Tree 16-to-4 priority encoder in VHDL

1st Yuxiao Hua
School of System Design and Intelligent Manufacturing
Southern University of Science and Technology
Shenzhen, China
1628280289@qq.com

2nd Dinghan Zhang
Department of Electronic and Electrical Engineering
Southern University of Science and Technology
Shenzhen, China
226743714@qq.com

***Abstract*—In this experiment, I constructed a cascading and a tree 16-to-4 priority encoder separately using Vivado, and proceeded simulations on each of them. Comparing the results, I came to the conclusion that, encoder of tree structure is more efficient than that of a cascading one, owing to shorter path of gates.**

***Index Terms*—digital system design, VHDL, Vivado, priority encoder**

I. Introduction

A priority encoder (see Figure 1) is a circuit that returns the codes for the highest-priority request. By using a conditional signal assignment or if statements to naturally describe this function, the shape of the specified priority network is a single cascading chain.

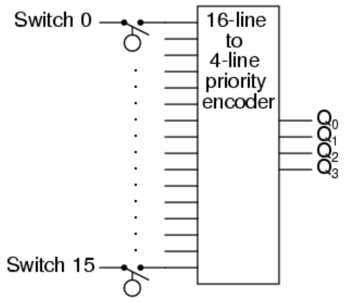


Fig. 1. Block diagram of a priority encoder

II. Pre-lab Preparation

Before constructing the tree encoder, I developed a VHDL code to realize a traditional 16-to-4 priority encoder using a conditional signal assignment (see Table 1). The outputs of the encoder are a 4-bit code and a 1-bit activity flag. Sketch the conceptual implementation of the design.

III. Experiment

In order to construct a tree priority encoder, firstly I shaped the layout of the circuit to a tree form so that to reduce the critical path of the designed 16-to-4 priority encoder. Secondly, I developed the block diagram of the design. Based on the block diagram, I wrote and tested the VHDL code of the design using component instantiation. Finally, the simulation of the design was proceeded. At last, I compared the layout and performance of the tree design with that of the cascading design.

Input S	Output Z, r
1 - - - - - - - - - - - - - - -	1111, 1
0 1 - - - - - - - - - - - - - -	1110, 1
0 0 1 - - - - - - - - - - - - -	1101, 1
0 0 0 1 - - - - - - - - - - - -	1100, 1
0 0 0 0 1 - - - - - - - - - - -	1011, 1
0 0 0 0 0 1 - - - - - - - - - -	1010, 1
0 0 0 0 0 0 1 - - - - - - - - -	1001, 1
0 0 0 0 0 0 0 1 - - - - - - - -	1000, 1
0 0 0 0 0 0 0 0 1 - - - - - - -	0111, 1
0 0 0 0 0 0 0 0 0 1 - - - - - -	0110, 1
0 0 0 0 0 0 0 0 0 0 1 - - - - -	0101, 1
0 0 0 0 0 0 0 0 0 0 0 1 - - - -	0100, 1
0 0 0 0 0 0 0 0 0 0 0 0 1 - - -	0011, 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 - -	0010, 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -	0001, 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0000, 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0000, 0

TABLE I
Truth table of a 16-to-4 priority encoder

IV. Results

A. Cascading priority encoder

1) Behavioral simulation: In order to figure out the full behavioral performance of the encoder, I applied a loop statement (see Appendix B), traversing from 0 to $2^{16} - 1$. The result met the expectation (see Figure 2).

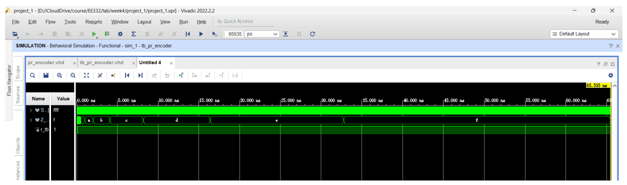


Fig. 2. Behavioral simulation of cascading encoder(overview)

Due to the high density at the beginning of the input series, I took a closer look at the waveform graph (see Figure 3). It can be seen from the chart that, the output r is 0 when S is *0000000000000000*.

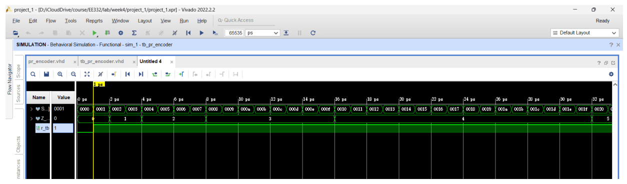


Fig. 3. Behavioral simulation of cascading encoder(detail)

Here is the schematic of the cascading encoder (see Figure 4). Due to the application of chain structure, the total time-delay is relatively large.

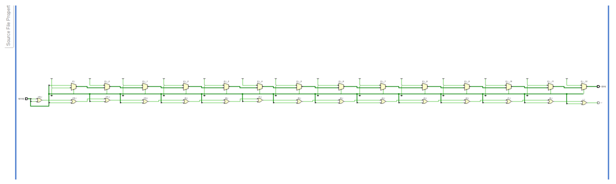


Fig. 4. Schematic of cascading encoder

2) Post-synthesis simulation: The result of post-synthesis functional simulation (see Figure 5) is the same as that of behavioral.

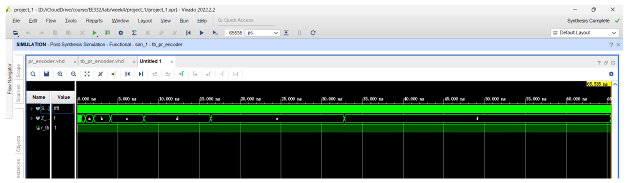


Fig. 5. Post-synthesis simulation of cascading encoder(functional)

However, considering the time-delay and the set-up time for sensitive list, the waveform of post-synthesis timing simulation (see Figure 6) at the beginning is unknown.

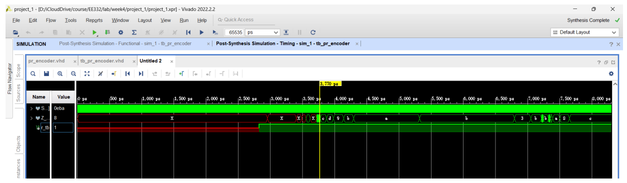


Fig. 6. Post-synthesis simulation of cascading encoder(timing)

3) Post-implementation simulation: The result of post-implementation simulation of functional (see Figure 7) is similar to that of post-synthesis simulation.

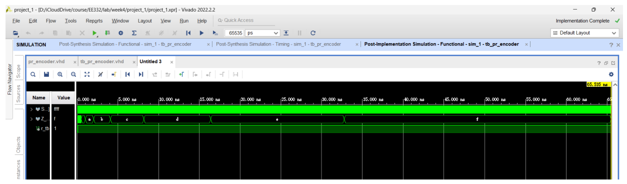


Fig. 7. Post-synthesis simulation of cascading encoder(functional)

So as he results of post-implementation simulation of timing (see Figure 8).

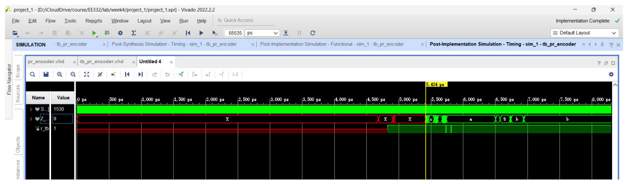
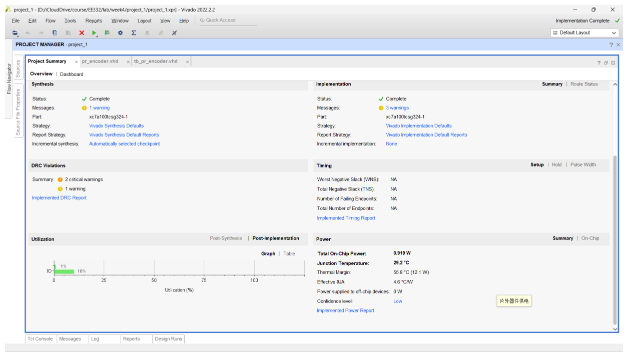


Fig. 8. Post-synthesis simulation of cascading encoder(timing)

The project summary (see Figure 9) is given as following. The efficiency of cascading structure is relatively low, hence I tried to design a tree structure one to enhance the performance.


project_1 - [D:/iCloudDrive/course/EE332/lab/week4/project_1/project_1.xpr] - Vivado 2022.2.2

Implementation Complete ✓

PROJECT MANAGER - project_1

Project Summary | pr_encoder.vhd | tb_pr_encoder.vhd

Overview | Dashboard

Synthesis

Status:	✓ Complete
Messages:	1 warning
Part:	xc7a100tcsg324-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	Automatically selected checkpoint

Implementation

Status:	✓ Complete
Messages:	3 warnings
Part:	xc7a100tcsg324-1
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

DRC Violations

Summary: 2 critical warnings
1 warning
Implemented DRC Report

Timing (Setup)

Worst Negative Slack (WNS):	NA
Total Negative Slack (TNS):	NA
Number of Failing Endpoints:	NA
Total Number of Endpoints:	NA

Implemented Timing Report

Utilization (Post-Implementation, Graph)

IO 1% 10%

Power (Summary)

Total On-Chip Power:	0.919 W
Junction Temperature:	29.2 °C
Thermal Margin:	55.8 °C (12.1 W)
Effective ϑJA:	4.6 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

片外器件供电

Fig. 9. Project summary of cascading encoder

B. *Tree priority encoder*

The purpose of the tree structure is to make the longest path shortest. It is essential to divide the problem into small problems, so as to deal with it in parallel and hierarchically (see Figure 10).

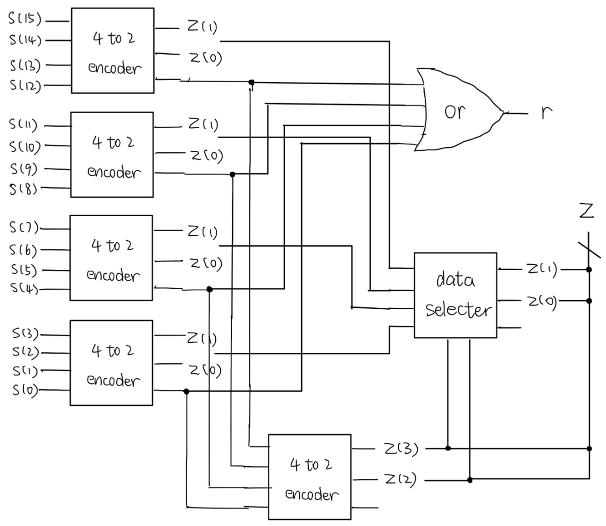


Fig. 10. Structure of tree encoder

The whole entity (see Appendix B) consists of four 16-to-4 priority encoders (see Appendix C), one 4-to-1 OR gate (see

Appendix D). The first two bit of Z is derived from the priority of r, and the last two bit come from the Z of the particular encoder.

1) Behavioral simulation: The behavioral simulation of tree priority encoder (see Figure 11) is similar to those of the cascading one.

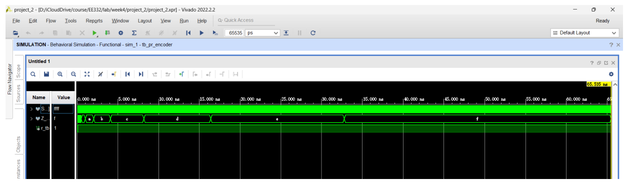


Fig. 11. Behavioral simulation of tree encoder

The schematic of the tree priority encoder (see Figure 12) is shown as following. It is the same as the one in the previous manual sketch. While the longest path is much shorter than the cascading one, the space complexity is relatively larger, as a result.

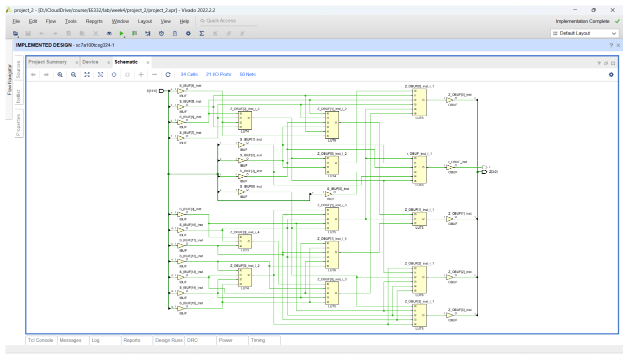


Fig. 12. Schematic of cascading encoder

2) Post-synthesis simulation: The post-synthesis simulation functional of tree priority encoder (see Figure 13) is similar to that of the cascading one.

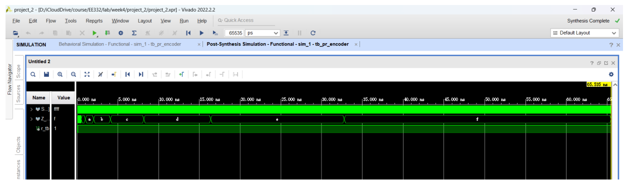


Fig. 13. Post-synthesis simulation of tree encoder(functional)

So as the post-synthesis simulation timing (see Figure 14).

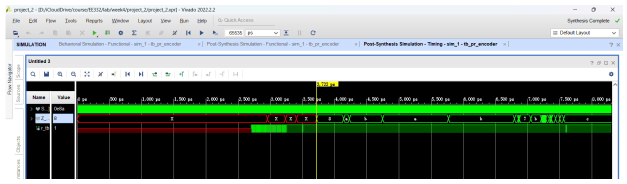


Fig. 14. Post-synthesis simulation of tree encoder(timing)

3) Post-implementation simulation: The post-implementation simulation functional of tree priority encoder (see Figure 15) is similar to that of the cascading one.

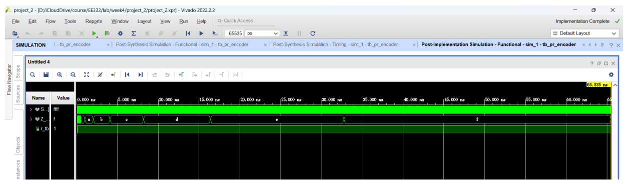


Fig. 15. Post-implementation simulation of tree encoder(functional)

However, due to the high time-delay of post-implementation simulation, the time to reach a normal state is much longer than the previous simulations (see Figure 16).

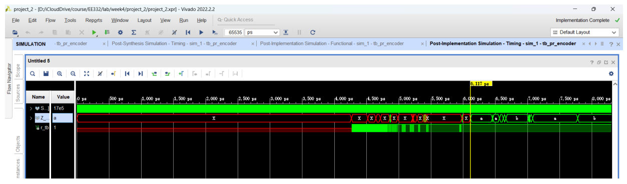


Fig. 16. Post-implementation simulation of tree encoder(timing)

V. Conclusion

In this experiment, I constructed a cascading and a tree 16-to-4 priority encoder, and proceeded simulations on them. Comparing the results, I came to the conclusion that, encoder of tree structure is more efficient than that of a cascading one, thanks to its shorter path of gates. In future, modeling design will be used more frequently, for its high efficiency and clear structure.

Appendix

A. *pr_encoder.vhd*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity pr_encoder is
    Port ( S : in STD_LOGIC_VECTOR (15 downto 0);
           Z : out STD_LOGIC_VECTOR (3 downto 0);
           r : out STD_LOGIC);
end pr_encoder;

architecture Behavioral of pr_encoder is
begin
    Z <= "1111" when S(15) = '1' else
         "1110" when S(14) = '1' else
         "1101" when S(13) = '1' else
         "1100" when S(12) = '1' else
         "1011" when S(11) = '1' else
         "1010" when S(10) = '1' else
         "1001" when S(9) = '1' else
         "1000" when S(8) = '1' else
         "0111" when S(7) = '1' else
         "0110" when S(6) = '1' else
         "0101" when S(5) = '1' else
         "0100" when S(4) = '1' else
         "0011" when S(3) = '1' else
         "0010" when S(2) = '1' else
         "0001" when S(1) = '1' else
         "0000" when S(0) = '1' else
         "0000";
    r <= S(15) or S(14) or S(13) or S(12) or
         S(11) or S(10) or S(9) or S(8) or
         S(7) or S(6) or S(5) or S(4) or
         S(3) or S(2) or S(1) or S(0);
end Behavioral;
```

B. *tb_pr_encoder.vhd*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;

entity tb_pr_encoder is
end tb_pr_encoder;

architecture Behavioral of tb_pr_encoder is
component pr_encoder is
    Port ( S : in STD_LOGIC_VECTOR (15 downto 0);
           Z : out STD_LOGIC_VECTOR (3 downto 0);
           r : out STD_LOGIC);
end component pr_encoder;

signal S_tb: STD_LOGIC_VECTOR (15 downto 0);
signal Z_tb: STD_LOGIC_VECTOR (3 downto 0);
signal r_tb: STD_LOGIC;
```

```vhdl
begin
    UUT: pr_encoder port map (S=>S_tb, Z=>Z_tb, r=>r_tb);
    process is
    begin
        this_loop: for i in 0 to 2**16-1 loop
            S_tb<=conv_std_logic_vector(i,16);
            wait for 1 ps;
        end loop this_loop;
    end process;
end Behavioral;
```

C. tree_pr_encoder.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith;

entity tree_pr_encoder is
    Port ( S : in STD_LOGIC_VECTOR (15 downto 0);
           Z : out STD_LOGIC_VECTOR (3 downto 0);
           r : out STD_LOGIC);
end tree_pr_encoder;

architecture structure of tree_pr_encoder is
    component encoder is
        port(S_ec:in STD_LOGIC_VECTOR (3 downto 0);
             Z_ec:out STD_LOGIC_VECTOR (1 downto 0);
             r_ec:out STD_LOGIC);
    end component encoder;

    component or_part is
        port(In_or:in STD_LOGIC_VECTOR (3 downto 0);
             Out_or:out STD_LOGIC);
    end component or_part;

    signal z1,z2,z3,z4:STD_LOGIC_VECTOR (1 downto 0);
    signal r1,r2,r3,r4:STD_LOGIC;
    signal o_t:STD_LOGIC_VECTOR (3 downto 0);

begin
    ec1:encoder port map(S(3 downto 0),z1,r1);
    ec2:encoder port map(S(7 downto 4),z2,r2);
    ec3:encoder port map(S(11 downto 8),z3,r3);
    ec4:encoder port map(S(15 downto 12),z4,r4);

    Z<="11" & z4 when r4='1' else
       "10" & z3 when r3='1' else
       "01" & z2 when r2='1' else
       "00" & z1;

    o_t <= r1 & r2 & r3 & r4;
    or1:or_part port map(o_t,r);
end structure;
```

D. encoder_part.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity encoder is
    port(S_ec:in STD_LOGIC_VECTOR (3 downto 0);
         Z_ec:out STD_LOGIC_VECTOR (1 downto 0);
         r_ec:out STD_LOGIC);
end encoder;

architecture Behavioral of encoder is

begin
    Z_ec<= "11" when S_ec(3)='1' else
           "10" when S_ec(2)='1' else
           "01" when S_ec(1)='1' else
           "00" when S_ec(0)='1' else
           "00";
r_ec<= S_ec(3) or S_ec(2) or S_ec(1) or S_ec(0);

end Behavioral;
```

E. or_part.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity or_part is
    port(In_or:in STD_LOGIC_VECTOR (3 downto 0);
         Out_or:out STD_LOGIC);
end or_part;

architecture Behavioral of or_part is
begin
    Out_or<=(In_or(3) or In_or(2)) or (In_or(1) or In_or(0));

end Behavioral;
```