**Hardware Design and Lab: Lab5**

**111060013 EECS 26' 劉祐廷**

**Catalog**

**1. Basic Question:**

**Many-to-one LFSR and One-to-many LFSR………………P3**

**2. Advanced Question:**

**Content-addressable memory (CAM) design…...………….P4**

**3. Advanced Question:**

**Scan Chain Design…….……………………….…………..…...P6**

**4. Advanced Question:**

**Built-in Self Test……………………………………….……..P8**

**5. Advanced Question:**

**Built-in Self Test FPGA…………………………………....P10**

**6. Advanced Question:**

**Mealy Sequence Detector…………………………………P15**

**6. What I Have Learned…...................................................P16**

1. **Advanced Question: Sliding window sequence detector**
2. **Finite State Diagram**

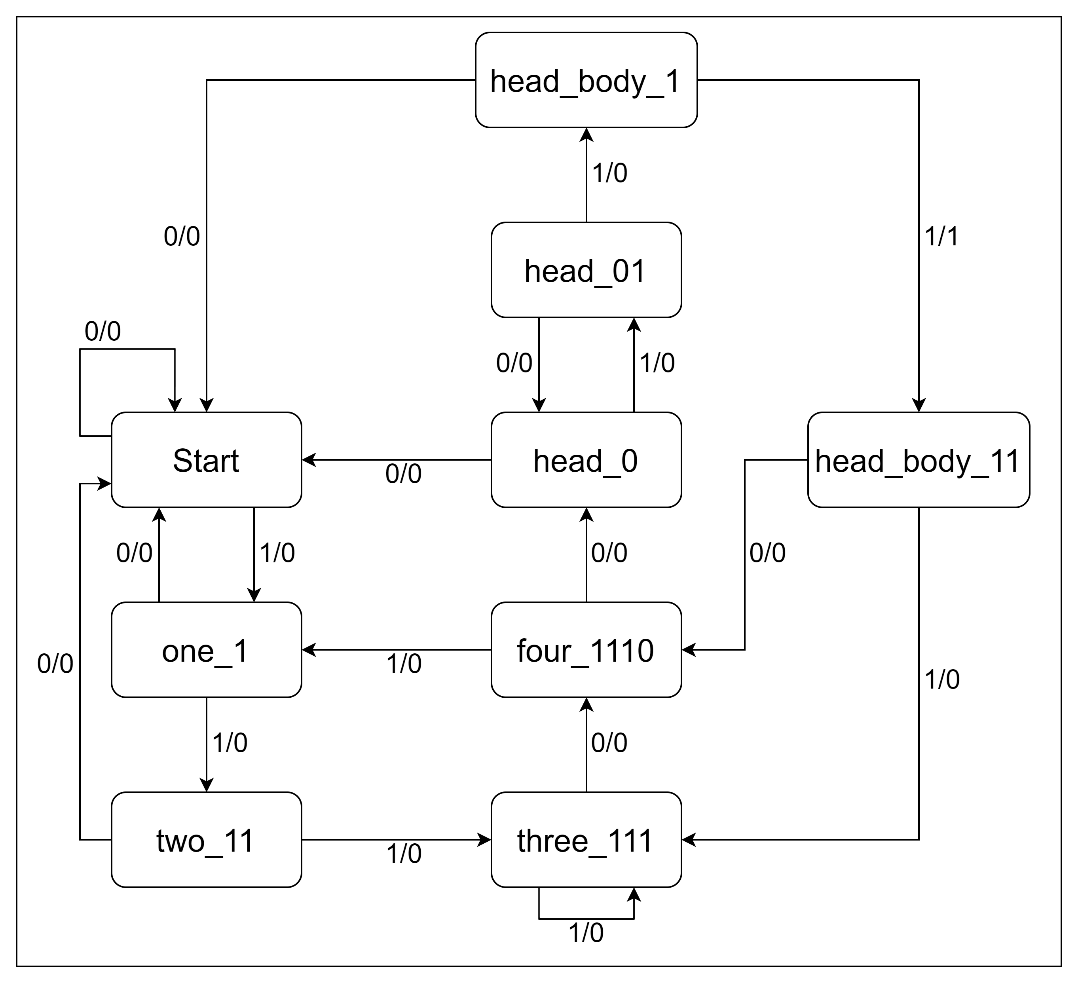
****

Figure 1.1

1. **Explanation**

**Start:** Detect 0 bit that match the pattern.

**one\_1:** Detect the first bit **1** of the pattern.

**two\_11:** Detect the second bit **1** of the pattern.

**three\_111:** Detect the third bit **1** of the pattern.

**four\_1110:** Detect the fourth bit **0** of the pattern.

**head:** It means 1110.

**head\_0:** Detect **0** of the "several **01**".

**head\_01:** Detect **1** of the "several **01**".

**body:** It means "several **01**".

**head\_body\_1:** Detect the second-to-last bit **1** of the pattern.

**head\_body\_11:** Detect the last bit **1** of the pattern

1. **Testbench**

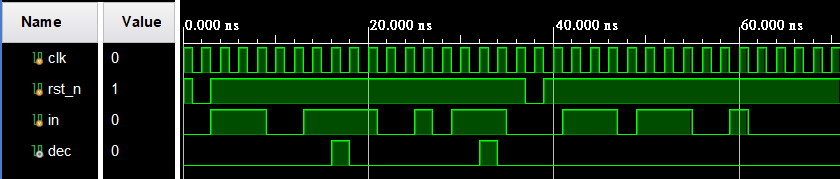


Figure 1.2

I use the testcase of the lab slide to check if there is something wrong and it seems that the design works properly.

1. **Advanced Question: Traffic light controller**
2. **Finite State Diagram**

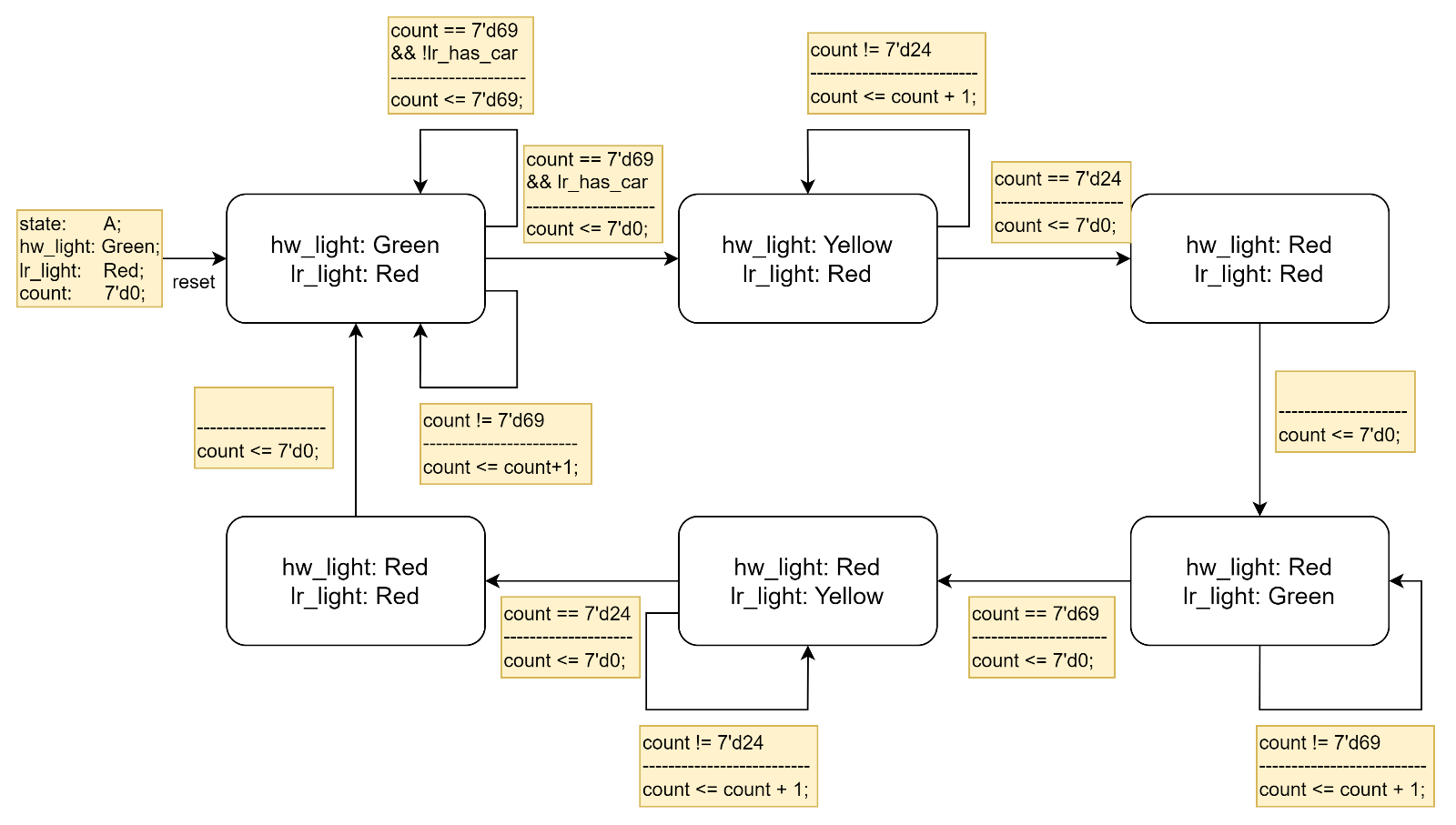
****

Figure 2.1

1. **Explanation**

I designed my circuit according to **Figure 2.1**. **count** remains **7'd69** is **lr\_has\_car == 1'b0** while the state is **hw\_light: Green, lr\_light: Red**, so that after counting 70 clock cycles, if **lr\_has\_car == 1'b1**, **hw\_light** can change into yellow immediately.

1. **Testbench**

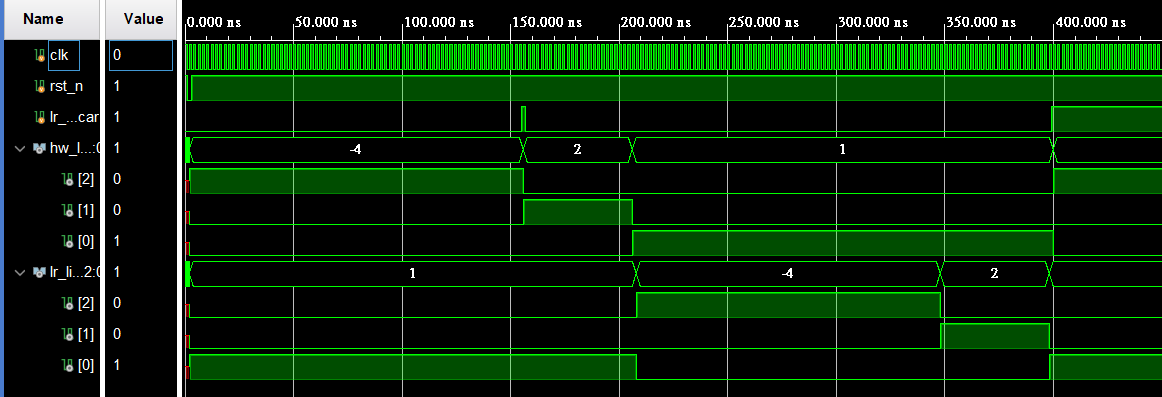
****

Figure 2.2

First, I test the case that **lr\_has\_car** is pulled up after 75 clock cycles, and it seems that nothing is wrong as **Figure 2.2** shows.

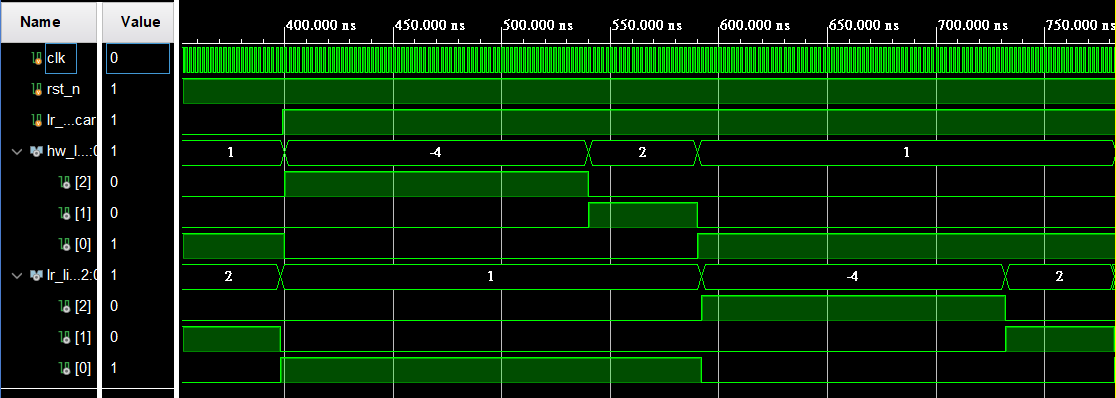


Figure 2.3

Second, I test the case that **lr\_has\_car** is pulled up from the beginning, and everything is correct as **Figure 2.3** shows.

1. **Advanced Question: Greatest common divisor**
2. **Block Diagram**

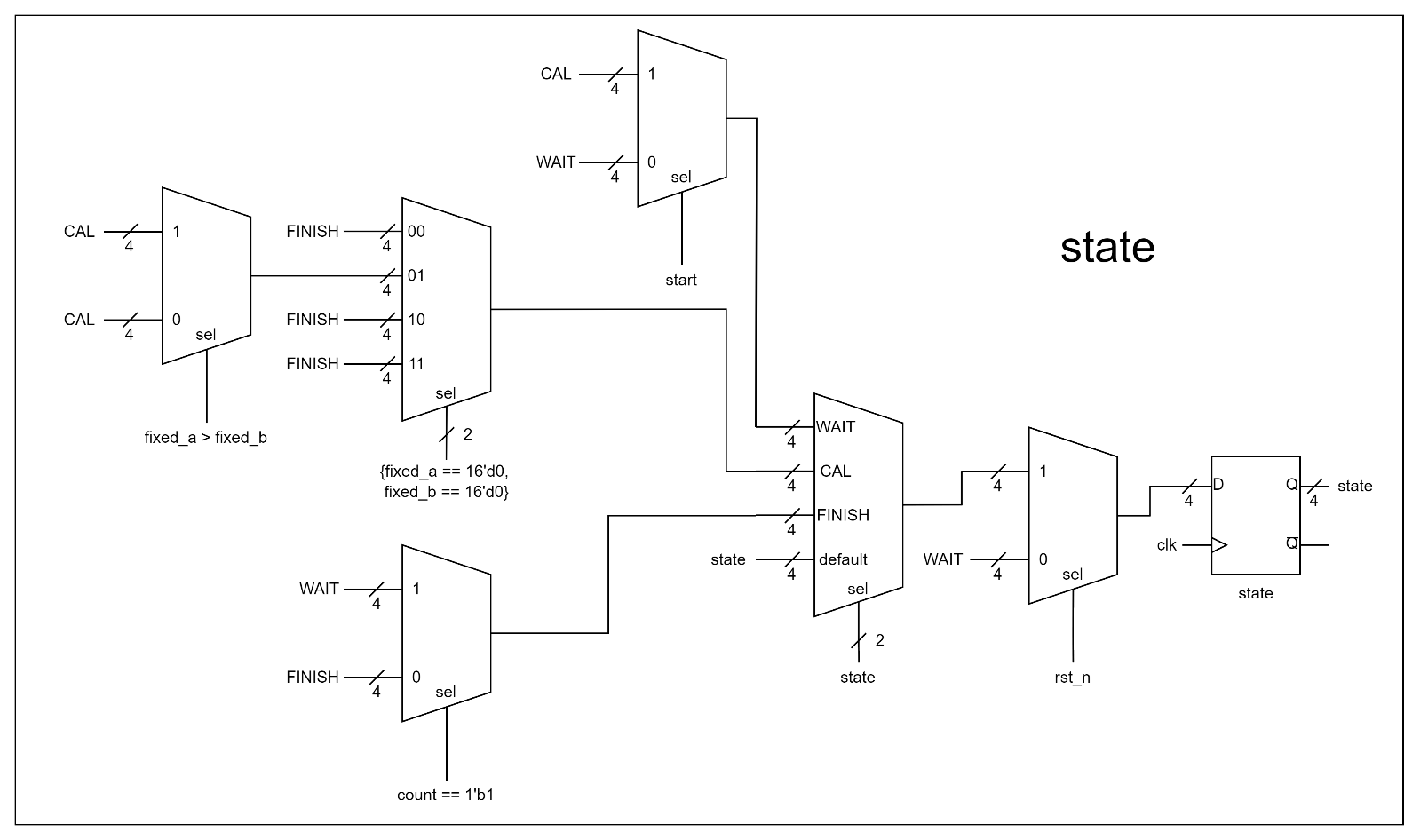
****

Figure 3.1

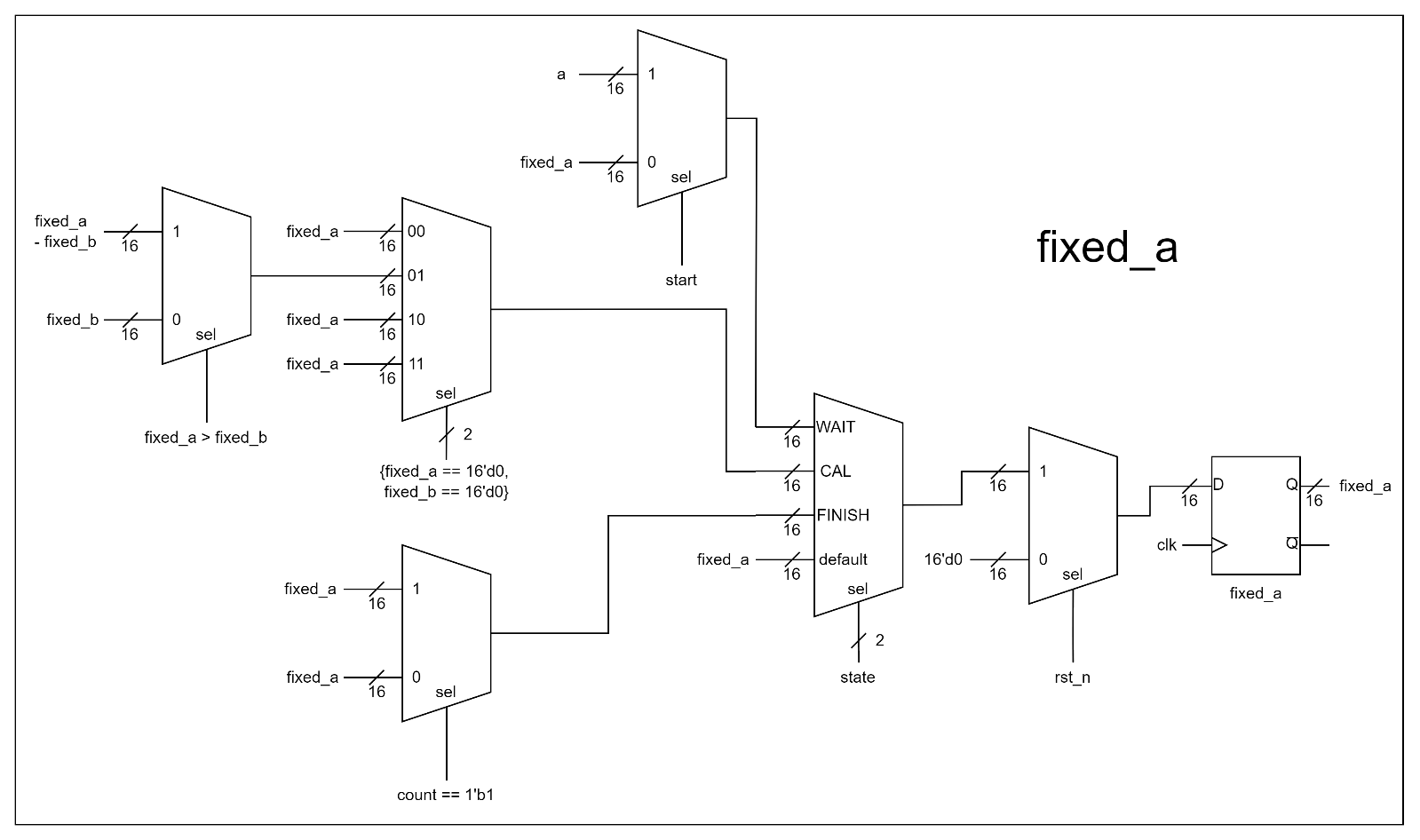
****

Figure 3.2

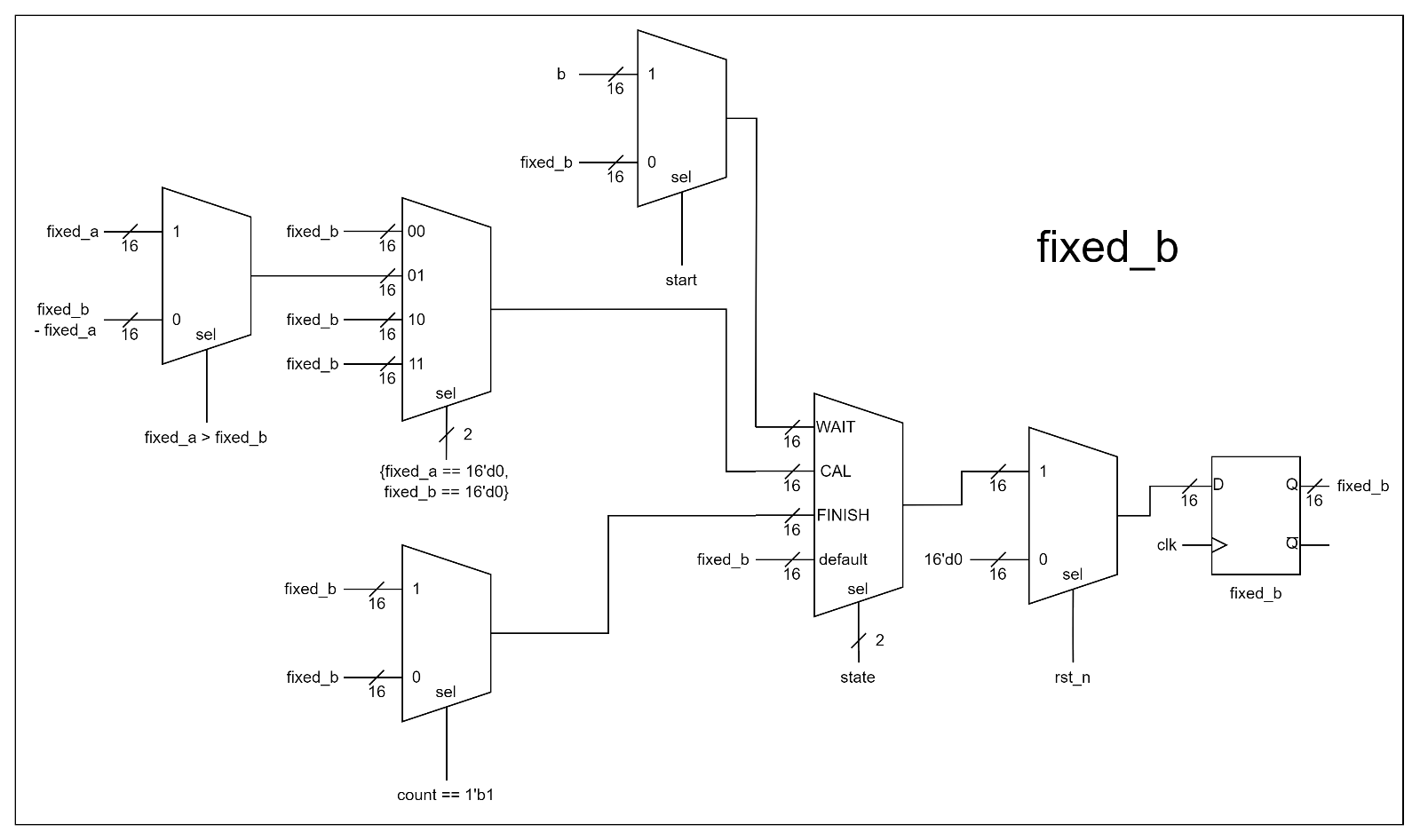
****

Figure 3.3

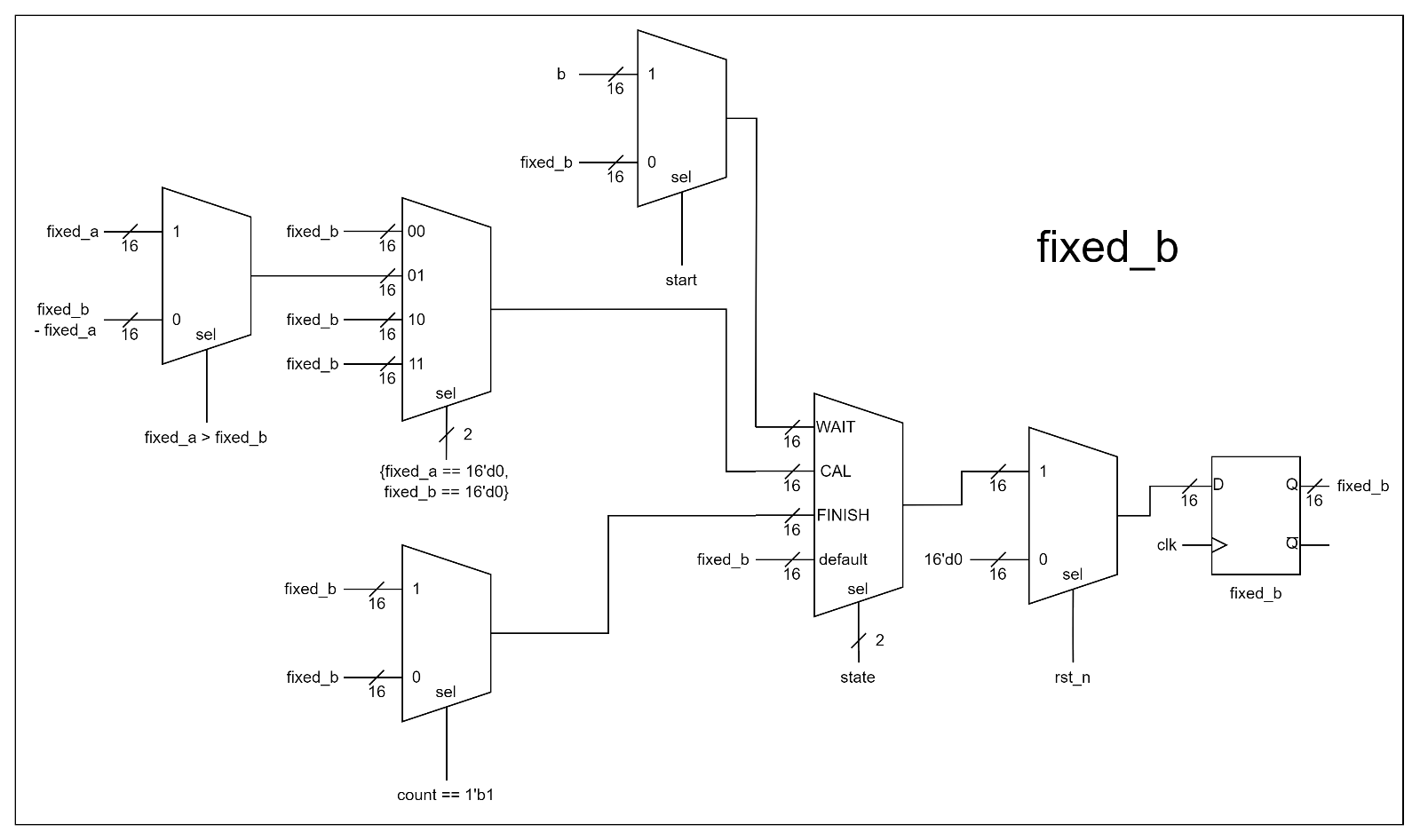
****

Figure 3.4

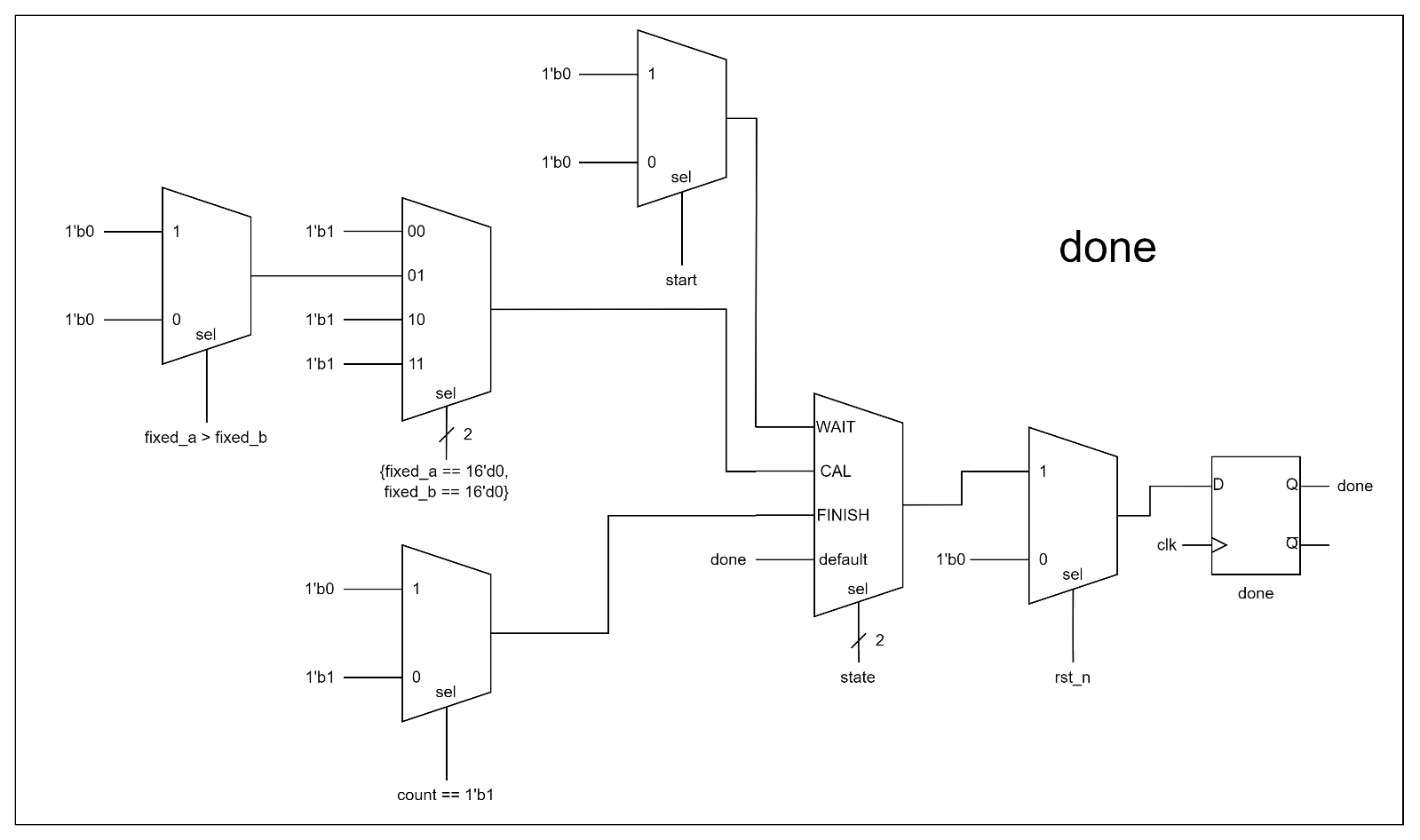
****

Figure 3.5

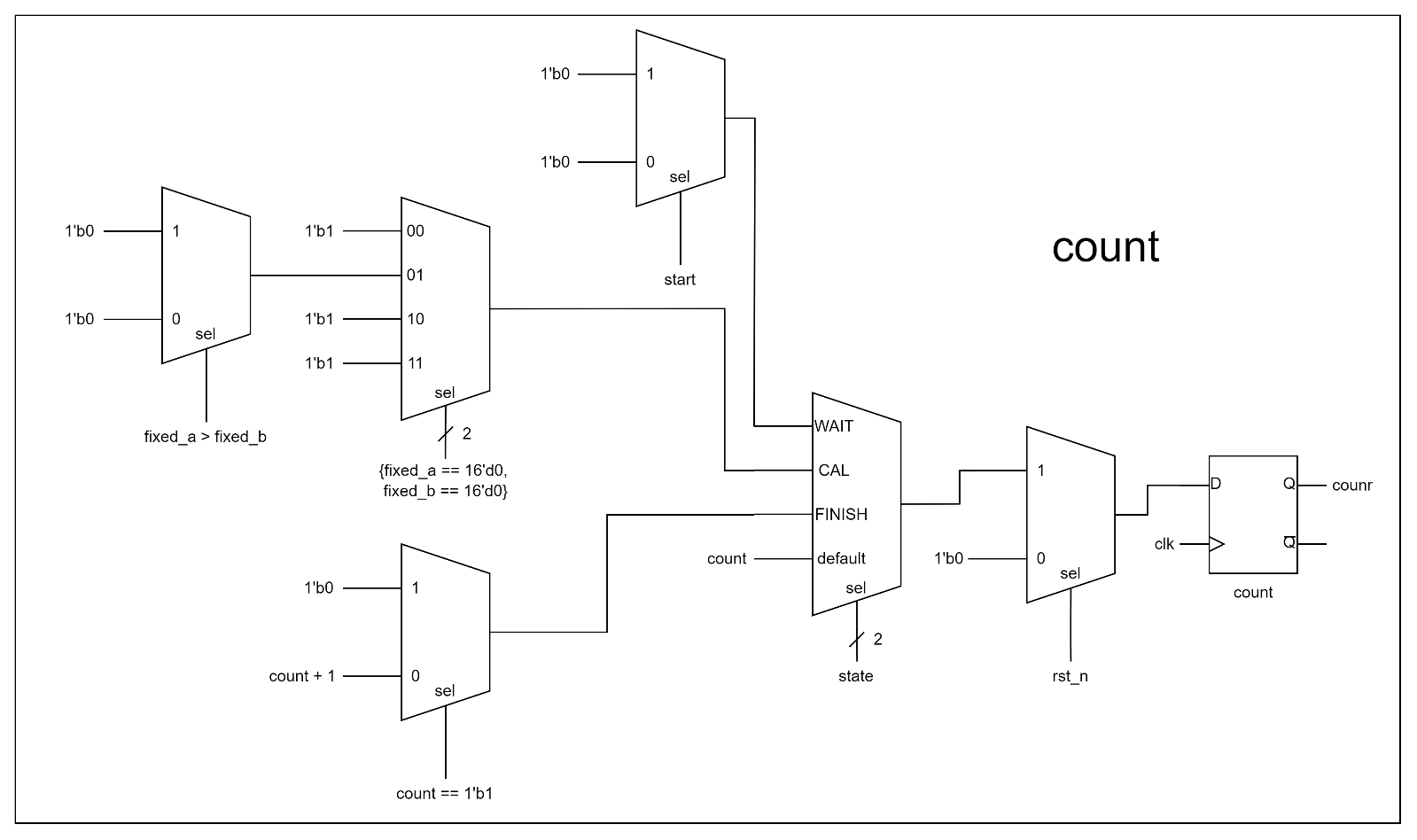
****

Figure 3.6

1. **Finite State Diagram**

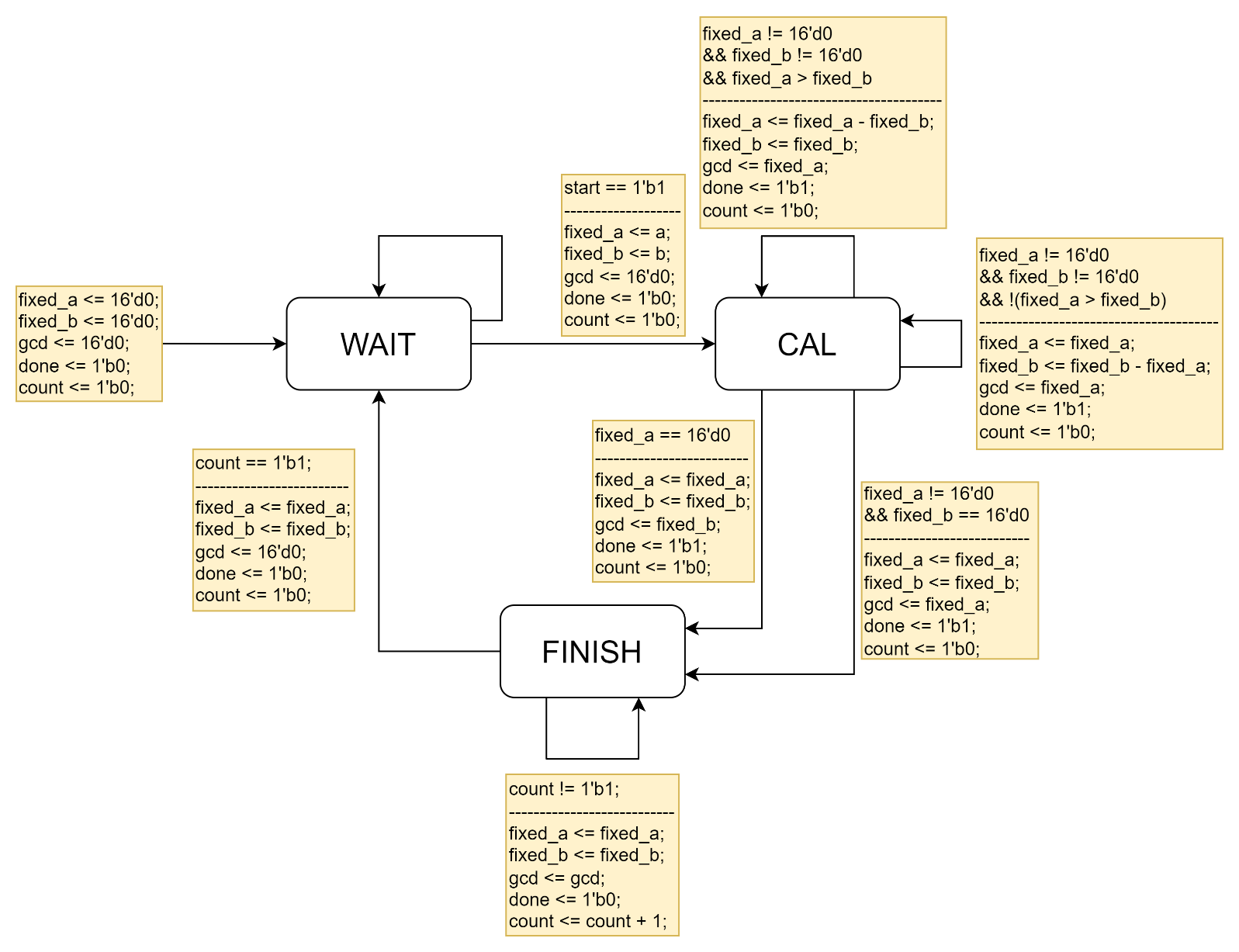


Figure 3.7

1. **Explanation**

In this problem, most of rules are provided in the lab slide. Therefore, I just follow the slide and use moore machine to realize the circuits. **Figure 3.1** to **Figure 3.6** show that the circuits I designed. Because **a, b** can change at any time, I use **fixed\_a** and **fixed\_b** to store the value while **state** change from **WAIT** to **CAL**.

1. **Testbench**
2. **Advanced Question: Built-in Self Test**
3. **Block Diagram**

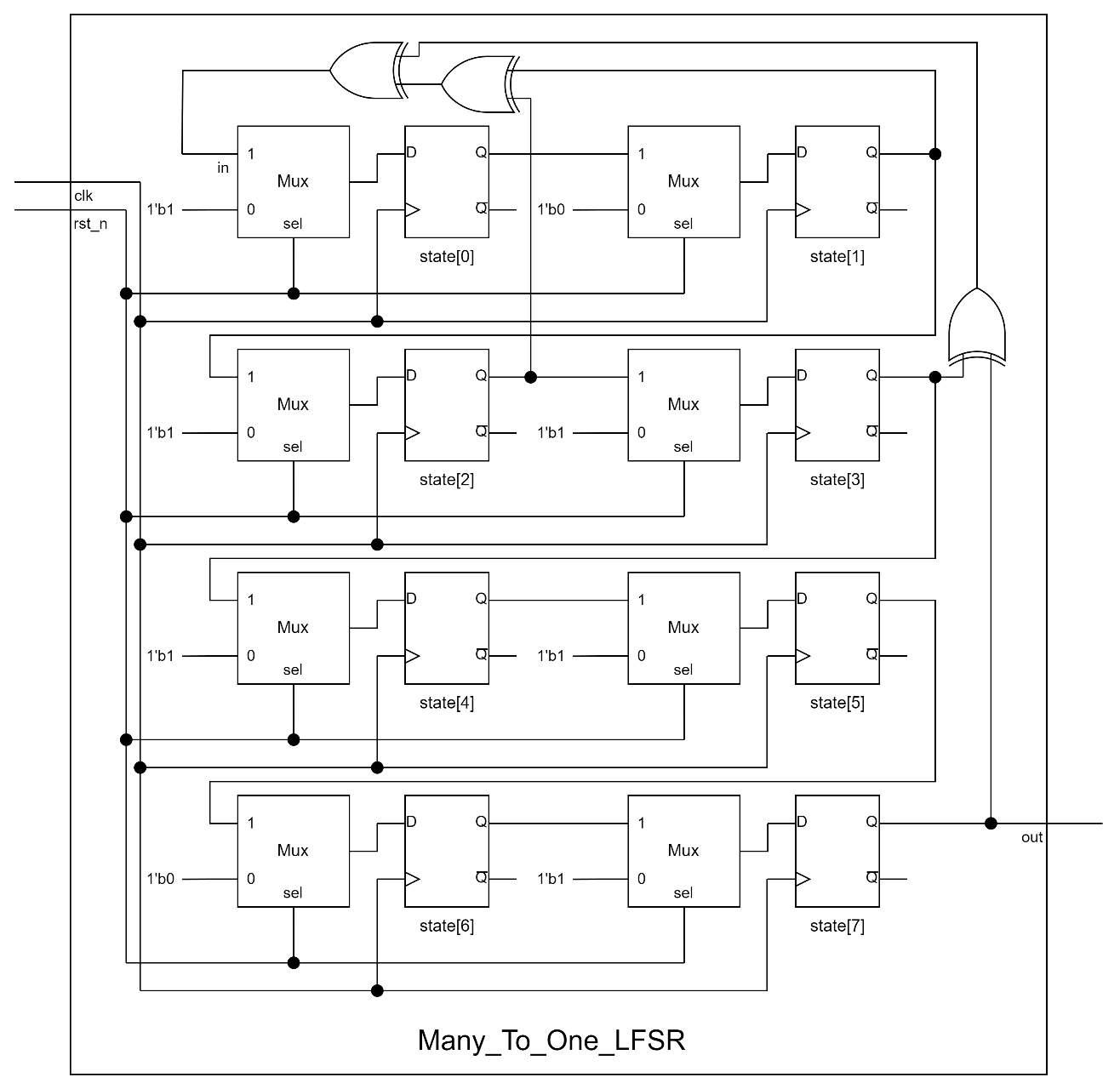
****

Figure 4.1 Many\_To\_One\_LFSR

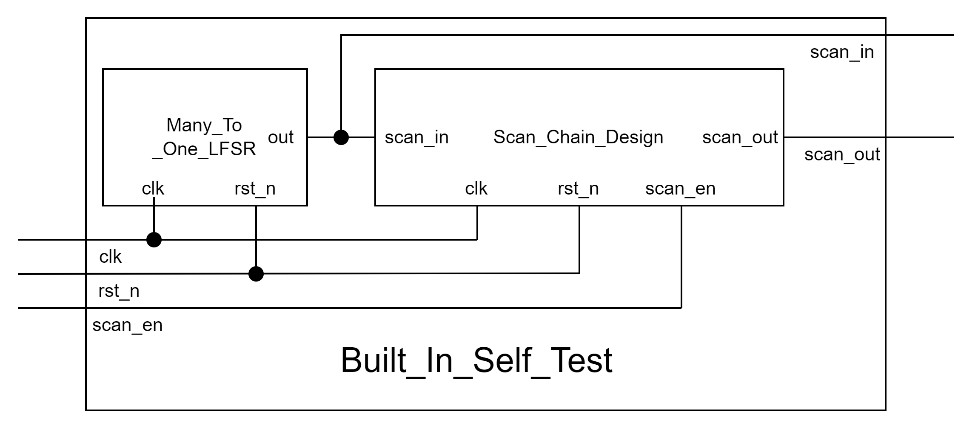
****

Figure 4.2 Built\_In\_Self\_Test

1. **Explanation**

In this problem, I only have to combine to modules I 've designed in the previous problems. **Figure 4.1** shows how I designed **Many\_To\_One\_LFSR**. **Figure 4.2** shows how I connect these two modules together.

1. **Testbench**

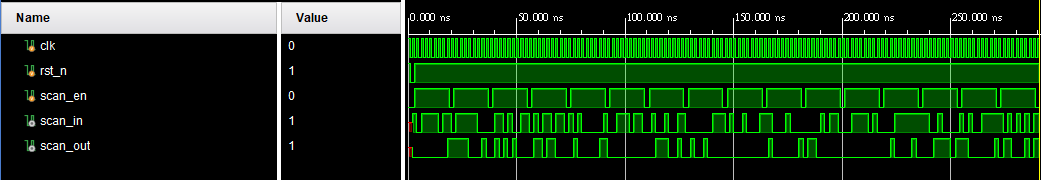


Figure 4.3 wave form

In this design, testcases are generated by **Many\_To\_One\_LFSR**. It seems that the method of how I designed in the testbench of **Scan\_Chain\_Design** is risky because there are some chances that I have some wrong cases in my testbench. Therefore, I write a code in C++ to simulate the testcases and check the results one by one. And every result seems right.

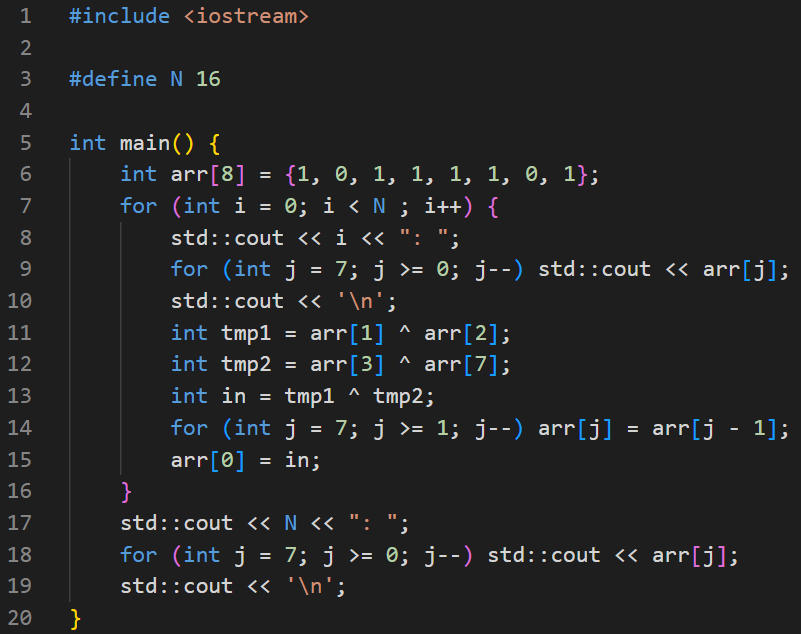


Figure 4.4

C++ code for simulating the testcases

1. **Advanced Question: Built-in Self Test FPGA**
2. **Block Diagram**

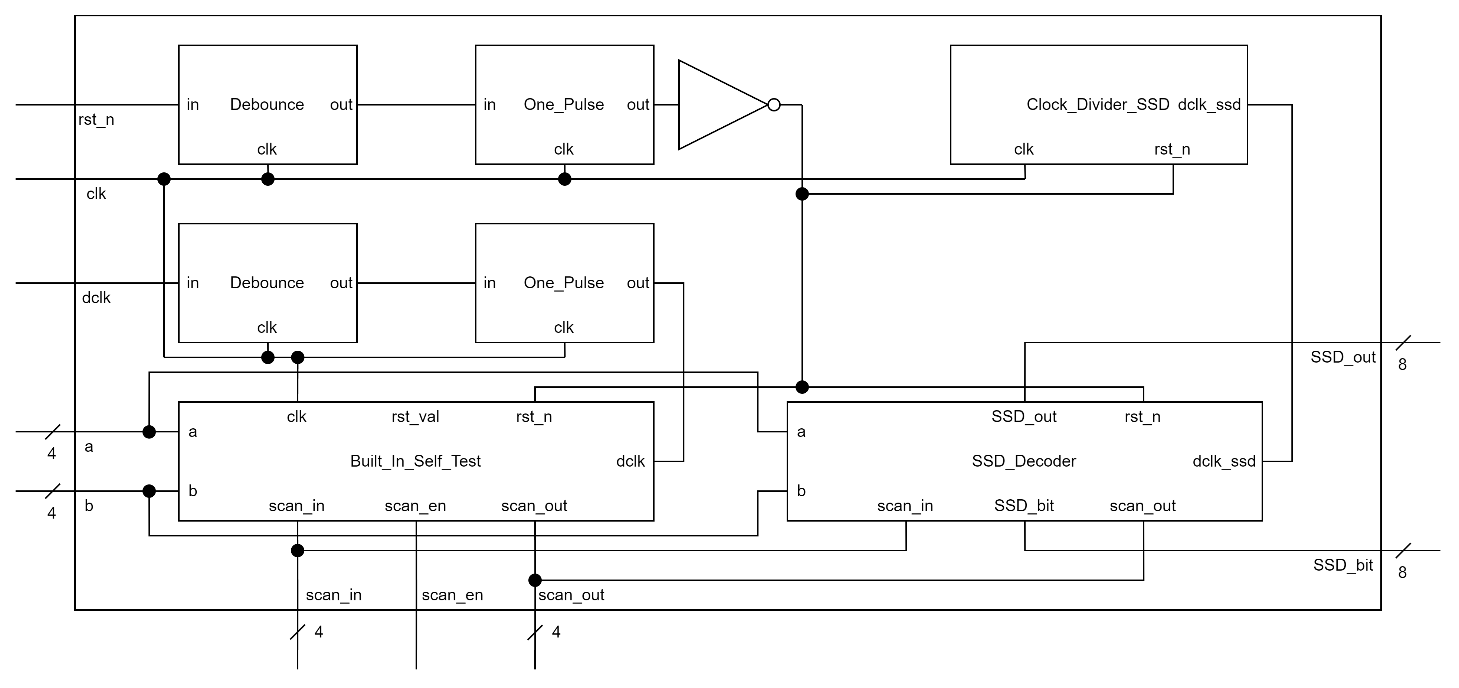
****

Figure 5.1

The whole design of this problem

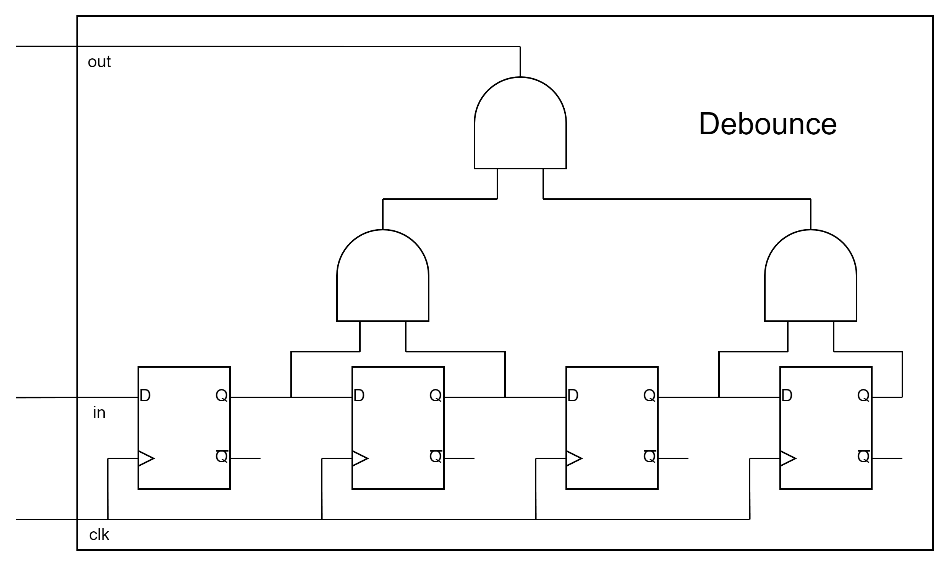
****

Figure 5.2

Debounce

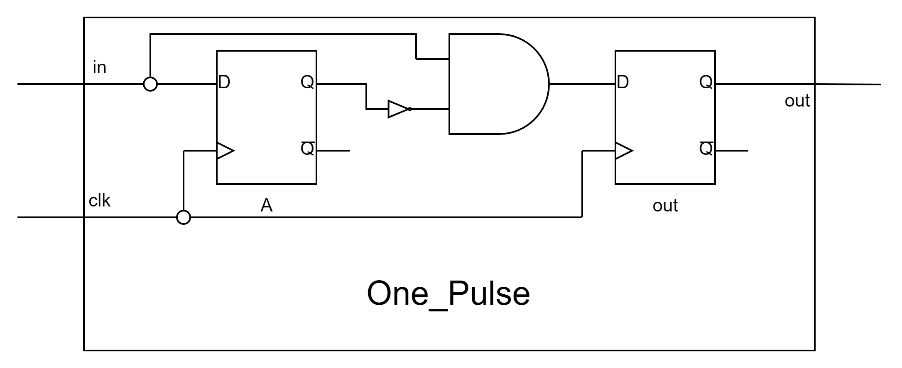
****

Figure 5.3

One\_Pulse

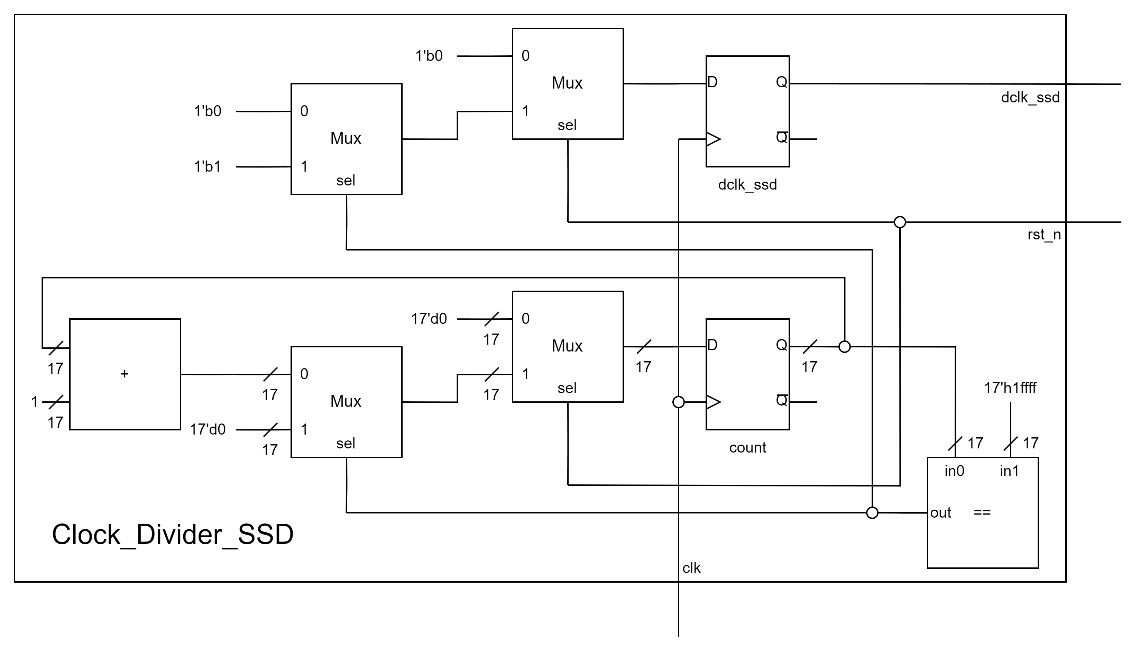
****

Figure 5.4

Clock\_Devider\_SSD

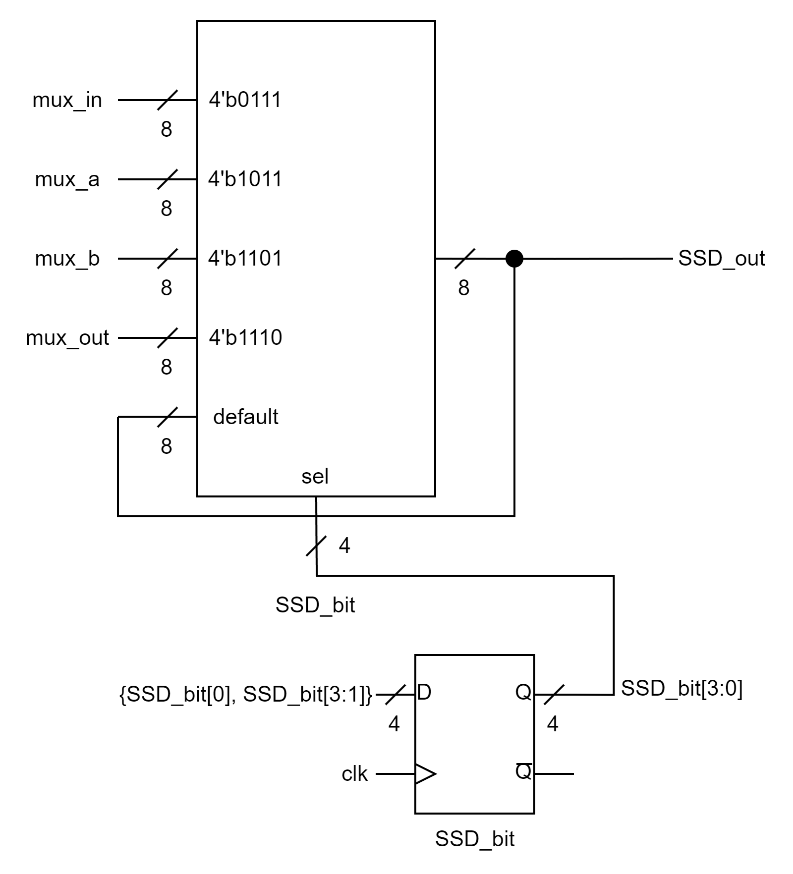
****

Figure 5.5

SSD\_Decoder

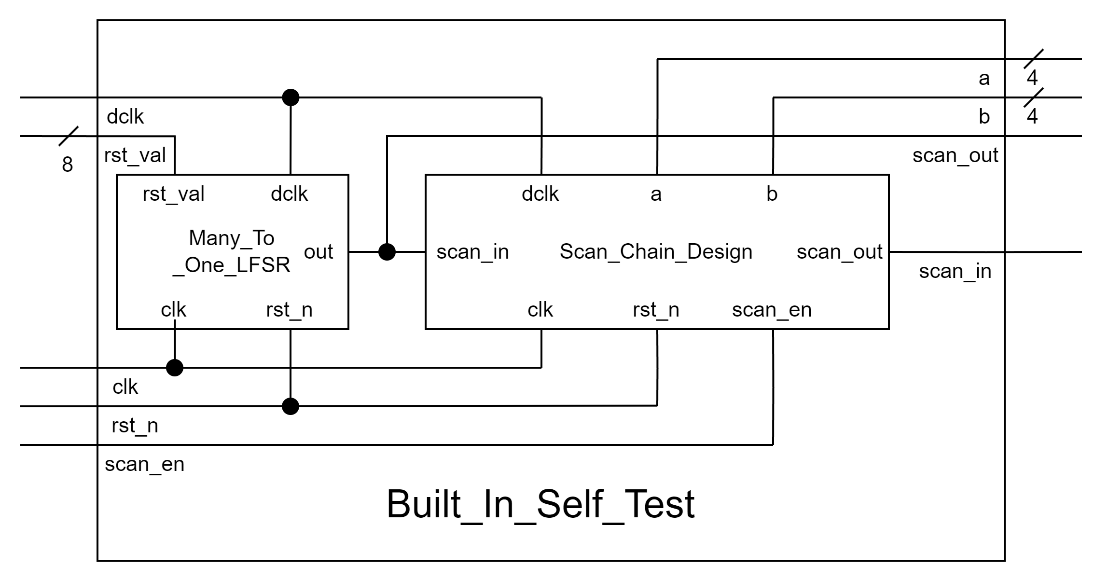
****

Figure 5.6

Built\_In\_Self\_Test

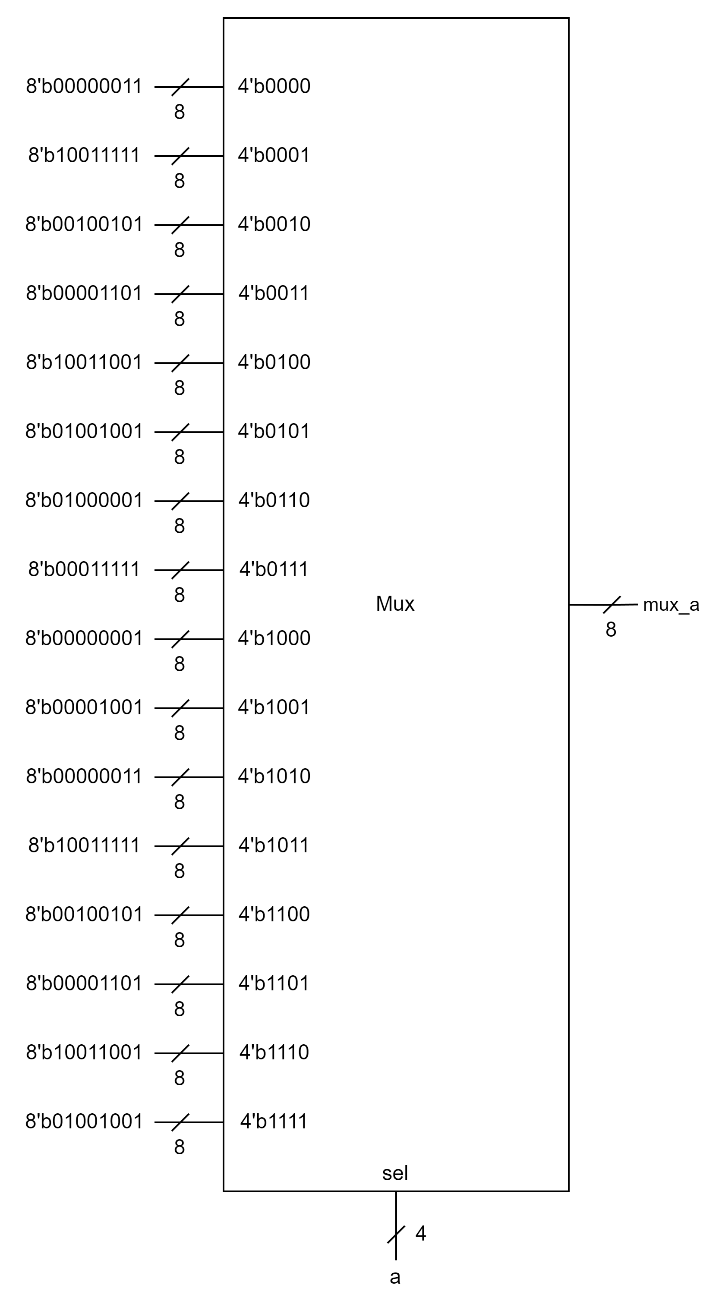
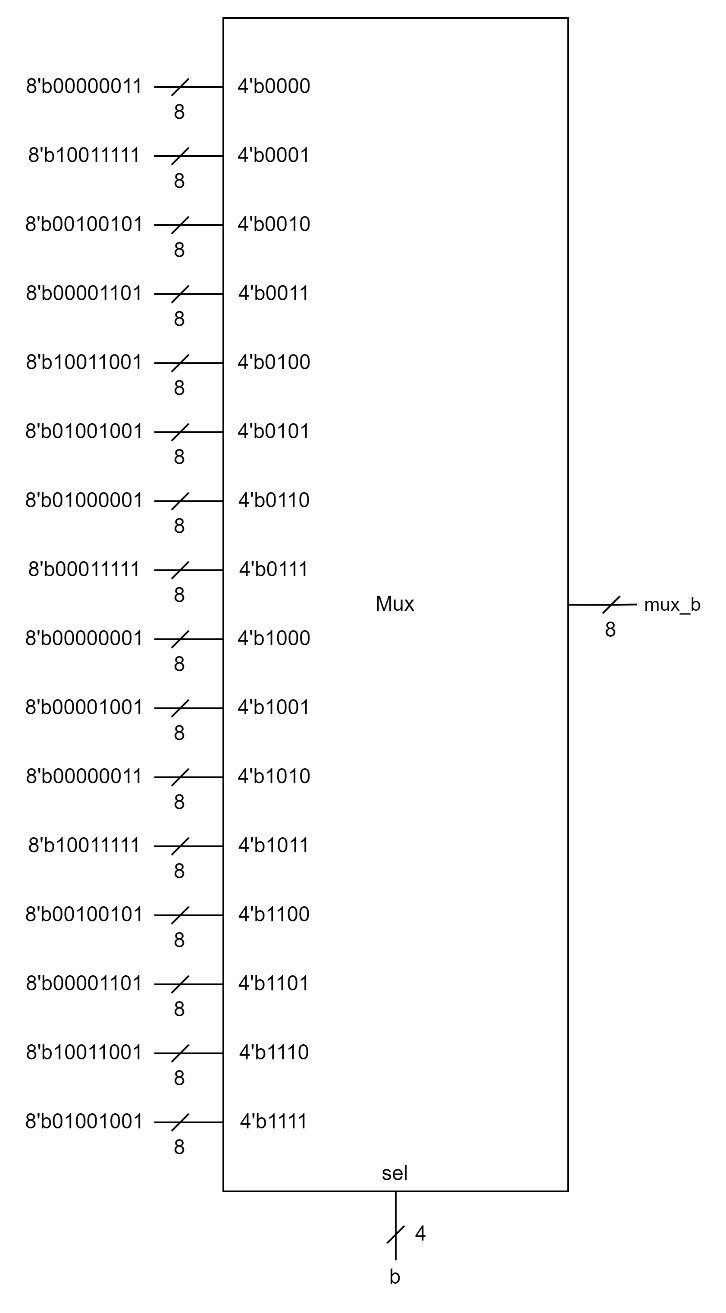
****

Figure 5.8

mux\_b

Figure 5.7

mux\_a

****

Figure 5.10

mux\_out

Figure 5.9

mux\_in

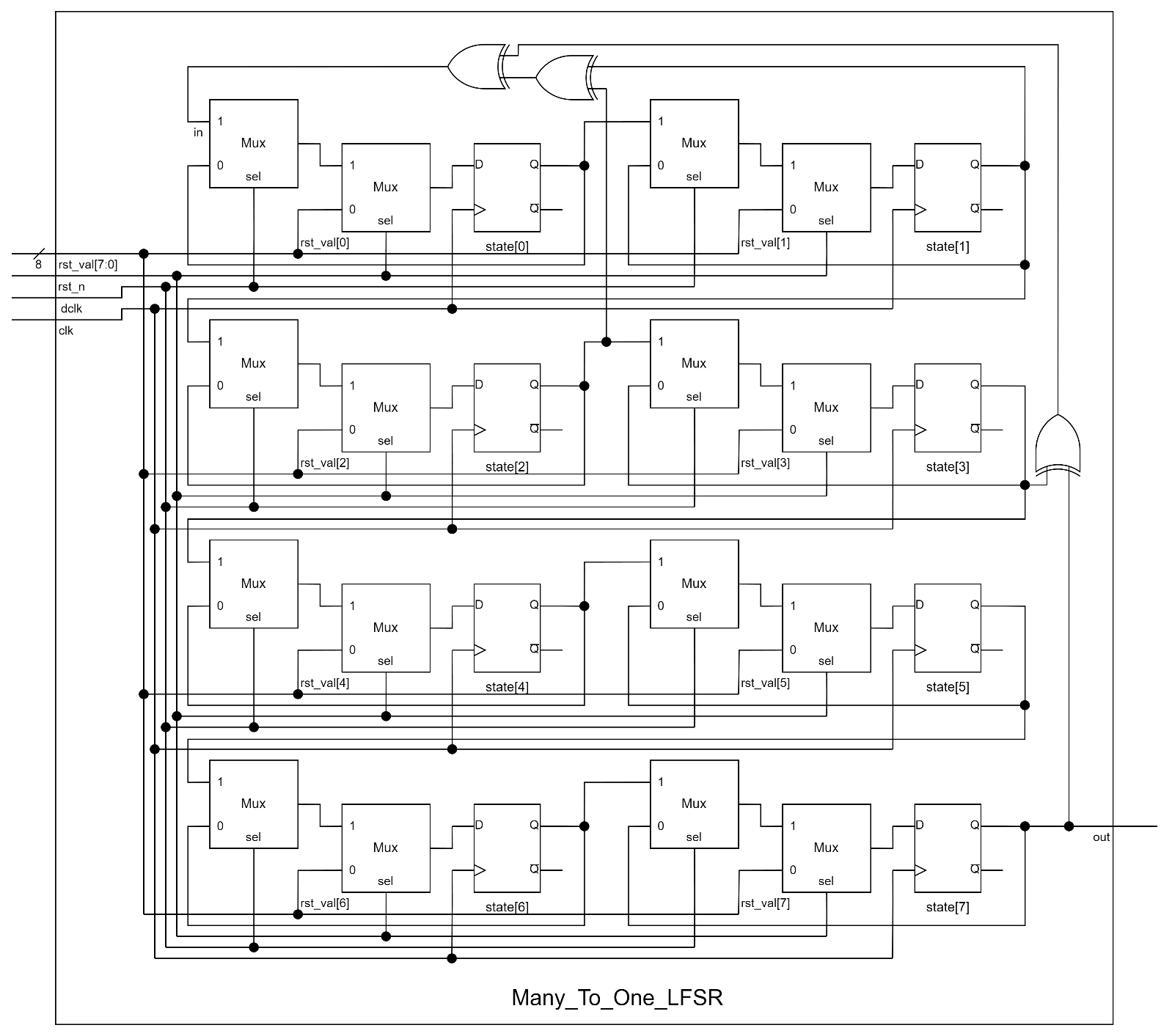
****

Figure 5.11

Many\_To\_One\_LFSR

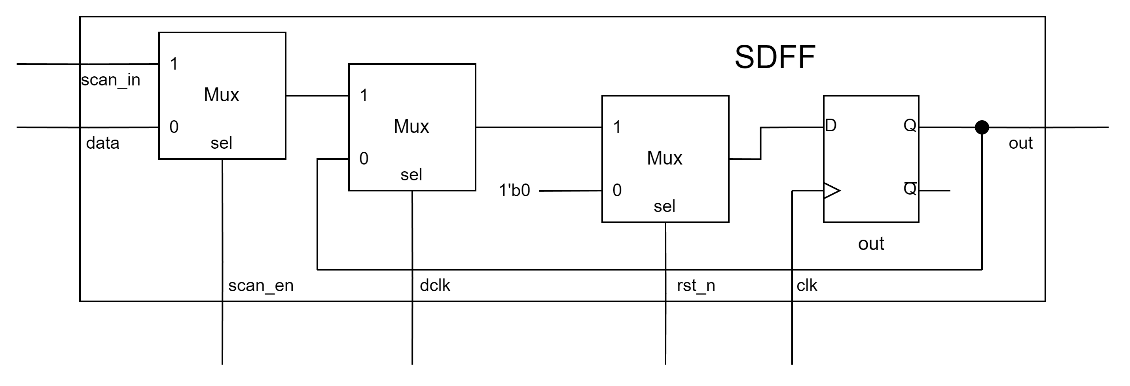
****

Figure 5.12

SDFF

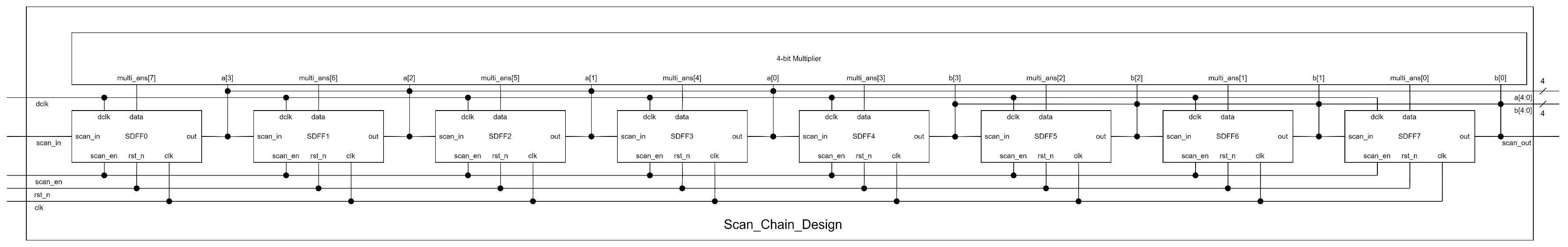
****

Figure 5.13

Scan\_Chain\_Design (Left Part)

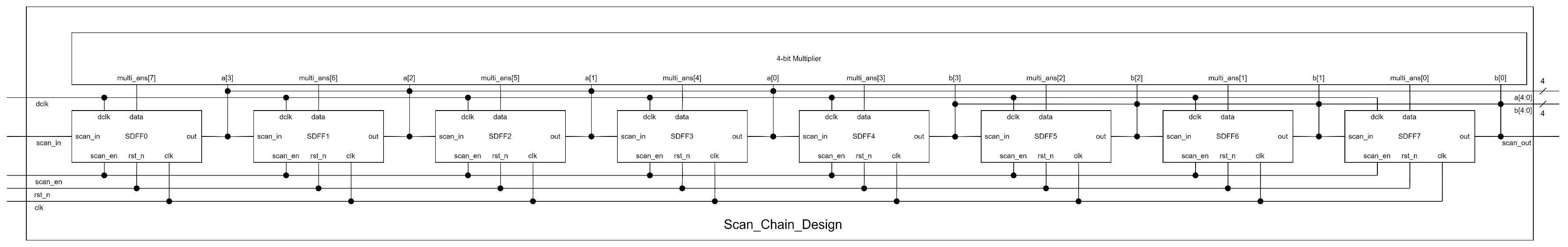
****

Figure 5.14

Scan\_Chain\_Design (Left Part)

1. **Explanation**

According to the specification, I divided the circuit into different modules as **Figure 5.1** shows. In order to work properly on FPGA boards, divided clock is needed. Most of the modules above are modified from the designs of the previous problems by adding a mux which selected by **dclk** before connect into DFFs. However, after testing on the FPGA board, I noticed that there is some chance that the DFFs will go through two states while I only press the button one time. To solve this problem, I use a divided clock to control **Debounce** module as **Figure 5.15** shows. In this way, it seems the condition mentioned above doesn't happen again while I was testing it on the FPGA board.

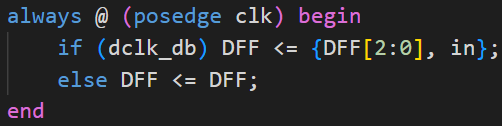


Figure 5.15

1. **Advanced Question: Mealy Sequence Detector**
2. **State Diagram**

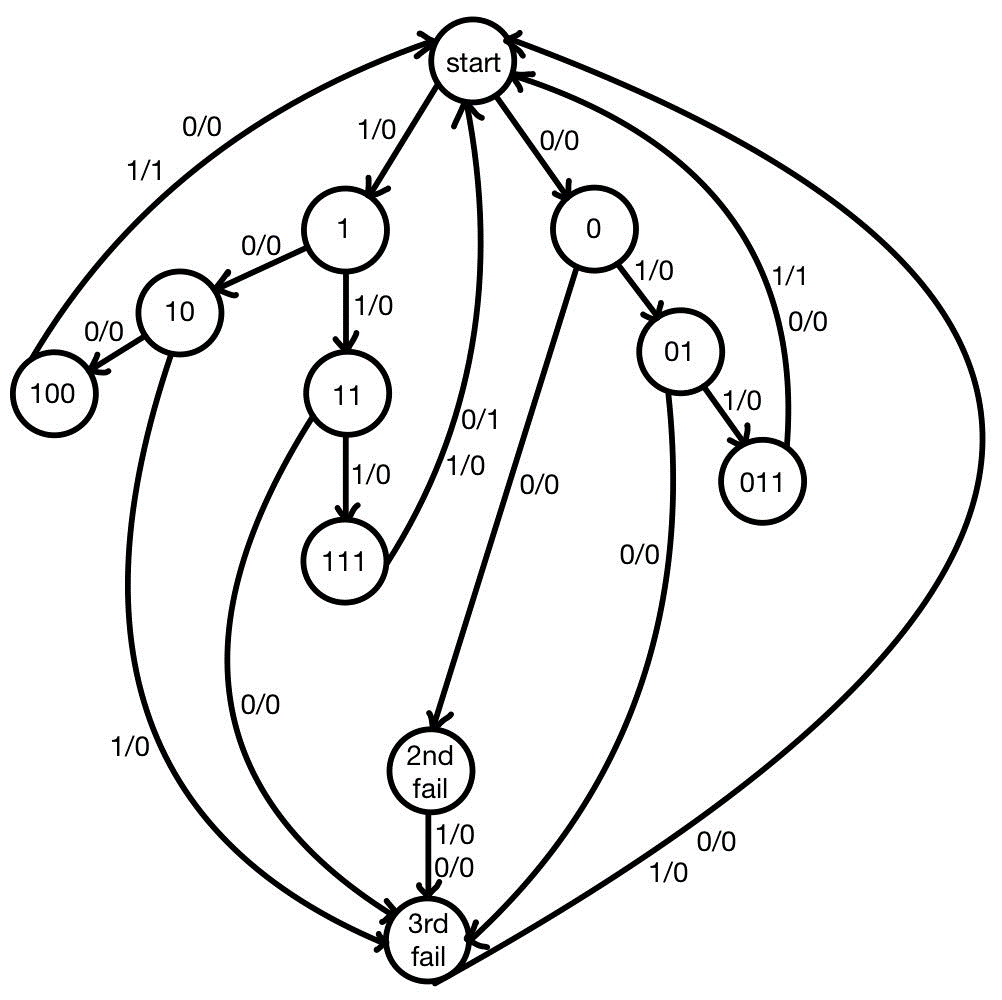


Figure 6.1

1. **Explanation**

The problem requires that the mealy machine should redetect every 4 bits. Therefore, I define a state called **start**, which means that there is no bit input. And then I define some states as **Figure 6.1** shows. The states can be classified into four different groups by how many bits are pushed in.

* **No input: start**
* **One-bit input: 0, 1**
* **Two-bit input: 01, 10, 11, 2nd fail**
* **Three-bit input: 100, 111, 011, 3rd fail**

**I**f the mealy machine detects a wrong input, the state will change into **2nd fail** or **3rd fail**, depending on the number of the bits that have been pushed in. For example, if the mealy machine detects the wrong bit while input the second bit, the state will be changed into **2nd fail**.

1. **What I Have Learned**

Most of modules are provided in the lecture slides, thus I spent much less time designing modules. However, it is difficult to design a testbench in this lab. In order to check the results, I used the method that I've learned from Lab2 and also wrote a C++ program to generate testcases correctly to help me debug. In this lab, I've learned that how to combined the skills that I've learned before to solve problems. I hope that I can use these skills more flexible in the future. In addition, I encountered a strange condition while I was doing the FPGA problem. The are some chances that if I press the **dclk** button once, the state will change twice. In order to solve this problem, I consulted my classmate who has study Logic Design Lab last semester. She told me that if clock is too fast, **Debounce** will have some chances that works unproperly. Usually, **Debounce** should use a 100Hz clock. To prevent from the same bug, I will take her advice and implement it in future problems.