

Progettazione di Sistemi Digitali

Simone Lidonnici

6 aprile 2024

Indice

1	Numeri binari	3
1.1	Numeri esadecimali	3
1.2	Operazioni con numeri non decimali	4
1.2.1	Somme e sottrazioni	4
1.2.2	Moltiplicazioni	4
1.3	Numeri binari con il segno	5
1.3.1	Complemento a due	5
1.4	Estendere il numero di bit	5
1.4.1	Numeri senza segno	5
1.4.2	Complemento a due	6
1.4.3	Shiftare i numeri binari	6
1.5	Numeri binari con la virgola	6
1.6	Floating point	7
1.6.1	Operazioni con floating point	8
2	Porte logiche	9
2.1	Tipi di porte logiche	9
2.2	Circuiti logici	10
2.3	Equazioni booleane	11
2.3.1	Somma di prodotti (SOP)	11
2.3.2	Prodotto di somme (POS)	11
2.4	Completezza delle porte logiche	11
3	Algebra booleana	12
3.1	Assiomi e Teoremi	12
3.2	Semplificare un'equazione	12
4	Circuiti logici	14
4.1	Regole dei circuiti	14
4.1.1	Circuiti con più output	14
4.1.2	Contention	15
4.1.3	Floating	15
4.2	Mappe di Karnaugh (K-maps)	16
5	Blocchi combinatori	18
5.1	Multiplexer (Mux)	18
5.2	Decoder	18
5.3	Teorema di Shannon	19
5.4	Delay	20
6	Circuiti sequenziali	22
6.1	Circuito bistabile	22
6.2	SR (Set/Reset) Latch	22
6.3	D Latch	23
6.4	Flip-Flop	23

6.4.1	Flip-Flop attivabili	23
6.4.2	Flip-Flop resettabili	24
6.4.3	Flip-Flop settabili	24
6.5	Logica sequenziale sincrona	24
7	Finite State Machine (FSM)	25
7.1	Diagramma di trascrizione	26
7.1.1	Codifica degli stati di una FSM	27
7.2	Differenza tra Moore e Mealy	27
7.2.1	Passare da Moore a Mealy	28
7.2.2	Passare da Mealy a Moore	29
7.3	Design di una FSM	29
8	Timing	31

1

Numeri binari

I **numeri binari** sono numeri composti solo dalle cifre 0 e 1. Per convertirli in base decimale bisogna moltiplicare ogni cifra del numero binario per 2^i , in cui i è la posizione partendo da destra e contando da 0.

Esempio:

$$1010_2 = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 10_{10}$$

Con N cifre in binario possiamo scrivere 2^N valori con range $[0, 2^N - 1]$

1.1 Numeri esadecimali

I **numeri esadecimali** si utilizzano per comodità perché è molto facile convertire i numeri binari in esadecimali.

Per convertire i numeri binari in esadecimale basta raggrupparli in gruppi da 4 cifre e scriverli con il numero corrispondente da 0 a 15 (i numeri dal 10 al 15 sono rappresentati dalle lettere A,B,C,D,E e F).

Esempio:

$$10100110_2 \implies 1010 = A, 0110 = 6 \implies A6_{16}$$

Per convertirli in decimale si utilizza lo stesso principio dei numeri binari ma con 16 al posto di 2.

Esempio:

$$4AF = 15 \cdot 16^0 + 10 \cdot 16^1 + 4 \cdot 16^2 = 1199$$

Le cifre di un numero esadecimale convertite in binario e decimale:

Decimale	Binario	Esadecimale
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Ogni cifra di un numero binario viene chiamata **bit** e vengono raggruppati in gruppi da 8 chiamati **byte**. Il bit più a sinistra in un byte è quello più significativo e quello più a destra quello meno significativo.

1.2 Operazioni con numeri non decimali

1.2.1 Somme e sottrazioni

Le operazioni di somma e sottrazioni si effettuano in modo normale ma al posto di avere il riporto a 10 si ha a 16 o a 2.

Esempi:

$$1011 + 0011 = 1110$$

$$3A09 + 1B17 = 5520$$

Di solito i sistemi utilizzano un numero di bit fisso e se un numero eccede il numero di bit massimo viene chiamato **overflow** e i bit in eccesso vengono scartati.

Esempio:

$$1001 + 1100 = \textcolor{red}{1}0101 = 0101 \text{ sbagliato}$$

1.2.2 Moltiplicazioni

Per eseguire una moltiplicazione si moltiplica il primo numero per ogni bit del secondo spostandosi a sinistra di una posizione ogni volta.

Esempio:

$$\begin{array}{r}
 0101 \quad \times \\
 0111 \quad = \\
 \hline
 0101 \quad + \\
 01010 \quad + \\
 010100 \quad + \\
 000000 \quad = \\
 \hline
 100011
 \end{array}$$

1.3 Numeri binari con il segno

Per scrivere un numero con il segno in binario si utilizza il bit più a sinistra per identificare il segno:

- 0 significa positivo
- 1 significa negativo

Esempio:

$$1110 = -6$$

$$0110 = 6$$

Con N cifre in binario possiamo scrivere 2^{N-1} valori con range $[-(2^{N-1}) - 1, 2^{N-1} - 1]$

1.3.1 Complemento a due

Per poter eseguire facilmente addizioni i numeri binari con segno si scrivono utilizzando un altro metodo: il **complemento a due**.

Il primo bit (a sinistra) viene considerato negativo mentre gli altri positivi.

Esempi:

$$1001 = -1 \cdot 2^3 + 1 \cdot 2^0 = -7$$

$$01101 = 2^3 + 2^2 + 2^0 = 8 + 4 + 1 = 13$$

Per invertire (passare da positivo a negativo o viceversa) un numero in complemento a due bisogna invertire gli 0 con gli 1 e poi sommare 1.

Esempio:

$$27 = 011011$$

$$-27 = 100100 + 1 = 100101$$

Se vengono sommati due numeri con lo stesso segno potrebbe succedere che il risultato esca dal range a causa di un'overflow e bisogna aggiungere un bit.

Esempio:

$0110 + 01010 = 10001 \implies$ sbagliato perché dalla somma di due numeri positivi è uscito un numero negativo, quindi va aggiunto un bit uguale a 0 all'inizio $\implies 010001$

1.4 Estendere il numero di bit

1.4.1 Numeri senza segno

Se durante un'operazione (somma o moltiplicazione) il risultato eccede il numero di bit massimo e si causa un overflow bisogna estendere il numero di bit. Per farlo basta aggiungere tutti 0 davanti agli operandi.

Esempio:

$1010 + 1000 = 10010 \implies$ 1 non viene considerato quindi aggiungiamo uno 0 davanti ad entrambi gli operandi:

$$01010 + 01000 = 10010$$

1.4.2 Complemento a due

Per estendere il numero di bit di un numero in complemento a due si aggiunge a sinistra la prima cifra significativa ripetuta.

Esempi:

$$1010 = 111010$$

$$0111 = 000111$$

1.4.3 Shiftare i numeri binari

Se moltiplichiamo o dividiamo un numero binario per una potenza di 2 possiamo spostare le cifre a destra o sinistra di quanti posti quanto l'esponente di 2 nella potenza.

Esempi:

$$1 \cdot 2^2 = 100$$

$$1000 \cdot 2^2 = 100000$$

$$1010/2^3 = 1 \text{ (il resto non viene considerato visto che stiamo lavorando con numeri interi)}$$

$$10000/2 = 11000 \text{ (nei numeri negativi quando si divide si aggiungono degli 1 all'inizio)}$$

1.5 Numeri binari con la virgola

I numeri binari con la virgola si possono scrivere in diversi modi. Nel più classico semplicemente continuiamo a sommare potenze di 2 anche per la parte decimale, solamente con esponente negativo.

Esempi:

$$0.5_{10} = 2^{-1} = 0.1_2$$

$$3.75_{10} = 3 + 2^{-1} + 2^{-2} = 11.11_2$$

Un altro metodo è il **fixed point** in cui si decide preventivamente dove sia la virgola e semplicemente si scrive il numero in modo normale.

Esempio:

$$01101100(\text{virgola al 4 posto}) = 0110.1100 = 6.75$$

1.6 Floating point

Numeri binari in floating point

Un numero binario secondo la notazione **floating point** si scrive in modo simile alla notazione scientifica, cioè:

$$\pm M \cdot B^E$$

In cui:

- M = mantissa, numero con una sola cifra prima della virgola
- B = base
- E = esponente

La scrittura standard è a 32 bit di cui 1 per il segno, 8 per l'esponente e 27 per la mantissa. All'esponente bisogna aggiungere 127 (quindi per scrivere 5 dovremo scrivere 132) e nella mantissa non dobbiamo considerare il numero prima della virgola perché è sempre 1. Per convertire un numero binario in questa notazione bisogna:

1. Convertire il numero in binario senza segno
2. Scrivere il numero in notazione scientifica
3. Completare i 32 bit in modo opportuno

Vista la lunghezza di questi numeri, vengono solitamente scritti in esadecimale.

Esempio:

228 = 11100100

$$11100100 = 1.11001 \cdot 2^7 = \overbrace{0}^{\text{segno}} \overbrace{10000110}^{\text{esponente}+127} \overbrace{110010000000000000000000}^{\text{mantissa}} = 0x43640000$$

Nella notazione floating point esistono dei numeri speciali che si scrivono con dei bit specifici:

Numero	Segno	Esponente	Mantissa
0	indifferente	00000000	00...00
∞	0	11111111	00...00
$-\infty$	1	11111111	00...00

Oltre questi esistono una serie di numeri chiamati **numeri denormalizzati** che hanno esponente uguale a 0 e mantissa diversa da 0 e servono per scrivere numeri con esponente minore del minimo possibile. In questi numeri si considera la cifra prima della virgola come 0 e non 1. Con questa estensione in floating point abbiamo:

- Numero massimo: 2^{126}
- Numero minimo normalizzato: 2^{-126}
- Numero minimo denormalizzato: 2^{-149}

Del floating point esistono altre due versioni con diversi bit:

- Half-precision: 16 bit, 1 per il segno, 5 per l'esponente e 10 per la mantissa. L'esponente va inserito sommato di 15.
- Half-precision: 32 bit, 1 per il segno, 11 per l'esponente e 52 per la mantissa. L'esponente va inserito sommato di 1023.

1.6.1 Operazioni con floating point

Per eseguire la **somma** tra due numeri in floating point:

1. Convertirli in binario e scriverli in notazione scientifica
2. Cambiare l'esponente minore per renderli uguali
3. Sommare le mantisse
4. Riscrivere il risultato in notazione scientifica
5. Arrotondare se non bastano i bit
6. Riscrivere nel formato floating point

Esempio:

$0x3FC00000 + 0x40500000$

$0x3FC00000 = 001111111100 \dots 00 = 1.1 \cdot 2^0 = 0.11 \cdot 2^1$

$0x40500000 = 01000000010100 \dots 00 = 1.101 \cdot 2^1$

$0.11 \cdot 2^1 + 1.101 \cdot 2^1 = 10.011 \cdot 2^1 = 1.0011 \cdot 2^2$

$1.0011 \cdot 2^2 = 0100000010011000 \dots 00 = 0x40980000$

Per eseguire la **moltiplicazione** tra due numeri in floating point:

1. Convertirli in binario e scriverli in notazione scientifica
2. Sommare gli esponenti
3. Moltiplicare le mantisse
4. Riscrivere il risultato in notazione scientifica
5. Arrotondare se non bastano i bit
6. Riscrivere nel formato floating point

Esempio:

$(1.1 \cdot 2^{10}) \cdot (1.0110 \cdot 2^{11})$

$2^{10} + 2^{11} = 2^{21}$

$1.0110 \cdot 1.1 = 10.0001$

$10.001 \cdot 2^{21} = 1.0001 \cdot 2^{22} = 01001010100001000 \dots 00 = 0x4A840000$

2

Porte logiche

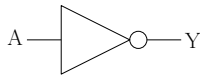
Le **porte logiche** sono delle funzioni che hanno un output e possono avere:

- 1 input: not, buffer
- 2 o più input: and, or, xor ...

2.1 Tipi di porte logiche

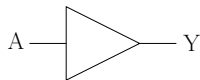
Ci sono molti tipi diversi di porte logiche:

NOT: $Y = \overline{A}$



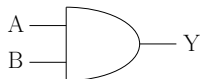
A	Y
0	1
1	0

BUFFER: $Y = A$



A	Y
0	0
1	1

AND: $Y = A \cdot B$



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

OR: $Y = A + B$



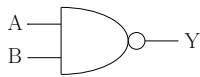
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR: $Y = A \oplus B$



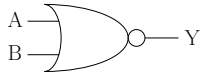
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NAND: $Y = \overline{A \cdot B}$



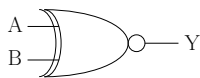
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR: $Y = \overline{A + B}$



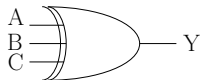
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XNOR: $Y = \overline{A \oplus B}$



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

XOR3: $Y = A \oplus B \oplus C$



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

2.2 Circuiti logici

Componenti di un circuito logico

Un **circuito logico** è composto da:

- nodi: input, output e nodi interni
- circuiti elementari

I circuiti sono di due tipi:

1. **Combinatori:** non hanno memoria e gli output sono definiti solo dagli input attuali
2. **Sequenziali:** hanno memoria e gli output sono definiti dagli input attuali e precedenti

I circuiti combinatori hanno 3 regole principali:

1. Tutti i circuiti elementari sono combinatori

2. Ogni nodo è un input o si collega ad esattamente un output
3. Non ci sono cicli

2.3 Equazioni booleane

Nelle equazioni booleane utilizzeremo delle definizioni precise:

- **Complemento:** opposto di una variabile (\overline{A})
- **Literal:** una variabile o un suo complemento (A o \overline{A})
- **Implicante:** prodotto di literal (AB o $B\overline{C}$)
- **Minterm:** prodotto che utilizza tutte le variabili di input (ABC o $\overline{A}B\overline{C}$)
- **Maxterm:** somma che utilizza tutte le variabili di input ($A + B + C$ o $\overline{A} + B + \overline{C}$)

2.3.1 Somma di prodotti (SOP)

Tutte le equazioni possono essere scritte come somma di prodotti.

Se abbiamo la tabella ci basta sommare i minterm che danno come risultato 1. Il risultato può anche essere scritto come sommatoria specificando l'indice dei minterm sommati.

Esempio:

A	B	Y	Minterm	Indice del minterm
0	0	0	$\overline{A}\overline{B}$	0
0	1	1	$\overline{A}B$	1
1	0	0	$A\overline{B}$	2
1	1	1	AB	3

In questo caso: $Y = \overline{A}B + AB = B$

Possiamo anche scrivere: $Y = \sum(1, 3)$

2.3.2 Prodotto di somme (POS)

Tutte le equazioni possono essere scritte come prodotto di somme.

Se abbiamo la tabella ci basta sommare i maxterm che danno come risultato 0. Il risultato può anche essere scritto come produttoria specificando l'indice dei minterm sommati.

Esempio:

A	B	Y	Maxterm	Indice del maxterm
0	0	0	$A + B$	0
0	1	1	$A + \overline{B}$	1
1	0	0	$\overline{A} + B$	2
1	1	1	$\overline{A} + \overline{B}$	3

In questo caso: $Y = (A + B) + (\overline{A} + B) = B$

Possiamo anche scrivere: $Y = \prod(0, 2)$

2.4 Completezza delle porte logiche

Le porte logiche **NAND** e **NOR** sono dette **funzionalmente complete** perché grazie a queste possono essere create tutte le altre porte logiche.

3

Algebra booleana

Dualità degli assiomi

Ogni assioma e teorema dell'algebra booleana può essere scritto sia con gli AND (\cdot) che con gli OR ($+$). Per farlo bisogna invertire i $+$ con i \cdot e gli 0 con gli 1. La versione alternativa dell'assioma o teorema viene chiamato **duale**.

3.1 Assiomi e Teoremi

L'algebra booleana ha diversi assiomi:

Assioma	Duale
$B = 0 \iff B \neq 1$	$B = 1 \iff B \neq 0$
$\overline{0} = 1$	$\overline{1} = 0$
$0 \cdot 0 = 0$	$1 + 1 = 1$
$1 \cdot 1 = 1$	$0 + 0 = 0$
$0 \cdot 1 = 1 \cdot 0 = 0$	$1 + 0 = 0 + 1 = 1$

Ci sono anche diversi teoremi:

Assioma	Duale
$B \cdot 1 = B$	$B + 0 = B$
$B \cdot 0 = 0$	$B + 1 = 1$
$B \cdot B = B$	$B + B = B$
$\overline{\overline{B}} = B$	
$B \cdot \overline{B} = 0$	$B + \overline{B} = 1$
$B \cdot C = C \cdot B$	$B + C = C + B$
$B \cdot C \cdot D = B(C \cdot D)$	$B + C + D = B + (C + D)$
$B(C + D) = (B \cdot C) + (B \cdot D)$	$B + (C \cdot D) = (B + C) \cdot (B + D)$
$B(B + C) = B$	$B + (B \cdot C) = B$
$(B \cdot \overline{C}) + (B \cdot C) = B$	$(B + \overline{C}) \cdot (B + C) = B$
$B(1 + C) = B$	$B + (0 \cdot C) = B$

3.2 Semplificare un'equazione

Preso un'equazione booleana, semplificarla significa ridurla al numero minimo di implicantii tramite teoremi o assiomi (detta anche **minimizzazione**). **Esempi:**

$$\overline{A}B + A = B + A$$

$$AB + A = A$$
$$\overline{AB} + AB = B$$

Teorema di De Morgan

Il **teorema di De Morgan** dice che se abbiamo un'equazione booleana con una somma o una moltiplicazione negata (\overline{AB} oppure $\overline{A + B}$), possiamo invertire l'operazione e negare i literal. Cioè:

$$\overline{AB} = \overline{A} + \overline{B}$$
$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

Esempio:

$$Y = \overline{(A + \overline{BD}) \cdot \overline{C}} = \overline{A + \overline{BD}} + \overline{\overline{C}} = \overline{A}BD + C$$

4

Circuiti logici

4.1 Regole dei circuiti

I circuiti logici, quando vengono disegnati, hanno diverse regole:

- Gli input vengono scritti a sinistra o in alto
- Gli output vengono scritti a destra o in basso
- Qualsiasi posta va da sinistra a destra

Inoltre vengono definite delle regole quando di intrecciano i fili:

- Se i fili formano una T sono collegati
- Se i fili formano una X con un punto sono collegati
- Se i fili formano una X senza punto non sono collegati



Nei primi due casi i fili sono collegati e nel terzo no

4.1.1 Circuiti con più output

Possono esserci circuiti che hanno più di un output, un esempio è il circuito a priorità, che ha questa tabella:

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

Molti valori in questa tabella non vengono considerati nel calcolare l'output quindi possiamo riscrivere la tabella in modo semplificato:

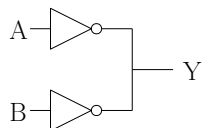
A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

X vuol dire **don't care**, cioè che il valore non cambia il risultato

4.1.2 Contention

La **contention** è una situazione che viene causata quando in un circuito si incontrano due valori diversi. Si indica con X (da non confondere con il don't care) e solitamente implica un problema.

Esempio:



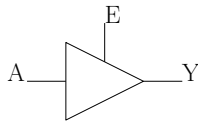
A	B	Y
0	0	1
0	1	X
1	0	X
1	1	0

4.1.3 Floating

Il **floating** è una situazione che viene causata quando un filo è disattivato o non c'è corrente, si indica con Z.

Un esempio è il **tristate buffer**, un buffer che è attivo solo quando un parametro E è uguale

a 1:



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

4.2 Mappe di Karnaugh (K-maps)

Le espressioni booleane possono essere semplificate in modo grafico utilizzando le **mappe di Karnaugh**. Questa mappa conterrà in ogni riquadro una riga della tabella della verità. Una volta che abbiamo scritto gli 0 e 1 nella tabella possiamo semplificarli in due modi:

- Utilizzando i minterm e gli 1 secondo le regole:
 1. Bisogna cerchiare ogni 1 almeno una volta
 2. Ogni cerchio deve contenere un numero di 1 pari ad una potenza di 2 (1,2 o 4)
 3. Ogni cerchio deve essere preso il più grande possibile
 4. Un cerchio può contenere don't care se servono a creare un cerchio più grande
 5. Alla fine ogni cerchio rappresenterà un prodotto tra le variabili cerchiare escludendo quelle che all'interno del cerchio appaiono sia negate che normali
- Utilizzando i maxterm e gli 0 secondo le regole:
 1. Bisogna cerchiare ogni 0 almeno una volta
 2. Ogni cerchio deve contenere un numero di 0 pari ad una potenza di 2 (1,2 o 4)
 3. Ogni cerchio deve essere preso il più grande possibile
 4. Un cerchio può contenere don't care se servono a creare un cerchio più grande
 5. Alla fine ogni cerchio rappresenterà una somma tra le variabili cerchiare escludendo quelle che all'interno del cerchio appaiono sia negate che normali

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

		AB			
		00	01	11	10
CD	00	1	0	0	1
	01	0	1	0	1
	11	1	1	0	0
	10	1	1	0	1

Si scrive 11 vicino a 01 perchè possono essere raggruppati solo riquadri che differiscono per solo una variabile. In questo caso avremo:

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

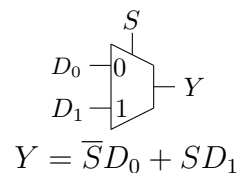
5

Blocchi combinatori

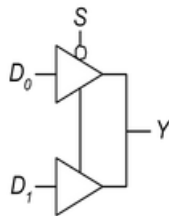
I blocchi combinatori sono un insieme di porte logiche che eseguono delle determinate operazioni specifiche, i due tipi più importanti sono **multiplexer (mux)** e **decoder**.

5.1 Multiplexer (Mux)

I multiplexer selezionano uno dei possibili N input (numero potenza di 2) tramite un numeri di selezionatori pari a $\log_2 N$. Anche il mux, come il NOR e il NAND permette di creare tutte le altre porte logiche. Viene disegnato:

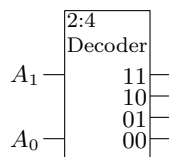


Se lo volessimo disegnare con le porte logiche classiche sarebbe:

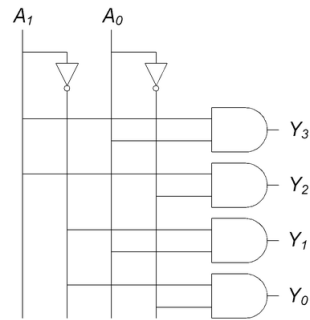


5.2 Decoder

I decoder hanno N input (numero potenza di 2) e un output per ogni combinazione possibile degli input, per un totale di 2^N output. Segue una politica **one-hot**, cioè che solo un output può essere attivo (uguale a 1) alla volta. Viene disegnato:



Se lo volessimo disegnare con le porte logiche classiche sarebbe:



I decoder possono essere usati per creare funzioni logiche partendo dai minterm.

5.3 Teorema di Shannon

Teorema di Shannon

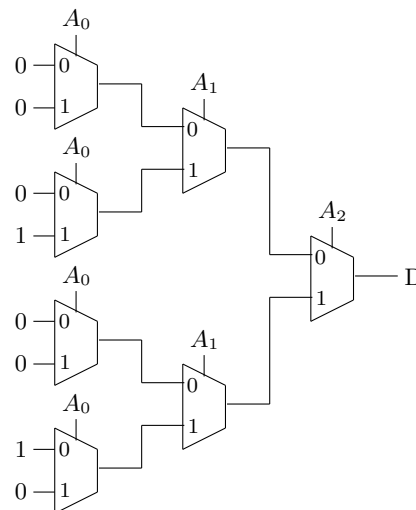
Data una funzione f con n variabili, questa può essere semplificata nella somma di due funzioni ad $n - 1$ variabili moltiplicate per i due possibili valori della variabile:

$$f(x_1, x_2, \dots, x_n) = x_1 \cdot f(0, x_2, \dots, x_n) + \overline{x_1} \cdot f(1, x_2, \dots, x_n)$$

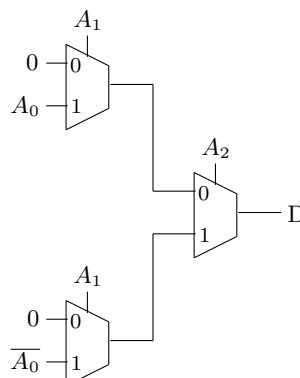
Questo teorema si può applicare ai mux per scrivere qualsiasi funzione.

Esempio:

A_2	A_1	A_0	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

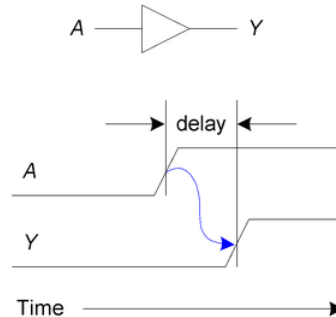


Semplificandolo otteniamo:



5.4 Delay

Il **delay** è il tempo da quando cambia un input fino a quando cambia l'output corrispondente. Viene calcolato da quando l'input è a metà del cambiamento fino a quando l'output è a metà del cambiamento.

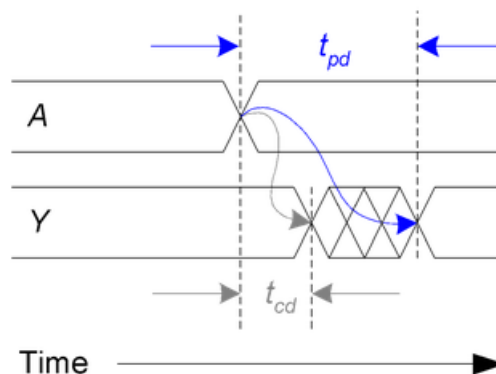


Il delay può essere causato da:

- capacità e resistenze del circuito
- limitazioni della velocità della luce

Ci sono due tipi di tempo che possiamo calcolare:

- **Propagation delay** (t_{pd}): tempo massimo per effettuare il cambiamento
- **Contamination delay** (t_{cd}): tempo minimo per effettuare il cambiamento

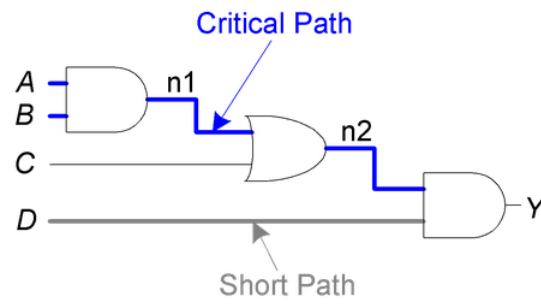


Il propagation delay si calcola sommando tutti i tempi massimi delle porte logiche del percorso più lungo mentre il delay minimo sommando tutti i tempi minimi delle porte logiche del percorso più corto.

I motivi per cui t_{pd} e t_{cd} sono diversi sono:

- diversi tempi di innalzamento o discesa
- multipli input e output, alcuni più veloci di altri
- rallentamenti dovuti alla temperatura

Esempio:



In questo caso:

- $t_{pd} = 2 \cdot t_{pd_AND} + t_{pd_OR}$
- $t_{cd} = t_{cd_AND}$

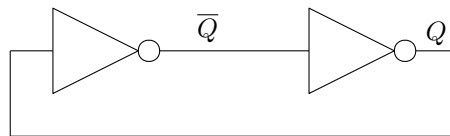
6

Circuiti sequenziali

I **circuiti sequenziali** sono circuiti che posseggono una memoria per immagazzinare i precedenti valori, quindi l'output dipende anche dai valori che ha in memoria. Rappresenta l'insieme dei valori passati racchiudendoli nello stato del sistema, cioè le informazioni su un circuito necessarie per spiegare i comportamenti futuri. Lo stato del sistema viene memorizzato in blocchetti chiamati **Latches** e **Flip-Flops**, che immagazzinano un bit dello stato.

6.1 Circuito bistabile

Il **circuito bistabile** è un tipo di circuito sequenziale con 2 output ma nessun input. Può immagazzinare un bit di stato.

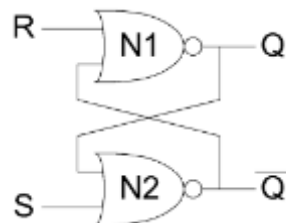


Se $Q = 0 \implies \overline{Q} = 1$

Se $Q = 1 \implies \overline{Q} = 0$

6.2 SR (Set/Reset) Latch

Il **SR Latch** è un circuito sequenziale che mantiene i valori di Q e \overline{Q} uguali finchè non vengono attivati degli input S o R che, rispettivamente, settano o resettano il valore di Q (e di conseguenza anche di \overline{Q}).



Questo circuito ha 4 casi possibili:

S	R	Q	\overline{Q}
0	0	Q precedente	\overline{Q} precedente
0	1	0	1
1	0	1	0
1	1	0	0

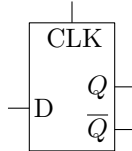
L'ultimo caso in verità non è possibile perchè $Q \neq \overline{Q}$ sempre.

6.3 D Latch

Il **D Latch** è un circuito sequenziale che ha 2 input, CLK che decide quando cambia l'output e D che è l'effettivo input. In questo modo si risolve il problema dell'SR latch e si può settare Q a 0 o 1 a piacimento.

Se $\text{CLK} = 1 \implies Q = D$

Se $\text{CLK} = 0 \implies Q$ rimane uguale

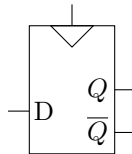


Questo circuito ha 4 casi possibili:

CLK	D	Q	\overline{Q}
0	X	Q precedente	\overline{Q} precedente
1	0	0	1
1	1	1	0

6.4 Flip-Flop

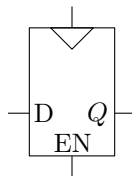
Il **flip-flop** memorizza il valore di D quando CLK passa da 0 a 1, ma non quando passa da 1 a 0.



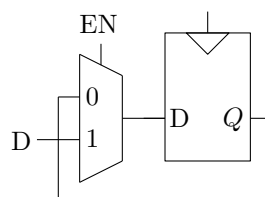
Ci sono diversi tipi di flip-flop.

6.4.1 Flip-Flop attivabili

I flip-flop attivabili sono una tipologia di flip-flop in cui si aggiunge un'altro input EN che permette di spegnere il flip-flop quando il valore di EN è uguale a 0.

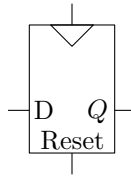


Il circuito interno è:



6.4.2 Flip-Flop resettabili

I flip-flop attivabili sono una tipologia di flip-flop in cui si aggiunge un'altro input R che permette di far diventare $Q = 0$ quando $R = 1$.

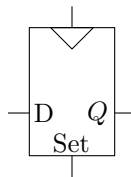


Ce ne sono due tipi:

- Sincrono: Q diventa 0 quando CLK passa da 0 a 1
- Asincrono: Q diventa 0 appena R diventa 1

6.4.3 Flip-Flop settabili

I flip-flop attivabili sono una tipologia di flip-flop in cui si aggiunge un'altro input S che permette di far diventare $Q = 1$ quando $S = 1$.



Ce ne sono due tipi:

- Sincrono: Q diventa 1 quando CLK passa da 0 a 1
- Asincrono: Q diventa 1 appena S diventa 1

6.5 Logica sequenziale sincrona

Nella logica sequenziale sincrona i cicli vengono interrotti da dei registri che contengono lo stato del sistema, lo stato cambia quando il clock passa da 0 a 1. Ci sono delle regole:

- Ogni circuito è un registro oppure un circuito combinatorio
- Almeno un elemento è un registro
- Tutti i registri hanno lo stesso clock
- Ogni circuito ha almeno un registro

7

Finite State Machine (FSM)

Una **macchina a stati finiti** è composta da:

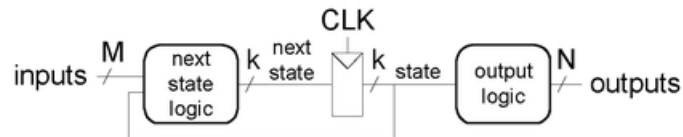
- Registri: memorizzano lo stato corrente e cambiano lo stato salvato memorizzato quando il clock si attiva (passa da 0 a 1)
- Logica combinatoria: calcola il prossimo stato e gli output

Nelle FSM lo stato successivo è definito da quello precedente.

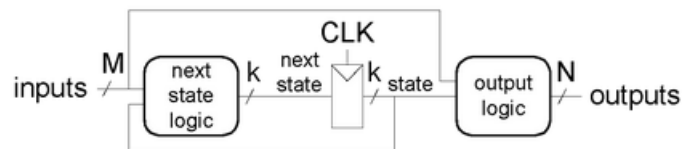
Le FSM sono di due tipi:

- Moore FSM: l'output dipende solo dallo stato corrente
- Mealy FSM: l'output dipende dallo stato corrente e dagli input

Moore FSM



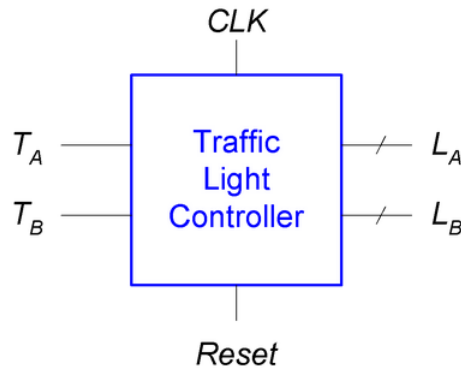
Mealy FSM



Esempio:

Controllo del traffico con:

- Sensori: $T_a, T_b = True$ quando c'è traffico
- Luci: L_a, L_b



7.1 Diagramma di trascrizione

Il **diagramma di trascrizione** è formato da stati, rappresentati con cerchi, e transizioni, rappresentate con archi. Dal diagramma di trascrizione si può scrivere la tabella di trascrizione, che dagli input e dallo stato corrente definisce il prossimo stato. Dopo si codificano gli stati con dei bit e si riscrive la tabella di trascrizione con gli stati codificati e infine si codificano gli output e si scrive la tabella degli output in base allo stato corrente.

Esempio:

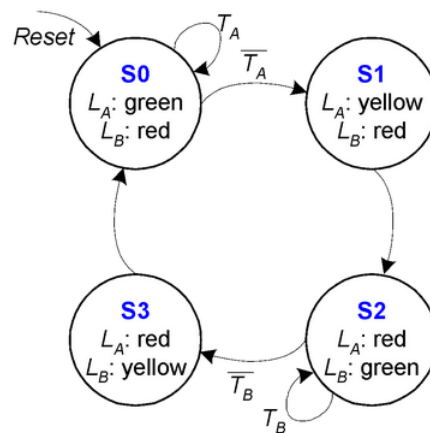


Tabella di trascrizione:

Stato attuale	Input		Prossimo stato
S	T_a	T_b	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

Tabella codificata:

Stato	Codifica	Stato attuale		Input		Prossimo stato	
		S_1	S_0	T_a	T_b	S'_1	S'_0
S0	00	0	0	0	X	0	1
S1	01	0	0	1	X	0	0
S2	10	0	1	X	X	1	0
S3	11	1	0	X	0	1	1
		1	0	X	1	1	0
		1	1	X	X	0	0

Tabella degli output:

Output	Codifica	Stato attuale		Output			
		S_1	S_0	L_{a_1}	L_{a_0}	L_{b_1}	b_0
green	00	0	0	0	0	1	0
yellow	01	0	1	0	1	1	0
red	10	1	0	1	0	0	0
		1	1	1	0	0	1

7.1.1 Codifica degli stati di una FSM

La codifica degli stati in una FSM può essere:

- Binaria: in cui ogni combinazione di bit rappresenta uno stato (es. 4 stati: 00, 01, 10, 11)
- One-hot: in cui solo un bit è uguale ad uno alla volta e serve un bit per ogni stato (es. 4 stati: 0001, 0010, 0100, 1000)

Stati irraggiungibili

Gli **stati irraggiungibili** in un FSM sono degli stati che non possono essere raggiunti e di solito si usano come stati di inizio e reset. Sono presenti in FSM con numero di stati non potenza di 2.

7.2 Differenza tra Moore e Mealy

La differenza principale tra le due FSM è che la Moore FSM ha bisogno di più stati di una Mealy FSM. Se l'output di una Mealy FSM fosse rallentata da un flip-flop, avrebbe gli output sincronizzati con una Moore FSM e questo evita output glitchati (sbalzi di valore veloci)

Esempio:

Una macchina che legge una stringa e dà in output 1 se legge una sequenza 01.

Moore:

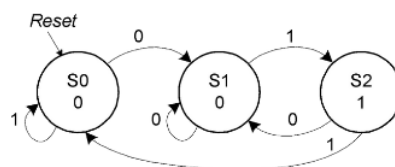


Tabella di trascrizione:

Stato	Codifica	Stato attuale		Input	Prossimo stato	
		S_1	S_0	I	S'_1	S'_0
S0	00	0	0	0	0	1
S1	01	0	0	1	0	0
S2	10	0	1	0	0	1
		0	1	1	1	0
		1	0	0	0	1
		1	0	1	0	0

$$S'_1 = S_0 I$$

$$S'_0 = \bar{I}$$

Tabella degli output:

Stato attuale		Output
S_1	S_0	Y
0	0	0
0	1	0
1	0	0

$$Y = S_1$$

Mealy:

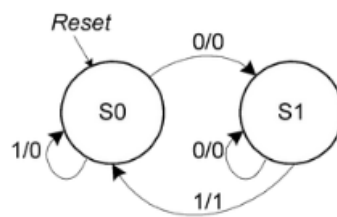


Tabella di trascrizione e output:

Stato	Codifica	Stato attuale	Input	Prossimo stato	Output
		S_0	I	S'_0	Y
S0	0	0	0	1	0
S1	1	0	1	0	0
		1	0	1	0
		1	1	0	1

$$Y = S_0 A$$

7.2.1 Passare da Moore a Mealy

Per passare da una FSM Moore a una FSM Mealy bisogna:

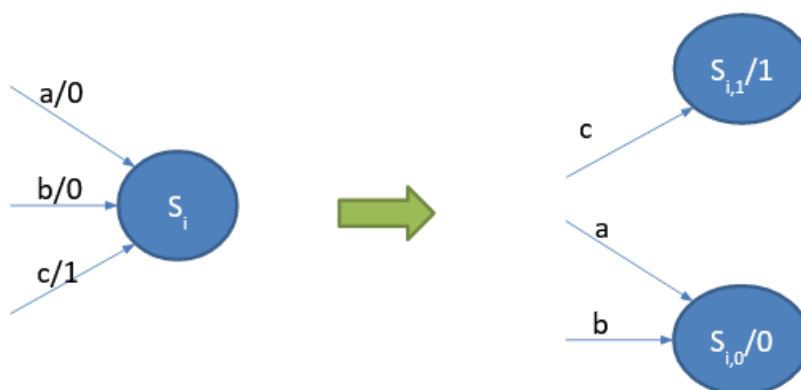
- Spostare gli output dagli stati agli archi
- Ogni arco entrante in uno stato deve avere l'output dello stato
- Se due stati hanno gli stessi archi uscenti possono essere uniti in un solo stato



7.2.2 Passare da Mealy a Moore

Per passare da una FSM Mealy a una FSM Moore bisogna:

- Spostare gli output dagli archi agli stati
- Se uno stato ha due archi entranti con output diversi bisogna dividerlo in due stati



7.3 Design di una FSM

Per fare il design di una FSM bisogna seguire dei passaggi:

1. Identificare input e output
2. Disegnare il diagramma di transizione
3. Scrivere la tabella di trascrizione
4. Codificare gli stati
5. Riscrivere la tabella di trascrizione:
 - 5.1 Moore: Riscrivere la tabella di trascrizione con gli stati codificati e la tabella degli output
 - 5.2 Mealy: Riscrivere la tabella di trascrizione e degli output con gli stati codificati
6. Scrivere l'equazione della **next state logic** e degli output

7. Disegnare il circuito

Invece per derivare una FSM da un circuito bidogna:

1. Esaminare circuiti, input, output e bit di stato
2. Scrivere le equazioni booleane degli stati e degli output
3. Creare la tabella della next state logic
4. Ridurre la tabella eliminando gli stati irraggiungibili
5. Assegnare a ogni combinazione di bit uno stato
6. Riscrivere la tabella con i nomi degli stati
7. Disegnare il diagramma di stato
8. Descrivere a parole la FSM

8

Timing