Fibonacci function

11911214 杨鸿嘉

pseudo-code:

```
        n1 = 0;
        n2 = 1;
        if(n=0) then{
out1:       result = n1;
          }
        elsif(n=1) then{
out2:     result = n2;
          }
        else{
op:         temp = n2;
            n2 = n2 + n1;
            n1 = temp;
            n = n-1;
            if(n=1) then {goto out2;}
                  else {goto op;}
          }
```

Step 1: Defining the input and output signals

Input signals:

Clk: system clock

Reset: asynchronous reset signal for system initialization

Start: command to starts the calculation of Fibonacci value

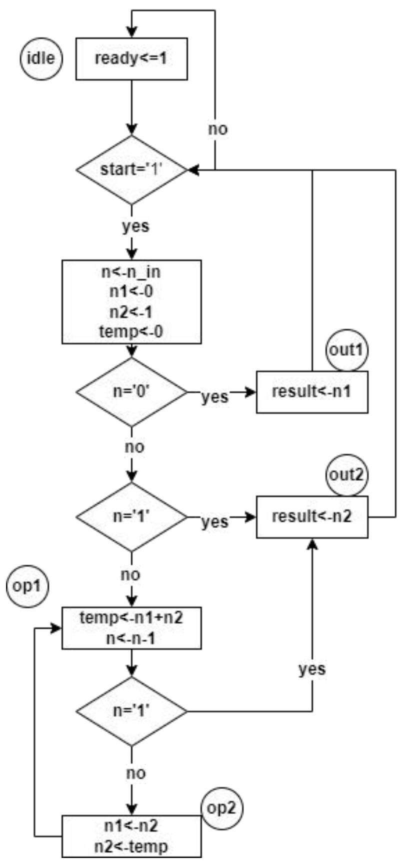
n: 6-bit signal of std_logic_vector for the independent variable of Fibonacci function

Output signals

Output: 43-bit signal of std_logic_vector for the function value of Fibonacci with input n

Ready: external status signal and it is asserted after output and when the state return to idle

Step 2: Converting the algorithm to an ASM chart



Step 3: Constructing the FSMD

3.1 The circuit require 5 registers, to store signals n1, n2, temp,n and result respectively

3.2

RT operation with the n1 register:

n1<-0 (in the idle state)

n1<-n2 (in the op2 state)

RT operation with the n2 register:

n2<-1(in the idle state)

n2<-temp (in the op2 state)

RT operation with the temp register:

temp<-0(in the idle state)

temp<-n1+n2(in the op1 state)

RT operation with the n register:

n<-n_in (in the idle state)

n<-n-1 (in the op1 state)

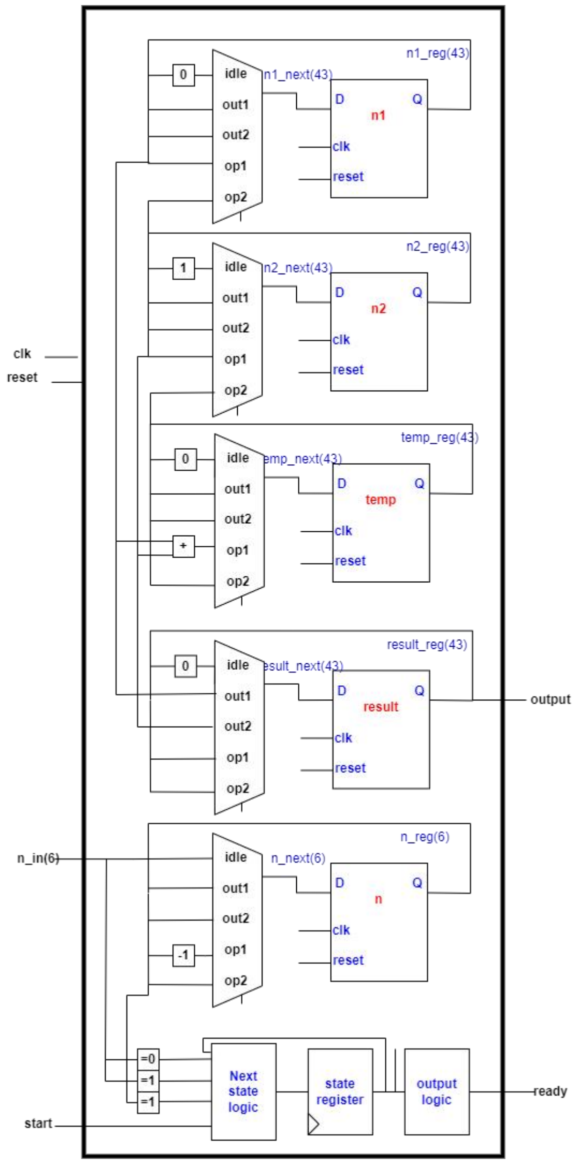
RT operation with the result register:

result<-0 (in the idle state)

result<-n1 (in the out1 state)

result<-n2 (in the out2 state)

3.3 Complete block diagram of Fibonacci function



Step 4: VHDL descriptions of Fibonacci function

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Fibonacci is
port (CLK, RESET, start: in std_logic;
        n: in std_logic_vector(5 downto 0);
       ready: out std_logic;
        output: out std_logic_vector(42 downto 0));
end Fibonacci;

architecture Behavioral of Fibonacci is
type state_type is (idle,out1,out2,op1,op2);
signal state_reg, state_next : state_type;
signal n1_reg,n1_next,n2_reg,n2_next,result_reg,result_next,
          temp_reg,temp_next: std_logic_vector (42 downto 0);
signal n_reg,n_next:std_logic_vector (5 downto 0);

begin
process (CLK, RESET) is
   begin
   if RESET = '1' then
      state_reg<=idle;
      n1_reg<=(others=>'0');
      n2_reg<=(0=>'1',others=>'0');
      result_reg<=(others=>'0');
      n_reg<=(others=>'0');
      temp_reg<=(others=>'0');
   elsif CLK'event and CLK='1' then
      state_reg<=state_next;
      n1_reg<=n1_next;
      n2_reg<=n2_next;
      n_reg<=n_next;
      result_reg<=result_next;
      temp_reg<=temp_next;
   end if;
end process;
```

```vhdl
process (start, state_reg, n1_reg,n2_reg,result_reg,n_reg,temp_reg) is
variable temp:std_logic_vector (42 downto 0);
   begin
   n1_next<=n1_reg;
   n2_next<=n2_reg;
   result_next<=result_reg;
   n_next<=n_reg;
   ready<='0';
   temp_next<=temp_reg;
   case state_reg is
      when idle=>
         if start = '1' then
            if (n="000000") then
               state_next<=out1;
            elsif (n="000001") then
               state_next<=out2;
            else
               state_next<=op1;
            end if;
         else
            state_next<=idle;
         end if;
         n_next<=n;
        -- ready<='1';
      when out1=>
         result_next<=n1_reg;
         state_next<=idle;
         ready<='1';
      when out2=>
         result_next<=n2_reg; --1
         state_next<=idle;
         ready<='1';
      when op1=>
         temp_next<=n1_reg+n2_reg;
         n_next<=n_reg-1;
         if (n_reg="000001") then
         state_next <= out2;
         else
         state_next <= op2;
         end if ;
     when op2=>
         n1_next<=n2_reg;
         n2_next<=temp_reg;
         state_next<=op1;
   end case;
end process;

output<=result_reg;

end Behavioral;
```

Testbench code

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Fibonacci_tb is
end Fibonacci_tb;

architecture Behavioral of Fibonacci_tb is
component Fibonacci
port(CLK, RESET, start: in std_logic;
          n: in std_logic_vector(5 downto 0);
             ready: out std_logic;
          output: out std_logic_vector(42 downto 0));
end component;

signal CLK_tb, RESET_tb, start_tb,ready_tb: std_logic;
signal n_tb: std_logic_vector(5 downto 0);
signal output_tb: std_logic_vector(42 downto 0);
begin
uut: Fibonacci port map(CLK=> CLK_tb,RESET=>RESET_tb,start=>start_tb,
                        n=>n_tb,ready=>ready_tb,output=>output_tb);
```

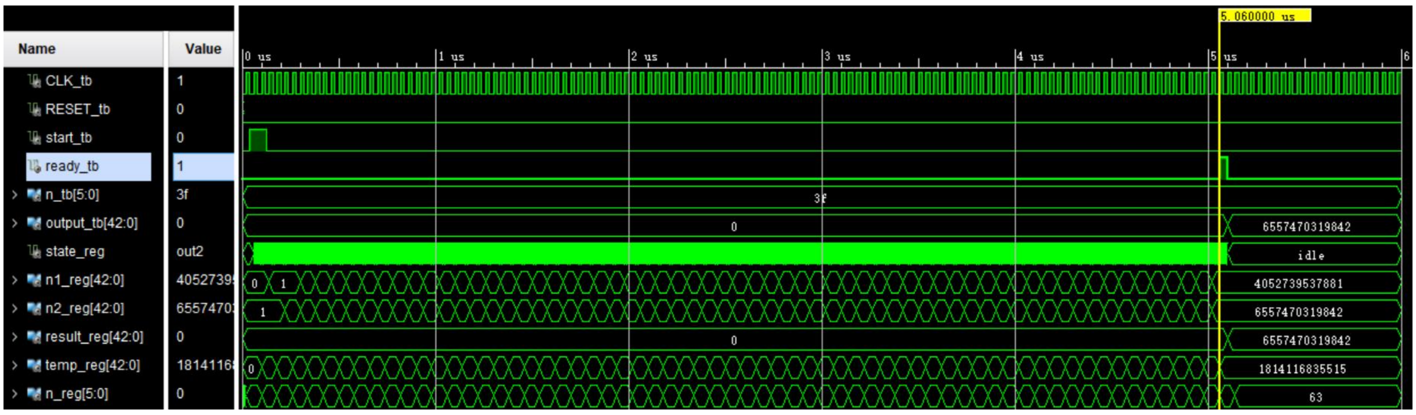
```vhdl
clock_gen: process
constant period : time :=50 ns;
begin
CLK_tb<= '0';
wait for period/2;
CLK_tb<='1';
wait for period/2;
end process;

RESET_tb<='1','0' after 10 ns;

n_tb<="111111";
start_tb<='0','1'after 50 ns,'0' after 160ns;
end Behavioral;
```

Step5 Simulation result

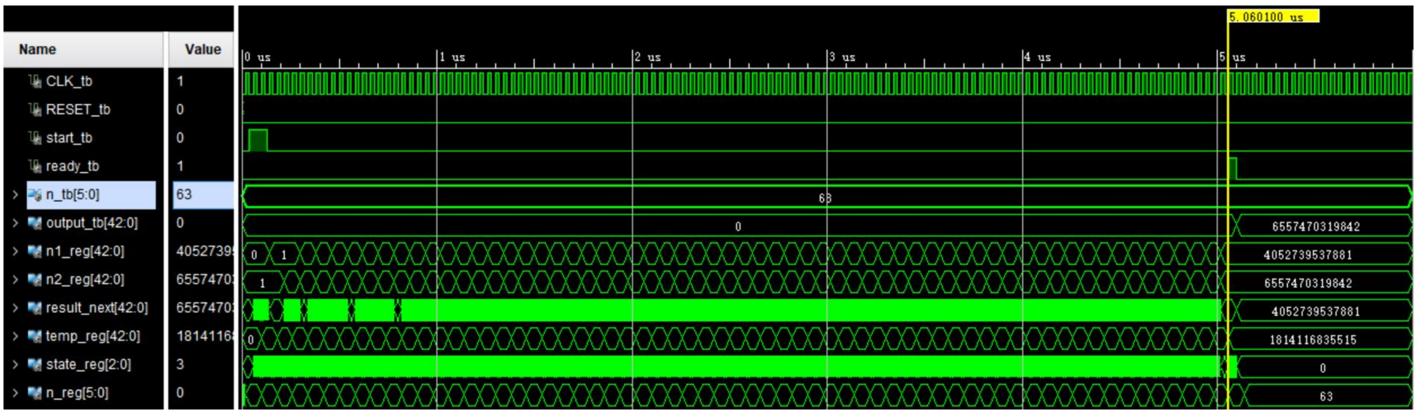
Behavior result:

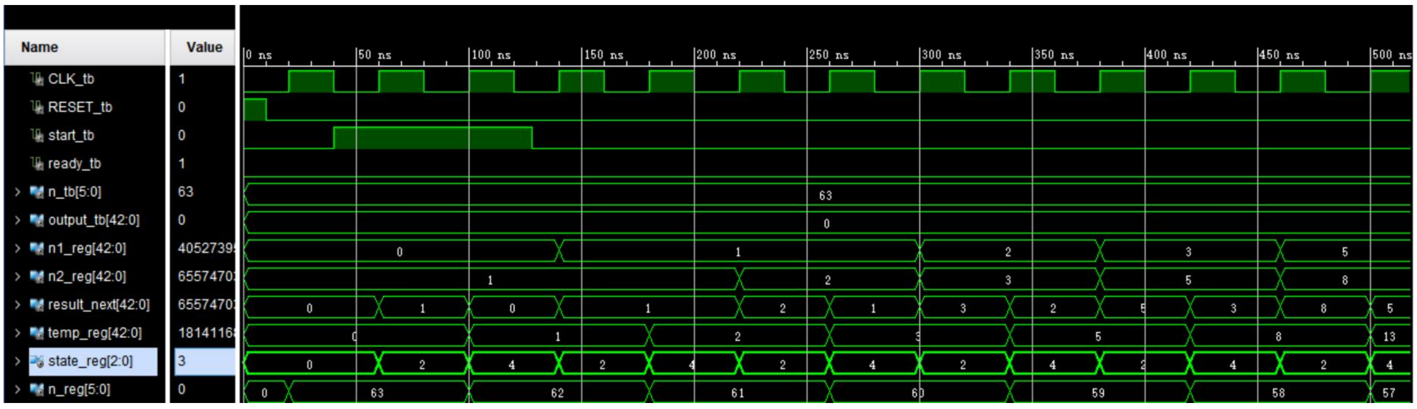


Behavior simulation

For the given input n=63, output is 6557470319842, which is correct. For the clk period of 40ns, that is, 25MHz, the calculation process finish at 5.06us. **This parameter for the clk will be explained later.**

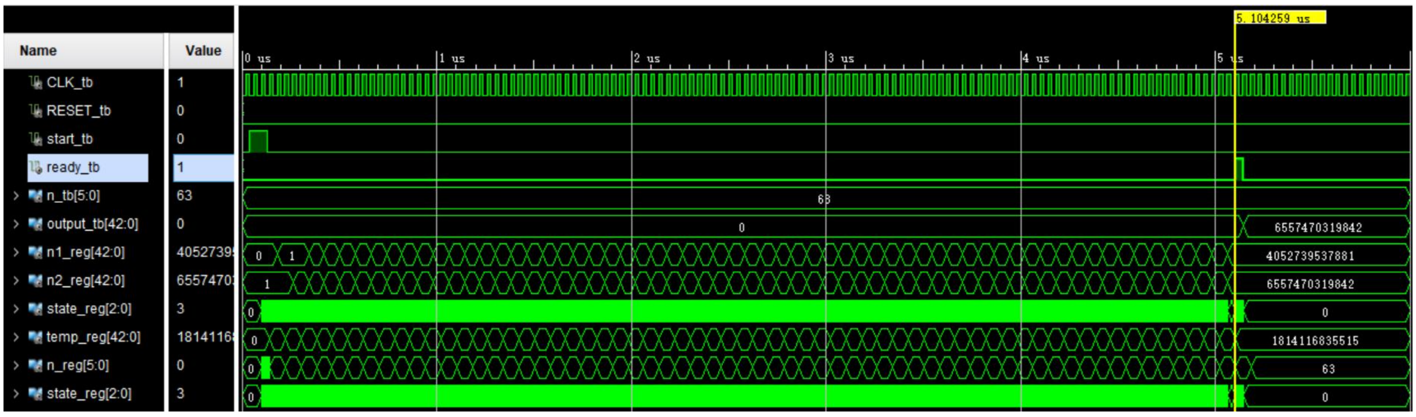
Post-synthesis functional simulation:





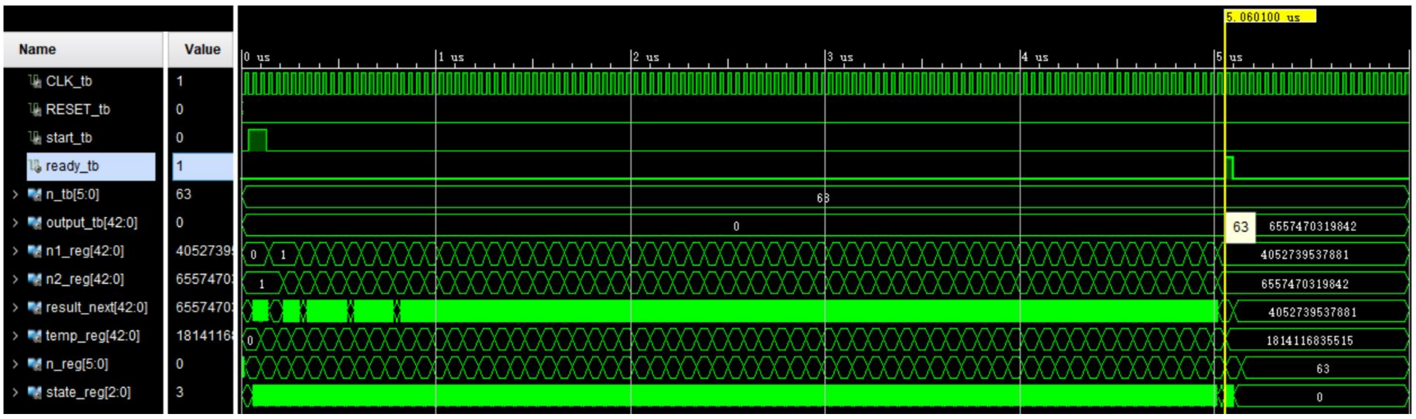
We can enlarge the index and see details. VIVADO synthesis our state into number, our five state are changed into number from 0 to 4, so the state_reg change from 4 to 2 is our state op1 and op2. And we can see the decrease of n_reg from 63 to 1.

Post-synthesis timing simulation:

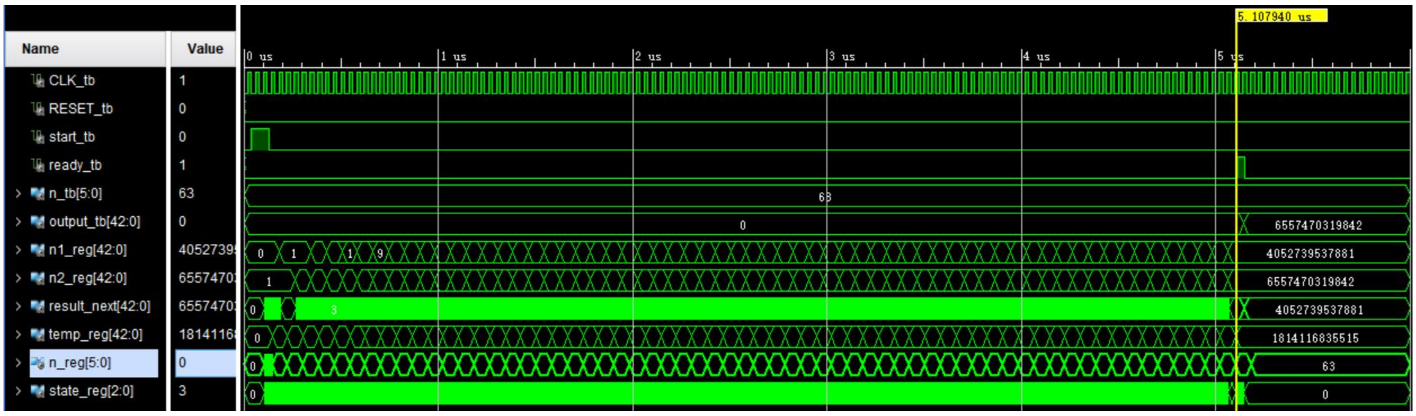


Finish time is longer than functional simulation for the delay of gate. The finish time is 5.104259us.

Post-Implementation functional simulation:



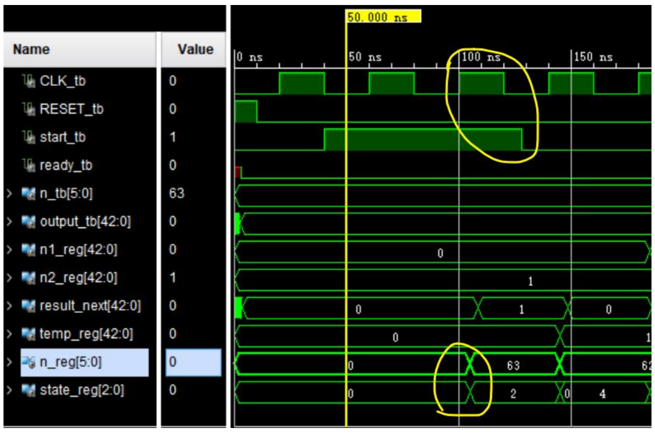
Post-Implementation timing simulation:



The finish time is 5.107940us, which is larger then synthesis timing result, this is due to the layout and wiring.

Fastest clk frequency:

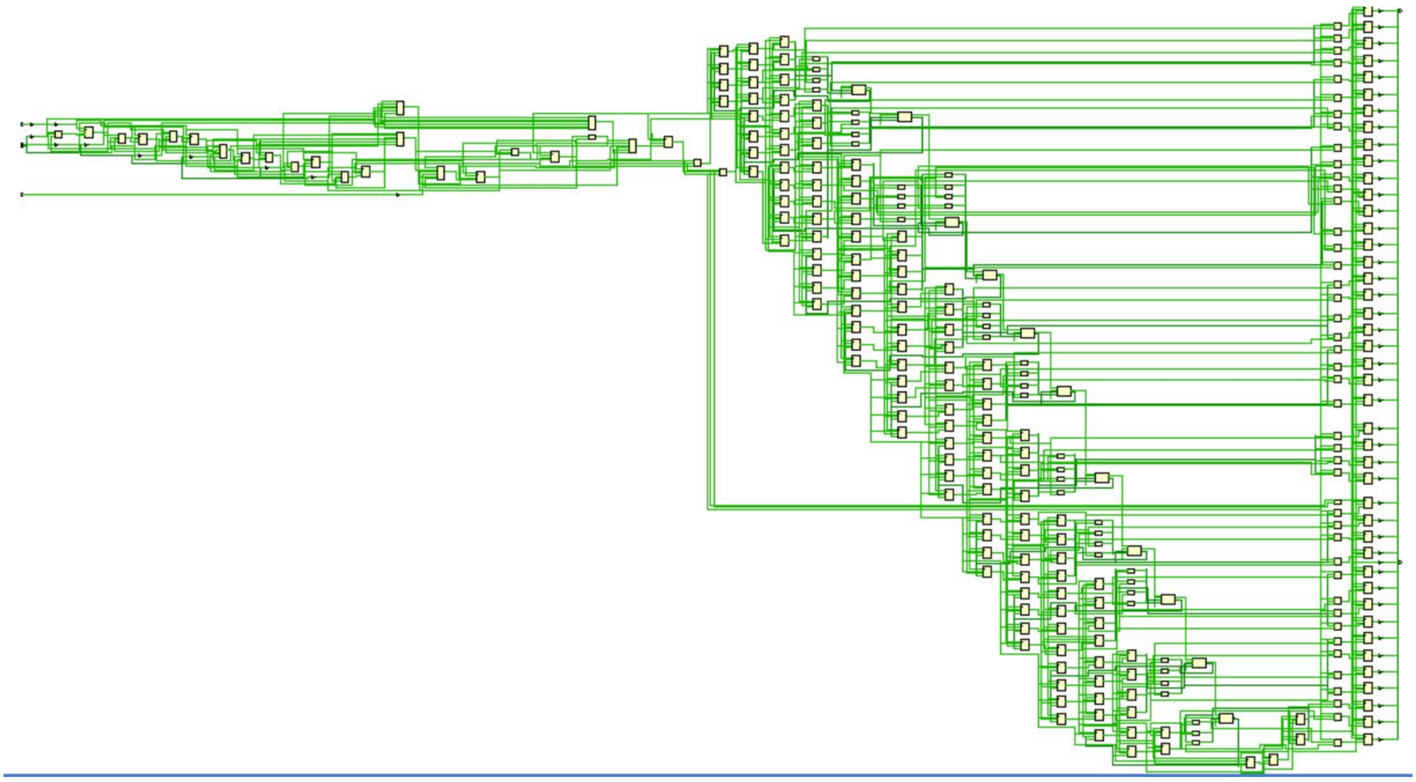
This Fibonacci function can not run at a infinity large frequency. For one thing the maximum frequency for the board is 100MHz. Another important reason is that due to the long length of the signal(43 bit) and complex circuits after synthesis, **if our clk period is too short, combinational circuit can not finish update in one period and therefore will cause mistake**. Due to my multiply try, I choose 40ns as time period and 25MHz, this is the fastest frequency which will not cause mistake.

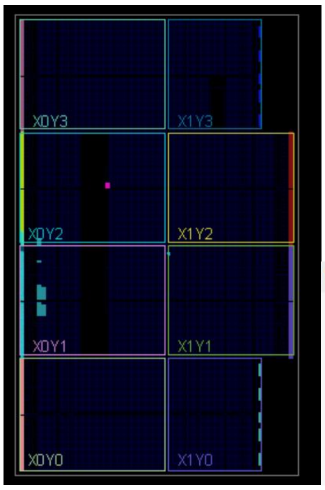


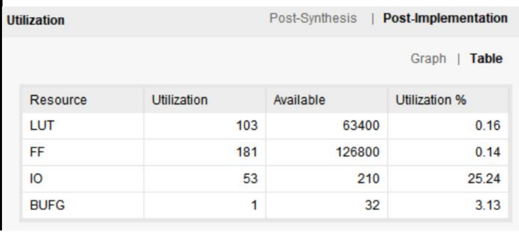
Name	Value
CLK_tb	0
RESET_tb	0
start_tb	1
ready_tb	0
n_tb[5:0]	63
output_tb[42:0]	0
n1_reg[42:0]	0
n2_reg[42:0]	1
result_next[42:0]	0
temp_reg[42:0]	0
n_reg[5:0]	0
state_reg[2:0]	0

We can see from the local part of implementation timing result here, although from our derivation, in the second clk the state can go to state op1(which is the number 2), however, **due to the combinational circuit do not finish the process after reset, it go to the state op1 in the third clk.** We should hold start signal on high level for at least 2 clk period to let the state machine go to the true working state.

Simulation Resource:



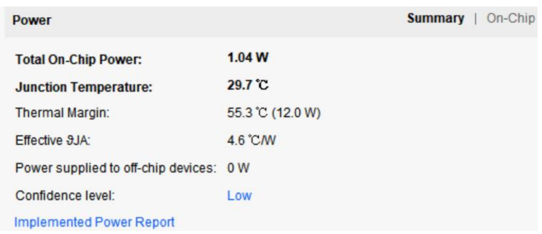




Utilization   Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	103	63400	0.16
FF	181	126800	0.14
IO	53	210	25.24
BUFG	1	32	3.13



Power   **Summary** | On-Chip

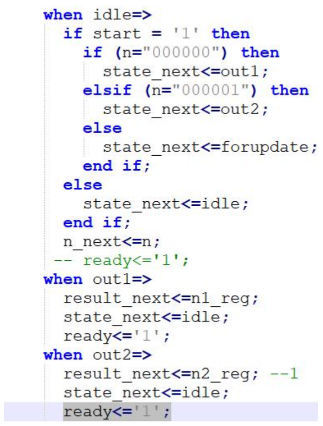
Total On-Chip Power:	**1.04 W**
Junction Temperature:	**29.7 ℃**
Thermal Margin:	55.3 ℃ (12.0 W)
Effective ϑJA:	4.6 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Further discussion:

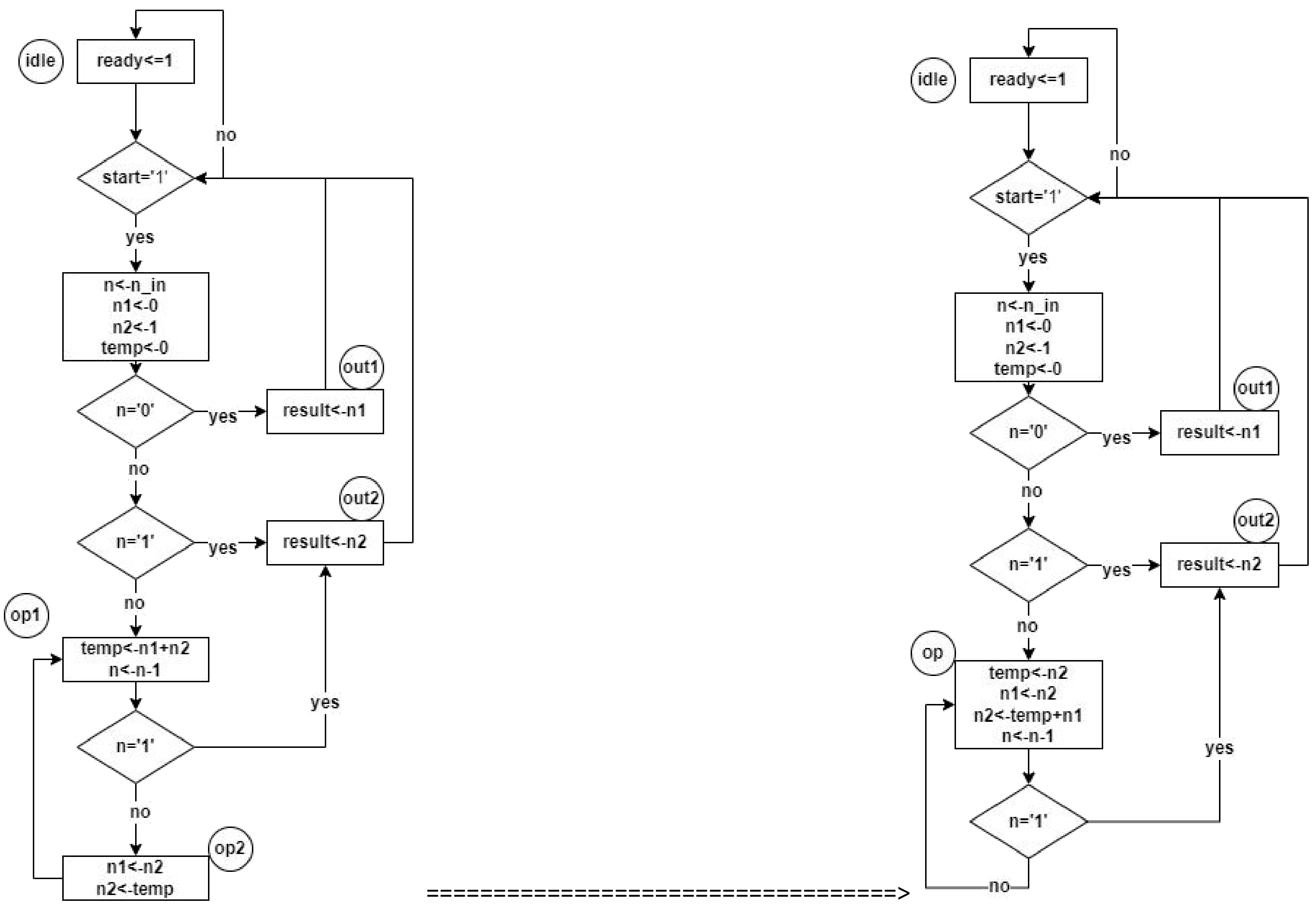
(1) In the lecture we use a state “load” to initialization the value for n and other signals. Actually, we can do this in the state “idle” and thus save a state time, but the premise is that this initialization process can be finished in one period. From the test I prove that in one period can finish initialization.

(2) As we said before, VIVADO change our state into number to mark them. So when the state change, for example in my simulation op1 is number 2 and op2 is number 4. When it change from 2(010) to 4(100), it will change to idle state 0(000) due to Competitive Risk, which will cause pulse of some signal like ready signal if we write “ready<=’1’” in the idle. So we set ready to 1 in other state like out1 and out2 to avoid this phenomenon.

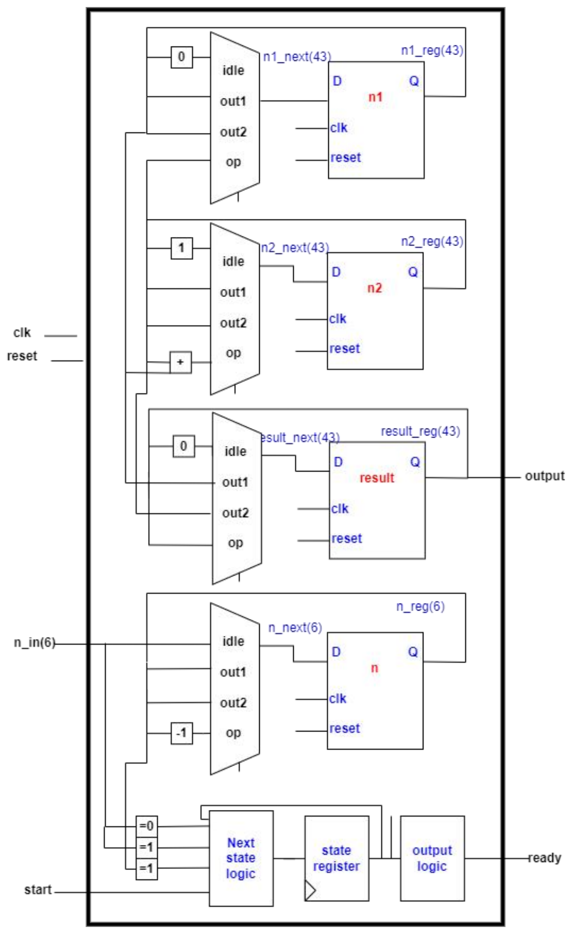


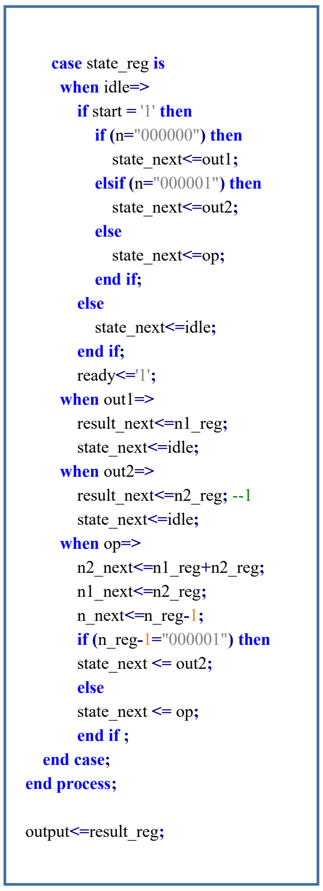
```
when idle=>
  if start = '1' then
    if (n="000000") then
      state_next<=out1;
    elsif (n="000001") then
      state_next<=out2;
    else
      state_next<=forupdate;
    end if;
  else
    state_next<=idle;
  end if;
  n_next<=n;
 -- ready<='1';
when out1=>
  result_next<=n1_reg;
  state_next<=idle;
  ready<='1';
when out2=>
  result_next<=n2_reg; --1
  state_next<=idle;
  ready<='1';
```

(3) Improvement of the ASM chart and simulation result.



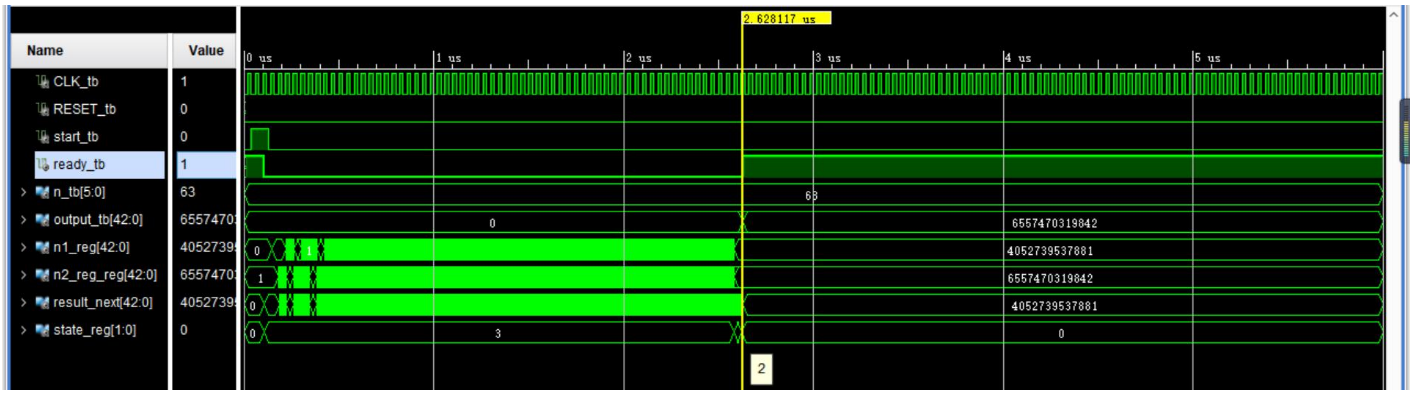
We improve the process can only use four state, and combine op1 and op2 together to op for variable can be assigned immediately. In our new design, there is no other RT except n1, n2, n and result. **We fully use the properties of delay in the VHDL synthesis to achieve the sum operations.** The ASM chart for the improved is:



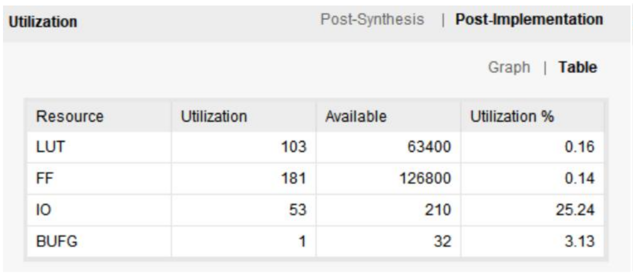


```
case state_reg is
 when idle=>
    if start = '1' then
       if (n="000000") then
          state_next<=out1;
       elsif (n="000001") then
          state_next<=out2;
       else
          state_next<=op;
       end if;
    else
       state_next<=idle;
    end if;
    ready<='1';
 when out1=>
    result_next<=n1_reg;
    state_next<=idle;
 when out2=>
    result_next<=n2_reg; --1
    state_next<=idle;
 when op=>
    n2_next<=n1_reg+n2_reg;
    n1_next<=n2_reg;
    n_next<=n_reg-1;
    if (n_reg-1="000001") then
    state_next <= out2;
    else
    state_next <= op;
    end if ;
  end case;
end process;

output<=result_reg;
```

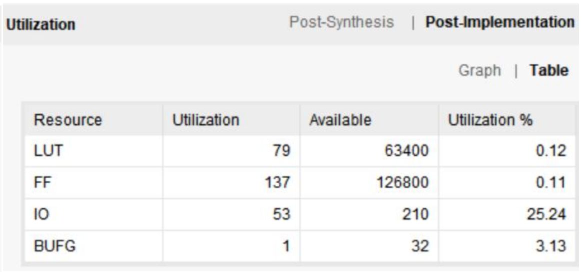


Above is the **Implementation timing** result and we only use 2.628us to get the true result, which is largely smaller than the design before.



Utilization — Post-Synthesis | **Post-Implementation** — Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	103	63400	0.16
FF	181	126800	0.14
IO	53	210	25.24
BUFG	1	32	3.13



Utilization — Post-Synthesis | **Post-Implementation** — Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	79	63400	0.12
FF	137	126800	0.11
IO	53	210	25.24
BUFG	1	32	3.13

Before simulation result

Improved simulation result

From the simulation report, our improved design use less resources.