48 process(CLK)
49 begin
50 if (CLK'event and CLK='1') then
51     if (Rythm_cnt = 0) then
52         Rythm_CLK <= '0';
53         Rythm_cnt <= Rythm_cnt + 1;
54     elsif (Rythm_cnt = 4999) then
55         Rythm_CLK <= '1';
56         Rythm_cnt <= Rythm_cnt + 1;
57     elsif (Rythm_cnt < 9999) then
58         Rythm_cnt <= Rythm_cnt + 1;
59     elsif (Rythm_cnt = 9999) then
```

```
60          Rythm_CLK <= '0';
61          Rythm_cnt <= 0;
62      end if;
63  end if;
64  end process;
65  --create the PWM output (50 Hz)
66  process(Rythm_CLK)
67  begin
68  if (Rythm_CLK'event and Rythm_CLK='1') then
69      if (PWMcnt = 0) then
70        PWM_L_OUT <= '1';
71        PWM_M_OUT <= '1';
72        PWM_H_OUT <= '1';
73        PWMcnt <= PWMcnt + 1;
74      elsif (PWMcnt = 14) then
75        PWM_L_OUT <= '0';
76        PWMcnt <= PWMcnt + 1;
77      elsif (PWMcnt = 49) then
78        PWM_M_OUT <= '0';
79        PWMcnt <= PWMcnt + 1;
80      elsif (PWMcnt = 94) then
81        PWM_H_OUT <= '0';
82        PWMcnt <= PWMcnt + 1;
83      elsif (PWMcnt = 99) then
84        PWMcnt <= 0;
85      else
86       PWMcnt <= PWMcnt + 1;
87      end if;
88  end if;
89  end process;
90
91  end Behavioral;
```

5.4 Direction

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
```

```
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity Direction is
35     Port ( ENABLE_IN : in STD_LOGIC;
36            SEN_RIGHT : in STD_LOGIC;
37            SEN_LEFT : in STD_LOGIC;
38            PWM_L_IN : in STD_LOGIC;
39            PWM_M_IN : in STD_LOGIC;
40            PWM_H_IN : in STD_LOGIC;
41            MOT_RIGHT : out STD_LOGIC;
42            MOT_LEFT : out STD_LOGIC);
43 end Direction;
44
45 architecture Behavioral of Direction is
46 signal sens: std_logic_vector(2 downto 0);
47 begin
48     sens <= ENABLE_IN & SEN_LEFT & SEN_RIGHT;
49     -- Motor Right
50     process (sens, PWM_L_IN, PWM_M_IN, PWM_H_IN)
51     begin
52         case sens is
53             when "100" => MOT_RIGHT <= PWM_H_IN;
54             when "101" => MOT_RIGHT <= PWM_L_IN; -- turn right
55             when "110" => MOT_RIGHT <= PWM_H_IN; -- turn left
56             when "111" => MOT_RIGHT <= PWM_M_IN; -- lost
57             when others => MOT_RIGHT <= '0'; -- ENABLE = 0
58         end case;
59     end process;
60     -- Motor Left
61     process (sens, PWM_L_IN, PWM_M_IN, PWM_H_IN)
62     begin
63         case sens is
64             when "100" => MOT_LEFT <=PWM_H_IN;
```

```
65              when "101" => MOT_LEFT <=PWM_H_IN; -- turn right
66              when "110" => MOT_LEFT <=PWM_L_IN; -- turn left
67              when "111" => MOT_LEFT <=PWM_M_IN; -- lost
68              when others => MOT_LEFT <='0'; -- ENABLE = 0
69          end case;
70      end process;
71  end Behavioral;
```