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`)

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/`.