Architettura degli elaboratori

UniVR - Dipartimento di Informatica

Fabio Irimie

1° Semestre 2023/2024

Indice

1 Introduzione **3**
- 1.1 Hardware . . . 3
- 1.2 Campionamento dei dati . . . 3

2 Sistemi di codifica **4**
- 2.1 Codifica di informazioni non numeriche . . . 4
- 2.2 Numeri interi assoluti . . . 4
- 2.3 Numeri interi relativi . . . 5
  - 2.3.1 Codifica a modulo + segno . . . 5
  - 2.3.2 Codifica in complemento a 2 . . . 6

3 Numeri razionali **8**
- 3.1 Codifica in virgola fissa . . . 8
  - 3.1.1 Errore percentuale . . . 9
- 3.2 Codifica in virgola mobile . . . 9
  - 3.2.1 Divisione di bit tra segno, mantissa ed esponente . . . 10

4 Modelli **12**
- 4.1 Tabelle di verità . . . 14
  - 4.1.1 Operatore prodotto . . . 14
  - 4.1.2 Operatore somma . . . 14
  - 4.1.3 Operatore negazione . . . 15

5 Transistor **15**
- 5.1 Transistor CMOS . . . 15
  - 5.1.1 Transistor N . . . 15
  - 5.1.2 Transistor P . . . 15
  - 5.1.3 Circuito di negazione (NOT) . . . 16
  - 5.1.4 Circuito del prodotto (AND) . . . 17
  - 5.1.5 Circuito della somma (OR) . . . 17

6 Espressione in somma di prodotti **18**
- 6.1 Tecniche di ottimizzazione . . . 19
- 6.2 Terminologia . . . 20

7 Assorbimento svolto graficamente **20**
- 7.1 Mappe di Karnaugh . . . 22

8 Metodo di Quine-McCluskey **24**
- 8.1 Esempio con funzione completamente specificata . . . 24
- 8.2 Esempio con funzione parzialmente specificata . . . 27

9 Circuiti Combinatori **30**
  - 9.0.1 PLA (Programmable Logic Array) . . . 30
  - 9.0.2 CPLD (Complex Programmable Logic Device) . . . 31
  - 9.0.3 FPGA (Field Programmable Gate Array) . . . 31
  - 9.0.4 SoC (System on Chip) . . . 32

10 Laboratorio **32**
10.1 SIS 32
10.2 Ottimizzazione 34

11 Modelli gerarchici **35**

12 Ottimizzazione approssimata multilivello **36**

13 Mapping tecnologico **36**

14 Sintesi a N livelli **37**
14.1 Network 38
14.2 Algoritmi 39
14.3 Script 39
14.3.1 full_simplify 40

15 Hardware design su FPGA con HDL **42**
15.1 EDA (Electronic Design Automation) 42
15.2 Verilog 42

16 Circuiti sequenziali **43**
16.1 Astrazione 46

17 Macchine a stati finiti (FSM) **48**
17.1 Rappresentazione delle FSM 49
17.1.1 State Transition Table (STT) 49
17.1.2 State Transition graph (STG) 50
17.2 Modello di Huffman 52
17.3 Codifica degli stati 53

18 Equivalenza tra macchina a stati **56**

1 Introduzione

L'informatica è nata per la risoluzione di problemi di calcolo, in particolare quelli di calcolo numerico. Per questo motivo i primi computer erano macchine che eseguivano operazioni aritmetiche. Per risolvere questi problemi si usano degli algoritmi che sono una sequenza di istruzioni semplici che portano poi a risolvere problemi di complessità variabile. Anche gli algoritmi hanno una complessità che deve essere adeguata alla risoluzione del problema.

1.1 Hardware

Un algoritmo deve essere trasformato in un processo di calcolo automatico, quindi deve essere implementato tramite hardware. Ci sono due tipi di hardware:

- **Embedded** che è un hardware dedicato ad un singolo compito. Ad esempio il microonde.
- **General purpose** non si sa l'utilizzo finale, quindi ha funzionalità generali ampliate dal software installato. L'hardware general purpose è programmabile attraverso il software. Un esempio è il PC.

In base al tipo di hardware l'algoritmo viene implementato in diversi modi:

- **Algoritmo** $\rightarrow$ **Software**: Tramite un linguaggio di programmazione
- **Algoritmo** $\rightarrow$ **Hardware embedded**: Tramite linguaggi di basso livello come C, Assembly o il sistema operativo.
- **Algoritmo** $\rightarrow$ **Hardware**: Tramite sintesi logica

1.2 Campionamento dei dati

Ogni cosa nel mondo è rappresentabile da funzioni continue nel tempo $f(t)$, ma con risorse finite è impossibile rappresentare infiniti dati, bisogna quindi campionarli.

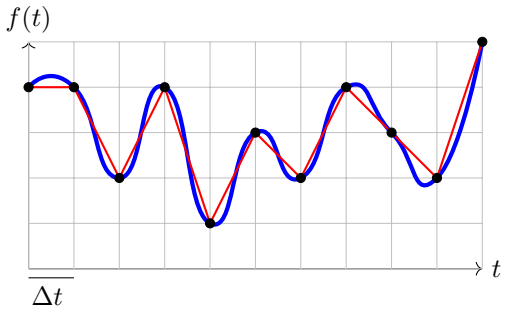


Figura 1: Funzione casuale continua nel tempo

Per campionare la funzione nella figura 1 bisogna scegliere un intervallo di tempo Δt e prendere un valore della funzione ogni Δt. In questo caso le linee verticali rappresentano il **campionamento**, mentre quelle orizzontali reppresentano la **discretizzazione o quantizzazione**. La linea rossa è una spezzata approssimata della funzione continua, infatti per il teorema di Shannon:

Teorema 1 *Deciso il grado di errore da voler compiere, esistono una precisa frequenza di campionamento e un intervallo di discretizzazione che garantiscono quell'errore.*

Il sistema di calcolo è ora diventato digitale, cioè elabora i segnali numerici in ingresso per produrre segnali numerici in uscita.

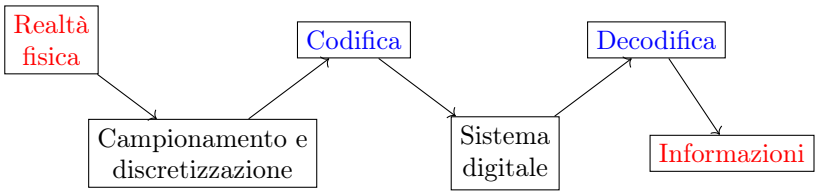


Figura 2: Dalla realtà fisica al sistema digitale

2 Sistemi di codifica

Ogni sistema digitale lavora in base binaria, quindi entrano N bit ed escono M bit. I bit in uscita devono essere codificati per realizzare delle informazioni. Ci sono 2 tipi di informazioni:

- **Informazioni intelleggibili**: sono già chiare agli esseri umani, come un testo scritto.
- **Informazioni non intelleggibili**: hanno bisogno di macchine per essere riprodotte, come le casse per l'audio.

2.1 Codifica di informazioni non numeriche

Ogni informazione deve avere un codice univoco in modo che il sistema digitale non possa sbagliare a decodificarla. Date M informazioni si ricavano $n = log_2(M)$ codici disponibili per rappresentarle.

Esempio 2.1
Con $M = 7$ informazioni:

- $n = log_2(7) \approx 3$ *bit*
- $2^3 = 8$ *codici disponibili*

2.2 Numeri interi assoluti

I numeri interi assoluti rappresentano solo i valori da 0 a $2^n - 1$, dove n è il numero di bit disponibile.

La codifica da base decimale a base binaria prende il nome di **codifica a modulo**

Esempio 2.2
Si deve convertire il numero 57_{10} *in base binaria*

$$n = log_2(57) = 6 \; bit \; (minimi)$$

$$\sum_{i=1}^{n-1} 2^n - 1 = 63 \; (codici \; massimi)$$

Si eseguono i seguenti passaggi:

1. *Si sottraggono le potenze di 2 partendo da* $n - 1$.
   - *Se la potenza* 2^i *è minore o uguale del numero, allora si moltiplica per 1.*
   - *Se la potenza* 2^i *è maggiore del numero, allora si moltiplica per 0.*
2. *Le sottrazioni continuano fino a quando si giunge a 0.*

$$57_{10} - 1 * 2^5 = 25_{10} - 1 * 2^4 = 9_{10} - 1 * 2^3 = 1_{10} - 0 * 2^2 = 1_{10} - 0 * 2^1 = 1_{10} - 1 * 2^0$$

$$57 = 111001$$

2.3 Numeri interi relativi

La codifica più ovvia per i numeri interi relativi è la codifica a **modulo + segno**. Tuttavia rappresenta varie problematiche, per cui si preferisce usare la codifica in **complemento a 2**.

2.3.1 Codifica a modulo + segno

$$\text{Intervallo: } -2^{n-1} \leq N \leq 2^{n-1} - 1$$

Il segno si rappresenta con un bit, 0 per il positivo e 1 per il negativo. Il bit più significativo è il bit del segno, mentre i bit meno significativi rappresentano il modulo.

1 bit: segno ±	7 bit: modulo

Figura 3: Bit dedicati alla codifica a modulo + segno

Considerando l'esempio 2.2 si hanno le seguenti rappresentazioni:

$$+57_{10} = \mathbf{0}|111001_2$$
$$-57_{10} = \mathbf{1}|111001_2$$

Sorge però un problema quando si vuole rappresentare il valore 0_{10}, che in binario risulterebbe:

$$+0_{10} = \mathbf{0}|000000_2$$
$$-0_{10} = \mathbf{1}|000000_2$$

Inoltre le somme che passano dal positivo al negativo e viceversa risultano errate.

2.3.2 Codifica in complemento a 2

$$\text{Intervallo: } -2^{n-1} \leq N \leq 2^{n-1} - 1$$

La codifica in complemento a 2 rimuove tutti i problemi della codifica in modulo + segno. Questa codifica infatti rende le somme molto più semplici. La somma facile infatti è l'obiettivo di questa codifica e parte dell'idea di trovare la codifica di -1, pertanto si cerca di formulare $-1 + 1 = 0$.

Obiettivo	Risultato
$????_2 +$	1111_2+
$0001_2 =$	$0001_2 =$
$0000_2 =$	0000_2

Tabella 1: Obiettivo della codifica in complemento a 2

Se si considera il numero di bit $n = 4$, allora l'intervallo di valori è $-2^3 \leq N \leq 2^3 - 1$:

$0_{10} = 0000_2$	$-1_{10} = 1111_2$
$1_{10} = 0001_2$	$-2_{10} = 1110_2$
$2_{10} = 0010_2$	$-3_{10} = 1101_2$
$3_{10} = 0011_2$	$-4_{10} = 1100_2$
$4_{10} = 0100_2$	$-5_{10} = 1011_2$
$5_{10} = 0101_2$	$-6_{10} = 1010_2$
$6_{10} = 0110_2$	$-7_{10} = 1001_2$
$7_{10} = 0111_2$	$-8_{10} = 1000_2$

Tabella 2: Codifica in complemento a 2 con $n = 4$ bit

I valori nel complemento a 2 ciclano, quindi se si somma 1 a 7 si ottiene -8.

Esempio 2.3
Sottrazione con il complemento a 2: $43 - 17 = 25$

$$n = 7 \textit{ bit}$$

1. *Per prima cosa si prende il valore assoluto del numero negativo* 17_{10} *e si converte in binario.*

$$17_{10} = 0010001_2$$

2. *Si inverte il numero trovato.*

$$!(0010001_2) = 1101110_2 = -18_{10}$$

3. *Si somma 1 al numero trovato.*

$$\begin{array}{r} 1101110 + \\ 0000001 = \\ \hline 1101111 \end{array}$$
$$1101111_2 = -17_{10}$$

Tabella 3: Somma di 1 al numero invertito

4. *Si somma il numero trovato al numero positivo.*

$$\begin{array}{r} 0010001 + \\ 1101111 = \\ \hline \mathbf{1}0011010 \end{array}$$

Tabella 4: Somma del numero positivo con il numero negativo

5. *Il risultato ottenuto è:*

$$\mathbf{1}0011010$$

Si osserva che c'è un bit in più rispetto a quelli disponibili (quello in grassetto), vuol dire che risulta in overflow[a]*, quindi si scarta il bit più significativo e si ottiene:*

$$0011010_2 = 26_{10}$$

che è il risultato corretto.

[a]Indica il "traboccamento", cioè se viene superato il limite massimo l'overfflow è un errore, non perchè sia sbagliata la somma, ma perchè il risultato non è codificabile con il numero di bit disponibili

Estensione del numero con il complemento a 2

- Se un numero è **positivo** va esteso con gli **0**

$+57_{10}+$	$0111001_2 +$
$+7_{10} =$	$\mathbf{0000}111_2 =$
$+64_{10}$	1000010_2

Tabella 5: Estensione di un numero positivo

- Se un numero è **negativo** va esteso con gli **1**

$+57_{10}+$	$0111001_2 +$
$-7_{10} =$	$\mathbf{1111}111_2 =$
$+50_{10}$	10110010_2

Tabella 6: Estensione di un numero negativo

3 Numeri razionali

I numeri razionali sono composti da una parte intera e una parte frazionaria. Si possono codificare in 2 modi:

- **Virgola fissa**(fixed point): viene usata maggiormente nei sistemi embedded quando si sa a priori il numero più grande e la precisione che si vuole ottenere
- **Virgola mobile**(floating point): viene usata maggiormente nei sistemi general purpose.

3.1 Codifica in virgola fissa

Esempio 3.1
Si hanno a disposizione 8 bit: 4 per la parte intera e 4 per la parte frazionaria. Vogliamo decodificare il numero 0110.1011_2*:*

$$\overbrace{\overset{2^3}{0}\ \overset{2^2}{1}\ \overset{2^1}{1}\ \overset{2^0}{0}}^{+6}.\overbrace{\overset{2^{-1}}{1}\ \overset{2^{-2}}{0}\ \overset{2^{-3}}{1}\ \overset{2^{-4}}{1}}^{\frac{1}{2}+\frac{1}{8}+\frac{1}{16}}$$

$$+6+\frac{1}{2}+\frac{1}{8}+\frac{1}{16}=6+\frac{11}{16}=\frac{107}{16}=6.6875$$

Se si vuole codificare un numero da decimale a binario bisogna tenere in considerazione che non è certo che il numero sia razionale anche in base 2, quindi bisogna approssimare per rappresentarlo.

Esempio 3.2
Prendiamo in considerazione $+4+\frac{3}{5}$*, in questo caso bisogna andare "a tentoni" e trovare la rappresentazione binaria che approssima con il minor errore possibile.*

$$4_{10}=0100_2$$

$$0.1001=\frac{9}{10}\Delta\frac{3}{80}$$

$$0.0111 = \frac{7}{16}\Delta - \frac{4}{80}$$
$$0.0110 = \frac{3}{8}\Delta\frac{9}{40}$$
$$\underline{0.1010 = \frac{5}{8}\Delta - \frac{1}{40}}$$

Δ *rappresenta l'errore, quindi la rappresentazione più vicina è* 0100.1010_2. *Però non è stato rappresentato* $\frac{3}{5}$, *ma* $\frac{1}{2} + \frac{1}{16} = \frac{9}{16}$.

Questo metodo è pesante perchè bisogna controllare più alternative.

3.1.1 Errore percentuale

Bisogna decidere se calcolarlo rispetto alla parte intera o a quella frazionaria. Nel seguente esempio viene calcolato l'errore percentuale rispetto alla parte frazionaria dell'esempio 3.2.

Esempio 3.3

$$\frac{1}{40} : \frac{3}{5} = \frac{1}{40} * \frac{5}{3} = \frac{1}{24} \approx 0.052\%$$

Il massimo errore che si può fare è l'overflow.

3.2 Codifica in virgola mobile

Gli standard della virgola mobile sono: IEEE 754. Questo standard è stato rivisto molte volte e ora viene usato da tutte le codifiche per i numeri in virgola mobile.
Il numero viene separato in 3 parti:

- **S**: Segno
- **e**: Esponente
- **M**: Mantissa

La struttura del numero è quindi:

$$N = \pm \cdot M^{\pm e}$$

Questo permette di dividere il numero in modo da poter scegliere quanti bit dedicare alla mantissa e quanti all'esponente. Si riscontrano però i seguenti problemi:

- Bisogna scegliere la base in cui fare la codifica $\rightarrow$ base 2
- Bisogna scegliere la divisione di bit tra *segno*, *mantissa* e *esponente* $\rightarrow$ 1 S, 23 M, 8 e
- La rappresentazione deve essere univoca $\rightarrow$ $1. \ldots_2$
- Bisogna trovare un modo per rappresentare gli errori

Se la mantissa e la base sono in base 2 la moltiplicazione e la divisione sono agevolate tramite l'utilizzo dello *shift*.

- $0110 \cdot 2 = 1100$ è uno shift a sinistra in binario.

$$\begin{array}{c} 0110 \\ \downarrow\downarrow\downarrow \\ 1100 \end{array}$$

Figura 4: Shift a sinistra in binario

- $1010/2 = 0101$ è uno shift a destra in binario.

$$\begin{array}{c} 1010 \\ \downarrow\downarrow\downarrow \\ 0101 \end{array}$$

Figura 5: Shift a destra in binario

3.2.1 Divisione di bit tra segno, mantissa ed esponente

Un numero è rappresentabile in 2 modi:

- Singola precisione 32 bit $\rightarrow$ float
- Doppia precisione 64 bit $\rightarrow$ double

Prendiamo in considerazione 32 bit, ora dobbiamo decidere quanti bit dedicare alla mantissa e all'esponente.

$$2^{\pm e}$$

$$\begin{array}{c} |e| = 4bit = 2^{+7} \\ 5bit = 2^{+15} \\ 6bit = 2^{+31} \\ 7bit = 2^{+63} \\ 8bit = 2^{+127} \end{array}$$

L'impatto dei bit sull'esponente è doppiamente esponenziale, quindi cresce tantissimo.

- **8 bit** all'esponente, quindi l'esponente può assumere valori da -127 a $+127$.
- **23 bit** alla mantissa, quindi la mantissa può assumere valori da 0 a $2^{23} - 1$
- **1 bit** al segno.

1 bit: segno ±	8 bit: esponente	23 bit: mantissa

Figura 6: Bit dedicati alla codifica in virgola mobile

Per la rappresentazione univoca la mantissa si codifica in virgola fissa. Cioè si parte da una mantissa con un **punto fisso** e dividendo o moltiplicando (shift) si può spostare la virgola per arrivare alla forma **1.00000...** e questa forma è la rappresentazione univoca.

Questa operazioe si chiama **normalizzazione** e visto che la rappresentazione è sempre la stessa l'*1.* non viene rappresentato, quindi viene inserito nella mantissa solo tutto ciò che viene dopo.

$$\begin{gathered}
11111111\ \pm\infty \\
11111110\ +127 \\
\dots \\
00000000\ \pm 0 \\
\dots \\
00000001\ -126 \\
00000000\ -127
\end{gathered}$$

Figura 7: Range dell'esponente

Si è deciso di codificare l'esponente in **Eccesso 127**. Quindi per rappresentare lo zero si usa come esponente il minore numero possibile: $1 \cdot 2^{-127} = 0$. Per codificare i numeri si somma 127 al numero desiderato e visto che i numeri possibili ora vanno da -127 a +127 se codifichiamo il risultato in modulo avremo dei numeri da 0 a 256.

Esempio 3.4
Si vuole decodificare il seguente numero:

$$1\ 01110111\ 0110...0$$

$$M = -(1 + \frac{1}{4} + \frac{1}{8}) * 2^e = -(\frac{11}{8}) * 2^e$$

$$e = (1 + 2 + 4 + 16 + 32 + 64) - 127 = 119 - 127 = -8$$

$$N = -\frac{11}{8} * 2^{-8}$$

Esempio 3.5
Codifica $+(4+\frac{1}{2}+\frac{1}{16})*2^{+34}$

1. *Sappiamo già che il numero è positivo quindi:*

$$S = 0$$

2. *Calcoliamo la mantissa:*

$$4+\frac{1}{2}+\frac{1}{16} = \underbrace{100}_{4_{10}}.\underbrace{10010\ldots0}_{\frac{1}{2}+\frac{1}{16}}$$

3. *La mantissa va normalizzata moltiplicando per 4:*

$$100.10010\ldots0 * 2^{+2} = 1.0010010\ldots0$$

$$M = 0010010\ldots0$$

4. *Calcoliamo l'esponente:*

$$e = 34+2 = 36$$

Si aggiunge 2 perchè abbiamo fatto lo shift di 2 bit.

$$e = 36+127 = 163$$

$$163_{10} = 10100011_2$$

5. *Il numero in virgola mobile è:*

$$0\ 10100011\ 0010010\ldots0$$

- $0\ 00000000\ 0...0 = +0$
- $1\ 00000000\ 0...0 = -0$

Quando l'esponente è tutto 1 e la mantissa tutta 0 allora equivale a $\pm\infty$ in base al primo bit. Se invece la mantissa è diversa da 0 con esponente tutti 1 allora rappresenta un errore NaN.

4 Modelli

Per un progetto bisogna creare un **modello** che rappresenti il sistema. Boole ha cercato di rappresentare tutte le algebre. Lo ha fatto attraverso una quintupla: $< B^n, \cdot, +, \{0,1\} >$

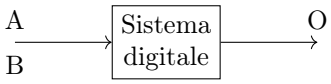


Figura 8: Modello di un sistema digitale

- B^n è l'insieme di valori
- $\{0, 1\}$ è l'alfabeto (sistema binario)
- "$\cdot$" e "$+$" sono 2 operatori

Bool garantisce che si può creare qualsiasi funzione utilizzando soltanto i 2 operatori:

$$f(B^n) \to B^m$$

Esempio 4.1
Si vuole creare un modello con 2 bit in entrata e 1 in uscita:

$$n = 2\ m = 1$$

$O = 1 \leftrightarrow A = B$
$f(B^2) \to B$

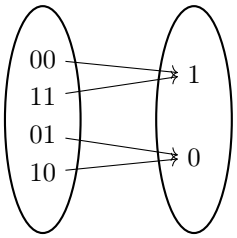


Figura 9: Modello di un sistema digitale

Per mappare i valori in ingresso con quelli di uscita si usa una ***tabella di verità****:*

A B	O
0 0	1
0 1	0
1 0	0
1 1	1

Tabella 7: Tabella di verità

Chiamiamo mintermine un punto dello spazio booleano in ingresso in cui la funzione vale 1. Il maxtermine è il contrario. L'insieme di mintermini $\{{m_0}^a, m_3\}$ *si chiama* ***ON-SET*** *L'insieme dei maxtermini* $\{m_1, m_2\}$

*si chiama **OFF-SET**. Basta uno dei due insiemi (ON-SET, OFF-SET) per definire la funzione.*

$$m_3 = A \cdot B$$

Dire che m_3 è il prodotto delle due variabili è un modo corretto per rappresentarlo.

$$m_0 = \bar{A} \cdot \bar{B}$$

Per rappresentare il mintermine basta fare il prodotto delle variabili se valgono 1 o delle variabili negate se valgono 0.

Per rappresentare la funzione si può usare la somma dei mintermini:

$$O = m_0 + m_3 = \bar{A} \cdot \bar{B} + A \cdot B = 0$$

Questa rappresentazione viene detta: Espressione in somma di prodotti

Teorema 2 *Dato un ON-SET c'è sempre una sola espressione in somma di prodotti che lo rappresenti.*

[a] m_n: n è il valore in modulo del relativo numero binario, m sta per modulo. $m_3 = 11_2$

4.1 Tabelle di verità

4.1.1 Operatore prodotto

A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

Tabella 8: Tabella di verità dell'AND

4.1.2 Operatore somma

A	B	O
0	0	0
0	1	1
1	0	1
1	1	1

Tabella 9: Tabella di verità dell'OR

4.1.3 Operatore negazione

A	O
0	1
1	0

Tabella 10: Tabella di verità del NOT

5 Transistor

È un "comando di accensione" che permette di accendere o spegnere un circuito.

5.1 Transistor CMOS

5.1.1 Transistor N

Mette in collegamento 2 punti:

- Se la corrente è 0V allora non c'è collegamento
- Se la corrente è 3V allora c'è collegamento

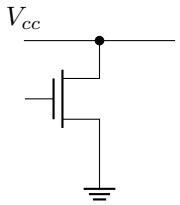


Figura 10: Transistor N

5.1.2 Transistor P

Mette in collegamento 2 punti:

- Se la corrente è 0V allora c'è collegamento
- Se la corrente è 3V allora non c'è collegamento

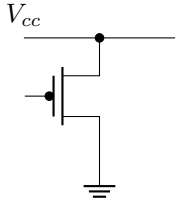


Figura 11: Transistor P

5.1.3 Circuito di negazione (NOT)

Si realizza con un transistor P e uno N in serie.

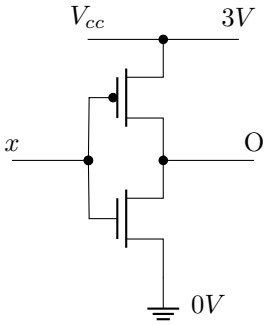


Figura 12: Circuito di negazione

La tabella della verità è:

x	O
0V	3V
3V	0V

Tabella 11: Tabella di verità del circuito

Se assegnamo ad ogni valore un numero binario:

x	O
0	1
1	0

Tabella 12: Tabella di verità del circuito in binario

Si può notare che è la funzione di negazione rappresentata con la seguente porta logica:

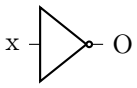


Figura 13: Porta logica NOT

5.1.4 Circuito del prodotto (AND)

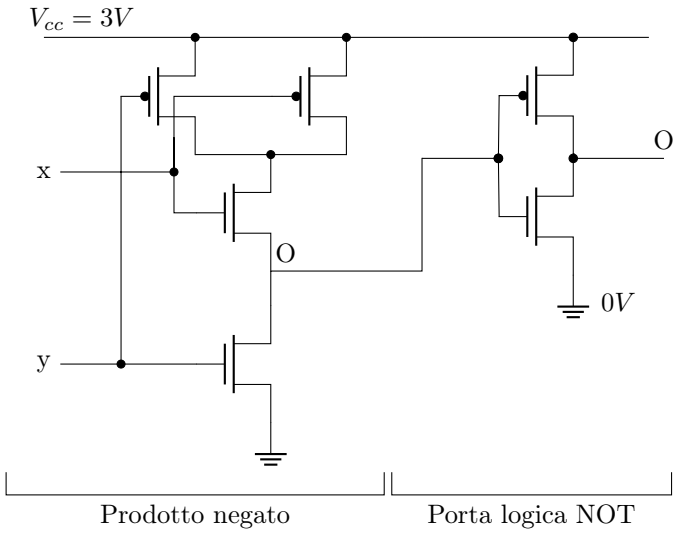


Figura 14: Circuito del prodotto

Il prodotto negato più il NOT è uguale ad un AND:

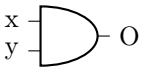


Figura 15: Porta logica AND

La tabella della verità è:

x y	O
0 0	0
0 1	0
1 0	0
1 1	1

Tabella 13: Tabella di verità del circuito

5.1.5 Circuito della somma (OR)

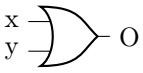


Figura 16: Porta logica OR

La tabella della verità è:

x y	O
0 0	0
0 1	1
1 0	1
1 1	1

Tabella 14: Tabella di verità della somma

6 Espressione in somma di prodotti

Il seguente circuito è un esempio di espressione in somma di prodotti dell'esempio 4.1:

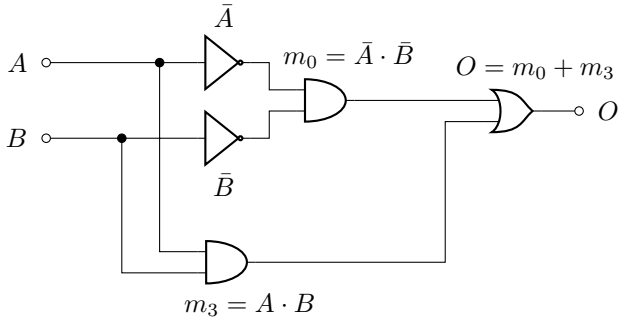


Figura 17: Circuito dell'espressione in somma di prodotti

I circuiti devono spesso tenere conto di alcune specifiche da ottimizzare:

- **Area**: minor numero di porte logiche
- **Latency**: più porte logiche si attraversano più sarà il ritardo
- **Power**: più porte logiche si attraversano più sarà il consumo
- **Safety**: più porte logiche si attraversano più sarà la probabilità di errore

Prendiamo in considerazione la funzione $f(B^3)^1 \to B$:

[1] Il numero di funzioni booleane possibili è $2^{2^3} = 256$ e il valore cresce esponenzialmente con l'aumento dei bit

X Y Z	O
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

Tabella 15: Tabella di verità della funzione

ON-SET= $\{m_1, m_3, m_5, m_7\}$
La funzione rappresentata con un'espressione in somma di prodotti è:

$$O = m_1 + m_3 + m_5 + m_7 = \bar{X}\bar{Y}Z + \bar{X}YZ + X\bar{Y}Z + XYZ$$

Proviamo a stimare le dimensioni di questo circuito. Si utilizza il concetto di **letterale** che è una coppia chiave-valore. La funzione O è composta da 12 letterali e questo numero è in relazione con il numero di transistor nel senso che se una funzione ha più letterali di un altra si può già sapere che avrà bisogno di un minor numero di transistor.

6.1 Tecniche di ottimizzazione

La regola principale dell'ottimizzazione è l'**assorbimento**: Preso un prodottp P moltiplicato ad un letterale a e la somma di questo prodotto, ma con il letterale negato $\bar{a}$ allora il risultato è $P \cdot (a + \bar{a})$ dove $(a + \bar{a})$ fa sempre 1, quindi rimane P.

$$aP + \bar{a}P = P \cdot (a + \bar{a}) = P$$

$$\underbrace{2 \cdot (|P| + 1)}_{\text{Cardinalità prima dell'assorbimento}} \quad \Rightarrow \quad \underbrace{|P|}_{\text{Cardinalità dopo l'assorbimento}}$$

Quindi se prendiamo come riferimento la funzione O si può applicare la regola dell'assorbimento per ridurre il numero di letterali:

$$\begin{array}{c} \bar{X}\bar{Y}Z + \bar{X}YZ + X\bar{Y}Z + XYZ \\ \searrow \quad \downarrow \quad \searrow \quad \downarrow \\ \bar{X}Z(\bar{Y} + Y) + XZ(\bar{Y} + Y) \end{array}$$

E riapplicando la regola si arriva al minimo:

$$\begin{array}{c} \bar{X}Z + XZ \\ \searrow \quad \downarrow \\ Z(\bar{X} + X) \end{array}$$

$$Z$$

6.2 Terminologia

Ogni mintermine è un prodotto (o implicante), ma dopo aver applicato la regola di assorbimento non è più un mintermine, ma soltanto prodotto (o implicante).

$$\bar{X}\bar{Y}Z \rightarrow \bar{X}Z$$

La Y non c'è più nel risultato dell'assorbimento, ciò vuol dire che non ci interessa il suo valore perchè non varia il risultato. Si può scrivere sia 11 che $1-1$
Quindi ad esempio:
$Z = --1 = 4$ mintermini: $\{001, 011, 101, 111\}$

Definizione 6.1
Implicante primo *è un implicante non contenuto in nessun altro implicante*

Definizione 6.2
La ***distanza di Hamming*** *è il numero di bit che differenziano 2 codici.*

$01\boldsymbol{10} \rightarrow 01\boldsymbol{01}$ *distanza di Hamming = 2*
$01\boldsymbol{0} \rightarrow 01\boldsymbol{1}$ *distanza di Hamming = 1*

7 Assorbimento svolto graficamente

Prendendo come riferimento la funzione $f(B^3)^1 \rightarrow B$ definita precedentemente (che chiameremo O) si può guardare la funzione come se fosse sul piano cartesiano con centro in un punto qualsiasi. Ogni punto adiacente al centro è un punto con distanza di Hamming = 1.

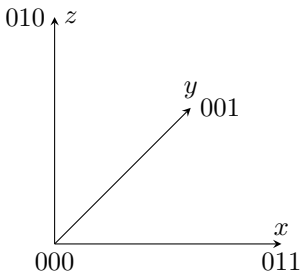


Figura 18: Funzione rappresentata su un piano cartesiano

L'assorbimento può essere fatto soltanto tra gli ON-SET con distanza di Hamming = 1. Per effettuare l'assorbimento ci si posiziona nel punto di un mintermine e si "guarda" in tutte le direzioni per eventuali altri mintermini con cui fare il prodotto.

Nella seguente figura i vertici rossi rappresentano gli OFF-SET e i vertici blu rappresentano gli ON-SET.

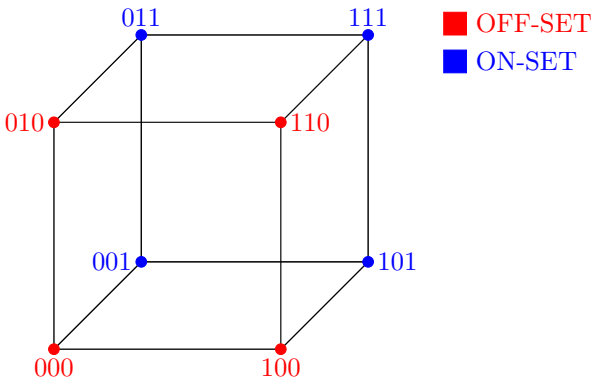


Figura 19: Funzione rappresentata su un cubo

Si trascura la faccia del cubo con l'OFF-SET per rendere la rappresentazione più semplice. Prendendo coppie di vertici dell'ON-SET sullo stesso lato del cubo si può fare il prodotto tra i 2 mintermini:

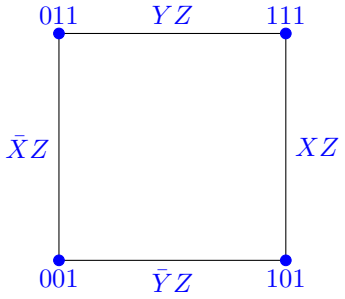


Figura 20: Prima semplificazione

Ora si può fare l'assorbimento anche tra i prodotti ottenuti dall'assorbimento:

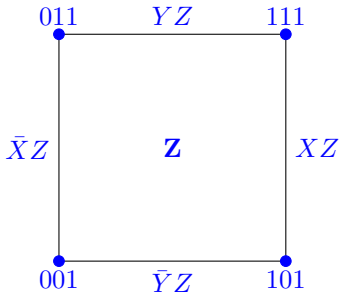


Figura 21: Seconda semplificazione

Si arriva quindi a dire che Z è un **implicante primo** perchè non c'è nessun altro implicante che lo contiene.

Definizioni utili 7.1
Quando si parla di implicante si può anche dire ***sottocubo*** *e l'implicante primo può essere chiamato anche* ***sottocubo di dimensione massima****.*

Implicante = Sottocubo
Implicante primo = Sottocubo di dimensione massima

Esistono condizioni favorevoli (come la funzione O) in cui un implicante primo contiene tutti i mintermini della funzione.
Ci sono più tipi di implicanti primi:

- **Essenziali**: includono almeno un mintermine che non è coperto da nessun altro implicante primo (fanno parte della soluzione finale).
- **Non essenziali**: Implicanti primi che coprono mintermini coperti anche da altri implicanti. Si identificano con l'**algoritmo di copertura**

7.1 Mappe di Karnaugh

Karnaugh ha creato una mappa che permette di rappresentare su un piano tutte le variabili booleane (nel caso della funzione O si mettono i valori del cubo nella tabella) in modo da poter fare l'assorbimento in modo più semplice. I valori posti sopra le celle sono messi in modo che siano a distanza di Hamming = 1. Nella seguente mappa di Karnaugh si possono vedere i valori della funzione O:

xy / z	00	01	11	10
0	0	0	0	0
1	1	1	1	1

Tabella 16: Mappa di Karnaugh della funzione O

In questa mappa si può vedere che Z è un implicante primo (o sottocubo di dimensione massima). Le mappe di Karnaugh sono come una sfera, quindi se si va oltre il bordo si torna dall'altra parte.

Un altro esempio di mappa di Karnaugh è il seguente:

Esempio 7.1
Prendiamo in considerazione una funzione casuale a 4 variabili:

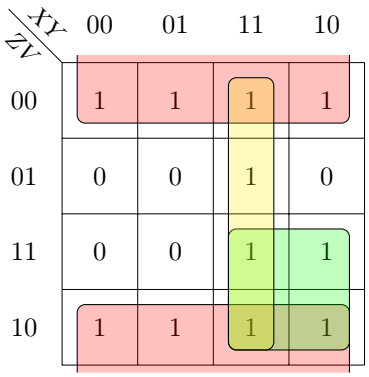


Tabella 17: Mappa di Karnaugh di una funzione casuale

Si può verificare che ci sono 3 implicanti primi essenziali:

- $\bar{V}$*: essenziale*
- XY*: essenziale perchè copre* 1101
- XZ*: essenziale perchè copre* 1011

$$O = \bar{V} + XY + XZ \quad (5\ letterali)$$

Per capire quali sono gli implicanti primi bisogna raggruppare gli 1 in rettangoli più grandi possibile, ma sempre di grandezza 2^n *(2, 4, 8, 16, ...). Per ciascun raggruppamento bisogna trovare le variabili che non cambiano il loro valore. Per il raggruppamento rosso:*

- X *cambia valore, passando da* 0 *in* 0000 *e* 0100 *a* 1 *in* 1100 *e* 1000*, quindi deve essere esclusa.*
- Y *cambia valore, passando da* 0 *in* 0000 *e* 1000 *a* 1 *in* 0100 *e* 1100*, quindi deve essere esclusa.*
- Z *cambia valore, passando da* 0 *in* 0000 *a* 1 *in* 0010*, quindi deve essere esclusa.*
- $\bar{V}$ *mantiene lo stesso stato in tutto il gruppo, quindi deve essere inclusa nel prodotto risultante*

Lo stesso ragionamento viene applicato per tutti i gruppi, fino ad arrivare al risultato finale.

Le mappe di Karnaugh sono utili soltanto se le variabili sono meno di 5, altrimenti bisogna usare più mappe.

8 Metodo di Quine-McCluskey

Questo metodo ha 2 versioni:

- Funzioni completamente specificate
- Funzioni parzialmente specificate

Si divide in 2 fasi:

1. Si espande il più possibile il problema per cercare il massimo grado di minimizzazione. (ad esempio trattando un *don't care* come 1 per permettere ulteriori ottimizzazioni)
2. Bisogna capire quali servono veramente.

8.1 Esempio con funzione completamente specificata

Esempio 8.1
Prendiamo una funzione completamente specificata

$$O = f(x, y, z, w) = \{m_1, m_4, m_5, m_6, m_7, m_9, m_{11}, m_{14}, m_{15}\}$$

m	$x\ y\ z\ w$
1	0 0 0 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
9	1 0 0 1
11	1 0 1 1
14	1 1 1 0
15	1 1 1 1

Tabella 18: Tabella dei mintermini

36 Letterali.
L'unico caso in cui due stringhe sono a distanza di Hamming $= 1$ *è quando il numero di* 1 *differisce di uno.*
Il metodo di Quine-McCluskey riordina le m *in base al numero di* 1 *che contengono, questo è il primo passo:*

m	x y z w	
1	0 0 0 1	$\surd$
4	0 1 0 0	$\surd$
5	0 1 0 1	$\surd$
6	0 1 1 0	$\surd$
9	1 0 0 1	$\surd$
7	0 1 1 1	$\surd$
11	1 0 1 1	$\surd$
14	1 1 1 0	$\surd$
15	1 1 1 1	$\surd$

Tabella 19: Tabella riordinata

Si individuano i gruppi che sono a distanza di Hamming 1. Nel prossimo passo confrontiamo i gruppi con 1 *bit* $= 1$ *e con* 2 *bit* $= 1$*, se sono a distanza di Hamming 1 allora si mette* don't care *nel bit che cambia. Nella prima colonna c'è la coppia di* m *che viene confrontata.*

m	$x\ y\ z\ w$	
1, 5	0 - 0 1	A
1, 9	- 0 0 1	B
4, 5	0 1 0 -	$\surd$
4, 6	0 1 - 0	$\surd$
5, 7	0 1 - 1	$\surd$
6, 7	1 0 1 -	$\surd$
6, 14	- 1 1 0	$\surd$
9, 11	1 0 - 1	C
7, 15	- 1 1 1	$\surd$
11, 15	1 - 1 1	D
14, 15	1 1 1 -	$\surd$

Tabella 20: Prima semplificazione

Tutti i mintermini della tabella 19 sono coperti da un implicante della tabella 20.

Ora si può semplificare anche la tabella 20 se i don't care sono nella stessa variabile:

m	$x\ y\ z\ w$	
4, 5, 6, 7	0 1 - -	E
6, 7, 14, 15	- 1 1 -	F

Tabella 21: Seconda semplificazione

I valori senza $\surd$ *sono implicanti primi perchè non sono coperti da nessun altro implicante della tabella 21. Anche i 2 valori nella tabella 21 sono*

implicanti primi.
Implicanti primi: A, B, C, D, E, F

$$A = 0 - 01 = \bar{X}\bar{Z}W$$

$$B = -001 = \bar{Y}\bar{Z}W$$

$$C = 10 - 1 = X\bar{Y}W$$

$$D = 1 - 11 = XZW$$

$$E = 01 - - = \bar{X}Y$$

$$F = -11 - = YZ$$

16 Letterali.
Ad ogni passo del metodo di Quine-McCluskey diminuisce il numero di letterali.

Ora bisogna trovare gli implicanti primi essenziali

m	A	B	C	D	E	F
1	1	1				
4					**1**	
5	1				1	
6					1	1
7					1	1
9		1	1			
11			1	1		
14						**1**
15				1		1

Tabella 22: Tabella con implicanti primi

E ed F sono essenziali perchè coprono m_4 *e* m_{14}*. Inoltre E ed F coprono anche* m_5, m_6, m_7 *e* m_6, m_7, m_{14}, m_{15}*.*

Tenendo in mente le m coperte, la tabella diventa:

	A	B	C	D
m_1	1	1		
m_9		1	1	
m_{11}			1	1

Tabella 23: Tabella senza implicanti essenziali

Euristica *È un metodo per trovare la soluzione corretta, ma non garantisce che sia ottima.*

*Se si cancella ad esempio m_1, m_9 perchè coperti da B, allora B domina A e C domina D per la regola di **dominanza per colonne**. Prendendo la colonna con più elementi ho più probabilità di trovare la soluzione ottima.*

$$O = E + F + B + C = \bar{X}Y + YZ + \bar{Y}\bar{Z}W + X\bar{Y}W$$

10 Letterali.
*La **pseudo-essenzialità** è l'essenzialità dopo aver già fatto un'ottimizzazione.*

Esempio 8.2
Dominanza per righe

	A	B	C	D
α	1		1	
β	1			1
γ		1	1	1
δ		1	1	
ϵ	1	1		1

Tabella 24: Tabella con implicanti primi

- *β dominato da ϵ*
- *δ dominato da γ*

Dobbiamo prendere β e δ perchè cancellando A e D si cancellano anche γ ed ϵ perchè sono dominati. La tabella diventa:

	A	B	C	D
α	1		1	
β	1			1
δ		**1**	1	

Tabella 25: Tabella senza implicanti essenziali

Si fa finta che B e D siano essenziali (essenzialmente scelti a caso).

8.2 Esempio con funzione parzialmente specificata

Potrebbe uscire nel risultato un *don't care*, ad esempio per condizioni di ingresso non utilizzate.

Prendiamo in considerazione una funzione booleana $f(B^n) = B$ parzialmente specificata che viene descritta tramite 3 insiemi:

- **ON-SET**: insieme delle configurazioni per cui vale 1
- **DC-SET**: insieme delle configurazioni per le quali la funzione non è specificata

- **OFF-SET**: insieme delle configurazioni per cui vale 0

L'intersezione fra i 3 insiemi deve essere vuota, mentre l'unione è l'insieme di tutte le configurazioni possibili.

Per conoscere tutti e 3 gli insiemi basta conoscerne 2 di essi.

Esempio 8.3
Funzione parzialmente specificata

x y z v	0
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	-
0 1 0 0	1
0 1 0 1	-
0 1 1 0	-
0 1 1 1	-
1 0 0 0	0
1 0 0 1	0
1 0 1 0	1
1 0 1 1	1
1 1 0 0	0
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

Tabella 26: Tabella della funzione parzialmente specificata

$$ON - SET = \{m_4, m_{10}, m_{11}, m_{13}, m_{14}, m_{15}\}$$
$$DC - SET = \{m_3, m_5, m_6, m_7\}$$

Il primo passo è quello di ampliare il problema, quindi si considerano i don't care come 1, e poi rioridnare i mintermini in base al numero di 1.

m	x y z v	
4	0 1 0 0	√
3	0 0 1 1	√
5	0 1 0 1	√
6	0 1 1 0	√
10	1 0 1 0	√
7	0 1 1 1	√
11	1 0 1 1	√
13	1 1 0 1	√
14	1 1 1 0	√
15	1 1 1 1	√

Tabella 27: Tabella riordinata

Il secondo passo è quello di tentare la semplificazione:

m	x y z v	
4, 5	0 1 0 -	√
4, 6	0 1 - 0	√
3, 7	0 - 1 1	√
3, 11	- 0 1 1	√
5, 7	0 1 - 1	√
5, 13	- 1 0 1	√
6, 7	0 1 1 -	√
6, 14	- 1 1 0	√
10, 11	1 0 1 -	√
10, 14	1 - 1 0	√
7, 15	- 1 1 1	√
11, 15	1 - 1 1	√
13, 15	1 1 - 1	√
14, 15	1 1 1 -	√

Tabella 28: Prima semplificazione

Ora si applica di nuovo la semplificazione:

m	x y z v	
4, 5, 6, 7	0 1 - -	A
3, 7, 11, 15	- - 1 1	B
5, 7, 13, 15	- 1 - 1	C
6, 7, 14, 15	- 1 1 -	D
10, 11, 14, 15	1 - 1 -	E

Tabella 29: Seconda semplificazione

Visto che non si possono fare ulteriori semplificazioni A, B, C, D, E sono tutti implicanti primi.

Ora si cerca di capire quali sono gli implicanti primi essenziali considerando però soltanto l'ON-SET:

m	A	B	C	D	E
m_4	**1**				
m_{10}					**1**
m_{11}		1			1
m_{13}			**1**		
m_{14}				1	1
m_{15}		1	1	1	1

Tabella 30: Tabella con implicanti primi

Si cancellano le righe A, C, E perchè sono implicanti primi essenziali, quindi si coprono anche tutte le righe delle colonne A, C, E che contengono 1. Si arriva quindi a coprire tutta la tabella e il risultato finale è:

$$O = A + C + E$$

9 Circuiti Combinatori

- **PROM**: Programmable Read Only Memory (ROM)
- **PLA**: Programmable Logic Array, attivano diverse porte logiche

I circuiti a 2 livelli sono composti da 2 livelli di porte logiche:

1. Porte AND
2. Porte OR

9.0.1 PLA (Programmable Logic Array)

Esempio 9.1
Un esempio di PLA:

$$O_1 = \bar{X}Y + YV + XZ$$

$$O_2 = \bar{X}\bar{Y} + YV + X\bar{Z}$$

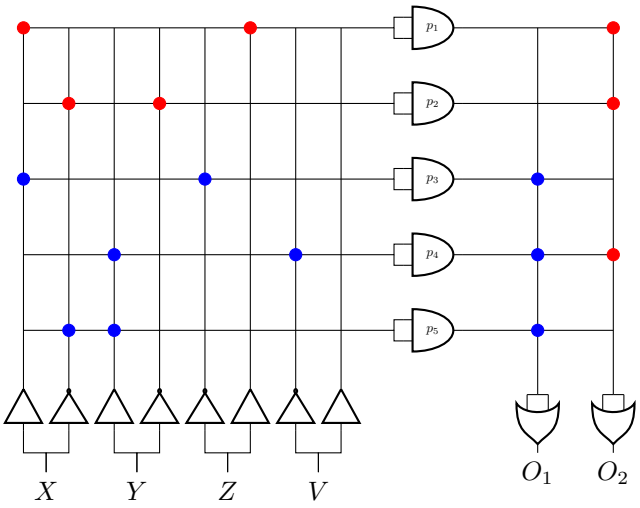


Figura 22: PLA

Prende in input coppie di variabili e le assegna ad un array di AND e l'output di questi AND viene assegnato ad un array di OR.

9.0.2 CPLD (Complex Programmable Logic Device)

I CPLD attivano diversi PLA:

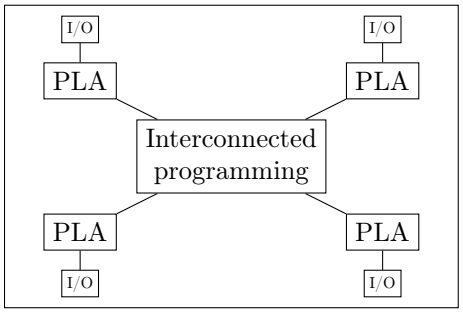


Figura 23: CPLD

9.0.3 FPGA (Field Programmable Gate Array)

I FPGA attivano diversi CPLD:

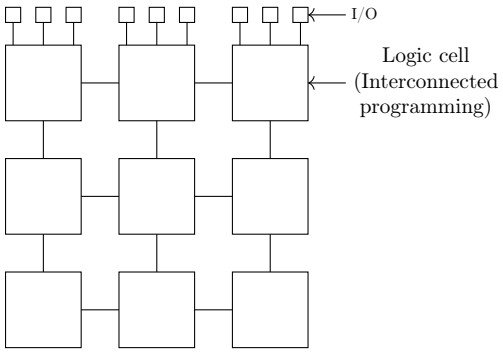


Figura 24: FPGA

9.0.4 SoC (System on Chip)

I SoC attivano diversi CPLD

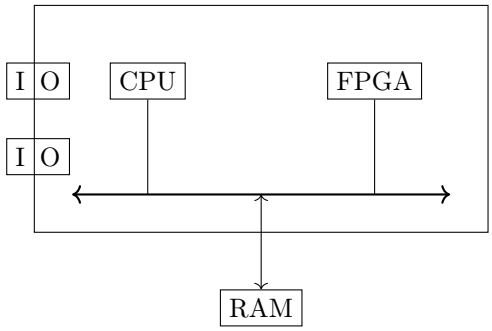


Figura 25: SoC

10 Laboratorio

10.1 SIS

È il successore di *Espresso* e permette di fare la sintesi di circuiti, cioè si genera passo dopo passo il layout per il silicio.

Il modello di codice di SIS è il seguente:

```
.model <model-male> // nome della funzione
.inputs <input-list> // elenco degli input
.outputs <output-list> // elenco degli output

.names // On-set/Off-set per ogni input
<command>
...
<command>

.end // il file deve essere
```

Esempio 10.1
Prendiamo in considerazoine la tabella di verità dell' implicazione logica:

a b	a $\Longrightarrow$ b
0 0	1
0 1	1
1 0	0
1 1	1

Tabella 31: Tabella di verità dell'implicazione logica

che si può scrivere anche nel seguente modo:

a b	a $\Longrightarrow$ b
0 -	1
1 0	0
1 1	1

Tabella 32: Tabella di verità ridotta

```
.model IMPLIES
.inputs I1 I2
.outputs O

.names I1 I2 O
// si definisce l'on-set o l'off-set
0- 1
11 1
.end
```

Esiste il comando **.exdc** che permette di specificare il **don't care set** di una funzione booleana.
La sintassi è la seguente

```
.exdc
.names [lista delle variabili]
[lista delle configurazioni, mettendo 1 come output] // l'1 come
output non vuol dire che si sta forzando il don't care a 1, ma
serve solo per il parser
```

Lista di comandi utili:

- **read_blif**: carica il modello sis;
- **simulate [valori in bit, separati da spazi]**: esegue un passo di simulazione del circuito;
- **help**: mostra i comandi disponibili;

- **help [comando]**: mostra la descrizione del comando;
- **print_stats**: fornisce informazioni sul circuito, quali numero di input (pi), output (po), elementi di memoria (latches), letterali (lits(sop)), numero di nodi (nodes);
- **quit**: esce da sis;
- **write_blif [nome file]**: salva il modello sis in un file;
- **write_blif**: stampa a video il file blif del circuito caricato in memoria senza dover lasciare l'ambiente SIS.
- **write_eqn**: stampa a video l'equazione del circuito caricato, l'espressione è scritta in somma di prodotti dove il simbolo "!" indica la negazione del letterale che lo segue.
- **full_simplofy**: ottimizza il circuito caricato, ma sarebbe meglio usare *write_eqn* prima di usare questo comando.

10.2 Ottimizzazione

Ottimizzare significa ridurre l'**area** o il **ritardo**.

- L'**area** di un circuito digitale è misurabile come il numero di porte logiche a 2 ingressi necessarie per la realizzazione del circuito
- Il **ritardo** di un circuito digitale è misurabile come il massimo numero di porte logiche che un segnale applicato agli ingressi deve attraversare per raggiungere l'uscita.
- L'**ottimizzazione** di un circuito digitale consiste nella trasformazione del circuito in uno *funzionalmente equivalente* avente area e/o ritardo minimi. Lo scopo è quello di ottenere circuiti piccoli e veloci.

Bisogna anche tenere conto del consumo energetico e della temperatura.

La minimizzazione di circuiti a 2 livelli avviene in 3 passo:

1. Si identificano tutti gli implicanti primi essenziali
2. Si identifica un insieme minimo di implicanti che coprano tutti i mintermini non coperti dagli implicanti primi essenziali
3. La funzione di copertura ottima è data dalla somma degli implicanti trovati ai punti 1 e 2

Per **circuiti a una uscita** esiste un metodo esatto (Mc Cluskey) che permette di trovare gli implicanti primi essenziali ed esiste sia un metodo esatto che approsimato anche per ottenere la funzione di copertura ottima.

Per **circuiti a più uscite** esiste solo un metodo *approssimato* per trovare la copertura ottima che si basa sul metodo esatto di identificazione degli implicanti primi essenziali di ogni singola uscita.

11 Modelli gerarchici

Un **componente gerarchico** è un componente del sistema che contiene al proprio interno dei sottocomponenti, ossia delle **istanze di altri componenti**. (Un sommatore di n bit può essere costruito da n sommatori).

Un componente si può includere in un modello *blif* con le seguenti istruzioni:

```
.subckt nomeComp paramFormale=paramAttuale // Crea l'istanza di un
  componente
.search nomeFileComp.blif // Include il file blif di un componente
```

Ad esempio, per includere un sommatore a 4 bit:

```
// sommatore.blif
.model SOMMATORE
.inputs A B CIN
.outputs O COUT

.names A B K
10 1
01 1

.names K CIN O
10 1
01 1

.names A B CIN COUT
11- 1
1-1 1
-11 1
.end
```

Si vuole includere il sommatore per creare un sommatore a 2 bit:

```
.model SOMMATORE2
.inputs A1 A0 B1 B0 CIN
.outputs O1 O0 COUT

.subsckt SOMMATORE A=A0 B=B0 CIN=CIN O=O0 COUT=C0
.subsckt SOMMATORE A=A1 B=B1 CIN=C0 O=O1 COUT=COUT
.search sommatore.blif
```

Ci sono dei componenti che possono essere utili, ad esempio:

- Generazione di un bit costante a 0:

```
.model zero1
.outputs uscita
.names uscita
.end
```

- Generazione di un bit a costante 1:

```
.model uno1
.outputs uscita
.names uscita
1
.end
```

12 Ottimizzazione approssimata multilivello

Consente al progettista di bilanciare area e ritardo di di un circuito con maggior grado di libertà rispetto alla minimizzazione a 2 livelli.

Tuttavia, non esistono tecniche esatte efficienti che portino alla realizzazione di configurazioni ottime usando la minimizzazione multi-livello; pertanto si ricorre a tecniche euristiche che garantiscono buone soluzioni in tempi di calcolo ragionevoli.

Alcuni comandi utili sono le seguentu:

- **sweep**: eliminazione dei nodi con un'unica linea di ingresso e di nodi con valore costante
- **eliminate**: eliminazione di un nodo interno alla rete. Si consideri che il nodo N rappresenti la funzione $y = (a + b) * c$, l'eliminazione di N prevede la sostituzione della variabile y in tutti i nodi che la utilizzano con l'espressione booleana $(a + b) * c$.
- **resub**: sostituzione di un nodo interno con un'insieme di nodi la cui funzionalità sia equivalente a quella del nodo sostituito. L'operazione viene effettuata per diminuire la complessità di un nodo
- **extract**: estrazione di una sottoespressione comune a più nodi che viene rappresentata con un nuovo nodo
- **simplify**: riduzione della complessità di ogni singolo nodo con algoritmo di Quine-McCluskey

13 Mapping tecnologico

La realizzazione di un circuito può usare due tecnologie fondamentali:

- **ASIC**: utilizzando le porte logiche di base messe a disposizione da chi si occupa di fondere il semiconduttore
- **FPGA**: utilizzando le porte logiche di base disponibili all'interno della board FPGA a disposizione

In entrambi i casi, le porte logiche di base sono contenute in una **libreria tecnologica** , la quale specifica anche le caratteristiche di **area e ritardo** delle porte logiche contenuta nella libreria.

I comandi principali sono:

- **read_library libreria**: carica la libreria tecnologica di nome "libreria". Le librerie sono specificate nel formato **genlib** (estensione .genlib, ad esempio synch.genlib e mcnc.genlib)
- **print_library**: visualizza informazioni inerenti alla libreria caricata
- **map**: esegue l'operazione di mapping. Le opzioni principali del comando sono:
  - **-m 0**: permette di ottenere un circuito ottimizzato rispetto all'area.
  - **-n 1**: permette di ottenere un circuito ottimizzato rispetto al ritardo.
  - **-s**: visualizza informazioni relative ad area e ritardo dopo il mapping:
    - **total gate area**: fornisce il valore dell'area come numero di celle standard della libreria tecnologica
    - **maximum arrival time**: indica il ritardo
- **write_blif -n**: mostra la rappresentazione del circuito associata alle porte della libreria
- **print_delay**: stampa informazioni relative al ritardo del circuito
- **reduce_depth**: riduce la lunghezza dei cammnini critici

14 Sintesi a N livelli

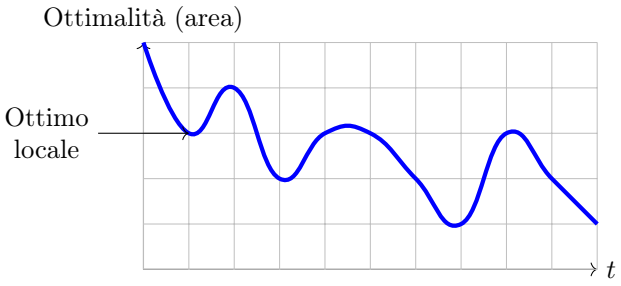


Figura 26: Ottimalità nel tempo

Definizioni utili 14.1
Il cammino critico è il percorso più lungo che il segnale deve effettuare e indica il ritardo.

In un circuito a 2 livelli riducendo l'area si riduce anche il ritardo.

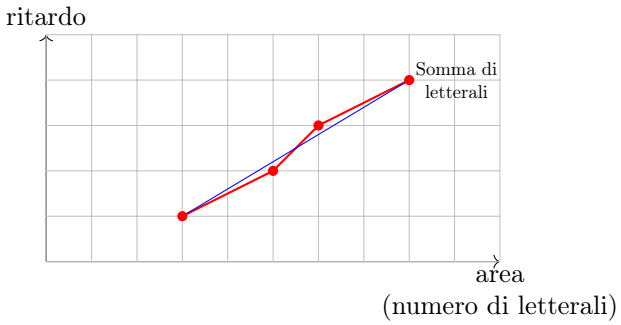


Figura 27: Ottimalità di circuiti a 2 livelli

In circuiti a N livelli il grafico del ritardo e dell'area è un'iperbole:

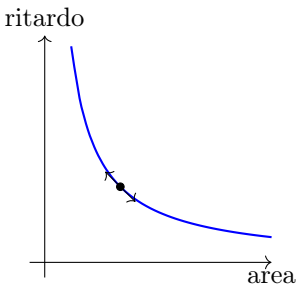


Figura 28: Curva di Pareto

Quindi bisogna trovare un buon compromesso tra area e ritardo, perchè diminuendo troppo l'area aumenta anche il ritardo.

14.1 Network

DAG Direct Acyclic Graph. Non permettono la creazione di cicli (Acyclic).

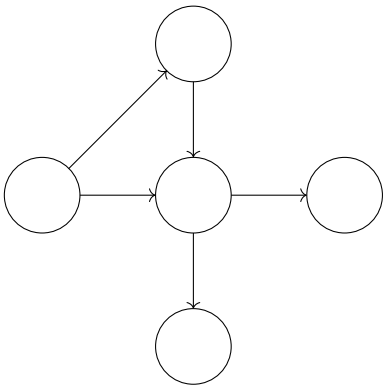


Figura 29: Esempio di DAG

Gli input sono dei nodi, e ogni nodo di ingresso avrà soltanto archi uscenti. I nodi di uscita avranno solo archi entranti. Il numero di letterali è dato dalla somma dei letterali di tutte le funzioni.

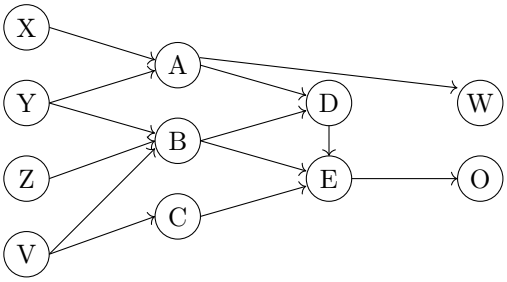


Figura 30: Esempio di network

Il numero di nodi può essere messo in relazione con il ritardo (più nodi ci sono più nodi devono attraversare i segnali, quindi più ritardo).

- Area: numero di letterali;
- Delay: numero di nodi;

14.2 Algoritmi

- **simplify**(Quine-McCluskey):trova la minimizzazione massima per ogni nodo del network
- **full_simplify**: è l'algoritmo più pesante per semplificare il circuito e utilizza BDB (Binady Decision Diagram) come struttura dati (struttura a grafo).

14.3 Script

In sis c'è una lista di **script** che permettono di minimizzare il circuito, ad esempio:

- **rugged**: permette di minimizzare il circuito;
- **sweep**: ripulisce il circuito dai nodi inutili
- **eliminate**: elimina un nodo dal circuito condensandolo nei nodi che lo circondano

14.3.1 full_simplify

Prendiamo come esempio il seguente network:

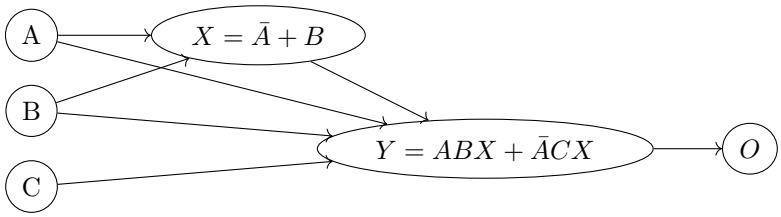


Figura 31: Esempio di network

Con un semplice **simplify** sul nodo X il circuito resta uguale. Si può notare dalla mappa di Karnaugh del nodo X che tutti i mintermini sono coperti e sono essenziali

AB / CX	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	1	1	1	0
10	0	0	0	0

Tabella 33: Mappa di Karnaugh del nodo X

Il **full_simplify**, invece calcola i **controllability don't care set**[2], che sarebbero le configurazioni di ingresso che non si possono presentare. ad esempio: ABX= (0,0,0), (0,1,0), (1,0,1), (1,1,0).
Se prendiamo in considerazione la tabella di verità del nodo X si può ottenere il controllability don't care set:

[2]Gli **observability don't care** sono valori di ingresso che non servono per calcolare l'uscita

A	B	X
0	0	1
0	1	1
1	0	0
1	1	1

Tabella 34: Tabella di verità del nodo X

A	B	C	X
0	0	-	0
0	1	-	0
1	0	-	1
1	1	-	0

Tabella 35: Controllability don't care set

Se si inseriscono questi don't care nella mappa di Karnaugh si osserva che sono disponibili altri raggruppamenti:

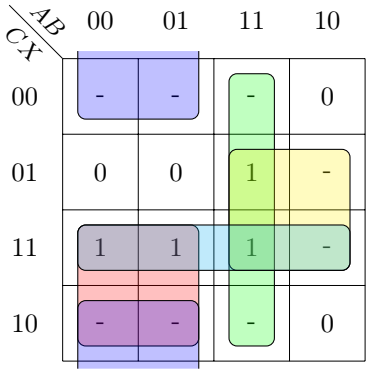
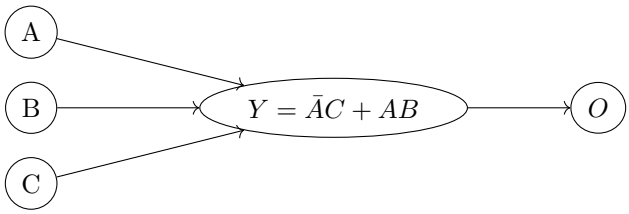


Tabella 36: Mappa di Karnaugh del nodo X con i don't care

Il risultato finale sarà:

$$Y = \bar{A}C + AB$$

Usando questi don´t care si può semplificare il circuito e si nota che il nodo X non è più necessario, quindi si può eliminare (sweep) e il network finale sarà il seguente:



Per poter realizzare il circuito minimizzato a 2 livelli bisogna fare il technology mapping che sarebbe l'ultimo passaggio per realizzare il network in qualcosa di reale.

- **FPGA**: $f(B^5) \to B$
- **ASIC**: libreria di porte logiche (composte da più porte elementari e date dal produttore)

L'algoritmo più famoso di technology mapping si chiama **tree mapping** e dopo l'applicazione di questo algoritmo si avrà una misurazione per area, ritardo e potenza.

15 Hardware design su FPGA con HDL

HDL è un linguaggio di modellazione che permette di progettare circuiti su hardware.

15.1 EDA (Electronic Design Automation)

La legge di Moore dice che ogni due anni il numero di transistor nei circuiti raddoppia.

Con gli anni aumenta anche l'astrazione, che però comporta anche perdita di dettagli.

Con verilog simuliamo il circuito (test bench) creando così un golden model, che sarà il modello di riferimento per realizzare il circuito con SIS. Nel caso in cui il modello fatto con sis si comporta come il bengh test, allora esso è corretto soltanto rispetto a quel bench test.

Si potrebbe scrivere un modello usando un linguaggio di programmazione comune, il problema è che i linguaggi di programmazinoe sono sequenziali e quindi eseguono le istruzioni in sequenza. Nei circuiti ogni porta logica continua a lavorare sui segnali che ha in ingresso, quindi non è sequenziale e per rappresentare l'hardware serve un linguaggio parallelo (i linguaggi di programmazione non sono fatti per questo). Per questo motivo c'è una netta differenza tra il tempo di simulazione e il tempo simulato (tempo reale). Un HDL che useremo è Verilog in cui tutto concorre (quindi in parallelo) e non ci sono istruzioni sequenziali.

15.2 Verilog

L'elemento principale da rappresentare è il modulo, che ha un nome e un elenco di porte di ingresso e di uscita. Le porte hanno 16 bit e sono messi in ordine dal 15 allo 0

```
module foo(clk, xi, yi, done); // clock, input, output
    input [15:0] xi,yi; // input a 16 bit
    output done;

    // always esegue il blocco di codice quando si verifica un evento
    sull'argomento
    always @(posedge clk) // serve per descrivere al livello RTL,
    posedge verifica quando il clock passa da 0 a 1
        begin:
            if (!done) begin
                if (x == y) cd <= x;
                else (x > y) x <= x - y;
            end
        end
endmodule
```

Ai dati si può assegnare un valore tra i seguenti:

- **0** bit falso
- **1** bit vero
- **X** dont'care o valore non definito
- **Z** stato di alta impedenza

In verilog ci sono diversi layer di astrazione:

- **Behavioral**: si descrive il circuito come un insieme di istruzioni
- **Data flow**: si descrive il circuito come un insieme di equazioni booleane
- **Gate level**: si istanziano le porte logiche

Inoltre si possono assegnare dei valori di partenza (initial statement) che vengono eseguiti una sola volta all'inizio della simulazione.

16 Circuiti sequenziali

È un circuito i cui valori di uscita non dipendono soltanto dai valori di ingresso correnti, ma anche da quelli precedenti. Questa la differenza dispetto a una rete combinatoria, che invece non tiene traccia della sequenza passata di input.

$$O_t = f(I_1, I_2, ..., I_k)$$

Definizioni utili 16.1
I circuiti combinatori sono un sottoinsieme dei circuiti sequenziali, per la precisione sono circuiti sequenziali con solo 1 stato.

Il sistema digitale **memorizza** gli ingressi fino a quel momento, cioè memorizza lo **stato** del sistema. Un esempio di circuito sequenziale è il seguente:

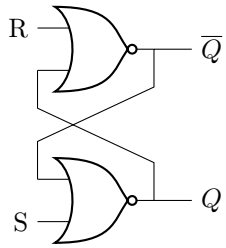


Figura 32: Latch SR

S R	Q_{t+1} $\bar{Q}_{t+1}$
1 0	1 0
0 1	0 1
0 0	Q_t $\bar{Q}_t$
1 1	non applicabile

Tabella 37: Tabella di verità del Latch SR

Se S e R valgono 0, allora il circuito mantiene lo stato precedente.

C'è bisogno di un "metronomo" che faccia scorrere il tempo, per i circuiti si utilizza un **clock** che è un segnale periodico che oscilla tra 0 e 1 e scandisce il tempo. Deve essere presente in tutti i sistemi digitali sequenziali.

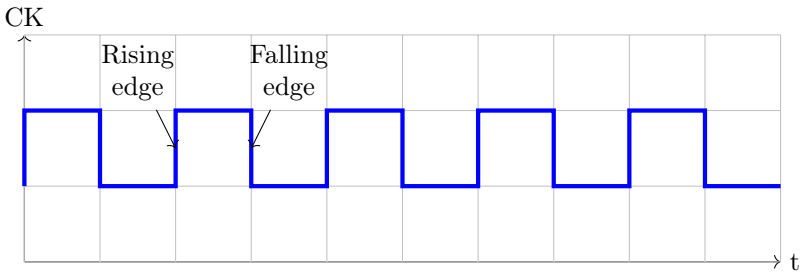


Figura 33: Clock

Di seguito c'è il circuito di un D-Latch che è un latch con un clock.

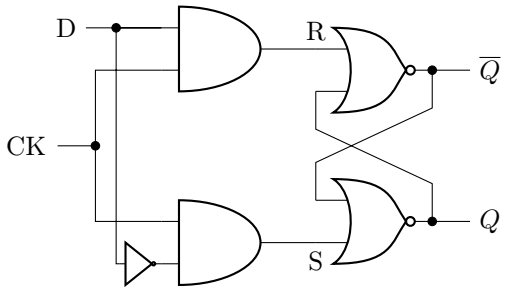


Figura 34: D Latch

D CK	Q_{t+1}
0 1	0
1 1	1
0 0	Q_t
1 0	Q_t

Si può scrivere anche:

D CK	Q_{t+1}
D 1	D
- 0	Q_t

Cioè, quando il clock è 1 il valore di Q è uguale a quello di D, mentre quando il clock è 0 il valore di Q è uguale a quello di Q_t.

Il grafico dei segnali è il seguente:

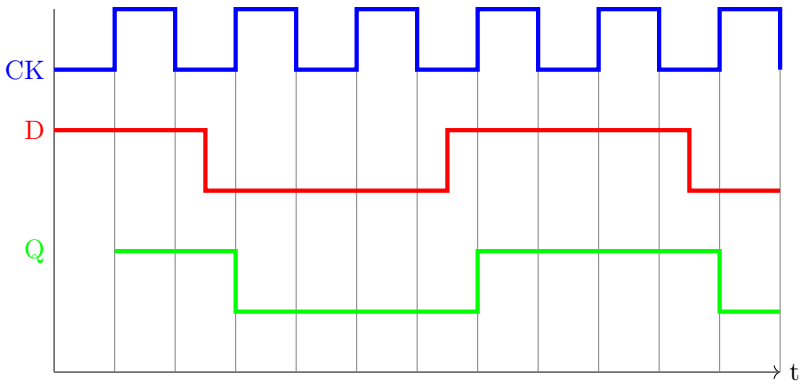


Figura 35: Grafico dei segnali del D Latch

Il circuito sincronizza il segnale di ingresso con quello del clock.

16.1 Astrazione

Per rendere più facile la realizzazione dei circuiti si astraggono le porte in componenti più complessi.

Alcuni esempi sono i seguenti:

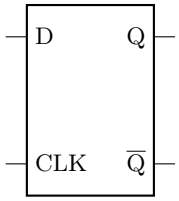


Figura 36: Simbolo del latch

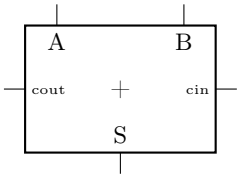


Figura 37: Simbolo dell'adder

Unendo 8 sommatori si ottiene un sommatore a 8 bit.

Oppure unendo N Latch si ottiene un registro a N bit:

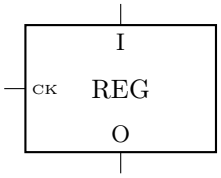


Figura 38: Simbolo del registro

Un altro esempio è la seguente tabella:

A B S	O
0 - 0	0
1 - 0	1
- 0 1	0
- 1 1	1

che assegna all'uscita il valore di A se S è 0, altrimenti assegna il valore di B. Questo circuito si chiama **multiplexer** (MUX) e si rappresenta col seguente circuito:

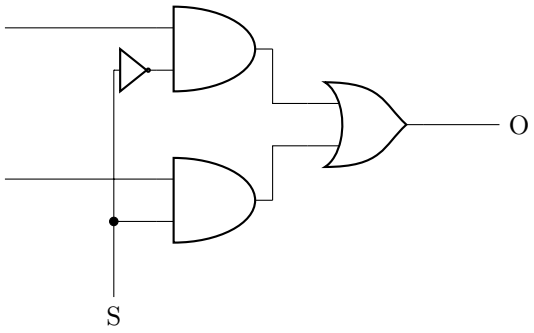


Figura 39: Circuito del multiplexer

Il simbolo del multiplexer è il seguente:

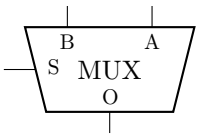


Figura 40: Simbolo del multiplexer

Esempio 16.1
Realizziamo con i precedenti componenti un contatore a modulo 1024.

Lo chiamiamo "count" e quando il reset $= 1$ *allora count* $= 0$*. Per contare 1024 numeri bisogna memorizzare 10 bit, quindi prendiamo un registro a 10 bit.*

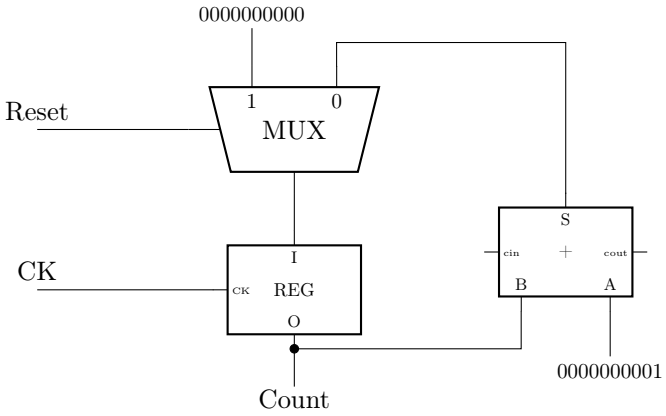


Figura 41: Contatore a modulo 1024

Il segnale del circuito è il seguente:

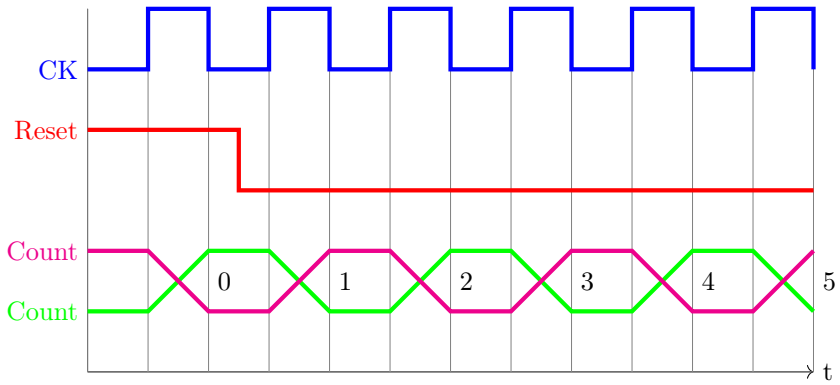


Figura 42: Grafico dei segnali del contatore

Non si può sapere l'uscita del circuito perchè ha un ritardo (dato dal cammino critico) che non permette al segnali di stabilizzarsi in tempo. Per questo motivo questo circuito non rappresenta un contatore, ma essenzialmente è un generatore casuale di numeri. Bisogna quindi creare un altro elemento di memoria.

Un altro elemento di memoria è il flip-flop D (oppure Latch Master-Slave):

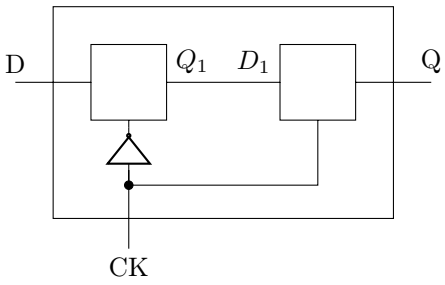


Figura 43: Flip-flop D

Il campionamento degli ingressi avviene sul fronte di salita del clock.

17 Macchine a stati finiti (FSM)

Per descrivere un qualsiasi comportamento sequenziale si usano come **modello** le FSM (Final State Machines). Tutte le macchine sensate avranno delle sequenze finite e conterranno l'evoluzione da stato presente a stato prossimo in base alla configurazione di ingresso e genereranno una configurazione di uscita. Esistono diversi tipi di SFM, ma in questo corso ci concentriamo solo sulle macchine **sincrone** e **deterministiche**.

- **Sincrone**: evolvono a ogni ciclo di clock (temporizzate)

- **Deterministiche**: dato un ingresso si ottiene sempre la stessa uscita

Una macchina a stati è una sestupla:

$$M =< S, I, O, \delta, \lambda, s >$$

- **S**: insieme, finito e non vuoto, degli stati
- **I**: alfabeto in ingresso, n bit in ingresso. $|I| = 2^n$
- **O**: alfabeto di uscita, m bit in uscita $|O| = 2^m$
- δ: funzione di stato prossimo, $\delta : S \times I \to S$
- λ: funzione di uscita, $\lambda : S \times I \to O$. Esistono più tipi di macchine in base alla funzione λ
  - **FSM di Mealy**: Genera l'uscita in base allo stato corrente e l'input corrente
    $$\lambda : S \times I \to O$$
  - **FMS di Moore**: Genera l'uscita in base allo stato corrente
    $$\lambda : S \to O$$

  I due tipi di macchine dal punto di vista dell'espressività sono equivalenti, quindi si può passare da una all'altra.
- **s**: stato iniziale (può non esserci), $s \in S$

17.1 Rappresentazione delle FSM

17.1.1 State Transition Table (STT)

Per ogni coppia S, I indica lo stato prossimo e l'uscita.

- Nelle **colonne**: simboli di ingresso
- Nelle **righe**: stato corrente
- Nelle **celle**:
  - **Mealy**: stato prossimo e uscita
  - **Moore**: stato prossimo

Esempio 17.1 (Mealy)

$$M =< \{A, B, C\}, \{0, 1\}, \{0, 1\}, \delta, \lambda >$$

	0	1
A	B/0	C/1
B	A/1	C/1
C	B/0	A/0

Tabella 38: State Transition Table di una FSM di Mealy

Esempio 17.2 (Moore)

$$M =< \{A, B, C\}, \{0, 1\}, \{0, 1\}, \delta, \lambda >$$

La funzione lambda viene scritta in una colonna separata: z

	0	1	z
A	B	C	0
B	A	C	1
C	B	A	1

Tabella 39: State Transition Table di una FSM di Moore

17.1.2 State Transition graph (STG)

Un grafo è un costrutto matematico preciso formato da una coppia $G =< V, E >$ dove:

- V è un insieme di nodi (vertex)
- E è un insieme di archi orientati (edges)

Un esempio di grafo è il seguente:

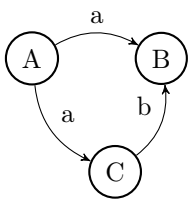


Figura 44: Esempio di STG

Esempio 17.3 (Mealy)

$$M =< \{A, B, C\}, \{0, 1\}, \{0, 1\}, \delta, \lambda >$$

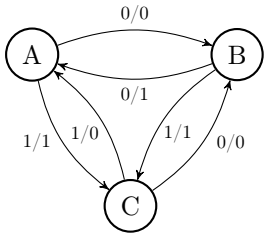


Figura 45: Esempio di STG con Mealy

Esempio 17.4 (Moore)

$$M =< \{A, B, C\}, \{0, 1\}, \{0, 1\}, \delta, \lambda >$$

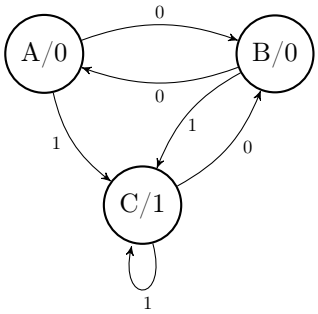


Figura 46: Esempio di STG con Moore

Esercizio 17.1
Progettare una macchina con 1 bit di ingresso x e 1 bit di uscita y. $y = 1$ se su x è stato letto un numero pari di 0, seguiti da un numero dispari di 1. y torna a 0 quando il numero di 1 diventa pari, oppure arriva uno 0.

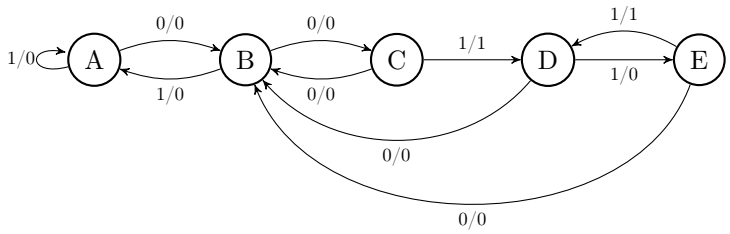


Figura 47: Esempio di STG con Moore

- ***A**: aspetto il primo 0 della sequenza*
- ***B**: ho letto una sequenza dispari di 0*
- ***C**: ho letto una sequenza di pari di 0*
- ***D**: ho letto una sequenza pari di 0 e una sequenza dispari di 1*
- ***E**: ho letto una sequenza pari di 0 e una sequenza pari di 1*

	0	1
A	B/0	A/0
B	C/0	A/0
C	B/0	D/1
D	B/0	E/0
E	B/0	D/1

17.2 Modello di Huffman

I circuiti sequenziali si realizzano fisicamente con il **modello di Huffman**. Questo modello divide il circuito in 2 parti:

- **Parte combinatoria**
- **Parte di registri**

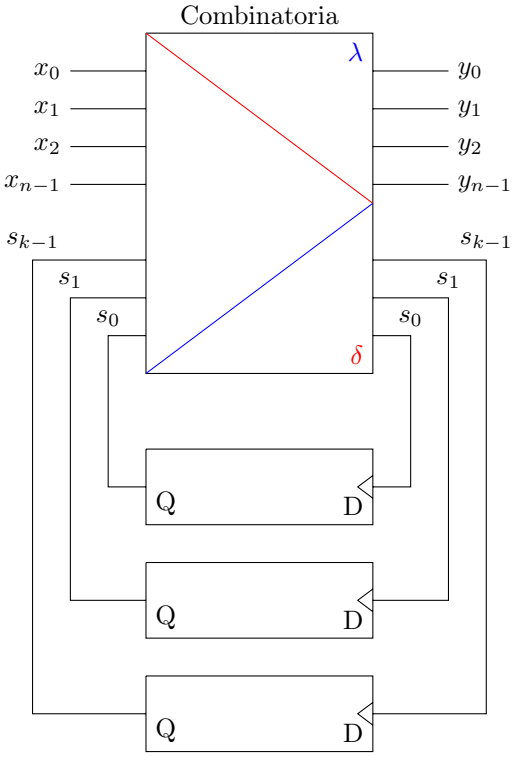


Figura 48: Modello di Huffman

17.3 Codifica degli stati

Prendiamo in considerazione la macchina a stati della figura 47.

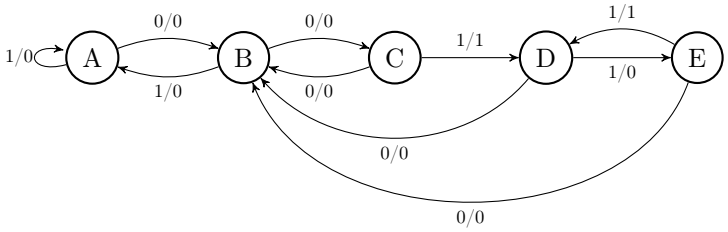


Figura 49: Esempio di STG con Moore

	0	1
A	B/0	A/0
B	C/0	A/0
C	B/0	D/1
D	B/0	E/0
E	B/0	D/1

I registri dovranno contenere la **codifica degli stati**. Come prima cosa bisogna capire quanti bit servono per rappresentare gli stati.

$$\sigma = |S|$$

Mi servono k bit dove:

$$k = log_2(\sigma) = log_2(5) = 3$$

Le seguenti **variabili di stato** si possono codificare con 3 bit:

- **A** = 000
- **B** = 001
- **C** = 010
- **D** = 011
- **E** = 100

Un metodo più "furbo" per codificare le variabili è assegnare ad ogni stato un codice che contenga soltanto un bit a 1. Questo metodo è chiamato **codifica one-hot**.

- **A** = 00001
- **B** = 00010
- **C** = 00100
- **D** = 01000
- **E** = 10000

Ora si mette insisme la parte combinatoria con la parte sequenziale. Si riscrive la tabella degli stati utilizzando la codifica a 3 bit:

$y_2\ y_1\ y_0$	$x = 0$	$x = 1$
0 0 0	001/0	000/0
0 0 1	010/0	000/0
0 1 0	001/0	011/1
0 1 1	001/0	100/0
1 0 0	001/0	011/1

La funzione δ è la seguente

$$\delta(y_2', y_1', y_0') = \delta(y_2 \wedge y_1 \wedge y_0 \wedge x)$$

E la tabella di verità della funzione è la seguente:

y_2 y_1 y_0 x	y_2' y_1' y_0'
0 0 0 0	0 0 1
0 0 0 1	0 0 0
0 0 1 0	0 1 0
0 0 1 1	0 0 0
0 1 0 0	0 0 1
0 1 0 1	0 1 1
0 1 1 0	0 0 1
0 1 1 1	1 0 0
1 0 0 0	0 0 1
1 0 0 1	0 1 1
1 0 1 0	- - -
1 0 1 1	- - -
1 1 0 0	- - -
1 1 0 1	- - -
1 1 1 0	- - -
1 1 1 1	- - -

La funzione λ è la seguente:

$$z = \lambda(y_2, y_1, y_0, x)$$

E la tabella di verità della funzione è la seguente:

y_2 y_1 y_0 x	z
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	0
0 1 0 0	0
0 1 0 1	1
0 1 1 0	0
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	-
1 0 1 1	-
1 1 0 0	-
1 1 0 1	-
1 1 1 0	-
1 1 1 1	-

Ora si può disegnare il modello di Huffman della funzione

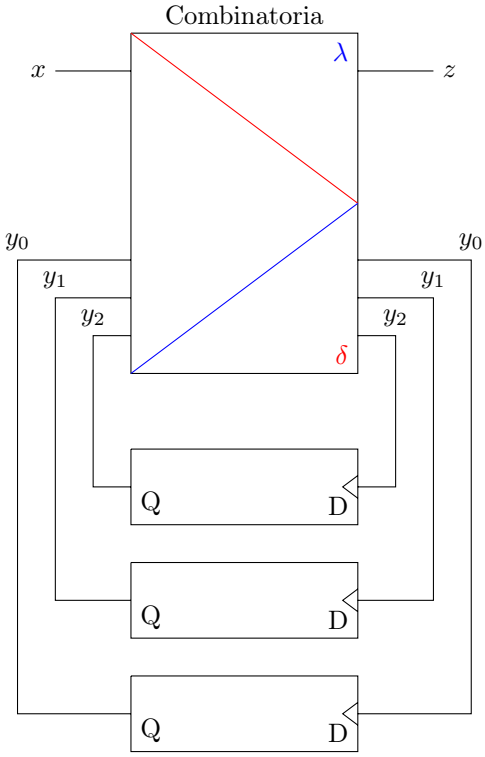


Figura 50: Modello di Huffman

18 Equivalenza tra macchina a stati

Le FSM si dividono in 2 tipi:

- **Completamente specificate**: per ogni coppia stato/ingresso è specificata la coppia stato prossimo/uscita
- **Parzialmente specificate**: per alcune coppie stato/ingresso non viene specificato lo stato prossimo **oppure** (anche entrambi) l'uscita

Prendiamo in considerazione 2 macchine:

$$M_1 =< S_1, I_1, O_1, \delta_1, \lambda_1 >$$

$$M_2 =< S_2, I_2, O_2, \delta_2, \lambda_2 >$$

Tali che $I_1 = I_2$ e $O_1 = O_2$, M_1 e M_2 sono equivalenti se e solo se per ogni stato $s_1^a \in S_1$ esiste uno stato $s_2^b \in S_2$ tale che ponendo la macchina M_1 nello stato s_1^a e la macchina M_2 nello stato s_2^b e applicando una qualsiasi sequenza di ingresso J_a le due sequenze di uscita prodotte sono identiche.