Progettazione di sistemi digitali

Corso del professore Pontarelli

Lugini Andrea

October 22, 2022

Contents

0.1 Legenda

- **a**: cifra
- **i**: posizione della cifra all'interno della stringa
- **N**: lunghezza della stringa

0.2 Potenze del due

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- $2^8 = 256$
- $2^9 = 512$
- $2^{10} = 1024$
- $2^{11} = 2048$
- $2^{12} = 4096$
- $2^{13} = 8192$
- $2^{14} = 16384$
- $2^{15} = 32768$
- $2^{16} = 65536$
- $2^{32} = 4294967296$
- $2^{64} = 18446744073709551616$

0.3 Binary numeric system

0.3.1 Il sistema binario

Il sistema binario è un sistema di rappresentazione dei numeri in base **2**, dove le uniche cifre disponibili sono **0** e **1** e sono identificati da un **pedice 2**.
Questo metodo è quello usato nei calcolatori digitali in quanto facilmente rappresentabile a livello elettronico (ON = 1, OFF = 0) tramite l'uso dei bit.
$R = [0,\ 2^N - 1]$
$\#R = 2^N$

0.3.2 Il sistema esadecimale

Sistema di rappresentazione in base 16, le cifre disponibili vanno da **0** a **F**, con le cifre da **A** fino a **F** che corrispondono ai valori decimali da **10** a **15**.
Sono identificati da un **pedice 16**.
Il valore di una cifra è dato dalla formula $valore = a_i * 2^i$.
N.B.: Ad una cifra esadecimale corrispondono 4 cifre binarie.

0.3.3 Conversioni fra binario, decimale ed esadecimale

- **B** $\Longrightarrow$ **D**: $\Sigma_{i=0}^{N-1} a_i * 2^i$.
- **H** $\Longrightarrow$ **D**: $\Sigma_{i=0}^{N-1} a_i * 16^i$.
- **D** $\Longrightarrow$ **B**: esistono 2 metodi, quello metodo della **divisione** ed il metodo della **sottrazione**.
- **D** $\Longrightarrow$ **H**: Uguale a D $\Longrightarrow$ B, ma usando 16 anzichè 2.
- **B** $\Longrightarrow$ **H**: Fare gruppi di 4 cifre (da ora in poi dette bit) alla volta, applicare il metodo B $\Longrightarrow$ D e convertire il valore decimale con la corrispondente cifra esadecimale
- **H** $\Longrightarrow$ **B**: Prendere il valore decimale corrispondente all singola cifra e convertirlo in binario col metodo B $\Longrightarrow$ D

0.3.4 Rappresentazione binaria di numeri con segno

Sign/Magnitude

Il **primo bit** è usato per memorizzare il segno (0 = +, 1 = -).
I restanti bit sono usati per memorizzare il valore assoluto del numero.
$R = [-(2^{N-1}) + 1,\ 2^{N-1} - 1]$
$\#R = 2^N - 1$
N.B.: Il numero 0 è memorizzato 2 volte, come +0 e -0.
Nonostante sia semplice rappresentare i numeri in questo formato, diventa molto

più complesso farci l'operazione binaria base, l'addizione, in quanto non segue lo stesso modello della classica addizione binaria. E' perciò raramente usato.

$$x = a_{N-1} * (-1) * \Sigma_{i=0}^{N-2} a_i * 2^i$$

Two's complement

Metodo effettivamente utilizzato per la rappresentazione dei numeri con segno. Il **MSB** rappresenta il valore $-(2^N - 1)$, i restanti bit rappresentano la "parte positiva" del numero.
$R = [-(2^{N-1}),\ 2^{N-1} - 1]$
$\#R = 2^N$

$$x = a_{N-1} * -(2^N - 1) + \Sigma_{i=0}^{N-2} a_i * 2^i$$

Con questo metoodo l'addizione si svolge come i classici numeri binari, prestando però attenzioni a errori di **overflow**. Se sommando due numeri **concordi** otteniamo un numero a loro **discordo** vuol dire che abbiamo avuto un errore di **out of range**, in quanto la somma di questi valori ci ha portato fuori dal range rappresentabile con i bit a nostra disposizione. L'operazione **taking the two's complement** è l'operazione di **inversione di segno** di un valore in two's complement.

- **Invertire** i bit
- **Sommare 1**

In two's complement a volte possono capitarci degli errori di overflow, ad esempio sommando un numero al suo inverso, dove possiamo però **trascurare il bit di overflow**. Dobbiamo però sempre prestare attenzione al **segno** del risultato per vedere se è corretto in base al segno degli operandi e l'operazione eseguita. Se, per esempio, moltiplicando un numero positivo ed uno negativo ci ritroviamo un risultato positivo sappiamo di essere incorsi in un errore.

0.3.5 Rimediare all'overflow

Entrambi i metodi si basano sull'estensione dei bit da N ad M, con M > N.

Sign-extension

Metodo usato per i numeri con rappresentazione **two's complement**, consiste nell'estendere il **MSB** in tutti i nuovi bit.

Zero-extension

Metodo usato per gli **unsigned**, consiste nell'estendere il valore **0** a tutti i nuovi bit.
Leggermente diverso per i numeri in formato **Sign/Magnitude**, dove il **MSB** viene lasciato con lo stesso valore pre-estensione.

0.3.6 Moltiplicazione

La moltiplicazione in binario per i numeri in **two's complement ed unsigned** funziona come la moltiplicazione posizionale decimale.

0.3.7 Shift

Caso particolare sono gli shift, dove si moltiplica per le potenze del **2**.
Anche qui bisogna prestare attenzione agli overflow ed errori di segno.

Left shift

$$A << N = A * 2^N$$

Basta aggiungere **N 0** in coda alla stringa binaria

Right shift

$$A >>> N = A * 2^{-N}$$

Per **two's complement** e **unsigned** basta aggiungere **N MSB** in testa alla stringa binaria.
Per i numeri **Sign/Magnitude** si aggiungono **N-1 0** e si lascia il **MSB** Uguale.

0.3.8 Frazioni e divisione

Il metodo di conversione da base decimale è il seguente:

1. Separare la parte intera da quella frazionaria
2. Convertire la parte intera
3. Moltiplicare la parte frazionaria per 2
4. Se $p < 1 \Longrightarrow 0$.
   Se $p \geq 1 \Longrightarrow 1, p = p - 1$.
5. Se $p \neq 0$ tornare al punto 4.

Diventa problematica però la rappresentazione dei numeri con la virgola.
Esistono 2 metodi.

0.3.9 Fixed Point

Si decide a priori quanti bit per la parte intera e quanti per la parte decimale.
Precisione $= 2^{-N}$.

0.3.10 Floating points

Notazione generica (anche detta scientifica): $\pm M * B^E$ Standardizzato da IEEE 754, il modello binario si basa sulla formula:

$$1, m * 2^{e-127}$$

Un floating point a 32 bit è memorizzato in memoria secondo il layout:

- 1 bit: segno
- 2-9 bit: esponente
- 10-32: mantissa

Poichè lavoriamo in sistema binario e il numero è memorizzato in forma **normalizzata**, sappiamo che l'1 prima della virgola è sempre presente e quindi **non memorizzato**.
Il metodo di conversione è il seguente:

1. Convertire il numero da decimale in binario
2. Rappresentarlo in notazine normalizzata, memorizzando di quanto abbiamo "spostato" la virgola
3. Settare il primo bit in base al segno del numero
4. Memorizzare nei bit per l'esponente lo "spostamento" +127

5. Memorizzare nei bit della mantissa il numero normalizzato ignorando l'**MSB**

Esistono poi dei casi speciali:

Valore rappresentato	Sign bit	Exponent bits	Mantissa bits
0	X	All 0	0
∞	0	All 1	0
$-\infty$	1	All 1	0
NaN	X	All 1	$\neq 0$
$(-1)^s * 2^{1-bias} * 0.m$	X	All 0	$\neq 0$

Esiste un motivo ben preciso per il quale nell'ultimo caso e = -126 e non -127. Con e = -127 avremmo un gap molto grande fra il numero più grande denormalizzato ed il numero più piccolo normalizzato, mentre con e = -126 il suddetto problema non si presenta in quanto il gap tra i valori denormalizzati e e il più grande denormalizzato e il più piccolo denormalizzato è uguale.

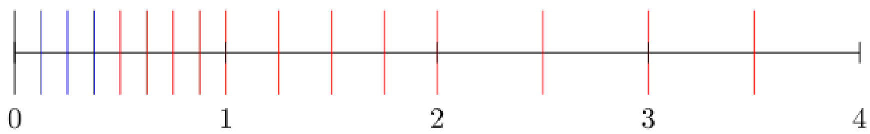


Figure 1: e = -126

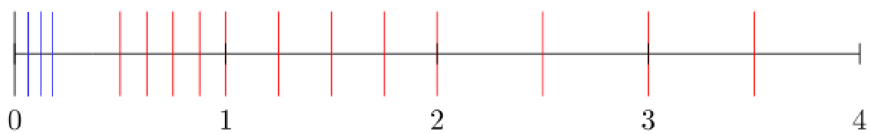


Figure 2: e = -127. Da notare il gap tra blu e rosso

Nella rappresentazione esadecimale di un numero floating point, possiamo dedurre il segno se il valore della prima cifra esadecimale è > 0x8.

Calcolare minimo e massimo

Potrebbe essere utile saper calcolare il minimo ed il massimo valore rappresentabile in un numero floating point, ignorando ovviamente 0 e numeri negativi.
Definiamo C = bit usati per la mantissa ed E = bit usati per esponente

$$M = (2.0 - 2^{-C}) * 2^{2^{E}-2-bias}$$

$$m = (2.0 - 1.0) * 2^{1-bias} = 2^{1-bias}$$
$$M_D = (1 - 2^{-C}) * 2^{1-bias}$$
$$m_D = (2^{-C}) * 2^{1-bias}$$

Per 32 bit:

- $M \approx 3.4 * 10^3 8$
- $m \approx 1.2 * 10^- 38$
- $M_D \approx 1.2 * 10^- 38$
- $m_D \approx 1.4 * 10^- 45$

Half precision Floating Point

- **16 bit**: 1 segno, 5 esponente, 10 mantissa.
- Bias: 15.
- Denormalized: $(-1)^s * 2^{-14} * 0.m$.
- $M = 65504$
- $m \approx 6.1 * 10^{-5}$
- $M_D \approx 6.1 * 10^{-5}$
- $m_D \approx 5.96 * 10^{-8}$

Double precision Floating Point

- **64 bit**: 1 segno, 11 esponente, 53 mantissa
- Bias: 1023
- Denormalized: $(-1)^s * 2^{-1022} * 0.m$.
- $M \approx 1.79 * 10^{308}$
- $m \approx 2.2 * 10^{-308}$
- $M_D \approx 2.2 * 10^{-308}$
- $m_D \approx 2.47 * 10^{-324}$

Approssimazioni dei Floating Points

- Problemi: **overflow** o **underflow**
- Soluzioni:
- **Toward zero**: Ignoriamo i bit dopo quelli disponibili.
  1.01001, 3 bit decimali disponibili: 1.010.
- **To nearest**: Verso il numero più vicino.
  1.100101 (1.578125), 3 bit decimali, 1.101
- **Down**: per difetto
- **Up**: per eccesso

Addizione

1. Prendere la mantisse e considerarle con la cifre sottintese
2. Shiftare la mantissa con l'esponente minore
3. Sommare le mantisse e normalizzare
4. Calcolare l'esponente corretto
5. Approssimare la mantissa

Moltiplicazione

1. Sommare esponenti
2. Prodotto mantisse
3. Normalizzare

0.4 Algebra booleana

0.4.1 Definzioni

- $\overline{X}$: **Complemento** negato di X
- X o $\overline{X}$: **Literal**
- **Implicant**: prodotto di literal
- **Minterm**: prodotto di tutte le variabili d'input
- **Maxterm**: somma di tute le variabili d'input

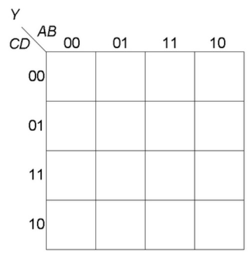
0.4.2 Dualità dell'algebra booleana

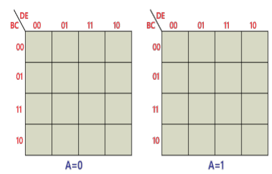
Identità	$B * 1 = B$	$B + 0 = B$
Null element	$B * 0 = 0$	$B + 1 = 1$
Idempotenza	$B * B = B$	$B + B = B$
Involuzione	$\overline{\overline{B}} = B$	$\overline{\overline{B}} = B$
Complemento	$B\overline{B} = 0$	$B + \overline{B} = 1$
Commutativa	$BC = CB$	$B + C = C + B$
Associativa	$(BC)\,D = B\,(CD)$	$(B + C) + D = B + (C + D)$
Distributiva	$B\,(C + D) = BC + BD$	$B + (CD) = (B + C)\,(B + D)$
1 teorema assorbimento	$B\,(B + C) = B$	$B + (BC) = B$
2 teorema assorbimento	$\left(B\overline{C}\right) + (BC) = B$	$\left(B + \overline{C}\right)(B + C) = B$
Teorema della ridondanza	$\left(B\overline{C}\right) + C = B + C$	$\left(B + \overline{C}\right) C = BC$
Teorema del consenso	$(BC) + \left(\overline{B}D\right) + (CD) = (BC) + \left(\overline{B}D\right)$	$(B + C)\left(\overline{B} + D\right)(C + D) = (B + C)\left(\overline{B} + D\right)$
De Morgan	$\overline{B_0 * B_1 * ...} = \overline{B_0} + \overline{B_1} + ...$	$\overline{B_0 + B_1 + ...} = \overline{B_0} * \overline{B_1} + ...$

I teoremi sono dimostrabili per: **induzione perfetta**, verificando la veridicita per tutti i possibili casi, oppure usando gli assiomi e gli altri teoremi per rendere il termine di sinistra uguale a quello di destra

0.4.3 Mappe di Karnaugh

Le k-maps sono uno strumento grafico di minimazzione di espressioni booleane, usate quando si hanno fino a 5 varabili (dopo diventa particolarmente scomodo). Consiste nel creare una tabella, fino a 4 variabili, o 2 tabelle, per 5 variabili, contenti i possibili stati delle variabili della nostra espressione.





Regole delle k-maps

- Ogni casella corrisponde ad un mintermine
- Ogni 1 deve essere cerchiato almeno una volta
- Ogni cerchio deve selezionare 2^{α} 1
- Ogni cerchio deve essere il più grande possibile
- Un cerchio può andare da angoli opposti ad angoli opposti
- I **don't care** vengono presi solo se aiutano a minimizzare l'equazione
- Il cerchio più grande è detto **Prime Implicant**

Queste sono le regole per ottenere una SOP, per la POS basta prendere gli 0 ed i maxtermini.

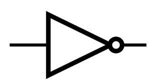
0.5 Circuiti logici

0.5.1 Porte logiche

Componente che svolge una funzione logica elementare.
Può avere da 1 a n input ed 1 output.

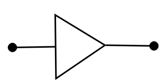
NOT

Nega il valore in entrata



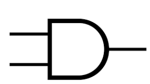
BUFFER

Usato per pulire il segnale



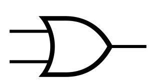
AND

Prodotto logico



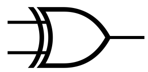
OR

Somma logica



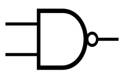
XOR

Somma logica esclusiva



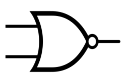
NAND

AND seguito da NOT



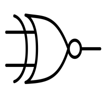
NOR

OR seguito da NOT



XNOR (Anche noto come EQ)

XOR seguito da NOT



0.5.2 Circuiti logici combinatori

- **Memoryless**
- No percorsi **ciclici**
- Ogni nodo è un input o si collega a un **solo** output
- Out = f(Input)
- Composti da sottocircuiti combinatori

0.5.3 Circuito logico sequenziale

- Ha un **internal state**
- Out, State = f(Input, State)

0.5.4 Descrivere circuito come operazioni logiche

Sum-of-products form

1. Scrivere tabella verità
2. Scrivere i **minterm** m che, presi gli input A e B delle proprie corrispondenti righe, restituisce **1**
3. Scegliere i minterm delle righe con $y = 1$ e nominarli progressivamente k_i
4. $y = \Sigma_{i=0}^{\#K} k_i$

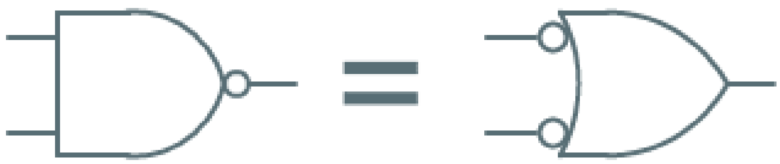
Product-of-sums form

1. Scrivere tabella verità
2. Scrivere i **maxterm** M che, presi gli input A e B delle proprie corrispondenti righe, restituisce **0**
3. Scegliere i maxterm delle righe con $y = 0$ e nominarli progressivamente k_i
4. $y = \Pi_{i=0} \# K k_i$

0.5.5 Bubble pushing

De Morgan ci permette di vedere, a livello circuitale, la correttezza del **bubble pushing**, ovvero l'uguaglianza fra una porta logica e la sua porta "opposta" con input ed output negati rispetto l'originale.

2-input NOR



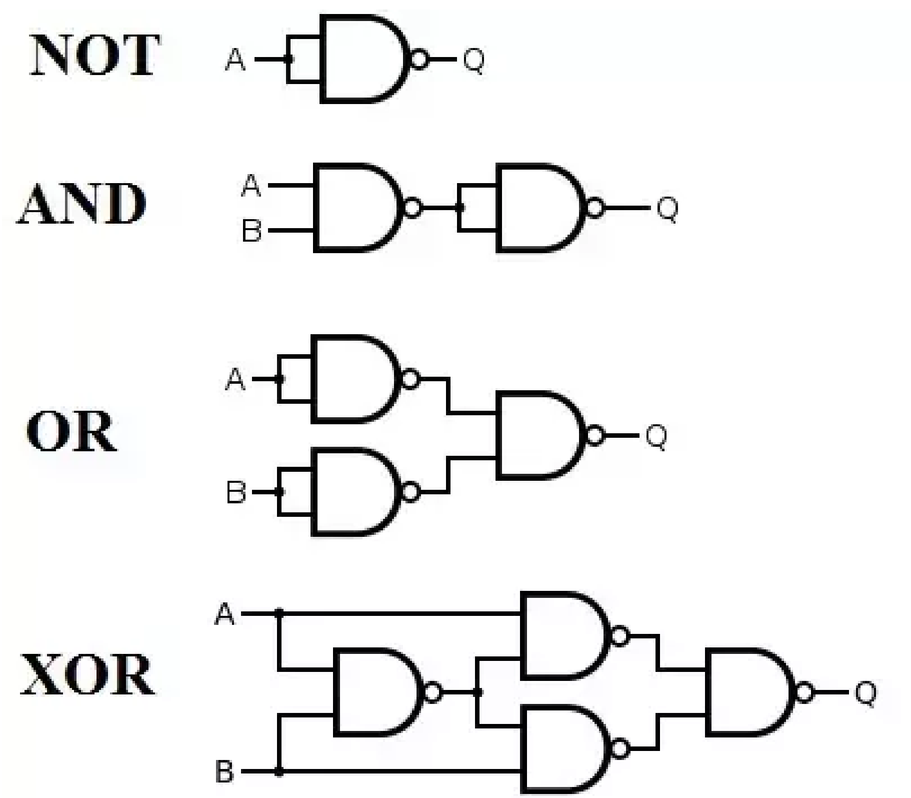
2-input NAND

2-input AND

2-input OR

0.5.6 NAND e NOR

Universalità



Convenienza

Conviene usare NAND e NOR piuttosto che OR ed AND in quanto sono realizzabili con meno transistor

0.5.7 Non associatività

Suddette porte logiche non godono della proprietà associativa

0.5.8 Multiple output circuits

I circuiti possono avere anche molteplici output.
In questo caso risulta facili pensare ad ogni output come ad un circuito a 2 livelli a parte, ma nella realtà non è fatto così in quanto nella maggior parte dei casi porterebbe all'uso di un numero maggiore di porte logiche e a circuiti più costosi e più lenti.

0.5.9 Codifica one-hot

La codifica one-hot è un tipo di codifica usata nei ciruiti a N input e/o N output dove solo uno dei bit ha il valore **1**

0.5.10 Don't cares

Segnalati nelle tabelle di verità e nelle k-maps con una **"X"**, indicano valori per i quali non ci interessa sapere se sono 1 o 0.

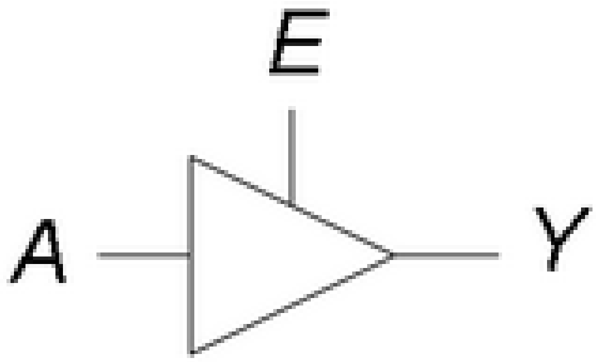
0.5.11 Contention

Una contention accade quando il circuito tenta di avere sia 0 che 1 come output.
Questa condizione indica spesso un bug nel circuito e viene indicato col simbolo **"X"**.

0.5.12 Floating

Un nodo flottante può avere come output 0, 1 od un valore nel mezzo, indicato col simbolo **"Z"**.
In questa condizione si può considerare il nodo come un **interruttore aperto**.
Questa proprietà è sfruttata per realizzare i **tristate buffers**.

Tristate Buffer



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

0.5.13 Tristate busses

I tristate buffer sono usati per la costruzione dei **tristate busses**, con la promessa che solo un buffer alla volta sia attivo