Digital System Design Lab Report, Lab5

Author: Ying Yiwen
Number: 12210159

Abstract

In this experiment, we designed and implemented a Fibonacci series generator through a finite state machine (FSM) using the VHDL language. The ASM diagram is firstly drawn to clarify the state transfer logic and provide a clear framework for code writing. In the experiment, the generator was tested for the cases of 5, 10 and 63 input terms, and the generator correctly outputs the corresponding Fibonacci numbers 5, 55 and 6557470319842, and the simulation results are completely consistent with the theoretical analysis. The success of the experiment proves the advantages of VHDL in efficient parallel processing and real-time computation.

Contents

1 Introduction **2**

2 Algorithm **2**

3 VHDL Code **3**

4 Simulation Result **5**

5 Conclusion **6**

1 Introduction

The Fibonacci series starts with 0 and 1, and each number is derived by adding the two previous numbers. The first few Fibonacci numbers are 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987 ......
In math, Fibonacci numbers are defined in a recursive way:

$$\begin{aligned} &F_0 = 0, \\ &F_1 = 1, \\ &F_n = F_{n-1} + F_{n-2} \quad \text{for } n \geq 2. \end{aligned} \tag{1}$$

VHDL is suitable for writing Fibonacci series generators because it is a hardware description language that maps algorithms directly to hardware structures, enabling efficient parallel processing and real-time computation. This not only allows the Fibonacci series generation process to be executed quickly, but also optimizes resource usage and is easy to scale and maintain, making it ideal for application scenarios that require high performance and real-time response.

2 Algorithm

Finite State Machine is suitable for computing Fibonacci Series, as the ASM chart shows:

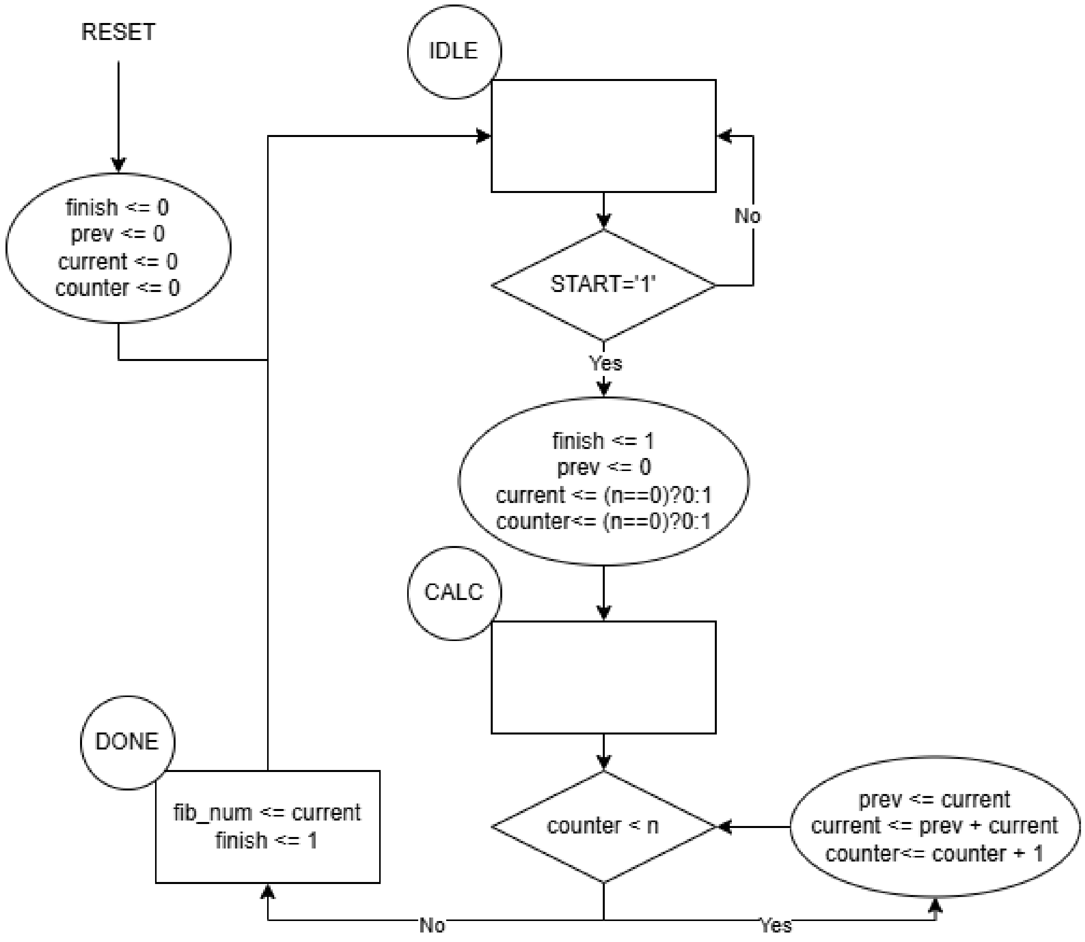


Figure 1: ASM chart of Fibonacci Series

3 VHDL Code

The VHDL code of fibonacci fsm is as follows:

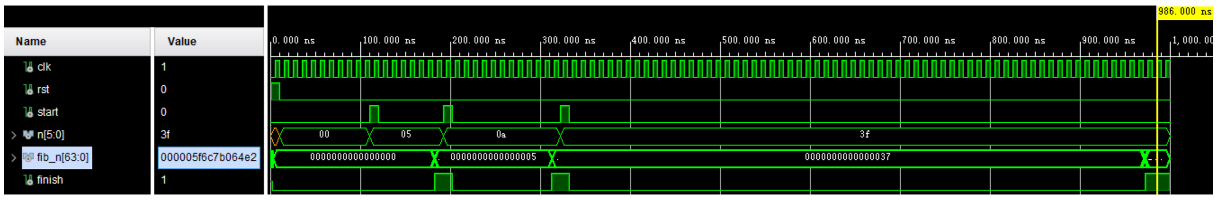
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity fibonacci_fsm is
  Port (
    clk : in  STD_LOGIC; -- state transfer
    rst : in  STD_LOGIC; -- reset signal
    start : in STD_LOGIC; -- start computing
    n : in  STD_LOGIC_VECTOR (5 downto 0); -- input
    fib_n : out STD_LOGIC_VECTOR (63 downto 0); -- output
    finish : out STD_LOGIC -- judge ok, for testbench to automatically run
  );
end fibonacci_fsm;

architecture Behavioral of fibonacci_fsm is
  type state_type is (IDLE, CALC, DONE);
  signal state : state_type := IDLE;
  signal prev : unsigned(63 downto 0) := (others => '0'); -- fib(n-2)
  signal current : unsigned(63 downto 0) := (others => '0'); -- fib(n-1)
  signal counter : integer := 0; -- count
  signal n_int : integer := 0; -- convert n to integer for comparison
begin
  n_int <= to_integer(unsigned(n));
  process(clk, rst)
  begin
    if rst = '1' then
      -- Reset state machine
      state <= IDLE;
      prev <= (others => '0');
      current <= (others => '0');
      counter <= 0;
      fib_n <= (others => '0');
      finish <= '0';
    elsif rising_edge(clk) then
      case state is
        when IDLE =>
          -- start computing
          if start = '1' then
            finish <= '0'; -- reset finish signal
            prev <= (others => '0');
            if n_int = 0 then
              current <= (others => '0');
              counter <= 0;
            else
              current <= (0 => '1', others => '0'); -- 1
              counter <= 1;
            end if;
            state <= CALC;
          end if;
        when CALC =>
```

```
          if counter < n_int then
            -- calculate fib(n)
            prev <= current;
            current <= current + prev;
            counter <= counter + 1;
          else
            state <= DONE;
          end if;
        when DONE =>
          -- output result
          fib_n <= std_logic_vector(current);
          finish <= '1';
          state <= IDLE;
      end case;
    end if;
  end process;
end Behavioral;
```

And we can use testbench to test the correctness of the code.

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

entity testbench is
  -- Port ( );
end testbench;

architecture Behavioral of testbench is

  signal clk       : STD_LOGIC := '0';
  signal rst       : STD_LOGIC := '1';
  signal start     : STD_LOGIC := '0';
  signal n         : STD_LOGIC_VECTOR (5 downto 0);
  signal fib_n     : STD_LOGIC_VECTOR (63 downto 0);
  signal finish    : STD_LOGIC;

begin

  -- Generate clock signal
  clk <= not clk after 5 ns;

  -- Instantiate the Unit Under Test (UUT)
  uut: entity fibonacci_fsm
    port map (
      clk     => clk,
      rst     => rst,
      start   => start,
      n       => n,
      fib_n   => fib_n,
      finish  => finish
    );

  -- Stimulus process
  stimulus_process : process
  begin
    -- Initialize signals
    rst <= '1';
    wait for 10 ns;
    rst <= '0';
    n <= "000000";
    start <= '0';
    wait for 100 ns;

    -- test: n = 5
    start <= '1';
    n <= "000101";
    wait for 10 ns;
    start <= '0';
    wait until finish = '1';
    wait for 10 ns;

    -- test: n = 10
    start <= '1';
    n <= "001010";
    wait for 10 ns;
    start <= '0';
    wait until finish = '1';
    wait for 10 ns;

    -- test: n = 63
    start <= '1';
    n <= "111111";
    wait for 10 ns;
    start <= '0';
    wait until finish = '1';
    wait for 10 ns;

    -- end of test
    wait;

  end process;

end Behavioral;
```

4 Simulation Result



Name	Value
clk	1
rst	0
start	0
n[5:0]	3f
fib_n[63:0]	000005f6c7b064e2
finish	1

Figure 2: Post-Implementation Timing Simulation Result

After synthesis and implementation, we can get the post-implementation timing simulation.

We test n = 5, 10, 63, and get the correct result. By therotical analysis, when n = 5, fibonacci$n$ = 5; when n = 10, fibonacci$n$ = 55; when n = 63, fibonacci$n$ = 6557470319842. The simulation result is the same as the theoretical result. In simulation, the result is 5, 37, 5F6C7B064E2 (in hexadecimal).

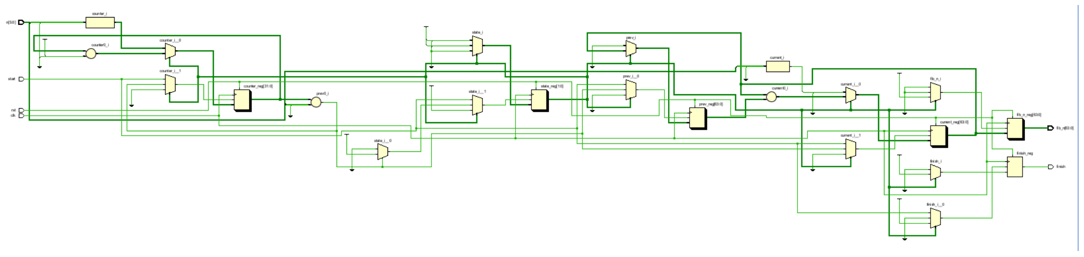


Figure 3: Schematic

5 Conclusion

In this experiment, we successfully designed and implemented a Fibonacci series generator using the VHDL language. Through the finite state machine (FSM) approach, we are able to efficiently compute the Fibonacci series with the specified number of terms. The experimental results show that the generator can correctly output the corresponding Fibonacci numbers, i.e., 5, 55, and 6557470319842 (theoretical values), when the number of input terms are 5, 10, and 63, respectively. The simulation results are in complete agreement with the theoretical analysis, verifying the accuracy and reliability of the design.

In addition, we adopted the way of drawing ASM diagrams first to clarify the state transfer logic of the finite state machine, which not only provides a clear framework for the writing of VHDL code, but also makes the whole design process more systematic and intuitive. In this way, we can better understand and realize the various steps of the algorithm, thus improving the readability and maintainability of the code.

Through this experiment, we not only deepen our understanding of Fibonacci series and its recursive nature, but also master the application of VHDL language in digital system design, especially in hardware description and algorithm implementation. In conclusion, this experiment not only achieves the expected goal, but also demonstrates the powerful function and flexibility of VHDL in hardware design.