Elettronica digitale: cenni

Non solo analogico

La gestione di informazione prevede tipicamente fasi di elaborazione, in cui occorre calcolare funzioni ("qual è la somma di questi due valori?"), eseguire confronti ("questi due valori sono uguali?"), …

Se si impiegano segnali analogici l'informazione viene tipicamente codificata direttamente nell'ampiezza dei segnali, così che, p.es., per stabilire se due valori sono uguali si devono confrontare le ampiezze dei rispettivi segnali

Impiegando della circuiteria analogica, per ogni tipo di operazione si realizza un circuito o componente specifico

Questo va molto bene per certe operazioni (somme come somme di tensioni, integrali come accumulo di quantità di carica su condensatori, ...), ma è assai critico per altre operazioni (banalmente: ripeti n volte un'operazione, dove n non è fissato a priori)

Oltre a ciò, la presenza di rumore rende problematiche le modalità di interpretazione (cioè di decodifica) dei segnali (p.es. che succede a un dato di bilancio codificato come una tensione elettrica se la tensione varia anche solo di poco?)

Dal discreto al digitale (al binario)

Questi problemi si possono risolvere solo occasionalmente mantenendo una codifica analogica e discretizzando l'insieme delle ampiezze ammissibili (p.es. usando le ampiezze 1, 2, …, 10 V per codificare i numeri interi da 1 a 10: digitale *non* è sinonimo di discreto!)

Da un segnale continuo si giunge a uno, corrispondente, digitale attraverso:

* discretizzazione nel tempo ← *campionamento*

per ogni campione ottenuto:

* discretizzazione nell'ampiezza
* codifica secondo un codice non-analogico

} ← *quantizzazione*

Spesso, ma non necessariamente, il codice è basato sulla combinazione di due simboli ("codice binario"), realizzati fisicamente come:

* aperto / chiuso di rélé
* tensione bassa / alta ai capi di un resistore di carico
* caricato / non caricato di un condensatore
* …

In ogni caso, il digitale è basato su una logica *fortemente* non lineare

Caratteristiche fisiche dei "componenti digitali"

* Range di temperature di funzionamento
* Valore nominale della tensione di alimentazione (5V, p.es.) e range di tensioni per i due livelli (2.0-5.5 e 0.0-1.0, p.es.) (ovviamente aumentando la tensione di alimentazione può aumentare la distanza tra i range che realizzano fisicamente i due livelli, e quindi l'immunità al rumore)
* Tempo di propagazione / commutazione (10 ns, p.es.)
* Dissipazione di potenza (10 mW, p.es.) (poiché P=VI, idealmente P=0 …: nei xMOS, analoghi a condensatori, le correnti di fuga sono estremamente ridotte, e quindi l'assorbimento di potenza è effettivamente ridotto)
* Capacità di fan-out, cioè numero di componenti pilotabili senza dover ricorrere a un amplificazione (10, p.es.)

Queste caratteristiche dipendono dalle modalità di realizzazione fisica dei componenti

Caratteristiche logiche dei "componenti digitali"

Al di là delle modalità di realizzazione fisica, ogni componente a m ingressi e n uscite viene caratterizzato come un sistema che riceve in ingresso m simboli 0/1 e produce in uscita, istantaneamente, n simboli 0/1

Dunque il comportamento di un componente è descritto completamente dalla tabella (chiamata "tavola della verità"; nome sfortunato: la verità con questi argomenti non c'entra proprio ...) che a ognuna delle 2^m m-uple di input associa una n-upla di output

Per esempio (m=2; n=2):

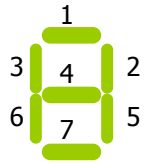
A	B	f(A,B)
0	0	0 0
0	1	0 1
1	0	0 1
1	1	1 0

(l'output è 1 solo se entrambi gli input sono 1)

... e poiché un componente a n output è pensabile come un insieme parallelo di n componenti, che producono ognuno un simbolo dell'output a partire dallo stesso insieme di input, ogni componente è ottenibile da componenti elementari (detti anche *porte logiche* (in inglese *gate*) che producono in output un simbolo

Un esempio

Pilotare un display a 7 segmenti: 4 simboli di input e 7 di output



Per cui, per esempio:

A_3	A_2	A_1	A_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	0	0	0	1	1	1	0	1	1	1
0	0	0	1	0	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...

Ogni segmento del display è pilotato da un gate a 4 ingressi

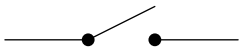
Dunque il problema diventa (per O_1, per esempio):

A_3	A_2	A_1	A_0	O_1
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	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1

Gate

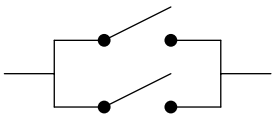
NOT: inverter

A	not A
0	1
1	0



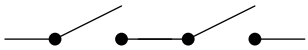
OR: somma booleana

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1



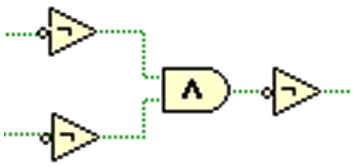
AND: prodotto booleano

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1



Equivalenze …

Per esempio:



equivale a

Infatti:

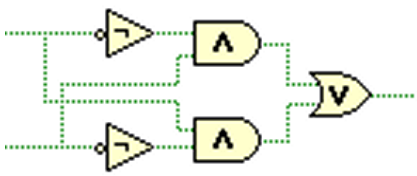
A	B	-A	-B	-A∧-B	-(-A∧-B)
0	0	1	1	1	0
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	1

Dunque ("leggi di De Morgan"): $-(-A \wedge -B) \equiv A \vee B$ e anche, dualmente: $-(-A \vee -B) \equiv A \wedge B$

Proviamo per esempio a costruire il gate xor (or esclusivo):

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

… A xor B vale 1 per –A e B oppure per A e –B, cioè:



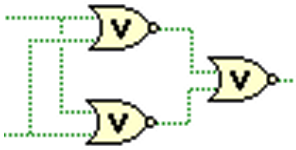
… il cui negato è un comparatore …

Riducibilità …

Dunque ogni gate a m ingressi e 1 uscita è esprimibile equivalentemente con (= riducibile a) una somma di prodotti dei simboli in ingresso o dei loro negati

Inoltre impiegando solo gate nor (cioè not or) è facile realizzare gate not e or

Gate not realizzato con un gate nor



Gate or realizzato con tre gate nor

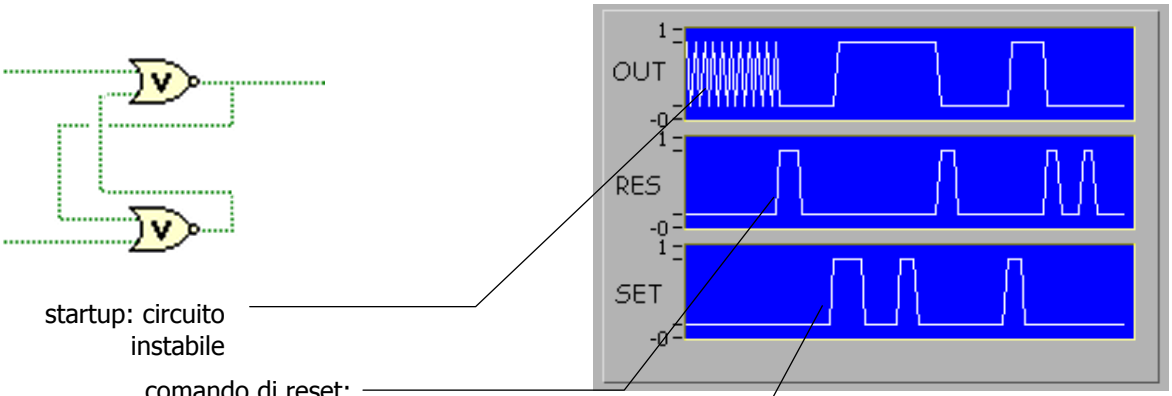
Dunque *con un solo tipo di gate è possibile realizzare tutte le funzioni logiche*

Oltre la logica combinatoriale

Tutti i gate visti finora trasformano istantaneamente gli input in output, secondo una logica *di tipo combinatoriale*

Dunque non sono in grado di fungere da unità di memoria

Il più semplice circuito con la capacità di fornire un simbolo in uscita in funzione non solo dei simboli presenti in quel momento in ingresso ma anche dei simboli forniti nel passato è il *flip-flop R-S* (reset-set), ottenuto mettendo in feedback l'uno sull'altro due gate nor



startup: circuito instabile

comando di reset: l'output va a 0 stabilmente, fino a successivo …

comando di set: l'output va a 1 stabilmente

Logica sequenziale

Il flip-flop R-S è un esempio di un dispositivo che opera secondo una logica *di tipo sequenziale*: è l'intera sequenza degli input passati a determinare l'output attuale

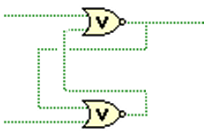
In alternativa, si può formalizzare il comportamento di questi dispositivi introducendo il concetto di *variabile di stato*: in ogni istante l'output del dispositivo dipende quindi non solo dall'input in quell'istante, ma anche dallo stato in cui si trova il dispositivo

NOTA: un flip-flop R-S realizza un'unità di memoria da 1 bit …

Un esempio /1

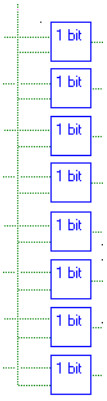
Adottando la logica di assemblaggio progressivo di componenti, proviamo a creare il prototipo di un'unità di memoria, per esempio da 4 byte, dunque da indirizzare con 2 bit

Partiamo da un bit:



che incapsuliamo in una "scatola nera"
con due input (set e reset)
e un output

Una successione di 8 bit costituisce un byte:
un dispositivo che possiamo pensare a 9 input
(8 pin di set e 1 pin, comune a tutti i bit, di reset)
e 8 output



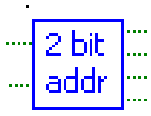
Rappresentiamo in questo modo:
il byte, con i due bus a 8 linee parallele

Un esempio /2

Ci dotiamo quindi di un dispositivo di input dati: per assegnare 8 valori 0 o 1

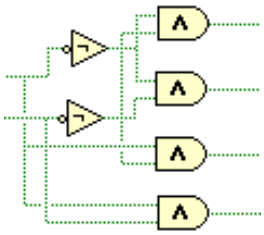
Dobbiamo ora risolvere il problema dell'indirizzamento: come scegliere in quale dei 4 byte scrivere la successione assegnata in input?

Ci occorre un dispositivo di input indirizzi (a 2 bit, dato che vogliamo indirizzare 4 unità di memoria), da accoppiare a un "indirizzatore", cioè un dispositivo combinatoriale con una tavola della verità:



In1	In0	Out3	Out2	Out1	Out0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

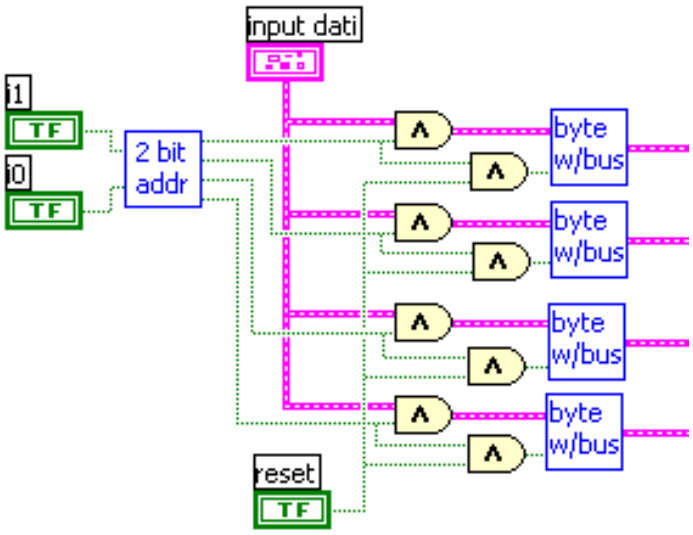
Abbiamo visto come realizzarlo combinando gate elementari:



In questo modo in ogni istante una e una sola linea in uscita si trova a valore 1

Un esempio /3

A questo punto dobbiamo solamente "mettere in and" le uscite del dispositivo di indirizzamento con le linee dati e la linea di reset, in modo che venga coinvolto nei comandi reset e set solamente il byte specificato dal dispositivo di input indirizzi

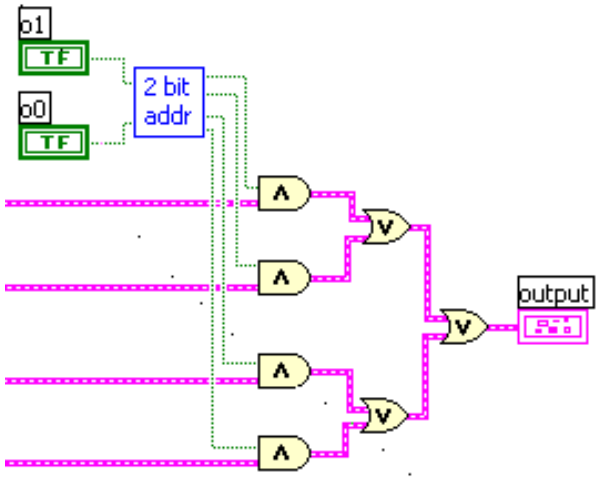


Abbiamo ottenuto così un sistema a 4 uscite, una per ogni byte

Un esempio /4

Per completare, dobbiamo passare da 4 a 1 uscita, "mascherando" le 4 linee con un dispositivo di indirizzamento mediante il quale specificare quale byte si vuole leggere

La logica non è diversa da quella già vista:



Abbiamo così ottenuto, complessivamente, un'unità di memoria con:

8 bit per l'input dei dati + 2 bit per la selezione del byte su cui scrivere + 2 bit per la selezione del byte da cui leggere; 1 linea di reset; 8 bit per l'output