Giao Tran

johnson_counter:

1. Testbench

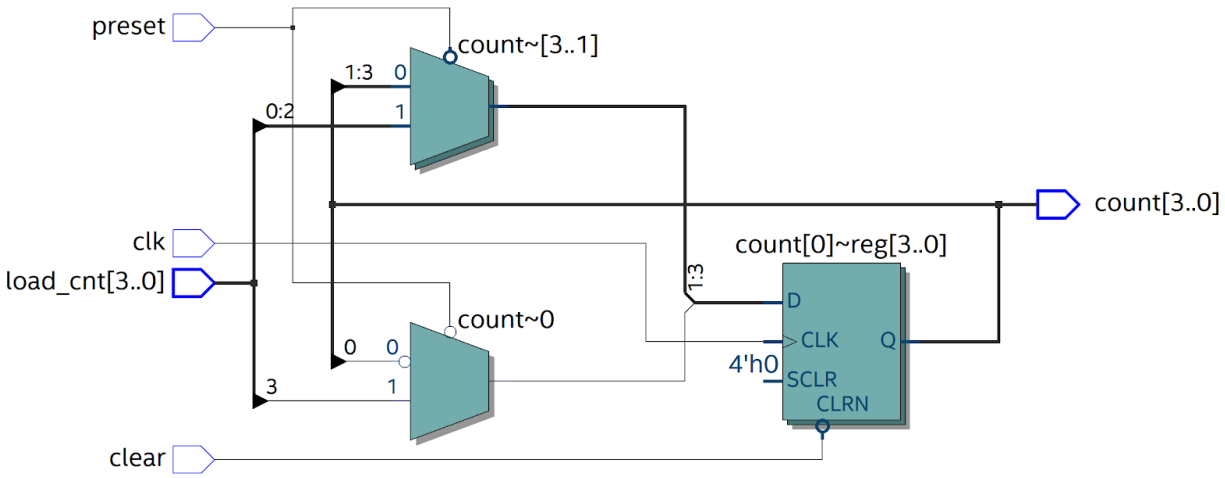
```
`timescale 1ns/1ns
//Johnson Counter Testbench Code
module johnson_counter_testbench;
 logic clock, reset, preset;
 logic[3:0] load_cnt, count;

// Instantiate design under test
johnson_counter design_instance(
.clk(clock),
.clear(reset),
.preset(preset),
.load_cnt(load_cnt),
.count(count)
);

initial begin
// Initialize Inputs
reset = 0;
preset = 1;
clock = 0;
load_cnt = 4'b0000;

// Wait 10 ns for global reset to finish and start counter
#10;
reset = 1;

#10;
preset = 0;
load_cnt = 4'b0000;

#20;
preset = 1;

// Wait for 200ns and reset counter
#340ns;
reset=0;

// Wait for 20ns and start counter again
#20ns;
reset=1;

#10;
preset = 0;
load_cnt = 4'b1000;

#10;
preset = 1;

// Wait for 10ns
#100ns;

// terminate simulation
$finish();
end

// Clock generator logic
always@(clock) begin
  #10ns clock <= !clock;
end

// Print input and output signals
initial begin
 $monitor(" time=%0t,  clear=%b  clk=%b  count=%d",$time, reset, clock, count);
end

endmodule
```

2. RTL model

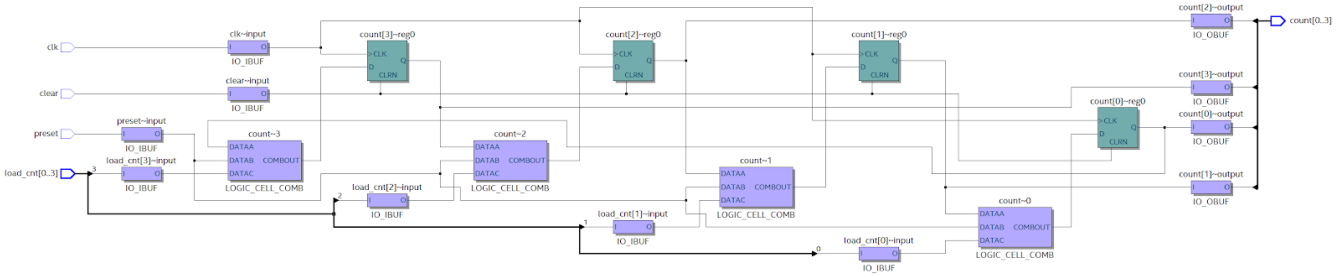
```
module johnson_counter (
    input logic clk, clear, preset,
    input logic [3:0] load_cnt,
    output logic [3:0] count
);

always @(posedge clk or negedge clear) begin
    if (!clear) count <=4'b0000;
    else if (!preset) count <=load_cnt;
    else begin
        count <= count>>1;
        count[3]<=~count[0];
        end
    end
endmodule: johnson_counter
```

The 4 bits johnson_counter code is very similar to that of the ring counter code, the only change is that the inverse of the output of the final shift register is fed as the input of the first shift register. Compared to the ring counter code, the only change in the johnson_counter code is the addition of the negation operator to inverse count[0]; count[3]<=~count[0].

3. RTL schematic

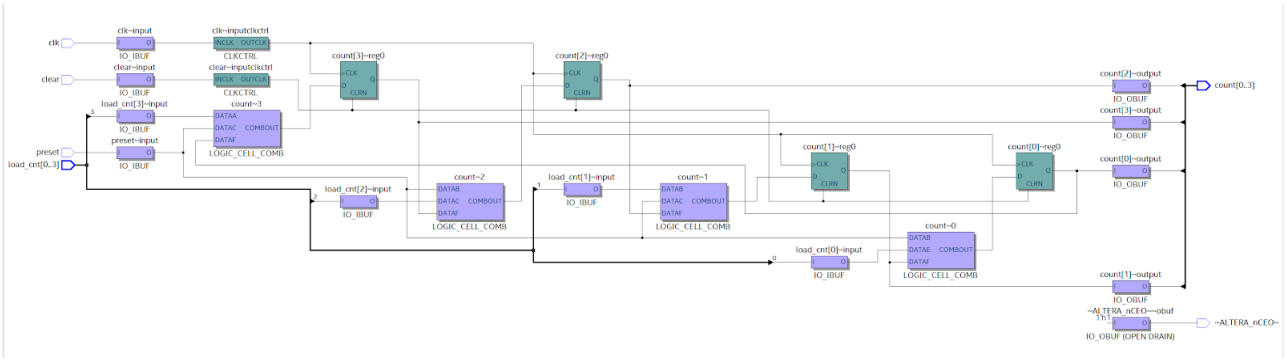


4. Post mapping schematic



There are 4 ALUT since there are 4 shift registers, each ALUT has 3 inputs: preset, load_cnt[0:3], and shift register output Q. The output of ALUT count~0, COMBOUT, is the input of shift register count[0], but the output of this register feeds into the input DATAA of ALUT count~3. This feedback behavior creates an 8 bit pattern (2N) that repeats every 8 clock cycles.

5. Post fitting schematic

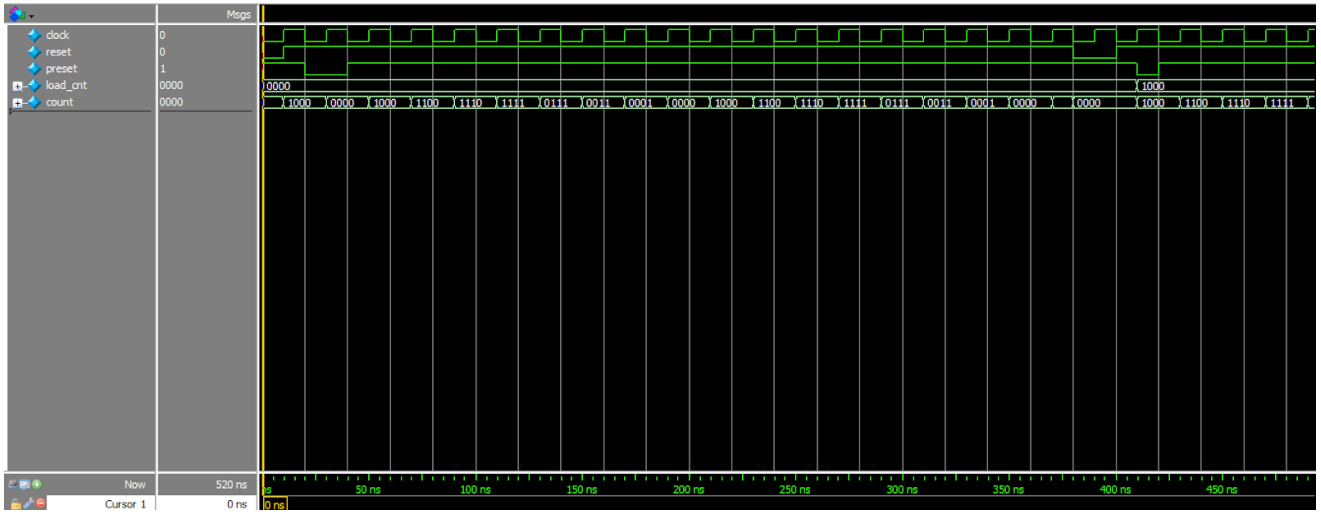


6. Resource Usage table

	Resource	Usage
1	Estimated ALUTs Used	4
1	-- Combinational ALUTs	4
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	4
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	4
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	0
4	-- 4 input functions	0
5	-- <=3 input functions	4
8		
9	Combinational ALUTs by mode	
1	-- normal mode	4
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	4
12		
13	Total registers	4
1	-- Dedicated logic registers	4
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	11
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	preset~input
21	Maximum fan-out	4
22	Total fan-out	39
23	Average fan-out	1.30

Number of ALUT: 4 (11 I/O pins)
Number of functions: 4 (3 input functions)

7. Simulation



The johnson_counter repeats a pattern every 8 clock cycles, the pattern goes 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000; then it repeats. The counter passes the 4 logic blocks 0, then to 4 logic block 1, shifting left each time. This in turn creates a pattern the repeats every 8 clock cycles since each passed block (0 and 1) takes 4 cycles.

```
Transcript
#
# Top level modules:
#       johnson_counter_testbench
# End time: 22:56:38 on Feb 01,2021, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
ModelSim> vsim work.johnson_counter_testbench
# vsim work.johnson_counter_testbench
# Start time: 22:56:43 on Feb 01,2021
# Loading sv_std.std
# Loading work.johnson_counter_testbench
# Loading work.johnson_counter
add wave -position insertpoint sim:/johnson_counter_testbench/*
VSIM 6> run -all
#  time=0,   clear=0  clk=0  count= 0
#  time=10,  clear=1  clk=1  count= 8
#  time=20,  clear=1  clk=0  count= 8
#  time=30,  clear=1  clk=1  count= 0
#  time=40,  clear=1  clk=0  count= 0
#  time=50,  clear=1  clk=1  count= 8
#  time=60,  clear=1  clk=0  count= 8
#  time=70,  clear=1  clk=1  count=12
#  time=80,  clear=1  clk=0  count=12
#  time=90,  clear=1  clk=1  count=14
#  time=100,  clear=1  clk=0  count=14
#  time=110,  clear=1  clk=1  count=15
#  time=120,  clear=1  clk=0  count=15
#  time=130,  clear=1  clk=1  count= 7
#  time=140,  clear=1  clk=0  count= 7
#  time=150,  clear=1  clk=1  count= 3
#  time=160,  clear=1  clk=0  count= 3
#  time=170,  clear=1  clk=1  count= 1
#  time=180,  clear=1  clk=0  count= 1
#  time=190,  clear=1  clk=1  count= 0
#  time=200,  clear=1  clk=0  count= 0
#  time=210,  clear=1  clk=1  count= 8
#  time=220,  clear=1  clk=0  count= 8
```