Lab 5

Ly Nguyen

March 1, 2022

An Introduction to Finite State Machines in VHDL using the Johnson Counter

Introduction

In this lab, we implemented a Johnson Counter, which is used to create Finite State Machines, in VHDL. This is another version of a counter that instead of counting up from 0 and incrementing by 1, it starts with the most significant bit and changes all the 0's in the vector to 1's, and then back to 0's after the whole vector is populated with 1's. With this, we learned to implement states and user created data types in VHDL.

Equipment and Method

Programs:

Quartus Prime Version 20.1.1 Build 720 11/11/2020 SJ Elite Edition

ModelSim SE-64 2021.2

Procedure:

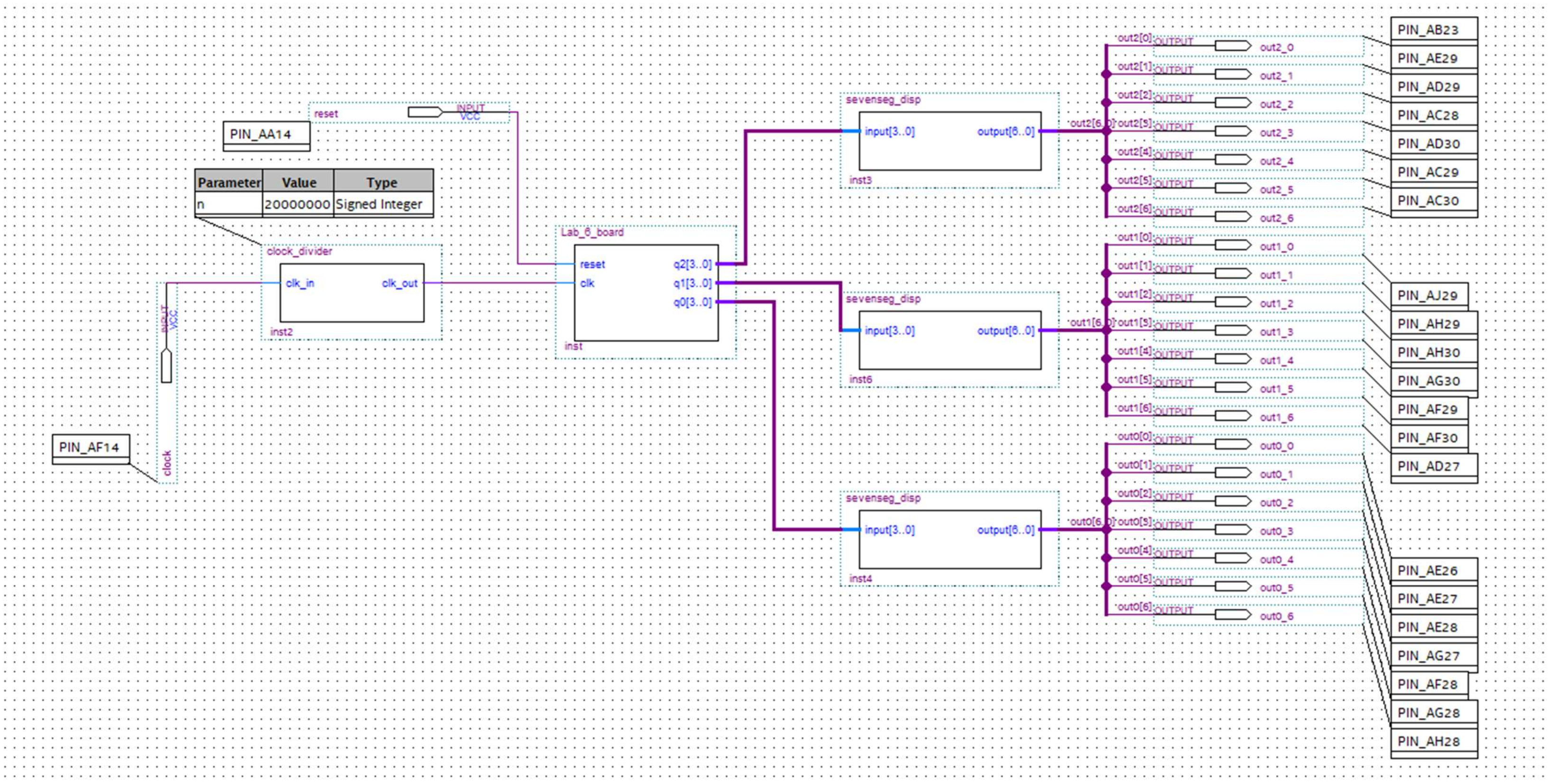
With the FSM Template (Appendix A) given in the lab handout, it has inputs clock and reset of type standard logic, an input to be filled in, and an output to be filled in. The input should be removed in this case since we just want our Johnson Counter to be cycling continuously, but the outputs are going to be the bits in which we are displaying, so q2, q1, and q0 all of type standard logic vector with 4 bits since the displays take an input of 4-bit vectors.

Following in the architecture portion of the code, we can see that there are user defined states, these should be populated with 6 states since we will be displaying a 3-bit Johnson Counter. The states can take on any name, so I used s0, s1, …, s5. After that there are some signals defined for present state and next state of type state (previously defined).

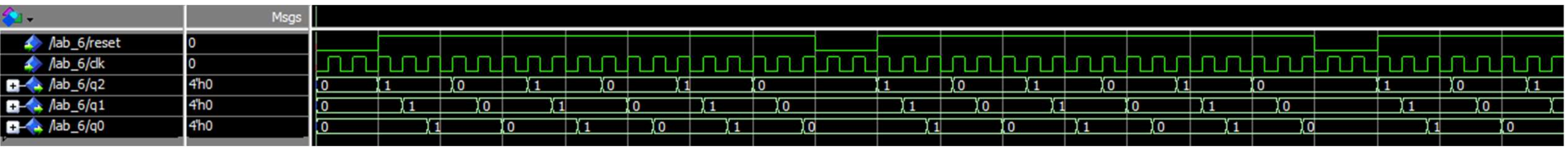
The FSM is then constructed after the begin command with he lower section of the FSM consisting of the clock and the reset. It reads that whenever reset is 1, so when the reset button is pressed, then the present state becomes the first state in the cycle, otherwise the present state cycles to the next state in the cycle on the rising clock edge. Next, in the upper section of the FSM, the process to assign the state cycle begins, where the first state outputs the first value, in this case it would be "000" where q2, q1, and q0 would all be equal to 0, and assigns the next state as the second state in the cycle. This is repeated for every state, so 6 times for the 3-bit Johnson Counter.

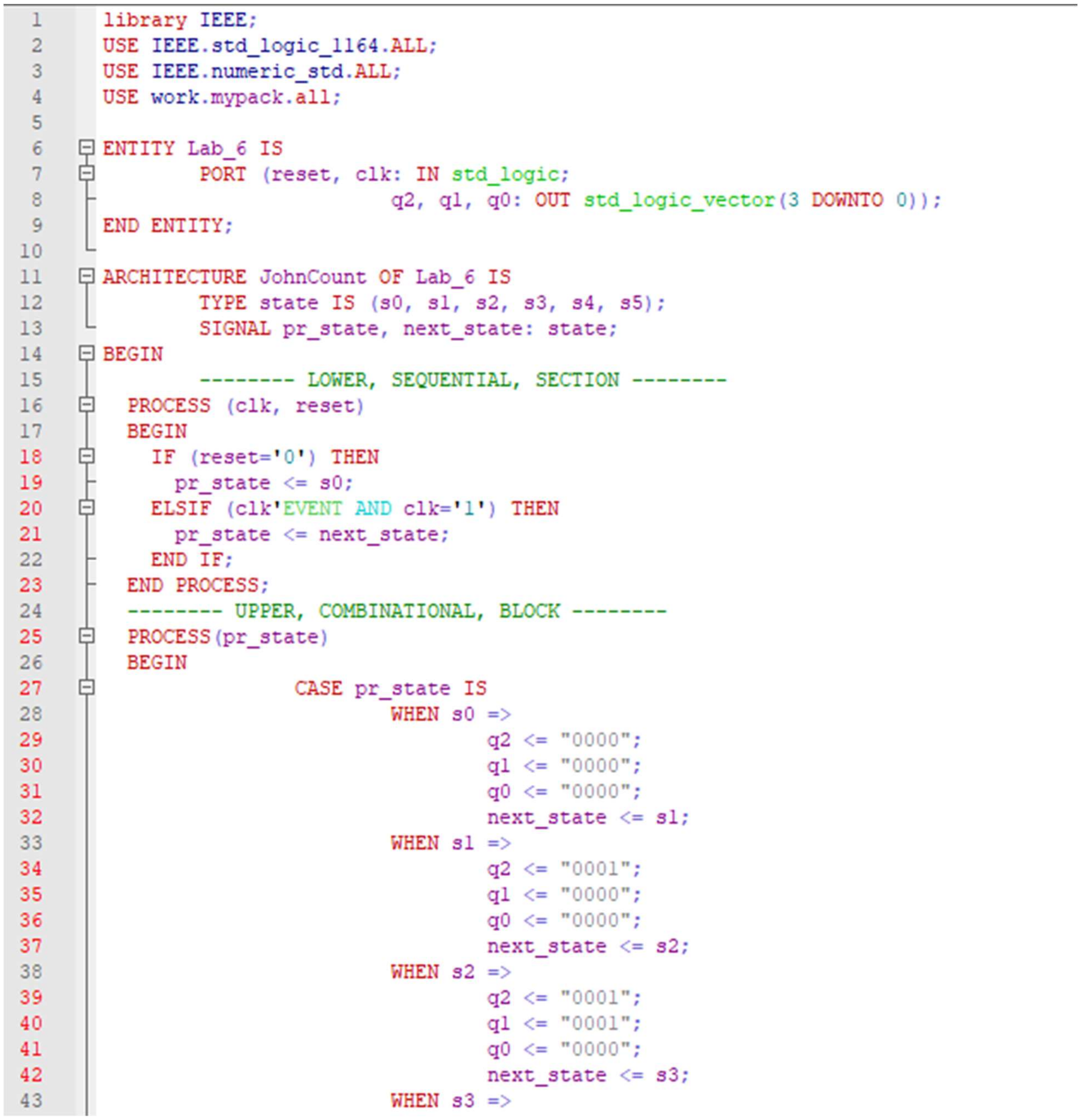
Block diagram:

Johnson counter takes an input of clock, which is using an internal clock through a clock divider, and a reset, which is a button in put on the board. The outputs are three 3-bit vector which feed into three seven segment displays, each corresponding to one bit in the Johnson Counter.



Results

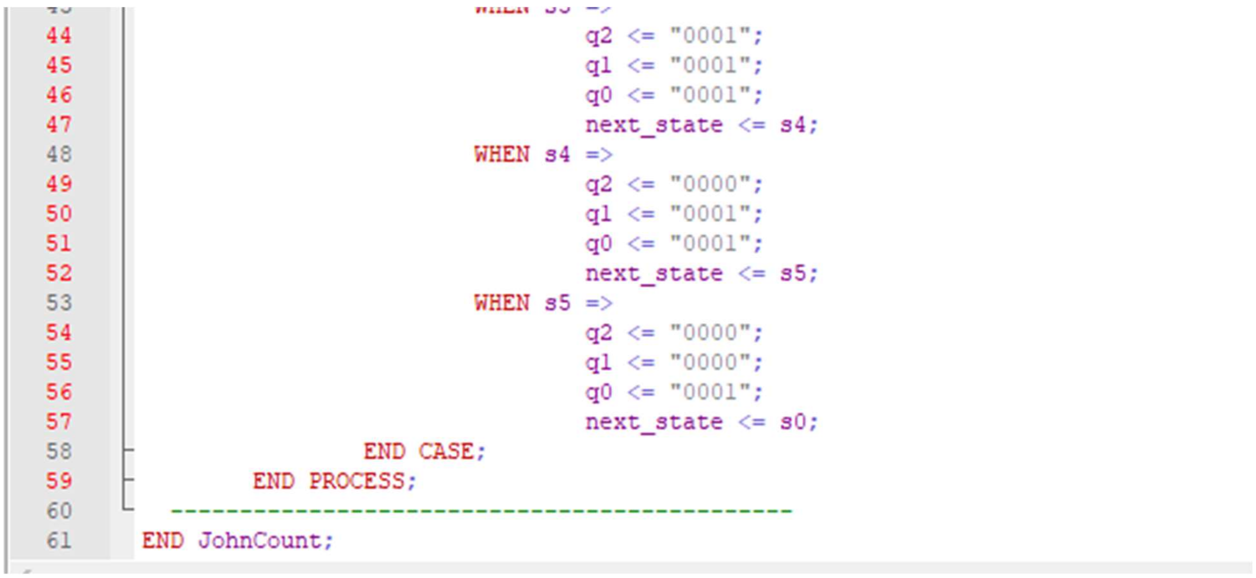




```vhdl
library IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.mypack.all;

ENTITY Lab_6 IS
        PORT (reset, clk: IN std_logic;
                        q2, q1, q0: OUT std_logic_vector(3 DOWNTO 0));
END ENTITY;

ARCHITECTURE JohnCount OF Lab_6 IS
        TYPE state IS (s0, s1, s2, s3, s4, s5);
        SIGNAL pr_state, next_state: state;
BEGIN
        -------- LOWER, SEQUENTIAL, SECTION --------
  PROCESS (clk, reset)
  BEGIN
    IF (reset='0') THEN
      pr_state <= s0;
    ELSIF (clk'EVENT AND clk='1') THEN
      pr_state <= next_state;
    END IF;
  END PROCESS;
  -------- UPPER, COMBINATIONAL, BLOCK --------
  PROCESS(pr_state)
  BEGIN
                CASE pr_state IS
                        WHEN s0 =>
                                q2 <= "0000";
                                q1 <= "0000";
                                q0 <= "0000";
                                next_state <= s1;
                        WHEN s1 =>
                                q2 <= "0001";
                                q1 <= "0000";
                                q0 <= "0000";
                                next_state <= s2;
                        WHEN s2 =>
                                q2 <= "0001";
                                q1 <= "0001";
                                q0 <= "0000";
                                next_state <= s3;
                        WHEN s3 =>
```



```
                                q2 <= "0001";
                                q1 <= "0001";
                                q0 <= "0001";
                                next_state <= s4;
                        WHEN s4 =>
                                q2 <= "0000";
                                q1 <= "0001";
                                q0 <= "0001";
                                next_state <= s5;
                        WHEN s5 =>
                                q2 <= "0000";
                                q1 <= "0000";
                                q0 <= "0001";
                                next_state <= s0;
                END CASE;
        END PROCESS;
  ----------------------------------------------
END JohnCount;
```

Video of Johnson counter in the zip file.

References

Lecture slides and lab handout

Discussion and Conclusion

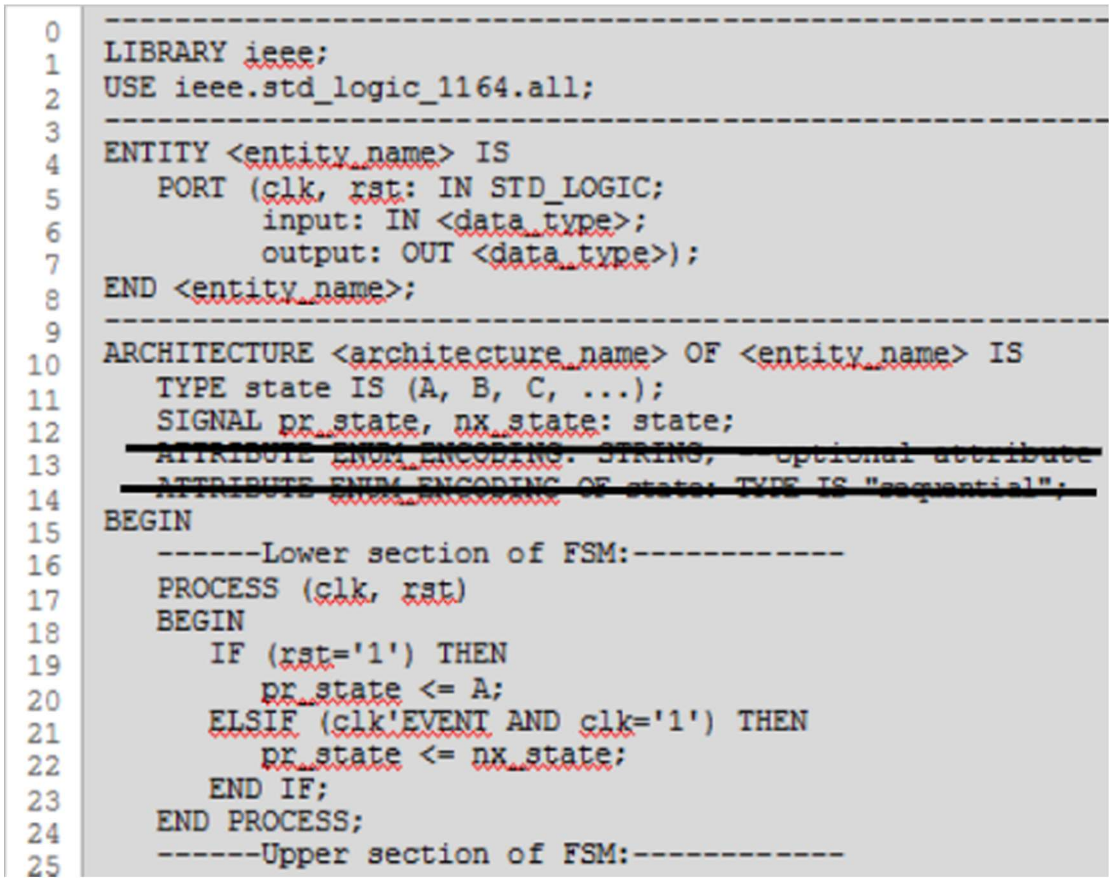
The only real debugging was trying to figure out how the counter should be displayed since it was a bit arbitrary. With the three vectors, I originally approached it with q2, q1, and q0 all having the same value, and just moving through the counter with actual bit values, so 000 → 0, 100 →4, 110→6, 111→7, etc. This caused some confusion but after discussing with the TA Cameron, we worked it out to just be q2, q1, and q0 alternating their displays from 0 and 1.

References

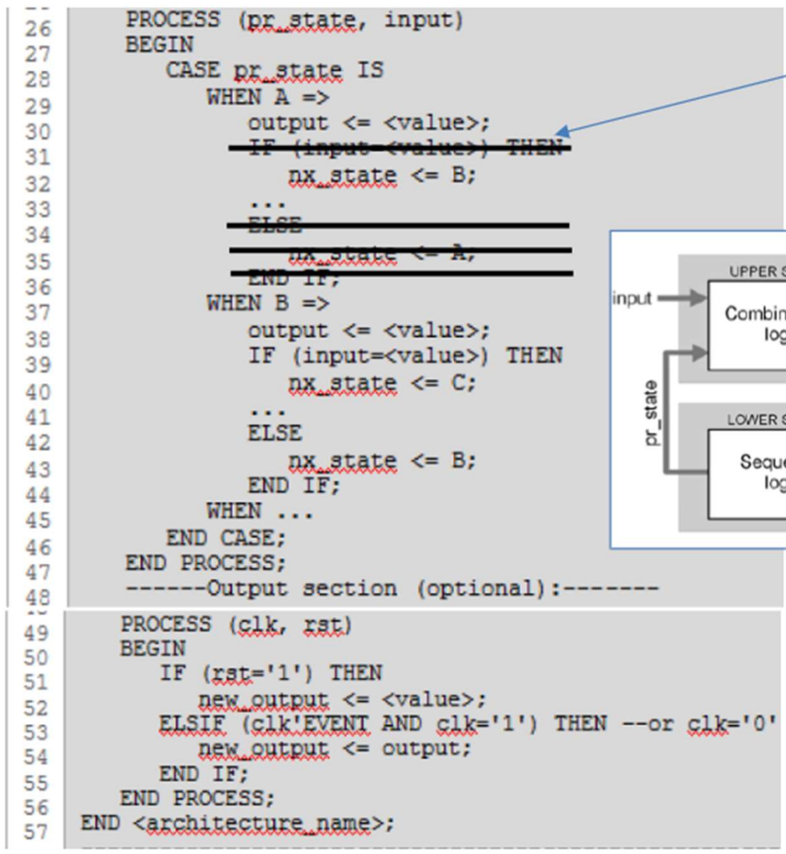
Lecture slides and lab manual

Appendices

Appendix A: FSM Template

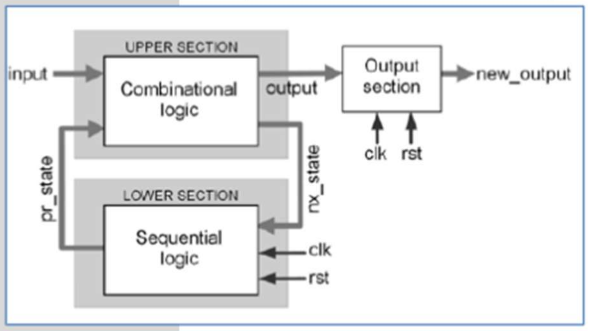


```
-----------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------------------------
ENTITY <entity_name> IS
   PORT (clk, rst: IN STD_LOGIC;
         input: IN <data_type>;
         output: OUT <data_type>);
END <entity_name>;
-----------------------------------------------------------
ARCHITECTURE <architecture_name> OF <entity_name> IS
   TYPE state IS (A, B, C, ...);
   SIGNAL pr_state, nx_state: state;
   ~~ATTRIBUTE ENUM_ENCODING: STRING; --optional attribute~~
   ~~ATTRIBUTE ENUM_ENCODING OF state: TYPE IS "sequential";~~
BEGIN
   ------Lower section of FSM:------------
   PROCESS (clk, rst)
   BEGIN
      IF (rst='1') THEN
         pr_state <= A;
      ELSIF (clk'EVENT AND clk='1') THEN
         pr_state <= nx_state;
      END IF;
   END PROCESS;
   ------Upper section of FSM:------------
```



```
   PROCESS (pr_state, input)
   BEGIN
      CASE pr_state IS
         WHEN A =>
            output <= <value>;
            ~~IF (input=<value>) THEN~~
               nx_state <= B;
            ...
            ~~ELSE~~
               ~~nx_state <= A;~~
            ~~END IF;~~
         WHEN B =>
            output <= <value>;
            IF (input=<value>) THEN
               nx_state <= C;
            ...
            ELSE
               nx_state <= B;
            END IF;
         WHEN ...
      END CASE;
   END PROCESS;
   ------Output section (optional):-------
   PROCESS (clk, rst)
   BEGIN
      IF (rst='1') THEN
         new_output <= <value>;
      ELSIF (clk'EVENT AND clk='1') THEN --or clk='0'
         new_output <= output;
      END IF;
   END PROCESS;
END <architecture_name>;
```

There is no **input** to a Johnson counter. Therefore you won't need the 'if' statement. However, you still need to update the state: (nxState <= …)



UPPER SECTION
input
Combinational logic
output
Output section
new_output
clk
rst
pr_state
nx_state
LOWER SECTION
Sequential logic
clk
rst