Un simulatore parallelo di circuiti elettronici

Porte logiche complesse

di Giuseppe Cardinale Ciccotti

Quando si intraprende sulle pagine di una rivista la realizzazione di un progetto a puntate è inevitabile che ci sia un certo numero di lettori che per caso o solo per curiosità, non seguano il discorso fin dall'inizio. Purtroppo se è nella natura stessa della struttura frazionata che il discorso possa risultare frammentario, è pure vero che costruendo modularmente il progetto, i problemi dovrebbero essere minimizzati.
Tutto ciò per dire che sarà necessario fare qualche riferimento alle puntate precedenti, perché il nostro simulatore poggia su mattoni elementari che abbiamo iniziato a illustrare nella scorsa puntata.
Il codice che viene pubblicato man mano costituisce quindi il patrimonio indispensabile per costruire il completo impianto simulatorio che ci siamo proposti.
Continueremo in questo numero ad inserire nuovi dispositivi sempre più complessi nella nostra libreria introducendo parimenti nuovi concetti di cui abbiamo bisogno

Logica positiva e logica negativa

Nel primo appuntamento abbiamo già posto i primi mattoni elementari e precisamente le porte AND, OR e NOT tramite i processi gAND, gOR, gNOT. Queste porte ci consentono di costruire tutti i dispositivi che realizzano le altre funzioni logiche più complesse ed infatti sempre nel numero scorso abbiamo visto come implementare un gate NAND ed uno XOR.

Tuttavia la logica AON (AND, OR, NOT) non è la più usata nelle applicazioni perché si tende ad utilizzare sempre gli stessi componenti per semplicità ed economia. Come molti lettori già sapranno è possibile realizzare tutte le funzioni logiche elementari utilizzando soltanto logica NAND oppure logica NOR. È possibile infatti giustapporre un certo numero di porte tutte uguali per realizzare un circuito che si comporti come la porta elementare.

Nella logica digitale esiste la possibilità di scegliere le definizioni da adottare nel senso che non necessariamente la massa deve corrispondere al livello logico 0 e la tensione +5 al livello logico 1.

Ci possono essere infatti due diverse possibili attribuzioni.

Se al +5 assegniamo il valore di 1 logico e alla massa lo 0 logico si parlerà di logica positiva.

Se viceversa al +5 assegniamo il valore di 0 logico e alla massa quello dell'1 logico si parlerà di logica negativa.

Usualmente si fa ricorso alla convenzione positiva ma se ragionate sulle ta-

Regole di addizione binaria

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 con riporto di 1

Per chi non le ricordasse o per coloro che non le conoscessero, queste sono le semplici regole di addizione binaria.

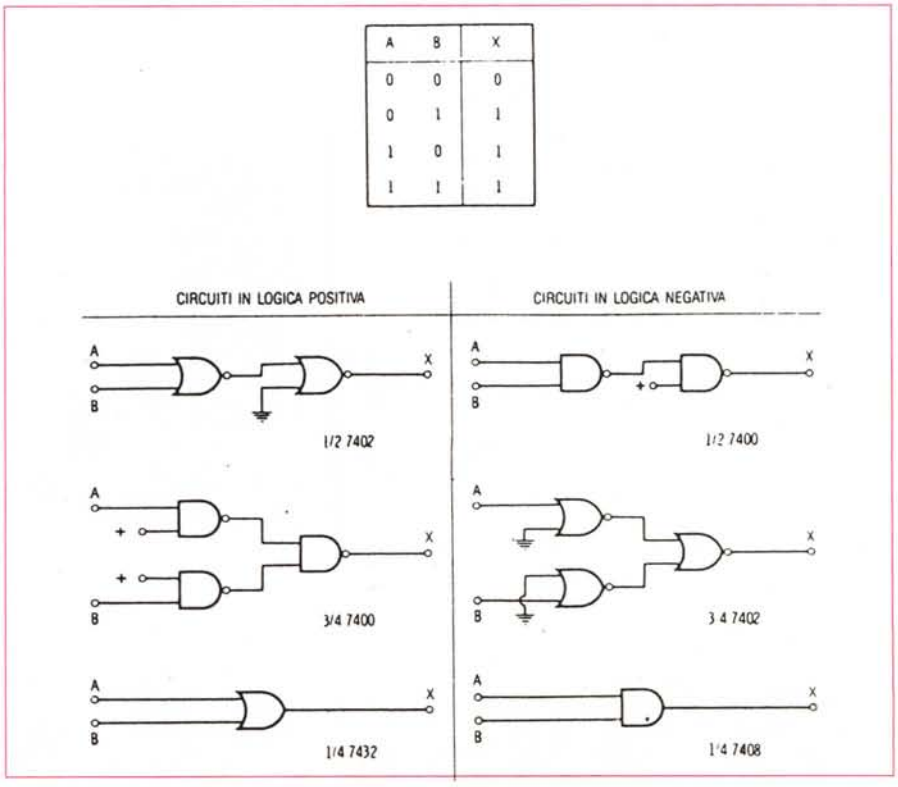


Figura 1 — Circuiti relativi alla porta OR in logica positiva e negativa.

belle della verità vi accorgerete che per ciascuna porta è possibile utilizzare entrambe le logiche.

Ad esempio, una porta NAND nella logica positiva corrisponde dal punto di vista hardware ad una porta NOR nella logica negativa.

La scelta sulla convenzione da usare dipende da come sono definiti i segnali di ingresso e di uscita del circuito e dalla necessità di contenere al massimo il numero di porte utilizzate.

Come abbiamo già detto è possibile realizzare qualsiasi tipo di funzione logica con sole porte NAND o NOR, questa considerazione è un fatto veramente importante perché per realizzare un circuito è possibile utilizzare un tipo soltanto di porte, con una notevole semplicità costruttiva.

Vediamo allora in figura 1 come si realizza la funzione OR in logica negativa e positiva, con circuiti NOR o NAND.

In figura 2 viceversa trovate il medesimo schema per la porta AND.

Come avrete senz'altro già notato, quando si realizza la funzione con porte NAND o NOR si arriva sempre a circuiti a più livelli di porte, in questo semplice caso due; allora questo modo di progettare le funzioni sarà detto NAND-NAND o NOR-NOR a seconda del tipo di porta utilizzato.

Se confrontate le soluzioni circuitali vi renderete subito conto che la porta AND è la porta OR in logica negativa e viceversa; vale a dire che per passare da una logica all'altra basta invertire la massa con la tensione positiva. Ad ulteriore conferma considerate il circuito in logica positiva dell'OR realizzato in NOR-NOR: il secondo NOR con un ingresso fisso a massa, funge da invertitore.

Nel circuito in logica negativa NAND-NAND invece, il secondo NAND per funzionare da invertitore ha necessità di avere un ingresso fisso a tensione positiva.

Per completare il quadro, in figura 3 e 4 sono illustrate rispettivamente la porta NOR e quella NAND.

Per quanto riguarda la funzione NOT è facile vedere che in logica positiva può essere realizzata con una porta NAND in cui un ingresso è fissato a 1 mentre in logica negativa con una porta NOR in cui un ingresso è messo a massa.

Da queste considerazioni nasce per noi la pressante esigenza di definire in maniera efficiente le porte NAND e NOR; potremo così costruire un circuito in logica NAND-NAND e NOR-NOR, senza generare un numero di processi eccessivo e quindi guadagnare un po' di efficienza.

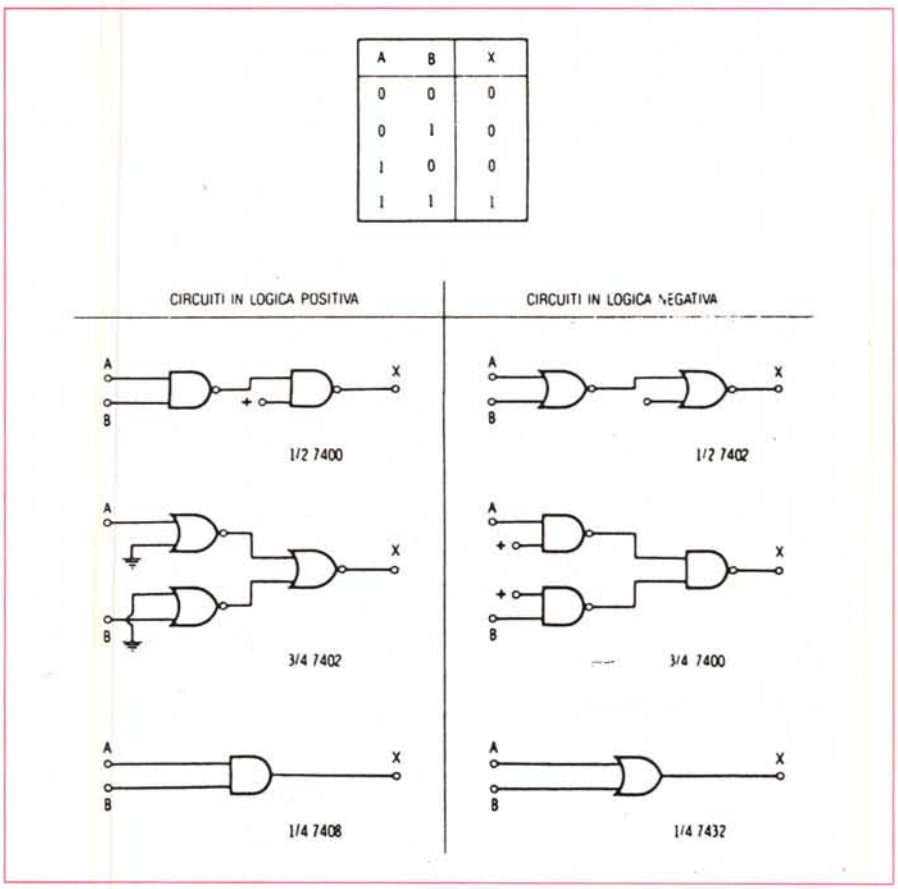


Figura 2 — Circuiti relativi alla porta AND in logica positiva e negativa.

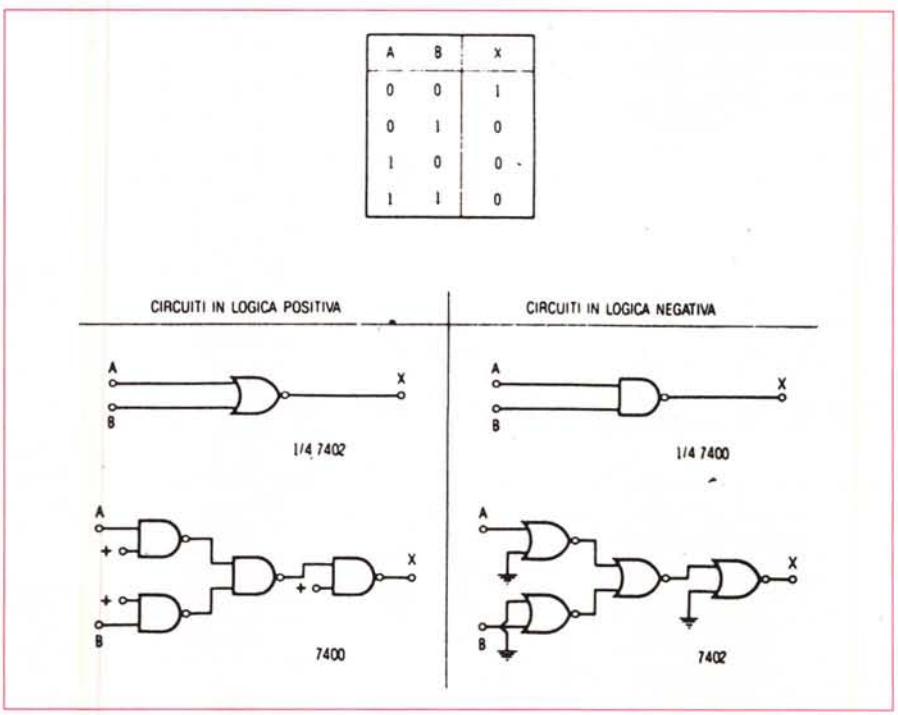


Figura 3 — Circuiti relativi alla porta NOR in logica positiva e negativa.

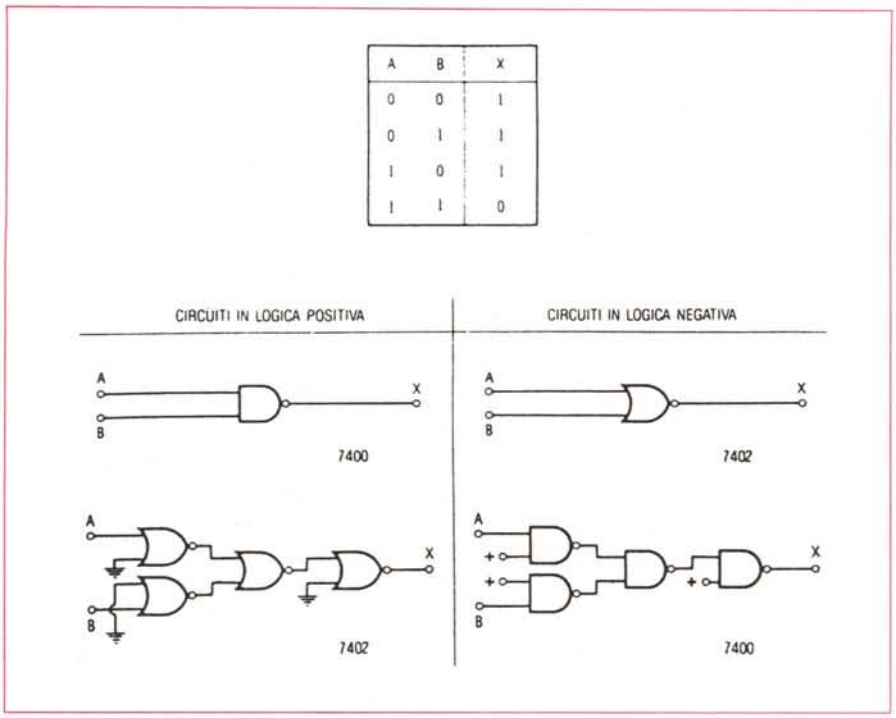


Figura 4 — Circuiti relativi alla porta NAND in logica positiva e negativa.

costruire un unico processo che realizzi il NAND. In figura 5 troverete il codice OCCAM per il processo NAND, questa volta non c'è bisogno di adoperare il prefisso g (gate) perché non esiste in OCCAM la parola riservata NAND e cioè non c'è possibilità di confusione, mentre in figura 6 quello per il processo NOR.

L'altro circuito fondamentale che avevamo visto la scorsa puntata era quello dell'OR esclusivo o come comunemente si dice XOR.

Come ricorderete era un circuito un po' complesso (neanche tanto) che ci aveva permesso di mettere in luce le caratteristiche dell'approccio scelto per costruire il nostro simulatore.

In figura 7 potete vedere come si può costruire una porta XOR in logica positiva; ci sono due diversi circuiti: nel primo è necessario avere anche gli ingressi complementati altrimenti bisogna calcolarli interponendo due NAND in configurazione di NOT come nel secondo circuito. Nel secondo circuito è presente la connessione che continueremo ad implementare con un semplicissimo processo CONNECT che abbiamo introdotto nella scorsa puntata.

Come vi rendete conto, tale circuito è

I processi NAND e NOR

Nella scorsa puntata abbiamo visto che la funzione NAND è realizzata connettendo un AND ed un NOT in cascata, quindi due processi OCCAM sequenziali; però per le ragioni espresse nel paragrafo precedente ci conviene

```
CHAN OF BOOL px, py, e, pout :

PROC NAND (CHAN OF BOOL x, y, output)
  BOOL dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := NOT(dx AND dy)
    output ! do
:
```

Figura 5 — Codice OCCAM per la porta NAND, è senz'altro più maneggevole e compatto.

```
PROC NOR (CHAN OF BOOL x, y, output)
  BOOL dx,dy,do:
  SEQ
    PAR
      x ? dx
      y ? dy
    do := NOT(dx OR dy)
    output ! do
:
```

Figura 6 — Processo NOR, l'impostazione rispecchia quella del processo NAND.

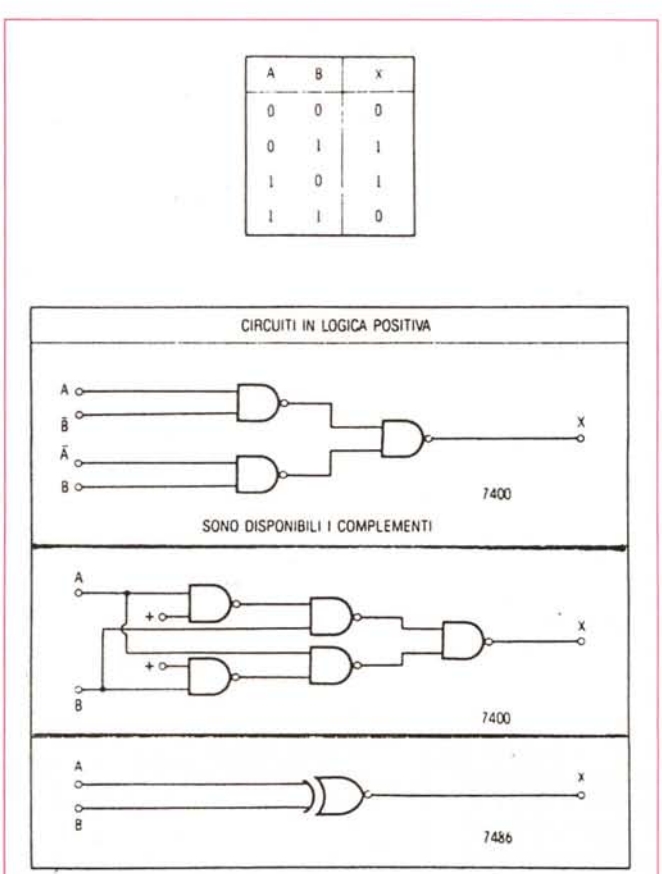


Figura 7 — Circuiti XOR in logica positiva e tabella della verità.

```
PROC gAND (CHAN OF INT16 x, y, output)
  INT16 dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := dx BITAND dy
    output ! do
:

PROC gOR (CHAN OF INT16 x, y, output)
  INT16 dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := dx BITOR dy
    output ! do
:

PROC NAND (CHAN OF INT16 x, y, output)
  INT16 dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := BITNOT(dx BITAND dy)
    output ! do
:

PROC NOR (CHAN OF INT16 x, y, output)
  INT16 dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := BITNOT(dx BITOR dy)
    output ! do
:

PROC gNOT (CHAN OF INT16 x, output)
  INT16 dx,do :
  SEQ
    x ? dx
    do := BITNOT dx
    output ! do
:

PROC XOR (CHAN OF INT16 x, y, output)
  INT16 dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := dx >< dy
    output ! do
:
```

Figura 8 — Codice per i processi delle porte implementati con valori interi a 16 bit piuttosto che con valori booleani. Gli operatori BITAND, BITOR e BITNOT sono operatori che agiscono sui bit proprio come il nome stesso specifica.

veramente un po' troppo complicato rispetto alla funzione che realizza, tra l'altro molto utile ed usata, perciò cercheremo di costruirla in modo più semplice con un solo processo OCCAM. In questa occasione introduciamo un cambiamento che ci servirà molto in futuro: invece di considerare le variabili sui canali come dei valori booleani, useremo delle variabili di tipo intero. Raggiungiamo due scopi, il primo è quello di utilizzare gli operatori sui bit invece che quelli relazionali sui booleani, il secondo sta nel fatto che lavorando con valori numerici possiamo introdurre anche stati diversi dallo 0 e dall'1 come lo stato non definito o il Tri-State.

```
PROC NAND (CHAN OF INT16 x, y, z, w, output)
  INT16 dx,dy,dz,dw,do :
  SEQ
    PAR
      x ? dx
      y ? dy
      z ? dz
      w ? dw
    do := BITNOT((dx BITAND dy) BITAND (dz BITAND DW))
    output ! do
:
```

Figura 9 — Processo NAND a 4 ingressi; con questa tecnica potete simulare porte a quanti ingressi volete!

In realtà dal punto di vista della programmazione non cambia proprio nulla se non la definizione dei canali e delle variabili.

Possiamo d'altra parte implementare facilmente l'XOR perché l'OCCAM ci fornisce l'operatore OR esclusivo sui bit con l'operando ><.

Quindi in figura 8 trovate il codice delle sei porte principali AND, OR, NOT, NAND, NOR e XOR realizzate con canali di INT e con gli operatori bitwise.

Porte a più ingressi

Tutte le porte che abbiamo implementato finora sono ad un massimo di due ingressi, mentre in realtà sono presenti dispositivi anche a otto ingressi. Dal nostro punto di vista non c'è nessun problema a realizzare tali porte perché basta scrivere processi OCCAM che abbiano un adeguato numero di canali di input per risolvere il problema, in figura 9 riportiamo il codice di una porta NAND a quattro ingressi abbastanza comune nelle applicazioni.

I lettori più attenti saranno ora in grado di scrivere semplicemente i processi che realizzano le porte con quanti ingressi servono.

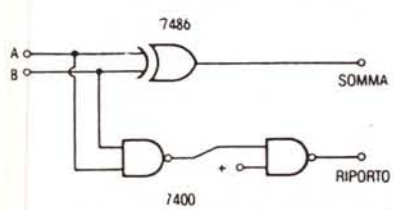
Un esempio: l'half-adder

Per terminare questa puntata in cui abbiamo fatto un ulteriore passo in avanti e non volendo anticipare l'introduzione dei circuiti sincroni (clock, flip-flop, memorie, etc.) per non spaventare i più inesperti fra i lettori, proponiamo un esempio semplice ma significativo: un circuito addizionatore.

Come ben sapete l'addizione di due cifre decimali, binarie o in qualsiasi base voi vogliate, produce sempre un risultato ed un riporto che può essere nullo o positivo. In figura 10 è mostrata la tabella della verità dell'addizionatore: co-

```
CHAN OF INT16 A, B, e1, e2, e3, e4, e5, somma, riporto, vpos:

PROC NAND (CHAN OF INT16 x, y, output)
  INT16 dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := BITNOT(dx BITAND dy)
    output ! do
:

PROC XOR (CHAN OF INT16 x, y, output)
  INT16 dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := dx >< dy
    output ! do
:

PROC kick (CHAN OF INT16 x, y, INT16 a,b)
  INT16 a,b :
  PAR
    x ! a
    y ! b
:

PROC basket (CHAN OF INT16 x)
  INT16 a :
  x ? a
:

PROC connect (CHAN OF INT16 x, y, z)
  INT16 a :
  SEQ
    x ? a
    y ! a
    z ! a
:

PAR                          -- main process HALF-ADDER
  kick (A, B, #FFFF, #0000)  -- valori di esempio
  connect (i1, e1, e3)
  connect (i2, e2, e4)
  vpos ! #FFFF  -- piedino a +Vcc fisso
  XOR (e1, e2, somma)
  NAND (e3, e4, e6)
  NAND (e6, vpos, riporto)
  basket (riporto)
  basket (somma)
:
```

Figura 10 — Half-Adder circuito, tabella della verità e codice OCCAM. Abbiamo indicato il valore TRUE con la word FFFF e il valore FALSE con la word 0000. Il canale vpos indica che l'ingresso della porta NAND è connessa ad una tensione positiva, in modo che ci sia un 1 fisso su quell'ingresso.



B	A	SOMMA	RIPORTO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

me vedete dovendo generare due funzioni la somma ed il riporto, la tabella definisce due uscite ai due ingressi. Confrontate la colonna della somma, con l'uscita della tabella della verità del circuito XOR, sono uguali!

Vale a dire che con una semplice porta XOR possiamo generare il bit di somma di due bit in ingresso.

Il riporto è invece diverso da 0 soltanto se gli ingressi sono entrambi uguali ad 1, corrisponde perciò ad un AND. Il circuito mostrato in figura realizza l'AND con due porte NAND sfruttando gli schemi delle figure precedenti.

Il codice OCCAM ormai non presenta più segreti per noi e come vedete la somma ed il riporto sono calcolate in parallelo.

Questo circuito pur essendo utile serve per addizionare al massimo le ultime due cifre di un numero binario in quanto non viene tenuto conto del riporto generato dalla somma precedente (ricordatevi come fate la somma di due numeri o per rinfrescarvi le idee guardate il riquadro!); per tale motivo questo circuito è chiamato Half-Adder cioè mezzo sommatore, infatti fa soltanto metà del lavoro!

Per completare l'opera in figura 11 trovate un Full-Adder: un bel circuito che si incarica di generare il bit di somma e quello di riporto tenendo conto del bit di riporto precedente oltre che dei due bit da sommare.

La tabella della verità ha naturalmente tre ingressi e due uscite con otto possibili risultati (il numero di risultati in uscita è pari a 2 elevato al numero di ingressi).

Dal punto di vista del sorgente OCCAM non c'è nessuna difficoltà; alla fine mettiamo in fila come ci pare gli undici processi, cinque porte, quattro connessioni, il Kick e il Basket, facciamo i corretti collegamenti ed ecco il risultato.

Un minimo di interattività

Come si dice l'appetito vien mangiando e così non credo che sia divertente ricompilare tutto il codice ogni volta che vogliamo cambiare il valore di uno dei bit di ingresso al circuito; è senz'altro più utile poter immettere da tastiera i valori mentre il programma gira. Proprio per questo sostituiamo i processi Kick e Basket con un processo che comunica con l'host utilizzando due procedure della libreria HostIO fornita con tutti i sistemi di sviluppo Transputer. In figura 12 è riportato il codice OCCAM della procedura Set_Input_&_Get_Out-

Bibliografia

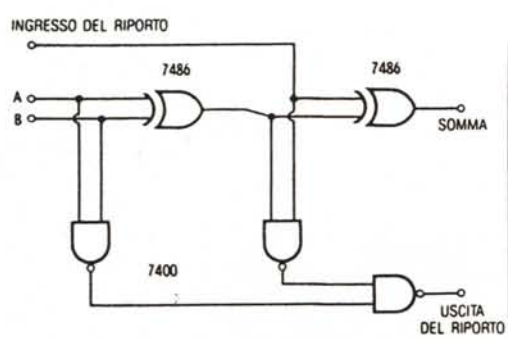
D. Pountain, D. May, ***«A tutorial introduction to Occam programming»***, BSP, 1988

Inmos ltd., ***«OCCAM 2 Reference Manual»***, Prentice-Hall, 1988

M. Morris Mano, ***«Digital Design»***, Prentice-Hall, 1984

Don Lancaster ***«Circuiti Logici TTL»***, Tecniche Nuove, 1986


```
CHAN OF INT16 A, B, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, somma, Rin, Rusc:

PROC NAND (CHAN OF INT16 x, y, output)
  INT16 dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := BITNOT(dx BITAND dy)
    output ! do
:

PROC XOR (CHAN OF INT16 x, y, output)
  INT16 dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := dx >< dy
    output ! do
:

PROC kick (CHAN OF INT16 x, y, INT16 a,b)
  INT16 a,b :
  PAR
    x ! a
    y ! b
:

PROC basket (CHAN OF INT16 x)
  INT16 a :
  x ? a
:

PROC connect (CHAN OF INT16 x, y, z)
  INT16 a :
  SEQ
    x ? a
    y ! a
    z ! a
:

PAR                          -- main process FULL-ADDER
  kick (A,B, #FFFF, #0000)  -- valori di esempio
  Rin ! #0000  -- riporto dello stadio precedente
  connect (A, e1, e3)
  connect (B, e2, e4)
  connect (Rin, e6, e8)
  connect (e5, e7, e9)
  XOR (e6, e7, somma)
  XOR (e1, e2, e5)
  NAND (e3, e4, e10)
  NAND (e8, e9, e11)
  NAND (e10, e11, Rusc)
  basket (Rusc)
  basket (somma)
:
```



R_{in}	B	A	SOMMA	R_{usc}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figura 11
Full-Adder, potete divertirvi a connettere più addizionatori fra loro per farne uno a più bit, semplicemente collegando il canale Rusc al canale Rin dell'addizionatore che fornisce il bit di peso immediatamente superiore della somma.
Provate a costruire un addizionatore di interi a 16, 32 o 64 bit!

```
PROC Set_Input(CHAN OF SP  fs,ts, CHAN OF INT16 A)
  INT16 x:
  INT bit:
  BOOL err, bit_ok:
  VAL prompt IS "Input value ? ":
  SEQ
    bit_ok:=FALSE
    err:=TRUE
    WHILE (err AND (NOT bit_ok))
    so.write.nl(fs,ts)
    so.write.string(fs,ts,prompt)
    so.read.echo.int(fs,ts,bit,err)
    IF
      NOT err
      IF
        bit=1
          x:=#FFFF
          bit_ok:=TRUE
        bit=0
          x:=#0000
          bit_ok:=TRUE
        TRUE
          SKIP
    A ! x
:

PROC Get_Output(CHAN OF SP  fs,ts, CHAN OF INT16 A)
  INT16 x:
  VAL promptout IS "Output value: ":
  SEQ
    A ? x
    so.write.string(fs,ts,promptout)
    so.write.int(fs,ts, INT(x),0)
    so.write.nl(fs,ts)
:

PROC Set_Input_&_Get_Output(CHAN OF SP  fs,ts, CHAN OF INT16 ingresso, risultato)
  SEQ
    Set_Input(fs,ts,ingresso)
    Get_Output(fs,ts,risultato)
:
```

Figura 12 — Procedura per la gestione dei valori di I/O dall'host verso il simulatore. Ogni volta che viene modificato un bit in ingresso a video viene stampato il risultato ritornato sul canale risultato.

put, essa si incarica di modificare il bit di uno degli ingressi al circuito e di riportare a video il valore dell'output all'uscita della rete logica. Non sono state realizzate due funzioni separate, anche se si sarebbe potuto, perché in questo modo serializzando l'output rispetto all'input si è sicuri che il risultato ritornato è proprio quello calcolato in funzione degli ingressi immessi.

Conclusioni

Continuando nella realizzazione del nostro simulatore abbiamo puntualizzato determinati concetti che ci serviranno in futuro e nel contempo ci siamo impratichiti nella costruzione di una rete combinatoria; è tempo ora di dedicarci alla logica sequenziale. Dalla prossima volta parleremo di flip-flop, clock, memorie e di circuiti sincroni; sarà necessario predisporre un certo numero di procedure OCCAM di servizio per una maggiore interattività e per interpretare correttamente i risultati prodotti. Arrivederci! MC