Logica circuitale.

Si dà per assodato l’uso di una rappresentazione binaria per le informazioni. Lo scopo della logica circuitale è quella di andare a interagire con le codifiche binarie per l’esecuzione di operazioni.

Si definiscono tre variabili (a, b, c) che possono assumere i valori in rappresentazione binaria (valori di un solo bit). Si definisca poi la variabile x che rappresenti il valore in uscita di una funzione f(a, b, c). Il caso più semplice è quello di una funzione f(a) (con una sola variabile binaria) in cui il numero di combinazioni possibili in ingresso e in uscita è limitato. Dato che a occupa un solo bit, si hanno due soli valori possibili in entrata (0, 1) e fino a 2 valori possibili in uscita (0, 0 / 0, 1 / 1, 0 / 1, 1). Ci sono quindi 4 funzioni possibili che si possono realizzare: la funzione costante x = 0, la funzione costante x = 1 (è quindi un po’ azzardato chiamarle funzioni), la funzione-identità x = a (1 se è 1, 0 se è 0) e la funzione di negazione x = 1 – a (è l’unica interessante perché fa qualcosa). Se vogliamo altre funzioni è necessario aumentare il numero di variabili in ingresso; nel caso di funzioni a due valori in ingresso (a, b) la “tavola di verità” che rappresenta i possibili esiti della funzione ha 2^2 = 4 righe. Il numero di funzioni totali che si possono costruire è invece sempre 2^(2^n) (in questo caso 2^4 =16). In generale, avere n variabili in ingresso permette di produrre 2^n funzioni diverse. Tra tutte le funzioni possibili solo alcune hanno un’utilità pratica, nel caso di due variabili un esempio è quella che restituisce 1 solo se entrambe hanno valore 1: questa funzione prende il nome di AND. Un altro esempio è la funzione OR: restituisce 1 se almeno una variabile ha valore 1. C’è poi lo XOR, che restituisce 1 se i due valori sono diversi (or esclusivo). Ci sono poi le negazioni: la negazione dell’AND (NAND) restituisce 0 solo se entrambe le variabili sono vere, la negazione dell’OR (NOR) restituisce 1 solo se entrambe le variabili sono false e la negazione dello XOR (XNOR) che restituisce 1 solo se le due variabili hanno lo stesso valore (= Funzione uguaglianza).

Con 3 variabili in ingresso il numero di funzioni possibili sarebbe 2^(2^3) = 256. Poiché 256 sono Tante funzioni, in genere si parte dalle funzioni base (AND, OR, XOR e le rispettive negazioni) e le si utilizza per la costruzione di tutte le altre funzioni più complesse.

Tra i metodi alternativi alla tavola di verità c’è quello dell’introduzione dei simboli di funzione:

* La negazione: a -triangolino (o)- u (NOT)
* La funzione AND:
  + A – semi disco –u
  + B – semi disco

La funzione OR:

A – doppio-semicerchio – u

B – doppio-semicerchio

La funzione XOR:

A – / - doppio semicerchio – u

B - / - doppio semicerchio

Le corrispondenti funzioni negate sono uguali ma con un pallino (o) subito dopo la “figura”.

* L’identità a\_\_\_\_\_u
* La costante 0\_\_\_\_\_u

Questa rappresentazione grafica permette di rappresentare più funzioni di fila (o meglio Concatenate).

Esiste anche un’altra rappresentazione, quella di tipo algebrico (Algebra Booleana).

Si scrivono delle formule in cui si usano come simboli quelli delle variabili in ingresso, il simbolo di negazione (barra sopra un altro simbolo: ) e i simboli + e \* che sono associati rispettivamente alle funzioni OR e AND. Il motivo per cui AND = \* è facile, due valori restituiscono 1 solo se moltiplichi 1\*1, mentre se uno dei due è 0 il prodotto sarà sempre 0. OR è il +, perché 0+1 = 1, 1+0=1, 0+0=0 (e fin qui tutto in regola con il funzionamento dell’OR); l’unico problema è quando si ha 1+1, poiché 2 necessita più di un bit per essere rappresentato, quindi per mantenere corretta la rappresentazione binaria si potrebbe associare l’operazione di somma allo XOR (lo 0 rappresenterebbe ciò che rimane dopo il riporto di 1), tuttavia si preferisce comunque associare alla somma la funzione OR. Si fa ciò per poter continuare a sfruttare le proprietà algebriche della somma (come la proprietà commutativa, distributiva, associativa ecc. Da notare che la stessa cosa vale anche per l’AND che viene associata al \*). Nella rappresentazione algebrica lo XOR non compare.

La funzione XOR si può però costruire, si prende la tavola di verità dello XOR (e si nota che il valore in uscita è 1 se in ingresso ci sono un 1 e uno 0). Si crea quindi la somma in forma normale disgiuntiva (-a\*b + a\*-b).

Si prenda per esempio una funzione inventata che accetti in ingresso 3 valori (e che restituisca, in ordine nella tavola di verità, 1 0 0 0 1 1 0 0). Si identificano le righe che contengono un valore 1 in uscita, si può quindi creare una formula in forma normale disgiuntiva come somma di tre prodotti: (-a\*-b\*-c)+(a\*-b\*-c)+(a\*-b\*c).

Graficamente, si potrebbe inventare una funzione AND a tre ingressi usando due funzioni AND a due ingressi:

--\_

-----funzione and --\_

----------------------------funzione and-----------

La prima delle tre funzioni (la triplice AND con valori negati) si può creare con la funzione appena rappresentata con un not (una freccia a triangolo con pallino) a ciascuno dei tre ingressi. Analogamente si potrebbe creare una funzione OR a tre ingressi che accetti i risultati delle tre funzioni AND.

NOTA: si può moltiplicare/fare uno split di un filo se si intende utilizzare il valore calcolato da una determinata funzione per più di una funzione successiva (o meglio, lo si “ricicla”. Una volta calcolati a negato, b negato e c negato si possono far partire altri fili da dopo che è avvenuta la negazione per riutilizzare i valori in più occasioni).

C’è da dire, inoltre, che sfruttando le proprietà dell’algebra booleana, se si guarda a una formula del tipo

(-a\*-b\*-c)+(a\*-b\*-c)+(a\*-b\*c) e in particolare ai primi due termini, si può fare un raccoglimento di (-b\*-c) e ottenere (-b\*-c)\*(a+-a). Ma a+-a è uguale a 1 (-a è a negato, non a \* -1) quindi i primi due termini si possono riscrivere (-b\*-c) -> (-b\*-c)+(a\*-b\*c) diventa la formula semplificata (a sua volta si potrebbe semplificare in –(b+c)+(a\*-b\*c) grazie alle formule di de Morgan, per risparmiare sui componenti.).

La semplificazione sfruttando le proprietà dell’algebra “viene bene” quando è una persona a farla manualmente. Quando si introducono degli algoritmi per farlo ciò si rivela più complicato. Per cercare di migliorare la situazione si è introdotta una rappresentazione delle funzioni alternativa alle tavole di verità: Le mappe di Karnaugh. Queste mappe sono delle matrici: le variabili vengono divise in due sottoinsiemi di variabili, uno viene messo in verticale e l’altro in orizzontale (nell’esempio a 3 variabili, 2 sono messe in verticale e 1 in orizzontale). Questo permette di ridistribuire le caselle su un piano anziché in riga.

|  |  |  |
| --- | --- | --- |
| ab \ c | 01 | 1 |
| 00 | 1 | 0 |
| 01 | 0 | 0 |
| 11 | 0 | 0 |
| 10 | 1 | 1 |

Si scambiano le due righe finali (non sono in ordine crescente, infatti 11 non è in fondo). Si fa ciò perché da riga a riga successiva così cambia un solo bit. Per capire perché si fa bisogna introdurre il concetto di Distanza di Hamming. Quando si vuole confrontare due stringhe di bit la distanza di Hamming ci dice quanti sono i bit diversi. (Quando nei bit si trovano due valori diversi si aggiunge 0, quando si trovano due valori diversi si aggiunge 1: per esempio 11001 10110 hanno distanza di Hamming = 4) Le stringhe 110001101 e 111001001, per esempio, hanno distanza di Hamming = 2.

Mantenere una distanza di Hamming = 1 nell’elenco dei valori delle variabili, torna utile perché all’interno della tabella qualsiasi riga si può considerare adiacente a ciascun’altra da cui disti 1. Ciò permette di scrivere subito la formula semplificata della formula u = (a\*-b) + (-b\*-c). Basta infatti vedere cosa caratterizza (o meglio cosa hanno in comune) tutte le caselle considerabili adiacenti (cioè con distanza di Hamming = 1) che contengono il valore 1. Nel caso di:

|  |  |  |
| --- | --- | --- |
| ab \ c | 01 | 1 |
| 00 | 1 | 1 |
| 01 | 0 | 0 |
| 11 | 0 | 0 |
| 10 | 1 | 1 |

La formula semplificata è addirittura u = -b.