POLITECNICO DI BARI

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

Master's Degree in Automation Engineering

Digital Programmable Systems' Project Work

FPGA BASED ADAPTIVE TRAFFIC LIGHT CONTROLLER DESIGN

Professor:

Dr. Eng. Martino De Carlo

Candidates:

Francesco Di Chio

Lorenzo Frari

ACADEMIC YEAR 2023–2024

Abstract

The main purpose of this project work is the development of an adaptive Field Programmable Gate Array (FPGA) based control system able to control a traffic light four-way junction.
The control system is based on seven HC-SR04 ultrasound sensors, one for each travel lane, able to monitoring the traffic density. Continuous monitoring of these sensors provides real time vehicle density to ensure proper signals generation.
The control system is implemented on a Terasic DE10-Lite FPGA, programmed using VHDL code on the software Quartus.
The proposed traffic light controller allows for reduced waiting times of vehicles, optimizing the average traffic flow, reducing traffic jams and consequently lowering the environmental pollution, thus making an important contribution in smart cities context.

Contents

Chapter 1

INTRODUCTION

The population increasing and growing cities are key factors related to the important increase of vehicles in urban cities, with big consequences regarding management traffic systems and environmental pollution.
In this context, an optimal and intelligent traffic managing system is required in order to reduce traffic jam and transportation delays.

Traditional traffic lights can improve this situation, although having a very limiting drawback : the duration of traffic lights colors referred to a specified travel lane doesn't adapt according to the real amount of vehicles crossing the junction where the traffic light is installed.
For all these reasons an intelligent traffic light controller is required, able to adapt the traffic light behaviour in response of the vehicle flow.

The paper shows all the most important characteristics and steps done in order to develop the controller.
The first part focuses on the case study and the main idea behind the controller, that is the a five state finite state machine which movement happens by means of special passing signals generated in a proper way.
After this first part, an hardware structure description is presented, including the DE10-Lite FPGA and seven HC SR-04 ultrasound sensors as main components, and LEDs, resistors, jumpers cable and breadboards as supporting parts for test circuit construction.
Then, a detailed description of the implemented VHDL code is presented in order to have a deeper understanding of the adaptive behaviours of the controller.

The proposed traffic light controller is ultimately characterized by two adaptive concept, applied both in case of low traffic density both in high traffic density one. This ensure an optimal traffic managing in every situation, proving to be an excellent device that could be implemented in the smart cities of the future.

Chapter 2

CASE STUDY

2.1 Traffic light crossing structure

The designed traffic light junction's controller is able to manage a four-way junction road, with seven travel lane as shown in figure 2.1 .
For each travel lane a traffic light is required in order to allow cars to pass when there is a green light, giving caution that the traffic is going to be stopped in few seconds in case of yellow light and stopping them in case of red light.

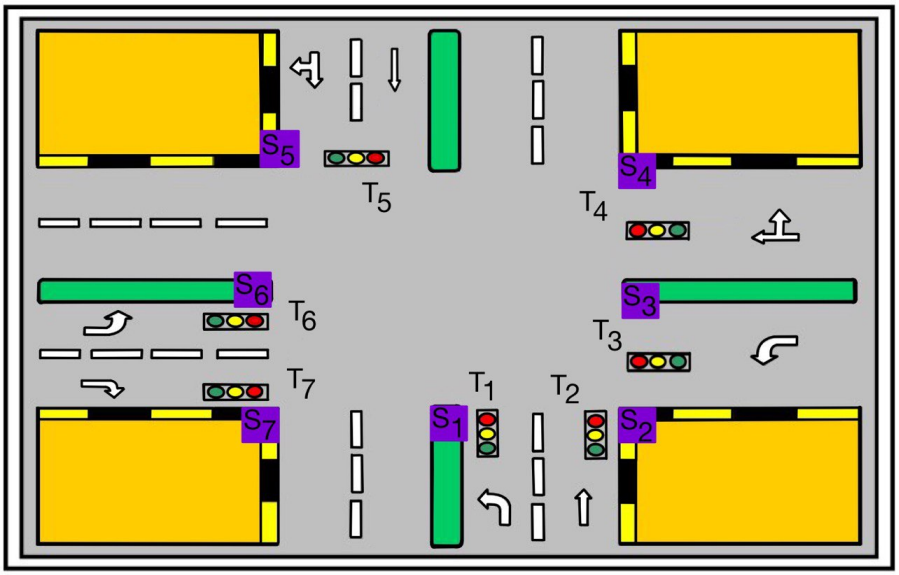

Figure 2.1: Case study's four way junction

The figure shows the seven traffic lights, counterclockwise named from T1 to T7, and the respective sensor for each traffic light, named from S1 to S7.
The sensors' placement in the figure shows their position in the real implementation: they are placed in close proximity to each traffic light, giving information about cars passing that traffic light.

This is an assumption that has important consequences on the entire project that will be clarify in the next sections : the position of the sensor makes that his working interval is during a green light. Indeed, in case of red light for a generic traffic light there are no cars passing close to the sensor, making it useless during this amount of time.
For this reason every sensor is turned off during the red light interval of his respective traffic light, resetting also the count value of cars passed during the previous green and yellow light.

2.2 State diagram

The switching order between traffic lights is made in a manner consistent with the norms, ensuring maximum safety e reliability.
This result, for instance, from the impossibility to have, for obvious safety reasons, simultaneously a green light for T1 and T5 or for T3 and T5.
The switching order controlled by the developed software is resumed in the following table:

	T1	T2	T3	T4	T5	T6	T7
st1	yellow	yellow	yellow	yellow	yellow	yellow	yellow
st2	red	green	red	red	green	red	red
st3	green	green	red	red	red	red	red
st4	red	red	red	green	red	red	green
st5	red	red	green	red	red	green	red

Table 2.1: Traffic light switching order

As shown from the table, the developed controller is based on a five state Finite State Machine (FSM), where the *passing signals* that allow the passing from one state to another are generated by the FPGA according to the VHDL code implemented and discussed in chapter 4.

The first state is an *idle state*, in which all the sensors are turned off and for all the traffic lights there is a yellow flashing light. In this situation the normal rules of precedence have to be followed by the drivers.
This state, reachable from all other states by means of a switch, can be useful during night or during maintenance periods. Is important to point out that this is the only state reachable by an external input: a switch; when the control switch is turned on, the traffic light starts his adaptive behaviour and the only way to return in this idle state is turning off the switch.
In the second state the T2 and T5 traffic lights are managed to be on, while all the other traffic lights are red.
An important concept, that will be clearer in the VHDL code discussion, is that in every state, apart from the first one, is managed both the time interval in which you have green light, both the transition to yellow light for the traffic lights interested in the state. After

the yellow light time interval, the correct passing signal is generated and the FSM goes to the next state.
In the third state T2 is still green and T1 is turned on, while the other traffic lights are red.
You can notice that, in this state, the traffic light T7 could be also turned on, but it was decided to choose a logic that would allow you to turn on two traffic lights in each state. However, with very small changes in the VHDL code, this behaviour could be easily implemented.
In the following state T4 and T7 are on, while in the last one T3 and T6 are on.
When the yellow light time interval for state 5 is completed, the passing signal is generated allowing the FSM to go to the state 2 and the cycle can restart.

The described behaviour of the traffic light controller is resumed in the following five state *finite state machine* (FSM). This approach is quite common in this type of controllers for many reasons :

- clearer visualization of the general behaviour of the controller;
- simpler visualization of the current state and the events managing for each state;
- simpler software implementation, especially on microcontrollers and FPGAs;
- big modularity that allows easy changes and improvements of the controller behaviour;

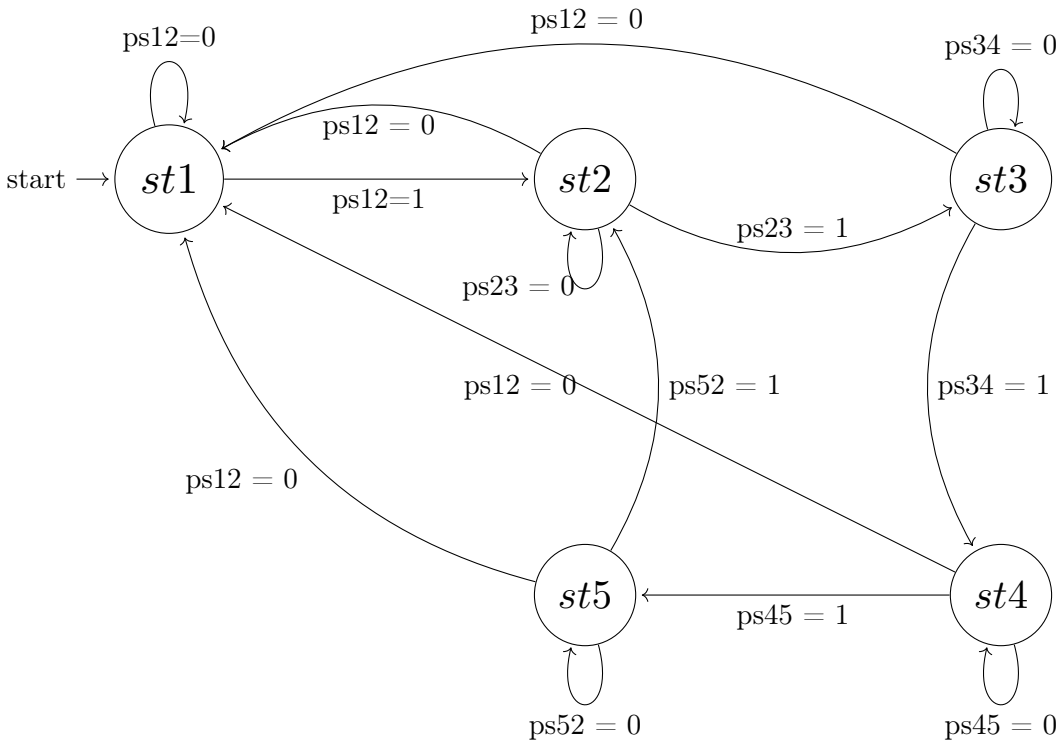

Figure 2.2: Finite state machine

The FSM clearly shows how the controller passes between the states: until the passing signal referred for each state is generated, the FSM remains in its current state and does it operations in terms of switching on the correct lights for each traffic light. All the operations done in every state by the controller will be clear in the VHDL code description in chapter 4. At the end of the operations for each state, the correct passing signal is generated and the FMS moves to the successive state.
You can also verify that the only way to return to the idle state is when there is the event *ps12 = 0*, corresponding to the external switch turned off, which can happen in any state being an external input.

A final block diagram scheme of the designed controller is shown below, giving a general idea of the project work.

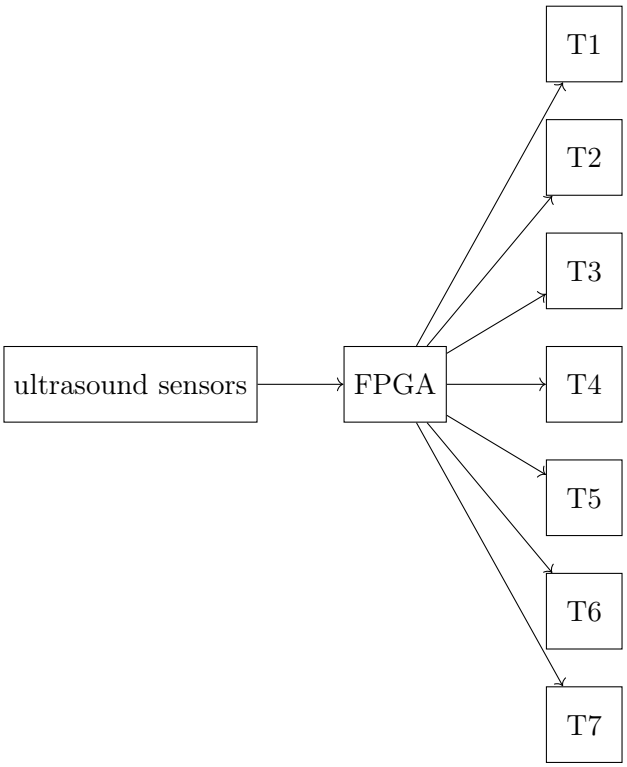

Figure 2.3: Controller's block diagram

Chapter 3

HARDWARE IMPLEMENTATION

3.1 Hardware components

In order to test the behaviour of the designed controller and having a better visualization of the whole managing process, a breadboard based test circuit has been implemented. A complete list of the components used in the project is showed below:

- DE10-Lite FPGA;
- HC SR-04 Ultrasound sensors;
- LEDs;
- Resistors;
- Jumper
- Breadboards

3.1.1 DE10-Lite FPGA

The DE10-Lite presents a robust hardware design built around the Altera MAX 10 FPGA. The MAX 10 FPGA is well equipped to provide cost effective, sigle-chip solutions in control or data path applications and industry-leading programmable logic for ultimate design flexibility.

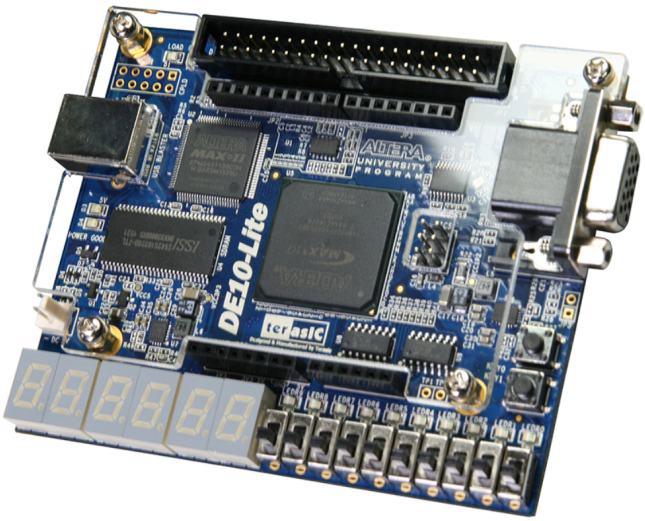

Figure 3.1: DE10-Lite FPGA

The DE10-Lite development board includes hardware such as:

- on-board USB Blaster;
- 3-axis accelerometer;
- Integreted dual ADCs;
- 50k programmable logic elements;
- 1.638 Kbits M9K memory;
- 5.888 Kbits user flash memory;
- 64MB SDRAM x16 bits data bus;
- 144 18 x 18 multiplier;
- 4 PLLs;
- 2x20 GPIO Header;
- Arduino Uno R3 connector;
- 10 LEDS;
- 10 Slide Switches;
- 2 Push Buttons with debounced;
- Six 7-Segments display;

From the circuit implementation point of view, the FPGA's external component used are:

- Switch SW0 : when it's off the controller is in the idle state, while it allows the adaptive behaviour of the controller when it's on;
- LEDs LEDR0, LEDR1 and LEDR2 : they are used to display the current state in which the FSM is located;
- Seven segments display : they are used in order to have an indication about the vehicles' detection from the ultrasound sensors;
- GPIO and Arduino Uno pin : through jumpers, they are used to carry the signals generated by the FPGA on the breadboars.

3.1.2 HCSR-04 Ultrasound sensor

The HCSR-04 is a non-contact position sensor, using ultrasound to detect an object's position.
The detection is subject to the shape of the object whose position is wanted to be found and it's output could be a bit noisy.
These characteristics makes this sensor not so suitable for control application; anyway, it's low cost and easy implementation have greatly contributed to it's popularity.
The sensor is provided by four pins, that are:

- VCC : pin for the 5V power supply voltage;
- GND : pin for 0V ground voltage;
- ECHO : output pin used for the object detection;
- TRIG : Trigger input;

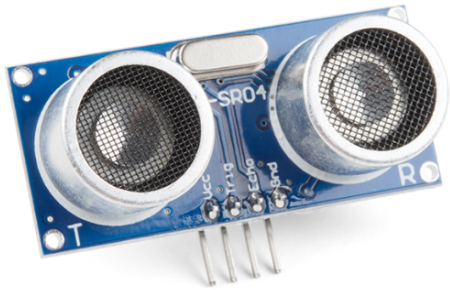

Figure 3.2: HCSR-04 Ultrasound sensor

The basic work principle includes:

- setting the IO trigger to HIGH level for at least 10us

- The module will send out an 8 cycle burst of ultrasound at 40 kHz. In case of object's detection, the wave generated from the reflection on the object generate a pulse signal back

- the output is a square wave signal which it's HIGH amount of time is proportional to the object's distance

For the purpose of the project, the sensor is used not to detect the cars distance form the sensor, but only to detect the passing of vehicles close to sensor. For this reason the signal generated from the sensor goes to increment a counter that gives a real time information of the traffic density.
For further sensor's specification consult the datasheet.

3.1.3 LEDs

21 diode LEDs are used to implement 7 traffic lights, so there is a "visual" output to the FPGA's logic.

Figure 3.3: red, yellow and green LEDs

3.1.4 Resistors

Pretty standard 220 ohm and 0.25 watt resistors are used in order to have a correct use of the LEDs. One resistor is placed in series to one LED, so it will prevent the LED from burning out.

Figure 3.4: Resistors of 220 ohm

A conceptual schematization of the power supply circuit for a single traffic light is illustrated in the following figure: the signals generated from the FPGA close only one switch at a time. For this reason there is always only one light on, in accordance with the designed VHDL code.

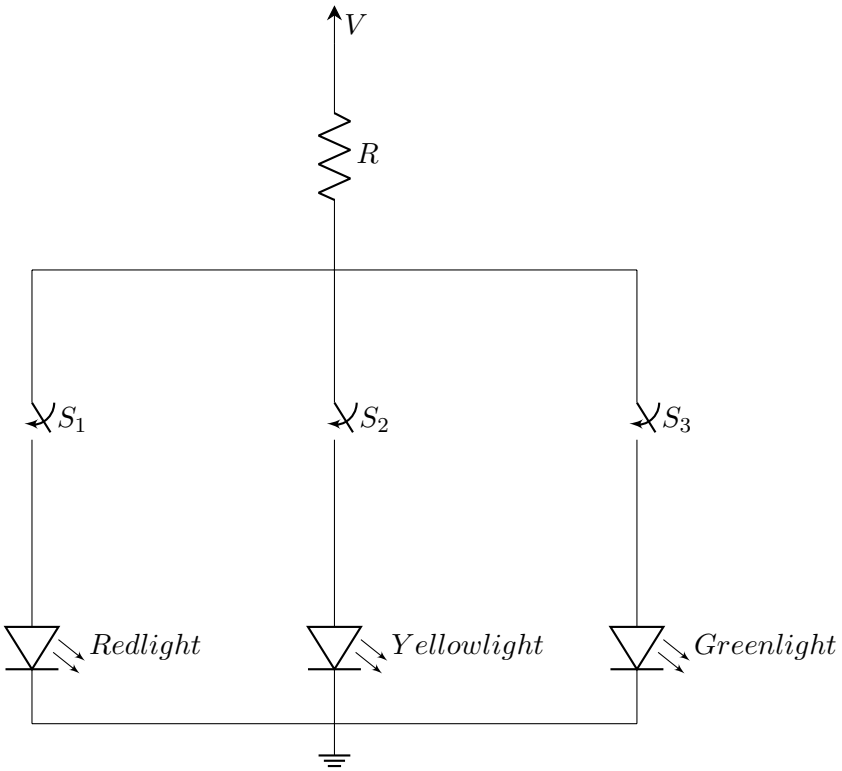

Figure 3.5: LEDs power supply circuit

3.1.5 Jumpers

Jumper wires are used for making electric connections between items on the breadboard and the FPGA's header pins. This provides the appropriate power supply according to the VHDL code designed for sensors and LEDs. They are made of aluminium and they have length of 20 cm.

Figure 3.6: Jumper cables

3.1.6 Breadboards

Breadboards are used for the placement of electrical components and making the electric circuit. There is no need of soldering and placing components on it it's as fast as simple. They comes in different sizes. In order to test the controller 7 small (35*47 mm) breadboards are used, one for each traffic light.

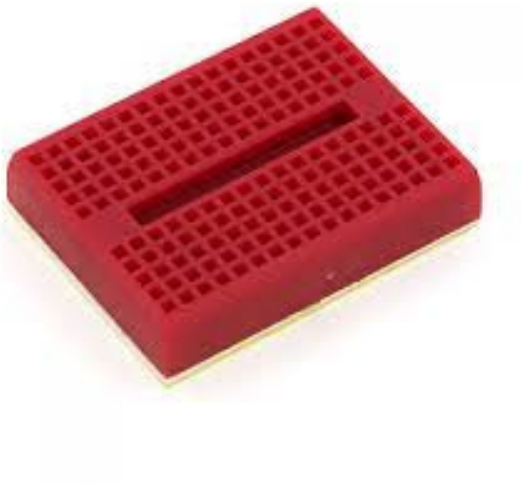

Figure 3.7: Breadboard

The arrangement of the hardware is showed in the following figure.

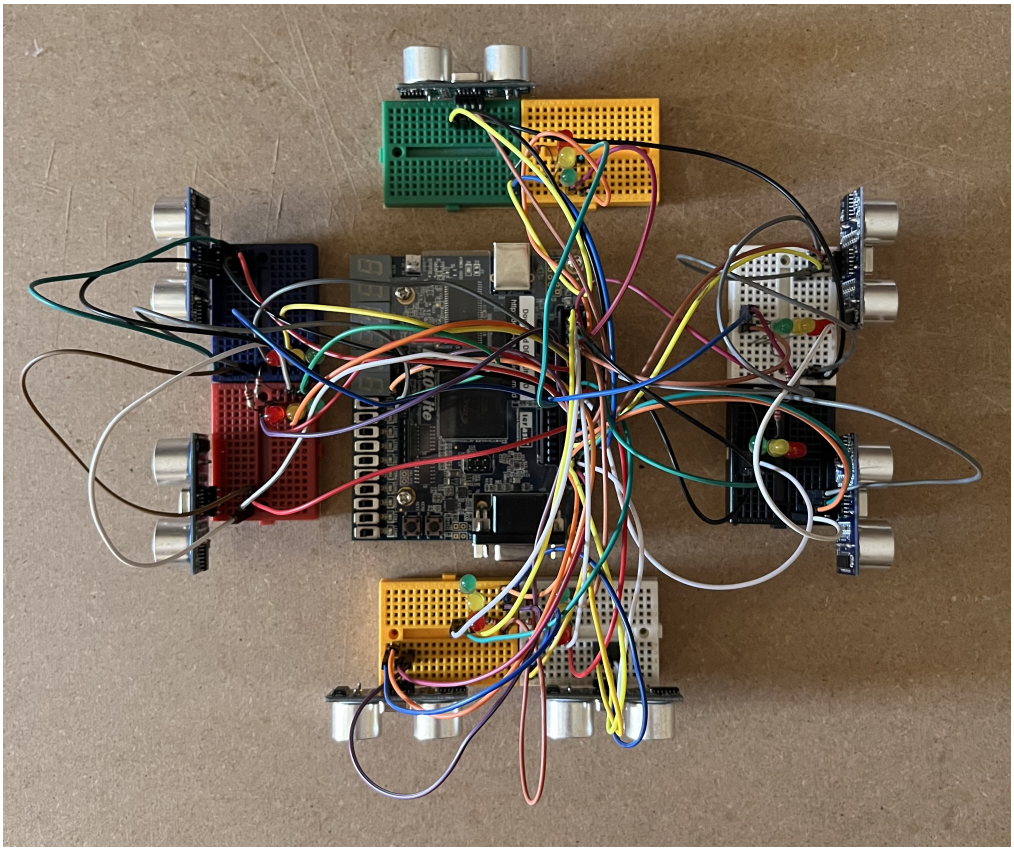

Figure 3.8: Full hardware implementation

Chapter 4

SOFTWARE IMPLEMENTATION

4.1 Quartus Prime

Quartus Prime is programmable logic device design software produced by Intel. It allows developers to analyze and synthesize HDL (hardware description language) designs and supports both VHDL and Verilog for hardware description, visual editing of logic circuits, and vector waveform simulation.
Quartus Prime is widely used for designing and implementing digital circuits on Intel FPGAs (Field-Programmable Gate Arrays).

4.2 VHDL CODE

The Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) is a language that describes the behavior of electronic circuits, most commonly digital circuits. VHDL can be used for designing hardware and for creating test entities to verify the behavior of that hardware, minimizing both cost both developing time.

For the implementation of the proposed controller a behavioral approach is used, so with a high level of abstraction. This approach makes the code similar to a classic programming language due to the presence of if-then-else or case-when constructs.
The sensor's behaviour has been described separately from the traffic lights behavior and has been imported in the Top level entity file as a component.

Every VHDL design description consists of at least one entity/architecture pair.

4.2.1 Top level entity

The top level entity describes the model from an external point of view, as a model interface. For this reason, here there is a list of all the I/O of the model and connected to physical pins of the board.

4.2.1.1 Designed controller top level entity

The top level entity of the designed controller and the libraries used are included in the following listing:

Listing 4.1: Traffic light top level entity

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity traffic_light is
        port(clk_fpga : in std_logic;
        passing_signal_12 : in std_logic;
        light_traffic1 : out std_logic_vector(2 downto 0);
        light_traffic2 : out std_logic_vector(2 downto 0);
        light_traffic3 : out std_logic_vector(2 downto 0);
        light_traffic4 : out std_logic_vector(2 downto 0);
        light_traffic5 : out std_logic_vector(2 downto 0);
        light_traffic6 : out std_logic_vector(2 downto 0);
        light_traffic7 : out std_logic_vector(2 downto 0);

        --people_light_traffic1 : out std_logic_vector(2 downto 0);
        --people_light_traffic2 : out std_logic_vector(2 downto 0);
        --people_light_traffic3 : out std_logic_vector(2 downto 0);
        --people_light_traffic4 : out std_logic_vector(2 downto 0);

        debug_led1 : out std_logic_vector(1 downto 0);
        debug_led2 : out std_logic_vector(1 downto 0);
        debug_led3 : out std_logic_vector(1 downto 0);
        debug_led4 : out std_logic_vector(1 downto 0);
        debug_led5 : out std_logic_vector(1 downto 0);
        debug_led6 : out std_logic_vector(1 downto 0);
        debug_led7      : out std_logic_vector(1 downto 0);

        led_for_visual_state : out std_logic_vector(2 downto 0);
        ECHO_in_pin : in std_logic_vector(6 downto 0);
        trigger_pin : out std_logic_vector(6 downto 0));
end traffic_light;
```

It can be seen that the higher level entity includes:

- clk_fpga: This input is used in order to use the internal clock of the FPGA. Indeed, through the Pin planner section in the Quartus Prime software, the pin is mapped with the 50 MHz internal clock of the FPGA;

- passing_signal_12 : input mapped with a physical switch on the FPGA. This is an important input, how you can see from the figure 2.2, because it allows the adaptive behaviour of the traffic light or not;

- light_traffic: it's a vector of 3 element and represent the actual lights (LEDs) of the single traffic light. Starting from the most significant one, the bits follow the sequence red-yellow-green. It was chosen to use 7 outputs, one for each traffic light, in order to have a clearer visualization and a more intuitive code;

- people_light_traffic : output representing the pedestrian traffic lights. The logic is the same described for the previous output. This part of code is commented because of the lack of enough pins on the board;

- debug_led: it's basically an "attribute" of the "object" sensor that has been imported as a component. This was largely used as debug system in order to verify the vehicles passing close to the sensor, but it was decided to left this feature implemented in the final project. It is a signal output and is connected to certain display's segment; each display for each sensor is used (except for the last display that show the output of 2 sensors);

- led_for_visual_state: this was used as debug output too. It shows, in a binary base, the current state of the finite state machine in which the system is;

- ECHO_in_pin: this is the signal output generated by the sensor, that is an input signal for the FPGA. In this case it was chosen to use a single seven-bits vector, where each bit is related to a specified sensor ;

- trigger_pin: this is the signal input for the sensor, then generated by the FPGA. It was chosen to use a single seven-bits vector as in the ECHO_in_pin case.

4.2.1.2 Ultrasound sensor top level entity

The ultrasound sensor behaviour is implemented in another file, imported subsequently in the main one thanks to a structural declaration that use only the top level entity of the sensor.

Listing 4.2: Ultrasound sensor top level entity

```
entity sensore_example is
port(clk : in std_logic;
          ECHOin : in std_logic;
          enable_cont : in std_logic;
          trigger : out std_logic;
          output_LED : out std_logic_vector(1 downto 0) :=
                                                (others => '0');
```

```
          cont_auto : out integer);
end sensore_example;
```

The full developed code for the ultrasound sensor is included in the appendix.

4.2.2 Architecture

The architecture describes all the operations done by the model. The implemented architecture is a behavioural one, with a small part containing a structural description, used to import the top level entity developed for the sensors.

4.2.2.1 Signals declaration part

The first part of a behavioural architecture is typically characterized by the declaration of all the signal that will be used in the following processes. This signal can be interpreted almost as "real" electric signal, with some sort of inertial delay.
All the process are executed in parallel way, and all the statements in a process are sequential.

Listing 4.3: Signals declaration part

```
architecture manage_traffic of traffic_light is


signal clk : std_logic := '0';
signal count : std_logic_vector (24 downto 0) :=
                                    "00000000000000000000000000";


type state is (tr_st1,tr_st2,tr_st3,tr_st4,tr_st5);
signal tr_pres_state : state := tr_st1;
signal tr_next_state : state;


signal passing_signal_23 : std_logic := '0';
signal passing_signal_34 : std_logic := '0';
signal passing_signal_45 : std_logic := '0';
signal passing_signal_52 : std_logic := '0';


signal fake_light_traffic : std_logic_vector(2 downto 0) := "010";


signal contatore_del_verde : integer := 30;
signal contatore_del_giallo : integer := 5;


signal enable2 : std_logic := '1';
```

```
signal enable3 : std_logic := '1';
signal enable4 : std_logic := '1';
signal enable5 : std_logic := '1';


signal cont_auto1 : integer := 0;
signal cont_auto2 : integer := 0;
signal cont_auto3 : integer := 0;
signal cont_auto4 : integer := 0;
signal cont_auto5 : integer := 0;
signal cont_auto6 : integer := 0;
signal cont_auto7 : integer := 0;


signal sum_st2 : integer := 0;
signal sum_st3 : integer := 0;
signal sum_st4 : integer := 0;
signal sum_st5 : integer := 0;


signal enable_sensore1 : std_logic;
signal enable_sensore2 : std_logic;
signal enable_sensore3 : std_logic;
signal enable_sensore4 : std_logic;
signal enable_sensore5 : std_logic;
signal enable_sensore6 : std_logic;
signal enable_sensore7 : std_logic;


signal variab_temp : integer;
```

In order to understand the following part of the project, a description of the signals is required. In particular:

- clk and count signals : these signals are used in the first process developed, able to generate, starting from the internal 50 Mhz clock, a 1 Hz clock that will be used in most of all processes implemented;
- tr_pres_state and tr_next_state signals : in accordance to the declaration of the new data type done in previous line, these signals are used in order to have the indication of the current state in which the FSM is and the next state in which it will move;
- passing_signal : these are very important signals for the whole controller behaviour. Indeed, the generation of these signal allows the FSM to move from one state to another when the signal has a logic value '1'.
  The logic chosen is as follows: *passing_signal_ij* allows the FSM to switch from the state i to the state j.

The absence of the passing_signal_12 in this part of code is due to fact that it is a particular external input, but its meaning and use follows the logic described;

- fake_light_traffic : signal used when the FSM is in the idle state in order to have a flashing yellow light for all the traffic lights;
- contatore_del_verde and contatore_del_giallo : they set the normal amount of time during the green light and the yellow light will be on;
- enables signal : particular signal used in the main process;
- cont_auto signals : these signals are used in order to have an indication of the amount of car detected by each of the seven sensor;
- sum_st signals : signal containing the information of the amount of car detected in every state. The sum_st_1 is missing because in the idle state the sensor are off;
- enable_sensore signals : signals that allows sensors to work or not;
- variab_temp : signal used in the main process.

The last part before the list of all the processes regards the import of the top level entity designed for the ultrasound sensor. In this way, you can use the interface of the sensor, simply mapping in the current project the I/O structure of the sensor.

Listing 4.4: Structural sensor declaration

```
component sensore_example is
port(clk : in std_logic;
    ECHOin : in std_logic;
    enable_cont : in std_logic;
    trigger : out std_logic;
    output_LED : out std_logic_vetor(1 downto 0) := (others => '0');
    cont_auto : out integer);
end component;
```

4.2.2.2 Ultrasound sensors structural description part

The import of the components ends the first part of the behavioural architecture. In this way the second part can start, in which there are all the processes that describe the controller.
This part starts mapping the signals of the traffic light's top level entity with the I/O structure of the sensor's top level entity. The syntax is as follows:

Listing 4.5: Mapping process

```
A1 : sensore_example port map(clk_fpga,ECHO_in_pin(0),enable_sensore1,
                              trigger_pin(0),debug_led1,cont_auto1);
A2 : sensore_example port map(clk_fpga,ECHO_in_pin(1),enable_sensore2,
                              trigger_pin(1),debug_led2,cont_auto2);
```

```
A3 : sensore_example port map(clk_fpga,ECHO_in_pin(2),enable_sensore3,
                              trigger_pin(2),debug_led3,cont_auto3);
A4 : sensore_example port map(clk_fpga,ECHO_in_pin(3),enable_sensore4,
                              trigger_pin(3),debug_led4,cont_auto4);
A5 : sensore_example port map(clk_fpga,ECHO_in_pin(4),enable_sensore5,
                              trigger_pin(4),debug_led5,cont_auto5);
A6 : sensore_example port map(clk_fpga,ECHO_in_pin(5),enable_sensore6,
                              trigger_pin(5),debug_led6,cont_auto6);
A7 : sensore_example port map(clk_fpga,ECHO_in_pin(6),enable_sensore7,
                              trigger_pin(6),debug_led7,cont_auto7);
```

Focusing on the first mapping process, you can see for example that the ECHOin pin in the structural sensor declaration is mapped with the LSB of the vector ECHO_in_pin declared in the traffic light's top level entity.
In this way, doing this process for all the seven sensors, you can have information, generated by the sensors, mapped into the signals or I/O structure of the traffic light's top level entity.

4.2.2.3 Five main processes implemented

The main part of the project is represented by the implementation of the five processes that make adaptive behavior possible.

The first implemented process generates, starting from 50 MHz internal clock passed in the sensitivity list of the process, a square wave signal whose period is one second. In other word, the process generates a new clock having 1 Hz frequency. This is done by means of a counter that counts the rising edge of the internal clock. When the counter reach a settled value, the toggle of the clk signal happens and the counter is resettled.
This new clock is "saved" on the clk signal and is used in other process.

Listing 4.6: 1 Hz clock generationg process

```
clk_process : process(clk_fpga)
        begin
           if rising_edge(clk_fpga) then
                count <= count + '1';
                if count = "10111110101111000001111111" then
                        clk <= not clk;
                        count <= "00000000000000000000000000";
                end if;
        end if;
end process clk_process;
```

The second process states, for each state in which the FSM can be, what is the next state in which the FSM will move, in accordance with the correct passing signal generated. This is managed by means of multiple case-when constructs.

The last if-construct of the process implements the return in the idle state, regardless of the current state, when the external switch is switched off.

Listing 4.7: Process that calculate the next state

```
next_state_traffic_process : process(tr_pres_state,
passing_signal_12,passing_signal_23,passing_signal_34,
passing_signal_45,passing_signal_52)

begin

        case tr_pres_state is
           when tr_st1 => case passing_signal_12 is
                            when '0' => tr_next_state <= tr_st1;
                            when '1' => tr_next_state <= tr_st2;
                          end case;

           when tr_st2 => case passing_signal_23 is
                            when '1' => tr_next_state <= tr_st3;
                            when '0' => tr_next_state <= tr_st2;
                          end case;

           when tr_st3 => case passing_signal_34 is
                            when '1' => tr_next_state <= tr_st4;
                            when '0' => tr_next_state <= tr_st3;
                          end case;
           when tr_st4 => case passing_signal_45 is
                            when '1' => tr_next_state <= tr_st5;
                            when '0' => tr_next_state <= tr_st4;
                          end case;
           when tr_st5 => case passing_signal_52 is
                            when '1' => tr_next_state <= tr_st2;
                            when '0' => tr_next_state <= tr_st5;
                          end case;
           end case;

        if passing_signal_12 = '0' then

              tr_next_state <= tr_st1;

        end if;

end process next_state_traffic_process;
```

The process just described only calculate the next state in which the FSM will move. For this reasons, a process that assign the next state value calculated to the new current state is required. In other word, a process that implements the real passage from one state to

another is needed.
For this reasons the following process has been implemented:

Listing 4.8: Process that allows the real passage throught the states

```
real_passaggio_next_to_pres_state_process : process(clk)
begin
    if rising_edge(clk) then
        tr_pres_state <= tr_next_state;
    end if;
end process real_passaggio_next_to_pres_state_process;
```

In order to have a clear visualization of the current state, a debug process has been implemented: the process, depending on the current state, is able to switch on or off three LEDs on the FPGA that represent, in binary format, the number of the current state.
This is done by means of a simple case-when construct.

Listing 4.9: Debug process for the visualization of the current state

```
vedere_stato_sui_led_process : process(tr_pres_state)
    begin
    case tr_pres_state is
        when tr_st1 => led_for_visual_state <= "001";
        when tr_st2 => led_for_visual_state <= "010";
        when tr_st3 => led_for_visual_state <= "011";
        when tr_st4 => led_for_visual_state <= "100";
        when tr_st5 => led_for_visual_state <= "101";
    end case;
end process vedere_stato_sui_led_process;
```

The fifth and last process is the most important one, due to the fact that there are all the operations done by the FSM in every state.
The process is too long, about 250 lines of code, so we preferred to insert it in the appendix. Here there is also the complete code of the controller to have a clearer view of the whole code.

Referring to the full code of the main process of the controller included in the listing 4.11 in the appendix, named *manage_every_traffic_state_process*, is important to understand the behavior implemented in the in the first two states.
In the idle state, the first operation is setting the green and yellow light counter to the default value. This is especially necessary when you return in this state with the external switch. Without these 2 lines of code you would not have the default value of counters for the next state.
Then, the sensors are all disabled and the values of counters referred to vehicles detected by the sensor are resettled. In this way, when we return in this state by means of the external switch, all the condition are resettled.

The next lines of code allows the flashing yellow light condition: every second (every 1 Hz clock's rising edge) the second bit of the vector fake_light_traffic, representing the yellow light, is toggled. In this way we have a flashing yellow light and, in order to have this behaviour for all the traffic lights, the signal is assigned to all the vectors representing the traffic lights, both for the vehicles both for pedestrian.

Here there is a description of the developed behaviour for the second state , that is the same for all the other states.
You can arrive in this state both from the idle state both from the state 5. If we arrive in this state from the idle state there are no problems, but in the second case two operations are needed. When the operations in the state 5 are concluded, the passing signal that allow the FSM to pass from state 5 to state 2 is set high. Then, as soon as you arrive in state 2, the passing_signal_52 is resettled, finishing the state switching.
This logic is recurrent : at the end of the operations done in a state, the passing signal is generated (settled high) and it is resettled as soon as you arrive in the next state.
After this operation , there is a setting operation on the signal enable. In every state i-th (i = 2:5) the main operations are done if the enable i-th is settled high. However, due to implementation reasons, at the end of every state the relative enable signal is resettled. For this reason is necessary to do a setting operation of that enable signal in the next state, in order to make possible to do the operations when the FSM will return in the state. This is the reason why in the state 2 there is the line *enable5 <= '1'*.
Later, the ultrasound sensors referred to this particular state are enabled. Then, the green light is on and the sensors start to detect the vehicles passing.
At this point there is the first implemented adaptive concept : if in the first ten seconds there are less then three passing cars (these parameters can be changed) means that there is no relevant traffic situation. Then there is no reason to continue to have a green light. The green light counter is set to 0 and the yellow lights are switched on, starting to decrement the yellow light counter.
It can be noted that the if-condition controls the value of variab_temp for the first nine seconds instead of ten seconds : this is related to the fact that, in general, the signals are evaluated in the following clock cycle, so there is a loss of a second that is compensated in this way.
Here we have the second implemented adaptive concept: the green light counter value that will be used for the next state is not fixed, but adapts to traffic conditions. In particular, the amount of time will be the sum of the default green light counter value (thirty seconds) plus one second for each car detected in the previous state 3. Obviously, this will not happen if in the next state 3 will pass less the 3 cars in the first 3 seconds.
After this important operation, the yellow light counter value is resettled, the enable2 signal is set to 0 avoiding the return in the main if-construct of this state for one second, the ultrasound sensors relative to this state are disabled and the passing signal allowing the passage from the state 2 to the state 3 is generated.

We can conclude that there are two main adaptive behaviours : the first one works when there is no traffic, with very short green light time duration ; the second one is referred to relevant traffic situation, with the green light time duration that depends, for each state, on the cars detected in that state previously.

Chapter 5

CONCLUSION AND FUTURE DEVELOPMENTS

An adaptive FPGA based traffic light controller has been successfully implemented using VHDL code on the Altera DE-10 Lite FPGA. A breadboard based circuit test, based on LEDs, resistors and ultrasound sensors, has also been implemented in order to have a real implementation of the project.
All the goals have been achieved, allowing an optimal traffic control for a quite complicated four-way and seven lane traffic light junction.

The two adaptive implemented behaviours, one for low traffic density and one in case of high traffic density, are possible thanks to seven ultrasound sensors, able to detect the vehicles flow for each travel lane during a green light time interval.
This allows better performance in terms of average waiting and traffic light switching time compared to the conventional pre-timed traffic light controller, overcoming their limitations.
The design of the proposed controller is done in order to be easy to realize and implement, allowing it to be easily modified by incorporating changes based on the topology of an area.

For the future use, the designed controller could be improved implementing the traffic lights for pedestrians, not tested in our project because of lack of available pins on the FPGA, or implementing other more efficient traffic control's behaviours, achievable using more sensors on each travel lane. This sensors could be used also in case of red light in order to have more information on the traffic state.
An integration with a cameras system could be also an improvement of the reliability of the system, with the use, for example, of the computer vision tools.

BIBLIOGRAPHY

- Simrit Kaur et al., *Adaptive Traffic Light Controller using FPGA*,2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology;
- S. Venkata Kishore et al., *FPGA based Traffic Light Controller*, International Conference on Trends in Electronics and Informatics;
- WM El-Medany, MR Hussain, *FPGA-Based Advanced Real Traffic Light Controller System Design*, IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications;
- Mohammad Ali et al., *Real-time Density-Based Dynamic Traffic Light Controller Using FPGA*, 2021 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE);
- DE10-Lite User Manual *https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/Boards/DE10-Lite/DE10_Lite_User_Manual.pdf* ;
- Ultrasonic Ranging Module HC S4-04 datasheet, *https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf* ;
- Quartus® Prime Design Software website, *https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html* ;
- Course Material of Digital Programmable Systems;

APPENDIX

Listing 5.1: Full code of the ultrasound sensor

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity sensore_example is
port(clk : in std_logic;
    ECHOin : in std_logic;
    enable_cont : in std_logic;
    trigger : out std_logic;
    output_LED : out std_logic_vector(1 downto 0) := (others => '0');
    cont_auto : out integer);
end sensore_example;



architecture behav_arch of sensore_example is

signal microsec : std_logic := '0';
signal counter : std_logic_vector(17 downto 0);
signal fake_cont_display : integer := 0;
signal enable : std_logic := '1';

begin



clock_process : process(clk)
variable count0 : integer := 0;
begin
        if(rising_edge(clk))then
                if(count0=24)then
                        microsec<= not microsec;
                        count0 := 0;
```

```
                end if;
                count0 := count0 + 1;
        end if;
end process clock_process;


process(microsec)
variable count1: integer:=0;
begin
        if rising_edge(microsec) then
                if count1 = 0 then
                        counter<="0000000000000000";
                        trigger <='1';
                elsif count1 = 10 then
                        trigger <='0';
        end if;
        if ECHOin = '1' then
                counter<=counter+1;
        end if;
        if count1 = 249999 then
                count1:=0;
        else
                count1:=count1+1;
        end if;
        end if;
end process;


process(ECHOin,enable_cont)

begin
if enable_cont = '1' then
    if falling_edge(ECHOin) then
        if (counter < 2321 and counter > 291 and enable = '1') then

                fake_cont_display <= fake_cont_display + 1;

                enable <= '0';
                output_LED <= "01";

        elsif(counter > 2321 and enable = '0')then
                output_LED <= "10";
                enable <= '1';

        end if;
    end if;
```

```
else
    fake_cont_display <= 0;
    output_LED <= "11";

end if;
cont_auto <= fake_cont_display;
end process;


end behav_arch;
```

Listing 5.2: Full code of the designed controller

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity traffic_light is
        port(clk_fpga : in std_logic;
        passing_signal_12 : in std_logic;
        light_traffic1 : out std_logic_vector(2 downto 0);
        light_traffic2 : out std_logic_vector(2 downto 0);
        light_traffic3 : out std_logic_vector(2 downto 0);
        light_traffic4 : out std_logic_vector(2 downto 0);
        light_traffic5 : out std_logic_vector(2 downto 0);
        light_traffic6 : out std_logic_vector(2 downto 0);
        light_traffic7 : out std_logic_vector(2 downto 0);

        --people_light_traffic1 : out std_logic_vector(2 downto 0);
        --people_light_traffic2 : out std_logic_vector(2 downto 0);
        --people_light_traffic3 : out std_logic_vector(2 downto 0);
        --people_light_traffic4 : out std_logic_vector(2 downto 0);

        debug_led1 : out std_logic_vector(1 downto 0);
        debug_led2 : out std_logic_vector(1 downto 0);
        debug_led3 : out std_logic_vector(1 downto 0);
        debug_led4 : out std_logic_vector(1 downto 0);
        debug_led5 : out std_logic_vector(1 downto 0);
        debug_led6 : out std_logic_vector(1 downto 0);
        debug_led7      : out std_logic_vector(1 downto 0);

        led_for_visual_state : out std_logic_vector(2 downto 0);
```

```
        ECHO_in_pin : in std_logic_vector(6 downto 0);
        trigger_pin : out std_logic_vector(6 downto 0));
end traffic_light;

architecture manage_traffic of traffic_light is


signal clk : std_logic := '0';
signal count : std_logic_vector (24 downto 0) :=
                                    "0000000000000000000000000";


type state is (tr_st1,tr_st2,tr_st3,tr_st4,tr_st5);
signal tr_pres_state : state := tr_st1;
signal tr_next_state : state;


signal passing_signal_23 : std_logic := '0';
signal passing_signal_34 : std_logic := '0';
signal passing_signal_45 : std_logic := '0';
signal passing_signal_52 : std_logic := '0';


signal fake_light_traffic : std_logic_vector(2 downto 0) := "010";


signal contatore_del_verde : integer := 30;
signal contatore_del_giallo : integer := 5;


signal enable2 : std_logic := '1';
signal enable3 : std_logic := '1';
signal enable4 : std_logic := '1';
signal enable5 : std_logic := '1';


signal cont_auto1 : integer := 0;
signal cont_auto2 : integer := 0;
signal cont_auto3 : integer := 0;
signal cont_auto4 : integer := 0;
signal cont_auto5 : integer := 0;
signal cont_auto6 : integer := 0;
signal cont_auto7 : integer := 0;


signal sum_st2 : integer := 0;
```

```
signal sum_st3 : integer := 0;
signal sum_st4 : integer := 0;
signal sum_st5 : integer := 0;


signal enable_sensore1 : std_logic;
signal enable_sensore2 : std_logic;
signal enable_sensore3 : std_logic;
signal enable_sensore4 : std_logic;
signal enable_sensore5 : std_logic;
signal enable_sensore6 : std_logic;
signal enable_sensore7 : std_logic;


signal variab_temp : integer;

component sensore_example is
port(clk : in std_logic;
    ECHOin : in std_logic;
    enable_cont : in std_logic;
    trigger : out std_logic;
    output_LED : out std_logic_vetor(1 downto 0) := (others => '0');
    cont_auto : out integer);
end component;

-------------------------------------------------
-- START OF ALL PROCESSES
-------------------------------------------------

begin

A1 : sensore_example port map(clk_fpga,ECHO_in_pin(0),enable_sensore1,
                              trigger_pin(0),debug_led1,cont_auto1);
A2 : sensore_example port map(clk_fpga,ECHO_in_pin(1),enable_sensore2,
                              trigger_pin(1),debug_led2,cont_auto2);
A3 : sensore_example port map(clk_fpga,ECHO_in_pin(2),enable_sensore3,
                              trigger_pin(2),debug_led3,cont_auto3);
A4 : sensore_example port map(clk_fpga,ECHO_in_pin(3),enable_sensore4,
                              trigger_pin(3),debug_led4,cont_auto4);
A5 : sensore_example port map(clk_fpga,ECHO_in_pin(4),enable_sensore5,
                              trigger_pin(4),debug_led5,cont_auto5);
A6 : sensore_example port map(clk_fpga,ECHO_in_pin(5),enable_sensore6,
                              trigger_pin(5),debug_led6,cont_auto6);
A7 : sensore_example port map(clk_fpga,ECHO_in_pin(6),enable_sensore7,
                              trigger_pin(6),debug_led7,cont_auto7);
```

```
clk_process : process(clk_fpga)
        begin
           if rising_edge(clk_fpga) then
                count <= count + '1';
                if count = "10111110101111000001111111" then
                        clk <= not clk;
                        count <= "00000000000000000000000000";
                end if;
        end if;
end process clk_process;


next_state_traffic_process : process(tr_pres_state,
passing_signal_12,passing_signal_23,passing_signal_34,
passing_signal_45,passing_signal_52)

begin

        case tr_pres_state is
           when tr_st1 => case passing_signal_12 is
                            when '0' => tr_next_state <= tr_st1;
                            when '1' => tr_next_state <= tr_st2;
                          end case;

           when tr_st2 => case passing_signal_23 is
                            when '1' => tr_next_state <= tr_st3;
                            when '0' => tr_next_state <= tr_st2;
                          end case;

           when tr_st3 => case passing_signal_34 is
                            when '1' => tr_next_state <= tr_st4;
                            when '0' => tr_next_state <= tr_st3;
                          end case;
           when tr_st4 => case passing_signal_45 is
                            when '1' => tr_next_state <= tr_st5;
                            when '0' => tr_next_state <= tr_st4;
                          end case;
           when tr_st5 => case passing_signal_52 is
                            when '1' => tr_next_state <= tr_st2;
                            when '0' => tr_next_state <= tr_st5;
                          end case;
           end case;

        if passing_signal_12 = '0' then

                tr_next_state <= tr_st1;
```

```
        end if;

end process next_state_traffic_process;



real_passaggio_next_to_pres_state_process : process(clk)
begin
    if rising_edge(clk) then
        tr_pres_state <= tr_next_state;
    end if;
end process real_passaggio_next_to_pres_state_process;



vedere_stato_sui_led_process : process(tr_pres_state)
    begin
    case tr_pres_state is
        when tr_st1 => led_for_visual_state <= "001";
        when tr_st2 => led_for_visual_state <= "010";
        when tr_st3 => led_for_visual_state <= "011";
        when tr_st4 => led_for_visual_state <= "100";
        when tr_st5 => led_for_visual_state <= "101";
    end case;
end process vedere_stato_sui_led_process;



manage_every_traffic_state_process : process(tr_pres_state,clk)

begin
if rising_edge(clk) then

    case tr_pres_state is

        when tr_st1 =>
                contatore_del_verde <= 30;
                contatore_del_giallo <= 5;

                enable_sensore1 <= '0';
                enable_sensore2 <= '0';
                enable_sensore3 <= '0';
                enable_sensore4 <= '0';
                enable_sensore5 <= '0';
                enable_sensore6 <= '0';
```

```
            enable_sensore7 <= '0';

            sum_st2 <= 0;
            sum_st3 <= 0;
            sum_st4 <= 0;
            sum_st5 <= 0;
            variab_temp <= 0;

            fake_light_traffic(1) <= not fake_light_traffic(1);

            light_traffic1 <=  fake_light_traffic;
            light_traffic2 <=  fake_light_traffic;
            light_traffic3 <=  fake_light_traffic;
            light_traffic4 <=  fake_light_traffic;
            light_traffic5 <=  fake_light_traffic;
            light_traffic6 <=  fake_light_traffic;
            light_traffic7 <=  fake_light_traffic;
            --people_light_traffic1 <= fake_light_traffic;
            --people_light_traffic2 <= fake_light_traffic;
            --people_light_traffic3 <= fake_light_traffic;
            --people_light_traffic4 <= fake_light_traffic;

    when tr_st2 =>
            passing_signal_52 <= '0';
            enable5 <= '1';

           if enable2 = '1' then
            enable_sensore2 <= '1';
            enable_sensore5 <= '1';


            contatore_del_verde <= contatore_del_verde-1;
            variab_temp <= variab_temp + 1 ;
            light_traffic1 <=  "100";
            light_traffic2 <=  "001";
            light_traffic3 <=  "100";
            light_traffic4 <=  "100";
            light_traffic5 <=  "001";
            light_traffic6 <=  "100";
            light_traffic7 <=  "100";
            sum_st2 <= cont_auto2 + cont_auto5;

            if variab_temp = 9 and sum_st2 <= 3 then
                    contatore_del_verde <= 0;
            end if;
```

```
            --people_light_traffic1 <=  "100";
            --people_light_traffic2 <=  "001";
            --people_light_traffic3 <=  "100";
            --people_light_traffic4 <=  "001";

            if contatore_del_verde <= 1 then

                    contatore_del_giallo <= contatore_del_giallo-1;
                    light_traffic1 <=  "100";
                    light_traffic2 <=  "001";
                    light_traffic3 <=  "100";
                    light_traffic4 <=  "100";
                    light_traffic5 <=  "010";
                    light_traffic6 <=  "100";
                    light_traffic7 <=  "100";
                    ----people_light_traffic1 <=  "100";
                    ----people_light_traffic2 <=  "010";
                    ----people_light_traffic3 <=  "100";
                    ----people_light_traffic4 <=  "010";
                    if contatore_del_giallo <= 1 then

                    contatore_del_verde <= 30 + integer((sum_st3));
                    contatore_del_giallo <= 5;
                    enable2 <= '0';
                    enable_sensore2 <= '0';
                    enable_sensore5 <= '0';
                    variab_temp <= 0;
                    passing_signal_23 <= '1';
                    end if;


            end if;
    end if;



    when tr_st3 =>
            passing_signal_23 <= '0';
            enable2 <= '1';
           if enable3 = '1' then
            enable_sensore1 <= '1';
            enable_sensore2 <= '1';

            contatore_del_verde <= contatore_del_verde-1;
            variab_temp <= variab_temp + 1 ;
```

```
            light_traffic1 <=  "001";
            light_traffic2 <=  "001";
            light_traffic3 <=  "100";
            light_traffic4 <=  "100";
            light_traffic5 <=  "100";
            light_traffic6 <=  "100";
            light_traffic7 <=  "100";
            sum_st3 <= cont_auto1 + cont_auto2;
            --people_light_traffic1 <=  "100";
            --people_light_traffic2 <=  "001";
            --people_light_traffic3 <=  "100";
            --people_light_traffic4 <=  "001";


            if variab_temp = 9 and sum_st3  <= 3 then
                    contatore_del_verde <= 0;
            end if;


            if contatore_del_verde <= 1 then
            contatore_del_giallo <= contatore_del_giallo-1;
            light_traffic1 <=  "010";
            light_traffic2 <=  "010";
            light_traffic3 <=  "100";
            light_traffic4 <=  "100";
            light_traffic5 <=  "100";
            light_traffic6 <=  "100";
            light_traffic7 <=  "100";
            --people_light_traffic1 <=  "100";
            --people_light_traffic2 <=  "010";
            --people_light_traffic3 <=  "100";
            --people_light_traffic4 <=  "010";
              if contatore_del_giallo <= 1 then
                    contatore_del_verde <= 30 + integer((sum_st4));
                    contatore_del_giallo <= 5;
                    enable3 <= '0';
                    enable_sensore1 <= '0';
                    enable_sensore2 <= '0';
                    variab_temp <= 0;
                    passing_signal_34 <= '1';
              end if;
            end if;
    end if;
```

```
        when tr_st4 =>
                passing_signal_34 <= '0';
                enable3 <= '1';

                if enable4 = '1'then
                enable_sensore4 <= '1';
                enable_sensore7 <= '1';

                contatore_del_verde <= contatore_del_verde-1;
                variab_temp <= variab_temp + 1 ;
                light_traffic1 <=  "100";
                light_traffic2 <=  "100";
                light_traffic3 <=  "100";
                light_traffic4 <=  "001";
                light_traffic5 <=  "100";
                light_traffic6 <=  "100";
                light_traffic7 <=  "001";
                sum_st4 <= cont_auto4 + cont_auto7;
                --people_light_traffic1 <=  "001";
                --people_light_traffic2 <=  "100";
                --people_light_traffic3 <=  "001";
                --people_light_traffic4 <=  "100";

                  if variab_temp = 9 and sum_st4  <= 3 then
                        contatore_del_verde <= 0;
                  end if;

                if contatore_del_verde <= 1 then
                contatore_del_giallo <= contatore_del_giallo-1;
                light_traffic1 <=  "100";
                light_traffic2 <=  "100";
                light_traffic3 <=  "100";
                light_traffic4 <=  "010";
                light_traffic5 <=  "100";
                light_traffic6 <=  "100";
                light_traffic7 <=  "010";
                ----people_light_traffic1 <=  "010";
                ----people_light_traffic2 <=  "100";
                ----people_light_traffic3 <=  "010";
                ----people_light_traffic4 <=  "100";

                  if contatore_del_giallo <= 1 then
                        contatore_del_verde <= 30 + integer((sum_st5));
                        contatore_del_giallo <= 5;
                        enable4 <= '0';
                        enable_sensore4 <= '0';
```

```
                    enable_sensore7 <= '0';
                    variab_temp <= 0;
                    passing_signal_45 <= '1';
                end if;
            end if;
    end if;

        when tr_st5 =>
                passing_signal_45 <= '0';
                enable4 <= '1';

                if enable5 = '1' then
                enable_sensore3 <= '1';
                enable_sensore6 <= '1';

                contatore_del_verde <= contatore_del_verde-1;
                variab_temp <= variab_temp + 1;
                light_traffic1 <=  "100";
                light_traffic2 <=  "100";
                light_traffic3 <=  "001";
                light_traffic4 <=  "100";
                light_traffic5 <=  "100";
                light_traffic6 <=  "001";
                ight_traffic7 <=  "100";
                sum_st5 <= cont_auto3 + cont_auto6;
                --people_light_traffic1 <=  "001";
                --people_light_traffic2 <=  "100";
                --people_light_traffic3 <=  "001";
                --people_light_traffic4 <=  "100";

                  if variab_temp = 9 and sum_st5  <= 3 then
                        contatore_del_verde <= 0;
                  end if;


                if contatore_del_verde <= 1 then
                contatore_del_giallo <= contatore_del_giallo-1;
                light_traffic1 <=  "100";
                light_traffic2 <=  "100";
                light_traffic3 <=  "010";
                light_traffic4 <=  "100";
                light_traffic5 <=  "100";
                light_traffic6 <=  "010";
                light_traffic7 <=  "100";
                --people_light_traffic1 <=  "010";
                --people_light_traffic2 <=  "100";
```

```
                --people_light_traffic3 <=  "010";
                --people_light_traffic4 <=  "100";

                if contatore_del_giallo <= 1 then
                        contatore_del_verde <= 30 + integer(sum_st2);
                        contatore_del_giallo <= 5;
                        enable5 <= '0';
                        enable_sensore3 <= '0';
                        enable_sensore6 <= '0';
                        variab_temp <= 0;
                        passing_signal_52 <= '1';
                end if;
        end if;
    end if;

end case;
end if;
end process manage_every_traffic_state_process;

end manage_traffic;
```