Esempi di reti sequenziali in VHDL, realizzazioni hardware di modelli dataflow

Esercitazione 05 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2018-19

Indice

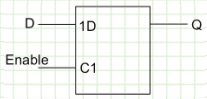
argomenti dell'esercitazione

in questa esercitazione si trattano:

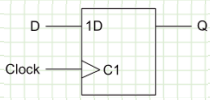
- componenti sequenziali:
  - latch, flip-flop, registri
  - contatori, registri a ingresso seriale
- realizzazione hardware di modelli SDF single-rate
  esempio, algoritmo GCD di Euclide:
  - analisi del grafo SDF
  - realizzazione hardware
- hardware pipelining:
  - miglioramento del throughput
  - cautela con il pipelining di grafi SDF con cicli
- esperienza di laboratorio
  - realizzazione hardware di un modello dataflow

latch, flip-flop, registri

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



latch di tipo D

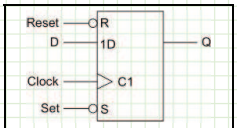


flip-flop di tipo D

```
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;
```

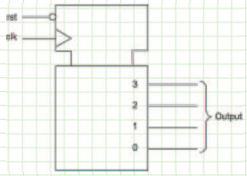
```
library ieee;
use ieee.std_logic_1164.all;
entity register is
 generic ( n : natural := 8 );
 port (
   d : in std_logic_vector(n-1 downto 0);
   clk : in std_logic;
   nrst : in std_logic;
   load : in std_logic;
   q : out std_logic_vector(n-1 downto 0)
 );
end entity register;

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;
```



flip-flop di tipo D con set e reset asincroni

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;
```


grafi SDF single-rate in hardware

ipotesi per la realizzazione hardware:

grafi SDF single-rate, tutti gli attori operano alla stessa frequenza di clock

tre regole per la realizzazione:

1. gli attori sono realizzati da circuiti combinatori
2. le code di comunicazione sono realizzate da connessioni (prive di memoria)
3. ogni token iniziale sulle code di comunicazione è rimpiazzato da un registro

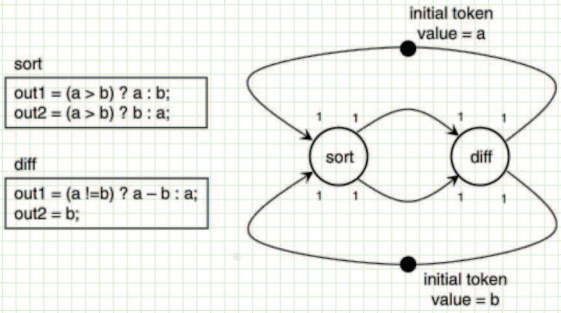
due definizioni:

- *cammino combinatorio* nel grafo SDF: un cammino aciclico privo di token iniziali
- *cammino critico* nel grafo SDF: cammino combinatorio di massima latenza

frequenza di clock massima per il circuito: reciproco della latenza del cammino critico

algoritmo: a ogni passo rimpiazza (a, b) con (|a-b|, min(a,b))

la coppia converge a (GCD(a,b), GCD(a,b))



Schaumont, Figure 3.10 - Euclid's greatest common divisor as an SDF graph

analisi PASS:

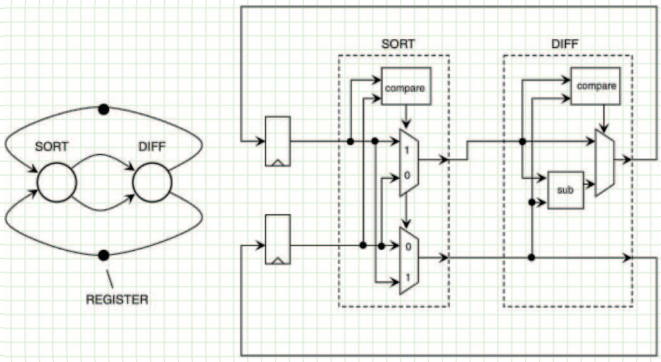
$$G = \begin{bmatrix} +1 & -1 \\ +1 & -1 \\ -1 & +1 \\ -1 & +1 \end{bmatrix} \begin{matrix} \leftarrow edge(sort,diff) \\ \leftarrow edge(sort,diff) \\ \leftarrow edge(diff,sort) \\ \leftarrow edge(diff,sort) \end{matrix} \qquad \text{rango}(G) = 1 \qquad q_{PASS} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

applichiamo le tre regole indicate per la realizzazione hardware del modello SDF:

- due circuiti combinatori realizzano gli attori
- un registro è posto su ciascuna delle due connessioni da diff a sort

la realizzazione degli attori è semplice, con pochi moduli di uso comune (multiplatori, comparatori e un sottrattore)

N.B. lo schema HW richiede un po' di immaginazione e una correzione

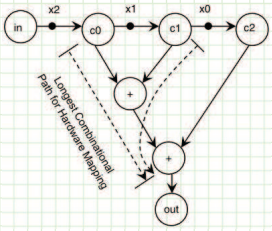


Schaumont, Figure 3.11 - Hardware implementation of Euclid's algorithm

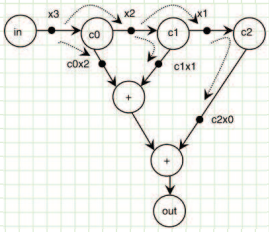
hardware pipelining

esempio di miglioramento del throughput mediante pipelining:

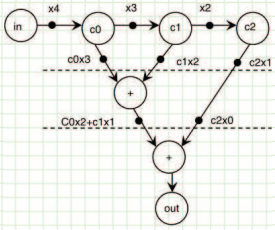
filtro digitale di somma pesata: x0·c2+x1·c1+x2·c0



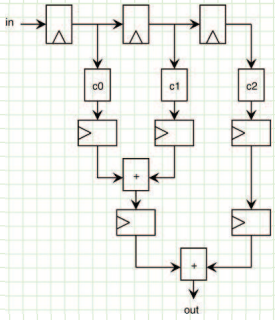
Schaumont, Figure 3.12
SDF graph of a simple moving-average application



Schaumont, Figure 3.13
Pipelining the moving-average filter by inserting additional tokens (1)



Schaumont, Figure 3.14
Pipelining the moving-average filter by inserting additional tokens (2)



Schaumont, Figure 3.15
Hardware implementation of the moving-average filter

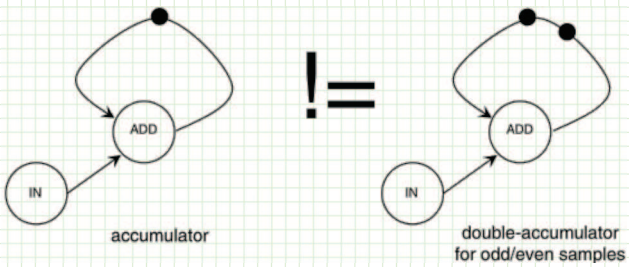
da notare:

- i token iniziali qui hanno il ruolo dei *delay* buffer nella definizione della trasformazione di pipelining
- il grafo SDF è aciclico... (v. appresso)

pipelining in grafi SDF con cicli

il pipelining, con l'aggiunta di token, può alterare il comportamento di un grafo SDF

in particolare, ciò può accadere se si aggiungono token all'interno di un ciclo, come dimostra questo esempio:



Schaumont, Figure 3.16 - Loops in SDF graphs cannot be pipelined

per applicare il pipelining senza alterare il comportamento funzionale del grafo, quando questo contiene cicli, i token aggiuntivi vanno posti al di fuori di qualsiasi ciclo nel grafo

per esempio, sulle linee di ingresso o uscita

esperienza di laboratorio

il circuito in figura 3.11 realizza il nucleo computazionale dell'algoritmo di Euclide per il calcolo del GCD, ma non presenta elementi di controllo atti a segnalare l'inizio e la fine del calcolo né a distinguere ingressi e uscita; gli obiettivi di questa esperienza sono: estendere il circuito a tal fine, produrne una descrizione in VHDL, simularne il comportamento, e realizzarlo sulla FPGA DE1-SoC

1. estendere lo schematico del circuito di figura 3.11 con tre segnali di input e due di output:
   - `a`, `b`: i dati in ingresso, da 5 bit ciascuno
   - `start`: input da 1 bit, per segnalare la presenza dei dati in ingresso
   - `gcd`: il risultato in uscita, da 5 bit
   - `done`: output da 1 bit, per segnalare la fine del calcolo e la presenza del risultato in uscita

   e con elementi aggiuntivi (flip-flop, multiplatori, eventuale comparatore) utili a realizzare l'obiettivo proposto
2. produrre una descrizione in VHDL del circuito progettato, direttamente oppure mediante una descrizione in Gezel tradotta in VHDL mediante il programma `fdlvhd`
3. creare un progetto Quartus `Euclid`, assegnarvi i file .vhd prodotti, compilare e simulare il comportamento del circuito con alcune coppie di dati in ingresso
4. creare un nuovo progetto Quartus `Euclid_on_DE1SoC`, assegnandovi i file .vhd precedenti e inoltre il decodificatore per display a 7 segmenti impiegato nell'esercitazione 4 e una nuova entità VHDL al livello radice che componga quella precedente con un'istanza del sudddetto decodificatore e mappi i segnali di I/O sui pin della FPGA come segue:
   - `a`, `b`: SW9-5, SW4-0
   - `start`: not KEY1
   - `RST`: not KEY0
   - `CLK`: CLOCK_50 (clock di sistema a 50 MHz)
   - `done`: LEDR0
   - `gcd`: LEDR1, HEX0
5. importare gli assegnamenti di pin DE1-SoC, compilare, programmare la FPGA con il file .sof che ne risulta e collaudarne il funzionamento con alcune coppie di dati in ingresso

riferimenti

letture raccomandate:

- Zwoliński, Ch. 6, Sect. 6.1-6.5.1
- Schaumont, Ch. 3, Sect. 3.2

letture per ulteriori approfondimenti:

- Schaumont, Ch. 3, Sect. 3.3

materiali utili per l'esperienza di laboratorio proposta
(fonte: Intel Corp. - FPGA University Program, 2016)

- Debugging of VHDL Hardware Designs on Intel's DE-Series Boards - For Quartus Prime 16.1, Sect. 4.1, 6-8

sorgenti VHDL:

- esempi in questa presentazione
- esempi nel testo di Zwoliński