Esempi di FSMD in Gezel e in VHDL

Esercitazione 06 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:

- metodi di realizzazione hardware di modelli FSMD
- esempio di progetto e realizzazione hardware di FSMD
  - progetto di una datapath di calcolo di mediana
  - sequenzializzazione mediante una FSMD
- descrizione di FSM in VHDL
  - stili di descrizione
  - codifica degli stati
  - ottimizzazione del ritardo
- esperienza di laboratorio:
  - progetto e realizzazione su FPGA di un testbench per il Collatz delay

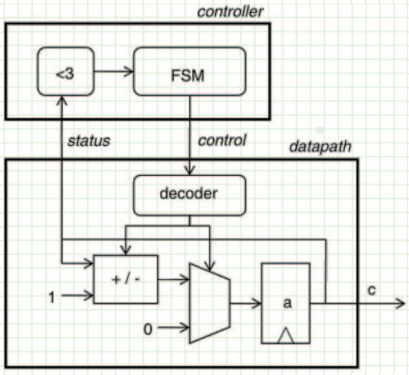
realizzazione hardware di modelli FSMD

realizzazione hardware di espressioni del datapath: stesse regole di base che in blocchi always, tuttavia:

la struttura hardware del datapath dipende anche dalla schedule di istruzioni della FSM...

perché?

vediamo l'esempio di contatore bidirezionale dalla lezione precedente:



Schaumont, Figure 5.8 - Implementation of the up-down counter FSMD

- il modello FSM implica che istruzioni diverse siano eseguite in cicli di clock diversi
- istruzioni mai concorrenti possono condividere hardware di operatori
  l'addizionatore/sottrattore in questo caso
- una decodifica locale converte la codifica delle istruzioni in segnali di controllo del datapath, per abilitare il percorso di esecuzione appropriato
- una decodifica locale nel controllore estrae informazione di stato del datapath per le condizioni nella FSM

esempio di progetto di FSMD: calcolo di mediana

specifica della funzione: calcolo della mediana in una lista di cinque numeri

problema: progettare un filtro che elabori una *stream* di dati, con un nuovo output per ogni nuovo input

filtri simili hanno applicazione nell'elaborazione delle immagini, per la riduzione del rumore

per un algoritmo rapido, senza ordinamento della lista:

in una lista di $2n+1$ elementi distinti, la mediana ha n elementi minori

```
dp median(in a1, a2, a3, a4, a5 : ns(32); out q1 : ns(32)) {
   sig z1, z2, z3, z4, z5, z6, z7, z8, z9, z10 : ns(3);
   sig s1, s2, s3, s4, s5 : ns(1);
   always {
      z1 = (a1 < a2);
      z2 = (a1 < a3);
      z3 = (a1 < a4);
      z4 = (a1 < a5);
      z5 = (a2 < a3);
      z6 = (a2 < a4);
      z7 = (a2 < a5);
      z8 = (a3 < a4);
      z9 = (a3 < a5);
      z10 = (a4 < a5);
      s1 = ((     z1 +      z2 +      z3 +      z4) == 2);
      s2 = (((1-z1) +      z5 +      z6 +      z7) == 2);
      s3 = (((1-z2) + (1-z5) +      z8 +      z9) == 2);
      s4 = (((1-z3) + (1-z6) + (1-z8) +    z10) == 2);
      q1 = s1 ? a1 : s2 ? a2 : s3 ? a3 : s4 ? a4 : a5;
   }
}
```

Schaumont, Listing 5.19 - GEZEL Datapath of a median calculation of five numbers

l'algoritmo esposto da questo datapath funziona anche quando alcuni elementi sono uguali

- la descrizione è quasi identica a quella di una funzione C per lo stesso algoritmo
- ma con una sostanziale differenza pratica:

  l'esecuzione degli assegnamenti in C è sequenziale, quella nel datapath è parallela

  dove la produzione di ogni output richiede un solo ciclo di clock!

a condizione che i dati in ingresso siano tutti simultaneamente disponibili ...

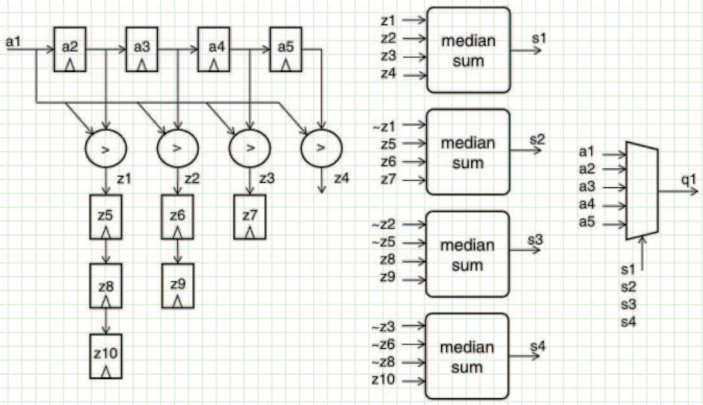
una FSMD può fare risparmiare molto spazio, come stiamo per vedere

datapath di un filtro di calcolo di mediana

l'algoritmo esposto impegna 192 linee di I/O e 10 comparatori

un filtro in streaming può ridurre l'I/O a 64 linee e impiegare solo 4 comparatori

a tal fine l'hardware usa registri per i quattro dati precedenti l'ultimo dalla stream di input e a ogni iterazione con un nuovo dato in input riusa i risultati memorizzati di sei confronti dalle tre iterazioni precedenti



Schaumont, Figure 5.9 - Median-calculation datapath for a stream of values

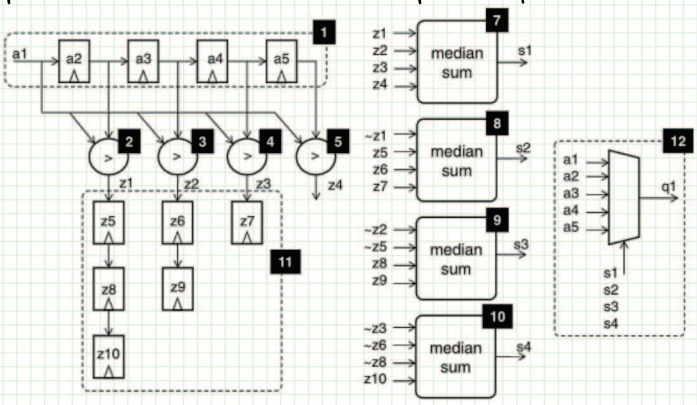
```
dp median(in a1 : ns(32); out q1 : ns(32)) {
   reg a2, a3, a4, a5 : ns(32);
   sig z1, z2, z3, z4;
   reg z5, z6, z7, z8, z9, z10 : ns(3);
   sig s1, s2, s3, s4, s5 : ns(1);
   always {
      a2 = a1;
      a3 = a2;
      a4 = a3;
      a5 = a4;
      z1 = (a1 < a2);
      z2 = (a1 < a3);
      z3 = (a1 < a4);
      z4 = (a1 < a5);
      z5 = z1;
      z6 = z2;
      z7 = z3;
      z8 = z5;
      z9 = z6;
      z10 = z8;
      s1 = ((     z1 +      z2 +      z3 +      z4) == 2);
      s2 = (((1-z1) +      z5 +      z6 +      z7) == 2);
      s3 = (((1-z2) + (1-z5) +      z8 +      z9) == 2);
      s4 = (((1-z3) + (1-z6) + (1-z8) +    z10) == 2);
      q1 = s1 ? a1 : s2 ? a2 : s3 ? a3 : s4 ? a4 : a5;
   }
}
```

sequenzializzazione mediante una FSMD

il filtro in fig. 5.9 accetta un nuovo input e produce un nuovo output a ogni ciclo di clock

si può ridurre ancora il numero dei componenti di calcolo distribuendo la computazione su più cicli di clock e imponendo una schedule sequenziale all'esecuzione delle operazioni, sì da riusare un solo comparatore e un solo modulo per il calcolo di s1, s2, s3, s4, a costo di aggiungere dei multiplatori e registri per i segnali interni

la figura illustra la schedule, la FSMD che realizza questa idea è in Schaumont Sez. 5.5.4, dove è anche presentata la FSMD di un testbench riprodotta qui accanto

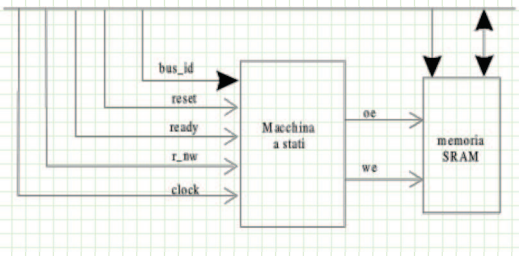


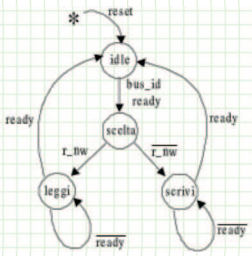
Schaumont, Figure 5.10 – Sequential schedule median-calculation datapath for a stream of values

```
dp t_median {
   sig istr, ostr : ns(1);
   sig a1_in, q1 : ns(32);
   use median(istr, a1_in, ostr, q1);
   reg r : ns(1);
   reg c : ns(16);
   always { r = ostr; }
   sfg init { c = 0x1234; }
   sfg sendin { a1_in = c;
                c = (c[0] ^ c[2] ^ c[3] ^ c[5]) # c[15:1];
                istr = 1; }
   sfg noin { a1_in = 0;
              istr = 0; }
}
fsm ctl_t_median(t_median) {
   initial s0;
   state s1, s2;
   @s0 (init, noin) -> s1;
   @s1 (sendin) -> s2;
   @s2 if (r) then (noin) -> s1;
              else (noin) -> s2;
}
```


descrizione di FSM in VHDL

un semplice esempio: controllore di memoria

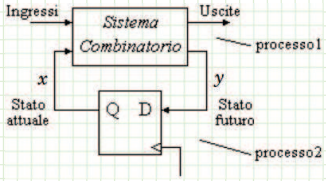




stato	uscite oe	we
idle	0	0
scelta	0	0
leggi	1	0
scrivi	0	1

Diagramma delle transizioni di un controllore di memoria

struttura generale hardware di FSM



Struttura hardware di FSM

nella descrizione VHDL, gli stati formano un tipo enumerato

stili di descrizione (v. codice allegato):

- con *due processi*: il modo più immediato di realizzare lo schema in figura, il clock attiva uno dei due processi
  - è lo stile della traduzione con fdlvhd
- con *un solo processo*: un solo segnale di stato, per FSM di Moore le uscite possono specificarsi con assegnamenti concorrenti, esterni al processo
- con *tre processi*: funzioni di transizione e di uscita descritte da processi combinatori distinti

tecniche di codifica degli stati

serve una codifica binaria del tipo enumerato dello stato per la sintesi di una descrizione in HDL di FSM
generata automaticamente dal sintetizzatore se non è specificata esplicitamente nella descrizione

codifiche in uso frequente:

- *sequenziale*: compatta, un insieme di n stati è codificato nell'insieme delle rappresentazioni binarie dei numeri naturali $< n$, con $\lceil \log_2 n \rceil$ bit
- *one-hot*: con tanti bit quanti sono gli stati, dove solo un bit è posto a 1 nella codifica (biiezione fra stati e posizioni dei bit nella codifica); codifica spesso generata dalla sintesi automatica
- *ad-hoc* per ottimizzare il ritardo, v. pagina appresso

la codifica esplicita può descriversi in VHDL definendo il tipo degli stati quale sottotipo di STD_LOGIC_VECTOR e dichiarando le codifiche come costanti di tale tipo; per esempio, ecco codifiche sequenziale e *one-hot* degli stati nell'esempio visto:

```
subtype stato is std_logic_vector(1 downto 0);
   constant idle : stato := "00";
   constant scelta : stato := "01";
   constant leggi : stato := "10";
   constant scrivi : stato := "11";
```

```
subtype stato is std_logic_vector(3 downto 0);
   constant idle : stato := "0001";
   constant scelta : stato := "0010";
   constant leggi : stato := "0100";
   constant scrivi : stato := "1000";
```

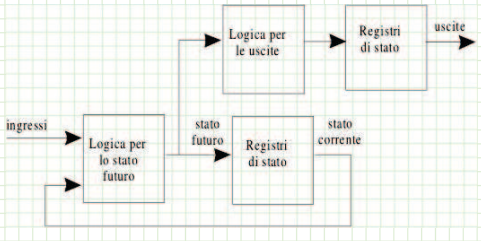
se la codifica non è suriettiva, *e.g.* una codifica *one-hot*, il costrutto `case stato is when ...` deve avere la clausola finale `when others ...`

ottimizzazione del ritardo

Uscite in funzione dello stato corrente (FSM di Moore)

lo schema a blocchi corrisponde alle descrizioni con due processi e con un solo processo nel codice allegato

il ritardo clock-to-output può risultare significativo

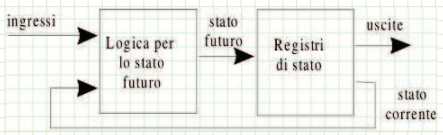


Uscite in funzione dello stato futuro

si può ridurlo al ritardo clock-to-q dei flip-flop calcolando le uscite in base al prossimo stato e anteponendo registri alle uscite

l'architettura con tre processi nel codice allegato esemplifica questa ottimizzazione

N.B. poiché i processi sono *sequenziali*, l'ottimizzazione è applicabile anche in un'architettura con due processi o con uno solo, posponendo l'aggiornamento delle uscite a quello dello stato



Riduzione del ritardo clock-uscita

un'altra tecnica di ottimizzazione consiste nel codificare lo stato in modo da far coincidere le uscite con alcuni dei bit di stato

e.g. nell'esempio visto:

idle	0 0 0
scelta	0 0 1
leggi	0 1 0
scrivi	1 0 0

esperienza di laboratorio

questa esperienza ha un duplice obiettivo:

1. estensione del modello dataflow prodotto nella terza esperienza di laboratorio per il calcolo del delay di traiettorie di Collatz a un modello FSMD che incorpori un generatore pseudocasuale per l'alimentazione del datapath con valori iniziali delle traiettorie, in modo simile all'istruzione `sendin` nell'esempio di testbench del filtro di calcolo della mediana, e alterni l'output del valore iniziale generato a quello del delay calcolato per la relativa traiettoria
   per il primo obiettivo si richiede la traduzione in VHDL e la simulazione per collaudarne la correttezza
2. realizzazione del modello sulla FPGA della DE1-SoC, con input di controllo dell'utente per abilitare la visualizzazione alterna, su schiera di cinque display a sette segmenti, in rappresentazione decimale

l'I/O del modello da realizzare per il primo obiettivo consta di

- input binario `start`, per abilitare l'inizio di una nuova traiettoria, con output del valore iniziale (da 16 bit) generato
- output binario `done`, di segnalazione della conclusione del calcolo del delay di una traiettoria
- input binario `dout`, per abilitare l'output del delay calcolato, sulla stessa porta di output del punto iniziale
- output `display` di numeri da 16 bit, multiplato per l'output alterno di punti iniziali e delay come indicato sopra

si suggerisce che la FSM di controllo, dopo lo stato iniziale di inizializzazione del generatore, attraversi in sequenza quattro stati:

1. attesa dell'input di controllo `start`
2. output del dato di inizio della traiettoria, avvio del calcolo del delay e attesa della sua conclusione
3. output `done` e attesa dell'input di controllo `dout`
4. output del delay e ritorno allo stato 1

per il secondo obiettivo, si può riusare il decodificatore per display a 7 segmenti impiegato nelle due esercitazioni precedenti, ma occorre alimentarlo con rappresentazioni BCD delle cifre dell'output a 16 bit: per questo occorrono cinque display ed è disponibile il package `bcd`, che fornisce la funzione `BCD_encode`
usare i tasti `KEY0`, `KE1`, `KEY2` per l'input `RST`, `start`, `dout`, il LED `LEDR0` per l'output `done`, e `CLOCK_50` per il clock

riferimenti

letture raccomandate:

Schaumont, Ch. 5, Sect. 5.4.4-5.5

letture per ulteriori approfondimenti:

C. Brandolese, Introduzione al linguaggio VHDL (Politecnico di Milano), Cap. 8

Zwoliński, Ch. 7, Sect. 7.1-2

materiali utili per l'esperienza di laboratorio proposta

Note sul VHDL, Corso di Architettura dei Sistemi Integrati, Capp. 10.1, 11, 13

sorgenti VHDL: decodificatore per display a 7 segmenti, package `bcd`

sorgenti di esempi in questa presentazione:

Gezel: Median parallelo, Median sequenziale,

VHDL: controllore SRAM