Queen's University
Department of Electrical and Computer Engineering
ELEC 271 Digital Systems
Fall 2018

Lab 4:
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.

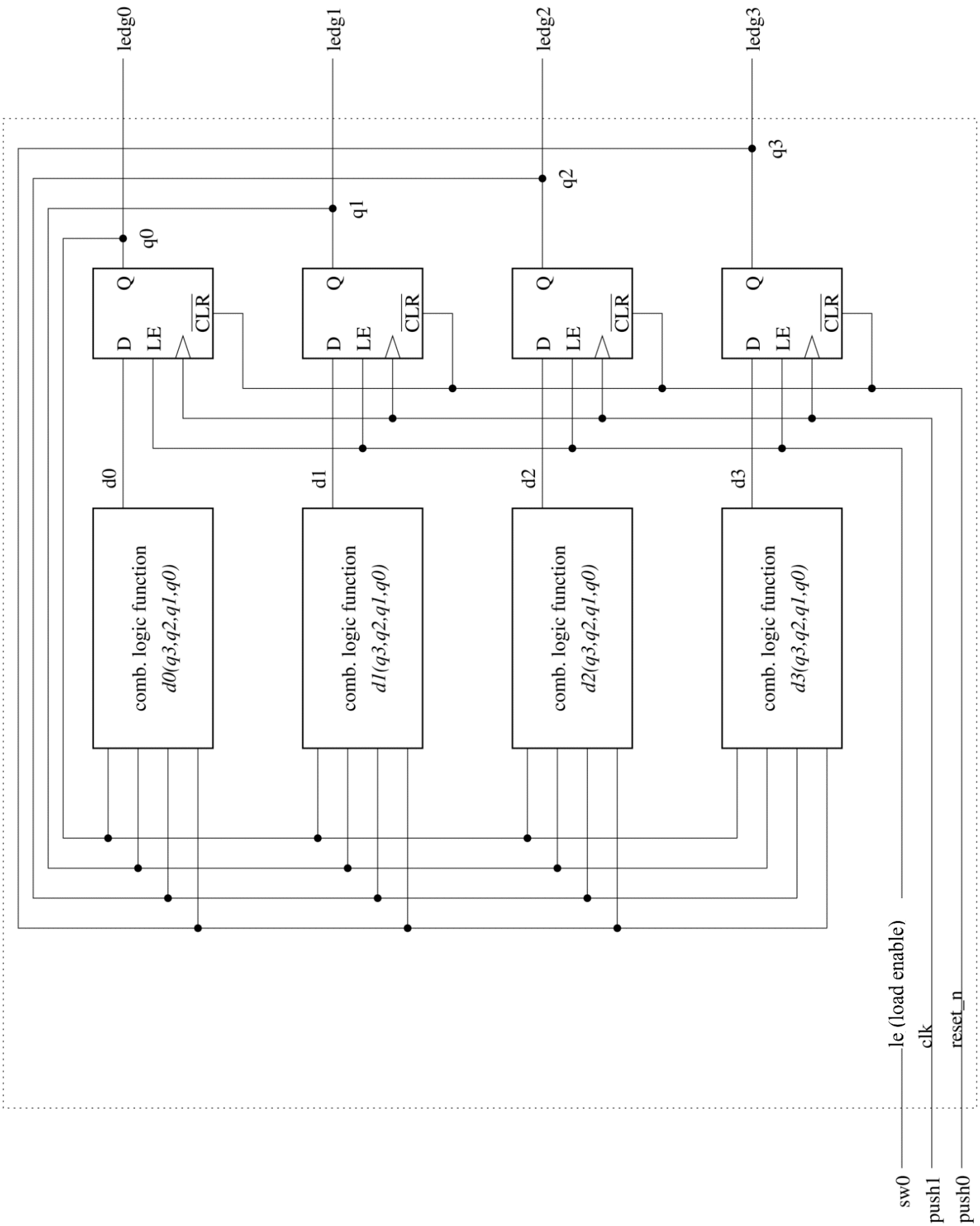
Overview of Relevant Concepts and Description of Procedure

- This exercise builds on previous activity to 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.
- The design problem is to develop optimized sum-of-products expressions for the four functions such that, over time (i.e., in successive clock cycles), the four 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 zero.
- On each successive clock edge after reset is deasserted, the flip-flops will capture the values provided at their inputs from the combinational logic blocks.
- During each cycle after an edge, the flip-flop outputs will be used by the combinational logic blocks to generate the next set of input values that will be captured on the next clock edge.
- A desired sequence of binary patterns will be provided (starting from 0000 that reflects the reset behavior) as the specification for the combinational circuit design to be done during your session.
- The sequence provided to you in your session will be used to generate a single combined truth table (see the worksheet) for the outputs of the four combinational logic blocks.
- The input valuation in each row of this truth table will represent *current* flip-flop outputs *q3*…*q0*. The four output columns will indicate the desired *next* data input values *d3*…*d0*.
- Only a few binary patterns will be specified for *q3*…*q0,* hence *there will <u>not</u> be sixteen rows in this truth table*. In other words, the four functions will be incompletely specified.
- Therefore, all of the other possible input valuations not listed in the left-hand side of the table can be treated as *don't-care cases* that allow more 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* serving as the column header and the pair *q1*/*q0* serving as the row header for each map.
- For each function output *d3* to *d0,* you will write the 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, *the 0 cells that are dictated by the truth table <u>must be shown explicitly</u> in the Karnaugh map because they set the limits on how much flexibility is available for optimization*.
- With appropriate judgement, an inspection of each Karnaugh map will lead to selection of a subset of the available don't-care cases to enable the largest groupings for logic optimization. <u>Don't care cells selected for treatment as 1 for optimization will be filled with *'d'* labels to document the choice but distinguish them from the actual 1 cells</u>. The empty cells not treated as 1 will be left empty,
- You will find the **largest groupings possible** with 1 and *'d'* cells, *while still respecting the 0 cells*.

- After writing the simplified logic expressions for the function outputs d3 to d0 on the worksheet, they will be transformed into VHDL signal assignment statements for inclusion in the description that is provided on page 5. Prepare a *lab4.vhd* file with the completed description.
- 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 *lab4* in a *LAB4* folder that contains the VHDL file *lab4.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 as 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-flop.

Credit Checkpoint

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


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 *lab4.vhd*, then completed with simplified functions *d3* to *d0*.

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

entity lab4 is
  port (
    clk, reset_n : in std_logic;
    le : in std_logic;
    ledg3, ledg2, ledg1, ledg0 : out std_logic
  );
end entity;

architecture logic of lab4 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 <= '0';  q2 <= '0'; q1 <= '0'; q0 <= '0';
  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:

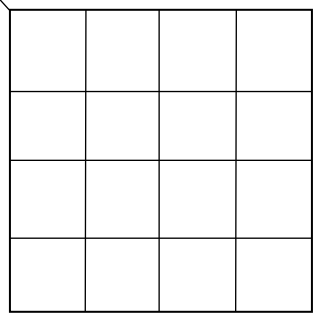
Write the q3…q0 sequence below.

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 complete the truth table so that it will be easier to transfer the 0/1 function output values to the Karnaugh maps shown below.]

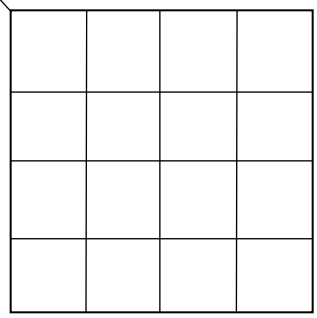
Karnaugh maps for the four functions to perform logic optimization with don't-care cases:

d3



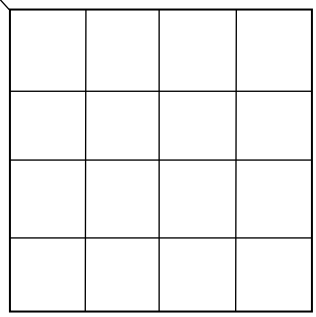
Result: _____

d2



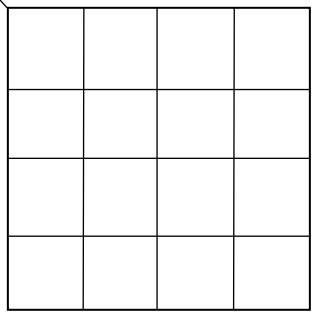
Result: _____

d1



Result: _____

d0



Result: _____