Digital Combination Locker

Maryam Gilsenan

Table of Contents

Introduction

The objective of this project is to design a 3 digit combination lock that could be used in a verity of applications. In this instance, it is assumed it would be used to lock a door preventing any entry to the room without the correct code sequence. The lock will be designed using a finite state machine and a synchronous design methodology.

A finite state machine is very helpful when designing a sequential circuit that has a next state logic. For the designing of the digital combination locker a FSM will allow the transitioning of states each time the operator passes in a value. After comparing each value entered by the operator to the correct code sequence and verifying the two values match, the transitioning of states will occur to verify the next input until the door is unlocked. A FSM as shown with this application proves to be a very useful tool especially for any application with an event sequence or a pattern.

To begin with, a derivation of an FSM model or a state diagram will be presented to allow for an overview understanding of the design. The diagram shows the interactions and transitions between the internal states in amore simpler format.

Specifications

Requirements

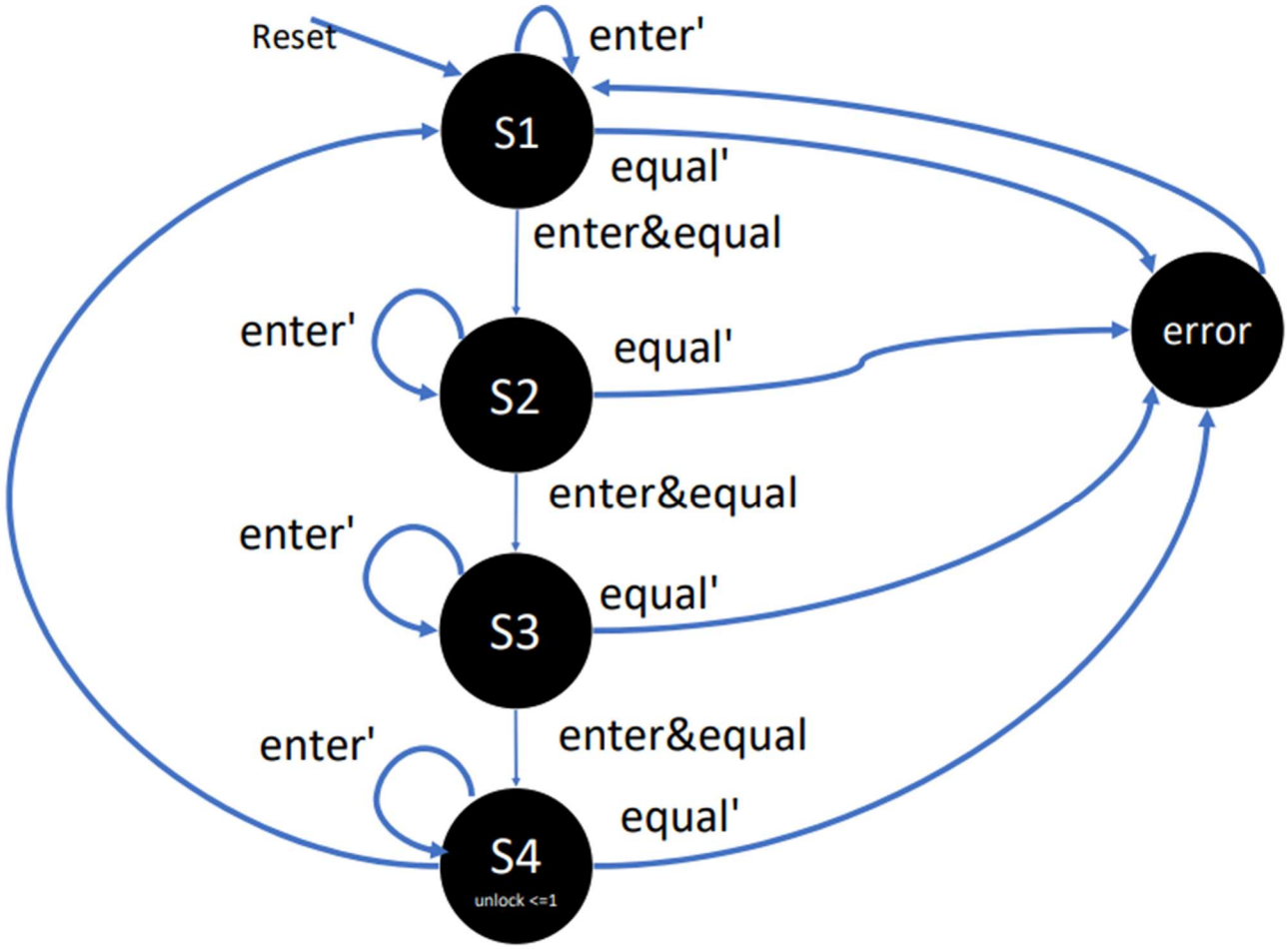
- The 3 digit combination lock must allow for a reset button, and an enter button.
- The code sequence can be loaded to the registers given the input values ld1, ld2, ld3.
- C1, C2, C3 are three registers that hold the lock combination for verification purposes
- The operator must enable enter after entering each key. That will allow for the verification of each value to occur and also move to the next state.
- The operator can return to a reset setting at any point in time.
- There are two main principle components in the circuit and that is the data path and controller.
- Only after all 3 digit combination values that are entered by the operator can the door open.

Operation

- Pressing reset begins the unlocking process.
- The operator is allowed to enter any digit in the keypad to attempt to unlock the door.
- After entering each digit, the operator will need to enable enter to move to the next state.
- The value entered is verified and compared to the value stored in the register.
- If successful, the same process will repeat until the third value is entered in which the door will become unlocked.
- If an incorrect value is entered by the operator, the 3 digit combinational locker will reset.
- For this case, a code sequence of 3, 1, 3 is assumed.
- The operator can return to the original state and restart the unlocking attempt by enabling reset.

Design

State Diagram



Code Organization

The entity of the VHDL code accepts ld1, ld2, and ld3 as inputs to load the values stored in the register. The code sequence stored in the registers are 3, 1, 3.

Enter is an input used to indicate that the operator has completed the entering of the value of the code sequence. Enabling enter after each entry of the value from the code sequence will allow for FSM to move to the next state.

Reset is an input value the will restart the process of unlocking. Resetting can happen at any point in time.

```
library ieee;
use ieee.std_logic_1164.all;
entity comb_lock is
port(
        ld1 : IN STD_LOGIC;
        ld2 : IN STD_LOGIC;
        ld3 : IN STD_LOGIC;
        enter : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        value : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        clk : IN STD_LOGIC;
        unlock : Out STD_LOGIC);
end comb_lock ;
```

Value is an input value for a user to enter the desired four bit digit. The value entered by the operator will be compared to the values stored in the register to verify the entry is correct.

Unlock is the output value that indicated the successful completion of the unlocking process and that the sequence of values entered by the operator is correct.

```
architecture comb_lock_arch of comb_lock is
type state_type is (s1, s2, s3, s4, error);
signal state_reg, state_next: state_type;
signal C1, C2, C3: std_logic_vector(3 downto 0);
```

The architecture is named comb_lock_arch.

The states declared in the architecture is s1, s2, s3, s4, and error.

As any FSM there is a signal of type state_type that stores the state of the state register *state_reg* and a next state *state_next.*

Three std logic vectors are also declared C1, C2, C3. This is where the stored values in the register will be loaded into for the comparison process with the input values.

```
process(ld1, ld2, ld3)
begin

        if ld1='1' then
                C1 <= "0011";
        elsif ld2='1' then
                C2 <= "0001";
        elsif ld3='1' then
                C3 <= "0011";
        end if ;
end process;
```

A process will the ld1, ld2, and ld3 signals is added.

When one of the ld signals are set to 1, the stored code sequence will be loaded into the signals C1, C2, C3.

```
-- next-state logic
process(state_reg, enter)
    begin
    case state_reg is
    when s1 =>
        if enter= '1' then
        --compare c1 to 011 (value), then assign next state
            if C1 = value then
             state_next <= s2;
            else
             state_next <= error;
            end if;
        else
         state_next <= s1;
        end if;
```

For the next state logic, there are five possible states for the finite state machine.

S1 is declared to be the first state. There are a few possible scenarios. The first being the state waiting to transition based on the input of the operator. If enter is enabled, then the value entered would be compared to the already stored value in the register. If correct it will enter the next state. If incorrect it will move to the error state as the state_next.

Otherwise, if enter has not been enabled, the state will remain the same until changed by the operator.

```
when s2 =>
    if enter= '1' then

    --compare c2 to 001 (value), then assign next state
        if C2 = value then
         state_next <= s3;
        else
         state_next <= error;
        end if;
    else
     state_next <= s2;
    end if;


when s3 =>
    if enter= '1' then
    --compare c3 to 011 (value), then assign next state
        if C3 = value then
         state_next <= s4;
        else
         state_next <= error;
        end if;
    else
     state_next <= s3;
    end if;

when s4 =>
    if enter= '1' then
     state_next <= s1;
    else
     state_next <= s4;
    end if;
```

Again a similar process will be repeated until it the process reaches the state s4 in which the door will become unlocked.

```
    when error =>
         state_next <= s1;
    end case;
end process;
```

In the case there is an error, the state in the FSM will move to the error state. Once in the error state, the process will return to the very beginning which is s1. The process again is repeated until no error is detected and the door can be unlocked.

```
-- moore output logic
unlock <= '1' when state_reg=s4 else '0';

end comb_lock_arch;
```

The moore output logic checks if the state register has reached state s4. If that is the case, the process is terminated successfully and the door becomes unlocked.

Controller: multi-segment VHDL code for the FSM

For the purpose of the controller, the code written is a multi-segment VHDL code. The state register, the next state register, and the output logic follow the description of a multi-segment VHDL code.

Data path: Process Statements

The data path segment in the code where the values are loaded into C1, C2, and C3 when its corresponding ld1, ld2, and ld3 are enabled use process statements for the assignment process.

Simulation

Compiling the code

Before compiling the code, a folder must be created to store the vhdl to run the program.

The folder can be created using *mkdir* or accessed using a *cd* command.

Once in the folder, the simulator modelsim can be run on the command terminal using the following command.

>> source /CMC/ENVIRONMENT/modelsim.env

```
[grace] [/home/m/m_gilsen/COEN313/Code] > source /CMC/ENVIRONMENT/modelsim.env
```

To compile the code use the command

>> vcom <<yourvhdlfile>>

```
[grace] [/home/m/m_gilsen/COEN313/Code] > vcom coen313project.vhd
```

If compiled successfully, you will see the following message:

```
[grace] [/home/m/m_gilsen/COEN313/Code] > vcom coen313project.vhd
Model Technology ModelSim SE-64 vcom 6.6g Compiler 2012.05 May 23 2012
-- Loading package standard
-- Loading package std_logic_1164
-- Compiling entity comb_lock
-- Compiling architecture comb_lock_arch of comb_lock
```

To run the simulator, the following command must be entered.

>>vsim -c comb_lock

A successful simulation will produce the following message:

```
[grace] [/home/m/m_gilsen/COEN313/Code] > vsim -c comb_lock
Reading /nfs/sw_cmc/x86_64.EL7/tools/mentor.2011/modelsim_6.6g/modeltech/tcl/vsim/pref.tcl

# 6.6g

# vsim -c comb_lock
# ** Note: (vsim-3812) Design is being optimized...
# //  ModelSim SE-64 6.6g May 23 2012 Linux 3.10.0-1160.6.1.el7.x86_64
# //
# //  Copyright 1991-2012 Mentor Graphics Corporation
# //             All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND
# //  PROPRIETARY INFORMATION WHICH IS THE PROPERTY
# //  OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
# //  AND IS SUBJECT TO LICENSE TERMS.
# //
# Loading std.standard
# Loading ieee.std_logic_1164(body)
# Loading work.comb_lock(comb_lock_arch)#1
#
#
```