CENTRO APPUNTI
centroappunti.it
CORSO LUIGI EINAUDI, 55/B - TORINO

Appunti universitari
Tesi di laurea
Cartoleria e cancelleria
Stampa file e fotocopie
Print on demand
Rilegature

NUMERO: **2510A** ANNO: 2021

A P P U N T I

STUDENTE: Petitto Gabriele

MATERIA: Elettronica dei sistemi digitali - prof. Zamboni

Il presente lavoro nasce dall'impegno dell'autore ed è distribuito in accordo con il Centro Appunti.
Tutti i diritti sono riservati. È vietata qualsiasi riproduzione, copia totale o parziale, dei contenuti inseriti nel presente volume, ivi inclusa la memorizzazione, rielaborazione, diffusione o distribuzione dei contenuti stessi mediante qualunque supporto magnetico o cartaceo, piattaforma tecnologica o rete telematica, senza previa autorizzazione scritta dell'autore.

ATTENZIONE: QUESTI APPUNTI SONO FATTI DA STUDENTI E NON SONO STATI VISIONATI DAL DOCENTE.
IL NOME DEL PROFESSORE, SERVE SOLO PER IDENTIFICARE IL CORSO.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 238

Elettronica dei sistemi digitali

Gabriele Petitto

Marzo 2021

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 7 di 238

Capitolo 1

Richiami di algebra booleana

1.1 Algebra booleana

L'algebra di Boole (anche detta **algebra booleana**), in matematica e logica matematica, è il ramo dell'algebra in cui le variabili possono assumere solamente i valori vero e falso (valori di verità), generalmente denotati rispettivamente come 1 e 0. La rappresentazione tabellare di una funzione booleana prende il nome di tabella di verità della funzione:

x_1	x_2	$x_1 + x_2$	$x_1 \cdot x_2$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Tabella 1.1: Tabella della verità delle funzioni OR e AND

L'algebra booleana si basa su una serie di assiomi e teoremi che ne definiscono le regole di funzionamento. Di seguito sono brevemente richiamati:

	Assiomi		Teoremi
1a.	$0 \cdot 0 = 0$	1t.	$x \cdot 0 = 0$
2a.	1+1=1	2t.	x+1=1
3a.	$1 \cdot 1 = 1$	3t.	$x \cdot 1 = x$
4a.	0+0=0	4t.	x+0=x

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 7 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 9 di 238

È facile verificare che entrambi i membri danno la stessa tabella di verità: le funzioni sono assolutamente equivalenti.

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Tabella 1.3: Tabella di verità dei membri dell'equazione 1.1

Com'è possibile costruire una funzione a partire dai risultati in output? Esistono fondamentalmente due modi, legati ai concetti di minterm e maxterm.

1.2.1 Minterm

Un **minterm** è una funzione booleana che assume il valore '1' in corrispondenza di un'unica configurazione di variabili d'ingresso (booleane) indipendenti. E' quindi una funzione che si "accende" solo per determinate righe (sequenza di variabili d'ingresso). La funzione totale può essere espressa come la somma dei minterm associati alle righe che danno in output il risultato '1'. Tale espressione di una funzione booleana a più variabili è detta **SoP** (Sum of product)
Esempio:

x_1	x_2	x_3	f	minterm
0	0	0	1	$m_0 = \bar{x}_1\bar{x}_2\bar{x}_3$
0	0	1	0	$m_1 = \bar{x}_1\bar{x}_2x_3$
0	1	0	1	$m_2 = \bar{x}_1x_2\bar{x}_3$
0	1	1	1	$m_3 = \bar{x}_1x_2x_3$
1	0	0	1	$m_4 = x_1\bar{x}_2\bar{x}_3$
1	0	1	1	$m_5 = x_1\bar{x}_2x_3$
1	1	0	0	$m_6 = x_1x_2\bar{x}_3$
1	1	1	1	$m_7 = x_1x_2x_3$

$$f(x_1, x_2, x_3) = m_0 + m_2 + m_3 + m_4 + m_5 + m_7 \tag{1.2}$$

Per creare i minterm associati ad una riga è sufficiente esprimerli come prodotto delle variabili indipendenti negando quelle che assumono valore '0'.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 9 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 11 di 238

Capitolo 2

Linguaggio VHDL

2.1 Introduzione

Il **VHDL** (acronimo di VHSIC Hardware Description Language, dove "VHSIC" è la sigla di Very High Speed Integrated Circuits) è un linguaggio di descrizione dell'hardware nato da un progetto del Dipartimento della difesa statunitense. Il VHDL nasce nel 1987 quando diventa lo standard IEEE 1076 e nel 1993 ne esce una versione aggiornata, lo standard IEEE 1164. Il motivo per cui è stato rivisitato è perché la prima versione non era in grado di descrivere completamente un segnale elettrico: rappresentare il segnale esclusivamente come binario non era sufficiente. Per includere tipi di dati non binari è necessario iniziare la scrittura del codice con la libreria:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
```

È, insieme al Verilog, il linguaggio più usato per la progettazione di sistemi elettronici digitali. È lo strumento fondamentale per la progettazione dei moderni circuiti integrati digitali e le sue applicazioni spaziano dai microprocessori (DSP, acceleratori grafici), comunicazioni (cellulari, TV satellitare), automobili (navigatori, controllo di stabilità) a molte altre.

2.2 Sintassi del VHDL

Accenniamo brevemente alcuni dettagli sulla sintassi del linguaggio VHDL:

- il VHDL non è case sensitive: possono essere usati indifferentemente caratteri maiuscoli e minuscoli

  ```
  entity=ENTITY=Entity
  ```


© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 11 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 13 di 238

2.3.1 Segnali

I **segnali** sono i cavi dei circuiti che sintetizziamo, sono identificati tramite caratteri alfanumerici ed eventualmente dall'underscore.
E' necessario però seguire alcuni accorgimenti nell'assegnazione dei nomi: devono iniziare con una lettera, non possono terminare con l'underscore, non possono esserci due underscore successivi e non possono contenere keywords di VHDL.

La dichiarazione di un segnale si fa nel seguente modo:

```
SIGNAL nome_segnale : tipo_segnale ;
```

Vediamo alcuni esempi di dichiarazione per esaminare i vari tipi di segnale che useremo:

```
SIGNAL x1 : BIT;
-- bit singolo

SIGNAL C : BIT_VECTOR( 1 TO 4 );
-- array di bit, può rappresentare un numero binario.
-- La notazione (1 TO 4) indica che il bit più
-- significativo è in posizione 1.
-- Esempio:
-- C<="1010" allora C(1)=1

SIGNAL Byte : BIT_VECTOR( 7 DOWNTO 0 )
-- array di bit, ovvero un numero binario. La notazione (7 DOWNTO 0)
-- indica che il bit più significativo è in posizione 7.

SIGNAL n : STD_LOGIC;
-- numero non binario {1,0,Z ...}

SIGNAL X : STD_LOGIC_VECTOR( 15 DOWNTO 0 );
-- array di numeri non binari

SIGNAL W : INTEGER := 0;
-- numero intero, attenzione: l'inizializzazione :=0, in generale,
-- non va fatta durante la dichiarazione

USE ieee.std_logic_arith.all;
SIGNAL Y : SIGNED( 7 DOWNTO 0 );
-- rappresenta un numero binario a 8 bit con segno
-- scritto in complemento a 2
-- deve essere richiamato il relativo package

USE ieee.std_logic_arith.all;
```


© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 13 di 238

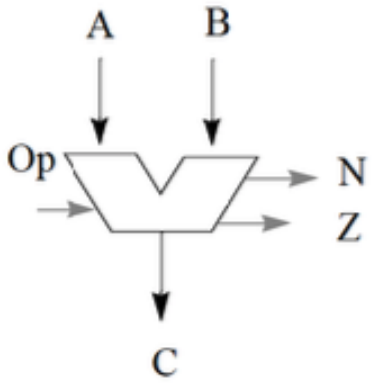
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 15 di 238

2.4.1 Entity

L'**entity** descrive le modalità con cui un oggetto si interfaccia con l'esterno. La dichiarazione di un' entity avviene nel seguente modo:

```
ENTITY nome_entity is
PORT( nomi_segnali : [modo] tipo_segnali);
END ENTITY nome_entity;
```

Esaminiamo un esempio:




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ALU32 IS
PORT( A, B: IN STD_LOGIC_VECTOR (31 DOWNTO 0);
          C : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
          Op: IN STD_LOGIC_VECTOR (5 DOWNTO 0);
          N, Z: OUT STD_LOGIC_VECTOR);
END ENTITY ALU32;
```

I **modi** indicano come il segnale si interfaccia all'entità, non esistono solo i modi di input e di output (che sono i più usati):

2.4.2 Architecture

La vista architetturale di un circuito si può dividere in due parti dal punto di vista descrittivo:

- parte dichiarativa, nella quale dichiaro che oggetti mi servono per descrivere l'architecture
- parte esecutiva (il body), nella quale descrivo gli oggetti dal punto di vista fisico

La dichiarazione di un architecture avviene nel seguente modo:

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 15 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 17 di 238

```
ENTITY half_adder IS
    PORT (a, b : IN STD_LOGIC);
          sum, carry : OUT STD_LOGIC);
END half_adder;

ARCHITECTURE behavioral OF half_adder IS
-- in questo caso non serve la parte dichiarativa
BEGIN
    sum <= (a XOR b);
    carry <= (a AND b);
END ARCHITECTURE behavior;
```

2.5 Assegnazioni dei segnali

Analizziamo quali sono le diverse tipologie di assegnazioni con cui attribuire ad un segnale un certo valore che, a seconda del metodo, può essere dipendente o meno da altre condizioni.

- Assegnazione semplice:

```
signal_name <= expression;
-- esempio:
SIGNAL x1, x2, x3, x4, f : STD_LOGIC;
f <= (x1 AND x2) OR (x3 AND x4);
```

- Assegnazione OTHERS:

```
SIGNAL S : STD_LOGIC_VECTOR ( n-1 DOWNTO 0);
S <= (OTHERS => '0');
-- assegna '0' a tutte le componenti del vettore che
-- non ho definito in modo diverso
-- per ora basta sapere che è un modo per
-- inizializzare registri e memorie
```

- Assegnazione selezionale:

```
WITH espressione SELECT
    nome_segnale <= espressione1 WHEN valore_costante1,
                    espressione2 WHEN valore_costante2;
-- attenzione! valore_costante1 e valore_costante2 devono
-- essere mutualmente esclusivi
-- E' meglio usare l' assegnazione condizionale
```

- Assegnazione condizionale:

```
nome_segnale <= espressione1 WHEN espressione_logica1 ELSE
                espressione2 WHEN espressione_logica2 ELSE
                espressione3;
```


© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 17 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 19 di 238

Figura 2.3: Multiplexer

Per descrivere il segnale in uscita Y c'è bisogno di una associazione con condizione. Può essere sia selezionale, sia condizionale:

```
-- assegnazione codizionale
Y <=A(0) WHEN S ="000" ELSE
    A(1) WHEN S ="001" ELSE
    A(2) WHEN S ="010" ELSE
    A(3) WHEN S ="011" ELSE
    A(4) WHEN S ="100" ELSE
    A(5) WHEN S ="101" ELSE
    A(6) WHEN S ="110" ELSE
    A(7);

-- assegnazione selezionale
WITH S SELECT
Y <=A(0) WHEN "000"
    A(1) WHEN "001"
    A(2) WHEN "010"
    A(3) WHEN "011"
    A(4) WHEN "100"
    A(5) WHEN "101"
    A(6) WHEN "110"
    A(7);
```

In modo alternativo e più compatto si può pensare di trasformare S in intero, per usarlo come indicatore:

```
Y <= A(TO_INTEGER(UNSIGNED(S)));
-- unsigned(S) specifica che il numero è
-- senza segno. E' necessario per
-- poterlo convertire a intero
```

Decoder Il decoder è un componente che in base alla combinazione dei bit presenti ai suoi ingressi, attiva una corrispondente combinazione di bit sulle linee di uscita.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 19 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 21 di 238

```
Y <= Conv_Std_Logic_Vector(2**(TO_INTEGER(UNSIGNED(S))),8) AND
     (7 DOWNTO 0 => A);
-- il decoder con enable può essere usato come demultiplexer
-- ma per questo corso non ci interessa
```

Prority encoder Il priority encoder (codificatore prioritario) è un circuito che comprime più ingressi binari in un numero inferiore di uscite. L'uscita di un priority encoder è la rappresentazione binaria del numero originale a partire da zero del bit di ingresso più significativo.

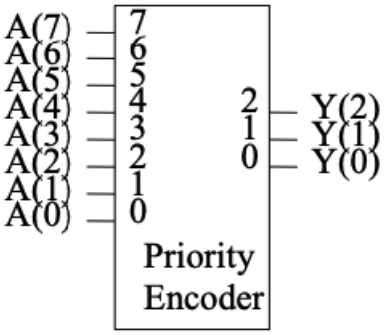


Figura 2.5: Priority encoder

```
Y <="111" when A(7) = '1' else
    "110" when A(6) = '1' else
    "101" when A(5) = '1' else
    "100" when A(4) = '1' else
    "011" when A(3) = '1' else
    "010" when A(2) = '1' else
    "001" when A(1) = '1' else
    "000";
```

2.6 Component e modellazione gerarchica

I **component** sono moduli VHDL che possono essere richiamati in altri circuiti logici digitali per implementare un circuito in maniera gerarchica. Invece di codificare un design complesso in un singolo codice VHDL, possiamo dividere il codice in sottomodelli, i component appunto, combinandoli usando la tecnica delle Port Map.
L'uso dei component agevola di gran lunga la costruzione di circuiti complessi, per fare ciò questi devono essere prima dichiarati:

```
-- dichiarazione quasi analoga ala entity
    COMPONENT nome_componente
        GENERIC (nome_parametri : INTEGER := valori_di_default);
```


© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 21 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 23 di 238

```
USE ieee.std_logic_1164.all ;

ENTITY mux4to1 IS
    PORT ( w0, w1, w2, w3 : IN STD_LOGIC ;
                        s : IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                        f : OUT STD_LOGIC ) ;
END mux4to1 ;

ARCHITECTURE Behavior OF mux4to1 IS
BEGIN
    WITH s SELECT
    f <= w0 WHEN "00",
         w1 WHEN "01",
         w2 WHEN "10",
         w3 WHEN OTHERS ;
END Behavior ;
```

Possiamo scrivere quindi il codice del multiplexer 16 to 1 in modo gerarchico ed estremamente più compatto definendo il component "mux4to1", ovvero il multiplexer 4 to 1:

```
-- file mux16to1.vhd
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
---------------------------------------- entity
ENTITY mux16to1 IS
PORT ( w : IN STD_LOGIC_VECTOR(0 TO 15) ;
       s : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
       f : OUT STD_LOGIC ) ;
END mux16to1 ;
---------------------------------------- architecture
ARCHITECTURE Structure OF mux16to1 IS
    SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
    -- dichiarazione component
    COMPONENT mux4to1
         PORT ( w0, w1, w2, w3 : IN STD_LOGIC ;
                             s : IN STD_LOGIC_VECTOR (1 DOWNTO 0) ;
                             f : OUT STD_LOGIC ) ;
    END COMPONENT ;
BEGIN
    Mux1: mux4to1 PORT MAP
       ( w(0), w(1), w(2), w(3), s(1 DOWNTO 0), m(0) ) ;
       -- associazione posizionale, più compatta
       -- l'ordine è importante
    Mux2: mux4to1 PORT MAP
       ( w(4) => w0, w(5)=> w1, w(6)=> w2, w(7)=> w3,
         s(1 DOWNTO 0) => s, m(1) => f ) ;
```


© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 23 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 25 di 238

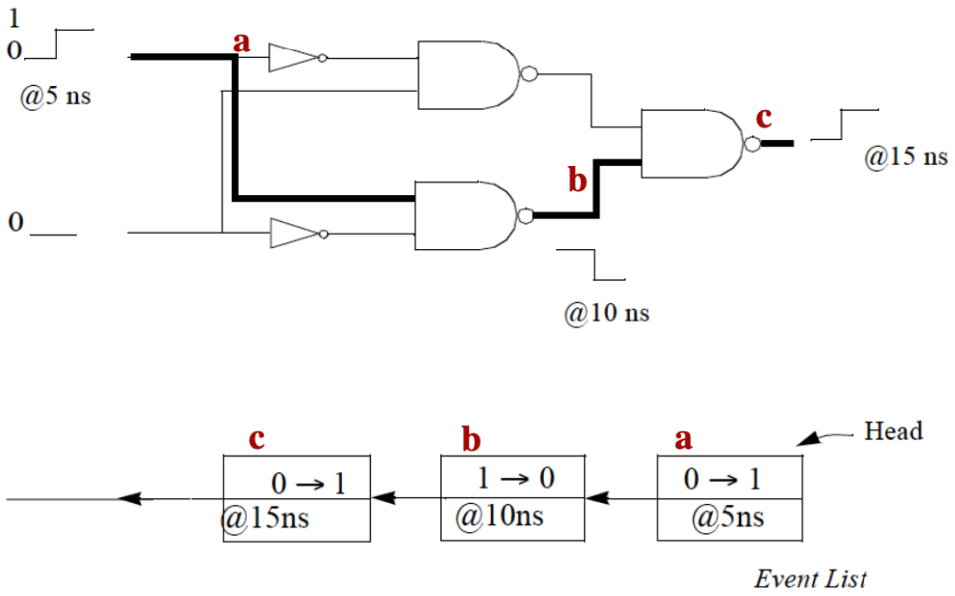
```
END ENTITY generic_or;

ARCHITECTURE behavioral OF generic_or IS
BEGIN
    or_out <= in1 OR in2;
END ARCHITECTURE behavioral;

-- file principale

ARCHITECTURE structural OF full_adder IS
COMPONENT generic_or
    GENERIC (n: positive);
    PORT (in1 : IN STD_LOGIC_VECTOR ((n-1) DOWNTO 0);
          in2 : IN STD_LOGIC_VECTOR ((n-1) DOWNTO 0);
       or_out : OUT STD_LOGIC_VECTOR ((n-1) DOWNTO 0) );
END COMPONENT;
-- righe di codice
BEGIN
    H1: half_adder PORT MAP (a => In1, b => In2, sum=>s1, carry=>s3);
    H2:half_adder PORT MAP (a => s1, b => c_in, sum =>sum, carry => s2
    O1: generic_or GENERIC MAP (n => 1)
                   PORT MAP (a => s2, b => s3, c => c_out);
END structural;
```

2.6.2 Costrutto GENERATE

Se dobbiamo instanziare un grande numero di components è improbabile pensare di instanziarli uno per uno nel body dell' architecture.
Il costrutto **generate** serve per iterare, attraverso un contatore intero i, questo procedimento, rendendo più facile e più compatta l'intera operazione.
Analizziamo direttamente un esempio pratico: il multiplexer 16 to 1 (già affrontato nella sezione 2.6):

```
---------------------------------------- architecture
ARCHITECTURE Structure OF mux16to1 IS
SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
    COMPONENT mux4to1
        PORT ( w0, w1, w2, w3 : IN STD_LOGIC ;
                              s : IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                              f : OUT STD_LOGIC ) ;
    END COMPONENT ;
BEGIN
    ---------------------------------------- inizio generate
    G1: FOR i IN 0 TO 3 GENERATE
        Muxes: mux4to1 PORT MAP(w(4*i+1), w(4*i+2),
                                w(4*i+3),s(1 DOWNTO 0), m(i));
```


© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 25 di 238



Il simulatore parte da @0ns, sapendo che non ci sono eventi fino a @5ns (dalla event list che gli è stata specificata) avanza fino a tale valore. Una volta raggiunto il primo evento effettua una simulazione lungo il path specificato (ciascuna simulazione è effettuata lungo un unico path), quando incontra l'evento successivo lo memorizza come tale assieme al relativo ritardo. Al termine di questa operazione il tempo di simulazione, fino ad ora fermo a @5ns, si sposta a @10ns ovvero all'istante di tempo in cui avviene l'evento appena simulato: @10ns. Si prosegue allo stesso modo per tutto l'intervallo di tempo di simulazione.

2.7.1 Modelli di ritardo in VHDL

Il linguaggio VHDL usa tre modelli di ritardo diversi:

- **ritardo inerziale**: è il modello di ritardo di default, si scrive nel seguente modo:

  ```
  segnale_di_output <= segnale_di_input AFTER 3 ns
  ```

  serve per descrivere il comportamento dell'harware , è un ritardo interno della porta e in quanto inerziale non è in grado di descrivere variazioni di segnale più piccole di se stesso le quali vengono, quindi, ignorate.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 27 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 29 di 238

Capitolo 3

Sintesi di circuiti combinatori

In elettronica digitale il modello più generale di circuito è quello composto da n ingressi ed m uscite entrambi funzioni del tempo. Se in ogni istante il valore istantaneo delle uscite è determinato univocamente da valori degli ingressi il circuito è detto **combinatorio**, allora esso non dipende esplicitamente dal tempo. Quando invece il valore istantaneo delle uscite dipende anche dalla storia passata del circuito allora si parla appunto di circuito **sequenziale**. In questo capitolo ci occuperemo del processo di sintesi dei circuiti combinatori.

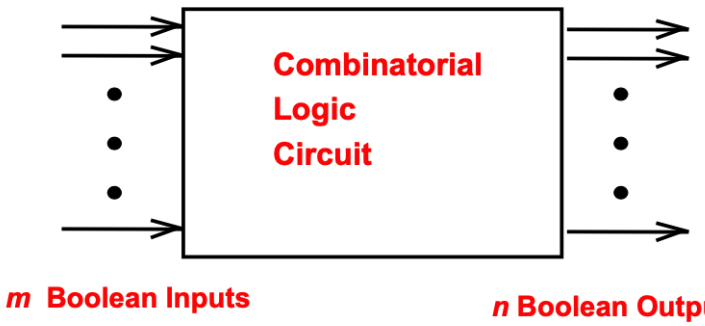


Figura 3.1: Diagramma a blocchi di un circuito logico combinatorio

Il processo che a partire dall'idea iniziale realizza il circuito fisico vero e proprio prende il nome di design del circuito è si divide in 5 fasi:

1. Specifica: si scrive la specifica del circuito, ovvero come si vuole che questo funzioni e cosa deve fare
2. Formulazione: si deriva la tabella di verità o la funzione booleana che relazione gli input con gli output

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 29 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 31 di 238

e colonna i valori delle variabili a cui il minterm è associato:

x \ y	0	1
0	$\bar{x}\bar{y}$	$\bar{x}y$
1	$x\bar{y}$	xy

Definiamo adesso il concetto di implicante, con il quale poi potremmo spiegare il metodo di semplificazione tramite mappa di Karnaugh. Un **implicante** è il termine comune a delle celle adiacenti, raggruppate secondo le potenze di due: un implicante di grado n=1 raggruppa $2^n = 2$ celle adiacenti, un implicante di grado n=2 raggruppa $2^n = 4$ celle e così via.
Come ci aiuta questa definizione nelle semplificazioni? Facciamo un esempio: si vuole semplificare la funzione:

$$y = \bar{x}\bar{y} + \bar{x}y \tag{3.1}$$

La funzione 3.1 semplificata corrisponde all'implicante tra le celle $\bar{x}\bar{y}$ e $\bar{x}y$ ovvero $\bar{x}$:

x \ y	0	1
0	$\bar{x}\bar{y}$	$\bar{x}y$
1	$x\bar{y}$	xy

Ragionando analogamente si può pensare di semplificare funzioni più complesse che, per esempio, contengono due implicanti:

$$y = \bar{x}\bar{y} + \bar{x}y + xy = \bar{x} + y \tag{3.2}$$

x \ y	0	1
0	$\bar{x}\bar{y}$	$\bar{x}y$
1	$x\bar{y}$	xy

x \ y	0	1
0	$\bar{x}\bar{y}$	$\bar{x}y$
1	$x\bar{y}$	xy

Allo stesso modo partendo dalla mappa di Karnaugh si può pensare di ricavare la funzione:

x \ y	0	1
0	1	1
1	0	1

x \ y	0	1
0	1	1
1	0	1

$$y = \bar{x}\bar{y} + \bar{x}y + xy = \bar{x} + y \tag{3.3}$$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 31 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 33 di 238

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Quindi costruiamo la mappa di Karnaugh della funzione:

x \ yz	00	01	11	10
0	0	1	0	0
1	0	1	1	1

x \ yz	00	01	11	10
0	0	1	0	0
1	0	1	1	1

La funzione semplificata scritta in **MSP, minimum sum of product,** è:

$$f(x, y, z) = \bar{y}z + xy \tag{3.5}$$

Si può scrivere anche in **MPS, minimum product of sum,** considerando nella mappa di Karnaugh i maxterm anziché i minterm:

x \ yz	00	01	11	10
0	0	1	0	0
1	0	1	1	1

x \ yz	00	01	11	10
0	0	1	0	0
1	0	1	1	1

La funzione semplificata in MPS è:

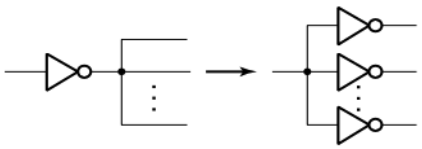
$$f(x, y, z) = (x + \bar{y}) + (y + z) \tag{3.6}$$

3.1.1 Don't care

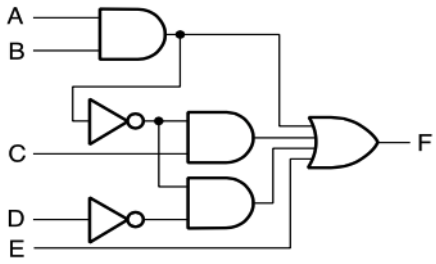
Talvolta potremmo non essere interessati ai valori che la funzione booleana assume con particolari combinazioni di variabili d'ingresso, in questo caso, per tali combinazioni, si specifica che la funzione "assume" il valore **don't care** (simboleggiato da -), ovvero un valore "jolly" che nelle mappe di Karnaugh può essere interpretato come '1' o '0' a seconda della convenienza:

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 33 di 238

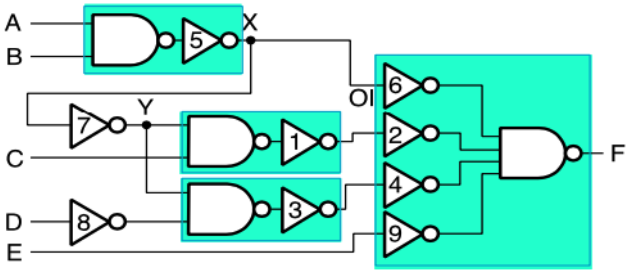
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 35 di 238



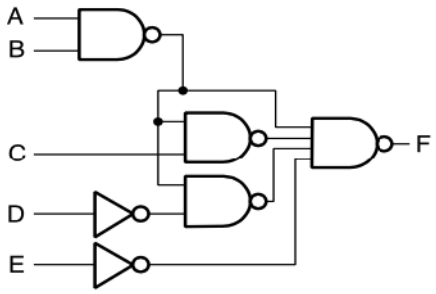
Esempio: Mappare con tecnologia NAND il seguente circuito:



Primo step: sostituire le porte logiche AND/OR



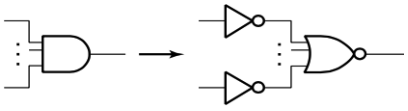
Secondo step: semplificare gli inverter



3.2.2 Mapping con tecnologia NOR

L'algoritmo per mappare una data funzione booleana esclusivamente usando tecnologia NOR è il seguente:

- Rimpiazzare le porte logiche AND (con NOR avente due inverter come ingressi) e OR (con NOR avente + inverter):



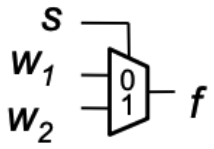
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 35 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 37 di 238

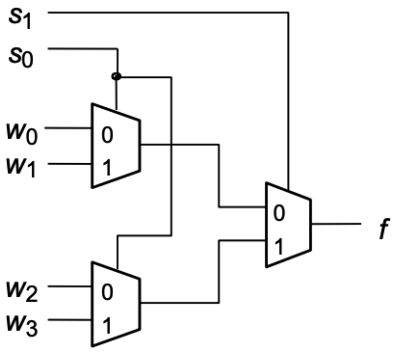
3.2.3 Mapping con tecnologia Multiplexer

Come abbiamo già visto nella sezione 2.5.1 il multiplexer è un selettore di linee di dato in grado di selezionare diversi segnali in ingresso. Una volta selezionati i segnali vengono raccolti e mandati in una singola linea di uscita.
Tipicamente un multiplexer ha n selezionatori di input $(S_0, S_1, ..., S_{n-1})$, 2^n dati in input ($w_{2^{n-1}}$, ..., w_0) e un output (Y), ma può essere progettato anche in modo da avere $m < 2^n$ dati in input.

Il multiplexer più elementare è il 2-to-1 multiplexer che, come si evince dal nome, riceve 2 dati in input, e, attraverso il selezionatore di input che è un singolo bit, sceglie quale dato far passare:



E' immediato intuire che multiplexer con $n > 1$ possono facilmente essere costruiti con un opportuno collegamento di 2-to-1 multiplexer (si veda figura 2.6):



quindi come è possibile mappare un circuito utilizzando una tecnologia multiplexer? E' estremamente semplice.
Si supponga di voler realizzare un circuito descritto dalla seguente tabella di verità:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 37 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 39 di 238

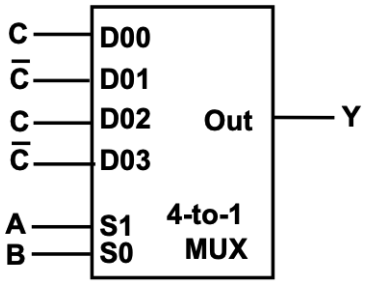


Figura 3.3: Mapping con multiplexer, approccio n.2

N.B.: Si può anche pensare di ridurre ulteriormente a uno (A) il numero di selezionatori di input, esprimendo poi l'uscita in funzione delle altre due variabili (B e C). Procedimento che diventa, ovviamente, più complesso.

Un metodo automatico per mappare una funzione a n variabili con un unico multiplexer consiste nello scomporre la suddetta funzione attraverso il teorema di espansione di Shannon.

Teorema di espansione di Shannon Il teorema dell'espansione di Shannon afferma che una qualsiasi funzione di n variabili booleane può essere scomposta nel seguente modo:

$$\begin{aligned} f(w_1, w_2, ..., w_n) &= \bar{w}_1 \cdot f(0, w_2, ..., w_n) + w_1 \cdot f(1, w_2, ..., w_n) = \\ &= \bar{w}_1 \cdot \bar{w}_2 \cdot f(0, 0, ..., w_n) + \bar{w}_1 \cdot w_2 \cdot f(0, 1, ..., w_n) + \\ &+ w_1 \cdot \bar{w}_2 \cdot f(1, 0, ..., w_n) + w_1 \cdot w_2 \cdot f(1, 1, ..., w_n) = \\ &= ... \end{aligned} \tag{3.8}$$

In questo modo è possibile realizzare un multiplexer con un numero di selettori di input pari al "grado di espansione della funzione". I selettori saranno le variabili rispetto cui si è realizzata l'espansione, i dati in input dipenderanno dalle restanti variabili.

Esercizio: Si mappi la seguente funzione booleana: $f = \bar{w}_1\bar{w}_3 + w_1w_2 + w_1w_3$ utilizzando un unico multiplexer con selettori di input w_1 e w_2.

Espandiamo la funzione rispetto a w_1 e w_2:

$$\begin{aligned} f = \bar{w}_1 \cdot \bar{w}_2 \cdot f(0, 0, w_3) + \bar{w}_1 \cdot w_2 \cdot f(0, 1, w_3) + \\ + w_1 \cdot \bar{w}_2 \cdot f(1, 0, w_3) + w_1 \cdot w_2 \cdot f(1, 1, w_3) \end{aligned} \tag{3.9}$$

Calcolando i cofattori otteniamo:

$$f = \bar{w}_1 \cdot \bar{w}_2 \cdot \bar{w}_3 + \bar{w}_1 \cdot w_2 \cdot \bar{w}_3 + w_1 \cdot \bar{w}_2 \cdot w_3 + w_1 \cdot w_2 \cdot 1$$

Ovvero:

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 39 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 41 di 238

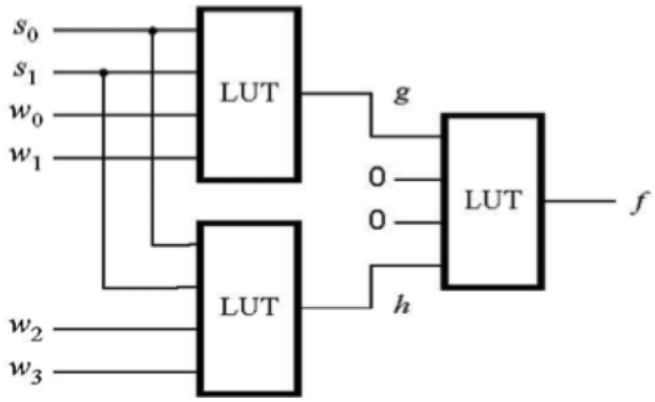


Figura 3.4: non ho capito perché il terzo blocco è un OR

3.3 Altre tecnologie

Richiamiamo brevemente altre tecnologie di rilievo, nonostante esse non siano usate, come quelle viste finora, per sintetizzare circuiti.

3.3.1 Decoder

Come abbiamo già visto nella sezione 2.5.1 il decoder è un componente che in base alla combinazione dei bit presenti ai suoi ingressi, attiva una corrispondente combinazione di bit sulle linee di uscita. Tipicamente un decoder ha n dati in input (w_{n-1}, ..., w_0), 2^n output (y_{2^n-1}, ..., y_0) e un enable (che setta le uscite a '0' quando vale '0').

Il decoder più elementare è il 2-to-4 decoder:

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	-	-	0	0	0	0

Tabella 3.4: Tabella di verità del 2-to-4 decoder

Altri più complessi possono facilmente essere costruiti con un opportuno collegamento di 2-to-4 decoder:

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 41 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 43 di 238

per un numero binario), quindi la combinazione d'ingresso può contenere più '1' (che verranno ignorati, don't care).

w_0	w_1	w_2	w_3	y_1	y_0	z
0	0	0	0	-	-	0
0	0	0	1	0	0	1
0	0	1	-	0	1	1
0	1	-	-	1	0	1
1	-	-	-	1	1	1

Tabella 3.8: Tabella di verità di un 4-to-2 priority encoder

3.3.3 Comparatore di bit

Un comparatore di bit è un circuito che compara una sequenza di bit confrontando bit per bit e restituendo '1' se la sequenza corrisponde, '0' altrimenti. Un comparatore di bit è realizzato tramite l'impiego delle porte logiche X-NOR la cui tabella di verità è di seguito illustrata:

a	b	f
0	0	1
0	1	0
1	0	0
1	1	1

Tabella 3.10: Tabella di verità di un X-NOR

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 43 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 45 di 238

Figura 3.8: Alee dinamiche

Come è possibile progettare un circuito in modo da sbarazzarsi delle alee statiche? Esaminiamo un esempio, il circuito di seguito illustrato dà un hazard statico quando passa da $(1,1,1) \rightarrow (0,1,1)$:

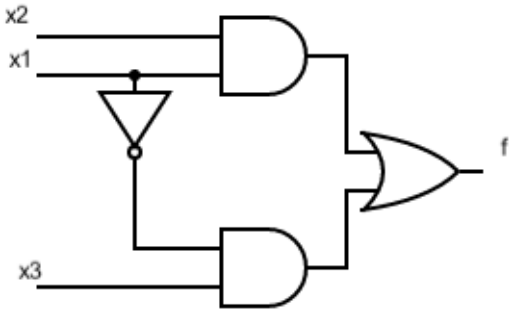


Figura 3.9: Circuito con hazard statico

Questo accade perché il circuito è stato sintetizzato dalla mappa di Karnaugh senza un implicante tra le celle $(1,1,1)$ e $(0,1,1)$:

x_3 \ x_1x_2	00	01	11	10
0	0	0	1	0
1	1	1	1	0

x_3 \ x_1x_2	00	01	11	10
0	0	0	1	0
1	1	1	1	0

$$f = \bar{x}_1 x_3 + x_1 x_2 \tag{3.10}$$

Per sbarazzarsi dell'hazard statico aggiungiamo alla funzione un implicante "ridondante" che comprenda le due celle in questione:

x_3 \ x_1x_2	00	01	11	10
0	0	0	1	0
1	1	1	1	0

La nuova funzione così ottenuta è:

$$f = \bar{x}_1 x_3 + x_1 x_2 + x_2 x_3 \tag{3.11}$$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 45 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 47 di 238

Capitolo 4

Logiche programmabili

Un dispositivo logico programmabile (Programmable Logic Device, PLD) è un circuito integrato programmabile largamente utilizzato nei circuiti digitali. A differenza di una porta logica, che implementa una funzione logica predefinita e non modificabile, un PLD, al momento della fabbricazione, non è configurato per svolgere una determinata funzione logica e dunque prima di poterlo utilizzare in un circuito è necessario programmarlo.

L'uso di PLD comporta molti vantaggi rispetto agli ASIC (Application Specific Integrated Circuit): si tratta infatti di dispositivi standard la cui funzionalità da implementare non viene impostata dal produttore, il quale può quindi produrre su larga scala a basso prezzo. Essi sono programmati direttamente dall'utente finale, consentendo la diminuzione dei tempi di progettazione, di verifica mediante simulazioni e di prova sul campo dell'applicazione. Il grande vantaggio rispetto agli ASIC è che permettono di apportare eventuali modifiche o correggere errori semplicemente riprogrammando il dispositivo in qualsiasi momento. Per questo motivo sono utilizzati ampiamente nelle fasi di prototipizzazione, in quanto eventuali errori possono essere risolti semplicemente riconfigurando il dispositivo. Di contro, per applicazioni su grandi numeri sono antieconomici, perché il prezzo unitario del dispositivo è superiore a quello degli ASIC (i quali hanno spesso, però, più elevati costi di progettazione).

Elenchiamo di seguito i principali dispositivi logici programmabili:

- **PLA** (Programmable Logic Array), costituiti da un array di gates AND programmabile, collegato ad un array di gates OR programmabile
- **PROM** (Programmable Read Only Memory), costituiti da un array di gates AND fissato, collegato ad un array di gates OR programmabile
- **PAL** (Programmable Array Logic), costituiti da un array di gates AND programmabile, collegato ad un array di gates OR fissato
- **FPGA** (Field- Programmable Gate Array)/**CPLD** (Complex Programmable Logic Device), complessi abbastanza da poter essere definiti "ar-

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 47 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 49 di 238

Si noti che per ottenere invece una struttura che funzioni come una porta logica AND è sufficiente negare gli ingressi:

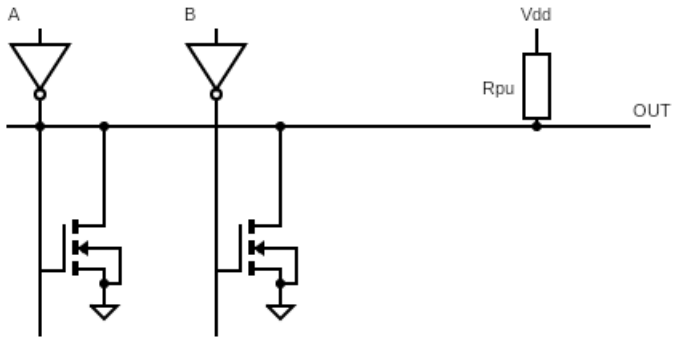


Figura 4.3: Struttura Wire-Or con ingressi negati (AND)

La programmabilità delle due strutture sta dunque nel fatto che per variare le funzioni in uscita si possono scollegare o meno gli ingressi dagli N-MOS rispettivi.

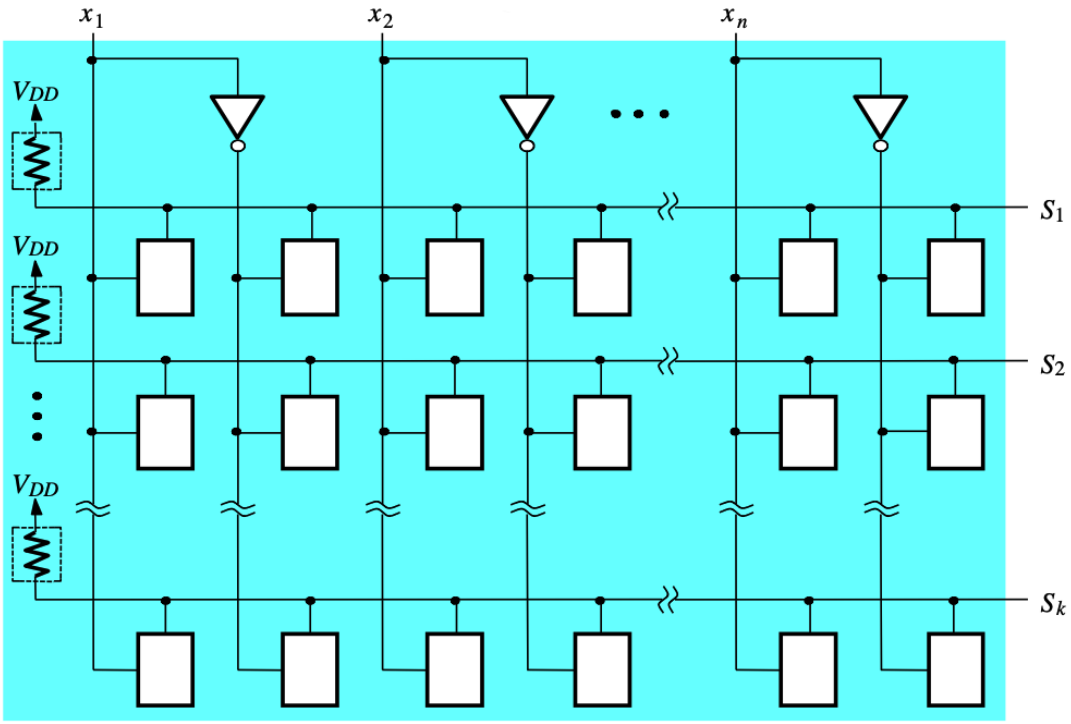


Figura 4.4: AND plane realizzato con più strutture Wire-Or i cui ingressi possono essere collegati direttamente o attraverso un inverter (negati), ottenendo tante uscite $(S_1, S_2, ..., S_k)$ quante sono le strutture

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 49 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 51 di 238

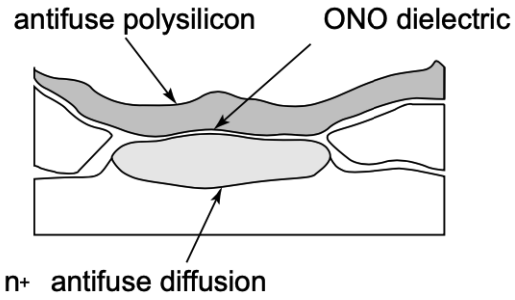


Figura 4.7: Antifusibile

Tecnologie RAM-based Le tecnologie RAM-based sono tecnologie riprogrammabili e volatili (non mantiene la memoria quando viene tolta l'alimentazione). Esse realizzano l'interruttore tra N-MOS e ground tramite un altro N-MOS pilotato da un bit di memoria RAM (quando vale '1' l'N-MOS realizza il collegamento a ground, altrimenti a '0' è un cortocircuito).

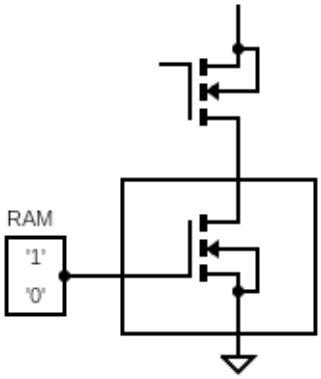


Figura 4.8: Tecnologia RAM-based

Tecnologie FLASH-based Le tecnologie FLASH-based sono tecnologie riprogrammabili e non volatili. Esse realizzano l'interruttore tra N-MOS e ground tramite un transistor EEPROM (E^2PROM) costituito da una cella flash che è possibile programmare in modo che sia sempre spenta o sempre accesa.

4.2 Logiche programmabili array-based

In questo sezione analizzeremo alcuni tra i principali dispositivi logici programmabili introdotti sul mercato a partire dagli anni '60, vale a dire PLA, PROM e PAL.

Il principio teorico alla base di questi dispositivi è stato già accennato nella sezione 1.2.1: qualsiasi funzione booleana a due o più variabili può essere espres-

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 51 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 53 di 238

- alcuni PLA presentano output che possono essere invertiti, ciò consente di esprimere funzioni anche nella forma POS

I principali difetti dei PLA sono invece:

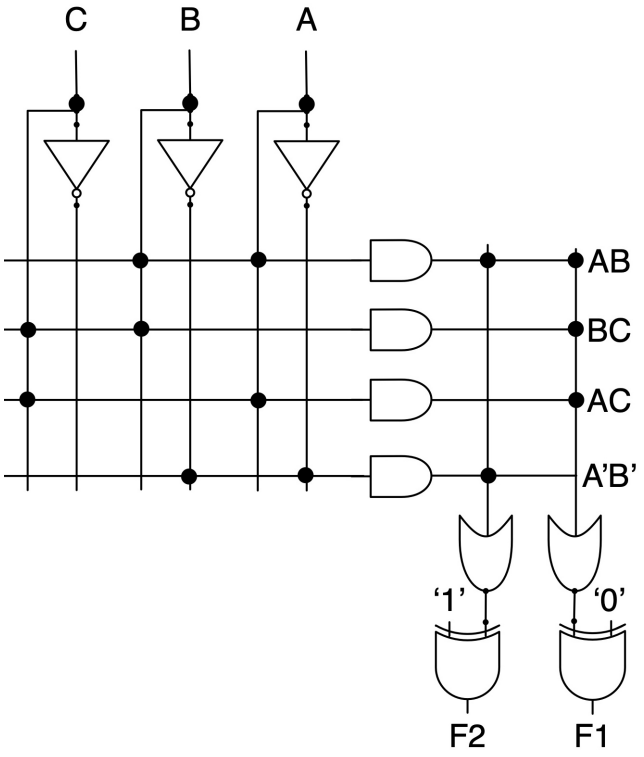
- non è detto che si riascano ad implementare tutte le funzioni possibili degli N ingressi (dovrei avere un PLA con 2^N righe)

Esempio: Si vuole realizzare un PLA a 3-input, 2-output con 4 termini prodotto che dia in uscita due funzione F1 e F2 tali che:

$F1 = A\bar{B} + \bar{A}B = A \oplus B$

$F2 = AB + BC + AC$

Poiché queste funzioni richiederebbero nell'array OR cinque termini prodotto diversi ma la PLA ha solo quattro righe, l'unico modo per realizzarla è collegando in uscita i segnali a due XOR:



Realizzazione con piani NOR Come abbiamo già visto, fisicamente siamo in grado di realizzare delle strutture Wire-Or che non sono né array OR né array AND ma si comportano come degli array NOR. E' possibile quindi programmare una funzione scritta in forma SOP attraverso due piani NOR? La risposta è sì, a patto di invertire l'uscita, in questo modo si lavora difatti con un array NOR e un array OR.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 53 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 55 di 238

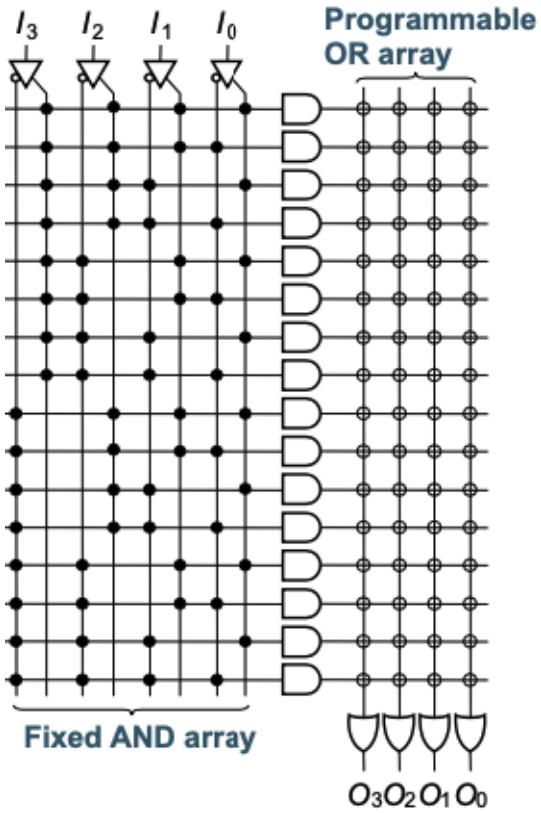


Figura 4.11: Dispositivi PROM

Una PROM ha N ingressi, M uscite e 2^N righe (termini prodotto), è a tutti gli effetti una memoria avente come input degli indirizzi di memoria (da qui il nome PROM). La parte programmabile sarebbe quindi l'array OR ovvero le locazioni di memoria, la parte fissa è l'array AND cioè gli indirizzi.

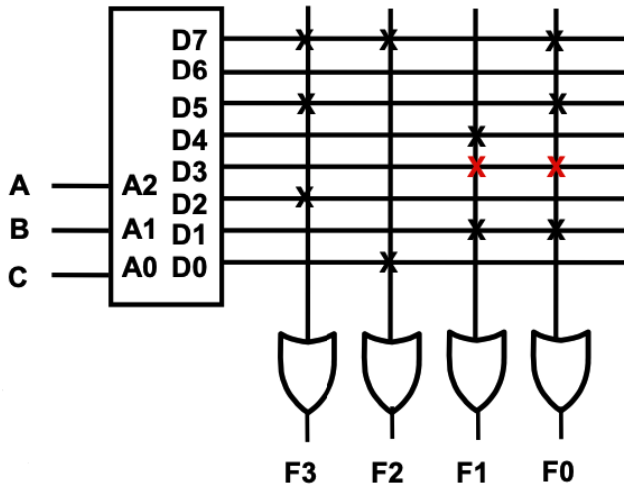
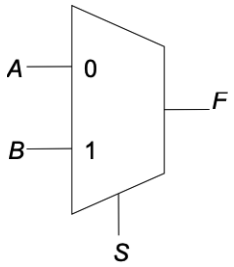


Figura 4.12: PROM utilizzata come memoria: all'indirizzo in ingresso (0,1,1) restituisce l'uscita (0,0,1,1)

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 55 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 57 di 238

all'interno non solo piani AND, OR ma anche flip-flops, XOR e altri elementi) e interconnessi tra loro, prendono il nome di "Complex Program-mable Logic Device" (CPLD).
Questi miglioramenti, tuttavia, non risolvevano il problema maggiore delle strutture Wire-Or: ciascuna delle linee è molto lunga e collegata a diversi transitor, ognuno dei quali ha una capacità parassita. Ciò non consente di lavorare a frequenze troppo alte.
Le tecnologie Wire-Or vennero dunque sostituite da tecnologie basate su celle logiche LUT. Anch'esse, a seconda degli ingressi, possono realizzare qualsiasi funzione booleana.



A	B	S	F
0	0	0	0
0	X	1	X
0	Y	1	Y
0	Y	X	XY
X	0	Y	$X\overline{Y}$
Y	0	X	$\overline{X}Y$
Y	1	X	X+Y
1	0	X	$\overline{X}$
1	0	Y	$\overline{Y}$
1	1	1	1

Tabella 4.2: Un 2-to-1 multiplexer può realizzare tutte le funzioni elementari a due variabili possibili, basta tenere fisso a '1' o '0' uno dei tre segnali in ingresso

Per realizzare funzioni più complesse basta collegare opportunamente 2-to-1 multiplexer, formando così una LUT (il numero di input della LUT definisce il numero di livelli di 2-to-1 mux che essa include al suo interno):

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 57 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 59 di 238

Come programmo fisicamente queste interconnessioni? Analogamente a quanto già accennato nella sezione 4.1. Ci riferiremo al caso delle interconnessioni RAM-based, per cui se in memoria programmo '1' realizzo la connessione, altrimenti è circuito aperto. Per scegliere il percorso di un dato all'interno della FPGA vengono programmati due tipi di box di connessioni:

- connect box, permette di collegare o meno le uscite dei blocchi logici alle linee di connessioni
- switch box, permette ai dati di cambiare o meno linea di connessione passando su quella longitudinale

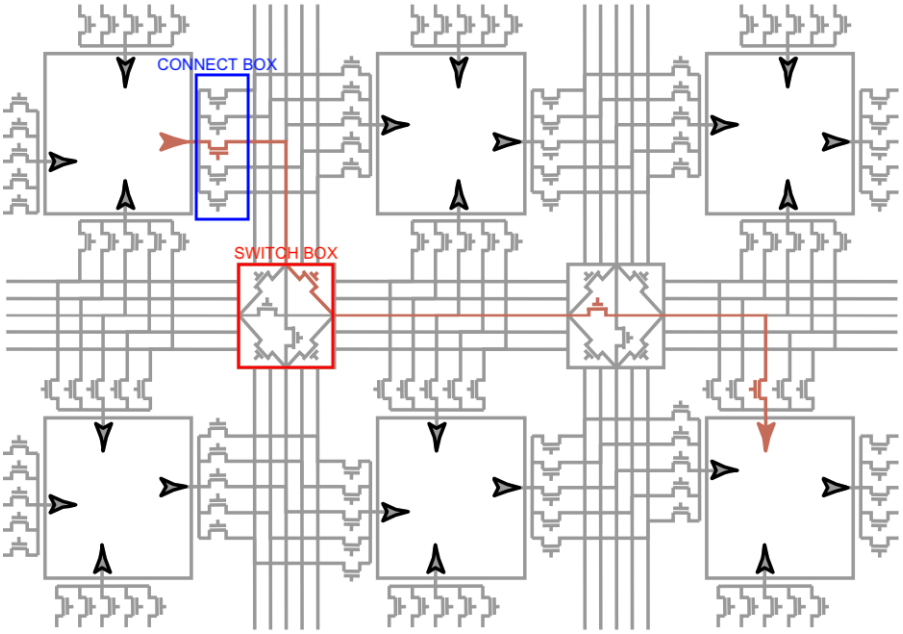


Figura 4.17: Maglia di connessione di una FPGA

4.4 Cenni ai circuiti integrati

Un "Application Specific Integrated Circuit" (ASIC), è un circuito integrato creato appositamente per risolvere un'applicazione di calcolo/elaborazione ben precisa (special purpose): la specificità della progettazione, focalizzata sulla risoluzione di un unico problema, consente di raggiungere delle prestazioni in termini di velocità di processamento e consumo elettrico difficilmente ottenibili con l'uso di soluzioni più generiche (general purpose).

Lo sviluppo di questo genere di circuiti è però molto costoso e per questo motivo il loro impiego è limitato a campi in cui possano essere usati in maniera massiccia, ovvero alti volumi di mercato, come l'elettronica di consumo (masterizzatori, schede elettroniche, apparati di rete, trasmissione audio-video) mentre per usi su scala più limitata vengono preferite, ad esempio, tecnologie riprogrammabili come le FPGA. Gli ASIC si dividono in:

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 59 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 61 di 238

Capitolo 5

Circuiti aritmetici

I circuiti aritmetici sono circuiti digitali in grado di implementare funzioni aritmetiche. In un primo momento analizzeremo operazioni tra numeri "unsigned", senza segno, tali per cui ciascuna cifra pesa solo il modulo del numero nella conversione decimale:
Esempio: $(1101)_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2 + 1 \cdot 2^0 = (13)_{10}$

5.1 Addizione tra numeri unsigned

Half Adder Per effettuare una somma tra numeri binari necessitiamo di una funzione che sommi bit a bit e restituisca due output: il risultato della somma e un eventuale riporto (carry).
La tabella di verità della suddetta funzione è la seguente:

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabella 5.2: Tabella di verità di un Half-Adder

L'elemento circuitale che sintetizza tale funzione prende il nome di Half-Adder:

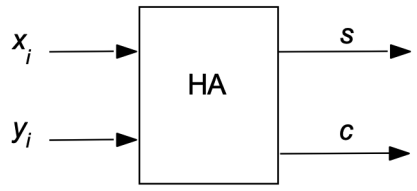


Figura 5.1: Simbolo circuitale dell' Half-Adder

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 61 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 63 di 238

Da cui ricaviamo le mappe di Karnaugh:

c_i \ x_iy_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0

c_i \ x_iy_i	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Tabella 5.8: Mappe di Karnaugh rispettivamente delle funzioni S_i e C_{i+1}

otteniamo:

$$S_i = x_i \oplus y_i \oplus c_i \tag{5.3}$$

$$C_{i+1} = x_i y_i + x_i c_i + y_i c_i \tag{5.4}$$

e quindi il corrispondente circuito:

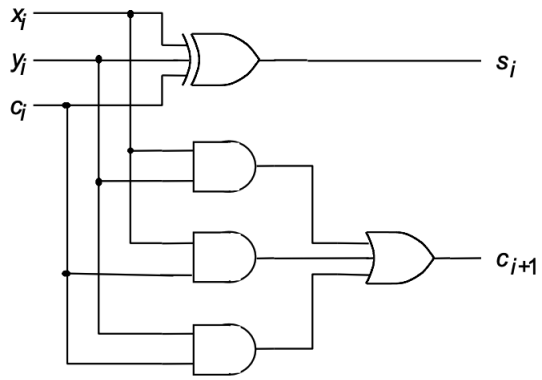


Figura 5.3: Descrizione circuitale elementare del Full-Adder

Si noti che esiste un altro modo per costruire un circuito full-adder, ovvero attraverso successive istanze del half-adder:

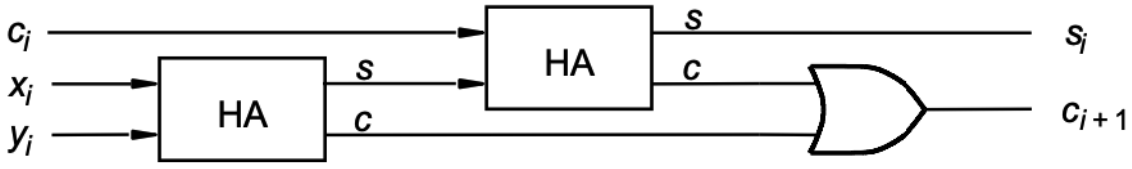


Figura 5.4: Implementazione del Full-Adder mediante blocchi di HA

E quindi, in riferimento alla Figura 5.2, otteniamo una descrizione circuitale del full-adder molto più intuitiva:

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 63 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 65 di 238

Scritto in VHDL:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
-- 4 bit ripple-carry adder

    ENTITY adder4 IS
        PORT ( Cin : IN STD LOGIC ;
    x3, x2, x1, x0 : IN STD_LOGIC ;
    y3, y2, y1, y0 : IN STD_LOGIC ;
    s3, s2, s1, s0 : OUT STD_OGIC ;
              Cout : OUT STD_LOGIC ) ;
    END adder4 ;

    ARCHITECTURE Structure OF adder4 IS
        SIGNAL c1, c2, c3 : STD_LOGIC ;
        -- dichiarazione  component full-adder
        COMPONENT fulladd
            PORT ( Cin, x, y : IN STD_LOGIC ;
                       s, Cout : OUT STD_LOGIC ) ;
        END COMPONENT ;
    BEGIN
        -- istanze dei 4 full-adder
        stage0: fulladd PORT MAP ( Cin, x0, y0, s0, c1 ) ;
        stage1: fulladd PORT MAP ( c1, x1, y1, s1, c2 ) ;
        stage2: fulladd PORT MAP ( c2, x2, y2, s2, c3 ) ;
        stage3: fulladd PORT MAP (Cin => c3, Cout => Cout,
                                  x => x3, y => y3, s => s3 ) ;
END Structure ;
```

Il simbolo con cui faremo riferimento al ripple-carry adder è il seguente:

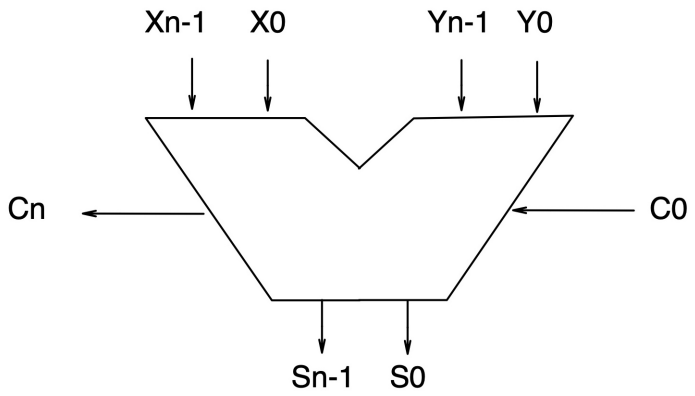


Figura 5.7: Ripple-Carry Adder, simbolo circuitale

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 65 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 67 di 238

5.3 Sottrazione tra numeri unsigned

Si immagini di voler sottrarre al minuendo M il sottraendo N, con M e N definiti come numeri unsigned (solo modulo) di n bit. Possiamo trovarci di fronte a due casi:

- $M \geq N$ allora non occorre il prestito finale, il risultato è semplicemente il numero che si ottiene con il metodo del prestito (borrow)
- M < N allora occorre il prestito finale, che manca. Il risultato può essere trovato sottraendo a 2^n (in binario) ciò che si ottiene dalla sottrazione senza borrow finale, ricordandosi che si sta trattando un numero negativo. Sottrazioni tra numeri unsigned quindi, in questo modo, sono molto complesse da sintetizzare.

Per risolvere il problema delle sottrazioni tra numeri binari unsigned introduciamo il concetto di complemento rispetto ad una radice (la radice è la base del sistema rispetto al quale è definita l'informazione, nel nostro caso r=2). Definiamo:

- complemento a 1 (di radice diminuita) di N, il numero definito come $(r^n - 1)$-N. Nel caso di radice binaria il complemento a 1 di un numero coincide con il suo stesso complementare:
  es: $CA_1((10001010)_2) = (01110101)_2$
- **complemento a 2** (di radice) di N, il numero definito come: $r^n - N$. Per i numeri binari esistono due modi immediati per calcolare il complemento a 2 di un numero N:
  - aggiungere '1' al complemento a 1: $CA_2 = CA_1 + 1$ (metodo usato dai calcolatori)
  - riportare uguali le prime cifre da sinistra fino al primo '1' (compreso) e complementare le altre cifre (metodo più semplice):
    es: $CA_2((10001010)_2) = (01110110)_2$

A cosa è servito definire il concetto di complemento?
Invece di calcolare M-N, calcoliamo $M + (2^n - N) = M + CA_2(N)$, notiamo che:

- se $M \geq N \rightarrow M - N = M + CA_2(N)$ scartando il riporto
- se $M < N \rightarrow M - N = -CA_2(N - M)$ ricordando che il segno è negativo

Abbiamo ricavato dunque un algoritmo abbastanza semplice per calcolare sottrazioni tra numeri unsigned. Si noti però che queste non hanno molto senso quando il risultato è un numero negativo, perché, con i numeri unsigned, non abbiamo modo di tenerne conto.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 67 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 69 di 238

5.4.1 Addizioni e sottrazioni in modulo e segno

L'algoritmo per svolgere addizioni e sottrazioni nella rappresentazione in modulo e segno è il seguente:

- se il segno dei due numeri è uguale:
  - il modulo si ottiene sommando i moduli, cioè numeri unsigned
  - il segno è lo stesso dei due numeri
- se il segno dei numeri è diverso (non si può avere overflow):
  - il modulo si ottiene sottraendo il modulo del secondo numero a quello del primo (è una sottrazione tra numeri unsigned, vedasi Sezione 5.3)
  - il segno è quello del numero con il modulo più grande

5.4.2 Addizioni e sottrazioni in complemento

Per svolgere addizioni e sottrazioni nella rappresentazione in complemento basta fare la somma dei numeri (comprensivi di bit di segno) come se fossero numeri unsigned, l'eventuale riporto:

- va sommato di nuovo alla somma senza riporto ottenuta, nel caso del complemento a 1
- va scartato, nel caso del complemento a 2 (per questo useremo sempre questa rappresentazione)

Bisogna poi, per entrambe le rappresentazioni, prestare attenzione al segno del risultato: se il segno dei due addendi è uguale (condizione necessaria per avere overflow) e il segno del risultato totale è diverso da questo, allora si è verificato overflow.
Un interpretazione grafica dell'aritmetica in complemento a 2 su 4 bit è la seguente:

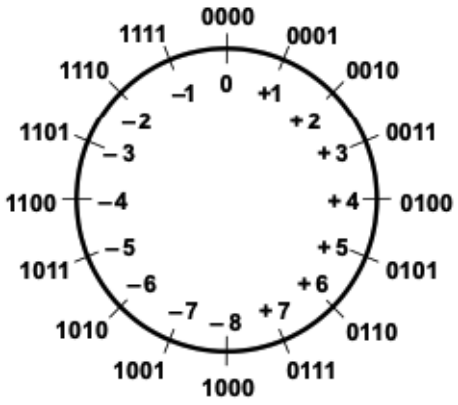


Figura 5.10: Schema grafico per aritmetica in CA_2

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 69 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 71 di 238

Come abbiamo già visto se in una somma di numeri in complemento a 2 il segno dei due addendi è uguale e il segno del risultato totale è diverso da questo, allora c'è stato overflow. Esaminando caso per caso è possibile dedurre:

$$O_v = C_n \oplus C_{n-1} \tag{5.8}$$

Se O_v vale '1' l'overflow si è verificato, altrimenti vale '0' (C_n e C_{n-1} sono rispettivamente l'ultimo e il penultimo riporto dell'operazione).
Esempio:

$C_4 = 0$	$C_3 = 1$	$C_2 = 1$	$C_1 = 0$	$C_0 = 0$
	0	1	1	1
+	0	0	1	0
	1	0	0	1

Tabella 5.14: $O_v = C_4 \oplus C_3 = 1$

Ma adesso si pone un altro problema: se non ci è permesso accedere all'interno del dispositivo sommatore/sottrattore (perché magari ci è stato venduto come blocco unico), come è possibile collegare ad una porta XOR C_{n-1} e C_n?
Supponiamo di avere un sommatore su 16 bit, sappiamo che:

$$C_{16} = Sum(16) \tag{5.9}$$

$$Sum(15) = C_{15} \oplus X(15) \oplus Y(15)$$

Possiamo manipolare la seconda equazione per ricavare C_{15} a partire dai termini che conosciamo:

$$X(15) \oplus Sum(15) = \cancel{X(15) \oplus X(15)} \oplus Y(15) \oplus C_{15}$$

Ripetendo con Y(15), otteniamo:

$$C_{15} = Sum(15) \oplus X(15) \oplus Y(15) \tag{5.10}$$

Mettendo in XOR i due termini iamo in grado di ricavare l'overflow del sistema.

5.5.1 Codice VHDL del sommatore/sottrattore

Adesso siamo in grado di descrivere il sommatore/sottrattore in modo completo e, quindi, il suo codice VHDL:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
-- usiamo il package numeric_std
-- perché dobbiamo fare operazioni tra numeri:
USE ieee.numeric_std.all ;
```


© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 71 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 73 di 238

5.6 Sommatori veloci

Il grande problema del ripple-carry adder è che è eccessivamente lento a causa del propagarsi del carry in ingresso attraverso i full-adder. Il ritardo più ampio si verifica quando il carry di ingresso si propaga fino all'ultimo full-adder, è quindi lineare rispetto al numero di bit su cui si lavora:

$$t_{adder} = (n-1)t_{carry} + t_{sum}$$

N.B.: E' importante ricordare che il ritardo è calcolato rispetto ai valori medi (50%) dei fronti di salita e di discesa, di modo che il ritardo complessivo di una catena di dispositivi sia additivo dei singoli ritardi.

L'obiettivo è aumentare la velocità di elaborazione del sommatore, vediamo quindi come realizzare dei **sommatori veloci** (fast-adder).

5.6.1 Carry-Bypass Adder

Il principio su cui si basa il carry-bypass adder è appunto quello di permettere al carry iniziale di bypassare la catena di full-adder quando il sommmatore si "accorge" che questo si propaga lungo tutta la catena. Come avviene questo? Richiamiamo l'equazione 5.6:

$$C_{i+1} = G_i + P_i C_i$$

La propagazione del carry attraverso un full-adder avviene quando propagate vale '1' quindi per passare dal primo al n-esimo full adder:

$$P_0 \cdot P_1 \cdot P_2 \cdot ... \cdot P_n = 1$$

Un carry-bypass si collega con un multiplexer avente per ingressi l'ultimo carry e quello iniziale e per selezionatore un "controllore di bypass" costruito con un AND dei propagate per scegliere quale ingresso considerare.

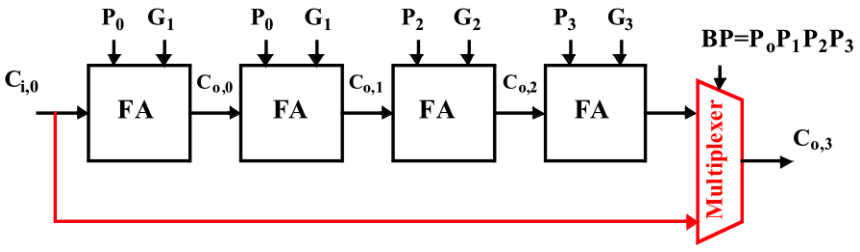


Figura 5.13: Carry-Bypass Adder

Nei sommatori a più bit tale struttura è ripetuta per effettuare un "controllo" ogni tot di bit, questo riduce il tempo medio di elaborazione rispetto al rippl-carry adder per un numero di bit molto grande:

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 73 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 75 di 238

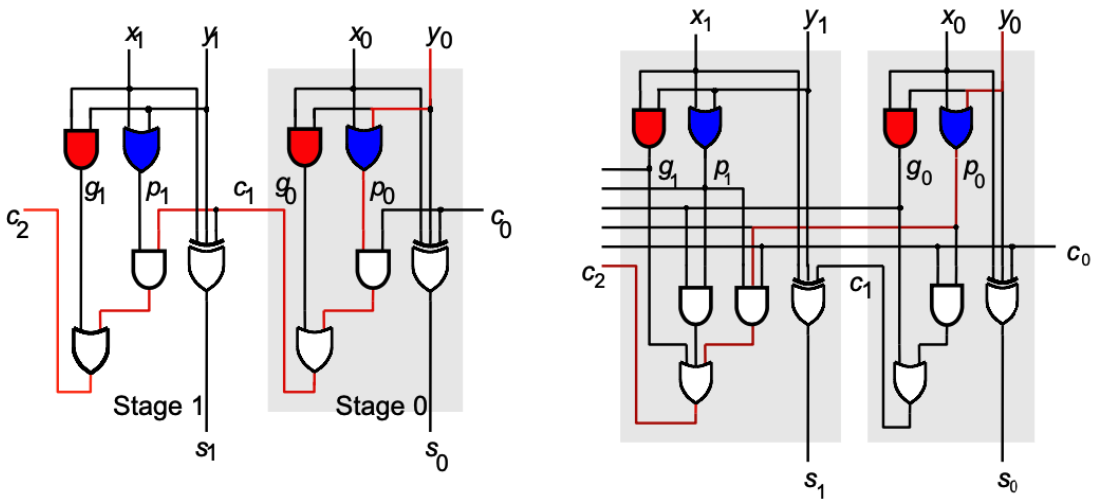


Figura 5.15: Confronto dei percorsi critici dei ripple-carry adder e carry-lookahead adder

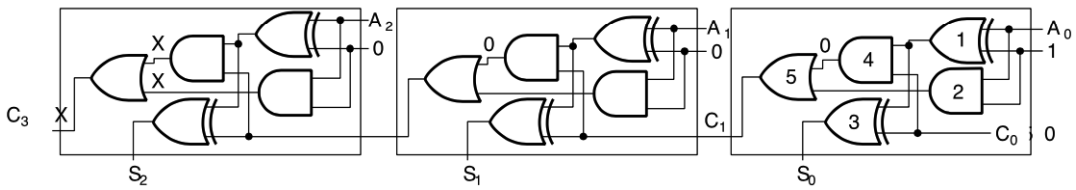
5.7 Altri circuiti aritmetici

5.7.1 Incrementer/decrementer

Gli incrementer/decrementer sono circuiti che aggiungono/sottraggono ad un segnale un valore costante, dunque una sequenza di bit nota. Possiamo ricavarli a partire dal ripple-carry adder con la tecnica design by contraction, un metodo per disegnare circuiti che implementino determinate funzioni aritmetiche a partire da circuiti più "grandi" che le contengano.

Esempio: Si disegni attraverso la tecnica design by contraction l'incrementer che, ad un numero di 3 bit, sommi 1.
++PROBABILE DOMANDA D' ESAME++

Consideriamo il ripple-carry adder e semplifichiamolo sapendo che uno dei due segnali in ingresso è costante e vale 001:



Il circuito si riduce ad avere solo quattro gates:

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 75 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 77 di 238

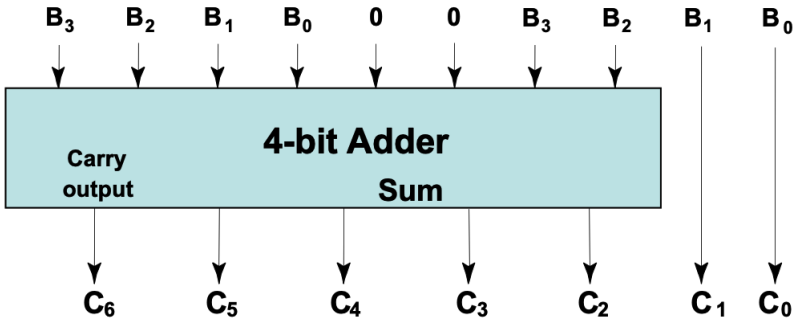


Figura 5.19: Moltiplicatore per 5

5.7.4 Zero Fill

Lo zero fill (riempimento dei zeri) è una procedura attuata per spostare un numero unsigned di n bit su m bit, con $m > n$. Consiste nell'anteporre al numero tanti 0 quanti sono i bit che servono per arrivare a m.
Esempio:

$$(1001001)_2 \rightarrow (00000001001001)_2$$

5.7.5 Extention

L' extention (estensione) è una procedura attuata per spostare un numero signed di n bit su m bit, con $m > n$. Consiste nell'anteporre al numero il bit di segno tante volte quanti sono i bit che servono per arrivare a m.
Esempio:

$$(1001001)_2 \rightarrow (11111111001001)_2$$

$$(0101010)_2 \rightarrow (00000000101010)_2$$

5.7.6 Moltiplicatore binario

Il moltiplicatore binario è un circuito aritmetico che dati due segnali unsigned in input, fornisce come output il risultato della moltiplicazione binaria. Come si realizza? Partiamo dal richiamare brevemente il metodo più comune per calcolare la moltiplicazione tra due numeri binari, questa difatti funzioe come una moltiplicaziione decimale: si moltiplica il primo numero per il primo bit del secondo, poi per il secondo bit "shiftandolo" di una posizione e così via. Il risultato è dato dalla somma di questi prodotti. Esempio:

				1	0	1	0	1	0
x						1	0	1	1
				1	0	1	0	1	0
			1	0	1	0	1	0	
		0	0	0	0	0	0		
+	1	0	1	0	1	0			
	1	1	1	0	0	1	1	1	0

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 77 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 79 di 238

esso manipola, in un moltiplicatore il ritardo aumenta quasi proporzionalmente a N^2.

5.8 Rappresentazione dei numeri decimali

Fin ora abbiamo trattato solamente numeri interi, ma è possibile rappresentare in codice binario numeri decimali? Sì, esistono prevalentemente due tipi di rappresentazione:

- **fixed-point** (virgola fissa), consiste nel fissare su n bit la posizione della virgola in modo che i bit alla sua sinistra rappresentino la parte intera e quelli alla sua destra la parte decimale. Tutti i circuiti aritmetici analizzati continuano a valere per i numeri fixed-point a patto che la posizione della virgola sia la stessa per tutti i segnali in gioco.

1	0	0	1	1
2^2	2^1	2^0	2^{-1}	2^{-2}

  Lo svantaggio della virgola fissa è che consente una rappresentazione limitata dei decimali.

  In VHDL il pacchetto:

```
USE ieee.fixed_pkg.all;
```

  tratta i numeri fixed point permettendo di definire la poiszione della virgola

- **floating-point** (virgola mobile), consiste nel dedicare a n bit un bit per il segno, una parte per la rappresentazione dell'esponente e una per la rappresentazione della mantissa (analoga alla notazione scientifica).

+/-	E	M

  In base a quanti bit sono dedicati alla rappresentazione del numero distinguiamo:

  - singola precisione, 32 bit totali(8 bit E, 23 bit M):
    $Valore = 1.M \times 2^{E-127}$
  - doppia precisione, 64 bit totali (11 bit E, 52 bit M):
    $Valore = 1.M \times 2^{E-1023}$

  La rappresentazione virgola mobile richiede un hardware molto complicato, tutti i circuiti analizzati fin ora non valgono più.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 79 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 81 di 238

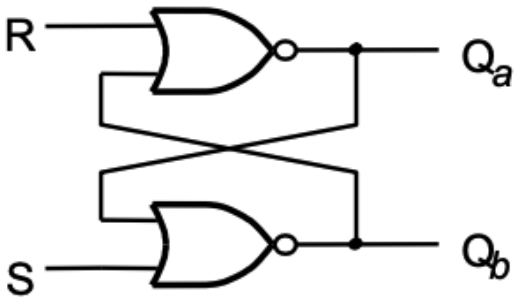


Figura 6.2: Latch SR

Caratterizzato dalla seguente tabella di verità:

Tabella 6.1: Tabella di verità del Latch SR

	S	R	$Q_a(t+1)$	$Q_b(t+1)$
hold	0	0	$Q_a(t)$	$Q_b(t)$
reset	0	1	0	1
set	1	0	1	0
force-0 (stadio proibito)	1	1	0	0

Come si vede dalla tabella il Latch SR è un elemento circuitale in grado non solo di memorizzare segnali, ma anche di intervenirvi per cambiarli. Il controllo e il settaggio sono realizzati dai due segnali S e R (set e reset): quando S e R valgono entrambi '0' i segnali Q_a e Q_b mantengono il valore di memoria, quando R vale '1' resetta Q_a, quando S vale '1' setta Q_a a '1'. Se entrambi R e S valgono '1', le uscite vengono forzate ad essere uguali ed in particolare '0'. Questo stadio è pericoloso per il circuito (stadio proibito): come si vede dalla Figura 6.3 quando si passa direttamente dallo stadio "force-0" allo stadio "hold" le uscite dovrebbero mantenere il risultato in memoria, ma non essendo più forzate allo stesso valore iniziano a cambiare alternativamente in loop.

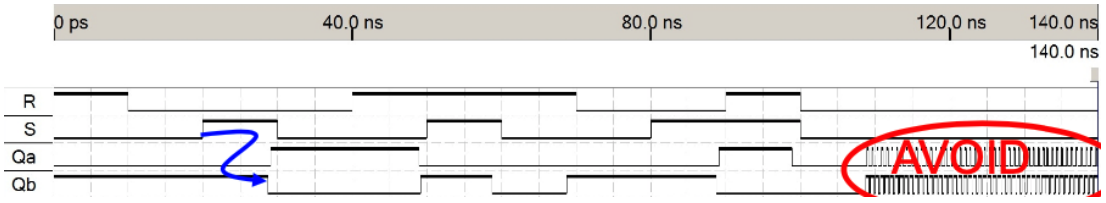


Figura 6.3: Diagramma di tempo Latch SR

Per completezza è di seguito riportato il codice VHDL del latch SR:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```


© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 81 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 83 di 238

N.B.: Il gated SR latch può essere realizzato anche solo con porte NAND, in quel caso la tabella di verità è la stessa ma con ingressi S e R negati.

Quando il segnale di sincronismo Clk vale '0' il segnale Q mantiene il valore in memoria a prescindere dai segnali S e R, quando Clk vale '1' il latch diventa trasparente ai segnali di "set"/"reset", quindi sensibile ai loro cambiamenti: abbiamo ottenuto un dispositivo di memoria sincrono.

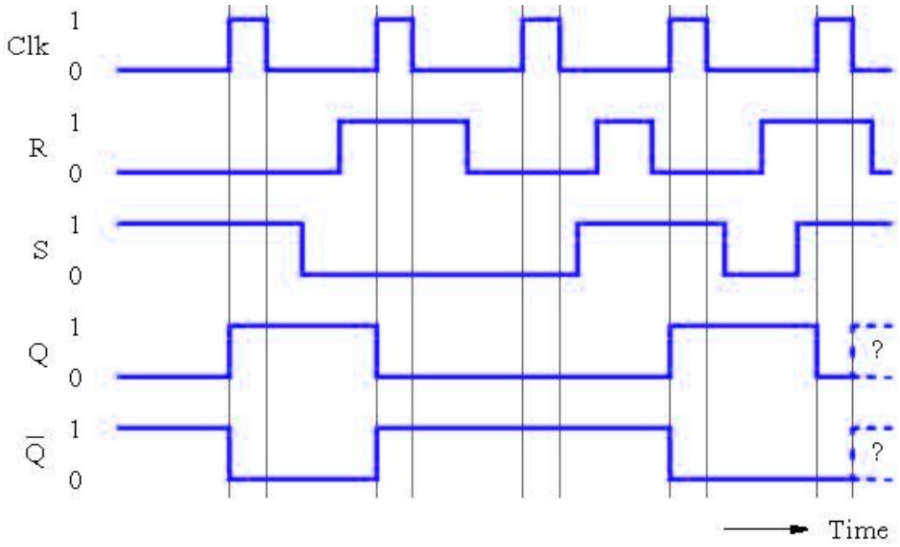


Figura 6.5: Diagramma di tempo Gated SR Latch

6.3 Gated D Latch

Il gated SR latch può essere semplificato notando che nella Tabella 6.5 lo stadio S='0' e R='0' è superfluo dato che anche lo stadio Clk='0' svolge il ruolo di memoria. Eliminando lo stadio proibito (in cui non vogliamo che il dispositivo entri) quando il segnale di "set" vale '1' quello di reset vale '0' e viceversa. Possiamo quindi pensare di collegare "set" e "reset" con un inverter, in questo modo otteniamo due vantaggi:

- un unico segnale D di settaggio che l'uscita Q "copia" quando Clk='1' (level-sensitive)
- impossibilità logica del latch di entrare nello stadio proibito

Clk	D	Q(t+1)
0	-	Q(t)
1	0	0
1	1	1

Tabella 6.5: Tabella della verità del Gated D Latch

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 83 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 85 di 238

Lo **statement process** è il costrutto vhdl che si usa per descrivere un circuito sequenziale. Esso si attiva, cioè realizza le istruzioni che contiene, solo quando uno dei segnali presenti nella sensitivity list viene "triggerato".

6.4 Edge Triggered Flip Flop

Abbiamo definito nella sezione precedente il gated D latch come un latch level-sensitive che copia cioè il dato del segnale di settaggio quando il segnale di Clk vale '1'. Vediamo adesso come realizzare un dispositivo di memoria sensibile ai fronti di un segnale, che copi cioè il segnale di settaggio nell'istante in cui il segnale di sincronismo transiti da un valore basso ad uno alto o viceversa. La configurazione di seguito illustrata è detta Master-Slave e realizza un dispositivo di memoria edge triggered, ovvero un flip flop.

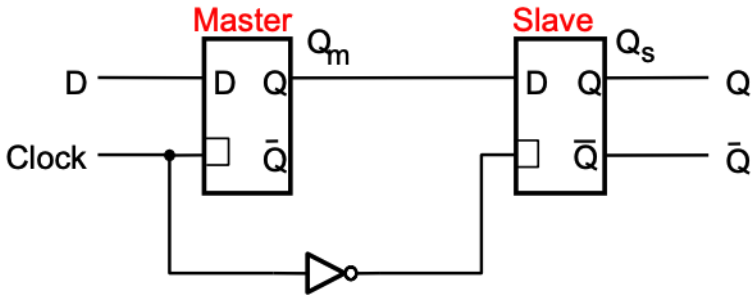


Figura 6.8: Mater-Slave Flip Flop

Il dispositivo "slave" è trasparente quando il "master" non lo è, perché il segnale di Clk è invertito. Ciò rende l'uscita Q sensibile (copia il valore di D) ai fronti di discesa (alto-basso) del segnale di Clock.

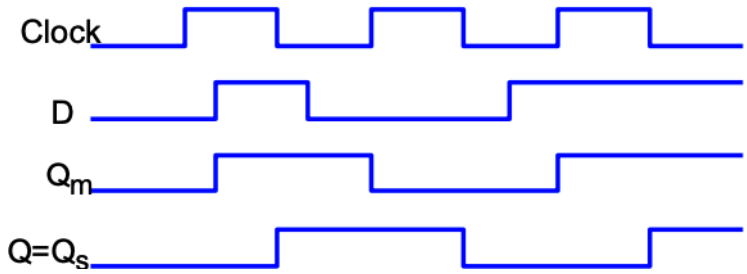


Figura 6.9: Diagramma di tempo Flip-Flop di tipo D

Il simbolo circuitale che corrisponde ad un Flip-Flop di tipo D negative edge triggered è il seguente (per i positive edge triggered è sufficiente invertire l'ingresso di Clock):

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 85 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 87 di 238

6.4.1 Preset/Clear asincorni e sincroni

Definiti i Flip Flop come elementi di memoria sincroni e sensibili ai fronti del Clock, potremmo voler intervenire sul valore d'uscita dall'esterno settandola direttamente a '1' o a '0' a prescindere dal segnale D. Introduciamo dunque due segnali di "preset" e "clear", possiamo farlo in modo sincrono o asincrono:

- Preset/Clear asincroni, non hanno bisogno del fronte di clock per attivarsi. Quando "preset" vale '0' setta l'uscita a '1', quando "clear" vale '0' setta l'uscita a '0'.

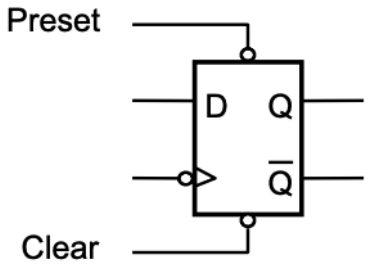


Figura 6.11: Flip Flop di tipo D con preset/clear asincroni

- Preset/Clear sincrono, settano il circuito solo nell'istante del fronte di Clock. Nell'esempio seguente è mostrato un Flip Flop di tipo D con Clear sincrono, può essere visto come un segnale che invece di settare l'uscita direttamente intervenendo nell'elemento di memoria, setta il segnale D che invece è "costretto" ad aspettare un segnale di Clock per essere copiato.

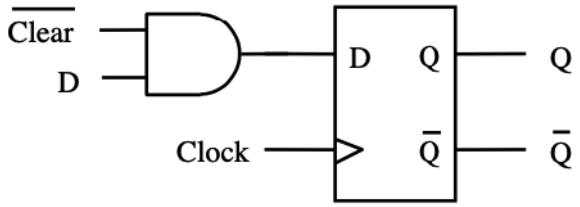


Figura 6.12: Flip Flop di tipo D con clear sincrone

N.B.: Nei due esempi precedenti sono stati definiti in modo diverso i segnali di "preset" e "clear": nel primo esempio sono stati fatti entrare invertiti nel flip flop, nel secondo sono indicati come segnati negati, sono due modi diversi per indicare la stessa cosa, cioè che sono segnali i cui effetti si verificano quando il loro valore è '0'.

Quali sono i vantaggi e gli svantaggi di queste metodologie di preset/clear?

Problematica	Preset/Clear sincrono	Preset/Clear asincrono
Area	Più piccola	Più grande, ma problema trascurabile

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 87 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 89 di 238

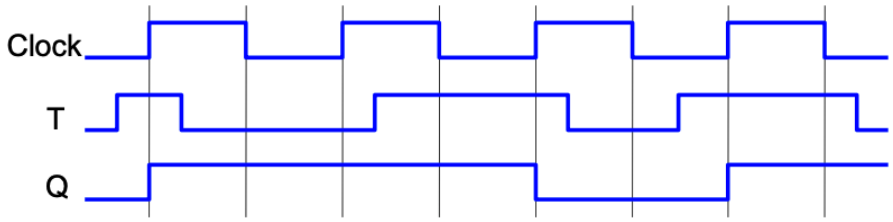


Figura 6.14: Diagramma di tempo Flip Flop di tipo T

Il simbolo circuitale che lo rappresenta è il seguente:

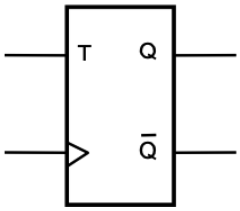


Figura 6.15: Simbolo circuitale del Flip Flop di tipo T

6.6 Flip Flop di tipo JK

Il JK Flip Flop è un tipo di Flip Flop relativamente poco utilizzato nelle applicazioni digitali, per questo li accenneremo soltanto.

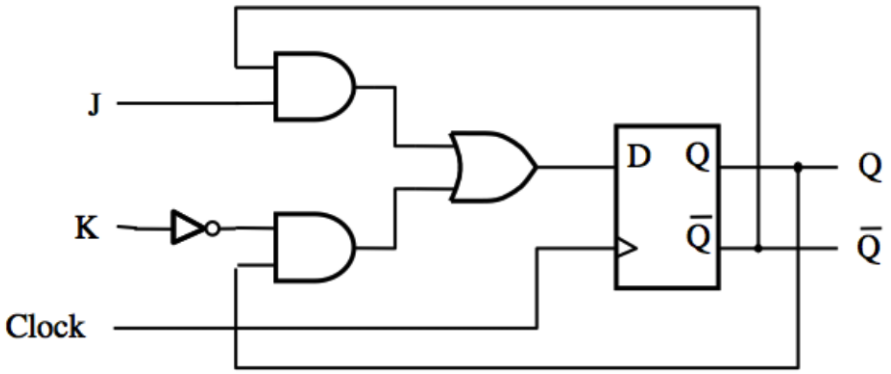


Figura 6.16: Flip Flop di tipo JK

Il Flip Flop di tipo JK è caratterizzato dalla seguente tabella di verità:

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

Tabella 6.11: Tabella di verità del Flip Flop di tipo JK

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 89 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 91 di 238

Capitolo 7

Macchine a stati

Ora che abbiamo esaminato sia i circuiti combinatori che quelli sequenziali cerchiamo di capire come realizzare a livello hardware un algoritmo completo, comprensivo quindi di parti combinatorie e parti sequenziali.

Un generico hardware di elaborazione (o processing element) è formato da due componenti:

- L'unità di controllo, ha il compito di coordinare tutte le azioni necessarie per l'esecuzione di una istruzione e di insiemi di istruzioni.
- Il datapath, è un insieme di unità di calcolo, come ad esempio le unità di elaborazione (ALU), i registri e i moltiplicatori necessari per l'esecuzione delle istruzioni

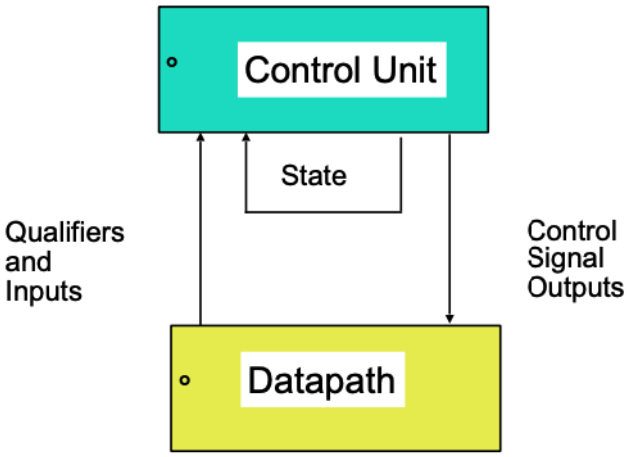


Figura 7.1: Configurazione di un processing element

L'unita di controllo può essere vista come una macchina a stati, ovvero un modello utilizzato per rappresentare e controllare un flusso di esecuzione.

Una macchina a stati finiti (quelle su cui ci concentreremo in questo corso), o FSM in breve, è un modello computazionale basato su una macchina ipotetica

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 91 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 93 di 238

- **Macchina di Mealy**: gli output dipendono sia dagli stati della macchina, sia dagli input.

Analizziamo le differenze affrontando un esempio di design.

7.1 Design di una macchina a stati

7.1.1 Design con macchina di Moore

Come si disegna una macchina a stati a partire dal flusso di esecuzione che deve descrivere? Dividiamo il processo di design in steps aiutandoci con un esempio, prima proviamo a risolvere il problema attraverso la macchina di Moore, quindi, alla fine, accenneremo a come risolverlo attraverso la macchina di macchina di Mealy.

- **Step 1: definire le specifiche della macchina.**

  Per prima cosa si devono definire le specifiche della macchina, ovvero quale deve essere il suo comportamento.

  Esempio: si realizzi una macchina a stati FSM che ha, oltre al clock, un segnale w di ingresso campionato sui fronti di salita, e che quando w vale '1' due volte consecutivamente forzi un segnale di output z, che altrimenti vale '0', a '1'.

Ciclo di Clock	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
w	0	1	0	1	1	1	0	1
z	0	0	0	0	0	1	1	0

Tabella 7.1: Tabella del flusso di esecuzione del problema tramite macchina di Moore

- **Step 2: disegnare il diagramma di stati**

  Il diagramma di stati (o pallogramma) è un diagramma che definisce gli stati (le ellissi) e le transizioni tra essi (le frecce). A ciascuno stato è assegnato un output: più stati possono avere gli stessi valori di output ma ciascuno è diverso dagli altri per come viene elaborato (ovvero per le frecce che lo connettono agli altri stati). Il diagramma è la parte più complessa del processo di design.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 93 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 95 di 238

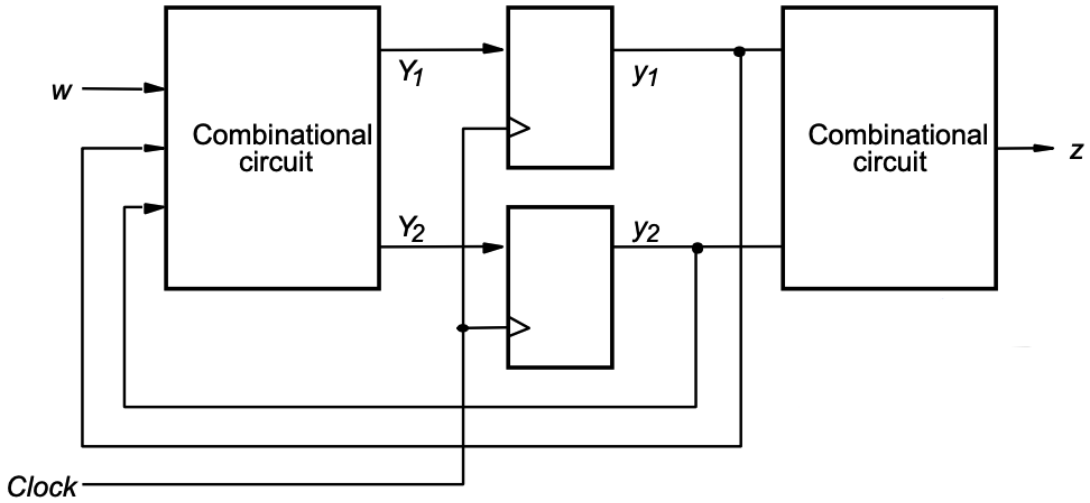
	Present state y_2y_1	Next state w=0 Y_2Y_1	Next state w=1 Y_2Y_1	Output z
A	00	0 0	0 1	0
B	01	0 0	1 0	0
C	10	0 0	1 0	1
	11	-	-	-

Tabella 7.5: Tabella degli stati-assegnati

N.B.: L'assegnazione migliore prevede che allo stadio di reset sia assegnato la combinazione nulla di variabili, in generale esistono più assegnazioni possibili.

- **Step 5: derivare le funzioni di transizione**

  Siamo arrivati ad aver descritto gli stati presenti tramite due variabili booleane y_1y_2, quelli futuri tramite altrettante variabili Y_1Y_2.



A questo punto dobbiamo derivare le funzioni booleane che definiscono la transizione tramite il valore di ingresso w (oltre che il circuito combinatorio che restituisce l'output). Conosciamo già la Tabella 7.5, quindi basta usare le mappe di Karnaugh:

y_1 \ y_2	0	1
0	0	0
1	1	1

$$z = y_2$$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 95 di 238

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 97 di 238

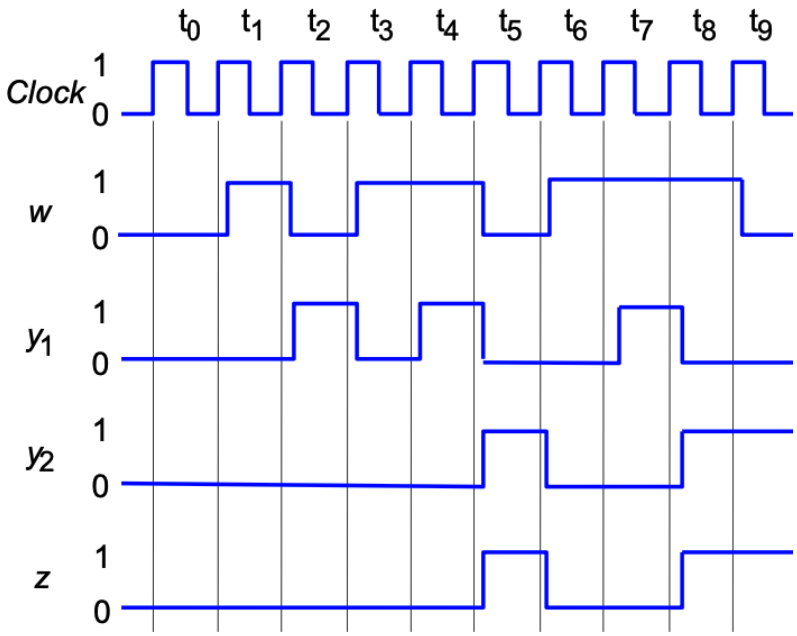


Figura 7.6: Diagramma di tempo macchina di Moore

Come si può vedere dal diagramma di tempo, la macchina di Moore, deve prima campionare il secondo w=1 per transire di stato e quindi restituire in output z=1.
Questo non accade con la macchina di Mealy che invece è più reattiva: come abbiamo già accennato l'output dipende sia dallo stato corrente sia dall'input , quindi z=1 si ha "istantaneamente" appena la macchina riceve (non campiona, sta in questo la differenza) il secondo ingresso.

Ciclo di Clock	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
w	0	1	0	1	1	1	0	1
z	0	0	0	0	0	1	1	0

Tabella 7.8: Tabella del flusso di esecuzione del problema tramite macchina di Moore

Ciclo di Clock	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
w	0	1	0	1	1	1	0	1
z	0	0	0	0	1	1	0	0

Tabella 7.9: Tabella del flusso di esecuzione del problema tramite macchina di Mealy

7.1.2 Design con macchina di Mealy

Il procedimento per disegnare una macchina di Mealy è del tutto identico al processo di design della macchina di Moore, eccezion fatta, ovviamente, per le

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 97 di 238