ECSE 425 Project 1: Finite State Machines

Noah Zwack *Allan Reuben*
260769910 *260783324*

Introduction

In this project, we were tasked with creating a finite state machine for identifying commented characters in C code. Our state machine was programmed in VHDL on Visual Studio Code and was tested for full functionality using ModelSim.

State Machine

Let us begin by going through the state machine we designed, which is pictured in Figure 1. The state machine is initialized in state 0, where the output is 0. In this state, only the '/' character will trigger a transition, which in this case takes us to state 1. Here, the output is still 0.

At this stage, there are three transitions that can occur. First, if the next character is '*', we have detected a block comment, so we transition to state 2. Second, if the next character is '/', we have detected a single-line comment, so we transition to state 5. If the next character is neither of those characters, then no comment has been detected, so we return to state 0.

In the case of a block comment being detected, we will begin to output 1 on the next detected character. We will continue to output 1 until the '*/' sequence is detected, at which point we will go to state 7.

In the case of a single-line comment being detected, we will also begin to output 1 on the next detected character. All subsequent characters will also produce an output of 1, up until the new-line character is detected. This character will produce an output of 1, then trigger a transition to state 7.

Once in state 7, if the next detected character is a '/', it will trigger a transition to state 1. Otherwise, the machine will transition to state 0.

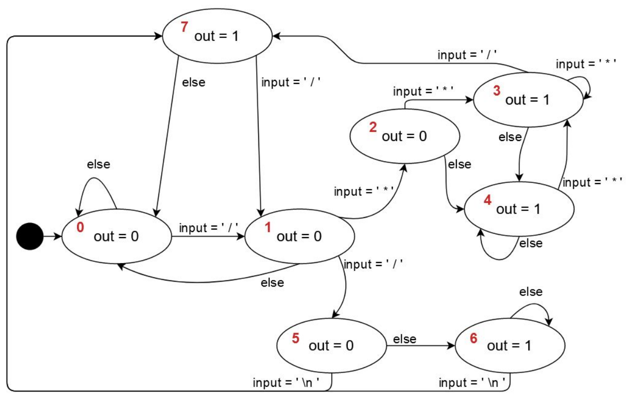


Figure 1: A diagram of the state machine designed for this project. The numbers in red are the state numbers.

Implementation

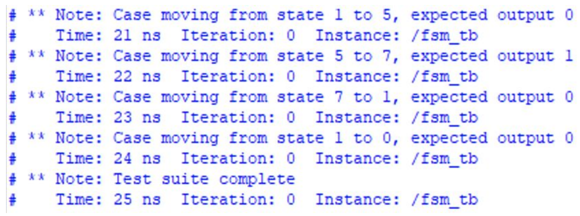
In order to implement the state machine described in the previous section, we decided to use a variable to keep track of the current state we are in at any given moment. Since we have eight different states, we used a 3-bit signal to represent the current state.

In our process block, we first check if the reset bit is set to high. If so, we set the state and the output to 0. Otherwise, if it is a rising clock edge, we enter a case statement. The case statement will check which state we are currently in, then check for the corresponding character. For example, if we are currently in state 1, the code will check first for the ‘*’ character, then the ‘/’ character, then for any other character. Depending on what the input is, it will take the appropriate transition by changing the current state and the output.

Test and Simulation Results

In order to test our VHDL state machine, we designed a testbench that examines all execution cases, and used ModelSim to run the testbench and verify our state machine design according to the diagram in Figure 1.

Our testbench followed the format of running tests in succession to transition between states, and verifying that the transition occurred correctly by checking the output. We used the REPORT line to print to console what test was executing to be able to identify bugs and correct them. Each test followed the format: printing information to the console; setting the desired input (a ‘*’, ‘/’, ‘/n’, or other character) to change the state; waiting one clock cycle for the output to be set; and asserting that the output is as expected. In Figure 2, we can see part of the execution in ModelSim’s transcript.



```
# ** Note: Case moving from state 1 to 5, expected output 0
#    Time: 21 ns  Iteration: 0  Instance: /fsm_tb
# ** Note: Case moving from state 5 to 7, expected output 1
#    Time: 22 ns  Iteration: 0  Instance: /fsm_tb
# ** Note: Case moving from state 7 to 1, expected output 0
#    Time: 23 ns  Iteration: 0  Instance: /fsm_tb
# ** Note: Case moving from state 1 to 0, expected output 0
#    Time: 24 ns  Iteration: 0  Instance: /fsm_tb
# ** Note: Test suite complete
#    Time: 25 ns  Iteration: 0  Instance: /fsm_tb
```

Figure 2: A representative log of our testbench execution.

No error in the log means that the test executed successfully. In our final testbench log, we detected no errors in the state machine. The few errors that we did detect were errors in the test suite, such as syntax errors and forgetting to set the reset input to 0 after testing for its functionality.

Our test suite successfully verified all state transitions and outputs in the state machine including reset functionality, making the state machine program suitable for identifying commented characters in C.

Conclusion

In conclusion, we have designed, implemented, and tested a state machine for detecting block comments and single-line comments in C. Our implementation was done in VHDL, following the state machine that we designed. We then created test cases that would test all the transitions in our state machine and simulated them through ModelSim.