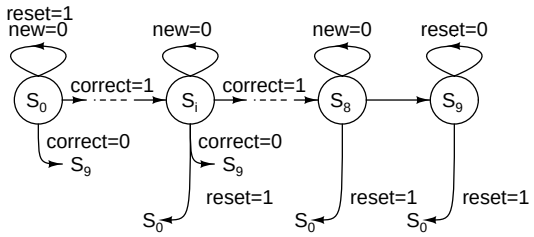
Solution for Assignment 1

The combination lock can be modelled as a state machine. From the description of its operation we can derive the following (incomplete) state transition diagram:



The state machine goes to the reset state, S_0 if the reset button is pushed. The state machine can only change state on the clock cycle in which any key is newly pressed. It steps through 8 additional states (S_1 to S_8) after each correct digit is pressed or goes to S_9 if the wrong digit is pressed. There is no 'correct' button in S_8 or S_9 so in these states the state machine always goes to S_9. The lock is opened only in state 8.

We can write the state transition table as follows:

reset	new	correct	state (S_i)	next state
0	0	X	S_i	S_i
0	1	0	S_i	S_9
0	1	1	S_i	S_{i+1}
1	X	X	S_i	S_0

where $S_i$represents any valid state (0 to 9), S_{i+1} represents the next state if the correct button is pushed, 'new' is a signal that is 1 only in the state following a new keypress, and 'correct' is a signal that is 1 when the correct key to advance to the next state is being pressed.

```
entity EC_lock is port (
        signal button : in bit_vector(9 downto 0) ;
        signal reset, clk : in bit ;
        signal locked : out bit ) ;
end EC_lock ;
```

```
architecture rtl of EC_lock is
   signal
      correct,     -- correct keypress for state
      new_button,  -- first clock for this keypress
      any,         -- any key is pressed
      prev_any     -- 'any' at previous clock
        : bit ;
   signal
      state,            -- current state
      state_if_reset,   -- state if reset asserted
      state_if_new,     -- state if new asserted
      state_if_correct, -- state if correct asserted
      state_next        -- next state if 'right' applied
        : bit_vector (3 downto 0) ;

   signal
      wrong_buttons,
      correct_buttons    -- correct buttons for state
        : bit_vector (9 downto 0) ;
begin

   -- next-state selection

   with reset select state_if_reset <=
      "0000" when '1',
      state_if_new when others ;

   with new_button select state_if_new <=
      state_if_correct when '1',
      state when others ;

   with correct select state_if_correct <=
      state_next when '1',
      "1001" when others ;

   -- alternative using selected assignment:

   -- state_if_reset <=
   --    "0000" when reset = '1' else
   --    state when new_button = '0' else
   --    state_next when correct = '1' else
   --    "1001" ;

   -- state register
   process(clk)
   begin
      if clk'event and clk = '1' then
         state <= state_if_reset ;
          end if ;
   end process ;

   -- detect if any keys are pressed
   with button select any <=
      '0' when "0000000000",
      '1' when others ;

   -- 'prev_any' is 'any' delayed by one clock
   process(clk)
   begin
```

```
      if clk'event and clk = '1' then
         prev_any <= any ;
      end if ;
   end process ;

   -- detect if a new key has been pressed
   new_button <= any and not prev_any ;

   -- next state (if correct button pushed)
   with state select state_next <=
      "0001" when "0000",
      "0010" when "0001",
      "0011" when "0010",
      "0100" when "0011",
      "0101" when "0100",
      "0110" when "0101",
      "0111" when "0110",
      "1000" when "0111",
      "1001" when "1000",
      "1001" when others ;

   -- determine if the correct key is being
   -- pressed for current state

   with state select correct_buttons <=
   --  9876543210       state
      "0000000001" when "0000",
      "0000000010" when "0001",
      "0000000100" when "0010",
      "0000001000" when "0011",
      "0000010000" when "0100",
      "0000100000" when "0101",
      "0001000000" when "0110",
      "0010000000" when "0111",
      "0000000000" when others ;

   wrong_buttons <= button xor correct_buttons ;

   with wrong_buttons select correct <=
        '1' when "0000000000" ,
        '0' when others ;

   -- assert output except in state 8
   with state select locked <=
      '0' when "1000",
      '1' when others ;

end rtl ;
```

The corresponding block diagram is shown if Figure 1.

The simulation output for the three test cases is shown in figures 2 through 4.

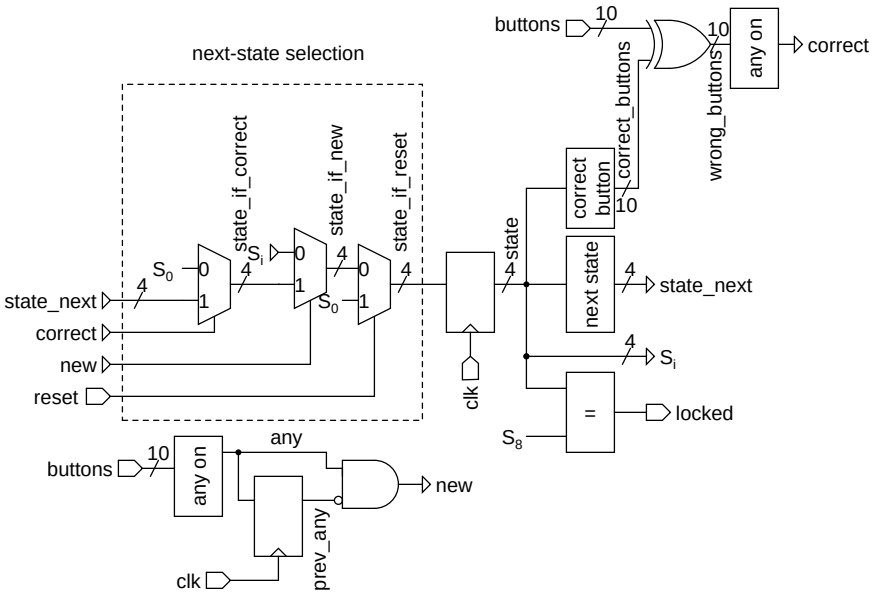


Figure 1: Block Diagram.

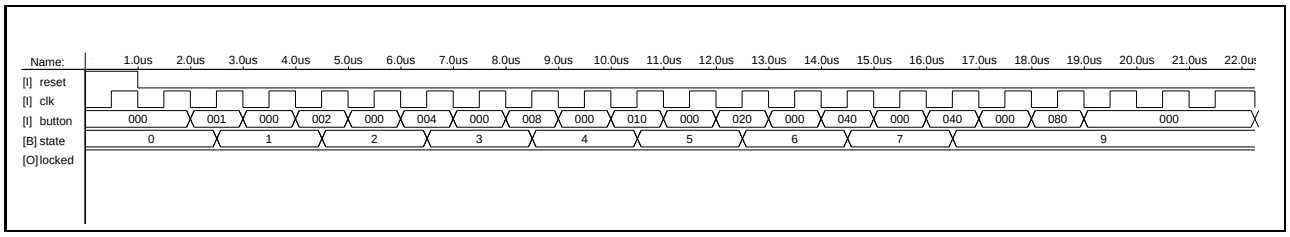


Figure 2: Incorrect Sequence.

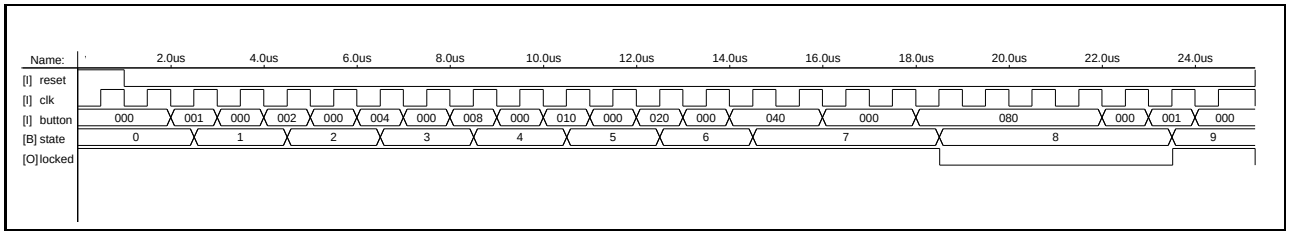


Figure 3: Correct Input.

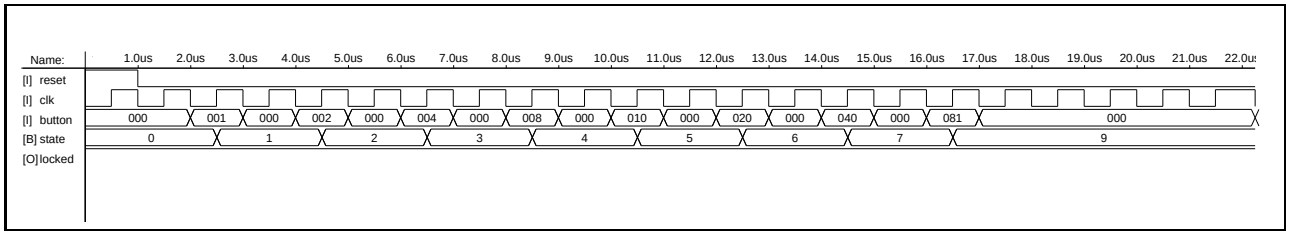


Figure 4: Two Keys at Once.