Lab 2 Exercise 5: Fibonacci function

姓名：王宇　　　**学号：**12112725

I. Design procedure

1. Defining the input and output signals

Input signals:

- n_in: input operands. 6-bit signals with std_logic_vector data type and interpreted as unsigned integers.

- start: command. The fibonacci function starts operation when the start signal is activated.

- clk: system clock;

- reset: asynchronous reset signal for system initialization.

Output signals

- r_out: the final result. 43-bit signals.

- ready: external status signal. It is asserted when the fibonacci circuit is idle and ready to accept new inputs.

2. Converting the algorithm to an ASM chart

Before we start to draw the ASM chart, we should think about the algorithm of fibonacci function. There are two main algorithm: recursion and loop.

For recursion, it's easy to understand. It just need to keep calling its own function. In software, recursion is easy to realize. But in hardware, especially in FPGA board, it realize logic functions through hardware circuits. So it's hard to realize recursion in FPGA, and easy to realize loop in FPGA. Besides, the time complexity of recursion is $\mathbf{O(N^2)}$. And the complexity of loop is **O(N)**. Therefore, we should use **loop algorithm**.

The ASM chart is as follows:

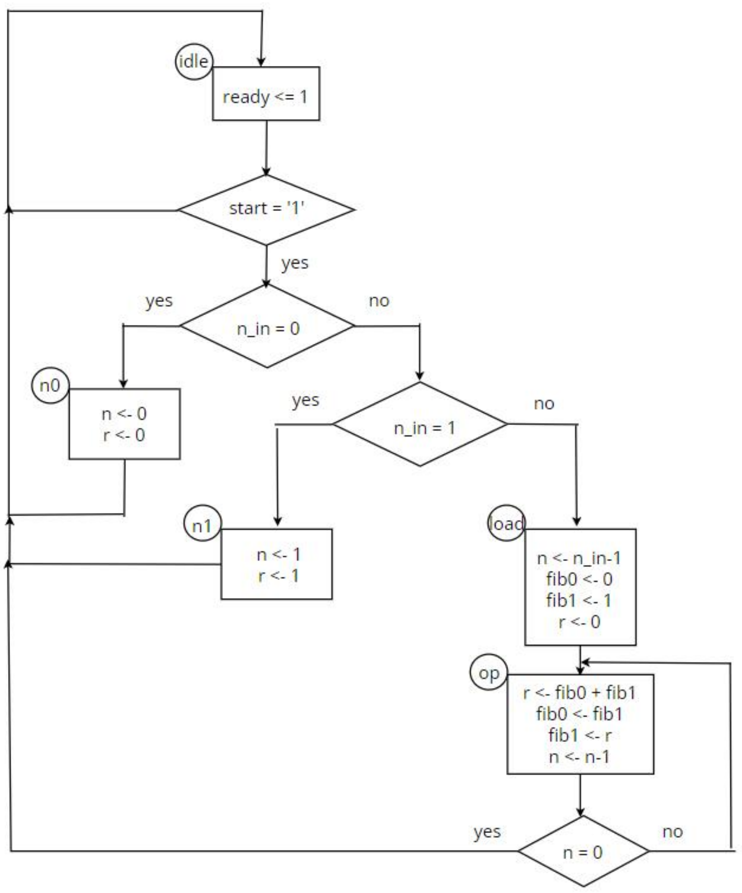


figure 1: ASM chart

3. Constructing the FSMD

3.1 List all possible RT operations in the ASM chart

The circuit require 4 registers, to store signals r, n, fib0, fib1 respectively. Besides, it need a state register.

3.2 Group RT operations according to their destination registers

- RT operation with **r register:**
  - r <- r (in the idle state)
  - r <- 0 (in the load and n0 state)
  - r <- 1 (in the n1 state)
  - r <- fib0 + fib1 (in the op state)
- RT operation with **n register:**
  - n <- n (in the idle state)
  - n <- 0 (in the n0 state)
  - n <- 1 (in the n1 state)
  - n <- n_in - 1 (in the load state)
  - n <- n-1 (in the op state)
- RT operation with **fib0 register:**
  - fib0 <- fib0 (in the idle, n0 and n1 state)
  - fib0 <- 0 (in the load state)

- fib0 <- fib1 (in the op state)

- RT operation with **fib1 register:**
- fib1 <- fib1 (in the idle, n0 and n1 state)
- fib1 <- 1 (in the load state)
- fib1 <- r (in the op state)

3.3 Derive the circuit for each group RT operation

- The conceptual diagram of the circuit associated with the **r register**:

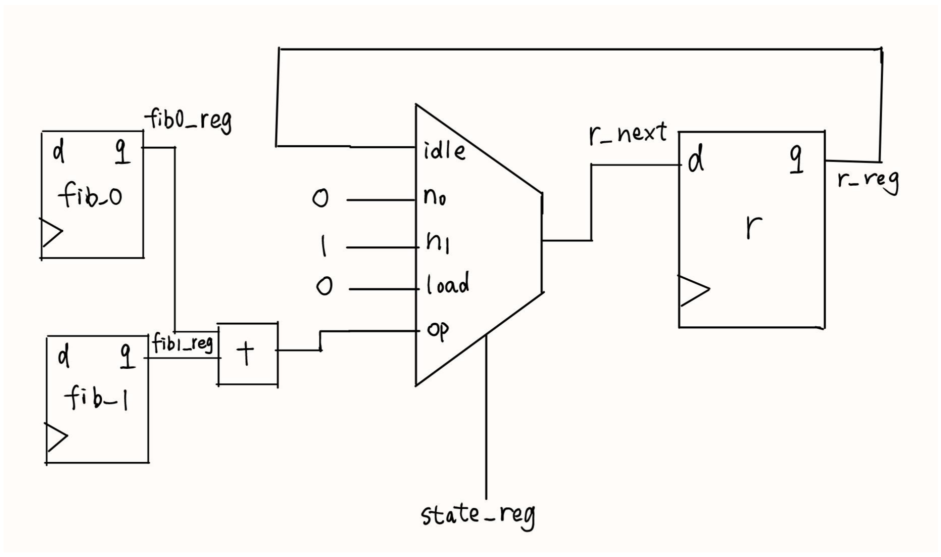


figure 2: conceptual diagram of r register

- The conceptual diagram of the circuit associated with the **n register**:

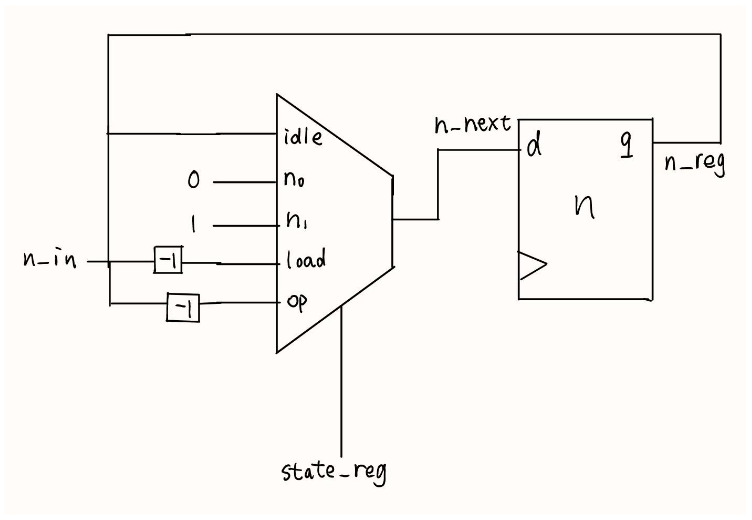


figure 3: conceptual diagram of n register

- The conceptual diagram of the circuit associated with the **fib0 register**:

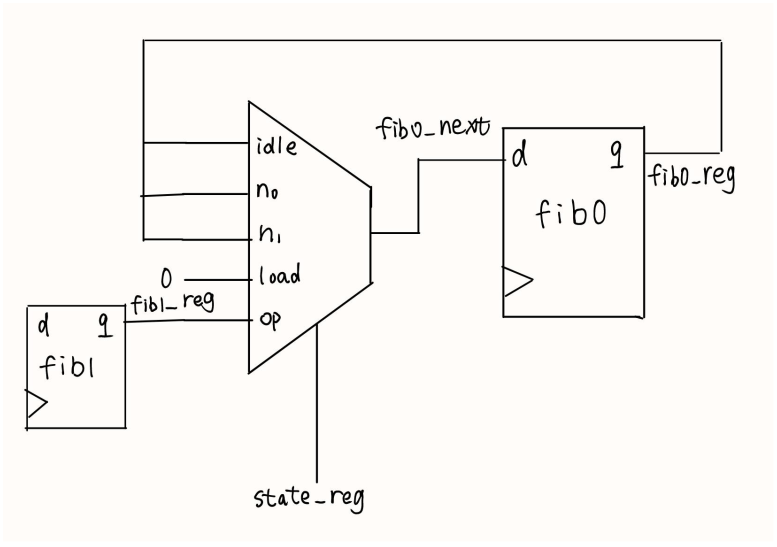


figure 4: conceptual diagram of fib0 register

- The conceptual diagram of the circuit associated with the **fib1 register**:

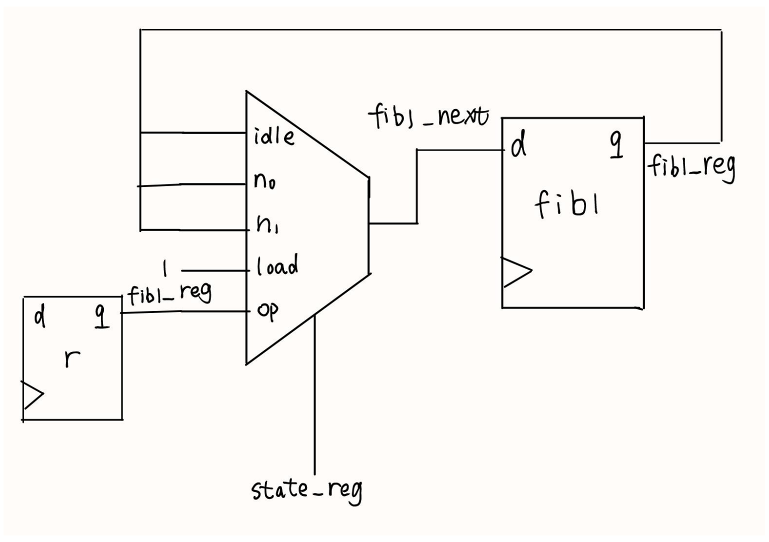


figure 5: conceptual diagram of fib1 register

3.4 Complete block diagram of fibonacci function

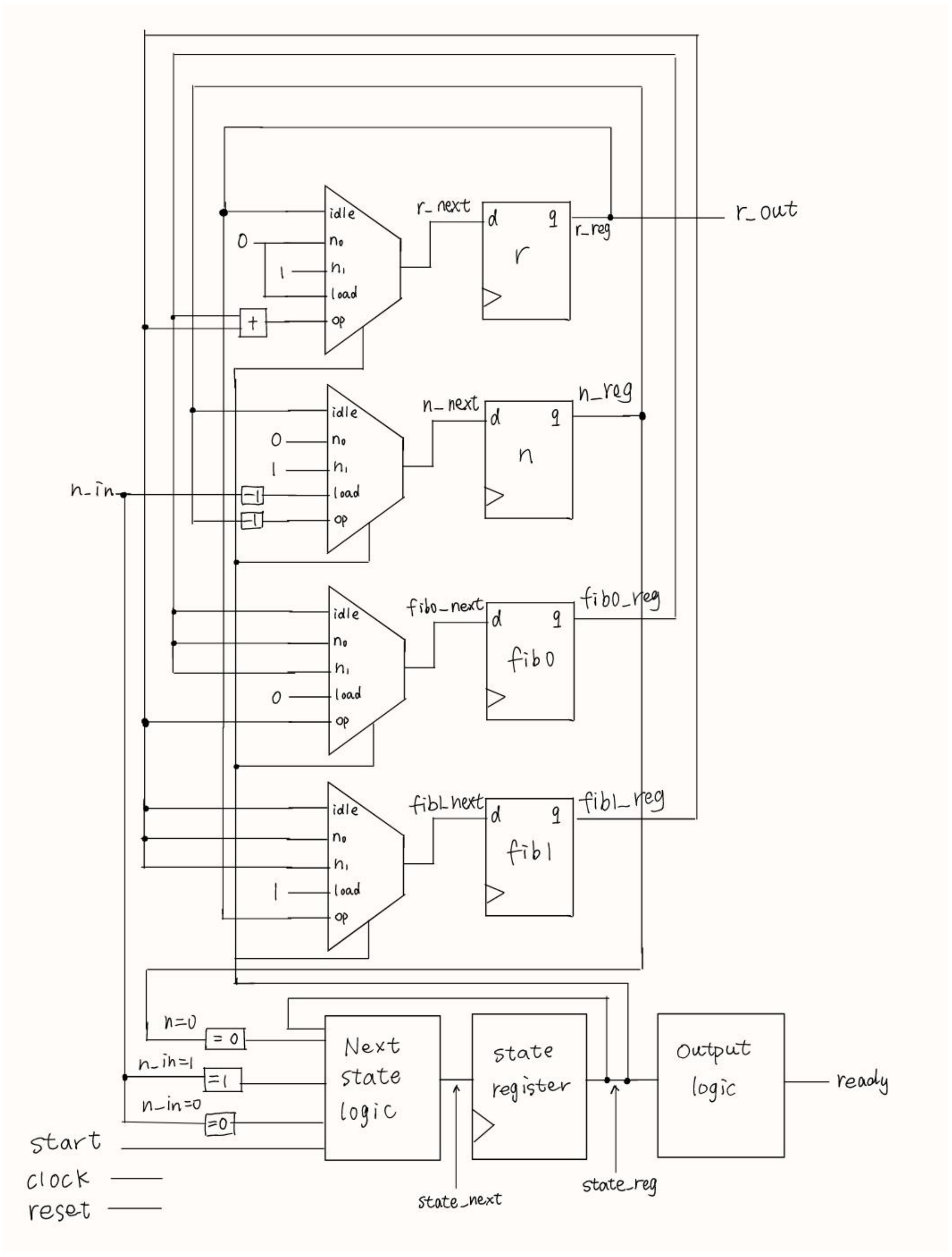


figure 6: complete block diagram

4. VHDL descriptions of FSMD

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

entity FSMD is
    Port (
        CLK : in STD_LOGIC;
        RST : in STD_LOGIC;
        start : in STD_LOGIC;
        n_in : in STD_LOGIC_VECTOR(5 downto 0);
        r : out STD_LOGIC_VECTOR(42 downto 0);
        ready : out STD_LOGIC
    );
end FSMD;

architecture Behavioral of FSMD is
    type state_type is (idle, n0, n1, load, op);
    signal state_reg, state_next : state_type;
    signal r_reg, r_next : STD_LOGIC_VECTOR(42 downto 0);
    signal n_reg, n_next : STD_LOGIC_VECTOR(5 downto 0);
    signal fib0_reg, fib0_next, fib1_reg, fib1_next : STD_LOGIC_VECTOR(42
downto 0);

begin
    -- state and data registers
    process(CLK, RST)
    begin
        if RST = '1' then
            state_reg <= idle;
            r_reg <= (others => '0');
            n_reg <= (others => '0');
            fib0_reg <= (others => '0');
            fib1_reg <= (others => '0');
        elsif CLK' event and CLK='1' then
            state_reg <= state_next;
            r_reg <= r_next;
            n_reg <= n_next;
            fib0_reg <= fib0_next;
            fib1_reg <= fib1_next;
        end if;
    end process;
```

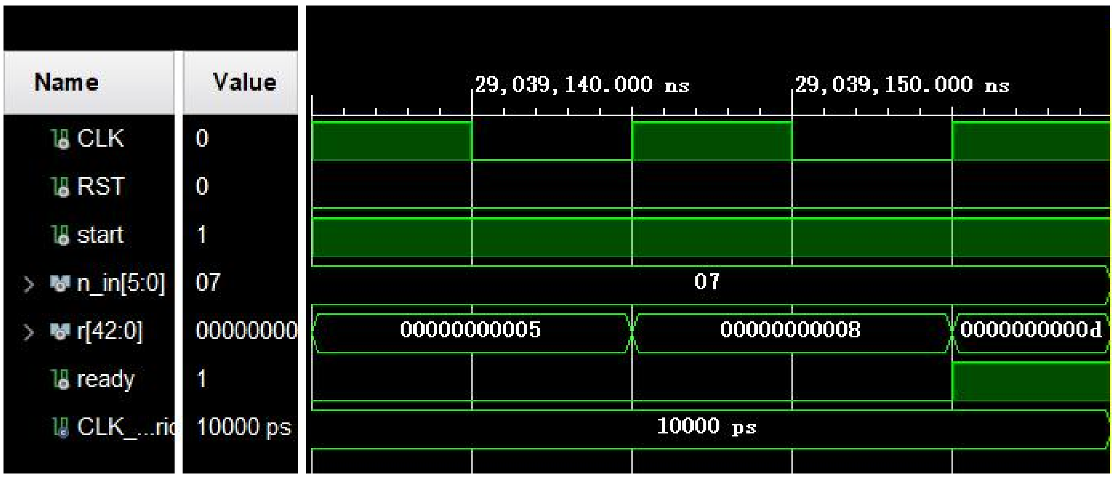
```
	-- combinational circut
	process(state_reg, start, n_in, r_reg, n_reg, fib0_reg, fib1_reg)
	begin
		-- default value
		r_next <= r_reg;
		n_next <= n_reg;
		fib0_next <= fib0_reg;
		fib1_next <= fib1_reg;
		ready <= '0';

		case state_reg is
			when idle =>
				if start = '1' then
					if n_in = "000000" then
						state_next <= n0;
					else
						if n_in = "000001" then
							state_next <= n1;
						else
							state_next <= load;
						end if;
					end if;
				else
					state_next <= idle;
				end if;
				ready <= '1';

			when n0 =>
				n_next <= "000000";
				r_next                                                              <=
"0000000000000000000000000000000000000000000";

			when n1 =>
				n_next <= "000001";
				r_next                                                              <=
"0000000000000000000000000000000000000000001";

			when load =>
				n_next <= n_in - 1;
				fib0_next                                                           <=
"0000000000000000000000000000000000000000000";
				fib1_next                                                           <=
"0000000000000000000000000000000000000000001";
				r_next                                                              <=
"0000000000000000000000000000000000000000000";
                            state_next <= op;

                    when op =>
                            r_next <= fib0_reg + fib1_reg;
                            fib0_next <= fib1_reg;
                            fib1_next <= fib0_reg + fib1_reg;
                            n_next <= n_reg - 1;
                            if n_reg = "000001" then
                                    state_next <= idle;
                            else
                                    state_next <= op;
                            end if;
                end case;
            end process;
            r <= r_reg;
        end Behavioral;
```

II. Test result & timing analysis

1. Test result

In the testbench, I assume n=7, and run the simulation. From the figure, we can see the final result is equal to 13, consistent with the correct answer.

The figure of 5 simulations are as follows:



```
Name        Value        29,039,140.000 ns          29,039,150.000 ns
CLK         0
RST         0
start       1
n_in[5:0]   07           07
r[42:0]     00000000     00000000005   00000000008   0000000000d
ready       1
CLK_...ri   10000 ps     10000 ps
```

figure 7: behavioral simulation result

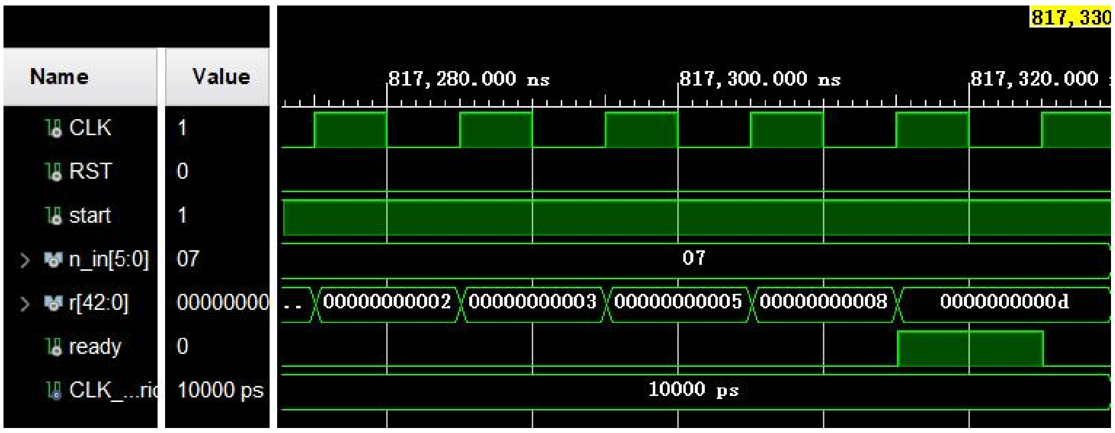


figure 8: Post-synthesis Functional Simulation

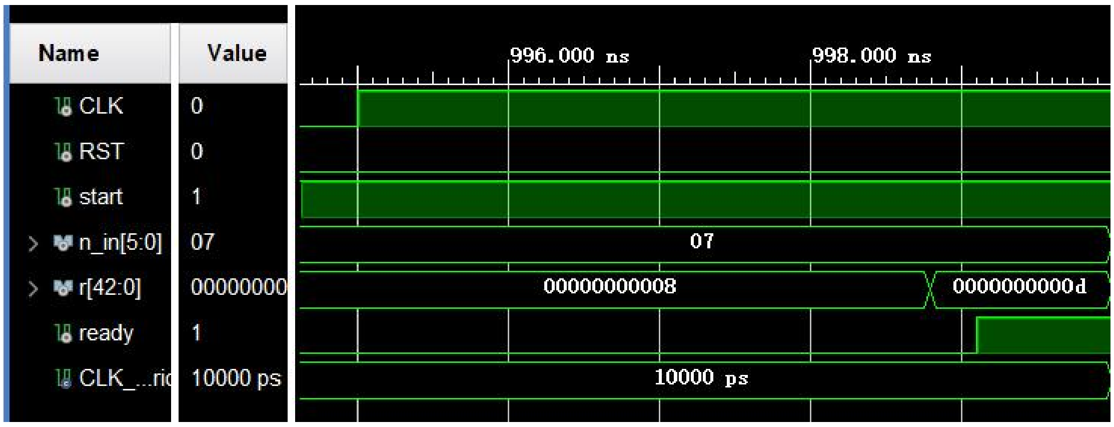


figure 9: Post-Synthesis Timing Simulation

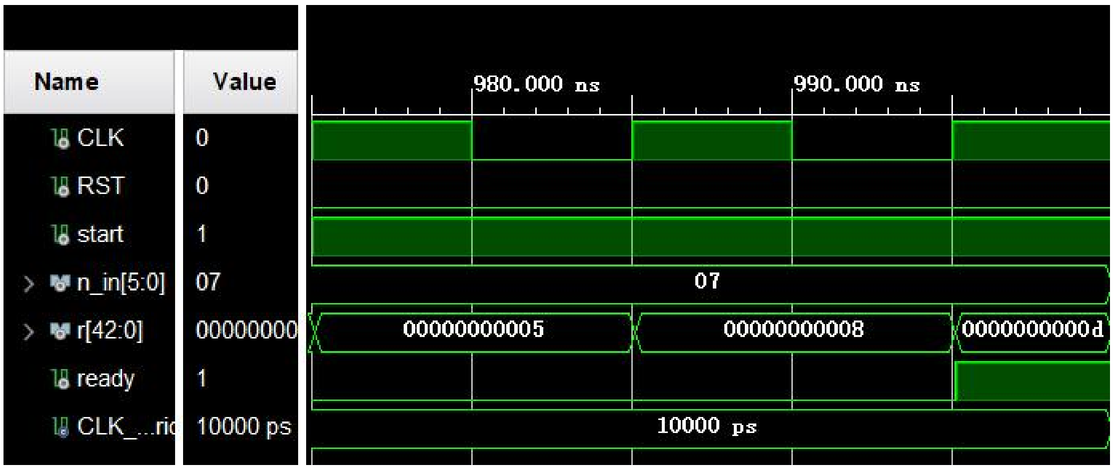


figure 10: Post-Implementation Functional Simulation

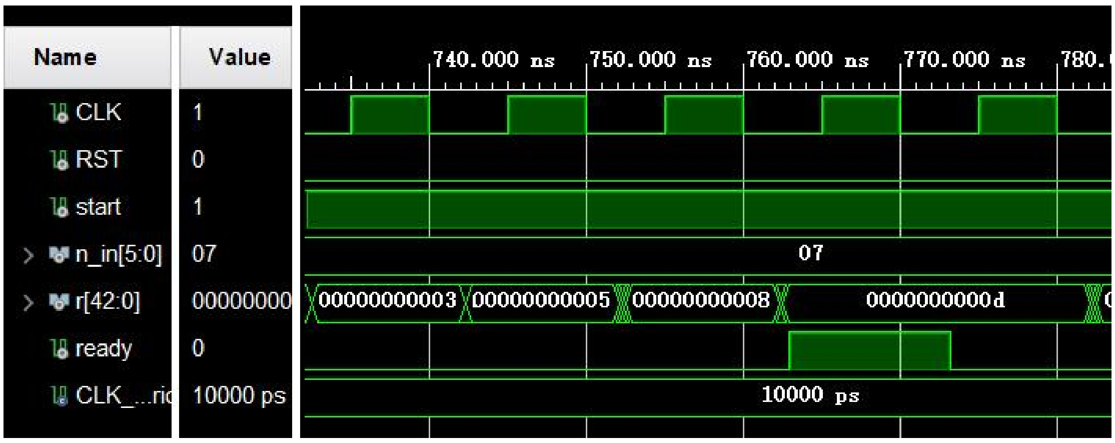
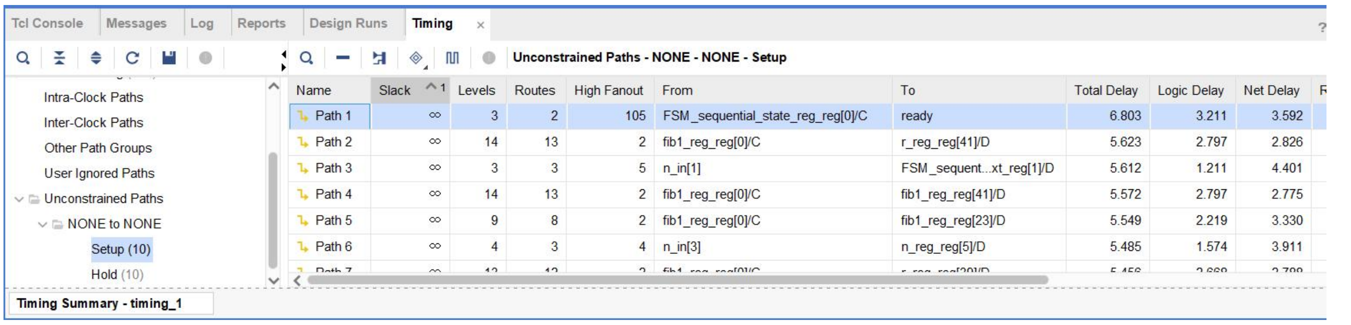


figure 11: Post-Implementation Timing Simulation

2. Timing analysis

Next, let’s analyze the timing of the circuit. According to the “Report Timing Summary”, we can find the critical path and max delay.



Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	3	2	105	FSM_sequential_state_reg_reg[0]/C	ready	6.803	3.211	3.592
Path 2	∞	14	13	2	fib1_reg_reg[0]/C	r_reg_reg[41]/D	5.623	2.797	2.826
Path 3	∞	3	3	5	n_in[1]	FSM_sequent...xt_reg[1]/D	5.612	1.211	4.401
Path 4	∞	14	13	2	fib1_reg_reg[0]/C	fib1_reg_reg[41]/D	5.572	2.797	2.775
Path 5	∞	9	8	2	fib1_reg_reg[0]/C	fib1_reg_reg[23]/D	5.549	2.219	3.330
Path 6	∞	4	3	4	n_in[3]	n_reg_reg[5]/D	5.485	1.574	3.911
Path 7	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

figure 12: critical path

In device, the path is as follows:

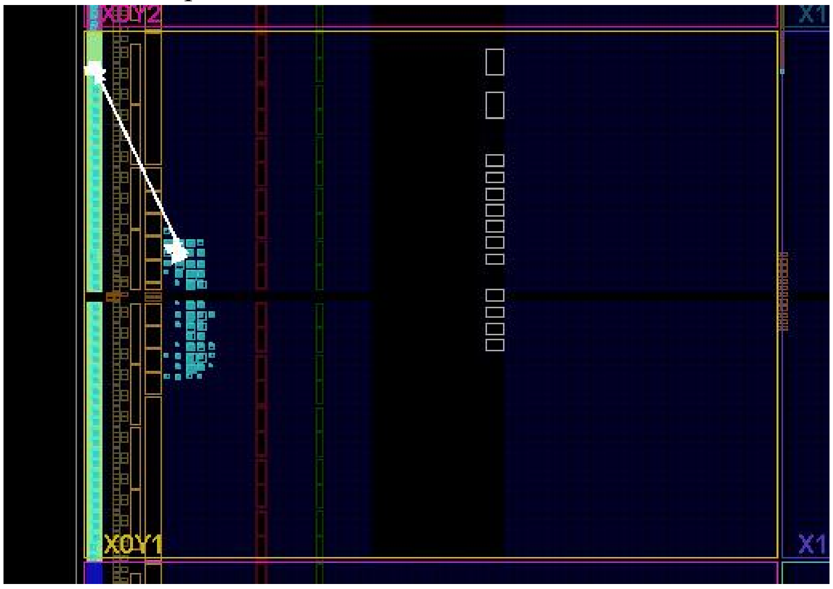


figure 13: critical path in device

In schematic, the path is as follows:

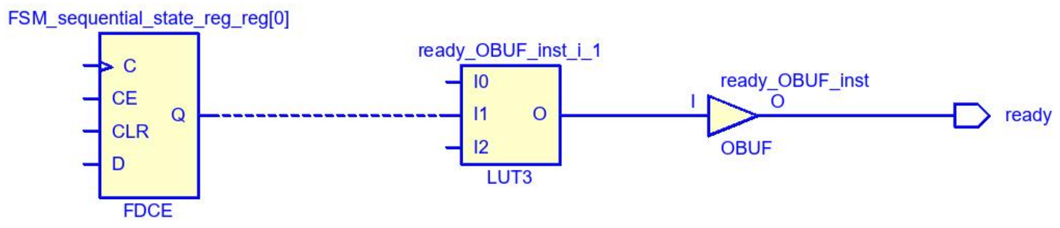


figure 14: critical path in schematic

From the figure, we can find the state register cost the most time. The reason may be that each step need to determine the current state in order to take action, and many times it needs to convey next state to state register.

III. I/O interface

Input signals:

- n_in: connect to switches (V10, U11, U12, H6, T13, R16);
- start: default value is 1;
- clk: connect to system clock E3;
- reset: connect to button C12.

Output signals

- r_out: I can't find a proper way to show for now. (It's too large)

IV. Conclusion

From the exercise, I learned how to design a FSMD. The whole procedure has 4 step.

1. We should defining the input and output signals, and determine the data type and bits of them.
2. We should determine the algorithm, and convert it to an ASM chart. We need to mark every state and every action.
3. We should constructing the FSMD. List all the required register, then list and draw the RT operations for every register, finally combined them to a complete block diagram.
4. We can code VHDL easily.