ECSE 425 Project 1 - Finite State Machines

Haoran Du
260776911
McGill University
Montreal, Canada
haoran.du@mail.mcgill.ca

Carlo D'Angelo
260803454
McGill University
Montreal, Canada
carlo.dangelo@mail.mcgill.ca

***Abstract*—In this project, the team built and tested a finite-state machine (FSM) to identify the commented characters in C code. The inputs to this FSM are the clock, an asynchronous reset signal and one ASCII character per clock cycle. The output is designed to be '0' if the input character is not part of a comment and '1' if it is. The FSM is also designed in a way to function on both of the C programming comment styles, namely '//...\n' and '/*...*/'.**

I. Introduction

The purpose of designing this finite-state machine (FSM) is to distinguish the comment parts in a block of C code. For instance, in the following block of C code showing as an example in Listing 1, we can see that the comments characters where the FSM outputs should be '1' is displayed in green.

```
/* This is the\n main method*/\n
int main(){\n
    // Lorem ipsum dolor sit amet\n
    return 0; \n
}
```

Listing 1. Example of C code with comments.

Specifically, the exit sequence for the comment ('\n' or '*/') is considered a comment while the opening sequence ('//' or '/*') is not. Therefore, the output requirements of this FSM is:

1) '1' in the clock cycle after the second character of the opening sequence appears;
2) '0' in the clock cycle after the second character of the exit sequence appears.

II. FSM Implementation

A. State Diagram

The state diagram displayed in Fig. 1 shows how the system will react when encountering different ASCII characters. Since there are two types of comments in C code, those that start with '//' and those that start with '/*', the FSM implementation has to have more states

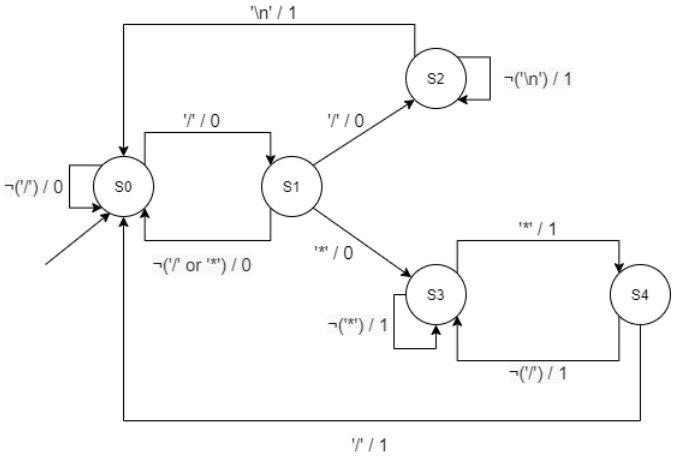

Fig. 1. State diagram for identifying commented characters in C code

to deal with these different cases. In particular, the S0-S1 states make sure that the system reacts accordingly to the comments' opening sequences. Then, after an opening sequence is detected, the S2 state takes care of '//' comments specifically and the S3-S4 states takes care of '/*' comments specifically, including how to react to their respective closing sequences.

When the FSM is at S2, S3, or S4, it also changes the output value to be '1' indicating that the input character is part of a comment.

In this design, the initial state is S0.

B. VHDL Implementation

The instructions of the project description state that one ASCII character per clock cycle is fed into the FSM. Therefore, the team designed the FSM to be time-triggered and explicitly change states and outputs at every rising edge of every clock cycle.

For example, the code block below in Listing 2 is the example of the transition between S0 and S1. In theory, one ASCII character input is given at the beginning of a

clock cycle. Then, at the rising edge of that clock cycle, the output and the state transition is determined based on the design in Fig. 1, i.e.,

if $input == /$ **then**
  $state \leftarrow S1$
  $output \leftarrow 0$
else
  $state \leftarrow S0$
  $output \leftarrow 0$
end if

```
begin
-- Insert your processes here
fsm: process (clk, reset)
begin
  if (reset = '1') then
    current_state <= S0;
    output <= 'X';

  elsif (rising_edge(clk)) then
    case current_state is
      when S0 =>
        if (input = SLASH_CHARACTER) then
          output <= '0';
          current_state <= S1;
        else
          output <= '0';
          current_state <= S0;
        end if;
      when S1 =>
          ...
```

Listing 2. VHDL implementation of FSM

Notice that, no matter what the system's current state is, if the reset signal is set to '1', then the state of the FSM is always reset to S0. The output, in this case, is 'X' which in std_logic represents UNKNOWN. Furthermore, by setting the reset signal to '1', it can also be ensured that the initial state of the FSM can be set to S0.

III. Design and Results of the Testbench

The team followed the given testbench starter code template and the instructions in the manual regarding the default resolution. Additionally, the following initialization code was implemented to ensure that the FSM always starts at S0.

```
--TODO: Thoroughly test your FSM
stim_process: PROCESS
BEGIN
  REPORT "Initialization (setting current
    state to S0)";
  s_reset <= '1';
  WAIT FOR 1 * clk_period;
  s_reset <= '0';
  WAIT FOR 1 * clk_period;
```

Listing 3. Testbench initialization

The team implemented 5 different test cases in our testbench to thoroughly test the FSM. The contents of the 5 cases and their respective results are presented below.

A. Test case 1: X (meaningless character)

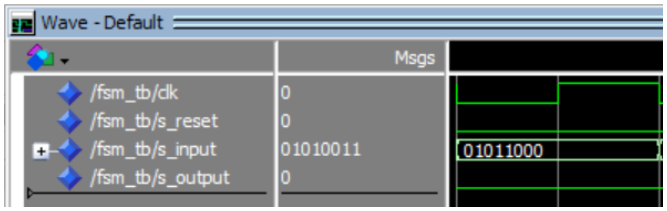

Fig. 2. Wave for test case 1 from ModelSim

B. Test case 2: //AS\nD

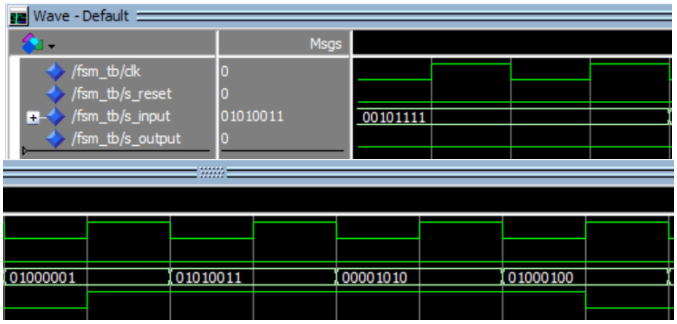

Fig. 3. Wave for test case 2 from ModelSim

C. Test case 3: /*A\nS*/D

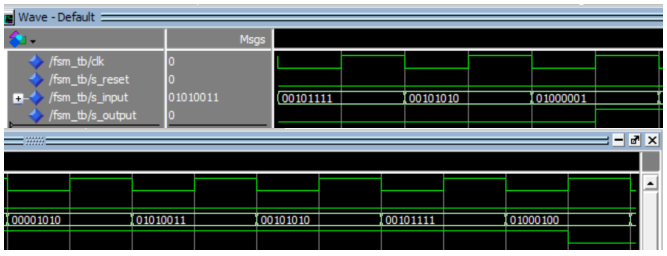

Fig. 4. Wave for test case 3 from ModelSim

D. Test case 4: //A(RESET)S

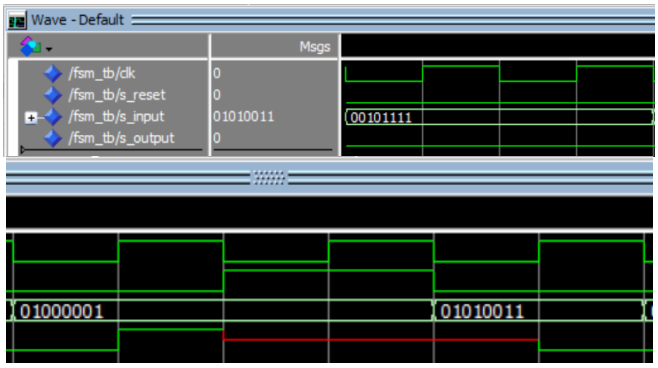

Fig. 5. Wave for test case 4 from ModelSim

E. Test case 5: /*A(RESET)S

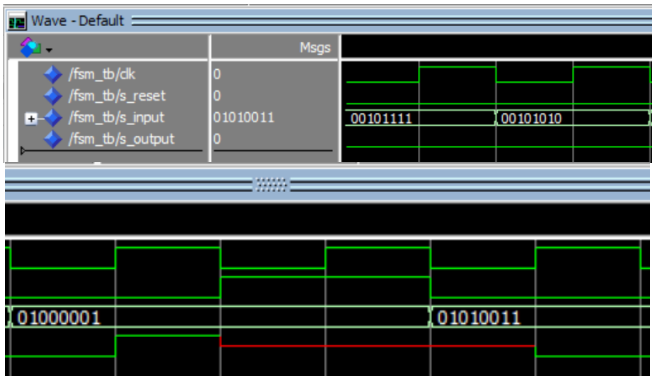

Fig. 6. Wave for test case 5 from ModelSim

IV. Conclusion

Based on the 5 signal results shown above obtained from the simulation of our testbench, it can be concluded that the FSM functions correctly pertaining to all the design requirements.