

**Department of Electrical Engineering**

**Lab Report 7: Digital Lock**

Name: Socheath Sok, ID: 014470701

Partner: Ashley Tran

Group: 11

Professor: Dr. Ray Wang

Class: EE 301

Date Performed: December 12, 2017

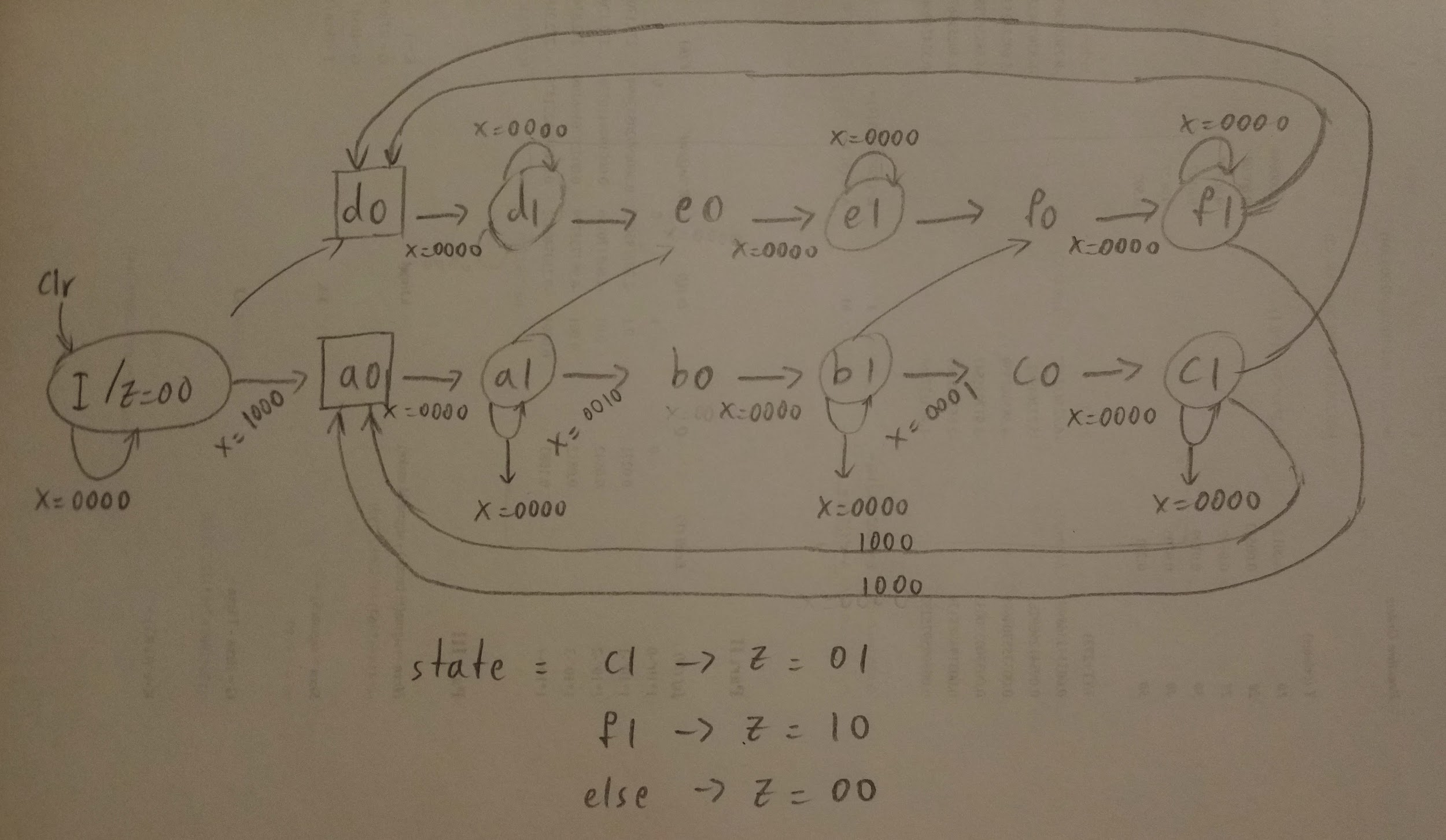
**Abstract**

The goal of this lab is to create a digital lock that has a three digit secret code (3-1-0) that are needed to be pressed in that exact order for the lock to open. The secret code can be entered using the push buttons provided on the board. While the code is being entered, all LEDs will be off. When a correct code is entered, an LED will light up. If the code is incorrect, a different LED will light to represent that incorrect state. For this design, a state diagram is needed to describe the functionality of the lock and the code for the lock combined with the provided debounce code were used as the components in a structural code. Additionally, a clear input is also included for resetting the state and a clock input is to be connected to the onboard 50 Mhz FPGA clock at location B8.

**Introduction**

For the state machine diagram, we need 13 states to represent all the possibility needed for the three digits secret code. Other than the initial state (‘I’), each digit entered requires two states to represent it. When a digit is entered, the state will move to the zero state, for example, A0(if the digit is correct), and with no change in input, the state will move automatically to the one state, which is A1 in this case. However, to move to the next zero state, a new input is required and this process will continue until the state reaches the final correct or incorrect state. All the states to the right of I are the correct states with the final state C1 that will signal the correct LED to light up. In the case that any digit is wrong, the state will move up to its equivalent error state in the top row and the final error state F1 will signal the wrong LED to light up. The complete state machine diagram for the digital lock with three digit code is shown below.

**State Machine Diagram for the Lock**



With the diagram completed, we moved on to create the VHDL code for the lock. The output of the lock will show when the clock is positive and the clear input will reset the state back to the starting state ‘I’. The changes in states were described in the ‘case’ statement with the help of the ‘if-elsif-else-end’ command. At the end of the case statements, we assigned the LEDs represents by output z (0:1) to the appropriate states. With this design, one LED is guarantee to light up after three digits are entered and the process can continue infinitely as more inputs are entered or stop and reset back at any time using the clear input. The code for the lock is shown in below.

**VHDL code for the Lock Design**

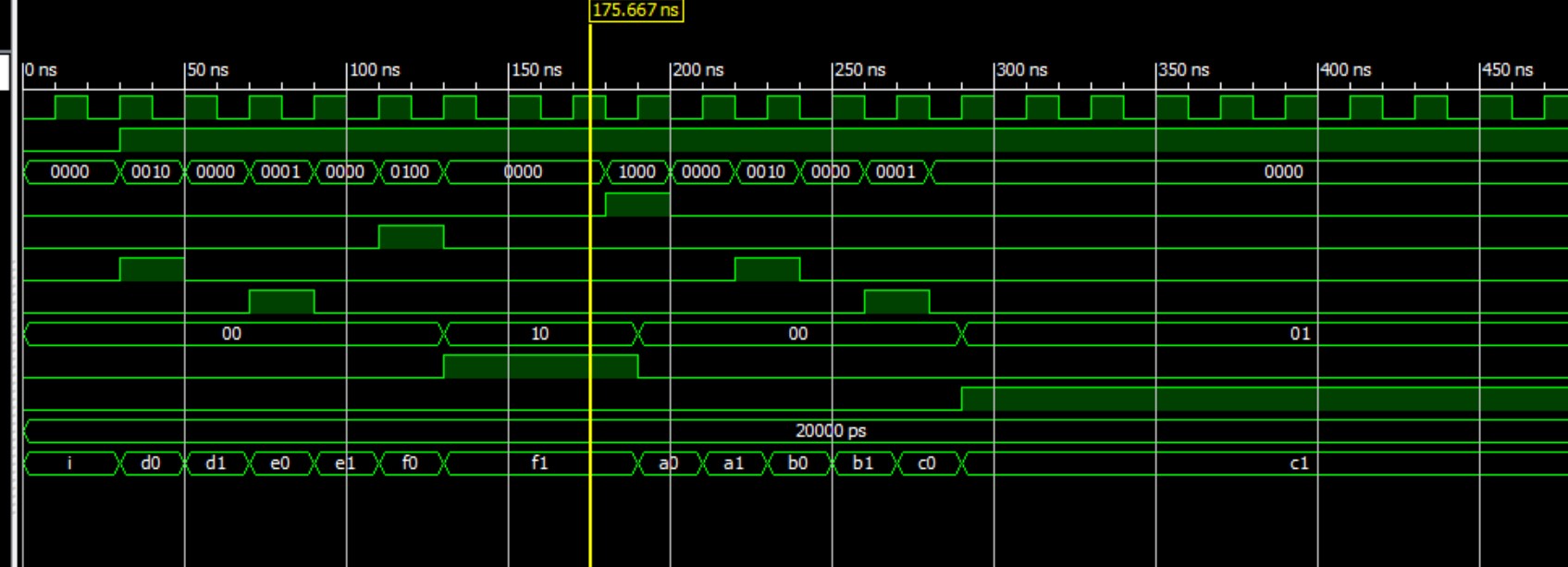
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity Lock is  Port ( clk : in STD\_LOGIC;  clr : in STD\_LOGIC;  x : in STD\_LOGIC\_VECTOR (3 downto 0);  z : out STD\_LOGIC\_VECTOR (1 downto 0));  end Lock;  architecture Behavioral of Lock is  TYPE State\_type is(I,A0,A1,B0,B1,C0,C1,d0,d1,e0,e1,f0,f1) ;  signal y : state\_type;  begin  process (clk,clr)  begin  if clr='0' then y <= I;  elsif (clk' event and clk = '1') then  case y is  when I =>  if x = "0000" then y <= I;  elsif x = "1000" then y <= A0;  else y <= d0;  end if;  when A0 =>  if x = "0000" then y <= A1;  else y <= A0;  end if; when A1 => if x = "0000" then y <= A1;  elsif x = "0010" then y <= B0; else y <= e0; end if; when B0 => if x = "0000" then y <= B1; else y <= B0; end if; when B1 => if x = "0000" then y <= B1; elsif x = "0001" then y <= C0;  else y <= f0;  end if; | when C0 =>  if x = "0000" then y <= C1;  else y <= C0;  end if;  when C1 =>  if x = "0000" then y <= C1;  elsif x = "1000" then y <= A0;  else y <= d0;  end if;  when d0 =>  if x = "0000" then y <= d1;  else y <= d0;  end if;  when d1 =>  if x = "0000" then y <= d1;  else y <= e0;  end if;  when e0 =>  if x = "0000" then y <= e1;  else y <= e0;  end if;  when e1 =>  if x = "0000" then y <= e1;  else y <= f0;  end if;  when f0 =>  if x = "0000" then y <= f1;  else y <= f0;  end if;  when f1 =>  if x = "0000" then y <= f1;  elsif x = "1000" then y <= A0;  else y <= d0;  end if;  end case;  end if;  end process;  z <= "01" when y = C1 else  "10" when y = f1 else  "00";  end Behavioral; |
| --- | --- |

Next, we created a testbench code to test our lock design. Under the ‘stimulus process’, we wrote a code to test the input clear, the incorrect three digits code and the correct three digits code. We also made changes to the waveform, so that the internal signals, which represents the states, are shown under its appropriate input and output. The testbench code and the timing diagram are shown below.

**TestBench Code**

| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  use ieee.std\_logic\_arith.all;  use ieee.std\_logic\_unsigned.all;  ENTITY lock\_only\_test IS  END lock\_only\_test;  ARCHITECTURE behavior OF lock\_only\_test IS  COMPONENT Lock  PORT(  clk : IN std\_logic;  clr : IN std\_logic;  x : IN std\_logic\_vector(3 downto 0);  z : OUT std\_logic\_vector(1 downto 0));  END COMPONENT;  signal clk : std\_logic := '0';  signal clr : std\_logic := '0';  signal x : std\_logic\_vector(3 downto 0) := (others => '0');  signal z : std\_logic\_vector(1 downto 0);  constant clk\_period : time := 20 ns;  BEGIN  uut: Lock PORT MAP (  clk => clk,  clr => clr,  x => x,  z => z );  clk\_process :process  begin  clk <= '0';  wait for clk\_period/2;  clk <= '1';  wait for clk\_period/2;  end process; | stim\_proc: process  begin  -- test clr  clr<='0';  wait for 30ns;  clr<='1';  -- test using incorrect secret code  x <= "0010";  wait for 20ns;  x <= "0000";  wait for 20ns;  x <= "0001";  wait for 20ns;  x <= "0000";  wait for 20ns;  x <= "0100";  wait for 20ns;  x <= "0000";  wait for 30 ns;  -- test using correct secret code  x <= "1000";  wait for 20ns;  x <= "0000";  wait for 20ns;  x <= "0010";  wait for 20ns;  x <= "0000";  wait for 20ns;  x <= "0001";  wait for 20ns;  x <= "0000"; wait;  end process;  END; |
| --- | --- |

**Timing Diagram**



Once the lock code is proven to be correct, we created a new module for the debounce code, provided as part of the instruction. With both component modules created, we created another module for the structural code of our digital lock. The structural code consists of six inputs: 4 for the push buttons (btn (0:3)), 1 for the switch representing the clear input and 1 for the onboard clock. The outputs, on the other hands, are just z (0:1), which are connected to the LEDs. btn (0:3) is connected to the input debounce module and z (0:1) are connected to the output of the lock module. A wire signal is also needed to link the output of the debounce module to the lock module. Lastly, the clear and the clock inputs are connected to both modules. The debounce code, the structural code and the constraint file are shown below.

**Debounce Code**

| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity debounce is  Port ( clock, clr: in STD\_LOGIC;  inp : in std\_logic\_vector(3 downto 0);  outp : out STD\_LOGIC\_vector(3 downto 0) );  end debounce;  architecture Behavioral of debounce is  signal count : STD\_LOGIC\_VECTOR (17 downto 0);  signal clk190 : std\_logic;  signal delay1, delay2, delay3: std\_logic\_vector(3 downto 0);  begin  PROCESS ( clr, clock )  BEGIN  IF clr = '0' THEN  count <= (others => '0') ; | ELSIF Clock'EVENT AND Clock = '1' THEN  count <= count+1 ;  END IF ;  END PROCESS ;  clk190<=count(17);  process (clr, clk190)  begin  if clr='0' then  delay1<="0000";  delay2<="0000";  delay3<="0000";  elsif clk190'event and clk190='1' then  delay1 <=inp;  delay2 <=delay1;  delay3<=delay2;  end if;  end process;  outp<=delay1 and delay2 and delay3;  end Behavioral; |
| --- | --- |

**Structural Code**

| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity Digital\_Lock is  Port ( btn : in STD\_LOGIC\_VECTOR (3 downto 0);  clk,clr : in STD\_LOGIC;  z : out STD\_LOGIC\_VECTOR (1 downto 0));  end Digital\_Lock;  architecture Behavioral of Digital\_Lock is  component debounce  Port ( clock, clr: in STD\_LOGIC;  inp : in std\_logic\_vector(3 downto 0);  outp : out STD\_LOGIC\_vector(3 downto 0));  end component; | component Lock  Port ( clk : in STD\_LOGIC;  clr : in STD\_LOGIC;  x : in STD\_LOGIC\_VECTOR (3 downto 0);  z : out STD\_LOGIC\_VECTOR (1 downto 0));  end component;  signal wire : std\_logic\_vector (0 to 3);  begin    Q1: debounce port map(clk,clr,btn,wire);  R1: Lock port map(clk,clr,wire,z);  end Behavioral; |
| --- | --- |

**Constraint File**

NET "clk" LOC = "B8" ;

NET "clr" LOC = "P11" ;

NET "btn(0)" LOC = "G12" ;

NET "btn(1)" LOC = "C11" ;

NET "btn(2)" LOC = "M4" ;

NET "btn(3)" LOC = "A7" ;

NET "z(0)" LOC = "M5" ;

NET "z(1)" LOC = "G1" ;

**Discussion**

The digital lock created from the state diagram should work fine, theoretically, without the debounce module. However, in practice, an electrical signal from the push button often makes or break contact, thus making the design unstable. With the debounce module, the signal will remain connected and a clean transition is provided to the outputs. Doing this lab help us see how the concept of state machine diagram can be put into practice to create a design that can be used in a real world.