Esempi di reti combinatorie e sequenziali in VHDL

Esercitazione 04 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2016-17

Indice

argomenti dell'esercitazione

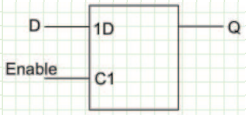
in questa esercitazione si trattano descrizioni in VHDL di componenti hardware di frequente impiego:

- componenti sequenziali:
  - latch, flip-flop, registri
  - contatori, registri a ingresso seriale
- componenti combinatori:
  - funzioni di ALU
  - decodificatori, multiplatori
  - display a 7 segmenti

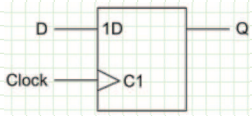
inoltre si propone un'esperienza di laboratorio in cui confluiscono diversi aspetti di VHDL illustrati dagli esempi qui presentati e alcuni componenti possono essere riusati per la realizzazione dell'esperimento proposto

latch, flip-flop, registri

- *latch*: memoria da un bit, sensibile al livello
- *flip-flop*: memoria da un bit, sensibile al fronte (*edge-triggered*)
- registro (parallelo): schiera di flip-flop



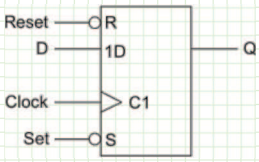
latch di tipo D



flip-flop di tipo D

```
library ieee;
use ieee.std_logic_1164.all;
entity latch is
 port (
  d : in std_logic;
  en : in std_logic;
  q : out std_logic
 );
end entity latch;

architecture beh of latch is
begin
process (d, en) is
begin
 if (en = '1') then
  q <= d;
 end if;
end process;
end architecture beh;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
 port (
  d : in std_logic;
  clk : in std_logic;
  q : out std_logic
 );
end entity dff;

architecture simple of dff is
begin
process (clk) is
begin
 if rising_edge(clk) then
  q <= d;
 end if;
end process;
end architecture simple;
```



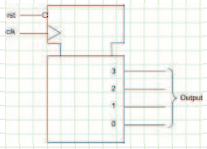
flip-flop di tipo D con set e reset asincroni

esercizio: correggere gli errori nel codice VHDL presentato nel testo di riferimento, p. 290

```
library ieee;
use ieee.std_logic_1164.all;
entity register is
  generic ( n : natural := 8 );
  port (
    d : in std_logic_vector(n-1 downto 1);
    clk : in std_logic;
    nrst : in std_logic;
    load : in std_logic;
    q : out std_logic_vector(n-1 downto 1)
  );
end entity register;

architecture beh of register is
begin
process (clk, nrst) is
begin
  if (nrst = '0') then
    q <= (others => '0');
  elsif (rising_edge(clk) and (load = 1)) then
    q <= d;
  end if;
end process;
end architecture beh;
```

contatori, registri a ingresso seriale

- contatori: registri di conteggio di specificati fronti di clock
  usati anche per realizzare temporizzatori
- registri a ingresso seriale: in parte simili ai contatori
  usati per l'input da linee dati seriali, l'output è parallelo



contatore binario

l'uso di *variabili* facilita la descrizione VHDL in stile comportamentale:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity counter is
  generic ( n : integer := 4 );
  port (
    clk : in std_logic;
    rst : in std_logic;
    output : out std_logic_vector((n−1) downto 0)
  );
end;
architecture simple of counter is
begin
  process(clk, rst)
    variable count : unsigned((n−1) downto 0);
  begin
    if rst = '0' then
      count := (others => '0');
    elsif rising_edge(clk) then
      count := count + 1;
    end if;
    output <= std_logic_vector(count);
  end process;
end;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity shift_register is
  generic ( n : integer := 4 );
  port (
    clk : in std_logic;
    rst : in std_logic;
    din : in std_logic;
    q : out std_logic_vector((n−1) downto 0)
  );
end entity;
architecture simple of shift_register is
begin
  process(clk, rst)
    variable shift_reg : std_logic_vector((n−1) downto 0);
  begin
    if rst = '0' then
      shift_reg := (others => '0');
    elsif rising_edge(clk) then
      shift_reg := shift_reg(n−2 downto 0) & din;
    end if;
    q <= shift_reg;
  end process;
end architecture simple;
```


funzioni di ALU

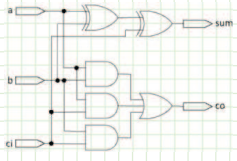
una ALU è una unità multifunzione: un input di controllo seleziona l'operazione da eseguire
un esempio di unità logica multifunzione ha la selezione specificata in tabella e può descriversi in VHDL come segue:

S	funzione
00	Q <= not A
01	Q <= A and B
10	Q <= A or B
11	Q <= A xor B

```
library ieee;
use ieee.std_logic_1164.all;
entity alu_logic is
  generic ( n : natural := 16 );
  port (
    a : in std_logic_vector((n−1) downto 0);
    b : in std_logic_vector((n−1) downto 0);
    s : in std_logic_vector(1 downto 0);
    q : out std_logic_vector((n−1) downto 0)
  );
end entity alu_logic;
```

```
architecture basic of alu_logic is
begin
  case s is
    when “00” => q <= not a;
    when “01” => q <= a and b;
    when “10” => q <= a or b;
    when “11” => q <= a xor b;
  end case;
end architecture basic;
```

le funzioni aritmetiche dell'ALU possono specificarsi in stile dataflow o, per operandi di più bit, in stile strutturale o comportamentale; quest'ultimo come segue



full-adder da 1 bit

```
library ieee;
use IEEE.std_logic_1164.all;
entity add_beh is
  generic(top : natural := 15);
  port (
    a : in std_logic_vector (top downto 0);
    b : in std_logic_vector (top downto 0);
    cin : in std_logic;
    sum : out std_logic_vector (top downto 0);
    cout : out std_logic
  );
end entity add_beh;
```

```
architecture behavior of add_beh is
begin
  adder:
  process(a,b,cin)
    variable carry : std_logic;
    variable tempsum : std_logic_vector(top downto 0);
  begin
    carry := cin;
    for i in 0 to top loop
      tempsum(i) := a(i) xor b(i) xor carry;
      carry := (a(i) and b(i)) or (a(i) and carry) or (b(i) and carry);
    end loop;
    sum <= tempsum;
    cout <= carry;
  end process adder;
end architecture behavior;
```

decodificatori, multiplatori

probabilmente le unità funzionali combinatorie più diffuse nell'hardware digitale:

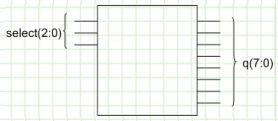
- decodificatore $n \rightarrow 2^n$
- multiplatore $n \times 2^n \rightarrow 1$

specifica VHDL generica del decodificatore (Zwolinski, sez. 4.2.3):

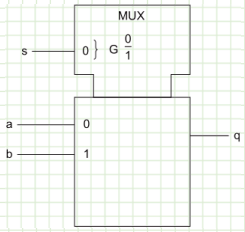
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity decoder is
 generic (n : positive);
 port (
   a : in std_logic_vector(n-1 downto 0);
   z : out std_logic_vector(2**n-1 downto 0)
 );
end entity decoder;

architecture rotate of decoder is
 constant z_out : bit_vector(2**n-1 downto 0) :=
   (0 => '1', others => '0');
begin
 z <= to_StdLogicVector (z_out sll
                  to_integer(unsigned(a)));
end architecture rotate;
```

specifica VHDL di un multiplatore a singolo input di selezione:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux21 is
 port (
   s : in std_logic;
   a : in std_logic;
   b : in std_logic;
   q : out std_logic
 );
end;

architecture simple of mux21 is
begin
 q <= a when s = '0' else
      b when s = '1' else
      'X';
end;
```



decodificatore 3-8



multiplatore a singolo input di selezione

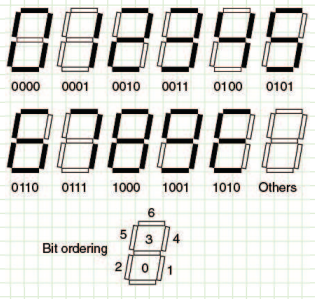
display a 7 segmenti

dispositivo familiare di uso quotidiano ...

un decodificatore per display a 7 segmenti converte un input a 4 bit nella configurazione binaria dei 7 bit, associati ai segmenti del display, che visualizza il carattere corrispondente alla cifra (esadecimale o decimale) che ha il valore codificato dall'input

l'esempio che segue (tratto da Zwolinski, sez. 4.2.3)

- visualizza cifre decimali
- visualizza E se il valore dell'input è ≥ 10
- in tutti gli altri casi il display è oscurato



Zwolinski, Figure 4.3 - Seven-segment display

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

entity seven_seg is
 port (a : in std_logic_vector(3 downto 0);
       z : out std_logic_vector(6 downto 0));
end entity seven_seg;

architecture with_select of seven_seg is
begin
 with a select
   z <= "1110111" when "0000",
        "0010010" when "0001",
        "1011101" when "0010",
        "1011011" when "0011",
        "0111010" when "0100",
        "1101011" when "0101",
        "1101111" when "0110",
        "1010010" when "0111",
        "1111111" when "1000",
        "1111011" when "1001",
        "1101101" when "1010"|"1011"|"1100"|"1101"|"1110"|"1111",
        "0000000" when others;
end architecture with_select;
```

esperienza di laboratorio

il simulatore di ALU a 8 bit realizzato da S. Lentini e G. Nicotra illustra come si possa costruire iterativamente l'ALU a 8 bit componendo opportunamente stadi di ALU a 1 bit

> lo schematico e un'illustrazione animata delle funzioni dell'ALU sono presentate nella parte "Unità Aritmetica Logica" dell'animazione Flash del progetto

usando i costrutti VHDL visti negli esempi di questa esercitazione:

1. costruire un modello VHDL dell'ALU a 1 bit
2. sulla base di tale risultato, costruire un modello generico dell'ALU a *n* bit
3. compilare e simulare istanze diverse del modello, ovvero per differenti valori di *n*
4. costruire un modello VHDL di decodificatore di cifre esadecimali per la visualizzazione su display a 7 segmenti
5. usando l'istanza a 3 bit del modello di ALU e il decodificatore prodotto al passo 4, programmare la FPGA usando gli switch per l'input alla ALU (6 switch per i due operandi e gli altri 6 per gli input di controllo: F0, F1, ENA, ENB, INC, INVA) e un display a 7 segmenti per la visualizzazione del risultato (carry incluso per le operazioni aritmetiche)

riferimenti

letture raccomandate:

> Wilson (2015) Capp. 20, 24, 21, 25, 27

letture per ulteriori approfondimenti:

> Zwolinski (2004) Sez. 4.1-5, 6.1-5

materiali utili per l'esperienza di laboratorio proposta
(fonti: Biblioteca DMI, Altera University Program, 2014)

> Tanenbaum: *Architettura dei calcolatori, Quinta Edizione* (Pearson, 2006) Sez. 3.2.3
>
> Making Qsys Components - For Quartus II 13.1 (solo Appendice A)