VHDL Testbench for ModelSim

Template - program action

The lock opens when the key "1" is pressed and then released.

Keypad and Statecounter

Good choice of data types makes the code self-explanatory!

K: in std_logic_vector(1 to 3);
R: in std_logic_vector(1 to 4);

K(3) = '1' bit
R(4) = '1' bit

Statecounter: q = "00001" bitvector
q(0) = '1' bit
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity codeclock is
  port( clk : in std_logic;
        K : in std_logic_vector(1 to 3);
        R : in std_logic_vector(1 to 4);
        q : out std_logic_vector(4 downto 0);
        Unlock : out std_logic);
end codeclock;

architecture behavior of codeclock is
  subtype state_type is integer range 0 to 31;
signal state, nextstate : state_type;
begin
  nextstate_decoder : -- next state decoding part
    process(state, K, R)
    begin
      case state is
        when 0 => if (K = '000' and R = '0000') then nextstate := 1;
                 else nextstate := 0;
                 end if;
        when 1 => if (K = '000' and R = '0001') then nextstate := 1;
                 else if (K = '001' and R = '0000') then nextstate := 2;
                  else nextstate := 0;
                  end if;
        when 2 to 30 => nextstate := state + 1;
        when 31 => nextstate := 0;
      end case;
      q <= nextstate;
    end process;
  end architecture;

-- display the state
debug_output : -- display the state
q <= conv_std_logic_vector(state, 5);

This code is given

It's easy to see that this is correct!

lockmall_with_error.vhd

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity codeclock is
  port( clk : in std_logic;
        K : in std_logic_vector(1 to 3);
        R : in std_logic_vector(1 to 4);
        q : out std_logic_vector(4 downto 0);
        Unlock : out std_logic);
end codeclock;

architecture behavior of codeclock is
  subtype state_type is integer range 0 to 31;
signal state, nextstate : state_type;
begin
  nextstate_decoder : -- next state decoding part
    process(state, K, R)
    begin
      case state is
        when 0 => if (((R(3)=0') and (R(3)=0') and (R(2)=0')) and (R(5)=1') and
                     ( (not ((K(1)=1')) and (R(1)=0')) and (R(4)=1')))) and
               ( not ((R(3)=1') and (R(1)=1')) and (R(4)=0'))))
          then nextstate := 1;
          else nextstate := 0;
          end if;
        when 1 => if (((R(3)=0') and (R(3)=0') and (R(2)=0')) and (R(5)=1') and
                     ( (not ((K(1)=1')) and (R(1)=0')) and (R(4)=1')))) and
               ( not ((R(3)=1') and (R(1)=1')) and (R(4)=0'))))
          then nextstate := 1;
          else nextstate := 0;
          end if;
        when 2 to 30 => nextstate := state + 1;
        when 31 => nextstate := 0;
      end case;
      q <= nextstate;
    end process;
  end architecture;

-- display the state
debug_output : -- display the state
q <= conv_std_logic_vector(state, 5);

output_decoder : -- output decoder part
    process(state)
    begin
      case state is
        when 0 to 1 => Unlock <= '0';
        when 2 to 31 => Unlock <= '1';
        end case;
    end process;

codeclock_register : -- the state register part (the flipflops)
    process(clk)
    begin
      if rising_edge(clk) then
        state <= nextstate;
        end if;
    end process;

begin
  if rising_edge(clk) then
    state <= nextstate;
    end if;
end behavior;

Now it's hard to see if this is correct or not?
Does both expressions mean the same?

( K = "100" and R ="0001" )

Is this really the same thing?

(((R(2)='0') and (R(3)='0') and (K(2)='0') and (K(3)='1'))) and
(not (( not ((K(1)='0') and (R(1)='0') and (R(4)='1'))) and
(not ((K(1)='1') and (R(1)='1') and (R(4)='0'))) )))

Someone "promises" that the code is correct - but how can you know that this is absolutely true?

\[ \text{tb_lockmall.vhd} \]

We need to write a VHDL-testbench

A test bench program can test all the possible key combinations and report if there is a problem ...

It can automatically loop through all possible key-presses and report on whether the lock trying to open or not.

There are \(2^7 = 128\) possible key combinations and we'd be totally exhausted if we tried to test them all by hand.

\textbf{entity - a testbench has no ports}

\begin{verbatim}
entity tb_codelock is
  -- entity tb_codelock has no ports
  -- because it's for simulation only
end tb_codelock;
\end{verbatim}
Some internal signals are needed

```vhdl
signal clk : std_logic := '0';
signal K_test : std_logic_vector(1 to 3);
signal R_test : std_logic_vector(1 to 4);
signal prev_K_test : std_logic_vector(1 to 3);
signal prev_R_test : std_logic_vector(1 to 4);
signal q : std_logic_vector(4 downto 0);
signal unlock : std_logic;
```

Our codelock is used as a component

-- we use our codelock as a component
```vhdl
component codelock
  port( clk : in std_logic;
       K : in std_logic_vector(1 to 3);
       R : in std_logic_vector(1 to 4);
       q : out std_logic_vector(4 downto 0);
       UNLOCK : out std_logic );
end component;
```

Generate a simulation clock

-- generate a simulation clock
```vhdl
clk <= not clk after 10 ns;
```

Instantiation and signal mapping

-- instantiation of the device under test,
-- mapping of signals
```vhdl
inst_codelock:
codelock
  port map (
    clk => clk,
    K => K_test,
    R => R_test,
    q => q,
    UNLOCK => unlock );
```
Two nested loops creates keystrokes

```vhdl
process
begin
  for k in 0 to 7 loop
    K_test <= conv_std_logic_vector(k,3);
    for r in 0 to 15 loop
      prev_K_test <= K_test;
      prev_R_test <= R_test;
      R_test <= conv_std_logic_vector(r,4);
      wait until CLK='1';
    end loop;
  end loop;
end process;
```

8·16=128 turns

report, severity note, severity error

Tests if state \( q = "00001" \) will be reached by any combination.

```vhdl
check:
process(q)
begin
  if ((q = "00001") and
      (prev_K_test = conv_std_logic_vector(1,3)) and
      (prev_R_test = conv_std_logic_vector(1,4)))
    then assert false report
      "Lock tries to open for the right sequence!"
      severity note;
  else if ((q = "00001"))
    then
      assert false report
      "Lock tries to open with the wrong sequence!"
      severity error;
    else report "Lock closed!" severity note;
  end if;
end process check;
```

**Simulate and find the error!**

What else besides pressing the "1" key could open the lock?