**Lab 2 Exercise 5: Fibonacci function**

**姓名**：王宇 **学号**：12112725

1. **Design procedure**
2. **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.

1. **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. And the complexity of loop is **O(N)**. Therefore, we should use **loop algorithm**.

The ASM chart is as follows:

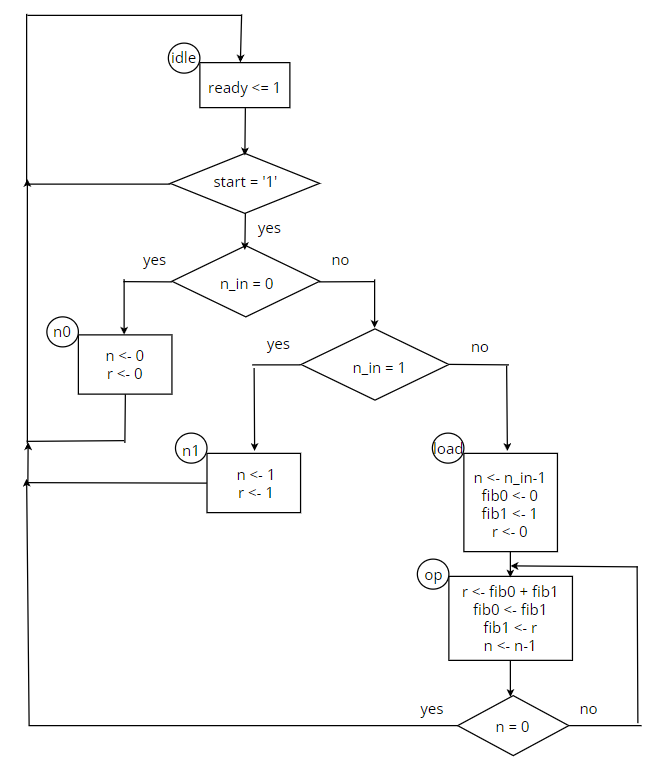


figure 1: ASM chart

1. **Constructing the FSMD**
   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.

* 1. **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**:

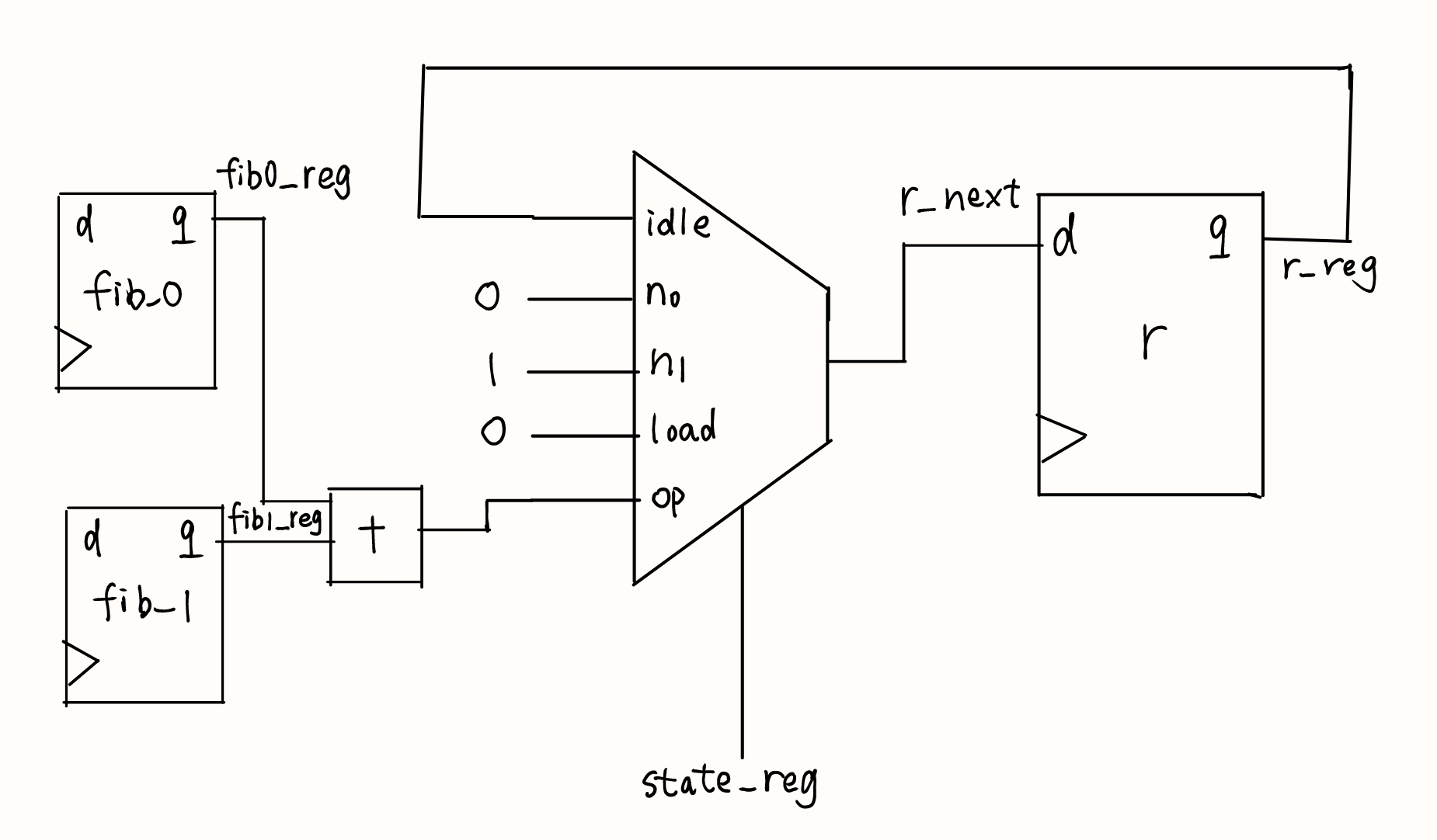


figure 2: conceptual diagram of r register

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

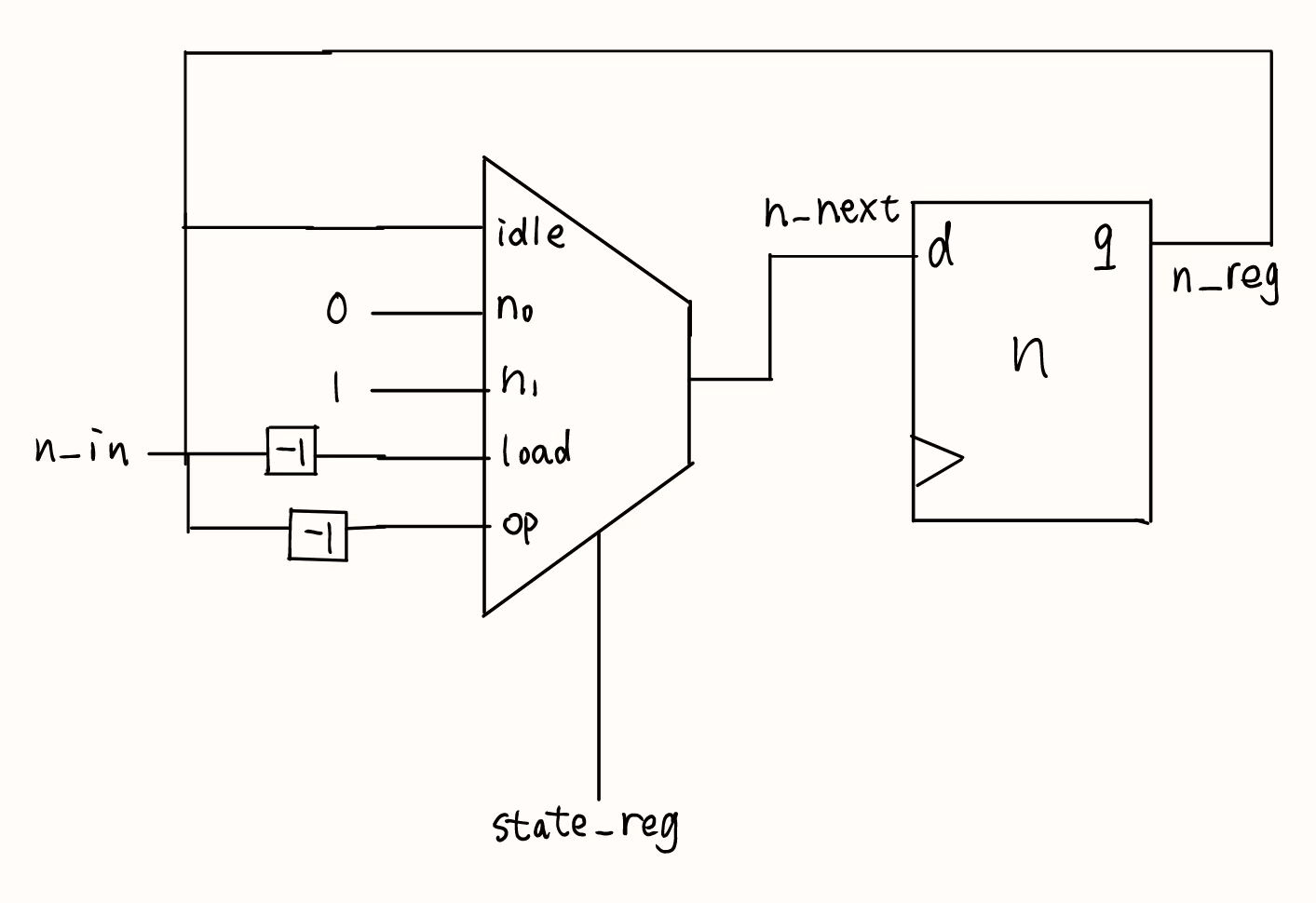


figure 3: conceptual diagram of n register

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

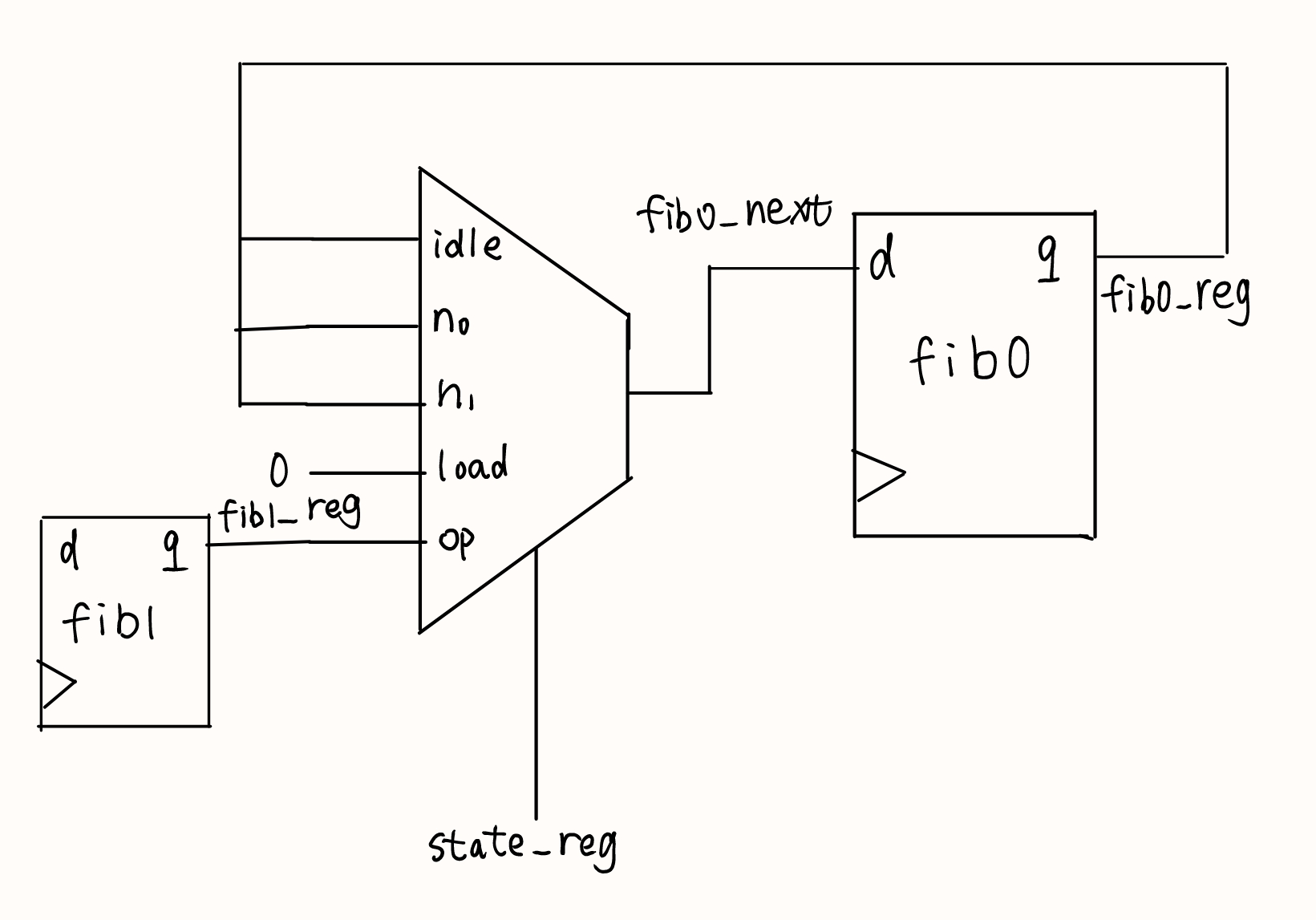


figure 4: conceptual diagram of fib0 register

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

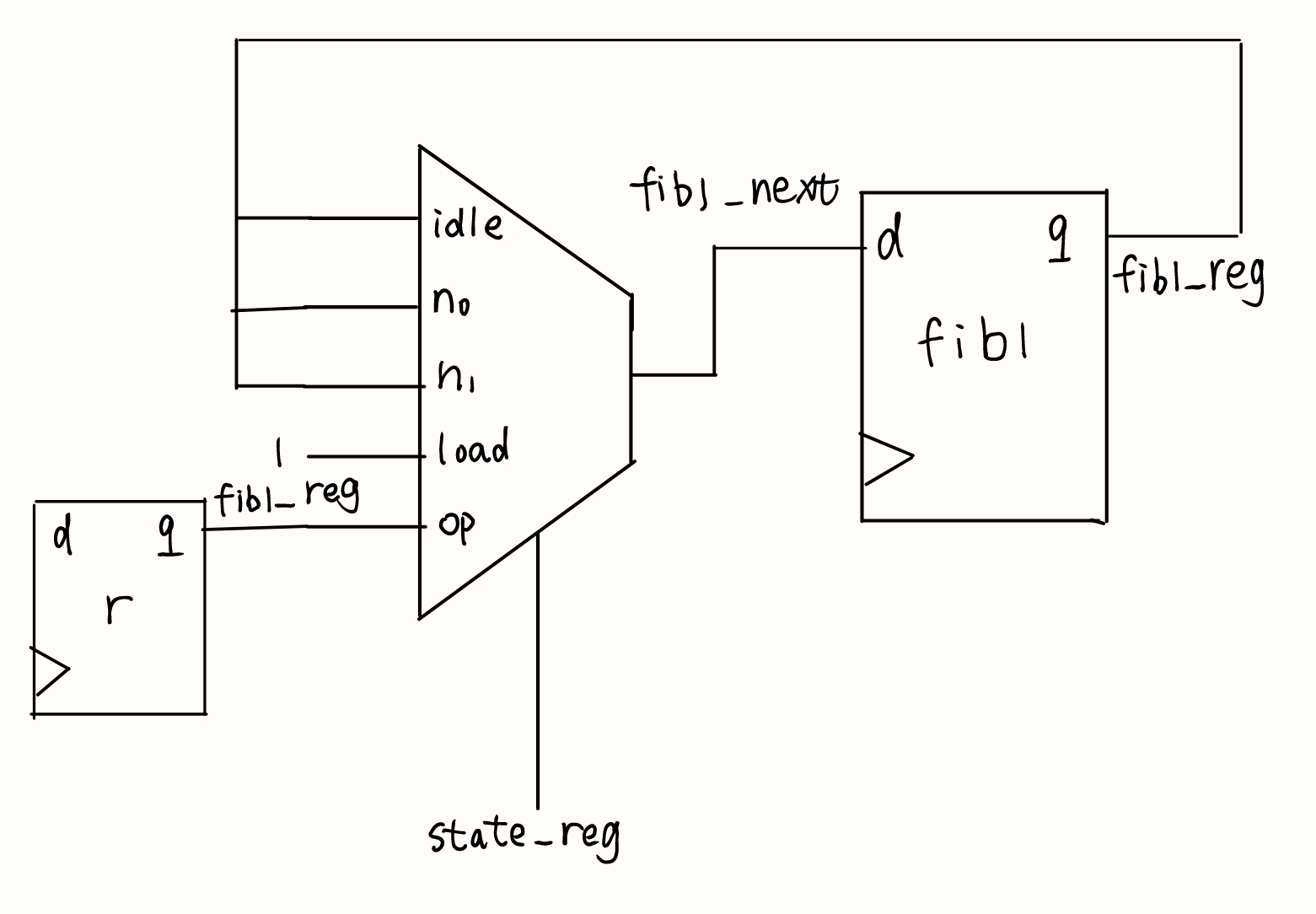


figure 5: conceptual diagram of fib1 register

**3.4 Complete block diagram of fibonacci function**

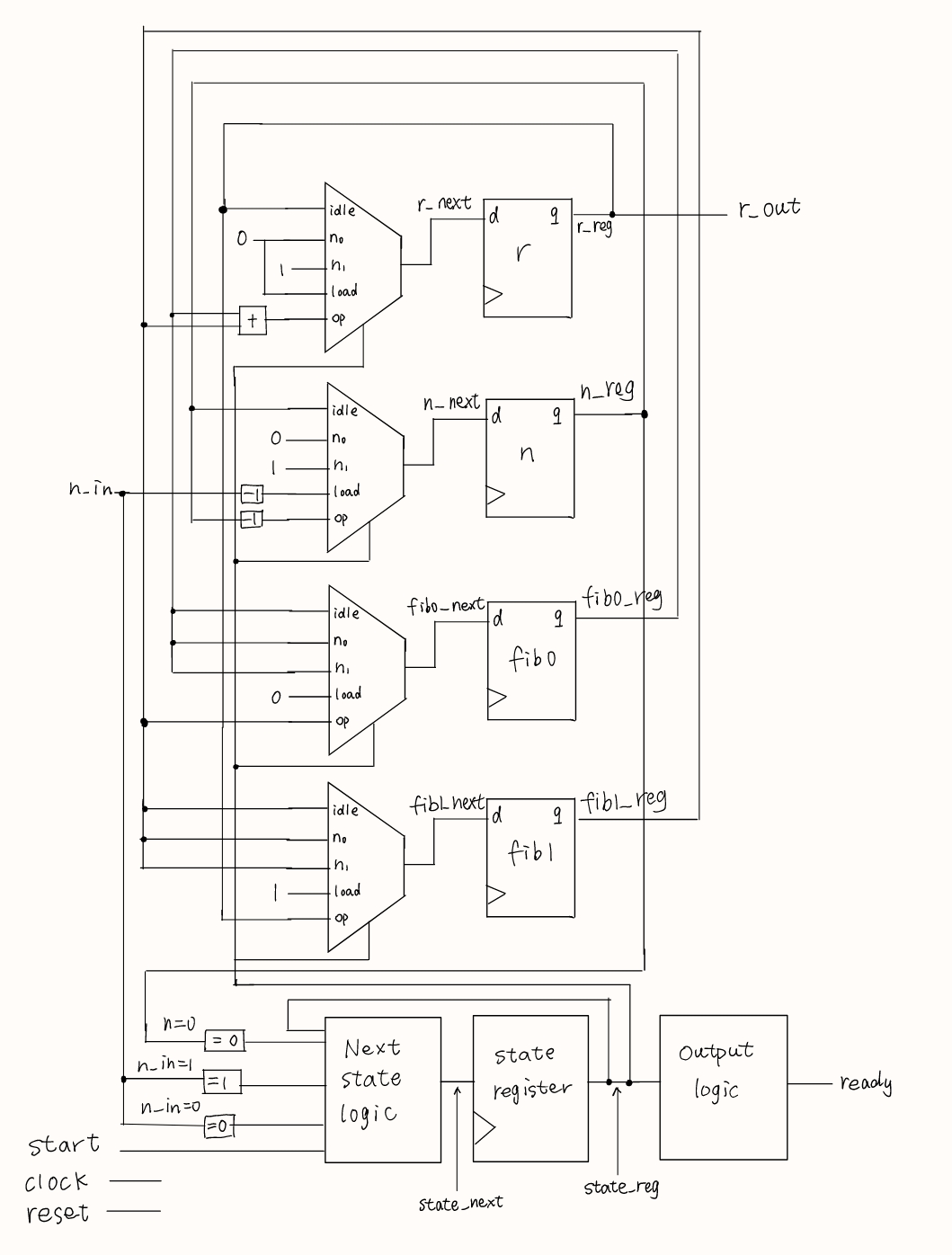
****

figure 6: complete block diagram

1. **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;

1. **Test result & timing analysis**
2. **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:

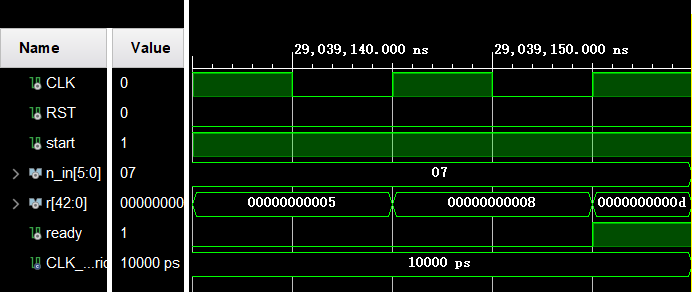


figure 7: behavioral simulation result

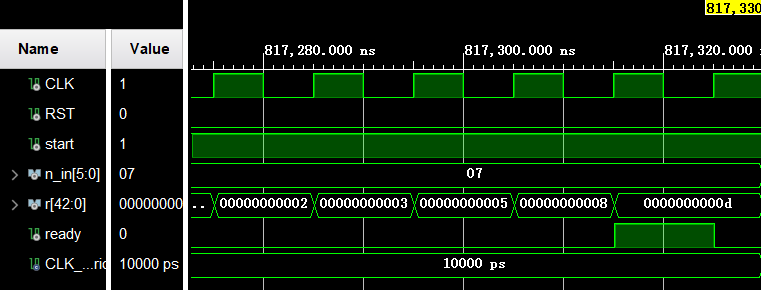


figure 8: Post-synthesis Functional Simulation

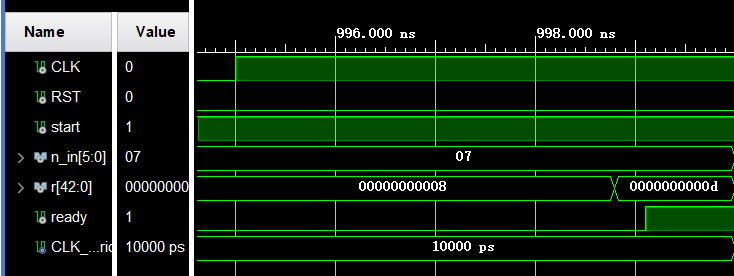


figure 9: Post-Synthesis Timing Simulation

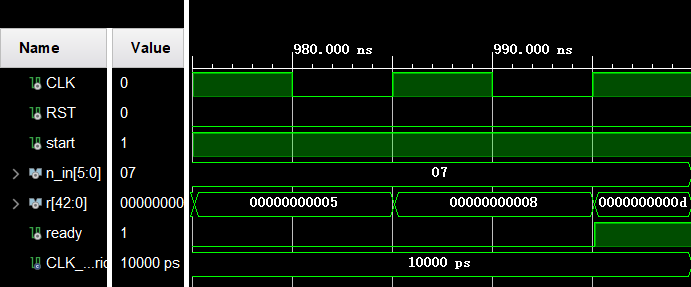


figure 10: Post-Implementation Functional Simulation

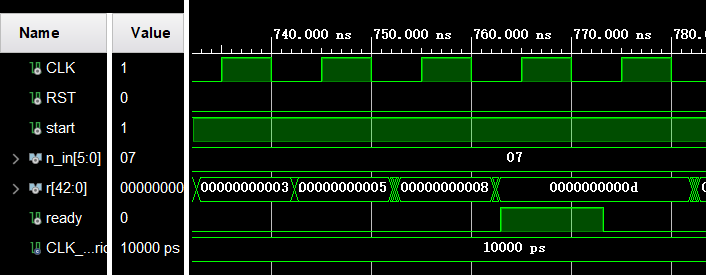


figure 11: Post-Implementation Timing Simulation

1. **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.

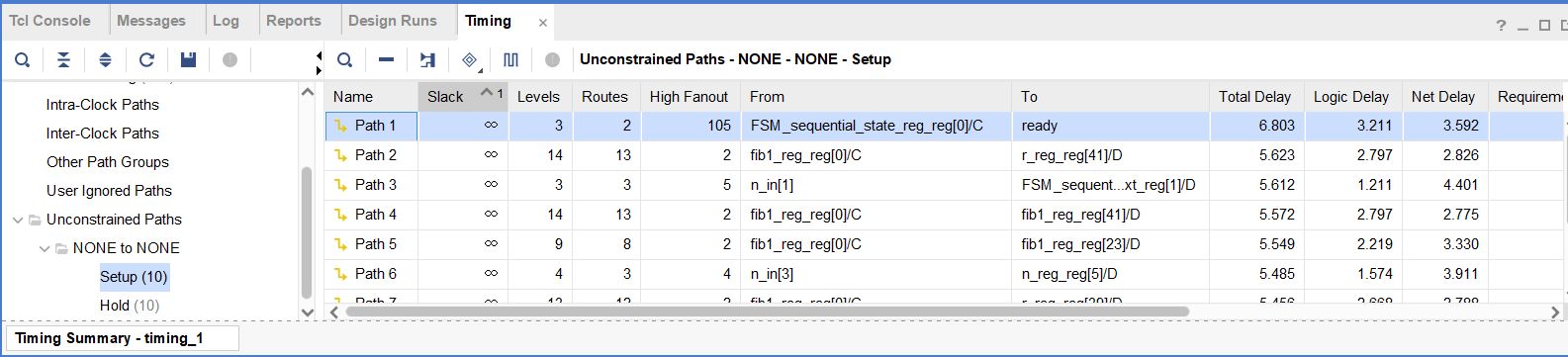


figure 12: critical path

In device, the path is as follows:

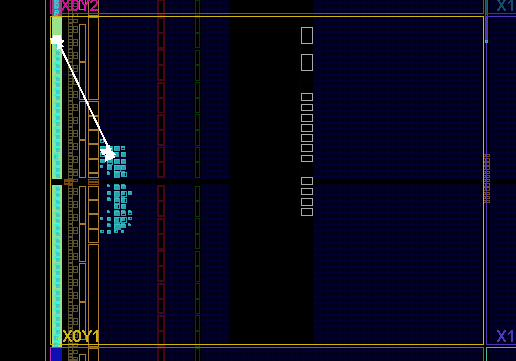


figure 13: critical path in device

In schematic, the path is as follows:

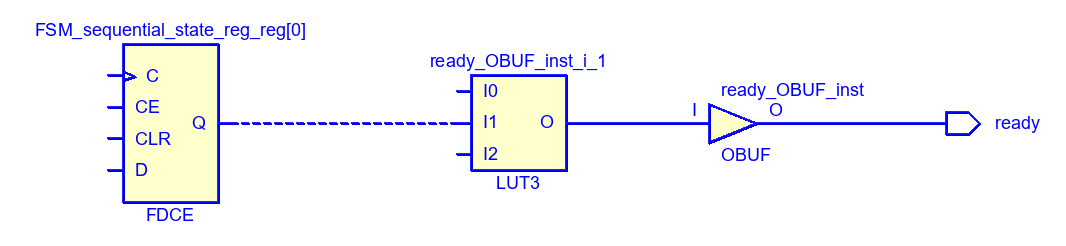


figure 14: critical path in schematic

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.

1. **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)

1. **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.