Tree-structure 16-to-4 Priority Encoder Design and Performance Comparison

Wang Zhuoyang[1]

[1] *12112907, Department of Electrical and Electronic Engineering, SUSTech. Email: glverfer@outlook.com*

Abstract

This is the lab report for the Digital System Designing course, which offers an overview of the principle and the implementation of the 16-to-4 priority encoder.

During the pre-lab exercises, a conditional signal assignment is used to construct a 16-to-4 priority encoder. The corresponding RTL circuit is composed of fifteen 2-to-1 multiplexers cascaded together, which theoretically results in significant delay. Therefore, we modified the design by using five 4-to-2 priority encoders to construct a tree-structured 16-to-4 priority encoder, artificially reducing the longest signal path length to decrease delay.

Also, we conducted simulation tests on both designs and analyzed the results.

Keywords: Priority Encoder; VHDL

Contents

Digital System Designing Laboratory Reports (2023)

Corresponding author
Wang Zhuoyang

1 Introduction

2 Pre-lab Exercises

Pre-lab exercise: “Refer to the 4-to-2 priority encoder on Page 81, develop a VHDL code to realize a 16-to-4 priority encoder using a conditional signal assignment. The outputs of the encoder are a 4-bit code and a 1-bit activity flag. Sketch the conceptual implementation of the design.”

In order to ensure the continuity of the report, the results of the pre-lab can be found in the following section.

3 Design and Implementation

We begin by defining the interface description for the 16-to-4 priority encoder that we aim to implement (Code 1).

```
entity priority_encoder_16_4 is
    port (
        S: in std_logic_vector(15 downto 0);
        Z: out std_logic_vector(3 downto 0);
        r: out std_logic
    );
end priority_encoder_16_4;
```

Listing 1. The port declaration for the 16-to-4 priority encoder.

S is a sixteen-input signal, Z is a four-output encoding result, and r indicates the validity of the result, which is low only when all inputs are at low level. Its truth table is similar to that of the 4-to-2 priority encoder in the Introduction section, which means the highest bit is used as a reference when multiple inputs are valid.

3.1 Old Scheme

Conceptual Diagram

Following the most primitive idea, we implement the priority encoder using conditional signal assignment, and its corresponding Conceptual Diagram is shown in the Figure 10.

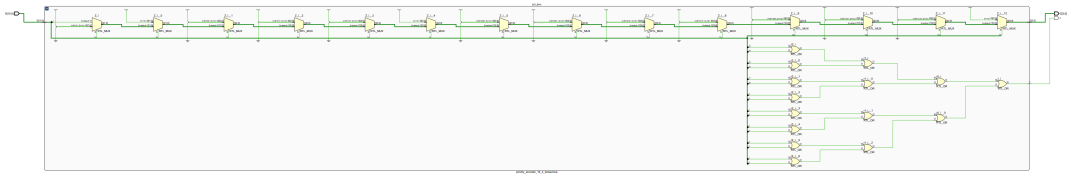


Figure 1. The RTL conceptual diagram of the behavioral architecture.

As can be expected, in order to achieve “priority”, it is necessary to use fourteen 2-to-1 multiplexers cascaded together, with higher priority signals placed towards the end to ensure that the higher bit result is selected when the higher bits are valid.

For ease of distinction, we refer to this implementation as the “Behavioral” architecture, which only describes its behavioral characteristics.

Theoretical Timing Analysis

Observing the above diagram, we can see that the depth of the RTL circuit corresponding to the Behavioral architecture is relatively large, with the longest signal transmission path passing through fourteen multiplexers.

Assuming the delay of each multiplexer is T, the longest delay of this design is $14T$.

In fact, according to the most basic idea, we should use 15 2-to-1 multiplexers, but the RTL circuit generated by Vivado only has fourteen. The reason for this is that when the input is “...01”

and "...00", the output is exactly "0001" and "0000", respectively. The circuit directly utilizes this feature to merge the last two cases, i.e., the last two bits of the input are equal to the last two bits of the output, thus saving one multiplexer.

Code Implementation

Below is its VHDL implementation (Code ???2). For the encoding result Z, we directly use conditional signal assignment to complete it. For the validity indication signal r, we add parentheses manually to minimize its delay, which is also reflected in the above diagram (Figure ???10).

```
architecture Behavioral of priority_encoder_16_4 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";
    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;
```

Listing 2. The 16-to-4 priority encoder using conditional signal assignment.

3.2 Optimized Scheme

Conceptual Diagram

To reduce the delay and shorten the longest signal path length, we can use five 4-to-2 priority encoders to implement the 16-to-4 priority encoder.

The core idea is to separate the calculation of the first two and last two bits of the result. We divide the input of sixteen bits into four groups of four and connect each group to a 4-to-2 priority encoder. At this point, the output r of the four encoders indicate the approximate location of the highest significant bit. Connecting the four r outputs to another 4-to-2 priority encoder yields the high two bits of the result and the overall output r.

Simultaneously, the outputs r of the four encoders determine which one of the four encoder outputs should be used for the low two bits of the result. We can use a four-way multiplexer to implement this.

The conceptual diagram is shown in Figure 2. The "RTL_LATCH" in the figure can be ignored, and it is not actually connected. Its existence is due to the Vivado compilation mechanism, which is a template.

Theoretical Timing Analysis

Firstly, the 4-to-2 priority encoder is implemented using the conventional method, which is realized by two 2-to-1 multiplexers (the reason for not using three is explained earlier). After the signal

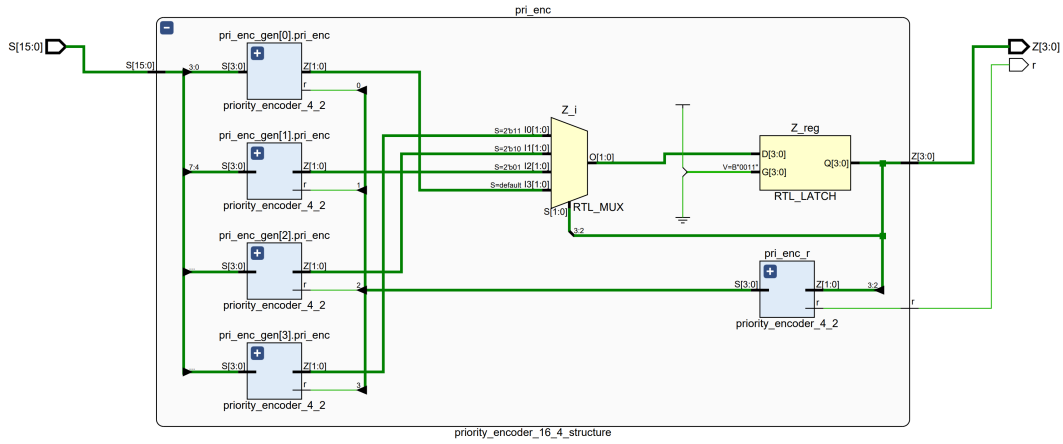


Figure 2. The RTL conceptual diagram of the structural architecture.

enters the encoder, the result is connected to a 4-to-1 multiplexer, which is the final output, while the validity output is connected to another 4-to-2 priority encoder and is also outputted.

Assuming the delay of each 2-to-1 multiplexer is T_2 and the delay of the 4-to-1 multiplexer is T_4, and ignoring the delay of the wires, we can calculate that the longest delay of the signal line in this design is:

$$T_{Z(min)} = \max\{2 \times T_2 + T_4, 4 \times T_2\} \tag{1}$$

And if the delay of the OR gate is T_{OR}, then the minimum delay of the output r is:

$$T_{r(min)} = 2 \times 3 \times T_{OR} = 6T_{OR} \tag{2}$$

Code Implementation

First, we implement the 4-to-2 priority encoder, which can be achieved using conditional signal assignment (Code 3).

```
architecture Behavioral of priority_encoder_4_2 is
begin
    Z <= "11" when S(3) = '1' else
         "10" when S(2) = '1' else
         "01" when S(1) = '1' else
         "00";
    r <= (S(3) or S(2)) or (S(1) or S(0));
end Behavioral;
```

Listing 3. The 4-to-2 priority encoder.

Then construct the 16-to-4 priority encoder according to the structural diagram above (Code 4).

```
architecture Structural of priority_encoder_16_4 is
    component priority_encoder_4_2 is
        port (
            S: in std_logic_vector(3 downto 0);
            Z: out std_logic_vector(1 downto 0);
            r: out std_logic
        );
    end component priority_encoder_4_2;

    signal r_inter: std_logic_vector(3 downto 0);

    type arrtype is array (3 downto 0) of std_logic_vector(1 downto 0);
```

```
    signal z_inter: arrtype;
    signal z_high2: std_logic_vector(1 downto 0);
begin
    pri_enc_gen: for i in 0 to 3 generate
        pri_enc: priority_encoder_4_2
            port map (
                S => S(4 * i + 3 downto 4 * i),
                Z => z_inter(i),
                r => r_inter(i)
            );
    end generate;

    pri_enc_r: priority_encoder_4_2
        port map (
            S => r_inter,
            Z => z_high2,
            r => r
        );

    Z(3 downto 2) <= z_high2;
    with z_high2 select
        Z(1 downto 0) <= z_inter(3) when "11",
                         z_inter(2) when "10",
                         z_inter(1) when "01",
                         z_inter(0) when others;
end Structural;
```

Listing 4. The 16-to-4 priority encoder using module instances.

4 Simulation

4.1 Testbench

To test the delay, we need to write a testbench for simulation (Code 5). The input signals are set to be enabled from the low bit to the high bit in sequence, with an interval of 20 ns.

```
    ...
    for uut: top_module use entity work.top_module(Behavioral); -- or
    Structural
    ...
    uut: top_module
        port map (
            S => S,
            Z => Z,
            r => r
        );

    process is
    begin
        S <= (others => '0');
        for i in 0 to 15 loop
            wait for 20 ns;
            S(i) <= '1';
        end loop;
        wait;
    end process;
    ...
```

Listing 5. The testbench.

4.2 Results

We ran three simulations for each of the two architectures, and the results are shown below (Figure 3, 4, 5, 6, 7 and 8).

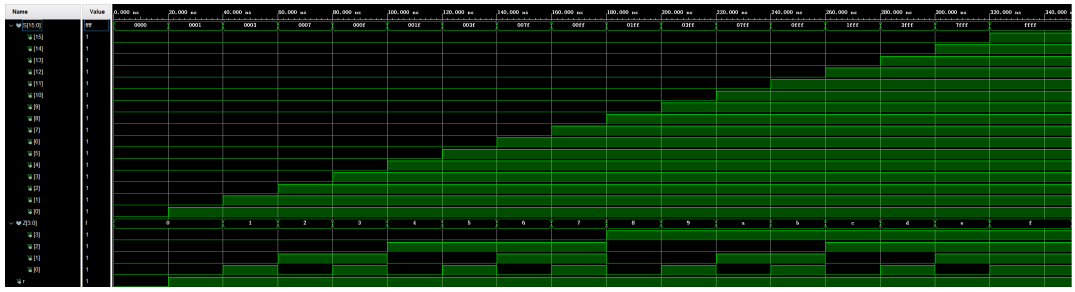


Figure 3. The result of Behavioral Simulation (Behavioral Architecture).

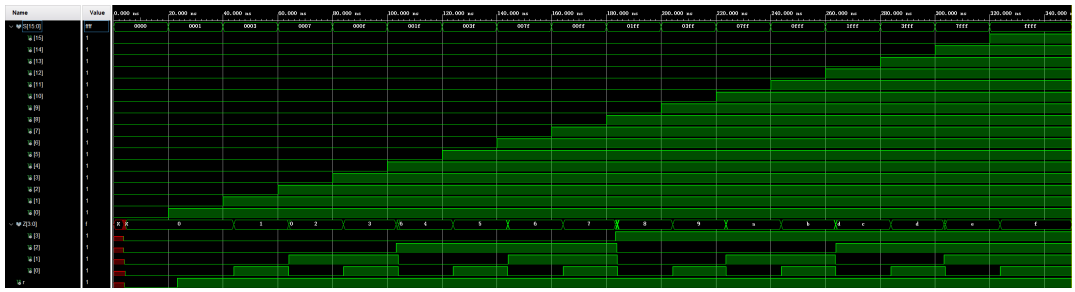


Figure 4. The result of Post-Synthesis Timing Simulation (Behavioral Architecture).

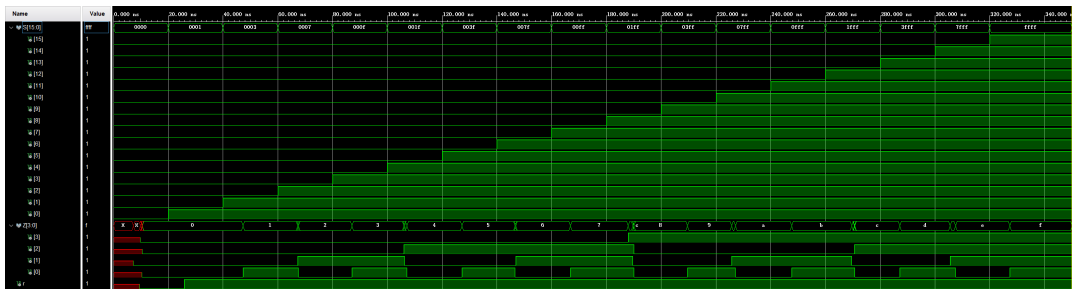


Figure 5. The result of Post-Implementation Timing Simulation (Behavioral Architecture).

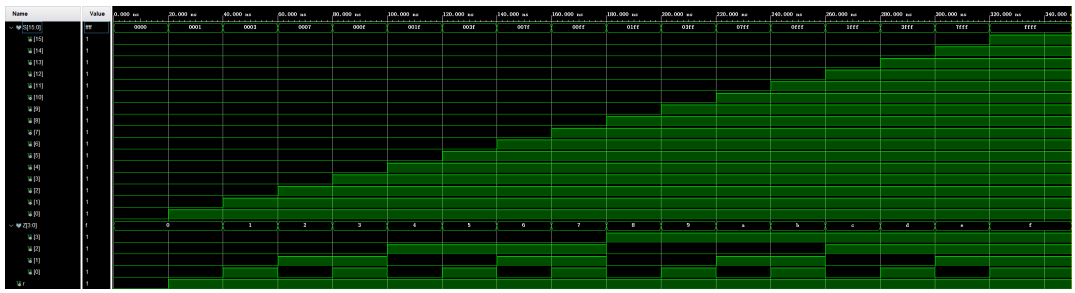


Figure 6. The result of Behavioral Simulation (Structural Architecture).

Firstly, the behavioral simulation and post-synthesis simulation were only performed to comply with the vague requirements of the report, and did not provide substantial help for analysis. The

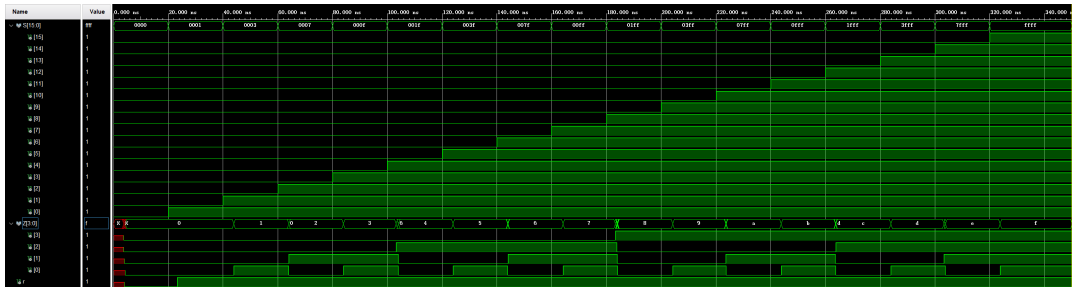


Figure 7. The result of Post-Synthesis Timing Simulation (Structural Architecture).

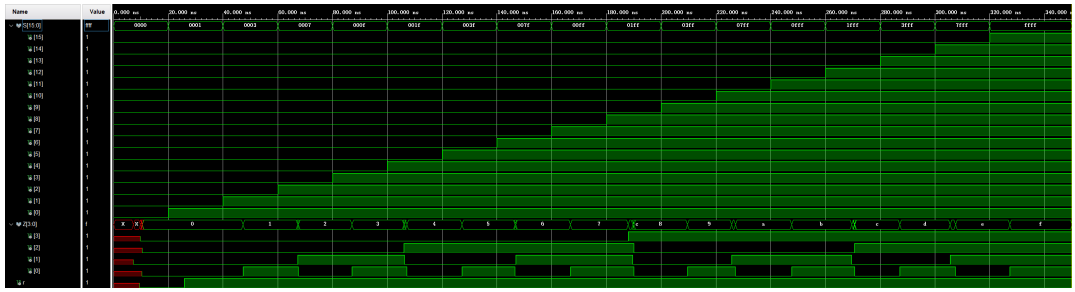
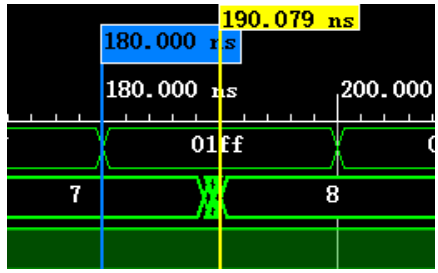


Figure 8. The result of Post-Implementation Timing Simulation (Structural Architecture).

differences between them and their detailed interpretation have been explained in the previous report.

We mainly analyze post-implementation timing simulation. We capture the node with the longest delay, as shown in the Figure 9.

(a) Behavioral



(b) Structural

Figure 9. The longest delay in post-implementation timing simulation.

It can be seen that our manually optimized implementation **runs slower than** the original implementation.

4.3 Reason Analysis

The reason for the above phenomenon is that the actual circuit implementation can use resources such as LUTs, which is completely different from the implementation of pure gate circuits. The original implementation allows more freedom for the compiler. For any combination logic circuit, the compiler can use a large number of LUTs to complete the implementation with only one layer. However, after we modified the code to make the logic more complex, it resulted in a negative optimization.

Specifically, we can examine the schematic of the two architectures after implementation (Figure 10 and 11).

We can clearly see from the implemented schematics of the two architectures that the function-

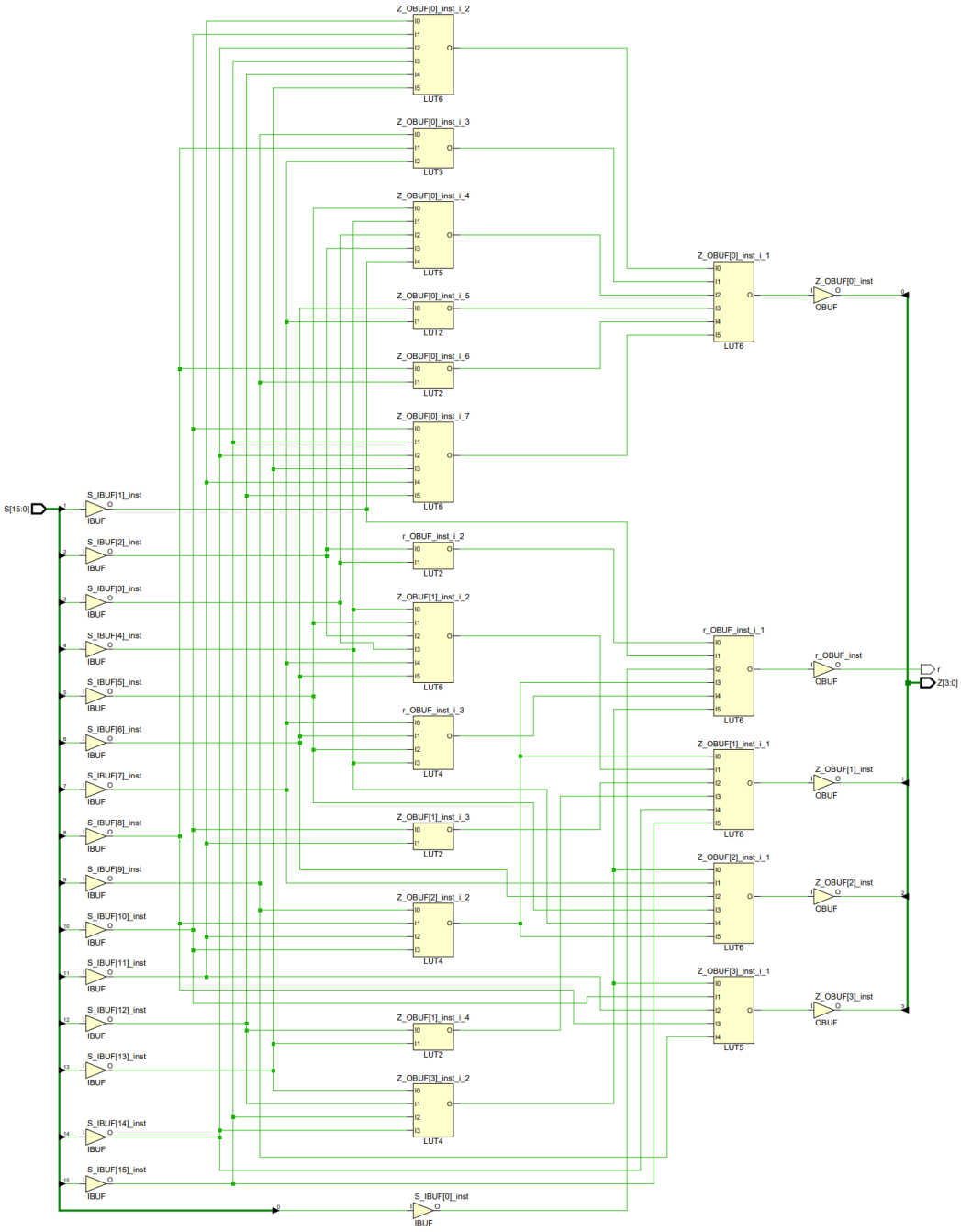


Figure 10. The implementation schematic of the Behavioral Architecture.

ality that could have been achieved using two levels of LUTs was implemented using three levels after our optimization, which resulted in an increased delay.

5 Extension

To verify the correctness, we conducted on-board testing (Omitted).

6 Conclusion

Overall, two important conclusions were demonstrated in this experiment:

1. Synthesizer is smarter than humans in most cases.

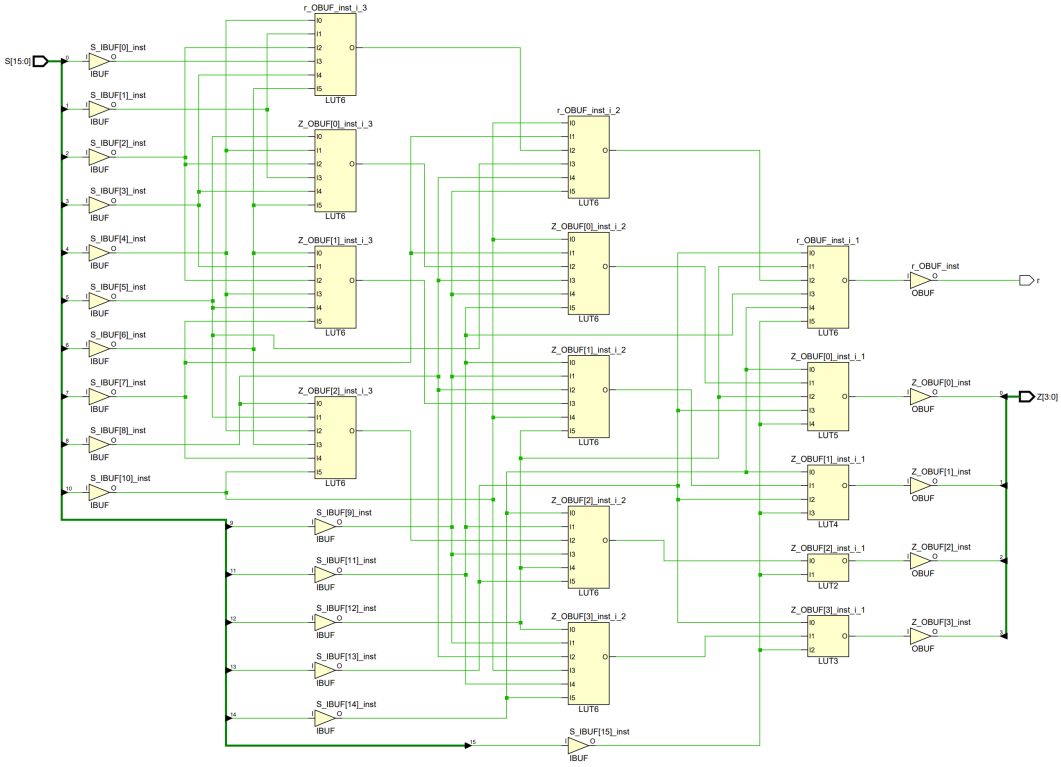


Figure 11. The implementation schematic of the Structural Architecture.

2. When designing combinational logic circuits on an FPGA, one should not adopt the mindset of gate-level circuits, but instead consider LUTs.

Of course, the construction process of the priority encoder also demonstrates that a high-bit priority encoder can be constructed from a low-bit priority encoder. Although there was no experiment in the report, it is also possible to use more 2-to-1 priority encoders to implement a 16-to-4 priority encoder. However, since the combinational logic circuit becomes more complex, as long as the scale is not too large, it can be directly implemented using LUTs. Therefore, this approach is almost meaningless, and it is speculated that the resulting delay may be even higher than the solution constructed by 4-to-2 encoders.