***Si progetti**, utilizzando il linguaggio VHDL, una rete logica in grado di realizzare il controllore di un distributore automatico di bevande. L'interfaccia di I/O della rete è la seguente:*

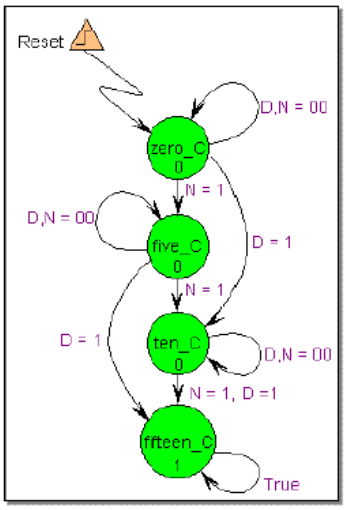
ESERCIZIO

```
entity vending15 is
port( reset  : in bit;  -- al posto del tipo bit si
      clock  : in bit;  -- può mettere std_logic
      dime   : in bit;
      nichel : in bit;
      output : out bit);
end vending15;
```

- *non da' resto e non è specificato cosa fa una volta raggiunti i 15 centesimi*
- *non è possibile inserire due monete contemporaneamente*
- *non è specificato quale evento permette di usicire dallo stato che rivela l 15c a parte il reset*

Il distributore accetta solo ed esclusivamente monete da 5 (nichel) e 10 (dime) centesimi. Al raggiungimento dell'importo di 15 centesimi deve attivare un segnale di abilitazione (output) per l'erogazione della bibita.


```
--Vending machine 15 Cent

entity vending15 is
port(
        reset   : in    bit;  -- Active high reset.
        clock   : in    bit;  -- Positive edge clock.
        dime    : in    bit;  -- Dime coin input.
        nichel  : in    bit;  -- Nickel coin input.
        output  : out   bit   -- Release output.
  );
end vending15;
```



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

entity vending15 is
  port( reset  : in bit;
        clock  : in bit;
        dime   : in bit;
        nichel : in bit;
        output : out bit);
end vending15;

architecture algoritmica of vending15 is

--- parte dichiarativa dell'architettura: è dichiarato un nuovo tipo
--- denominato "state" e definito attraverso l'elenco dei valori che può assumere

type state is (zero_c, five_c, ten_c, fifteen_c);

--- segnale interni che rappresentano rispettivamente lo stato presente
--- (uscita dei registri di stato) e quello futuro (ingresso dei registri di stato)

  signal present_state, next_state: state;
```



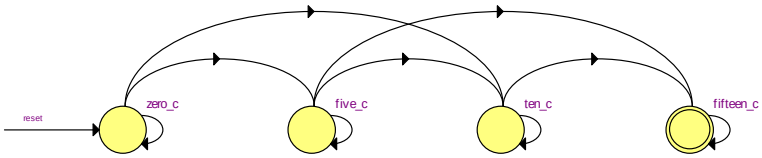
```
begin
---- parte combinatoria della FSM: rete combinatoria che definisce
---- il valore dello stato futuro e dell'uscita per ogni configurazione
---- di stato presente e per ogni valore degli ingressi

  process(present_state, nichel, dime) -- nella sensitivity list
                                       -- il segnale che rappresenta lo stato corrente
                                       -- (present_state) e i segnali di ingresso
  begin

    case present_state is

      when zero_c =>
            output <= '0';  --- macchina di Moore l'uscita dipende
                            --- solo dallo stato corrente
        if(nichel = '1')
          then next_state <= five_c;
        elsif(dime = '1')
          then next_state <= ten_c;
        else
          next_state <= present_state;
        end if;
```

```
when five_c =>
        output <= '0';
    if(nichel = '1')
       then next_state <= ten_c;
    elsif(dime = '1')
       then next_state <= fifteen_c;
    else
      next_state <= present_state;
    end if;

when ten_c =>
        output <= '0';
    if(nichel = '1')
      then next_state <= fifteen_c;
    elsif(dime = '1')
      then next_state <= fifteen_c;
    else
      next_state <= present_state;
    end if;

when fifteen_c =>
    output <= '1';
    next_state <= present_state;

 when others =>   ---- permette di definire lo stato futuro  se inderteminato
     output <= '0';
    next_state <= zero_c;
 end case;
 end process;
```

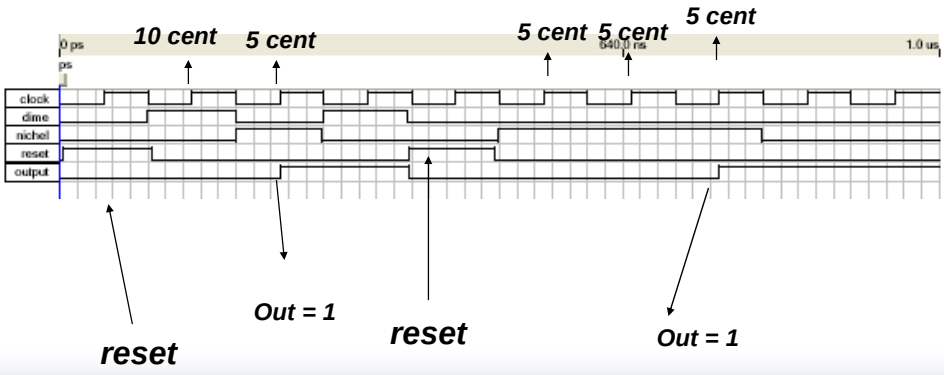


```
--- parte sequenziale della FSM: registri di stato
 process(reset,clock)    --- reset asincrono, compare nella sensitivity list
 begin
   if reset = '1'
     then present_state <= zero_c;
   elsif(clock'event and clock = '1')
     then present_state <= next_state;
   end if;
 end process;

end algoritmica;
```

RTL Viewer



Simulazione funzionale



ESERCIZIO 1 Elettronica dei Sistemi Digitali
Linguaggi di Descrizione Hardware

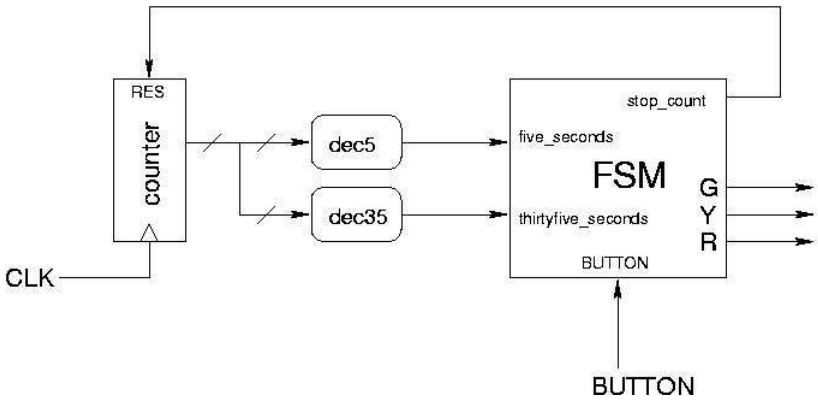
Si progetti, utilizzando FPGA Altera e linguaggio VHDL, una rete di controllo per un semaforo pedonale. Il semaforo è normalmente verde e, in seguito alla pressione di un pulsante diventa giallo per 5 secondi e successivamente rosso per 30 secondi. Si ipotizzi di avere a disposizione, per generare il clock, un oscillatore a 1 kHZ. Sia lecito supporre, per semplicità, che il pulsante in seguito alla pressione attivi il segnale corrispondente per UN SOLO ciclo di clock.

Per rendere piu agevoli le simulazioni, le durate dei segnali giallo e rosso possono essere assunte pari a 5 e 30 cicli rispettivamente, INDICANDO COME COMMENTO come cambia il codice VHDL corrispondente.

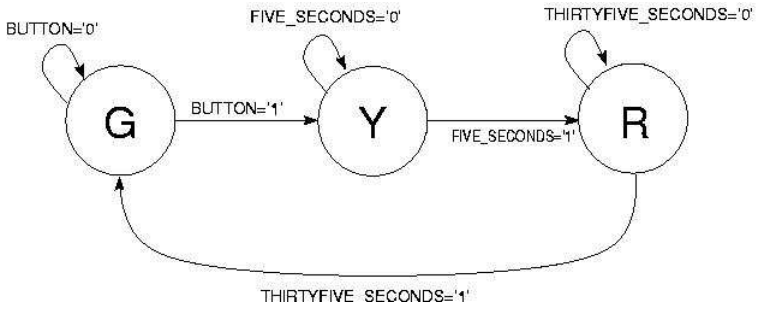
Si utilizzi, per motivi di uniformità, la seguente interfaccia di I/O.

```
entity TRAFFICLIGHT is
   port (
      BUTTON    : in  std_logic;
      CLOCK     : in  std_logic;
      RESET     : in  std_logic;
      RED       : out std_logic;
      YELLOW    : out std_logic;
      GREEN     : out std_logic
   );
end TRAFFICLIGHT;
```


Schema a blocchi semplicato



Digramma degli stati




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


entity COUNTER is
  port (
    CLK   : in  std_logic;
    RESET : in  std_logic;
    VALUE : out unsigned(15 downto 0)     -- per contare 35000 cicli, corrispondenti a  35 secondi a 1 kHZ di
                                                    --- freq. servono 16 bit 2^16 - 1 = 65535
    );
end COUNTER;

architecture A of COUNTER is
signal next_value, value_temp : unsigned(15 downto 0);
begin  -- A

  process(CLK)
    begin
      if CLK'event and CLK='1' then
        if RESET='1' then
          value_temp <= (others => '0');
        else
          value_temp <= next_value;
        end if;
      end if;
   end process;

  VALUE <= value_temp;
  next_value <= value_temp + conv_unsigned(1,16);

end A;
```

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

entity TRAFFICLIGHT is
  port (
    BUTTON : in  std_logic;
    CLOCK  : in  std_logic;
    RESET  : in  std_logic;
    RED    : out std_logic;
    YELLOW : out std_logic;
    GREEN  : out std_logic);
end TRAFFICLIGHT;

architecture ESERCIZIO of TRAFFICLIGHT is

  -- definisco il tipo stato e i segnali che mi serviranno per la FSM
  type stato is (G,Y,R);
  signal current_state, next_state : stato;

  -- segnali necessari a memorizzare i dati generati internamente  all'architecture
  signal current_count      : unsigned(15 downto 0);
  signal five_seconds       : std_logic;
  signal thirtyfive_seconds : std_logic;
  signal stop_counter       : std_logic;

  -- dichiaro l'intenzione di usare un componente esterno  e ne specifico l'interfaccia di IO
  component COUNTER
    port (
      CLK   : in  std_logic;
      RESET : in  std_logic;
      VALUE : out unsigned(15 downto 0)
    );
  end component;
```

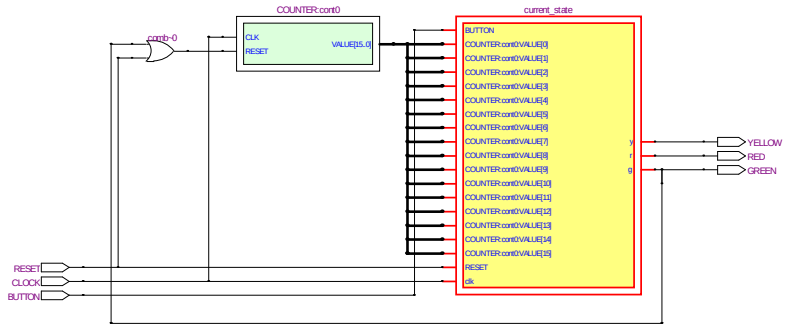


```
  begin  -- ESERCIZIO

    cont0 : COUNTER port map (                  -- istanziazione de componente contatore

     CLK   => CLOCK,
     RESET => stop_counter or RESET,
     VALUE => current_count);

   --  statement che rappresenta il blocco combinatorio DEC5  mostrato sullo schematico
         five_seconds <= '1' when current_count = conv_unsigned(2,16) else '0';

   --CASO REALE: five_seconds <= '1' when current_count = conv_unsigned(5000,16) else '0';

     -- questo processo realizza in maniera equivalente alla precedente
     -- la rete di decodifica combinatoria corrispondente ai 35 secondi
     -- Tuttavia viene utilizzato un PROCESS;

     process(current_count)
       begin
                -- CASO REALE: if current_count = conv_unsigned(35000,16) then
       if current_count = conv_unsigned(5,16) then
         thirtyfive_seconds <= '1';
       else
         thirtyfive_seconds <= '0';
         -- e' un processo combinatorio, l'uscita  va calcolata in ogni ramo di esecuzione del processo!!!!
       end if;

     end process;

   ---- il processo appena descritto è equivalente allo statement
   ---  thirtyfive_seconds <= '1' when current_count = conv_unsigned(35000,16) else '0';
```

```
-- parte sequenziale della macchina a stati

process(CLOCK)
    begin
      if CLOCK'event and CLOCK='1' then
        if RESET='1' then
          current_state <= G;
        else
          current_state <= next_state;

        end if;
      end if;
    end process;
```



```
-- parte combinatoria della macchina a stati.

process(current_state, BUTTON, five_seconds, thirtyfive_seconds)

    begin

     case current_state is

      when G => RED <= '0';          -- macchina di Moore
            YELLOW <= '0';
            GREEN <= '1';
            stop_counter <= '1';

             if BUTTON = '1' then
              next_state <= Y;
            else
              next_state <= G;
            end if;

      when Y => RED <= '0';
            YELLOW <= '1';
            GREEN <= '0';
            stop_counter <= '0';

            if five_seconds = '1' then
              next_state <= R;
            else
              next_state <= Y;
            end if;

      when R => RED <= '1';
                  YELLOW <= '0';
                  GREEN <= '0';
                  stop_counter <= '0';

                  if thirtyfive_seconds = '1' then
                    next_state <= G;
                  else
                    next_state <= R;
                  end if;

           when others => RED <= '0';
                        YELLOW <= '0';
                        GREEN <= '0';
                        stop_counter <= '1';
                        next_state <= G;
          end case;

     end process;

    end ESERCIZIO;
```



Contatore

Reset sincrono

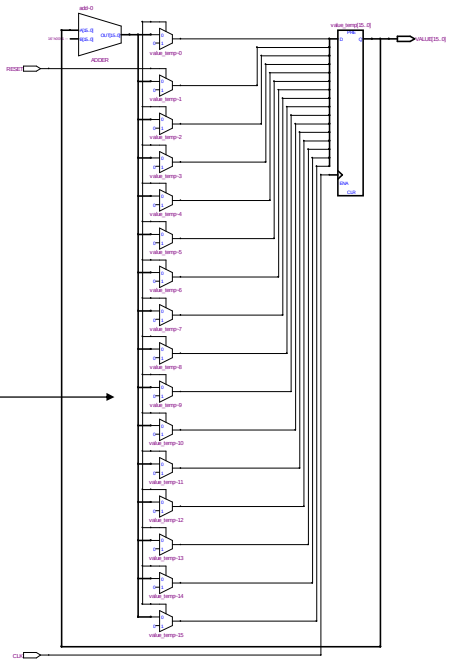
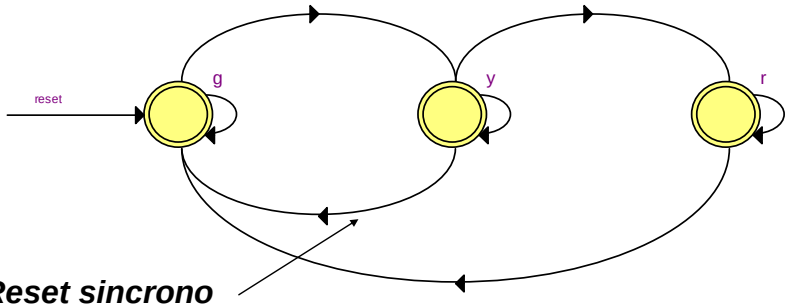
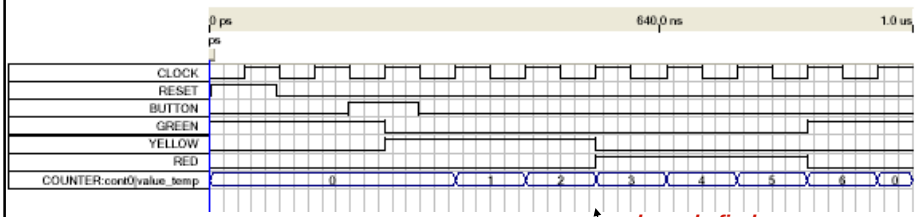


Diagramma degli stati generato da RTL viewer



Reset sincrono

Simulazione funzionale



da node finder:
Insert register pre-synthesis

L'uscita del contatore varia da 0 a 2^n-1
Al primo ciclo l'uscita vale 0

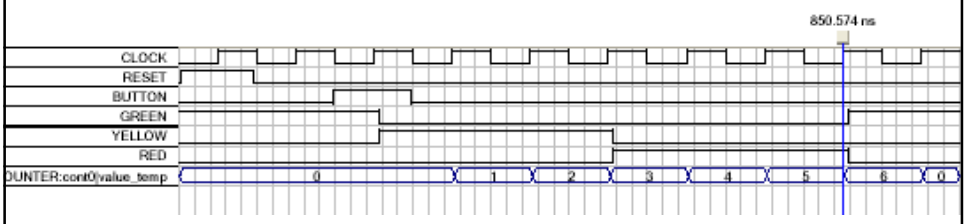
La condizione per il cambio di stato era stata posta al valore 2, dopo quindi 3 cicli
In generale se voglio che il contatore riparta da 0 dopo N cicli, la condizione deve essere posta a N-1

Risultato compilazione

- Start compilation

Flow Status	Successful - Wed Nov 07 11:06:28 2007
Quartus II Version	5.0 Build 148 04/26/2005 SJ Full Version
Revision Name	trafficlight
Top-level Entity Name	TRAFFICLIGHT
Family	Stratix
Met timing requirements	Yes
Total logic elements	29 / 10,570 (< 1 %)
Total pins	6 / 336 (1 %)
Total virtual pins	0
Total memory bits	0 / 920,448 (0 %)
DSP block 9-bit elements	0 / 48 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP1S10F484C5
Timing Models	Final

Simulazione timing



Timing analyzer

- Start timing analyzer
  - fmax = 239.98 MHz
  - Tck = 4.167 ns
  - from register Counter:cont0|value_temp11 to register current_state.r

Esercizio 2:

- Esame del 5 luglio 2004
- In rete è disponibile una soluzione dettagliata

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


entity CICLOCOMPUTER is
   port (
      GIROCOMPIUTO : in  std_logic;
      CLOCK        : in  std_logic;
      RESET        : in  std_logic;
      MODE         : in  std_logic;
      VALORE       : out unsigned(15 downto 0)
   );
end CICLOCOMPUTER;

architecture A of CICLOCOMPUTER is
signal cont_distanza, next_distanza, giri_sec, next_giri_sec, giri_sec_frozen : unsigned(14 downto 0);
signal cicli, next_cicli : unsigned(14 downto 0);
signal wave1Hz : std_logic;
signal select_output : std_logic;

type modo_funzionamento is (velocita, distanza);
signal current_state, next_state : modo_funzionamento;
```



```
begin
process(clock)   -- contatore della distanza
begin
     if clock'event and clock = '1' then
          if reset = '1' then
          cont_distanza <= conv_unsigned(0,15);

          elsif girocompiuto = '1' then
         cont_distanza <= next_distanza;

          end if;
    end if;
end process;
next_distanza <= cont_distanza + conv_unsigned(1,15);
-- contatore della velocita, ogni secondo azzero il conteggio
process(clock)   -- contatore della velocita, ogni secondo azzero il conteggio
begin
     if clock'event and clock = '1' then
          if reset = '1' or wave1Hz = '1' then
          giri_sec <= conv_unsigned(0,15);
          elsif girocompiuto = '1' then
         giri_sec <= next_giri_sec;
          end if;
    end if;
end process;
next_giri_sec <= giri_sec + conv_unsigned(1,15);
```

Contatori con ingresso di abilitazione

i contatori sono descritti e non instaziati

```
-- ogni secondo vado a campionare i giri calcolati
--  registro
process(clock)
begin
        if clock'event and clock = '1' then
             if reset = '1' then
             giri_sec_frozen <= conv_unsigned(0,15);
            elsif wave1Hz = '1' then
            giri_sec_frozen <= giri_sec;
           end if;
        end if;
end process;
```



```
-- macchina a stati per commutare tra distanza e velocita
process(clock)
begin
  if clock'event and clock = '1' then
    if reset = '1' then
      current_state <= distanza;
    else
      current_state <= next_state;
    end if;
  end if;
end process;

process(current_state, mode)
begin
  case current_state is
    when velocita =>
      if mode = '1' then
        next_state <= distanza;
      else
        next_state <= velocita;
      end if;
      select_output <= '0';

    when distanza =>
      if mode = '1' then
        next_state <= velocita;
      else
        next_state <= distanza;
      end if;
      select_output <= '1';

    when others =>
      next_state <= distanza;
      select_output <= '0';
  end case;
end process;
```

```
process(clock)
begin
        if clock'event and clock = '1' then
                if reset = '1' then
                cicli <= conv_unsigned(0,15);
                else
                cicli <= next_cicli;
                end if;
        end if;
end process;

next_cicli <= conv_unsigned(0,15) when cicli = conv_unsigned(9,15) else cicli + conv_unsigned(1,15);
--- nel caso reale avrei dovuto sostiture 9999 a 9

wave1Hz <= '1' when cicli = conv_unsigned(0,15) else '0';


-- seleziono il valore da portare in uscita. Devo convertire da giri a metri,
-- ma essendo 1 giro = 2 metri, basta shiftare a sinistra di 1;
valore(0) <= '0';
valore(15 downto 1) <= cont_distanza when select_output = '0' else giri_sec_frozen;


end A;
```


Es: 3
Moltiplicatore/Accululatore (MAC)

- Si realizzi un moltiplicatore-accumulatore (MAC) a pipeline in cui:
- il primo stadio campiona i dati in ingresso
- il secondo stadio campiona il prodotto
- il terzo stadio campiona e porta sull'uscita la somma di accumulo
- Per gli ingressi si adotti una rappresentazione a 4 bit

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

-- Nota: il segnale di reset e' INDISPENSABILE per resettare ad un valore noto
-- i FF del registro.
entity mac is
port (clk,reset  : in std_logic;
      in_chan1,in_chan2 : in unsigned(3 downto 0);
      out_chan          : out unsigned(7 downto 0) );
end mac;

architecture s of mac is

component reg
port ( clk,reset: in std_logic;
       reg_in  : in  unsigned(7 downto 0);
       reg_out : out unsigned(7 downto 0) );
end component;

signal ck_in1,ck_in2 :                  unsigned(3 downto 0);
signal prod,ck_prod,sum,newsum : unsigned(7 downto 0);
```



```
begin
  -- Sincronizzazione degli ingressi, potrebbe essere descritto
  -- attraverso una subentity gerarchica (non fa' differenza)

 process(clk,reset)
  begin
    if reset='1' then
       ck_in1<="0000";
       ck_in2<="0000";
    elsif clk'event and clk='1' then
          ck_in1<=in_chan1;
          ck_in2<=in_chan2;
       end if;
    end if;
  end process;

  -- Blocco di moltiplicazione, potrebbe essere ugualmente descritto
  -- attraverso una subentity gerarchica

    prod <= ck_in1 * ck_in2;
```

```
 -- Registro di pipeline, potrebbe essere descritto ugualmente attraverso
   -- un processo (non fa' differenza)

          Pipe_reg  : reg port map (clk,reset,prod,ck_prod);

   -- Result Accumulator
          newsum <= sum+ck_prod;

   -- Registro accumulatore, potrbbe essere descritto ugualmente attraverso
   -- un processo (non fa' differenza)

  acc_reg  : reg port map (clk,reset,newsum,sum);

   -- Il risultato viene portato sui pins di uscita
   out_chan <= sum;
end s;
```



```
--------------------------------------------------------------------------
-- DATA_REG : General purpose Register entity
--------------------------------------------------------------------------
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;
   use IEEE.std_logic_unsigned.all;

entity reg is
port ( clk,reset : in std_logic;
       reg_in  :     in  unsigned(7 downto 0);
       reg_out : out unsigned(7 downto 0) );
end reg;

architecture b of reg is
begin
  process(clk,reset)
  begin
    if reset='1' then
       reg_out <= ( others => '0');
    elsif clk'event and clk='1' then
            reg_out <= reg_in;
          end if;
    end if;
  end process;
end b;
```