Structured Digital Design
EEN 316 48

Project #2
Programmable Sequence Detector

Connor McCullough

Jia Xu, TA

University of Miami

6 December 2013

Abstract

The circuit implemented in this project is a behavioral model of a sequence detection state machine, which allows a programmable sequence for detection, as well as variable length for the sequence. Up to 8 sequences can be checked for at once, all with different lengths. This circuit is widely used in encoding and decoding of serial bit streams, such as MIDI or audio compression formats. The flexibility in being able to program the sequence and length allow it to have many potential uses. For this particular circuit, the behavioral method is much more preferable to structural, which would require extensive use of programmable logic elements as well as memory elements. Instead of modeling the entire digital circuit, all that is required to emulate this circuit using the behavioral method is the state diagram.

Table of Contents

Introduction

This circuit has a serial one-bit input which is checked each cycle and 8 bit output which sends pulses on the bit corresponding to the sequence detected. There are 2 different parameters: the actual sequence being detected, and the length of the sequence. These parameters are available for all 8 programmable sequences. There are two entities in this project, the basic mealy model emulation of the state diagram for an 8 bit sequence detector, and the control unit which implements 8 separate instances of the model, writing the outputs to a byte which is the output. The model is emulated using a behavioral model, and the control unit is implemented with a structural model.

Objectives

This report will discuss the implementation of an 8 bit sequence detector, which includes creating a state diagram, coding of the model, and programming the correct sequences to get the desired results.

Requirements and Equipment

ModelSim Altera
DE2 Board

Design and Implementation

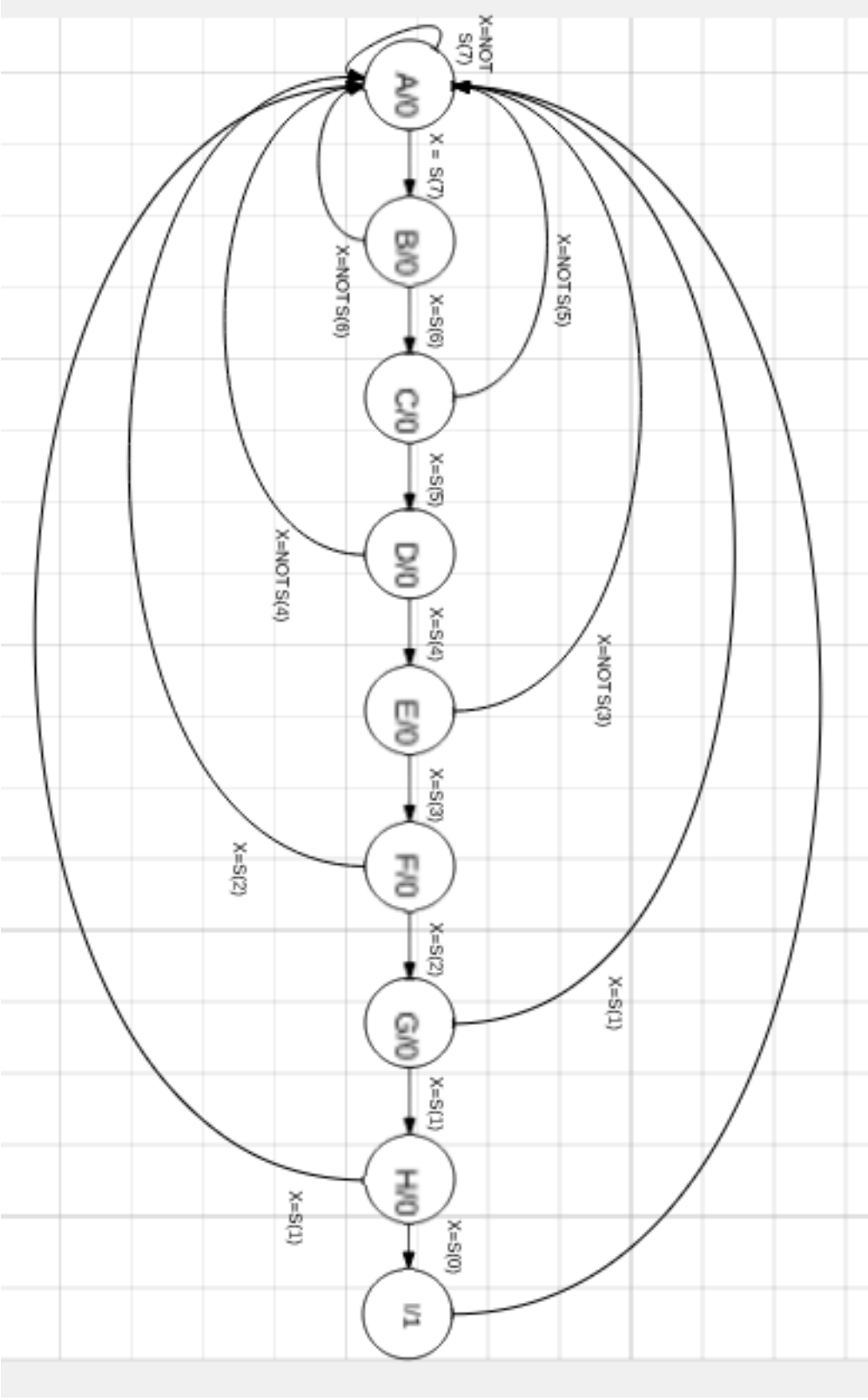
The first process of this project was creating a state diagram that the behavioral model will follow. Unlike a traditional state digram, the state changes on this diagram are based on variables. If the input equals the respective programmed bit, the state will increment. If the input equals the complement of that bit, the state will return to the first state. The state diagram is for a length of 8. The capability of variable length was implemented later in the project.

The first step of implementing the behavioral model was creating a specific data type for the state machine: an enumerated list with letters A to H representing the possible states. Next, the state will be set to 'A' when the reset pulse is 1. After that, the state can be changed on the rising edge of every clock cycle. The state machine is modeled using a case statement, incrementing the state if the input matches the programmed sequence, else setting the state back to A. The output is '0' until the entire sequence has been completed, in which a '1' is returned.

After this aspect was successfully implemented and tested, the capability of variable length was added. This meant that the case for each step will check the length, and based on whether the sequence of that length has been completed, either continue incrementing the state and returning 0, or returning 1 and going back to the A state.

The final step was creating a Control Unit, which, using a structural method, creates 8 instances of the behavioral mealy model. This also creates 8 instances of the parameters and maps them to the correct instance. The outputs for each model are each written to a specific bit in the Control Unit's output byte.

State Diagram (8 bit sequence)



VHDL Code

Mealy Model for Single Sequence Detector:

```
library ieee;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity mealy is
port (clk: in std_logic;
      reset: in std_logic;
      length: in integer;
      sequence: in std_logic_vector (7 downto 0);
      input: in std_logic;
      output: out std_logic
    );
end entity;

architecture behavioral of mealy is

  type state_type is (A,B,C,D,E,F,G,H);
  signal current, nexts: state_type;

  begin
    process(clk,reset)
      begin
        if (reset='1') then
          current <= A;
        elsif (rising_edge(clk)) then
          current <= nexts;
      end if;
    end process;

process (clk,current, input)
begin
  if (rising_edge(clk)) then
  case current is
    when A =>
    if(input=sequence(7)) then
      if length = 1 then
        output <= '1';
        nexts <= A;
      else
        output <= '0';
        nexts <= B;
      end if;
```

```
else
  output <= '0';
  nexts <= A;
end if;

when B =>
if(input=sequence(6)) then
  if length = 2 then
    output <= '1';
    nexts <= A;
  else
    output <= '0';
    nexts <= C;
  end if;
else
  output <= '0';
  nexts <= A;
end if;

when C =>
if(input=sequence(5)) then
  if length = 3 then
    output <= '1';
    nexts <= A;
  else
   output <= '0';
   nexts <= D;
  end if;
else
  output <= '0';
  nexts <= A;
end if;

when D =>
if(input=sequence(4)) then
  if length = 4 then
    output <= '1';
    nexts <= A;
  else
   output <= '0';
   nexts <= E;
  end if;
else
  output <= '0';
  nexts <= A;
end if;
```

```
when E =>
if(input=sequence(3)) then
  if length = 5 then
    output <= '1';
    nexts <= A;
  else
    output <= '0';
    nexts <= F;
  end if;
else
  output <= '0';
  nexts <= A;
end if;

when F =>
if(input=sequence(2)) then
  if length = 6 then
    output <= '1';
    nexts <= A;
  else
    output <= '0';
    nexts <= G;
  end if;
else
  output <= '0';
  nexts <= A;
end if;

when G =>
if(input=sequence(1)) then
  if length = 7 then
   output <= '1';
    nexts <= A;
  else
    output <= '0';
    nexts <= H;
  end if;
else
  output <= '0';
  nexts <= A;
end if;

when H =>
if(input=sequence(0)) then
  output <= '1';
```

```
      nexts <= A;
    else
      output <= '0';
      nexts <= A;
    end if;
  end case;
  end if;
end process;
end architecture;
```

Sequence Detector Control Unit:

```
library ieee;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity CU is port
      (clock: in std_logic;
      re: in std_logic;
      length1, length2, length3, length4, length5, length6,
length7, length8: in integer;

sequence1,sequence2,sequence3,sequence4,sequence5,sequence6
,sequence7,sequence8: in std_logic_vector (7 downto 0);
      input: in std_logic;
      output: out std_logic_vector(7 downto 0)
      );
end entity;

architecture behavioral of CU is

  component mealy is port
    (clk: in std_logic;
    reset: in std_logic;
    length: in integer;
    sequence: in std_logic_vector (7 downto 0);
    input: in std_logic;
    output: out std_logic);
  end component;

  begin
    M1: mealy port
map(clock,re,length1,sequence1,input,output(7));
    M2: mealy port
map(clock,re,length2,sequence2,input,output(6));
    M3: mealy port
map(clock,re,length3,sequence3,input,output(5));
```

```
    M4: mealy port
map(clock,re,length4,sequence4,input,output(4));
    M5: mealy port
map(clock,re,length5,sequence5,input,output(3));
    M6: mealy port
map(clock,re,length6,sequence6,input,output(2));
    M7: mealy port
map(clock,re,length7,sequence7,input,output(1));
    M8: mealy port
map(clock,re,length8,sequence8,input,output(0));
end architecture;
```

Results and Simulations

Do Script:

```
force -freeze sim:/cu/clock 1 0
force -freeze sim:/cu/re 1 0
force -freeze sim:/cu/length1 1 0
force -freeze sim:/cu/length2 2 0
force -freeze sim:/cu/length3 3 0
force -freeze sim:/cu/length4 4 0
force -freeze sim:/cu/length5 5 0
force -freeze sim:/cu/length6 6 0
force -freeze sim:/cu/length7 7 0
force -freeze sim:/cu/length8 8 0
force -freeze sim:/cu/sequence1 10101010 0
force -freeze sim:/cu/sequence2 10101010 0
force -freeze sim:/cu/sequence3 10101010 0
force -freeze sim:/cu/sequence4 10101010 0
force -freeze sim:/cu/sequence5 10101010 0
force -freeze sim:/cu/sequence6 10101010 0
force -freeze sim:/cu/sequence7 10101010 0
force -freeze sim:/cu/sequence8 10101010 0

force -freeze sim:/cu/clock 1 0
force -freeze sim:/cu/re 1 0
force -freeze sim:/cu/length1 1 0
force -freeze sim:/cu/length2 2 0
force -freeze sim:/cu/length3 3 0
force -freeze sim:/cu/length4 4 0
force -freeze sim:/cu/length5 5 0
force -freeze sim:/cu/length6 6 0
force -freeze sim:/cu/length7 7 0
force -freeze sim:/cu/length8 8 0
force -freeze sim:/cu/sequence1 10101010 0
```

```
force -freeze sim:/cu/sequence2 10101010 0
force -freeze sim:/cu/sequence3 10101010 0
force -freeze sim:/cu/sequence4 10101010 0
force -freeze sim:/cu/sequence5 10101010 0
force -freeze sim:/cu/sequence6 10101010 0
force -freeze sim:/cu/sequence7 10101010 0
force -freeze sim:/cu/sequence8 10101010 0
run
force -freeze sim:/cu/re 0 0
force -freeze sim:/cu/clock 0 0
force -freeze sim:/cu/input 1 0
run
force -freeze sim:/cu/clock 1 0
run
force -freeze sim:/cu/input 0 0
force -freeze sim:/cu/clock 0 0
run
force -freeze sim:/cu/clock 1 0
run
force -freeze sim:/cu/clock 0 0
force -freeze sim:/cu/input 1 0
run
force -freeze sim:/cu/clock 1 0
run
force -freeze sim:/cu/clock 0 0
force -freeze sim:/cu/input 0 0
run
force -freeze sim:/cu/clock 1 0
run
force -freeze sim:/cu/input 1 0
force -freeze sim:/cu/clock 0 0
run
force -freeze sim:/cu/clock 1 0
run
force -freeze sim:/cu/clock 0 0
force -freeze sim:/cu/input 0 0
run
force -freeze sim:/cu/clock 1 0
run
force -freeze sim:/cu/input 1 0
noforce sim:/cu/clock
force -freeze sim:/cu/clock 0 0
run
force -freeze sim:/cu/clock 1 0
run
force -freeze sim:/cu/clock 0 0
```

```
force -freeze sim:/cu/input 0 0
run
force -freeze sim:/cu/clock 1 0
run
```

Synthesis

The simulation was programmed in order to show all aspects of the project work correctly. The sequences are all set to "10101010", but the lengths vary, so the first sequence has a length of 1 and the last sequence a length of 8. A serial stream of alternating 1's and 0's is then input. This means that the different output bits should return 1 whenever the respective sequences complete. As expected, the first input '1' returns an output of '10000000', meaning the first sequence, which has a length of 1, has been detected. After the next '1' is input, the first output bit is 0 and the second, corresponding to length 2, is '1'. Following the 3rd one, both bits 1 and 3 return '1', which is correct because the three bit sequence has completed, and the single bit sequence has completed once again. After the fourth '0' is input, the total serial stream to this point has been '1010'. The second and fourth bits both return '1'. The fifth input '1' returns '1' in the fifth and first bits. It can be seen that any input '1' will return '1' in bit 1 due to the method it was programmed. This shows that there are several possible creative uses if this code if programmed a certain way. The correct bits are set for the 6th, 7th, and 8th inputs too as seen from the simulation. Also, it can be seen that the output is following the clock rising edge, which is important.

Conclusion

The Circuit implemented in this project was a programmable variable length behavioral simulation of a sequence detector. This was implemented by modeling the state diagram using a case statement to set the state, and if statements to determine the output and next state. The behavioral model, while accomplishing the same task as a structural model, was much more efficient and easier to implement. While a hard-coded sequence detector is fairly easy to implement structurally, the programmable sequence introduces programmable logic elements into the state machine design. The variable length also complicates the state diagram immensely, and was easier to add after the basic state diagram was implemented.

Once the behavioral method was chosen as the best way to model the circuit, the coding process went smoothly. VHDL syntax was still the biggest complication, as the debugger often returns ambiguous errors. The sequence detector was not functioning correctly when the function was called after any clock event. Adding an if statement so that the state machine simulation is only run on the rising edge of the clock signal fixed these errors. One limitation is it would have been more user-friendly if the length of the sequence vector was actually changed when the length variable is changed, but that implementation is not possible in VHDL. With an actual programmed GUI, this would be solved.

Also, there is no specification for what is done with the output bits once they have been written. However, this leaves the project to be more versatile and flexible. The

most important thing I learned about this project is that the behavioral implementation in VHDL greatly simplifies state machine design, compared to actual creation of a state machine circuit. Increased flexibility such as programmable sequences and variable length were easy to implement, compared to a structural implementation which would require completely redoing the circuit.

Works Cited

EEN 304 Notes - Info on state machine design
EEN 316 Notes - VHDL and ModelSim tutorials
lucidchart.com - Creation of state diagram