Problem Set 5 Solutions

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;

-- here is the declaration of entity

entity la_rewarder is
 port (clk,  go, SRAM_busy, SRAM_rdy: in std_logic;
       min: buffer std_logic_vector(2 downto 0);
       max: buffer std_logic_vector(2 downto 0);
       staff_hours_adr: out std_logic_vector(2 downto 0);
       staff_hours : in std_logic_vector(2 downto 0);
       minmax_start, minmax_done : buffer std_logic;
       timeoff : buffer std_logic_vector(3 downto 0);
       rdy : out std_logic);
end la_rewarder;


--the assumption is that the payroll module functions in such way to provide
--a signal SRAM_busy to denote that it is updating the system and SRAM_rdy
--when the data on the bus is valid (corresponding to the values stored at
--staff_hours_adr).

-- here is the body of the architecture

architecture state_machine of la_rewarder is

-- first declaration of all states
type StateType is (idle,idle2, rew_LA1, rew_LA12, rew_LA2, rew_LA22, rew_LA3,
rew_LA32, rew_LA4, rew_LA42, rew_finish);

 signal present_state, next_state : StateType;
 signal LA_points1 : std_logic_vector(1 downto 0);
 signal LA_points2 : std_logic_vector(1 downto 0);
 signal LA_points3 : std_logic_vector(1 downto 0);
 signal LA_points4 : std_logic_vector(1 downto 0);

  component maxmin
      port(
      clk, SRAM_busy, SRAM_rdy : in std_logic;
      minmax_start : buffer std_logic;
      min : buffer std_logic_vector(2 downto 0);
      max : buffer std_logic_vector(2 downto 0);
      staff_hours : in std_logic_vector(2 downto 0);
      minmax_done : buffer std_logic);
      end component;
    begin
     process(present_state, clk, min, max, SRAM_busy, go, timeoff, LA_points1,
LA_points2, LA_points3,
     LA_points4, SRAM_rdy, minmax_done,min, max, staff_hours)
      begin
      timeoff <= "0000";      --start off with timeoff being zero for all LA's
            rdy <= '0';              -- start off with the output not calculated
            minmax_start <= '0';
        case present_state is
          when idle =>
           timeoff <= "0000";       -- in idle, no LA gets time off
```

```
                              --in idle, any address is asserted
           rdy <= '0';        --the timeoff is not calculated yet
           minmax_start <= '0';
           if (SRAM_busy = '1') then
           next_state <= idle;      --SRAM_busy means that the payroll is
updating the SRAM table
                        --of weekly hours
           timeoff <= "0000";
           rdy <= '0';

           elsif (minmax_done = '0' and go <= '1') then --this is the state
where the payroll has updated
                                          --the records, but the minor fsm
                                          --has not calculated the new min and
max values;
                                          --min and max values have to be
                                          --calculated after each payroll
update.

           minmax_start <= '1';                 --the major fsm asserts the
start signal that the
                                          --minor fsm uses to see when it needs
                                          --to perform the next calculation.

           next_state <= idle2;                       --next_state = wait for
minmax to calculate min and
           rdy <= '0';                          --max values no output is ready
at this point
           timeoff <= "0000";

           elsif (minmax_done = '1' and go = '1') then      --this is the state
when the payroll has finished
                                          --updating the records and the
                                                --minmax has finished
calculating the min and max
           next_state <= rew_LA1;                     --values next state checks
the first LA's points,
                                          --LA corresponds to 011 in the table
           staff_hours_adr <= "011";
           timeoff <= "0000";
           rdy <= '0';
           minmax_start <= '0';
           else
           next_state <= idle;                        --safe bet to play is that
if we are getting anything
                                          --else to return to the idle
                                          --state
           end if;

          when idle2 =>                         --this is the sate where we wait
for the minor minmax
                                          --fsm
          timeoff <= "0000";
          rdy <= '0';
          minmax_start <= '1';

          if(minmax_done = '0') then
          next_state <= idle2;
          else
          next_state <= rew_LA1;                --minmax is done - go to the
next state and check the
                                          --LA points for the first LA
          staff_hours_adr <= "011";
          timeoff <= "0000";
          rdy <= '0';
          minmax_start <= '0';                        --the LA we are checking
for is 011
          end if;


        when rew_LA1 =>
        staff_hours_adr <= "011";
        rdy <= '0';
        timeoff <= "0000";
        minmax_start <= '0';
        if (SRAM_rdy = '0') then                --we abstract from the SRAM
interface and use
                                          --SRAM_rdy signal to denote when
                                          --the valid data (or LA hours) is

asserted on the bus
        staff_hours_adr <= "011";
        next_state <= rew_LA1;
        else
        next_state <= rew_LA12;
        staff_hours_adr <= "011";
        end if;

        when rew_LA12 =>
        rdy <= '0';
        timeoff <= "0000";
        if(LA_points1 = 11) then                --if the LA has accumulated 3
points (highly unlikely ;) )

                                          --then he gets time
                                          --off. For LA1 that means asserting

bit0 or equivalently
                                          --adding 1 to the timeoff
                                          --value.
                                          --reminder - it is possible that an LA

will lose a point
                                          --at LA_points = 3
        timeoff <= timeoff+1;                   --reminder - to keep things
simple, it is possible that
                                          --an LA will miss
        LA_points1 <= "00";                     --we reset his points to zero
        next_state <= rew_LA2;                  --transition into the next state
and check LA2
        elsif (staff_hours < min) then          --if the LA has not worked the
required hours (very likely),
                                          --then he loses a
                                          --point in his LA_points vector

        LA_points1 <= LA_points1 -1;
        rdy <= '0';
        timeoff <= timeoff;                     --the timeoff value does not
change because he does not
```

```
                                          --have enough poitns
        next_state <= rew_LA2;
        elsif (staff_hours > max) then          --if he worked harder than
predicted (never happens :) )
                                          --then he gets a point
                                          --in his LA_points vector
        LA_points1 <= LA_points1+1;
        timeoff <= timeoff;                     --the timeoff value remains what
it was when we entered
                                          --the state, because the
                                          --LA didn't have a sufficient amount
of points
        rdy <= '0';
        next_state <= rew_LA2;
        else
        LA_points1 <= LA_points1;               --if he worked just enough, then
the points and timeoff
                                          --do not change.
        rdy <= '0';
        timeoff<= timeoff;
        next_state <= rew_LA2;
        end if;

        when rew_LA2 =>
        rdy <= '0';                             --we move to the next LA
        staff_hours_adr <= "100";               --his adress is 100
        timeoff <= timeoff;
        if (SRAM_rdy = '0') then                --wait until the hours are on
the bus
        staff_hours_adr <= "100";
        rdy <= '0';
        timeoff <= timeoff;
        next_state <= rew_LA2;                  --if the SRAM values aren't
ready, remain in the state
        else
        next_state <= rew_LA22;                 --otherwise transition into the
next state to compare
                                          --the obtained values
        end if;

        when rew_LA22 =>
        rdy <= '0';
        timeoff <= timeoff;
        if(LA_points2 = 11) then                --if he has accumulated 3
points, reward him by giving
                                          --him some time off
        timeoff <= timeoff+2;                   --for LA2 this is done by
incrementing timeoff by 2.
        LA_points2 <= "00";                     --set his points to 00 for the
next cycle; again he may
                                          --lose a point b/c
                                          --we chose not to check his hours
against the max value
        next_state <= rew_LA3;                  --go and check the values for
LA3
        elsif (staff_hours < min) then          --if he didn't work enough then
he loses a point
```

```
LA_points2 <= LA_points2 -1;
rdy <= '0';
timeoff <= timeoff;                     --and doesn't get any time off
next_state <= rew_LA3;                  --we go to the next LA
elsif (staff_hours > max) then
LA_points2 <= LA_points2 + 1;
timeoff <= timeoff;
rdy <= '0';
next_state <= rew_LA3;
else
LA_points2 <= LA_points2;
rdy <= '0';
timeoff<= timeoff;
next_state <= rew_LA3;
end if;

when rew_LA3 =>
rdy <= '0';
staff_hours_adr <= "101";
timeoff <= timeoff;
if (SRAM_rdy = '0') then
staff_hours_adr <= "101";
rdy <= '0';
timeoff <= timeoff;
next_state <= rew_LA3;
else
next_state <= rew_LA32;
end if;

when rew_LA32 =>
rdy <= '0';
timeoff <= timeoff;
if(LA_points3 = 11) then
timeoff <= timeoff+4;
LA_points3 <= "00";
next_state <= rew_LA4;
elsif (staff_hours < min) then
LA_points3 <= LA_points3 -1;
rdy <= '0';
timeoff <= timeoff;
next_state <= rew_LA4;
elsif (staff_hours > max) then
LA_points3 <= LA_points3 + 1;
timeoff <= timeoff;
rdy <= '0';
next_state <= rew_LA4;
else
LA_points3 <= LA_points3;
rdy <= '0';
timeoff<= timeoff;
next_state <= rew_LA4;
end if;

when rew_LA4 =>
rdy <= '0';
staff_hours_adr <= "110";
timeoff <= timeoff;
```

```
        if (SRAM_rdy = '0') then
        staff_hours_adr <= "110";
        rdy <= '0';
        timeoff <= timeoff;
        next_state <= rew_LA4;
        else
        next_state <= rew_LA42;
        end if;

        when rew_LA42 =>
        rdy <= '0';
        timeoff <= timeoff;
        if(LA_points4 = 11) then
        timeoff <= timeoff+8;
        LA_points4 <= "00";
        next_state <= rew_finish;
        elsif (staff_hours < min) then
        LA_points4 <= LA_points4 -1;
        rdy <= '0';
        timeoff <= timeoff;
        next_state <= rew_finish;
        elsif (staff_hours > max) then
        LA_points4 <= LA_points4 + 1;
        timeoff <= timeoff;
        rdy <= '0';
        next_state <= rew_finish;
        else
        LA_points4 <= LA_points4;
        rdy <= '0';
        timeoff<= timeoff;
        next_state <= rew_finish;
        end if;

        when rew_finish =>          --in this state we assert rdy to be high,
the timeoff value
                              --remains at the
                              --same value as it was in the previous state and
is
                              --constantly asserted.
                              --we stay in this state until the payroll updates
the weekly
                              --hours table,
                              --at which point, our timeoff value needs to be
calculated again.
        rdy <= '1';
        timeoff <= timeoff;
        if (SRAM_busy = '0') then
        next_state <= rew_finish;
        else
        next_state <= idle;
        end if;

          when others =>
          next_state <= idle;
   end case;
   end process;
```

```
process(clk, present_state, next_state, SRAM_busy)
   begin
    if rising_edge(clk) then
      if(SRAM_busy = '1') then
      present_state <= idle;        --SRAM_busy means that the payroll is
updating the staff
                              --weekly hours table,
                              --which automatically means that the current min
and max
                              --are no longer valid
                              --the fsm should return to idle and wait until the
payroll
                              --is done updating
      else
      present_state <= next_state;
      end if;
    end if;
 end process;


   instmaxmin : maxmin
   port map (     clk => clk,
            minmax_start =>minmax_start,
            SRAM_busy => SRAM_busy,
            SRAM_rdy => SRAM_rdy,
            min => min,
            max => max,
            staff_hours => staff_hours,
            minmax_done => minmax_done);

end architecture state_machine;




library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;

-- here is the declaration of entity

entity maxmin is
 port (clk, SRAM_busy, SRAM_rdy   : in std_logic;
      minmax_start, minmax_done : buffer std_logic;
      min : buffer std_logic_vector(2 downto 0);
      max : buffer std_logic_vector(2 downto 0);
      staff_hours : in std_logic_vector(2 downto 0));
end maxmin;

architecture state_machine of maxmin is

-- first declaration of all states
type StateType is (idle,read_TA1, read_TA12, read_TA2, read_TA22, read_TA3,
read_TA32, finish);
```

```
 signal staff_hrs_adr :  std_logic_vector(2 downto 0);
 signal present_state, next_state : StateType;

    begin
     process(present_state, clk, minmax_start, staff_hours, min, max,
staff_hours, SRAM_busy, SRAM_rdy)
      begin
      minmax_done <= '0';
       min <="000";
       max <= "111";
        case present_state is
          when idle =>
           minmax_done <= '0';
           min <= "000";
           max <= "111";

           if(minmax_start = '1') then
           staff_hrs_adr <= "000";
           minmax_done <= '0';
           min<="111";
           max<="000";
           next_state <= read_TA1;
           else
           minmax_done <= '0';
           next_state <= idle;
           end if;

           when read_TA1 =>
           minmax_done <= '0';
           min <= "111";
           max <= "000";
           staff_hrs_adr <= "000";
           if(SRAM_rdy = '1') then
           next_state <= read_TA12;
           minmax_done <= '0';
           else
           next_state <= read_TA1;
           minmax_done <= '0';
           min <= "111";
           max <= "000";
           end if;

           when read_TA12 =>
           minmax_done <= '0';
           min <= staff_hours;
           max <= staff_hours;
           next_state <= read_TA2;

           when read_TA2 =>
           minmax_done <= '0';
           min <= min;
           max <= max;
           staff_hrs_adr <= "001";
           if(SRAM_rdy = '1') then
           next_state <= read_TA22;
           minmax_done <= '0';
           else
```

```
next_state <= read_TA2;
minmax_done <= '0';
min <= min;
max <= max;
end if;

when read_TA22 =>
minmax_done <= '0';
if(staff_hours < min) then
min <= staff_hours;
max <= max;
minmax_done <= '0';
next_state <= read_TA3;
elsif(staff_hours > max) then
max <= staff_hours;
min <= min;
minmax_done <= '0';
next_state <= read_TA3;
else
max <= max;
min <= min;
minmax_done <= '0';
next_state <= read_TA3;
end if;


when read_TA3 =>
minmax_done <= '0';
min <= min;
max <= max;
staff_hrs_adr <= "010";
if(SRAM_rdy = '1') then
next_state <= read_TA32;
minmax_done <= '0';
else
next_state <= read_TA3;
minmax_done <= '0';
min <= min;
max <= max;
end if;

when read_TA32 =>
minmax_done <= '0';
if(staff_hours < min) then
min <= staff_hours;
max <= max;
minmax_done <= '1';
next_state <= finish;
elsif(staff_hours > max) then
max <= staff_hours;
min <= min;
minmax_done <= '1';
next_state <= finish;
else
max <= max;
min <= min;
minmax_done <= '1';
```

```
           next_state <= finish;
           end if;

           when finish =>
           min <= min;
           max <= max;
           minmax_done <= '1';
           if(SRAM_busy = '0') then
           next_state <= finish;
           minmax_done <= '1';
           else
           next_state <= idle;
           minmax_done <= '0';
           end if;


          when others =>
           next_state <= idle;
   end case;
  end process;

process(clk, present_state, next_state, SRAM_busy)
   begin
    if rising_edge(clk) then
      if (SRAM_busy = '0') then
      present_state <= next_state;
      else
      present_state <= idle;
      end if;
   end if;
 end process;

end architecture state_machine;
```