Queen's University
Department of Electrical and Computer Engineering
ELEC 271 Digital Systems
Fall 2019

Lab 3:
Logic Circuits with Multiple Flip-Flops and Optimization of Functions with Don't-Care Cases

Objectives

This laboratory activity for *ELEC 271 Digital Systems* provides the opportunity for students to:

- implement logic circuits with multiple flip-flops, and
- perform logic optimization using Karnaugh maps in the presence of don't-care cases.

This activity involves the use of the worksheet included in the description. *Each student must have a paper copy of the worksheet ready at the beginning of the laboratory session*, and each student must complete it (with reasonable neatness) during the session.

Part 1: Sum-of-Products Optimization with Don't-Cares

- Building on previous activity, this exercise consider multiple flip-flops and logic optimization with don't-care cases. It combines various concepts in the course related to both combinational logic and sequential logic to convey representative system design that achieves specified behavior.
- The circuit to be considered for this exercise consists of four flip-flops and four combinational logic blocks that are connected to the D inputs of those flip-flops, as shown in the diagram on page 4.
- The four flip-flops share the same clock, asynchronous clear, and load-enable inputs, but the D and Q connections are unique. Thus, the flip-flops can be viewed collectively as a 4-bit *register*. In VHDL, the *std_logic_vector* type provides a convenient means of representing registers. In this exercise, however, the flip-flops will be defined as a collection of single-bit instances of *std_logic*. The reason is purely a practical one for simpler signal identification in this exercise that avoids the use of parentheses for identifying a specific bit, which would be necessary with *std_logic_vector*.
- The circuit has the Q outputs feeding back as inputs to the combinational logic blocks. In more general circuits, signals from external sources could also be inputs to the combinational logic. Nonetheless, the circuit in this exercise represents the fact that actual digital systems involve combinational logic for generating the inputs to flip-flops, and the flip-flops themselves that hold state information from one clock cycle to the next.
- Optimized sum-of-products expressions will be developed for the four comb. logic blocks so that on successive clock edges, the flip-flop outputs generate a repeating sequence of binary patterns.
- Upon assertion of the active-low reset signal, the flip-flop outputs will be forced to an initial pattern.
- On each successive rising clock edge after reset is deasserted, the flip-flops will capture the values provided at their inputs from the combinational logic block outputs.
- *During* each cycle after an edge, the combinational logic blocks will use the current flip-flop outputs to generate the *next* D input values that will be captured on the next rising clock edge.
- A desired sequence of binary patterns will be provided (including an initial pattern for the reset case) as the specification for the combinational circuit design to be done during your session.
- From specifications, a truth table will be developed. The input valuation in each row will represent *current* flip-flop outputs *q3*…*q0*. The function outputs will be the *next* data input values *d3*…*d0*.
- Only a binary patterns will be specified for *q3*…*q0, hence there will not be sixteen truth-table rows.* In other words, *the four functions will be incompletely specified* (slide 17 in Optimized Logic).
- Therefore, the remaining input valuations that are missing in the left-hand side of the table can potentially be treated as *don't-care cases* to provide flexibility for enhanced logic optimization.
- From the truth table, four Karnaugh maps will be developed for *d3*, *d2*, *d1*, and *d0*, with the pair *q3*/*q2* down the left side of the Karnaugh map, the pair *q1*/*q0* across the top of the map.
- For each function output *d3* to *d0,* you will write the *specified 0 and 1 output values* in the output column of the table into Karnaugh map cells. *The empty cells will then represent don't-care cases*.
- Normally, it is useful to leave 0 cells blank for sum-of-products optimization. But in this case, you are being instructed to write the few 0 cases into the Karnaugh map so that both the 0s and 1s formally defined by the incomplete truth table are distinguished from the empty cells that, in this case, *represent missing input valuations for the incompletely specified functions* (not 0 outputs).
- With appropriate judgement, an inspection of each Karnaugh map will lead to treating some of the don't-cares as 1s in order to create the largest possible groups for logic optimization. Don't care cells selected for treatment as 1 for optimization should be filled with ***'d'*** labels to document the the decisions, and to distinguish them from the actual 1 cells from the truth table. Remaining don't-cares that are not useful for making larger groups will be treated as 0s and left empty. Doing so distinguishes the don't-cares-treated-as-0s from the actual-output-0s from the truth table.
- Find the **largest groups** with 1 and *'d'* cells, *while still respecting the 0 and empty cells*.

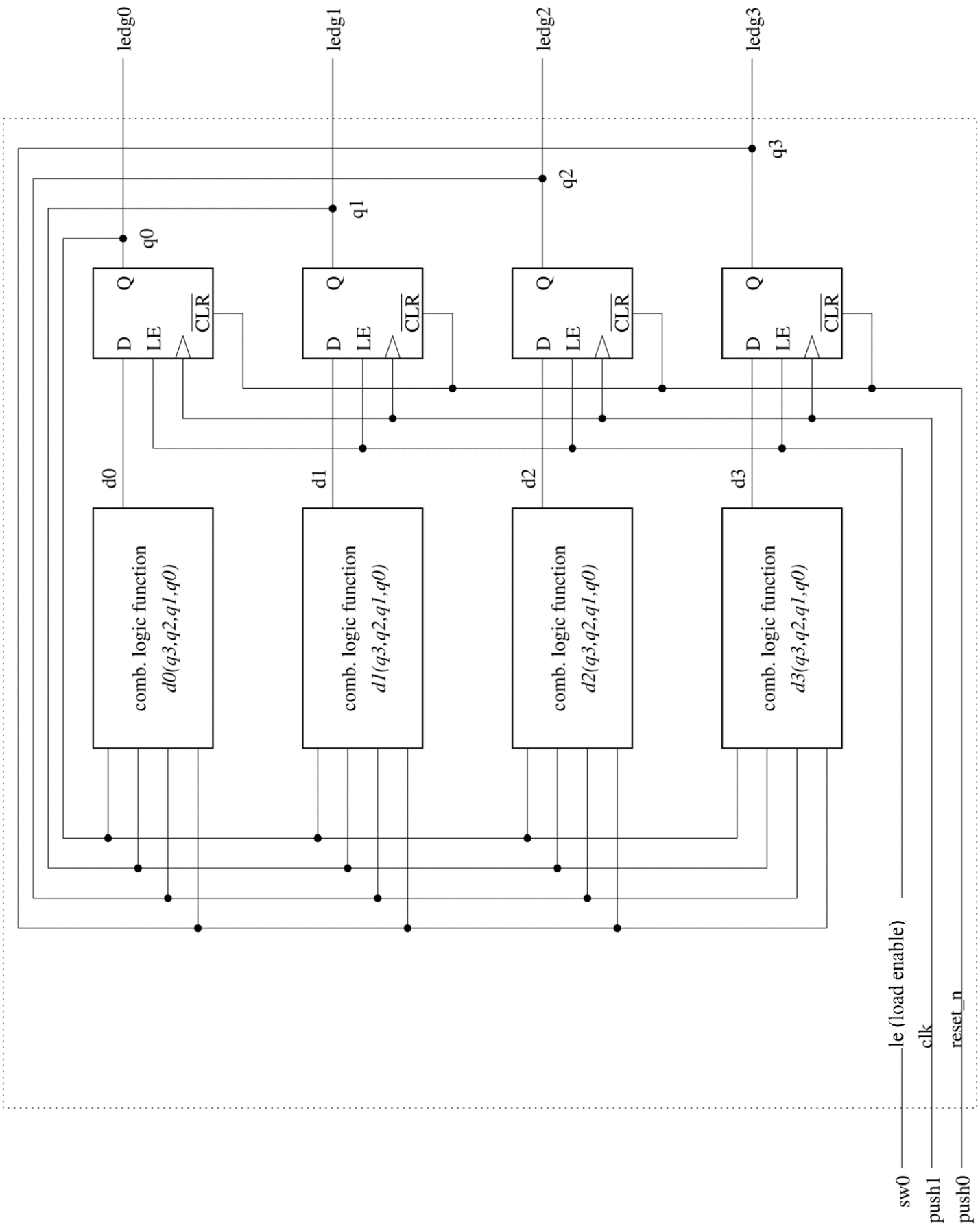
- After writing the simplified sum-of-products expressions for the outputs *d3* to *d0* on the worksheet, they are to be transformed into VHDL signal assignment statements for inclusion in the description that is provided on page 5. The *lab3.vhd* file should have been prepared before the session.
- Note how a single process is used for describing the behavior of all four flip-flops. They share the same clock, reset, and load-enable connections. Signal assignment statements in the *then* parts of the *if* statements handle the unique D and Q connections.
- Using the Quartus software, create a project called *lab3* in a *LAB3* folder that contains the VHDL file *lab3.vhd*. Perform all of the usual project-creation steps *including the selection of the proper Cyclone III chip as well as the desired configuration for the unused pins and voltage standard.*
- Synthesize the circuit so that the Pin Planner will be populated with the input/output port names.
- Open the Pin Planner and consult the Terasic DE0 User Manual to complete the necessary pin assignments for SW0, BUTTON1, BUTTON0, LEDG3, LEDG2, LEDG1, and LEDG0.
- Resynthesize the circuit and program the chip.
- Using BUTTON0 for reset, BUTTON1 for the clock, and SW0 for the load-enable control, test the circuit to verify that it cycles through the specified binary patterns correctly as clock edges are applied. Note that pressing *and holding* BUTTON1 produces the low (logic-0) value for the clock, then *releasing* BUTTON1 creates the low-to-high (positive) edge that triggers the flip-flops.

Credit Checkpoint

- ☐ completed VHDL file with signal assignment statements to implement logic circuit
- ☐ demonstration of circuit behavior in custom logic simulator
- ☐ proper printed worksheet, completed by each student

Part 2: Product-of-Sums Optimization with Don't-Cares

- Having optimized and tested four output functions in sum-of-products representation using some of the missing cases in the truth table as don't-cares, the final step is to repeat the optimization with don't-cares but this time for product-of-sums representation.
- ***It is not necessary to create new Karnaugh maps.***
- Having applied good judgement for sum-of-products optimization with don't cares, the largest possible groups of cells with actual **1**s and **d**s treated as 1s should have been identified.
- The remaining cells are actual **0**s and (empty) **d**s treated as 0s. Therefore, the simplified product-of-sums representation with don't cares can be derived directly from the Karnaugh maps previously generated in Part 1.
- The simplified sum terms can be written simply by inspection from the original Karnaugh maps, i.e., by focusing on the actual 0s and the empty cells *outside* the groups marked for Part 1.
- A second blank line has been provided on the worksheet for the "Result" under each Karnaugh map. For Part 2, write the simplified product-of-sums representation for each function directly beneath the solution derived earlier in Part 1.
- If it is helpful in deriving the simplified functions for Part 2, the existing Karnaugh maps can first be marked up (lightly) with *dotted* or *dashed* outlines for new groups of actual **0**s and (empty) **d**s treated as 0s. These new groups should **not** overlap with any of the groups from Part 1. Then, the simplified sum terms for written for each of the new groups.
- Finally, comment out the sum-of-product functions in *lab3.vhd*, insert product-of-sums functions, recompile the project, and test the behavior on the hardware. *Demonstrate the working circuit.*


ledg0
ledg1
ledg2
ledg3
q0
q1
q2
q3
Q
D
LE
CLR
d0
d1
d2
d3
comb. logic function
d0(q3,q2,q1,q0)
d1(q3,q2,q1,q0)
d2(q3,q2,q1,q0)
d3(q3,q2,q1,q0)
le (load enable)
clk
reset_n
sw0
push1
push0

This template should be placed in a file *lab3.vhd*, then completed with simplified functions *d3* to *d0*.

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

entity lab3 is
  port (
    clk, reset_n : in std_logic;
    le : in std_logic;
    ledg3, ledg2, ledg1, ledg0 : out std_logic
  );
end entity;

architecture logic of lab3 is

signal d3, d2, d1, d0 : std_logic;  -- flip-flop d inputs
signal q3, q2, q1, q0 : std_logic;  -- flip-flop q outputs

begin

-- single process encompassing all four flip-flops
-- (i.e., a register, but without using std_logic_vector type)

the_flipflops : process (clk, reset_n)
begin
  if (reset_n = '0') then
    q3 <= ???;  q2 <= ???; q1 <= ???; q0 <= ???;
  elsif (clk'event and clk = '1') then
    if (le = '1') then
      q3 <= d3; q2 <= d2; q1 <= d1; q0 <= d0;
    end if;
  end if;
end process;

-- combinational functions that generate
-- the new d value from the current q outputs
-- (optimized by exploiting don't-care cases)

d3 <= ?????;
d2 <= ?????;
d1 <= ?????;
d0 <= ?????;

-- associate the q outputs inside the architecture
-- with the output ports for the LED pins

ledg3 <= q3;
ledg2 <= q2;
ledg1 <= q1;
ledg0 <= q0;

end architecture;
```

Combined truth table for outputs of four combinational logic functions derived from given sequence:

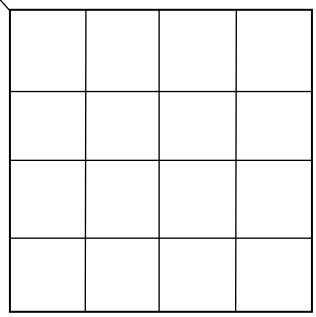
Write the q3…q0 sequence below.

inputs				outputs			
q3	*q2*	*q1*	*q0*	*d3*	*d2*	*d1*	*d0*

[It is recommended that you write the input valuations in *increasing binary order* as you construct the truth table. Doing so will make it easier to transfer the 0/1 function output values to the Karnaugh maps shown below.]

Karnaugh maps for the four functions to perform logic optimization *with don't-care cases*:

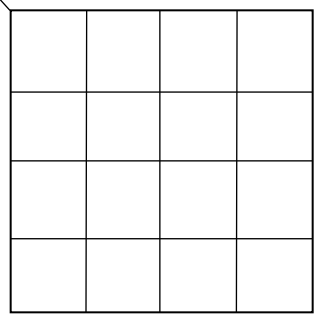
d3



Result: _____

_____

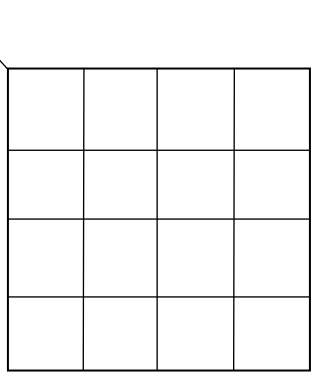
d2



Result: _____

_____

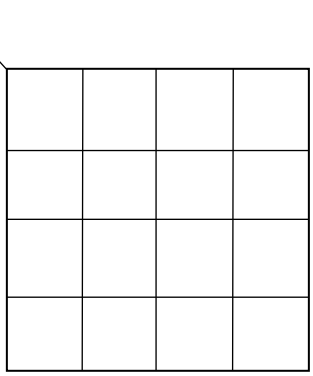
d1



Result: _____

_____

d0



Result: _____

_____