Practice Quiz 3 - Solutions

Solution

(a) From the requirements there are three combinations of output values required:

(i) `raise=1` and `lower=0`

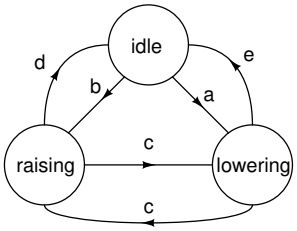
(ii) `raise=0` and `lower=1`

(iii) `raise=0` and `lower=0`

and these may be sufficient to implement the state machine. We can pick names for these states. For example: `raising`, `lowering` and `idle` respectively.

You may have decided to use the position of the door and/or whether it was in motion as a state. While these are obvious choices for the *door* state, they may not necessarily correspond to the controller's states.

(b) The state transitions are derived from the description of the controller's functionality and are show in the diagram below.



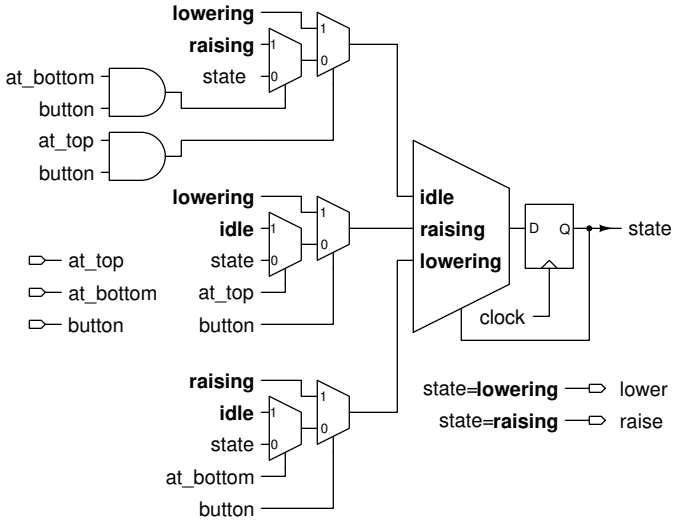
(a) at_top and button
(b) at_bottom and button
(c) button
(d) at_top
(e) at_bottom

(c) In this case the outputs are a function only of the state:

```
raise <= '1' when state = raising else '0' ;

lower <= '1' when state = lowering else '0' ;
```

(d) A schematic of the controller would be as follows:



From the schematic we can derive the VHDL:

```
-- practice quiz 3

library ieee ;
use ieee.std_logic_1164.all ;


entity quiz3a is
  port (
    at_top, at_bottom, button: in std_logic;
    clock: in std_logic;
    lower, raise: out std_logic ) ;
end ;

architecture rtl of quiz3a is

  type state_t is ( idle, raising, lowering ) ;
  signal state, state_next : state_t ;
  signal idle_next, raising_next, lowering_next: state_t ;

begin

  idle_next <= lowering when at_top and button else
               raising when at_bottom and button else
               state ;

  raising_next <= lowering when button else
                  idle when at_top else
                  state ;

  lowering_next <= raising when button else
                   idle when at_bottom else
                   state ;

  with state select state_next <=
    idle_next when idle,
    raising_next when raising,
    lowering_next when lowering ;

  state <= state_next when rising_edge(clock);
```

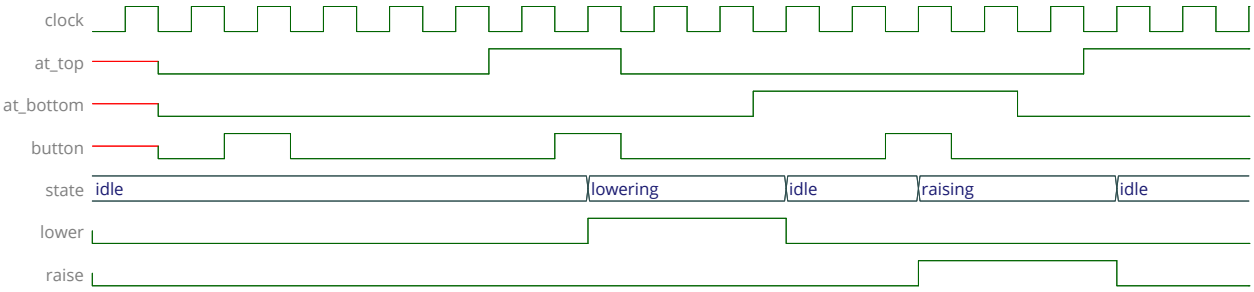


Figure 1: Simulation Results

```
  -- outputs

  lower <= '1' when state = lowering else '0' ;
  raise <= '1' when state = raising else '0' ;

end rtl ;
```

To check the solution we can simulate the design using a testbench such as:

```
-- testbench for practice quiz 3 solution

library ieee;
use ieee.std_logic_1164.all;

entity quiz3a_tb is

end ;

architecture rtl of quiz3a_tb is
  signal at_top, at_bottom, button:  std_logic;
  signal clock:  std_logic := '0' ;
  signal lower, raise: std_logic ;
  signal done: boolean := false ;
begin

  dut: entity work.quiz3a port map (
    at_top, at_bottom, button, clock,
    lower, raise ) ;

  process
    type tv_t is array (natural range <>) of
      std_logic_vector (2 downto 0) ;
    variable tv: tv_t (0 to 16) := (
      "000", "001", "000", "000", "000",
      "100", "101", "000", "000", "010",
      "010", "011", "010", "000", "100",
      "100", "100" ) ;
  begin

    for i in tv'low to tv'high loop
      wait until falling_edge(clock);
      (at_top, at_bottom, button ) <= tv(i) ;
    end loop ;

    done <= true ;
  end process ;

 clock <= not clock after 0.5 us when not done ;

end rtl ;
```

for which the simulation results are shown in Figure 1.

Other solutions are possible. For example, the four-state solution described above with two idle states (one with the door at the top and one with the door at the bottom).

This design has a flaw which shows up in the first part of the simulation results. Can you spot the problem? How could you fix it?