In this lab, you will design a base 5 modulo 125_{10} 3-digit counter. Binary encodings of the base 5 digits are given in the table below.

Digit	Code
$0_{B=5}$	100_b
$1_{B=5}$	001_b
$2_{B=5}$	110_b
$3_{B=5}$	011_b
$4_{B=5}$	111_b

Table 1: Binary encoding of digits.

Unused binary sequences are considered *don't cares*.

Please download the template file called *tp34.circ* from the moodle page of the course and use it for your design. The top level module of the file is named $TP34$. You must not change that name nor make any other module top. You must also neither modify nor rename any of the components existing in the template file.

Components that you can use in the design are: NOT, AND, OR, NAND, NOR, XOR, and XNOR gates, D-Flipflops, any wiring and I/O components, and subcircuits that you created. If you use any other components, the total number of points awarded to your design will be halved.

From the moodle page of the course, you may also download the board mapping file TP34-GECKO4_EDUCATION_EPFL_EDITION-MAP.xml, that you can use to check if all the interfaces remain unchanged. If you try to download the design to Gecko4Education using this mapping file, all of the interfaces should be in the mapped list.

Solution .circ file is to be submitted for automated grading at the following link: **https://digsys.epfl.ch**. You are allowed to make ten submissions, out of which only the last one will be counted. Functionality of each of the questions below assumes functionality of all the previous questions. Hence, the grader will check for functionality of questions in reverse order, and award the points for the first correctly answered one and all the preceding ones. Note that it is only necessary to upload the file containing the solution of the last question that you have solved.

The last 25 points will be awarded after successful demonstration of the functionality of the last question correctly answered, to the teaching assistants, on Gecko4Education.

A Note on Using the Grader
Besides producing a brief report with the scores awarded, the grader outputs two plain text files for each graded question: *golden_qN.txt* and

trace_qN.txt, where N is the question number. Both files contain the values of the output signals for each clock cycle of the simulation. Outputs of the submitted solution are dumped into *trace_qN.txt*, while those of the reference model are stored in *golden_qN.txt*. The console output of the grader will point you to the first cycle at which a mismatch between the submitted solution and the reference model occurred. You may then use any file comparison tool to track the issue. For instance, running *vim* with a *-d* switch on the two files will open them side by side, highlighting the mismatches (e.g., you may run *vim -d golden_q1.txt trace_q1.txt*).

From the moodle page of the course you may also download a small script for trace manipulation named *tracer.py*. This allows you to hide some signals, letting you concentrate on the ones that are problematic. The script can also convert the outputs to decimal numbers. Besides being useful for catching errors in state changes more quickly, this can also help you detect small mistakes such as bit permutation, which will result in conversion failures. To use the script, call it with *python tracer.py -cmd sig_name input_file -o output_file*. Where *cmd* can be either *hide* for hiding a signal or *dcd* for converting it to a decimal value; *sig_name* is the name of the signal that is being manipulated (only one at each call); *input_file* is the file that is to be manipulated and *outfile* is an optional destination file. If the destination file is not specified, the output will be printed on the screen. Piping is not supported. To hide or convert multiple signals, you can simply call the script again on the intermediate files.

Question 1 [20 pts]: Design an upward counter that counts from $000_{B=5}$ to $444_{B=5}$ in a loop, and displays the count on 7-segment displays included in the template file. Display $Disp2$ should display the most significant digit, and display $Disp0$ the least significant one. Furthermore, bit $j \in [0, 3)$ of the encoding of digit $i \in [0, 3)$ should be displayed on LED Q_{ij}, of the template. For example, LED Q_{22} should display the most significant bit of the encoding of the most significant digit. System should be reset by the button $reset$. State upon reset should be $202_{B=5}$ and the counter should start counting upwards. Clock generator from the template should be the only clock generator used.

Question 2 [20 pts]: Extend the functionality of the counter, so that it counts upwards while button But from the template is pressed and downwards while it is released. Again, state upon reset should be $202_{B=5}$, but now the counting direction should depend on whether the button is pressed or not.

Question 3 [35 pts]: Change the functionality of the counter so that briefly pressing the button But from the template achieves the following: Once the button is pressed, a change in the direction of counting is scheduled for the clock cycle immediately after the next transition to the value $123_{B=5}$. For example, if the counter is counting upwards at the time of pressing the button, and holds the value $121_{B=5}$, a partial sequence of values observed in the future is: $121_{B=5}, 122_{B=5}, 123_{B=5}, 122_{B=5}, 121_{B=5}, 120_{B=5}$. You can assume that the button will not be pressed when the counter is in the state $123_{B=5}$. Hence, it

is up to you to define the behavior in that case. State upon reset should be $202_{B=5}$ and the counter should start counting upwards.