ECSE425 Lab 1 Report

Zeying Tian
McGill University
Montreal, Canada
zeying.tian@mail.mcgill.ca

Junjian Chen
McGill University
Montreal, Canada
junjian.chen2@mail.mcgill.ca

Hongtao Xu
McGill University
Montreal, Canada
hongtao.xu3@mail.mcgill.ca

***Abstract*—In this project, our group designed and tested a finite-state machine (FSM) to identify the commented characters in C code. There are five states in our FSM to represent the comment symbols in C language, including ’//..\n’ and ’/*...*/’.**

I. Introduction

The purpose of designing this finite-state machine (FSM) is to distinguish the comment symbols in a block of C code. The inputs to this FSM are the clock, one ASCII character per clock cycle,and an asynchronous reset signal. The output is ’1’ or ’0’ to tell the code is commented or not, respectively. In the design part, we defined five states to tell if the code is commented or not, when the code will be commented and when to exit the commented status. In specific, the open symbol for the comment are ’//’ or ’/*’ while the exit symbol are ’\n’ or ’*/’. In this way, the output is ’0’ in the clock cycle after the second character of the exiting symbol comes out, while the output is ’1’ in the clock cycle after the second character of the opening symbol comes out.

II. Methods

A. State Diagram of Our Implementation

The state diagram of our implementation is presented as in Figure 1. There are five states in our implementation, namely `S_Code`, `S_CommentEntry`, `S_Comment1`, `S_Comment2`, and `S_Comment2Exit`. The `S_Code` state represents the non-commentary state, i.e., the state when the current character is not part of a comment. The `S_CommentEntry` specifies the state when we are about to enter a comment section, which takes place as we detect the / character. State `S_Comment1` represents the single-line comment which begins with //. Likewise, state `S_Comment2` stands for the multi-line comment section which is enclosed by $/*$ and $*/$. Finally, the state `S_Comment2Exit` represents the state at which we are about to exit the multi-line comment, and this happens when we see a $*$ character inside the multi-line comment. Then transitions among states and their corresponding outputs are labeled in Figure 1.

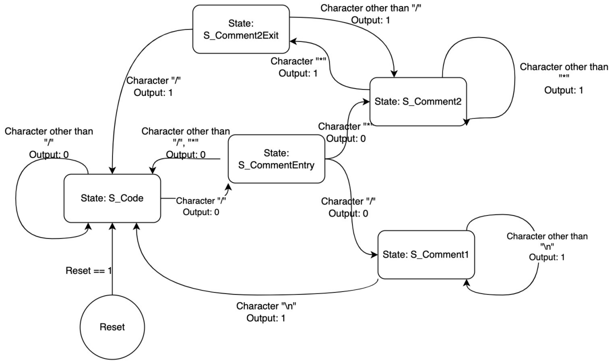

Fig. 1. The State Diagram

B. Explanation of Our Implementation

First of all, we define our five states and a variable `current_state` to store the current state, as in Listing 1.

```
architecture behaviour of FSM is
-- S_Code: When the code is not commented
-- S_CommentEntry: When the code is going to be commented i.e When meeting a "/"
-- S_Comment1: When the code is commented with "//"
-- S_Comment2: When the code is commented with "/*"
-- S_Comment2Exit: When the code is already commented with "/*" and going to exit commented status
--                 i.e When meeting a "*"
 type state_type is (S_Code, S_CommentEntry, S_Comment1, S_Comment2, S_Comment2Exit);
 signal current_state: state_type;
```

Listing 1. Signal and State Definition

Next, we define the reset signal, which will force the `current_state` to be `S_Code`, our initial state. The reset action can be toggled by the `reset` signal, as shown in Listing 2.

```
if reset’event and reset=’1’ then
current_state <= S_Code;
```

Listing 2. Reset State

Then, we define the behavior of our state machine with `case` and `when` keywords. Inside each state, we specify the actions with if-else statements. For instance, as we are in the `S_Code` state, we check if the current character is /, which has an `ASCII` code of 47, or 101111 in binary. If this is the case, we shall move on to the next state, `S_CommentEntry`, and assign the output the value 0. If this is not the case, we shall stay in the current state `S_Code` and give output a value of 0. The behavior described above is illustrated by Listing 3.

```
when S_Code => -- When the code is not commented
 if input="101111" then -- When meeting "/", it is going to be commented
   output <= ’0’;
    current_state <= S_CommentEntry;
```

```
  else -- When meeting any other char, not commented
   output <= ’0’;
   current_state <= S_Code;
  end if;
```

Listing 3. S_Code Behavior

Following the same paradigm, we have defined the behaviors for the other four states as in Listing 4, Listing 5, Listing 6, Listing 7.

```
when S_CommentEntry => -- When the code is going to be commented
 if input="101111" then -- When meeting "/" for the second time, it’s commented by "//"
   output <= ’0’;
    current_state <= S_Comment1;
  elsif input="101010" then -- When meeting "*", it’s commented by "/*"
   output <= ’0’;
    current_state <= S_Comment2;
  else -- When meeting any other char, still not commented, go back to code status
   output <= ’0’;
    current_state <= S_Code;
  end if;
```

Listing 4. S_CommentEntry Behavior

```
when S_Comment1 => -- When the code is comment with "//"
 if input="1010" then -- When meeting "\n", it’s a new line, so exit commented status
   output <= ’1’;
    current_state <= S_Code;
  else -- When meeting any other char, still remain commented
   output <= ’1’;
    current_state <= S_Comment1;
  end if;
```

Listing 5. S_Comment1 Behavior

```
when S_Comment2 => -- When the code is comment with "/*"
 if input="101010" then -- When metting "*", it’s going to exit commented status
   output <= ’1’;
    current_state <= S_Comment2Exit;
  else -- When meeting any other char, still remain commented
   output <= ’1’;
    current_state <= S_Comment2;
  end if;
```

Listing 6. S_Comment2 Behavior

```
when S_Comment2Exit => -- When the code is going to exit commented state by "*/"
 if input="101111" then -- When metting "/", the commented status is exited with "*/", go back to code status
   output <= ’1’;
    current_state <= S_Code;
  else -- When meeting any other char, still remain commented
   output <= ’1’;
    current_state <= S_Comment2;
  end if;
```

Listing 7. S_Comment2Exit Behavior

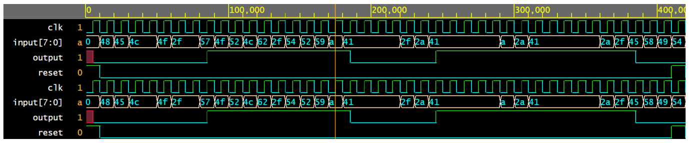
0
100,000
200,000
300,000
400,00
clk 1
input[7:0] a 0 48 45 4c 4f 2f 57 4f 52 4c 62 2f 54 52 59 a 41 2f 2a 41 a 2a 41 2a 2f 45 58 49 54
output 1
reset 0
clk 1
input[7:0] a 0 48 45 4c 4f 2f 57 4f 52 4c 62 2f 54 52 59 a 41 2f 2a 41 a 2a 41 2a 2f 45 58 49 54
output 1
reset 0

Fig. 2. The Output of the FSM

III. Discussion of Our Test Results

In the testbench, we used the testcode: hello//world/try\n aaaa/*aaaaa\n*aaaaa*/exit As we can see in the Figure 2, our testcode includes all the conditions. In specific, we set the clock of 10ns per clock cycle.

The output of the words hello// is 0 in the EPWave which means not commented, as shown in Figure 3. It is noted that the output of // is 0 since the opening sequence is not considered a comment.

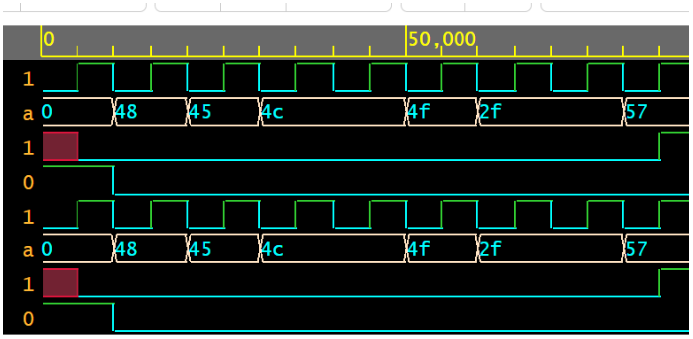

Fig. 3. The Output of ”hello//”

Since the code is commented by //, the output of the following words world/try\n” should be 1, as shown in Figure 4. It is noted that the output of \n is 1 since the exit sequence is considered comment.

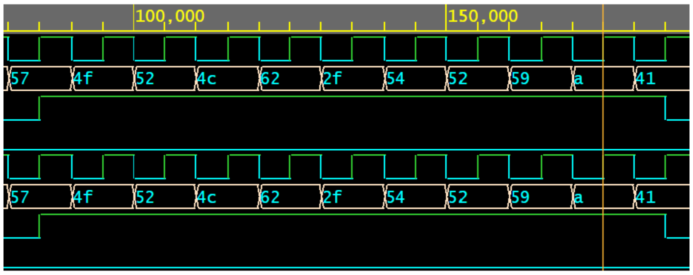
100,000
150,000
57 4f 52 4c 62 2f 54 52 59 a 41
57 4f 52 4c 62 2f 54 52 59 a 41

Fig. 4. The Output of ”world/try\n”

As we exit the commented status by entering a new line(i.e. \n), the output of aaaa/* is 0, as illustrated in Figure 5.

As we can see from Figure 6, the output of aaaaa\n*aaaaa*/ is 1, since the code is commented by /*. If the code is commented by /*, we cannot exit commented status by entering a new line, so the output does not turns to 0 when the code encounters \n.

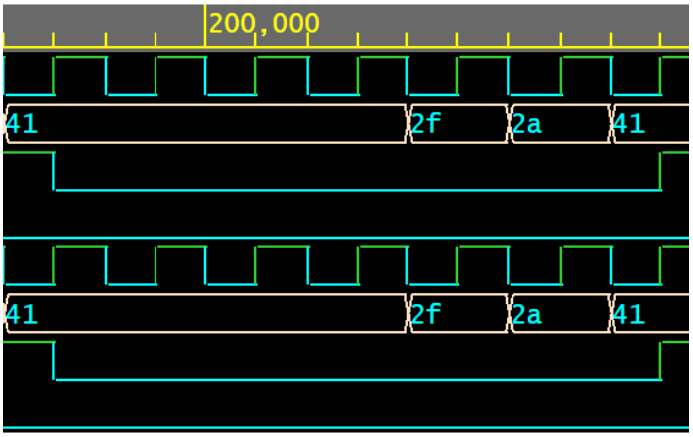

Fig. 5. The Output of ”aaaa/*”

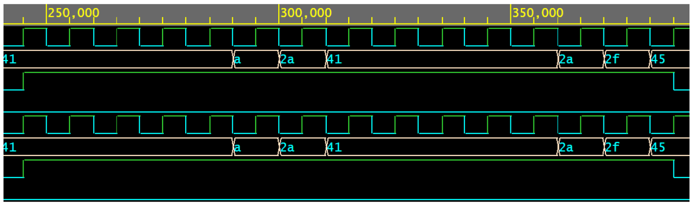

Fig. 6. The Output of ”aaaaa\n*aaaaa*/”

Finally, as shown in Figure 7, the output of `exit` is 0 because we have exited the commented status by $\star$/.

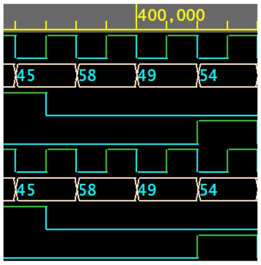

Fig. 7. The Output of ”exit”

In conclusion, this FSM functions correctly include all the design requirements.