Progettazione di sistemi digitali

Leonardo Ganzaroli

Indice

Introduzione

Questi appunti sono derivanti principalmente dalle slide del corso di *Progettazione di Sistemi Digitali* che ho seguito durante la laurea Triennale di informatica all'università "La Sapienza".

1 Basi numeriche

I numeri più comunemente utilizzati sono basati sul sistema posizionale, ossia ogni cifra di un numero assume un valore diverso in base alla sua posizione.

Questo resterà lo standard anche per le altre basi che saranno viste a seguire, l'unica differenza sarà la quantità di simboli usati per rappresentare i numeri.

1.1 Base Decimale

La base che viene usata tutti i giorni, usa $0 \dots 9$.

$$5374_{10} = 5 * 10^3 + 3 * 10^2 + 7 * 10^1 + 4 * 10^0$$

1.2 Base Binaria

Il sistema utlizzato da qualsiasi dispositivo digitale, usa solamente 0 e 1.

$$1101_2 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$$

Una singola cifra viene definita bit, in un numero binario si individuano 2 bit importanti:

1 1 0 0 1 | Più significativo
Meno significativo

1.3 Base Esadecimale

Un'altra base che verrà incontrata molto spesso, usa $0 \dots 9$ e le lettere da A ad F, ha quindi un totale di 16 simboli per rappresentare i numeri.

Decimale	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Esadecimale	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Tabella 1: Corrispondenza tra basi

1.4 Conversione tra basi

Dato che l'unica differenza tra le basi sta in quanti simboli usano risulta semplice convertire un numero da una base all'altra.

1.4.1 Decimale → Binario

Esistono 2 metodi:

1.
   - Trovo la potenza di 2 più grande che sia minore del numero
   - La sottraggo al numero
   - Ripeto i punti precedenti usando la differenza della sottrazione fino a raggiungere 0

2.
   - Divido il numero per 2
   - Segno il resto della divisione
   - Ripeto i punti precedenti usando il quoziente della divisione fino a raggiungere 0

Alcune potenze del 2 per comodità:

Potenza	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9
Numero	1	2	4	8	16	32	64	128	256	512

Tabella 2: Potenze del 2

Per meglio capire i metodi converto il numero 23.

Metodo 1:

- $23 - 2^4 = 7$
- $7 - 2^2 = 3$
- $3 - 2^1 = 1$
- $1 - 2^0 = 0$

Se considero adesso un numero binario come una tabella posso comporlo mettendo 1 nelle colonne delle potenze usate + 1 (da dx a sx).

pos. 5	pos. 4	pos. 3	pos. 2	pos. 1
1	0	1	1	1
2^4	2^3	2^2	2^1	2^0

Quindi $23_{10} = 10111_2$.

Metodo 2:

Dividendo	Quoziente	Resto
23	11	1
11	5	1
5	2	1
2	1	0
1	0	1

Basta adesso leggere dal basso verso l'alto per avere il numero.

1.4.2 Esadecimale → Binario

Per convertire basta associare ad ogni simbolo esadecimale il rispettivo valore binario:

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Tabella 3: Associazione

Per convertire 4AF16 prendo l'equivalente in binario e sostituisco:

$$4 = 0100,\ A = 1010,\ F = 1111,\ 1 = 0001,\ 6 = 0110$$

Concatenando i risultati ottengo $4AF16_{16} = 01001010111100010110_2$

2 Numeri binari

Andiamo adesso ad approfondire i numeri binari che saranno la base fondamentale utilizzata da qui in poi.

2.1 Unità di misura

Alcune unità utili sono:

- Bit = un singolo 0/1
- Byte = 8 bit
- Nibble = 4 bit = Mezzo byte
- Kilobyte = 1024 byte
- Megabyte = 1024 kilobyte
- ...

2.2 Operazioni aritmetiche

Le operazioni hanno il medesimo funzionamento della loro controparte decimale.

2.2.1 Somma

+	0	1
0	0	1
1	1	10

Tabella 4: Tabella delle somme

Per esempio $91 + 109 = 200$:

$$\begin{array}{r} {\scriptstyle 1\,1\,1\,1\,1\,1} \\ 1011011 \\ +1101101 \\ \hline 11001000 \end{array}$$

2.2.2 Sottrazione

La sottrazione si può svolgere con il metodo classico oppure come vedremo tra poco si può ricondurre ad una somma con un numero negativo.

2.2.3 Moltiplicazione

×	0	1
0	0	0
1	0	1

Tabella 5: Tabella moltiplicativa

Esempio $91 \times 5 = 455$:

$$\begin{array}{r} 1011011 \\ * \quad 101 \\ \hline + \quad 1011011 \\ + \ 0000000 \\ +1011011 \\ \hline 111000111 \end{array}$$

2.3 Overflow

L'overflow è un problema che si presenta quando un'operazione restituisce un risultato con più bit di quanti se ne possono rappresentare, in questo caso i bit "in più" verranno scartati e questo può portare a dei risultati incorretti.

2.4 Numeri negativi

Per rappresentare in binario i numeri negativi ho 2 alternative:

- **Sign/Magnitude**

  Uso il bit più significativo per indicare il segno del numero.

  Questo metodo ha 3 problemi:

  1. Con x bit posso rappresentare solo 2^{x-1} valori invece di 2^x
  2. Ho 2 rappresentazioni dello $0 \rightarrow 00, 10$
  3. L'addizione non funziona

- **Complemento a 2**

  Il bit più significativo assume il suo effettivo valore, ma se è uguale ad 1 quel valore sarà considerato negativo.

  Questo metodo risolve i problemi dell'altro, serve però una piccola "trasformazione" per portare un numero in complemento a 2.

2.4.1 Complemento a 2

Per ottenere un numero in complemento a 2 ci sono 2 passaggi:

1. Inverto tutti i bit (Complemento ad 1)
2. Sommo 1 al numero ottenuto

Per rappresentare -25:

- $25 = 011001 \rightarrow 100110$
- $100110 + 1 = 100111$

Ottengo $100111 = -1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = -25$

2.5 Numeri razionali

Anche in questo caso ci sono 2 metodi di rappresentazione.

2.5.1 Virgola fissa

Si scelgono due quantità di bit che andranno a rappresentare rispettivamente la parte intera e quella decimale, il "punto" viene inserito implicitamente tra le due parti.

Se per esempio si decide di usare 4 bit per entrambe le parti:

0	1	1	0	1	1	0	0
2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

Tabella 6: 6.75

2.5.2 Virgola mobile

Questo sistema è simile alla notazione scientifica, i numeri vengono trattati quindi nella forma:

$$\pm \text{ Mantissa} \times \text{Base}^{\text{Esponente}}$$

Solitamente si lavora con stringhe di bit di lunghezza prefissata, le 2 più comuni sono:

Precisione	Lunghezza	Segno	Esponente	Mantissa
Singola	4 byte	1 bit	1 byte	23 bit
Doppia	5 byte	1 bit	11 bit	52 bit

Per convertire un numero in questo formato si seguono questi passaggi:

1. Trasformare il numero in virgola fissa **senza segno**
2. Riscriverlo in notazione scientifica binaria
3. Riempire i campi
    - Ignorare il primo 1 nella mantissa
    - L'esponente va sommato a 127 (Standard IEEE 754, se Double 1023)

Per esempio (-11.625):

1. In virgola fissa senza segno = 1011.101
2. In notazione = 1.011101×2^3
3. Riempo i campi
    - Segno = 1
    - Esponente = $127 + 3 = 130 = 10000010$
    - Mantissa = 011101

1	10000010	011101000...

Sono presenti alcuni valori speciali:

Numero	Segno	Esponente	Mantissa
0	X	00000000	0000...
∞	0	11111111	0000...
$-\infty$	1	11111111	0000...
NaN	X	11111111	Non-zero

Tabella 7: Valori speciali

Somma Per sommare 2 numeri:

1. Portarli in notazione
2. Portarli allo stesso esponente
3. Sommare le mantisse
4. Trasformare il risultato

Esempio:

N_1	0	10000000	0100...	2.5
N_2	0	10000010	10100100...	13.125

Trasformo in notazione e porto allo stesso esponente:

$$N_1 = 1.01 * 2^1$$

$$N_2 = 1.101001 * 2^3 = 110.1001 * 2^1$$

Sommo:

$$\begin{array}{r} 1.01 \\ +110.1001 \\ \hline 111.1101 \end{array}$$

Ritrasformo:

$$111.1101 * 2^1 = 1.111101 * 2^3$$

0	10000010	111101...	15.625

3 Logica booleana

Lavorando con valori binari è essenziale fare una parentesi sulla logica booleana, perché i circuiti logici lavorano con valori binari e il loro funzionamento è espresso tramite equazioni booleane.

L'algebra di Boole si discosta da quella elementare perché:

- Le variabili possono assumere solo 2 valori (di verità):
  - Vero
  - Falso
- Le operazioni usate sono quelle logiche:
  - AND
  - OR
  - NOT
  - ...

3.1 Assiomi

Alcune formule fondamentali:

Nome	Teorema	Duale
Identità	$B * 1 = B$	$B + 0 = B$
Elemento nullo	$B * 0 = 0$	$B + 1 = 1$
Idempotenza	$B * B = B$	$B + B = B$
Involuzione	$B = \bar{\bar{B}}$	
Complemento	$B * \bar{B} = 0$	$B + \bar{B} = 1$
Commutatività	$B * C = C * B$	$B + C = C + B$
Associatività	$B * (C * D) = (B * C) * D$	$B + (C + D) = (B + C) + D$
Distributività	$B * (C + D) = (B * C) + (B * D)$	$B + (C * D) = (B + C) * (B + D)$
Assorbimento	$B * (B + C) = B$	$B + (B * C) = B$
Combinazione	$(B * C) + (B * \bar{C}) = B$	$(B + C) * (B + \bar{C}) = B$
Consenso	$(B * C) + (\bar{B} * D) + (C * D) = (B * C) + (\bar{B} * D)$	$(B + C) * (\bar{B} + D) * (C + D) = (B + C) * (\bar{B} + D)$
De Morgan	$\overline{ABC\ldots} = \bar{A} + \bar{B} + \bar{C} + .\bar{.}.$	$\overline{A + B + C\ldots} = \bar{A}\bar{B}\bar{C}.\bar{.}.$

Tabella 8: Assiomi booleani

4 Circuiti logici

Un circuito logico è un insieme di dispositivi digitali interconnessi che può essere visto come una scatola nera, il suo scopo è quello di ricevere un certo numero di input binari e produrre uno o più output binari.

I valori binari in questione vengono associati fisicamente alla presenza di tensione (1) o meno (0).

Essendo trattati come scatole nere vengono utilizzati 4 elementi per descriverli:

1. **Input**
2. **Output**
3. **Specifiche temporali**
4. **Specifiche funzionali**

Si possono inoltre distinguere 2 tipi di circuiti:

1. **Combinatori**
   - No memoria
   - L'output dipende solamente dall'input attuale
2. **Sequenziali**
   - Ha memoria
   - L'output dipende dall'input attuale e da quelli passati

4.1 Circuiti combinatori

Per poter essere definito combinatorio un certo circuito deve necessariamente rispettare alcune regole:

- Ogni elemento deve essere combinatorio
- Non devono esserci cicli
- Ogni nodo è input oppure si connette ad un solo output

4.1.1 Porte logiche

Una porta logica è uno dei circuiti logici più semplici esistenti, rappresentano la base per creare dei circuiti molto più grandi.

Essendo dei circuiti a loro volta possiamo ignorare gli effettivi componenti interni e descriverne il funzionamento tramite le "tabelle di verità".

Transistor I transistor sono dei componenti creati "drogando" il silicio in modo da renderlo conduttivo, sono dei componenti essenziali presenti nella stragrande maggioranza dei dispositivi elettronici.

Hanno 2 principali modalità di funzionamento, una di queste è quella da "switch" (interruttori). Questo li rende degli ottimi candidati per la creazione di porte logiche.

Ne esistono di diversi tipi ma un tipo estremamente adatto è il **MOSFET**, lo useremo per creare fisicamente le porte.

Ne esistono di 2 tipi con funzionamento inverso l'uno dell'altro, la loro combinazione permette di creare le porte:

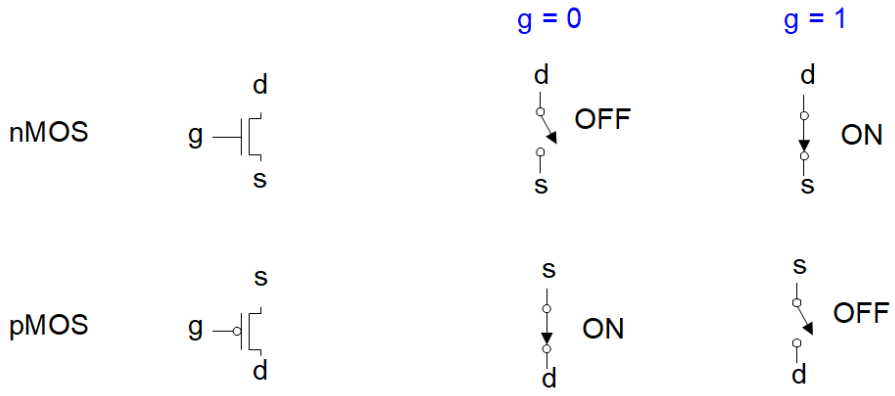


Figura 1: MOS

Lo schema generale per creare delle porte è il seguente:

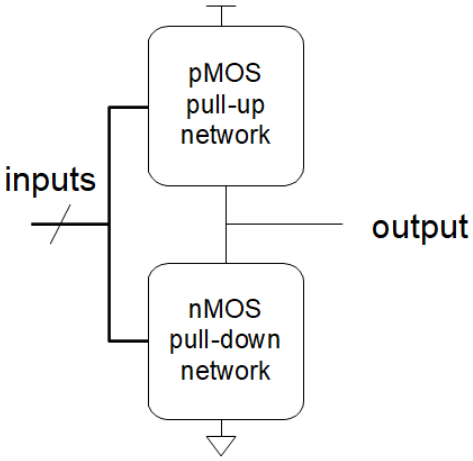
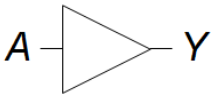


Figura 2: Schema porta logica MOS

N.B. Le porte create con questo schema sono negate.

Lista delle porte principali Le seguenti sono le porte più semplici, con queste si possono creare circuiti più complessi.

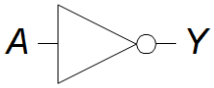
- **Buffer**:



A	Y
0	0
1	1

Figura 3: Buffer

- **NOT**:



A	Y
0	1
1	0

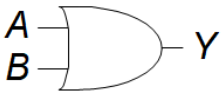
Figura 4: NOT

- **AND**:

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

Figura 5: AND

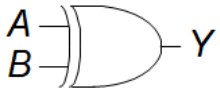
- **OR**:



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

Figura 6: OR

- **XOR**:

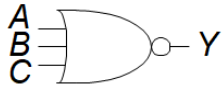


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Figura 7: XOR

N.B. Di queste porte esistono anche le versioni con più input.

Come visto sopra si possono combinare più porte tra loro, per esempio combinando una NOT ed una OR a 3 input:



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

Figura 8: NOR3

N.B Le porte NAND e NOR sono dette "funzionalmente complete", questo perché possono essere usate per creare tutte le altre porte logiche.

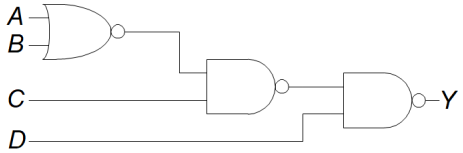
4.1.2 Bubble pushing

Il bubble pushing è una tecnica che sfrutta il teorema di De Morgan, permette di modificare direttamente il circuito senza passare per le equazioni.

Funzionamento:

1. Si parte dall'output
2. Si spingono le "bolle" verso l'input trasformando la porta
3. Si tolgono le "bolle" dove possibile

Partendo da questo schema per esempio:



Svolgo i seguenti passaggi:

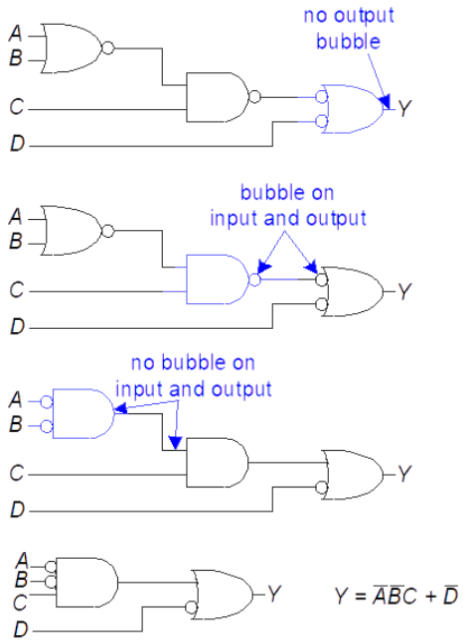


Figura 9: Procedimento bubble push

4.1.3 Implementazione con MOS

Come detto prima lo schema usato crea delle porte negate, risulta quindi importante la porta NOT:

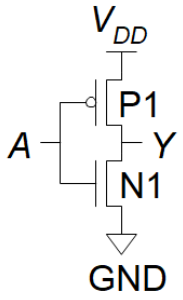


Figura 10: NOT MOS

Altre 2 porte sono:

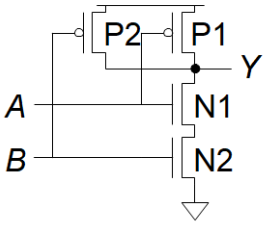


Figura 11: NAND MOS

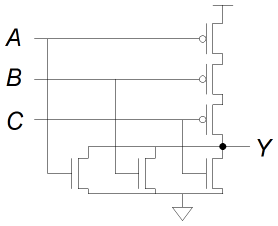


Figura 12: NOR3 MOS

In breve:

- AND = serie sotto, parallelo sopra
- OR = parallelo sotto, serie sopra

4.1.4 Livelli logici

L'uso di livelli di tensione come input può portare dei problemi dovuti a:

- Rumore termico
- Interferenze elettromagnetiche

In caso i valori vengano sballati in modo importante da una di quest'ultime possono esserci dei problemi di falsa lettura, per ovviare al problema vengono stabiliti dei margini per cercare di contrastare il problema e si stabilisce una zona "proibita" tra i 2.

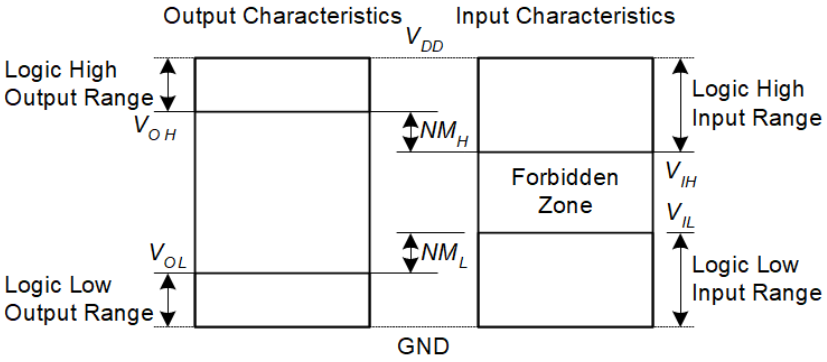


Figura 13: Creazione margini

4.1.5 Ritardi

Un ulteriore problema è quello dei ritardi che si sviluppano nei circuiti, questi sono dovuti a fenomeni fisici come la generazione del calore da parte dei componenti o l'uso di componenti con diverse caratteristiche.

Anche questo fenomeno può portare a false letture (se un singolo cambiamento in input ne comporta diversi in output si parla di *Glitch*), i 2 ritardi più importanti sono:

- Ritardo di propagazione
- Ritardo di contaminazione

Sono rispettivamente il tempo massimo e minimo trascorso dal cambio di input a quello di output, di conseguenza si individuano in un circuito due percorsi:

- Critico (o lungo)
- Breve

4.1.6 Equazioni booleane

Il modo in cui si può esprimere l'output di un crcuito partendo dall'input.

Definizioni

- **Letterale** = Variabile o suo complemento
- **Complemento** = Letterale con barra superiore ($\bar{A}$)
- **Prodotto** = AND = * = Letterali senza spazio tra loro (ABD)
- **Somma** = OR = +
- **Implicante** = Prodotto di letterali
- **Mintermine** = Prodotto che include tutti gli input
- **Maxtermine** = Somma che include tutti gli input
- **Analisi** = Circuito → Equazione
- **Sintesi** = Equazione → Circuito

SOP/POS 2 forme in cui può essere scritta ogni equazione:

- **SOP** (Somma di prodotti)
  - Ogni riga ha un mintermine
  - Prendo le righe con output 1
  - Complemento se input è 0
- **POS** (Prodotto di somme)
  - Ogni riga ha un maxtermine
  - Prendo le righe con output 0
  - Complemento se input è 1

Esempio:

A	B	Y	Maxt	Mint
0	0	0	$A+B$	$\bar{A}\bar{B}$
0	1	1	$A+\bar{B}$	$\bar{A}B$
1	0	0	$\bar{A}+B$	$A\bar{B}$
1	1	1	$\bar{A}+\bar{B}$	AB

$$SOP = \bar{A}B + AB$$
$$POS = (A+B)(\bar{A}+B)$$

Semplificazione Applicando gli assiomi booleani è possibile semplificare equazioni complesse:

$$\begin{aligned} Y = & AB + BC + \bar{B}\bar{D} + A\bar{C}\bar{D} \\ & AB + BC + \bar{B}\bar{D} + A\bar{C}\bar{D} + A\bar{D} \text{ (Consenso)} \\ & AB + BC + \bar{B}\bar{D} + (A\bar{C}\bar{D} + A\bar{D}) \text{ (Associatività)} \\ & AB + BC + \bar{B}\bar{D} + A\bar{D} \text{ (Assorbimento)} \\ & AB + BC + \bar{B}\bar{D} \text{ (Consenso)} \end{aligned}$$

4.1.7 Mappe di Karnaugh

Le mappe di Karnaugh sono un modo grafico per minimizzare le equazioni booleane di massimo 4 variabili, usandole si ottiene una forma SOP (o POS) minima dell'equazione.

X e Z Si tratta di 2 valori speciali che possono essere incontrati:

- **X** $\rightarrow$ Indica un valore ignoto o che viene ignorato
- **Z** $\rightarrow$ Indica un valore di alta impedenza

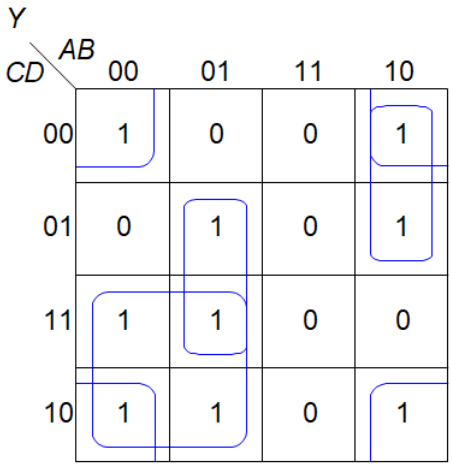
La differenza fondamentale tra i 2 è che Z può essere cambiato in presenza di un altro segnale elettrico, come nel caso del buffer tri-state.

Per usare le mappe correttamente bisogna seguire delle regole:

- Ogni 1 va cerchiato almeno una volta
- Ogni cerchio deve contenere 2^n elementi in ogni direzione
- Il cerchio deve essere il più grande possibile
- Il cerchio può estendersi oltre i bordi
- I valori X si cerchiano solo se sono utili
- Ogni casella è un mintermine

Esempio:

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



Risulta $Y = \bar{A}C + \bar{A}BD + A\bar{B}\bar{C} + \bar{B}\bar{D}$

Se volessi invece la forma POS dovrei:

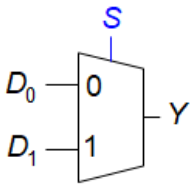
- Cerchiare gli 0
- Considerare ogni casella come maxtermine
- Considerare gli 1 come complemento

Otterrei $Y = (\bar{B} + C + D)(\bar{A} + \bar{B})(A + B + C + \bar{D})(\bar{A} + \bar{C} + \bar{D})$

4.1.8 Altri componenti importanti

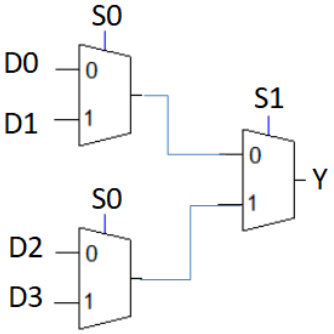
- Multiplexer

  Permette di scegliere tramite uno o più segnali quale input collegare all'output:



S	Y
0	D_0
1	D_1

Figura 14: Multiplexer 2 input



S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Figura 15: Multiplexer 4 input

- Decoder

  Componente con x input e 2^x output, può avere un solo output "attivo" alla volta.

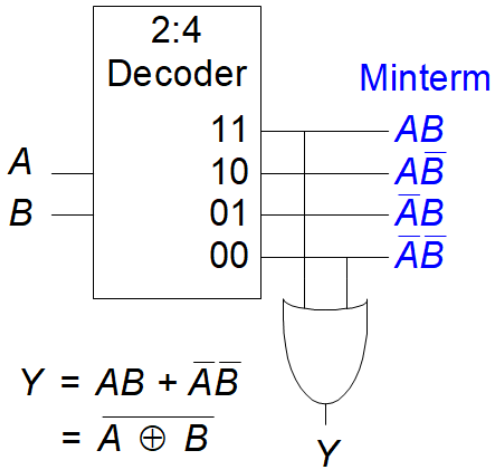


Figura 16: Decoder con esempio di logica

4.2 Circuiti sequenziali

A differenza di quelli combinatori sono presenti dei componenti che permettono di mantenere informazioni riguardo lo stato precedente del sistema.

4.2.1 Definizioni

- **Stato** = Tutte le informazioni su un circuito necessarie per spiegare il suo comportamento futuro
- **Latch/Flip-flop** = Elementi che mantengono un bit di stato
- **Circuiti sequenziali sincron**i = Logica combinatoria + banco di flip-flop

4.2.2 Elementi di stato

Gli elementi di stato hanno il compito di conservare lo stato del circuito per influenzarlo nel futuro, i componenti di questo tipo sono:

- **Circuito bistabile**

  Elemento base usato per costruirne altri, ha 2 output, no input e 2 possibili forme:

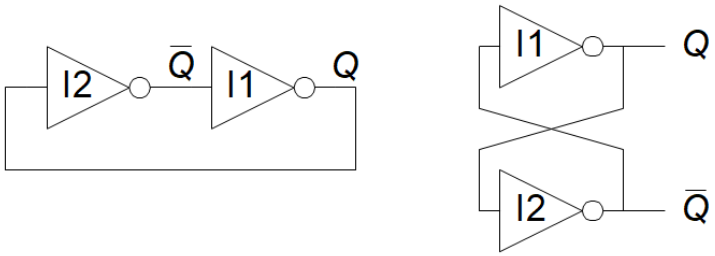
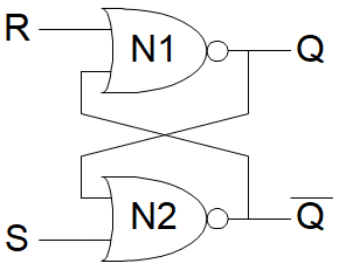


Figura 17: Circuito bistabile

  Mantiene l'output costante indipendentemente dalla forma e dal fatto che Q sia 1 o 0.

- **Latch SR**

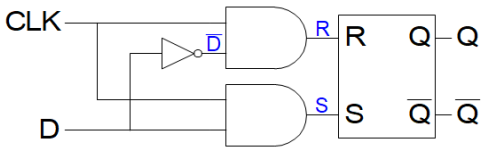


S	R	Q	$\bar{Q}$
0	0	Q_{prev}	$\bar{Q}_{prev}$
0	1	0	1
1	0	1	0
1	1	0	0

Figura 18: Latch SR

  $S = SET,\ R = RESET$, mantiene lo stato precedente se entrambi sono 0, ha uno stato illegale.

- **Latch D**



CLK	D	Q	$\bar{Q}$
0	X	Q_{prev}	$\bar{Q}_{prev}$
1	0	0	1
1	1	1	0

Figura 19: Latch D

$D = DATA$, $CLK = CLOCK$, mantiene lo stato precedente quando CLK è 0 ed evita lo stato invalido del Latch SR.

- **Flip-flop D**

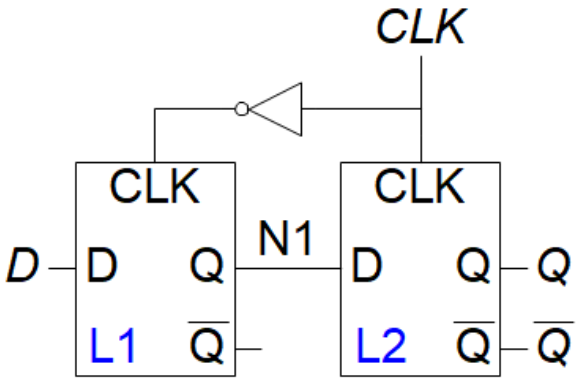


Figura 20: Flip-flop D

$Q = D$ quando il clock sale da 0 a 1, quel valore viene mantenuto fino alla prossima salita del clock.

- **Flip-flop E** (multiplexer + Flip-flop D)

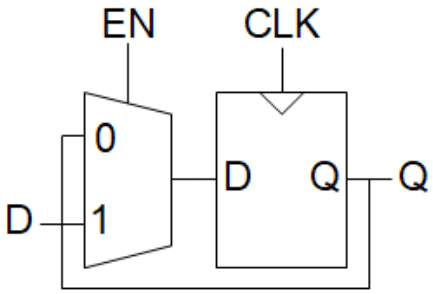


Figura 21: Flip-flop E

Se $EN = 0$ allora mantiene lo stato precedente, altrimenti funziona normalmente.

- **Flip-flop R** (AND + Flip-flop D)

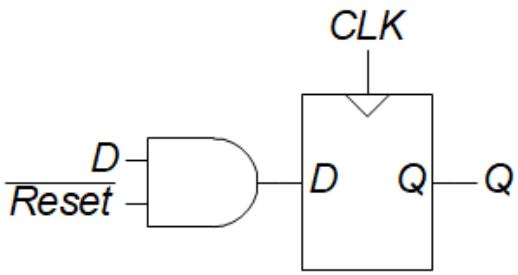


Figura 22: Flip-flop R

Se $RESET = 1$ allora $Q = 0$, altrimenti funziona normalmente, il cambio può essere sincrono (aspetta salita clock) o asincrono (istantaneo).

- **Flip-flop S**

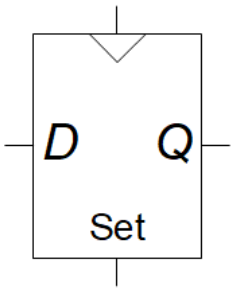


Figura 23: Flip-flop E

Se $SET = 1$ allora $Q = 1$, altrimenti funziona normalmente.

4.2.3 Circuiti sincroni

La caratteristica principale di questi circuiti è la loro sincronizzazione con un clock, quindi il loro stato cambierà ad ogni salita del clock.

Devono inoltre rispettare delle regole:

- Ogni elemento è un circuito combinatorio o un registro
- Tutti i registri usano lo stesso clock
- È presente almeno un registro
- Ogni ciclo contiene almeno un registro

I due tipi più comuni sono le FSM e le pipeline.

4.2.4 FSM

Una macchina a stati finiti è composta da 3 elementi:

1. **Next state logic** → Calcola il valore dello stato successivo
2. **Registri** → Conservano lo stato corrente
3. **Output Logic** → Si occupa degli output

Ce ne sono 2 tipi:

- **Moore** = output dipendenti solo dallo stato attuale
- **Mealy** = output dipendenti dallo stato attuale e dagli input

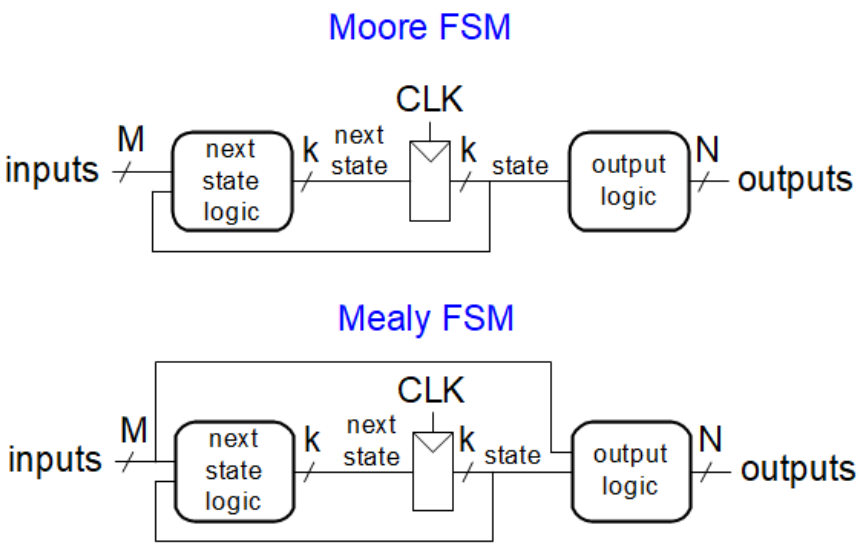
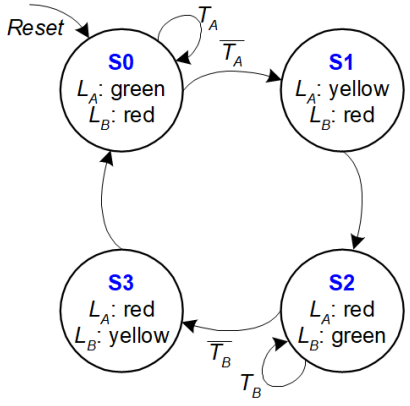


Figura 24: Schema FSM

Diagramma delle transizioni Il diagramma delle transizioni si usa per rappresentare graficamente gli stati di una FSM e le transizioni tra di essi.

- **Stati** = Cerchi
- **Transizioni** = Archi

Esempio semaforo:



Current State	Inputs		Next State
S	***TA***	***TB***	***S'***
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

Figura 25: Semaforo FSM

Lavorando con i registri bisogna associare ad ogni stato una certa stringa binaria, ci sono 2 alternative:

- Codifica semplice

Stato	Stringa	
S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

Tabella 9: Cod. semplice

- Codifica One-hot

Stato	Stringa			
S_0	0	0	0	1
S_1	0	0	1	0
S_2	0	1	0	0
S_3	1	0	0	0

Tabella 10: Cod. 1-hot

Usando la prima ottengo:

State	Encoding
S0	00
S1	01
S2	10
S3	11

Current State		Inputs		Next State	
$S1$	$S0$	TA	TB	$S'1$	$S'0$
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

Figura 26: Transizioni semaforo

Posso adesso ricavare i valori dei prossimi stati:

- $S'_1 = S_1 \oplus S_0$
- $S'_0 = \bar{S}_1\bar{S}_0Ta + \bar{S}_1\bar{S}_0T_b$

Faccio la stessa cosa per l'output:

Output	Encoding
green	00
yellow	01
red	10

Current State		Outputs			
$S1$	$S0$	$LA1$	$LA0$	$LB1$	$LB0$
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Figura 27: Output semaforo

Ed avrò:

- $L_{a1} = S_1$
- $L_{a0} = \bar{S}_1S_0$
- $L_{b1} = \bar{S}_1$
- $L_{b0} = S_1S_0$

Posso quindi creare lo schema completo:

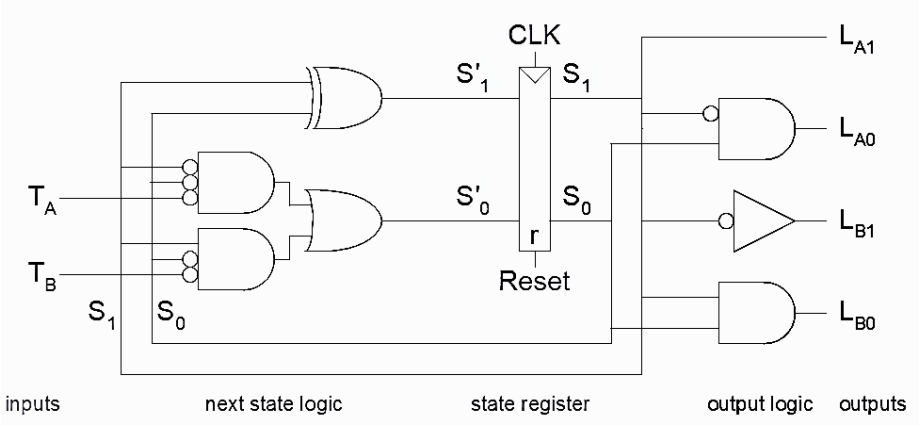


Figura 28: Schema semaforo

Mealy e Moore La differenza tra i 2 tipi sta nella gestione dell'output, uno associa gli output agli stati, l'altro alle transizioni.

Esempio:

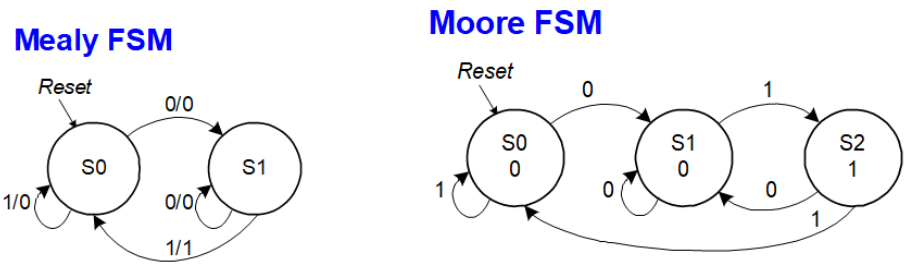


Figura 29: Differenze

Nell'automa di Mealy sugli archi sono indicati gli input/output.

La differenza risulta ancora più evidente controllando le tabelle:

Current State		Inputs	Next State	
$S1$	$S0$	A	$S'1$	$S'0$
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

State	Encoding
S0	00
S1	01
S2	10

Current State		Output
$S1$	$S0$	Y
0	0	0
0	1	0
1	0	1

Figura 30: Moore

Current State	Input	Next State	Output
$S0$	A	$S'0$	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

State	Encoding
S0	0
S1	1

Figura 31: Mealy

Intercambiabilità Si può passare tranquillamente tra i due tipi di automi seguendo alcuni passaggi:

Moore → Mealy

Basta spostare l'output dallo stato agli archi entranti in quello stato.

Mealy → Moore

- Fare l'opposto del paragrafo predecente
- Se ci sono output diversi entranti in un solo stato creare un nuovo stato per ogni output diverso
- In caso di nuovi stati aggiungere transizioni per mantenere la logica di funzionamento

Creazione FSM Progettazione:

1. Identifica input e output
2. Disegna il diagramma delle transizioni
3. Crea la tabella dell transizioni
4. Codifica gli stati
5. Crea le equazioni
6. Crea lo schema

Processo inverso:

1. Esamina il circuito
2. Ricava le equazioni
3. Crea le tabelle
4. Se possibile semplifica le tabelle
5. Assegna ad ogni combinazione valida uno stato
6. Riscrivi la tabella
7. Crea il diagramma delle transizioni

4.2.5 Timing

Dato il funzionamento dei flip-flop risulta importante che il segnale in ingresso resti stabile durante la salita del clock, in particolare non deve cambiare valore già prima della salita. (Tempo di setup)

Il discorso è simile riguardo l'output, infatti anch'esso dopo la salita potrebbe rimanere instabile per un certo periodo di tempo. (Tempo di hold)

Un ulteriore discorso va fatto riguardo la presenza di registri multipli, funzionando con lo stesso clock ma essendo collegati tramite un circuito servono delle garanzie sulla stabilità nel tempo.

In particolare l'input di R_2:

- Deve essere stabile per almeno il tempo di setup di R_2 prima della salita
- Deve essere stabile almeno per il tempo di hold dopo la salita

Un ultimo elemento da considerare è il possibile sfasamento del clock tra i vari componenti, infatti ad ognuno potrebbe arrivare in istanti diversi per varie ragioni e creare dei problemi.

4.3 Parallelismo

Esistono 2 tipi di parallelismo:

- **Spaziale**

  Dispositivi hardware multipli che svolgono più compiti

- **Temporale**

  Un certo compito viene diviso in più fasi

Alcune definizioni importanti:

- **Token** = Gruppo di input processati per produrre gruppi di output
- **Latenza** = Tempo impiegato da un token per passare dall'inizio alla fine del circuito
- **Throughput** = Numeri di token processati per unità di tempo

5 Blocchi costruttivi digitali

Per blocchi costruttivi digitali si intendono tutti gli elementi digitali semplici con funzioni ed interfacce ben strutturate, che possono essere usati come base per sistemi più grandi.

I blocchi in questione sono:

- **Porte logiche**
- **Multiplexer**
- **Decoder**
- **Registri**
- **Contatori**
- **Circuiti aritmetici**
- **Array di memoria**
- **Array logici**

5.1 Blocchi aritmetici

5.1.1 Sommatori

Ci sono 2 tipi di sommatori:

- **Half-adder**

  Esegue la somma degli input ricevuti

- **Full-adder**

  Come sopra ma ha in aggiunta l'input del riporto

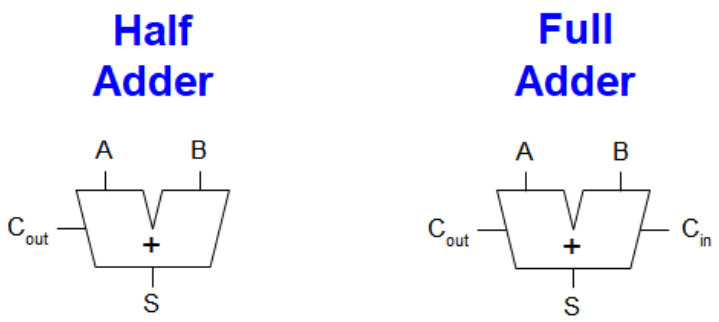


Figura 32: Sommatori

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$, $C_{out} = AB$

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C,$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

In caso di sommatori a più bit ci sono 3 modi per passare i riporti:

1. Ripple-carry
2. Carry-lookahead
3. Prefix

Ripple-carry Il modo più semplice ma anche il più lento, semplicemente vengono messi in catena i sommatori.

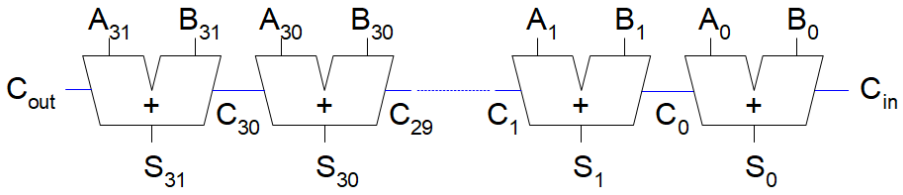


Figura 33: Catena di adder

Carry-lookahead Un certo sommatore in una catena può generare un riporto oppure propagare quello in input, sottoforma di equazioni diventa:

- **Genera** $= AB$
- **Propaga** $= A + B$
- **Riporto in uscita** $= AB + (A + B)C_{in}$

A questo punto posso dividere la catena in blocchi di lunghezza uguale ed usare questo metodo per passare il riporto dai blocchi precedenti a quelli successivi, in questo modo si va a ridurre il tempo di attesa.

La formula generica per un blocco diventa:

$$\begin{aligned} P_{i:j} &= P_i * P_{i-1} * P_{i-2} \ldots * P_j \\ G_{i:j} &= G_i + P_i(G_{i-1} + P_{i-1}(G_{i-2} \ldots * G_j)) \ldots \\ C_i &= G_{i:j} + P_{i:j} * C_{j-1} \end{aligned}$$

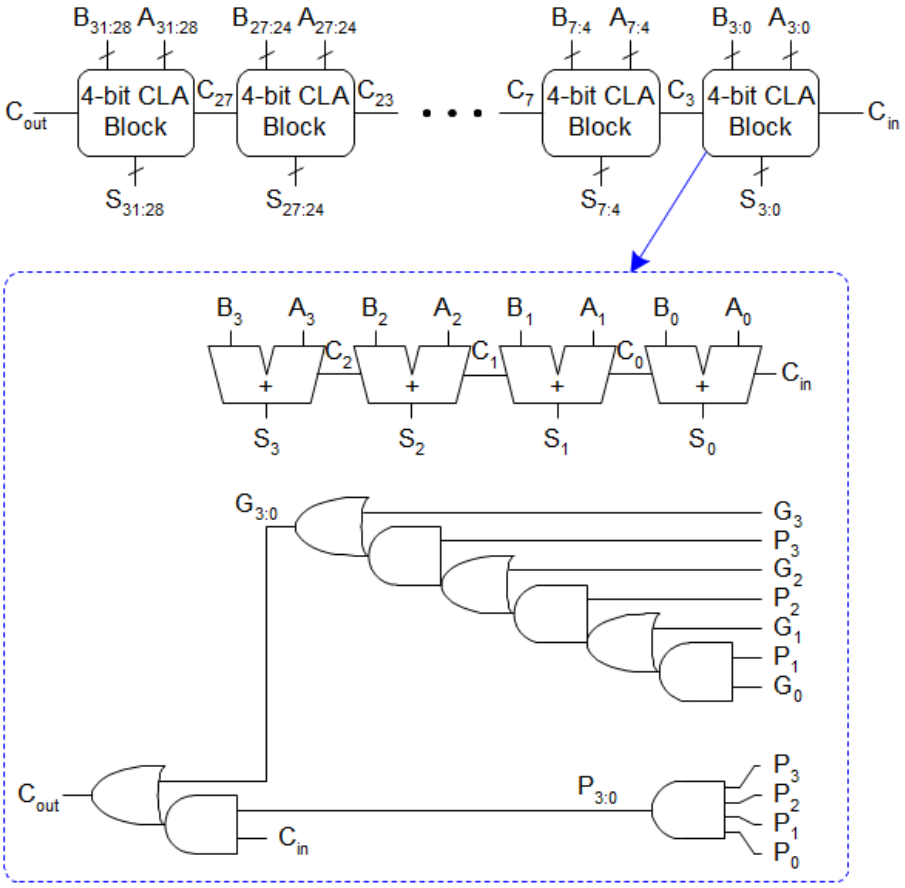


Figura 34: Schema Carry-lookahead

5.1.2 Sottrattori

Basta sommare B in complemento a 1 (negandolo) e aggiungere 1 tramite C_{in}:

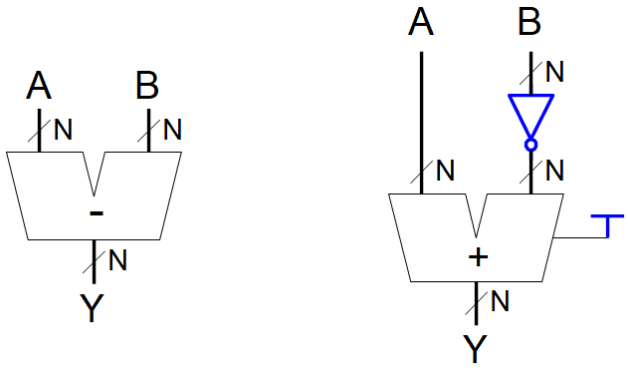


Figura 35: Sottrattore

5.1.3 Comparatori

Per controllare se due stringhe sono uguali basta usare una porta XNOR usando come input l'iesimo bit di entrambe le stringhe e poi fare un AND di tutte le porte XNOR.

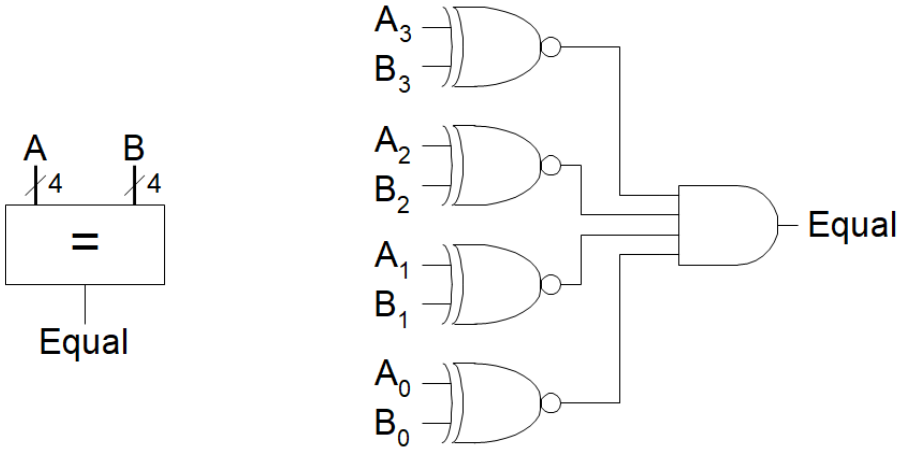


Figura 36: Comparatore eguaglianza

5.1.4 Shifter

Traslano la stringa a destra o a sinistra:

- **Logico**

  Sposta gli elementi della stringa di un certo numero di posizioni a destra o sinistra, inserisce 0 negli spazi rimasti vuoti.

  $11011 >> 2 = \mathbf{00}110$, $11011 << 2 = 011\mathbf{00}$

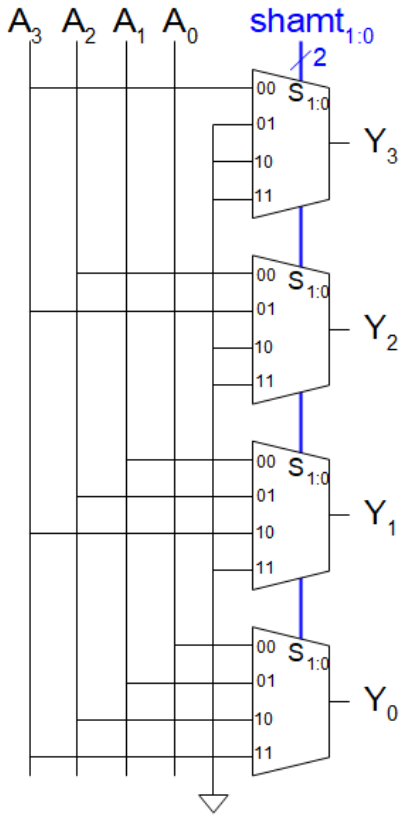


Figura 37: Shifter DX nibble

- **Aritmetico**

  Uguale al logico ma quello a dx riempe con il MSB.

- **Rotatore**

  Ruota un certo numero di bit all'altra estremità della stringa.

  $11011\ ROL\ 2 = 110\mathbf{11}$, $11011\ ROR\ 2 = \mathbf{11}110$

5.2 Memorie

Matrici di bit usate per contenere grandi quantità di dati, hanno:

- 2^x righe con x input d'indirizzo
- Colonne pari agli input/output dati

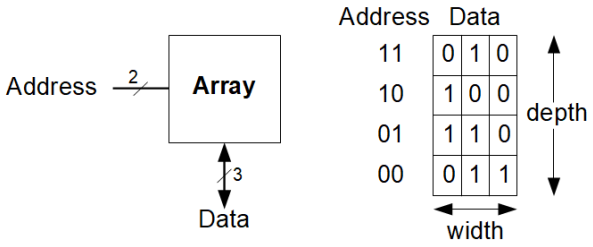
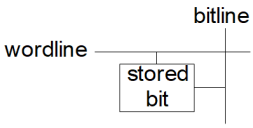


Figura 38: Array 2×3

Per leggere una singola cella bisogna dare in input l'indirizzo della stringa e poi leggere l'output dati giusto.



wordline	stored bit	bitline
0	X	Z
1	0	0
1	1	1

Figura 39: Singola cella di memoria

Tipi di memorie

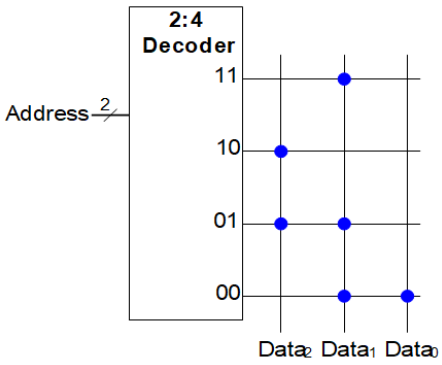
- **RAM** (Random access memory)
  - Volatile = Perde i dati senza corrente
  - Lettura/scrittura veloce

  Ci sono 2 tipi di RAM:

  - **DRAM** (Dinamica)
  - **SRAM** (Statica)

  La differenza sta nel modo di immagazzinare i dati.
- **ROM** (Read only memory)
  - Non volatile = Mantiene i dati senza corrente
  - Lettura veloce
  - Solitamente solo lettura

Logica Data la struttura ed il fuzionamento delle memorie si può sfruttare un decoder per la gestione degli indirizzi e rappresentare la matrice tramite la **Dot Notation**, in questo modo si possono utilizzare per compiere logica tramite gli output (visti come equazioni).



$$\text{Data}_0 = \text{Add}\bar{\text{r}}_1\text{Add}\bar{\text{r}}_0$$
$$\text{Data}_1 = \text{Add}\bar{\text{r}}_1 + \text{Addr}_0$$
$$\text{Data}_2 = \text{Addr}_1 \oplus \text{Addr}_0$$

Figura 40: Esempio logica

5.3 Array logici

Sono dei componenti in cui si svolgono operazioni logiche sequenziali.

PLA

- M input
- zona AND
- N implicanti generati da quest'ultima
- zona OR
- P output

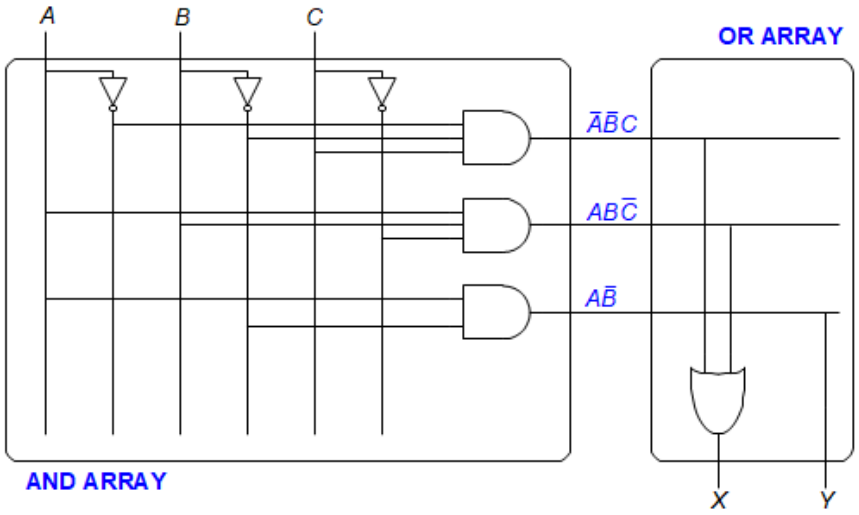


Figura 41: Esempio PLA

FPGA Composti da:

- Elementi logici
- Elementi input/output
- Interconnessioni tra questi
- RAM e moltiplicatori (in alcuni casi)

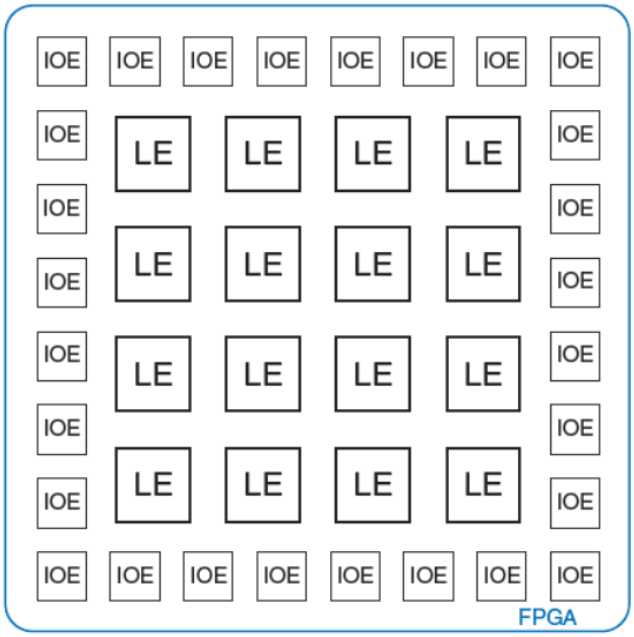


Figura 42: Schema FPGA

Gli elementi logici sono composti a loro volta da:

- Lookup tables
- Flip-flop
- Multiplexer

5.4 Altri blocchi

Contatore Un contatore con x output genererà in sequenza tutte le 2^x possibili combinazioni.
Se $x = 3 \implies (000, 001, 010, 011, 100, 101, 110, 111, 000, 001, \ldots)$

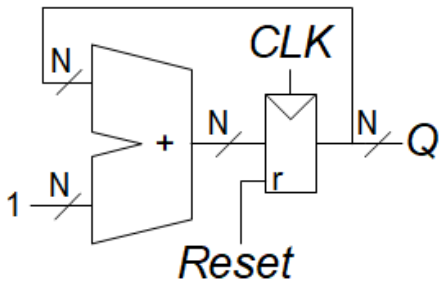


Figura 43: Schema contatore

Shift register "Shifta" un nuovo bit In e l'ultimo bit Out ad ogni colpo di clock.

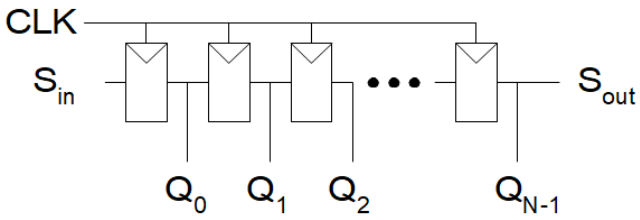


Figura 44: Schema Shift Register

6 Linguaggi descrittivi dell'Hardware

I linguaggi descrittivi dell'hardware servono a specificare funzioni logiche e produrne gli schemi ottimizzati.

Permettono inoltre di simulare dei circuiti fornendo degli input e di generare una lista di tutti i componenti presenti e i loro collegamenti.

6.1 SystemVerilog

Si tratta di uno dei linguaggi di questo tipo più utilizzati.

6.1.1 Sintassi ed altre cose basilari

- Case sensitive
- I nomi non possono iniziare con numeri
- Gli spazi sono ignorati
- Commento singolo //
- Commento più righe /* */
- Assegnamento bloccante con =
- Assegnamento non bloccante con <=
- Gli assegnamenti condizionali si possono fare con l'operatore ternario
- Sono presenti X e Z
- case() ... endcase è uno switch
- casez() contempla valori Z/?
- I numeri è opportuno scriverli nella forma $N'Bvalore$ con:
  - N = numero di bit
  - B = base usata (binaria,ottale,decimale,esadecimale),(b,o,d,h)
  - valore = il numero
- Si può inserire un delay tramite $\#x$ con x = numero di colpi di clock

Un modulo può essere descritto in 2 modi:

- **Strutturale** = Descrivendo come i suoi sotto-moduli sono interconnessi
- **Comportamentale** = Descrivendo cosa deve fare

6.1.2 Moduli comportamentali

Algorithm 1 $Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$

```
module example(input logic a,b,c, output logic y);
assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
endmodule
```

6.1.3 Moduli strutturali

Algorithm 2 NAND3

```
module and3(input logic a,b,c, output logic y);
assign y = a & b & c;
endmodule

module inv(input logic a, output logic);
assign y =~ a;
endmodule

module nand3(input logic a,b,c, output logic y);
// Definisco un segnale interno
logic n1;

// Definisco le istanze
and3 andgate(a,b,c,n1);
inv inverter(n1,y);
endmodule
```

6.1.4 Array

Si possono dichiarare array di bit tramite [i:j].

Normalmente le operazioni tra array avvengono su tutti i bit se non meglio specificato, in alternativa si può ridurre un array ad una singola variabile (tramite operazione) facendo per esempio $y = \&\ a$.

6.1.5 Logica sequenziale

Si possono creare dei componenti singoli tramite il costrutto always @ (args), in particolare:

- **always_ff**: flip-flop
- **always_latch**: latch
- **always_comb**: logica combinatoria

N.B. Bisogna rispettare la logica di funzionamento del componente scelto.

Algorithm 3 D Flip-flop 2 bit

```
module flop(input logic clk, input logic [1:0]d, output logic [1:0]q);
always_ff @ (posedge clk)
        q <= d;
endmodule
```