```
---------------------------------------------------------------------------------
-- Name: Capt Jeff Falkinburg
-- Date: Spring 2016
-- Course: ECE 281
-- File: Lsn20_stoplight.vhd
-- HW:   Lecture 20
-- Purp: Basic Stoplight State Machine
-- Doc:  None
-- Academic Integrity Statement: I certify that, while others may have
-- assisted me in brain storming, debugging and validating this program,
-- the program itself is my own work. I understand that submitting code
-- which is the work of other individuals is a violation of the honor
-- code.  I also understand that if I knowingly give my original work to
-- another individual is also a violation of the honor code.
---------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity stoplight is
    Port ( C : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           R : out  STD_LOGIC;
           Y : out  STD_LOGIC;
           G : out  STD_LOGIC);
end stoplight;

architecture Behavioral of stoplight is

type stoplight_state is (red, green, yellow);
signal next_state, current_state : stoplight_state;

begin

--next state logic
   process (C, current_state)
   begin
      case current_state is
         when red =>
            if C = '1' then
               next_state <= green;
            else
               next_state <= red;
            end if;
         when green =>
            if C = '1' then
               next_state <= green;
            else
               next_state <= yellow;
            end if;
         when yellow =>
            next_state <= red;
         when others =>
            next_state <= red;
      end case;
   end process;

```

```
--Alternate next state logic
-- next_state <=  green when (current_state = red and C = '1') else
--                red when (current_state = red and C = '0') else
--                green when (current_state = green and C = '1') else
--                yellow when (current_state = green and C = '0') else
--                red when (current_state = yellow) else
--                red;

-- state memory

   process (clk, reset) -- Only the state flip-flop needs to be in a process
   begin                -- But we went through a case example for continuity
      if reset = '1' then
         current_state <= yellow;  -- Reset into yellow
      elsif (rising_edge(clk)) then
         current_state <= next_state;
      end if;
   end process;

-- output logic

   R <= '1' when current_state = red else '0';
   G <= '1' when current_state = green else '0';
   Y <= '1' when current_state = yellow else '0';


end Behavioral;
```