Name: 13th January 2025

CA Test — Variant 1

1. Consider a combinational device which has a 4-bit input denoted by ***i***, respectively a 4-bit output denoted by ***o***. If the binary representation of the input ***i*** is a palindrome value (e.g., the number 5 having the binary representation **101** designates a palindrome value) then the output ***o*** will be obtained by switching the *most significant half* with the *least significant half* of the input. When the input ***i*** is not a palindrome value, the output ***o*** is equal to the input ***i***.

a) Based on the aforementioned problem statement, complete the following truth table and perform the minimization procedure of the output functions using the Karnaugh maps:

Inputs				**Outputs**			
I_3	I_2	I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	?	?	?	?
0	0	0	1	?	?	?	?
0	0	1	0	?	?	?	?
0	0	1	1	?	?	?	?
0	1	0	0	?	?	?	?
0	1	0	1	?	?	?	?
0	1	1	0	?	?	?	?
0	1	1	1	?	?	?	?
1	0	0	0	?	?	?	?
1	0	0	1	?	?	?	?
1	0	1	0	?	?	?	?
1	0	1	1	?	?	?	?
1	1	0	0	?	?	?	?
1	1	0	1	?	?	?	?
1	1	1	0	?	?	?	?
1	1	1	1	?	?	?	?

b) Implement the module, called **pal_4b**, after the minimization using Verilog language.

c) Write a *testbench*, using Verilog language, for *exhaustive checking* of the **pal_4b** module.

Note: Subpoint (a) will be entirely solved on a sheet of paper while subpoints (b) and (c) will be solved in the Altera Modelsim environment.

***Score**: P**1.** a) 1p ; b) 1p ; c) 1p | P**2.** a) 1p ; b) 1p ; c) 1p | P**3.** a) 0.6p ; b) 0.6p c) 0.6p d) 0.6p e) 0.6p | 1 bonus point | **Total**: 10 p*

2. Consider a sequential circuit, representing a **Divide-By-9** counter that only counts the even numbers (excluding the odd values). The custom counter has a *clock* **(clk)** input, an asynchronous *reset* **(rst)** line, a synchronous *clear* **(clr)** input, and a *count up* **(c_up)** signal. The output *divided clock* (**dclk**) will be active each time the counter reaches the value **N - 1**.

a) According to the problem statement, complete the truth table construct of the *counter*:

Q (on 4 bits)				**D (on 4 bits)**			
$q[3]$	$q[2]$	$q[1]$	$q[0]$	$d[3]$	$d[2]$	$d[1]$	$d[0]$
0	0	0	0	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?

The output Q is connected to the input D in a feedback loop via logical gates that are used for updating the value of the counter.

b) Design, on a sheet of paper, the *hardware configuration* of the **Divide-By-9** counter.

c) Implement the **div_9_cntr** using the *instantiation procedure* in Verilog language.

Note: Subpoints (a) and (b) will be entirely solved on a sheet of paper while subpoint (c) will be solved in the Altera Modelsim environment.

3. Consider a custom input preprocessing unit, inspired by the 256-bit Secure Hash Algorithm 2 (SHA-2) preprocessing phase. The input preprocessing unit operates with packets of 4 bits and blocks of 64 bits. The initial message has **L** bits, **L** being a multiple of 8. The initial message is padded with 3 bits of 0 followed by **k** bits of 1s, so that: **L** + 3 + **k** ≡ 56 (mod 64). The size in bytes of the initial message is appended after the ***k*** bits of 1s, as a number represented on 8 bits. Each data packet is delivered, one at a time, via the output of a multiplexing unit, called **pkt_mux**. The **pkt_mux** module has an input **msg_len** on 8 bits (representing the message length), an input ***pkt*** on 4 bits (designating the initial message represented in ASCII format), and several selection lines denoted by **pad_pkt**, **zero_pkt**, **hi_mgln**, and **lo_mgln**. If **pad_pkt** is active the multiplexer will deliver 3 bits of 0 followed by a bit of 1 at its output. When **zero_pkt** is active the multiplexer will provide 4 bits of 0 at its output. If **hi_mgln** is active, the multiplexer will output the most significant half of the **msg_len** input. When **lo_mgln** is active, the multiplexer will provide the least significant half of the **msg_len** input. If none of the previously described selection lines is active, the **pkt_mux** module will deliver the **pkt** input at its output. Only one of the selection lines can be active at a time. Construct the custom input preprocessing unit, by implementing:

***Score**: P**1.** a) 1p ; b) 1p ; c) 1p | P**2.** a) 1p ; b) 1p ; c) 1p | P**3.** a) 0.6p ; b) 0.6p c) 0.6p d) 0.6p e) 0.6p | 1 bonus point | **Total**: 10 p*

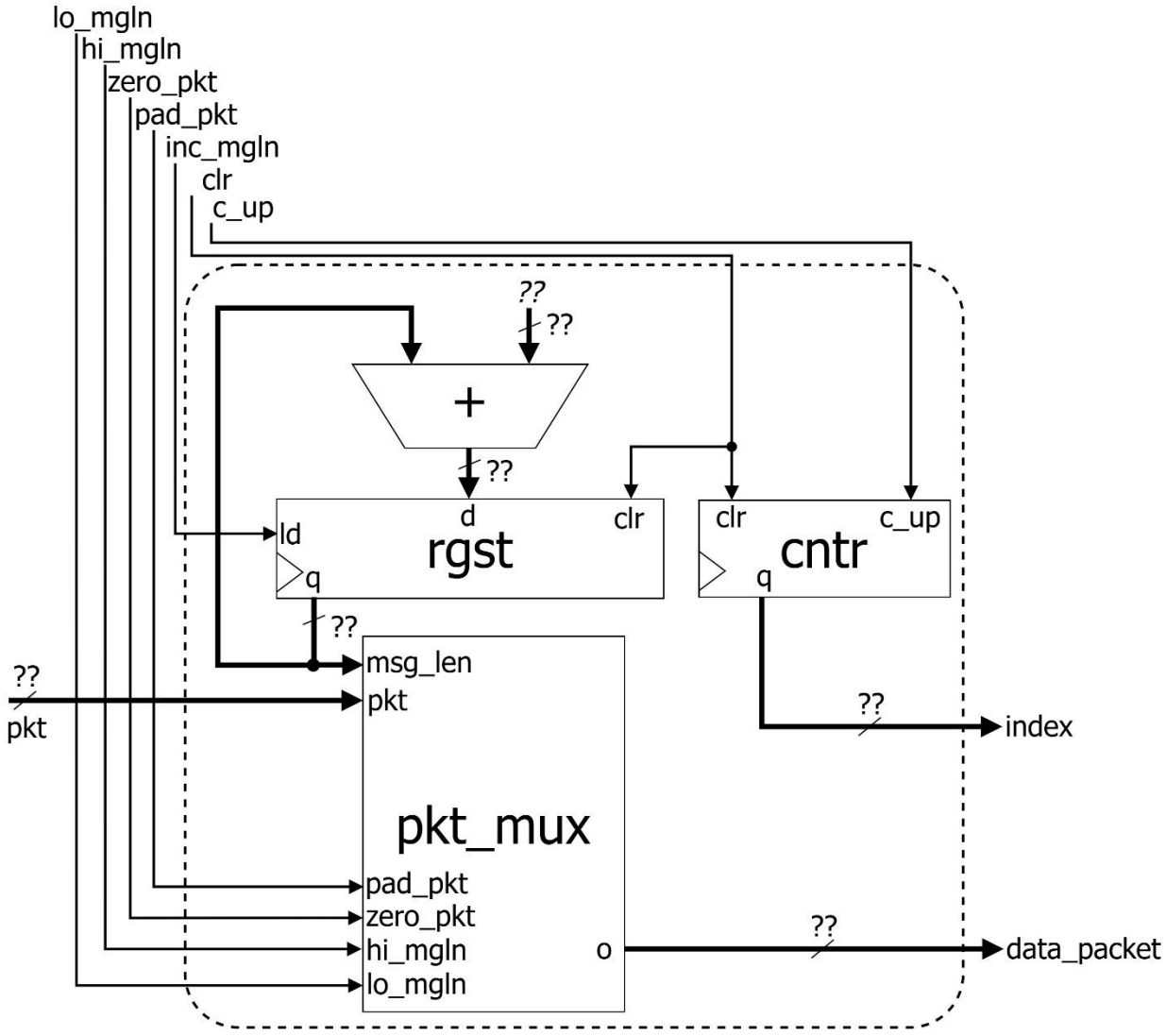
a) a parameterized **adder** module for incrementing the message length.

b) a parameterized **rgst** module for updating the value of the message length.

c) a parameterized **cntr** module for monitoring the data packet delivery.

d) a parameterized **pkt_mux** module for generating the data packets.

e) a parameterized **prep_unit** by instantiating the modules from the previous subpoints (a → d).



The clock and reset signals were omitted for clarity.

Note: Subpoints (a), (b), (c), (d), and (e) will be entirely solved in the Altera Modelsim environment.

***Score**: P**1.** a) 1p ; b) 1p ; c) 1p | P**2.** a) 1p ; b) 1p ; c) 1p | P**3.** a) 0.6p ; b) 0.6p c) 0.6p d) 0.6p e) 0.6p | 1 bonus point | **Total**: 10 p*