1 The Story

This section is just for fun. You can skip it if you want. Technical descriptions start from Section 2.

After visiting Asteroids B-325 to B-330 and meeting some absurd adults, including the Lamp Lighter on Asteroid B-329, the Little Prince discovered another planet he could visit — Asteroid CS-173. Similar to all the other asteroids, there was an upset adult on this planet.

"Hi, I am Little Prince. Are you a resident of this asteroid?" asked the Little Prince.

"Oh, hi. Yes, I am a traffic police," answered the man. "Sorry, I am too busy to talk to you, little boy. I have to control all these traffic lights at this crossroad. This is a very important task. A tiny mistake of mine may cause a serious traffic accident."

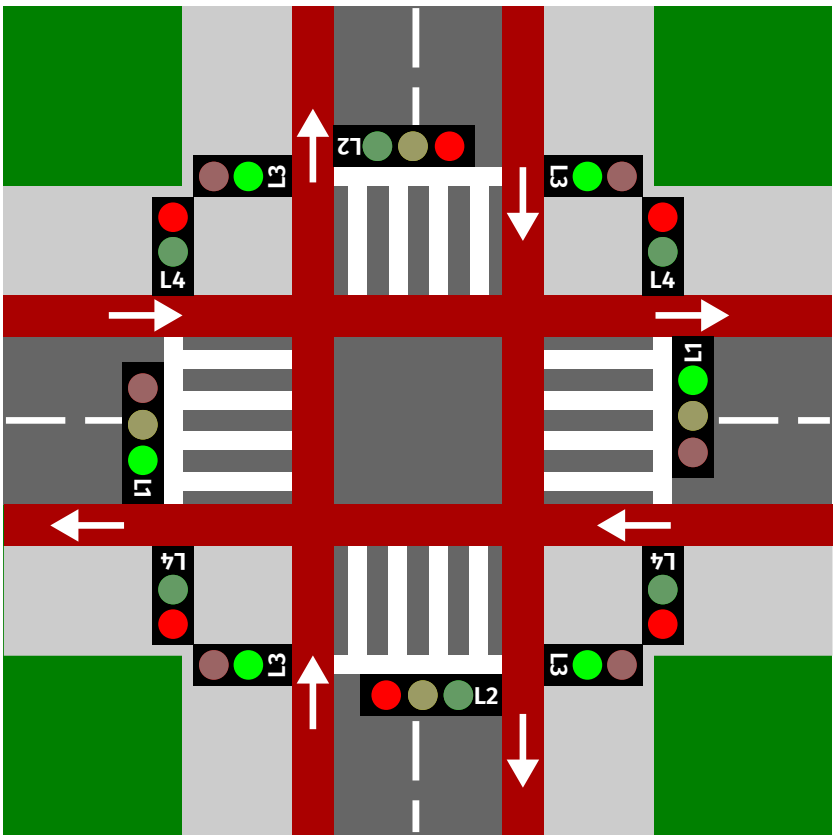


Figure 1: The crossroad on Asteroid CS-173.

"Hmm, why don't you automate them?" asked the Little Prince.

".....Sorry, what? Automate? How?" responded the absent-minded traffic police, not even looking at the Little Prince. "No, this is impossible. There are too many lights to control. You see, the two lights for vehicles cannot be green at the same time; otherwise, cars coming from different directions will crash into each other. Besides that, I also have to take care of the lights for pedestrians. They can only cross the road safely when the cars are not moving in this direction, so I have to turn on the red lamp of the corresponding light....."

The Little Prince watched him turning the lights on and off while murmuring continuously.

"Moreover, there is also the complication of rush hours. In the morning, people rush to schools and offices; and in the evening, people are tired and want to go home as early as possible. During these periods, they are especially impatient and cannot even bear waiting one more second for red lights."

The Little Prince looked puzzled. "What do you do with impatient people?"

"I make everything faster than usual. Green lights stay for a shorter time, so that people going in the other direction also wait for shorter red lights."

"Wow, brilliant! But..." The Little Prince was impressed at first, but then he noticed a problem. "A shorter time is maybe not a problem for cars because they are fast, but wouldn't that be too short for pedestrians to cross the road? What if there is a grandma who walks very slow?"

"Yes, that is indeed a problem. So I designed different rules for rush hours." The traffic police looked at the Little Prince for the first time. He looked much more enthusiastic now, with his eyes shining. "Imagine that you are a pedestrian who wants to go from here to that diagonal corner. Normally, you have to cross two roads separately, waiting twice for red lights. That takes a lot of time." The Little Prince nodded in agreement.

"So, I invented a novel way: I stop all the vehicles with red lights on both directions, and turn on both of the green lights for pedestrians. This way, you can cross the crossroad diagonally."

"Ah, that is so smart," praised the Little Prince.

"Thank you." The traffic police smiled, but the sparks in his eyes faded away quickly. "I like my job, actually, but it is also exhausting. I have to keep concentrated at every moment. I have to make sure I don't get anything wrong."

"You know who is really good at doing things accurately for a long time without being exhausted?" asked the Little Prince.

"You? Are you a superman?" The traffic police was confused.

"No, not me," laughed the Little Prince. "Machines! I saw a lot of those on Earth."

"But I have no idea how they work," sighed the traffic police. "How do I teach the machine my brilliant traffic light system?"

"I think I know some people who can help you with that," said the Little Prince, getting on his aircraft at the same time. "I'll bring you good news. See you soon!" The Little Prince flew away Asteroid CS-173 with excitement.

2 System Specifications

In this TP, we are going to implement a traffic light system which controls two traffic lights for vehicles (L1 and L2) and two traffic lights for pedestrians (L3 and L4), as shown in Figure 1. The mapping on the Gecko4Education board is shown in Figure 2. The traffic lights for vehicles have a green lamp, a yellow lamp, and a red lamp. The traffic lights for pedestrians have only green and red lamps, but is equipped with a seven-segment display for counting down.

- On Asteroid CS-173, there are d seconds in a day (d is a constant parameter). Instead of using hours and minutes, people report time in seconds on this planet. For example, "let's meet at 10s" means we meet at 10 seconds after midnight (at the beginning of the eleventh second). "ds today" is the same time as "0s tomorrow". (Like in the real world, 24h today is equivalent to 0h tomorrow.) It is guaranteed that $10 \leq d \leq 500$.

- Each day, there are two rush-hour periods: from r_1s to r_2s (i.e., starting r_1s and ending before r_2s) and from r_3s to r_4s. Every day is the same; there are no weekends or holidays. It is guaranteed that $1 < r_1 < r_2 < r_3 < r_4 < d$.

- During normal hours (i.e., not in rush-hour periods), each traffic cycle is c_n seconds, whereas during rush hours, each traffic cycle is c_r seconds. The following constraints are ensured to hold:

$$\begin{aligned} d + r_1 - r_4 \mod c_n &= 0, \\ r_2 - r_1 \mod c_r &= 0, \\ r_3 - r_2 \mod c_n &= 0, \\ r_4 - r_3 \mod c_r &= 0. \end{aligned}$$

  Also, $1 \leq c_n \leq 99$ and $1 \leq c_r \leq 99$. Note that the start of a day (i.e., 0s) does not necessarily match to the start of a cycle.

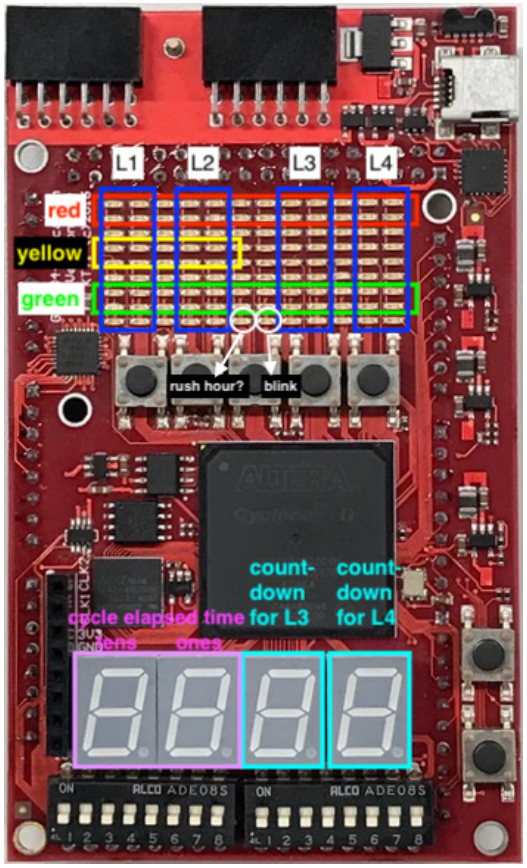


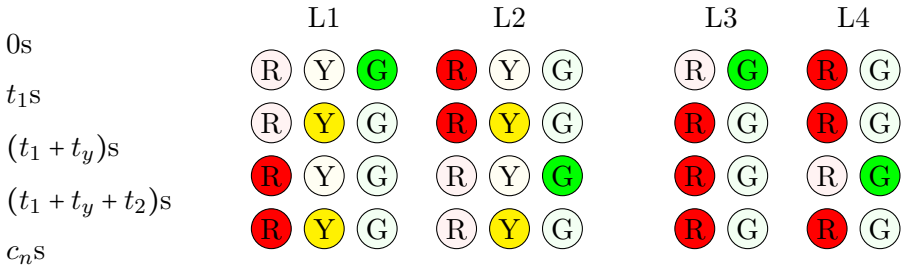
Figure 2: Traffic lights mapped on the Gecko4Education board.

- In each traffic cycle during normal hours, the following happens in sequence (also shown in Figure 3a):

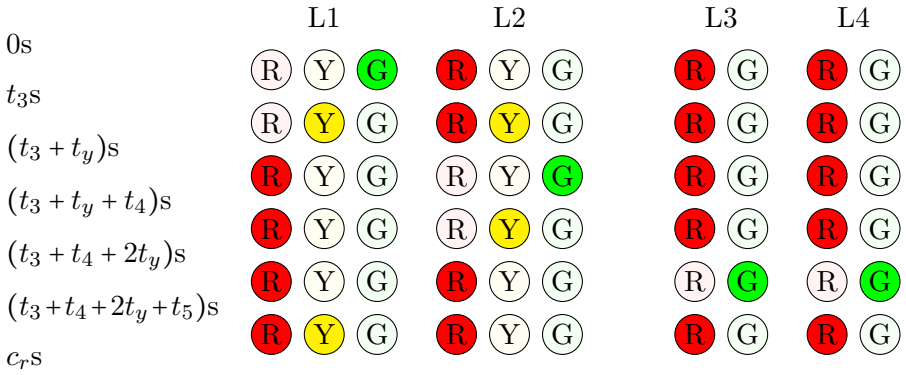
  1. L1 and L3 are green for t_1 seconds (the red and yellow lamps of L1 and the red lamp of L3 are dim), while L2 and L4 are red (the green and yellow lamps of L2 and the green lamp of L4 are dim).
  2. L1 turns yellow (its red and green lamps are dim) and L3 turns red for t_y seconds. At the same time, the yellow lamp of L2 turns on (its red lamp stays bright and green lamp stays dim) and L4 stays red.
  3. L1 and L3 are red for t_2 seconds (the green and yellow lamps of L1 and the green lamp of L3 are dim), while L2 and L4 turn green (the red and yellow lamps of L2 and the red lamp of L4 are dim).
  4. The yellow lamp of L1 turns on (its red lamp stays bright and green lamp stays dim) and L3 stays red for t_y seconds. At the

same time, L2 turns yellow (its red and green lamps are dim) and L4 turns red.

It is ensured that $t_1 + t_2 + 2 \cdot t_y = c_n$.



(a) Light transition during normal hours.



(b) Light transition during rush hours.

Figure 3: Traffic light transitions.

- In each traffic cycle during rush hours, the following happens in sequence (also shown in Figure 3b):

  1. For t_3 seconds, L1 is green while L2, L3 and L4 are red.
  2. L1 turns yellow and the yellow lamp of L2 turns on (its red lamp stays bright and green lamp stays dim) for t_y seconds, while L3 and L4 stay red.
  3. L1 turns red and L2 turns green for t_4 seconds, while L3 and L4 stay red.
  4. L2 turns yellow for t_y seconds, while L1, L3 and L4 stay red.
  5. L2 turns red and L3 and L4 turn green for t_5 seconds, while L1 stays red.

6. The yellow lamp of L1 turns on (its red lamp stays bright and green lamp stays dim) and L3 and L4 turn red for t_y seconds, while L2 stays red.

It is ensured that $t_3 + t_4 + t_5 + 3 \cdot t_y = c_r$.

- Whenever L3 or L4 is green, their seven-segment displays show the remaining green-time in seconds. For example, in the beginning of a normal-hour cycle, the seven-segment display of L3 counts down from t_1 to 0. As soon as the count-down goes to 0, L3 turns red and the seven-segment display turns off (hence we actually cannot see the "0"). Assume that t_1, t_2 and t_5 are smaller than 10 so that one decimal digit is always enough.

In addition to the main traffic light system, there are a few signals required for easier debugging and grading:

- An LED blinking every second, being bright in the first half second and dim in the second half. (You can assume the number of clock cycles in a second is even, thus there is no rounding issue.)
- An LED showing whether it is currently a rush hour or not.
- Two seven-segment displays showing the elapsed time in each traffic cycle, i.e., how many seconds have passed after the start of the current cycle.

Upon reset, the system should start being at r_1s of a day, i.e., at the beginning of the first rush-hour period.

3 General Instructions and Rules

Please download the archive *tp78.zip*. It contains the entity files (suffixed with *-entity.vhd*) and architecture files (suffixed with *-rtl.vhd*) for all the modules used in this TP. To make sure that the automated grader recognizes the modules and interfaces correctly, please do not change any of the file, entity, architecture, or port names and write the architecture descriptions in the specified files only.

The solution is to be submitted for grading one module at a time, by uploading the architecture source file (*-rtl.vhd) to **https://digsys.epfl.ch**. The total number of uploads allowed is 24, which makes 3 attempts per module, on average. We will not impose a hard limit on the number of

attempts per module. The only constraint is that the total does not exceed 24.

The top-level module that directly interfaces with the FPGA board is `tp78`. It provides a power-on reset (POR) signal, connects to the FPGA's 50 MHz clock generator, and instantiates the `system` entity, where you will implement your design.

The entity `system` has two inputs and five outputs:

- `clk: in std_logic` is the 50 MHz system clock.
- `reset: in std_logic` is the power-on reset.
- `leds: out std_logic_vector(0 to 107)` connects to the LED array.
- `disp_tens: out std_logic_vector(0 to 7)` connects to the seven-segment display for the tens digit of the cycle elapsed time.
- `disp_ones: out std_logic_vector(0 to 7)` connects to the seven-segment display for the ones digit of the cycle elapsed time.
- `disp_L3: out std_logic_vector(0 to 7)` connects to the seven-segment display for L3.
- `disp_L4: out std_logic_vector(0 to 7)` connects to the seven-segment display for L4.

The values of all the needed parameters $(d, r_1, r_2, r_3, r_4, c_n, c_r, t_1, t_2, t_3, t_4, t_5, t_y)$ are defined in `packages/parameters.vhd`. Please note that the values used in the grader may be different from the given ones and your design should work for any possible values within the ranges given in Section 2. The value of how many clock cycles there are in a "second" is also defined in this file as a parameter, which is not 50 million because we want to speed up the video to fit in 30 (real) seconds. You can change the values of these parameters when you do simulations for testing (for example, use a smaller value for `ticks_per_second`), but please revert to the given values when synthesizing the bitstream for video recording.

You are not allowed to use multiplication, division, modulo or remainder operators. This will be checked by searching for the following strings in your code: "`*`", "`/`", " `mod` " and " `rem` " (the last two with preceding and tailing spaces). So, please make sure you do not have these strings even in the comments. The addition (`+`) and subtraction (`-`) operators are allowed.

4 VHDL Skeleton

This section describes the structure of the provided VHDL code skeleton and gives you some hints on how the complicated system can be divided into small modules. Each task (module) is tested separately in the auto-grader, so you can conquer them one by one. Before submitting any code, simulate it with Modelsim as in TP5 and observe the waveform to make sure it works correctly.

4.1 Counting the Time

Besides the reset signal, the only input to the system is the 50 MHz clock signal. Hence, the first task is to get some useful information from the clock. Roughly speaking, the whole traffic light system is based on counting *seconds*. Hence, first we want to extract the second-pulses from the ultra-fast clock.

Task 4.1 Clock divider (15 points)

We have seen a clock divider in TP6 with a blinking LED. Implement a similar clock divider in `clock_divider-rtl.vhd` that outputs:

- A `pulse` signal that is 1 for one clock cycle at the end of every second, and stays at 0 for the rest of time.
- A `blink` signal that is 1 in the first half of each second and 0 for the second half.

Make sure that you do not hard-code the constant for the clock frequency, but use `ticks_per_second` defined in `parameters.vhd` instead.

Next, we need a finite state machine to count the second-pulses and manage which traffic cycle the system is currently at.

Task 4.2 Cycle manager (20 points)

Implement a cycle manager in `cycle_manager-rtl.vhd` which counts the `pulse` generated by the clock divider and outputs:

- Whether it is rush hour now.
- How many seconds have passed since the start of the current traffic

cycle.

Recall that the state upon reset should be at the beginning of the first rush-hour period.

4.2 Managing the Traffic Lights

There are two types of traffic cycles, namely rush hours and normal hours, which behave very differently. Thus, we focus on one type at a time, implement the control logic of the traffic lights for normal and rush hours separately, and then combine the signals together with multiplexers. Note that these modules do not receive the clock and reset signals, meaning that they are fully combinational.

Task 4.3 Light manager for normal hours (10 points)

Implement the light manager for normal hours in `light_manager_normal-rtl.vhd`.

Task 4.4 Light manager for rush hours (10 points)

Implement the light manager for rush hours in `light_manager_rush-rtl.vhd`.

Task 4.5 Light manager (5 points)

Instantiate the two specialized light managers and choose the correct signals based on which type of traffic cycle it is currently (signal `is_rush_hour`). (Implement `light_manager-rtl.vhd`)

4.3 Displays

The entity `led_driver` maps the traffic light signals derived from the light manager to the LED array on the FPGA board. This entity is already implemented and instantiated in `system` for you.

For the seven-segment displays for L3 and L4, we need to convert from the binary-coded decimal (BCD) digits derived from the light manager to the seven LEDs on the display, similarly as we did in TP1.

Task 4.6 Seven-segment display (10 points)

Implement the encoder for seven-segment displays in `bcd_to_7seg-rtl.vhd`.

4.4 Putting Everything Together

Now that all needed components are implemented, it is time to put everything together. **It is highly recommended to test each individual modules separately and thoroughly before going into this final step.** The most basic debugging strategy is to always test from smaller modules and assemble the system layer by layer.

In `system-rtl.vhd`, some modules are already instantiated and their pins are connected, including `led_driver` mentioned earlier and `bin_to_bcd` which splits an 8-bit binary number into two 4-bit numbers representing the tens and ones digits in decimal. For example, $00011001_2 = 25_{10}$ is split into $0010_2 = 2$ and $0101_2 = 5$. This module is implemented for you and you are encouraged to look into its architecture and understand how it works.

Task 4.7 The traffic light system (5 points)

Instantiate all your modules in `system-rtl.vhd` and connect their input/output pins together.

5 Simulation and Hardware Testing

After passing all the testbenches in the auto-grading system, the last 25 points come, again like in TP3/4, from a video demonstrating that it also works on the real board.

Task 5.8 Hardware demonstration (25 points)

Synthesize the whole system with Quartus and upload the bitstream file (`.sof`) to the RFA job on Jenkins to produce a video of the traffic light system working on the Gecko4Education board. The maximum number of submissions is 50. Note that only the last submitted video before the deadline and within 50 trials will be graded.

Full or partial points will be given by matching your video and auto-grader results with either one of the following cases:

- Passed the test for clock divider and the video demonstrates the blinking LED: 5 points.
- Passed the tests for clock divider and cycle manager (at least the `is_rush_hour` part) and the video demonstrates the blinking LED and the rush-hour indicator LED: 8 points.
- Passed the tests for clock divider, cycle manager and seven-segment decoder and the video demonstrates the blinking LED, the rush-hour indicator LED, and the cycle elapsed time: 15 points.
- Passed all tests on the grader and the video demonstrates the entire system working: 25 points.

To synthesize the bitstream, follow the instructions in TP6 to create a new Quartus project. Choose `tp78/quartus/` as the working directory. The top-level design is `tp78`. Remember to add all the files in `tp78/entities/`, `tp78/architectures/` and `tp78/packages/` to the project. Choose the same board as in TP6 (`EP4CE30F23C8`). The needed TCL scripts are already provided, so there is no need to download from the website. Simply run `tp78/quartus/tp78.tcl` and the pin assignments should be done. Click on the compile button to run the synthesis and generate the bitstream file. If the compilation is successful, you will find the SOF file in `tp78/quartus/output_files/`.