**Hardware Design and Lab: Lab5**

**111060013 EECS 26' 劉祐廷**

**Catalog**

**1. Advanced Question:**

**Sliding Window Sequence Detector……………...………………P3**

**2. Advanced Question:**

**Traffic Light Controller…………………………..…...………….P5**

**3. Advanced Question:**

**Greatest Common Divisor….………………...……….………..P7**

**4. Advanced Question:**

**Booth Multiplier……………………………………….….…..P12**

**5. What I Have Learned…...................................................P14**

1. **Advanced Question: Sliding Window Sequence Detector**
2. **Finite State Diagram**

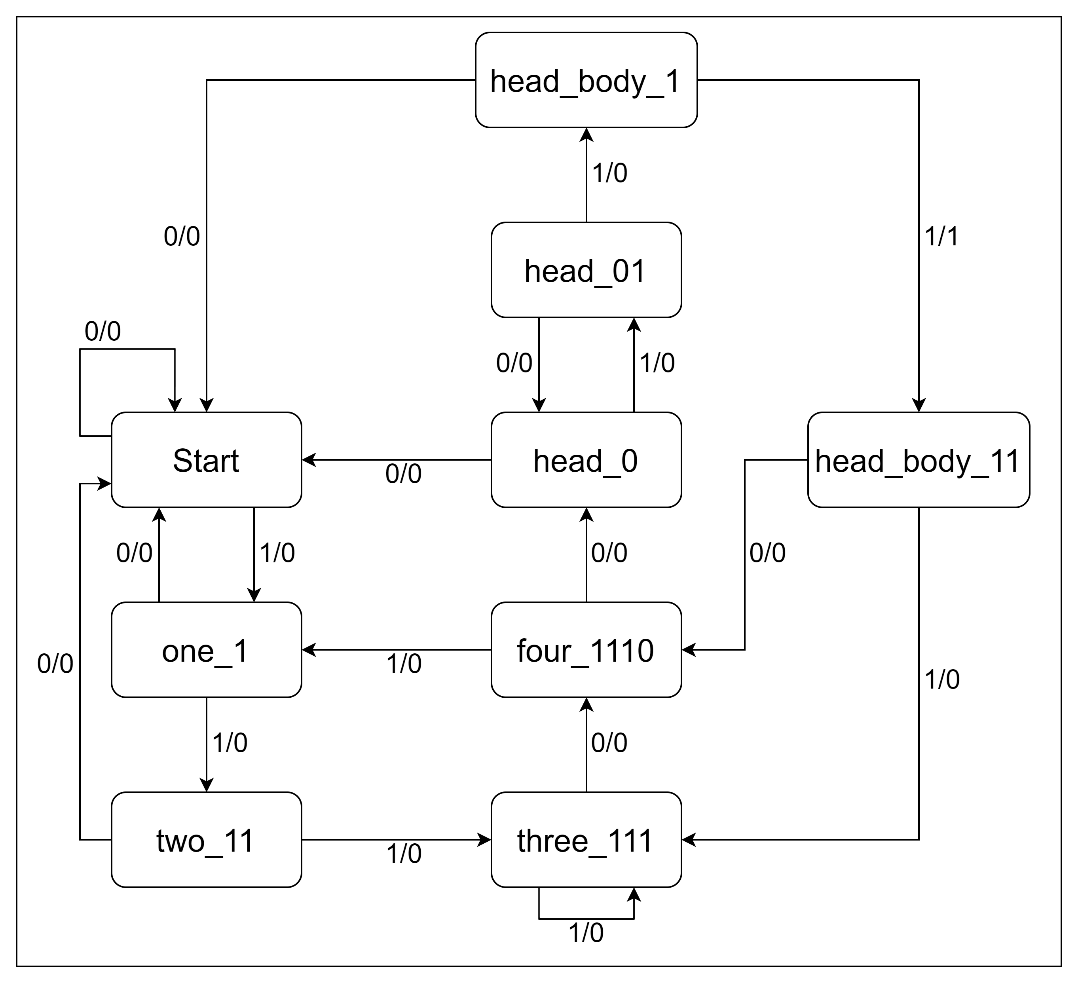
****

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**

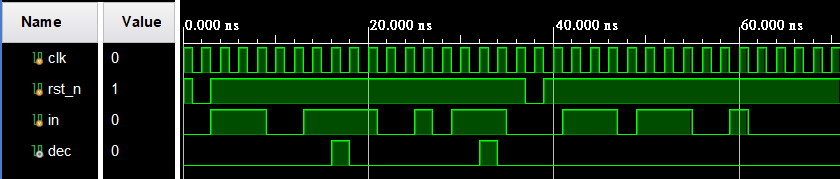


Figure 1.2

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**

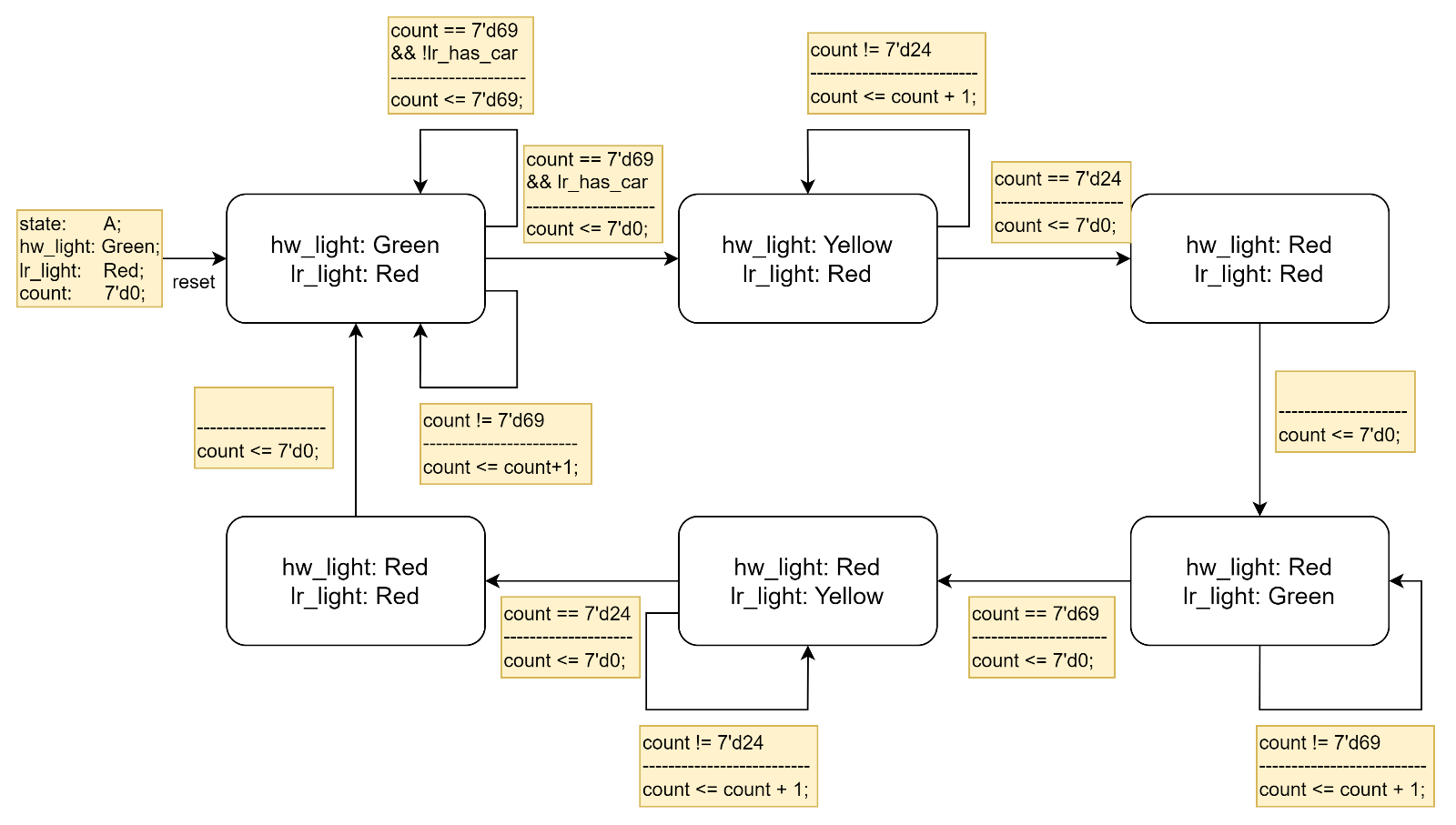
****

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**

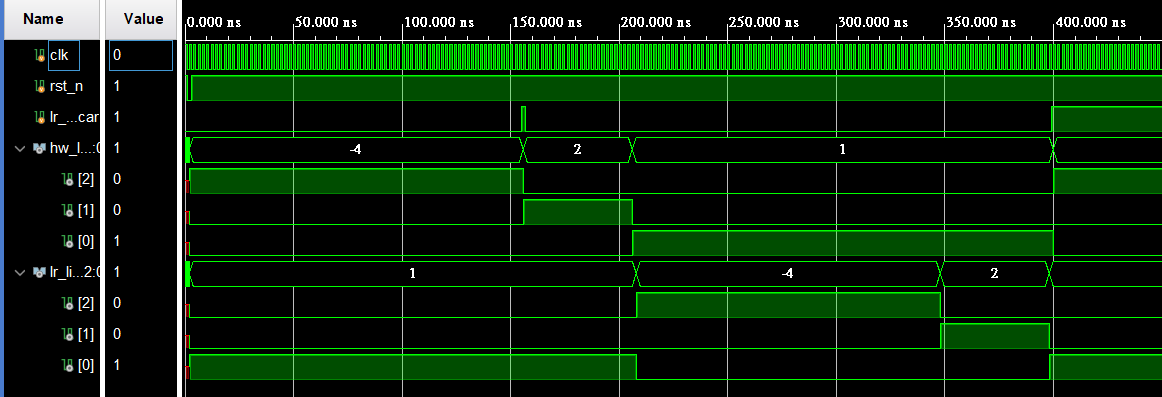
****

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.

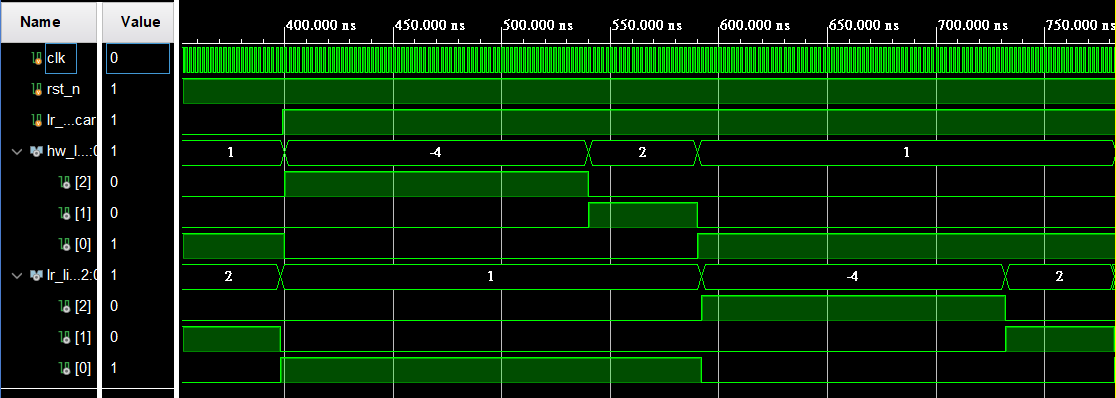


Figure 2.3

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**

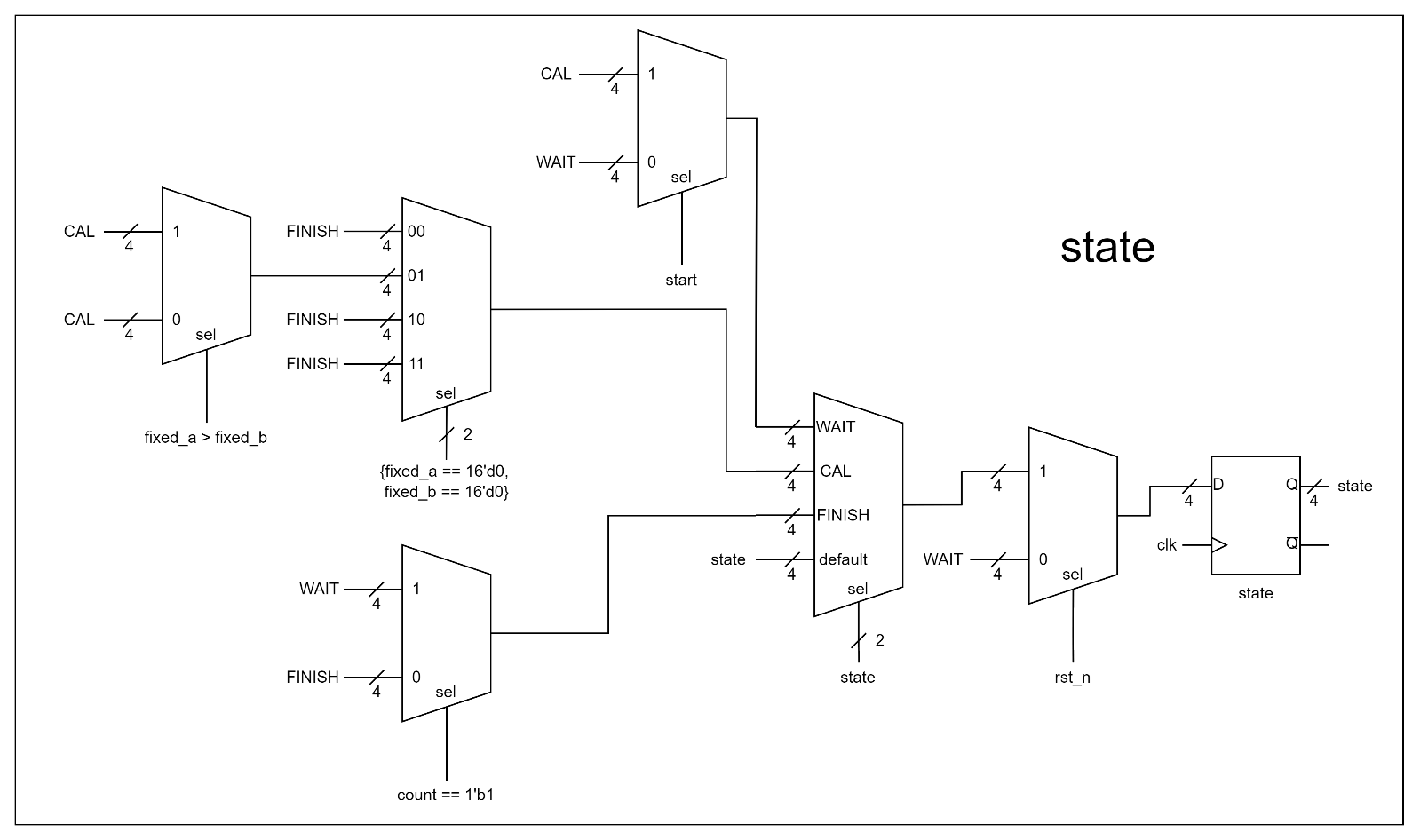
****

Figure 3.1

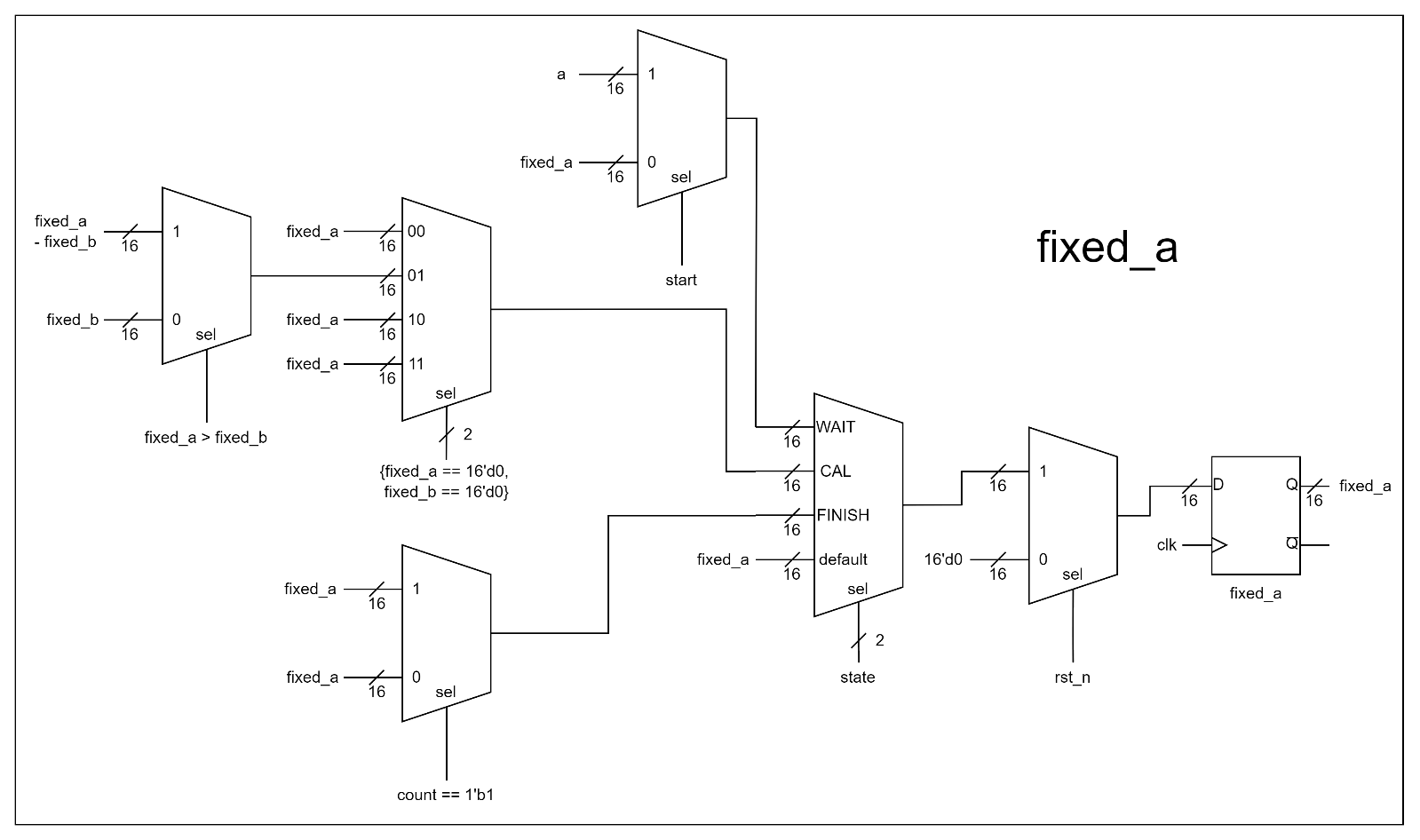
****

Figure 3.2

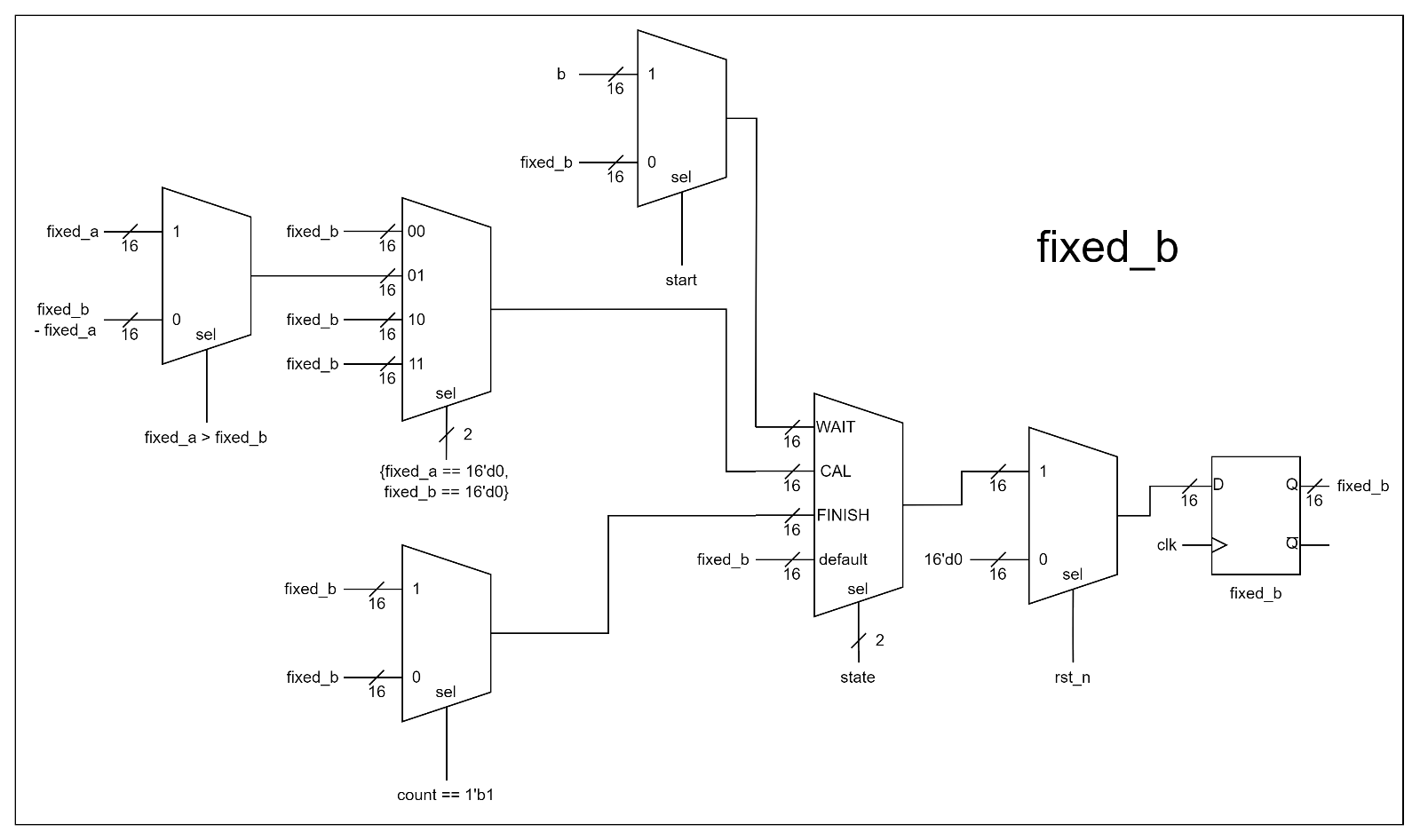
****

Figure 3.3

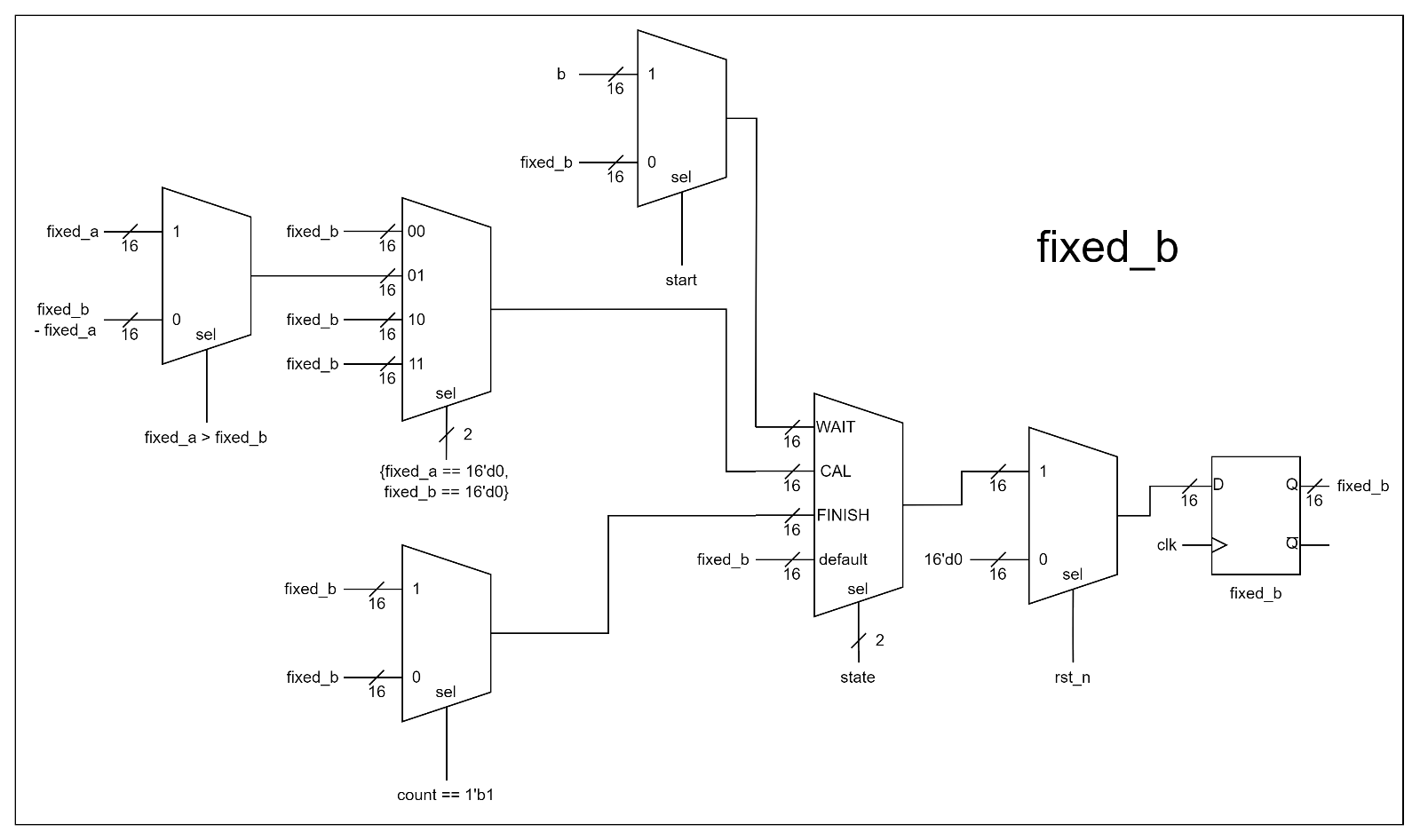
****

Figure 3.4

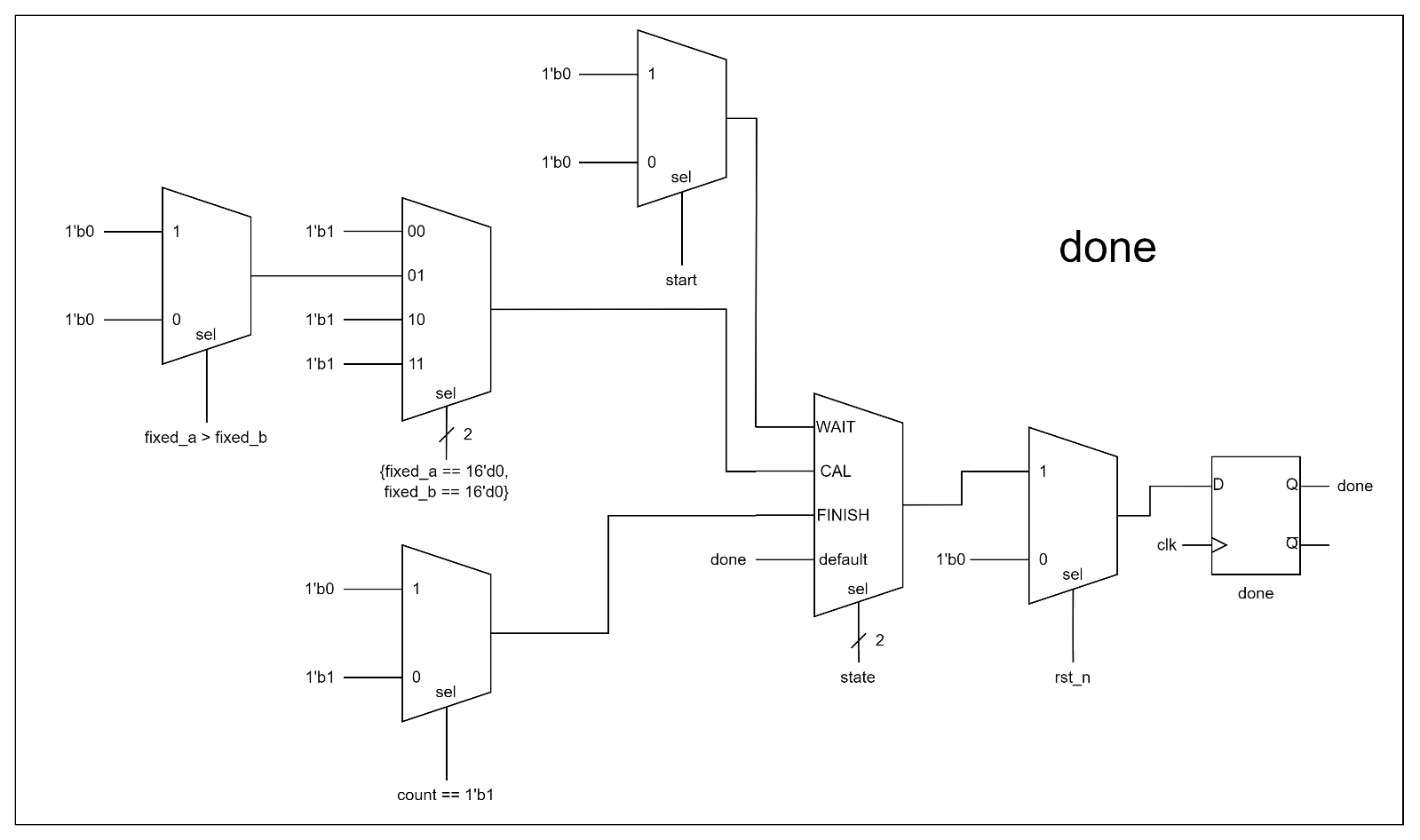
****

Figure 3.5

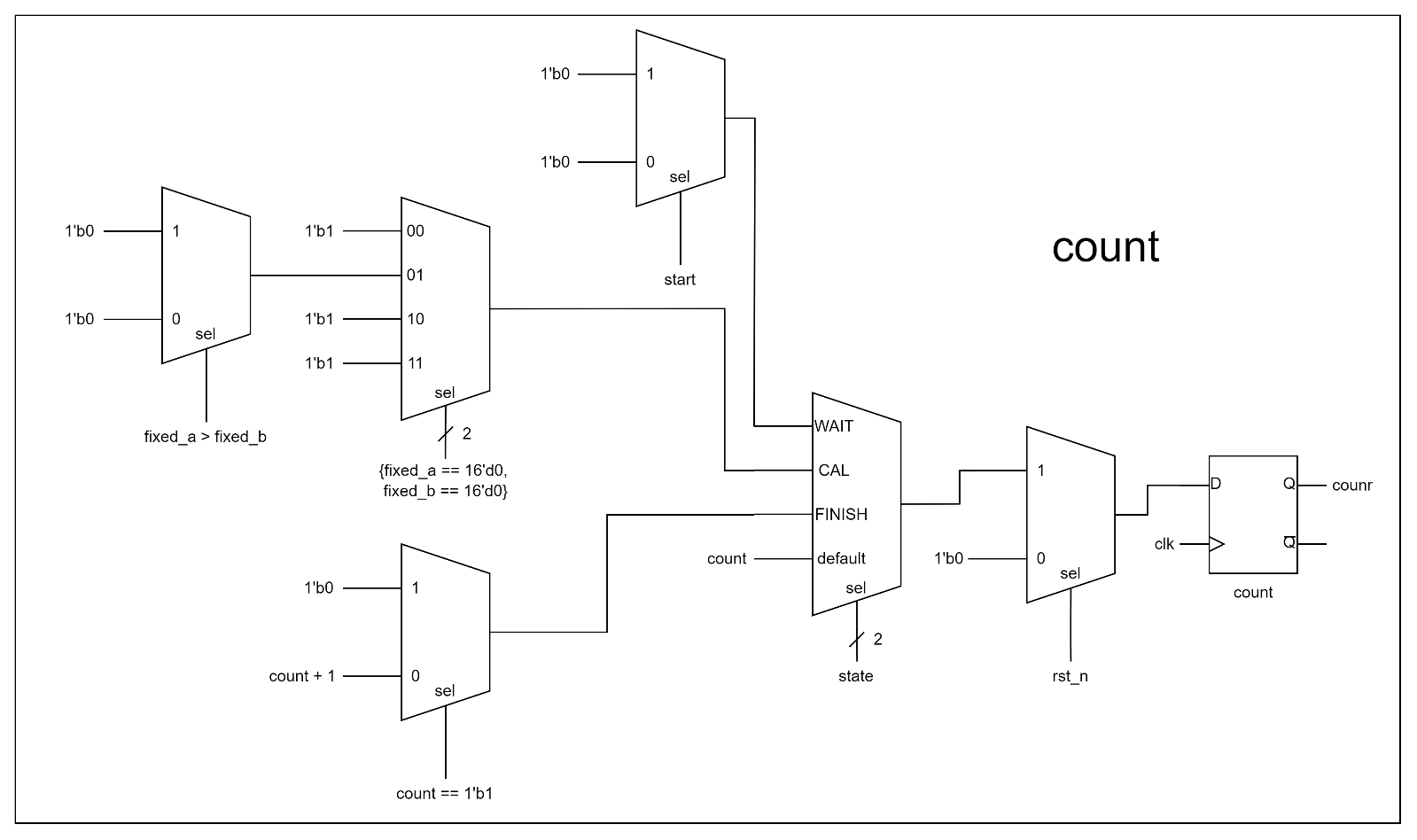
****

Figure 3.6

1. **Finite State Diagram**

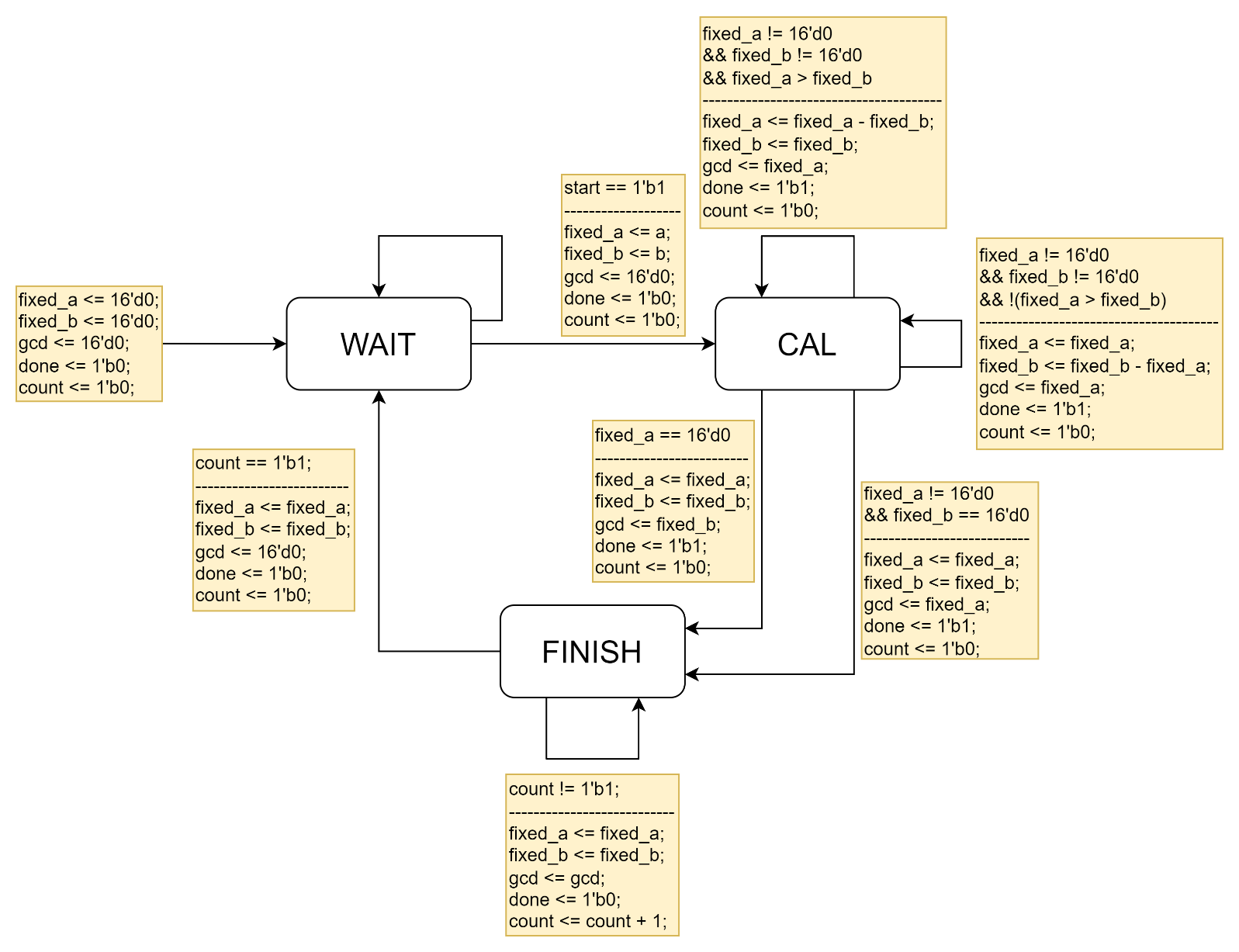


Figure 3.7

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**

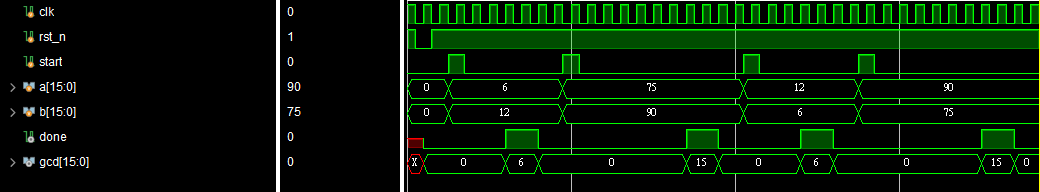


Figure 3.8

To test the design, I set **a** and **b** to different numbers and then change them and calculate them again. In this way, I can test if the Euclidean algorithm works properly. It seems that everything is right as **Figure 3.8** shows.

1. **Advanced Question: Booth Multiplier**
2. **Finite State Diagram**

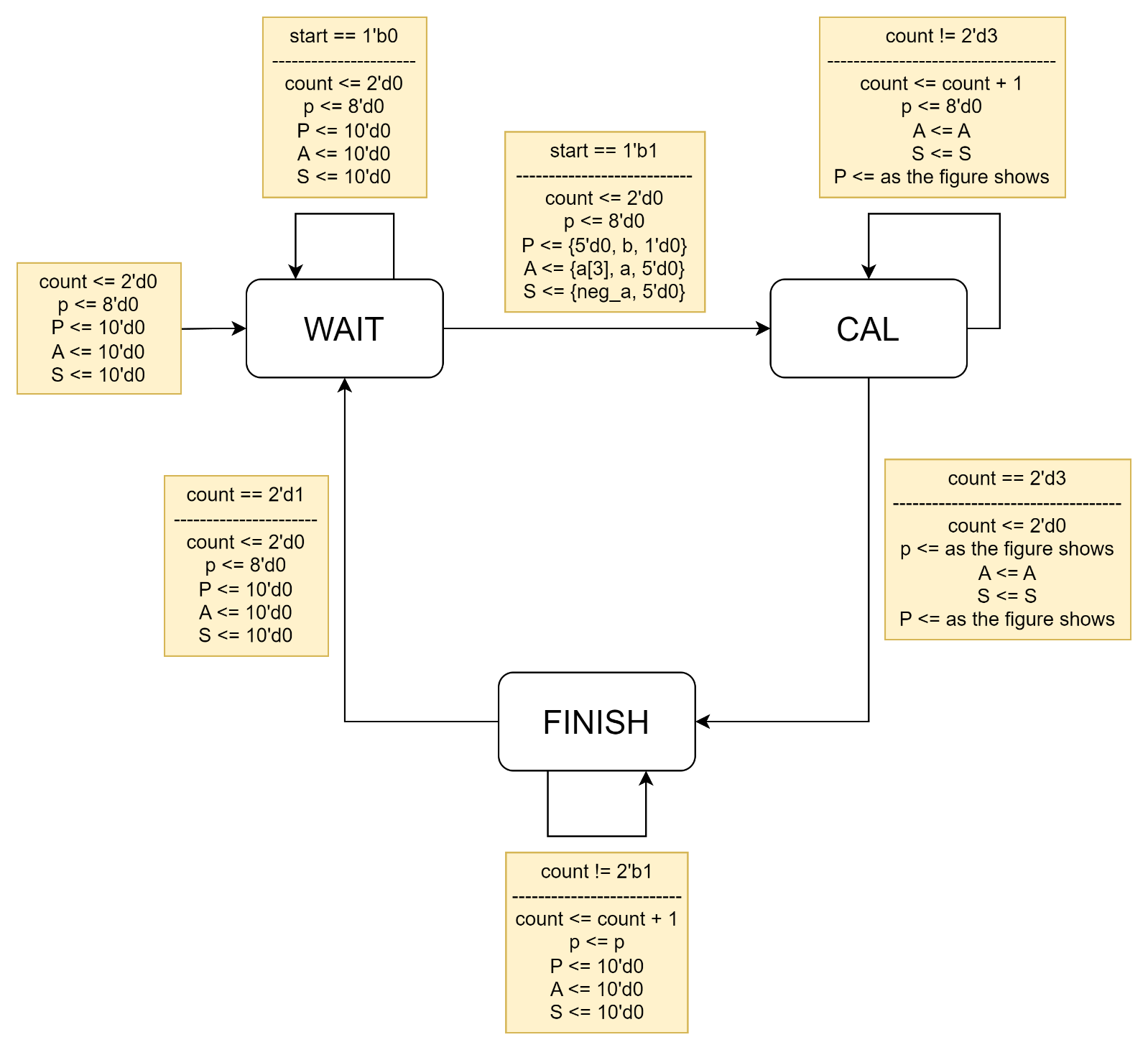
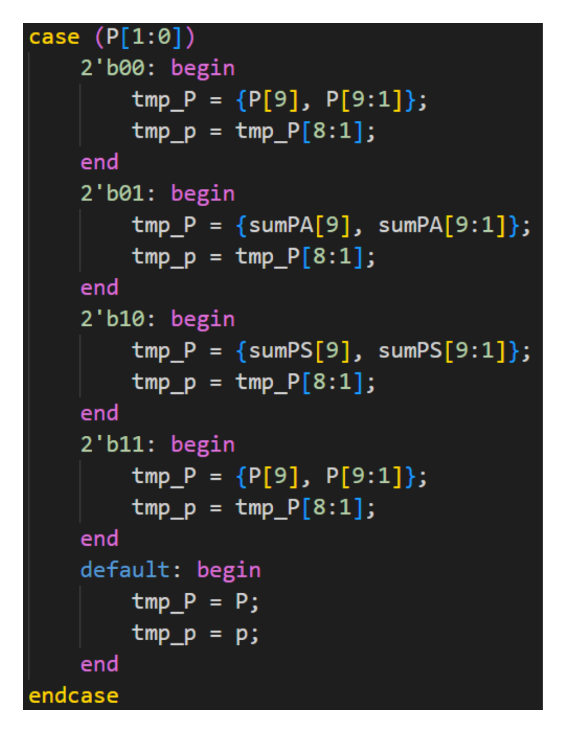
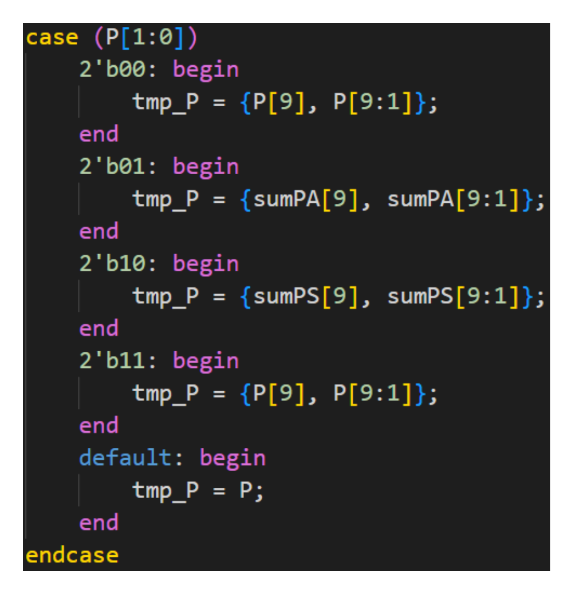
****

Figure 4.1

****

1. **Explanation**

In this problem, I designed a Moore machine according to **Figure 4.1**. Because Booth method needs the complement of the multiplicand, I use 5 bits for multiplicand to prevent from the error cause by overflow while it is equal to -8.

1. **Testbench**

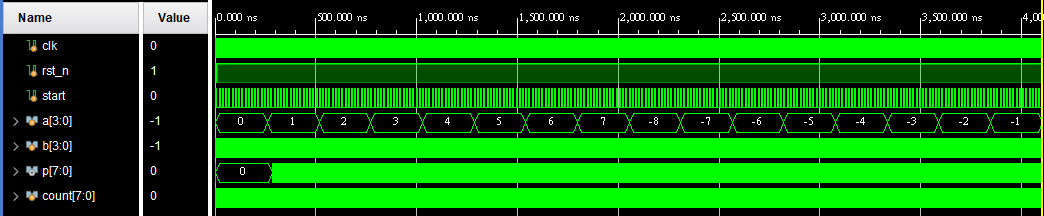


Figure 4.2

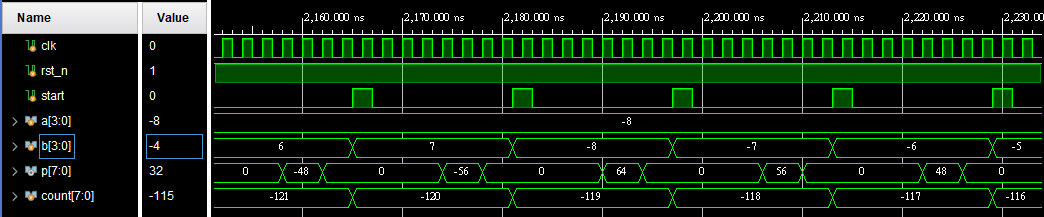
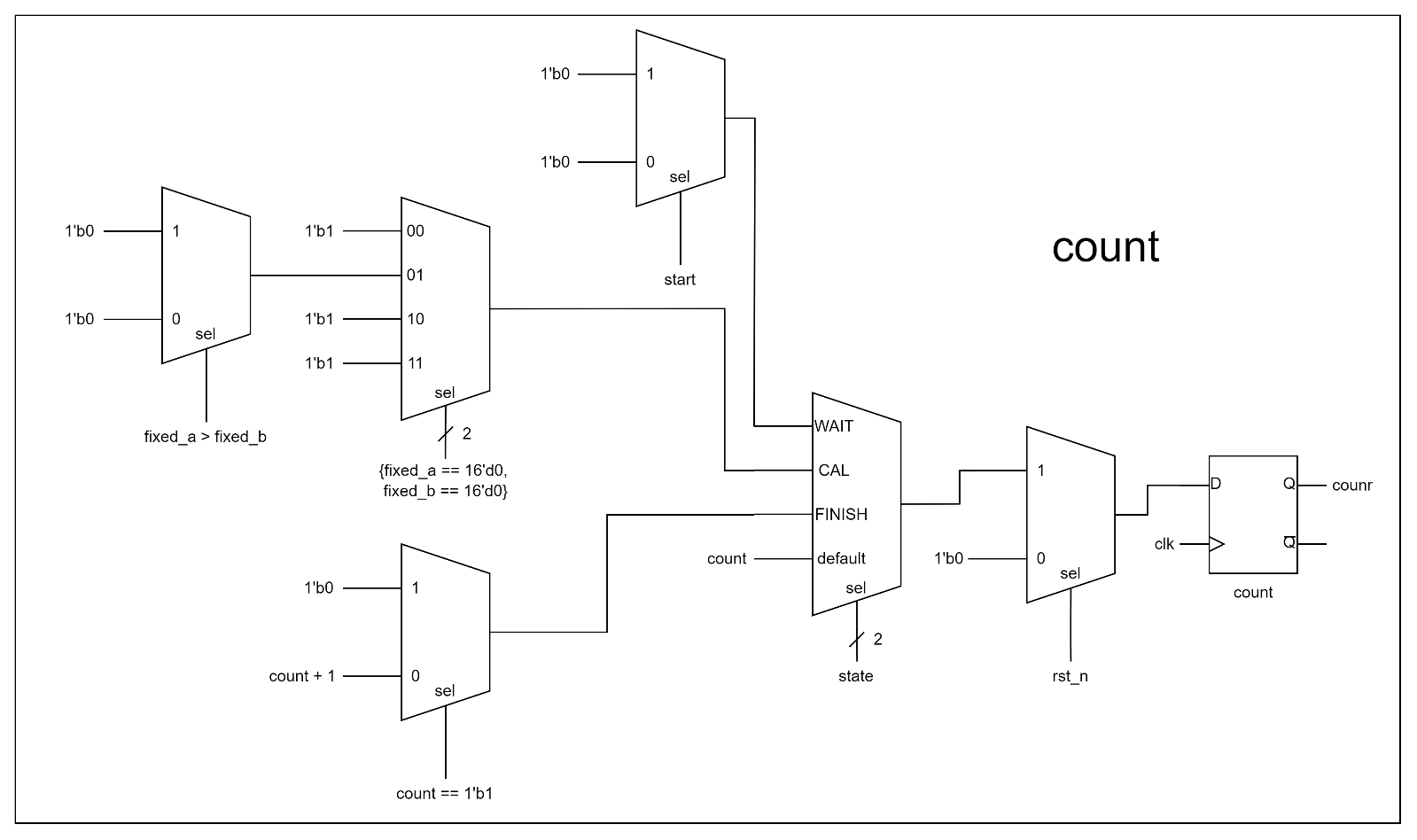


Figure 4.3

As **Figure 4.3** shows, I test every possible case to check if my design is correct or not. And for the overflow problem mentioned in **Explanation** above is prevented. We can see that while -8 is the multiplicand, the results are correct.

1. **What I Have Learned**

****In this lab I've found that after I realized the circuits according to the finite state diagram, there may be some redundant multiplexers in my circuits, such as whatever the selection signal is 1 or 0, the output is the same. After drawing out the block diagram, I can clearly see that which mux is redundant and can be deleted. However, close to the end of the semester, there are a lot of projects that I need to do, which steals my time away. Therefore I didn't have time to simplify my code to reduce redundant mux.

For example, in **Figure 3.6**, these two mux can be replaced by a wire.