coordinamento di Andrea de Prisco

Un simulatore parallelo di circuiti elettronici

prima parte

Il risolutore di espressioni booleane

di Giuseppe Cardinale Ciccotti

Il nostro mondo, intendo quello reale, è un'interazione di esseri, ognuno indipendente dall'altro che agisce secondo una propria logica (logica ?!). Gran parte delle cose nel nostro mondo accadono contemporaneamente e ci pare assolutamente normale. Perché dunque quando invece progettiamo un algoritmo lo pensiamo come una sequenza strettamente ordinata di passi? A quest'interrogativo filosofico, i lettori più dogmatici potranno rispondere che i classici problemi dell'informatica sono stati risolti con algoritmi seriali, ma questo è vero soltanto perché si è assunto che la progettazione di un algoritmo sia essenzialmente seriale

Se poi consideriamo campi non classici come il calcolo simbolico ad esempio subito la tesi dogmatica non può più essere sostenuta. La simulazione è un'altra delle attività dove lo sviluppo di algoritmi paralleli permette di semplificare in maniera decisiva le cose.

È proprio questo quello che nel presente e nei prossimi articoli ci proponiamo di dimostrare, «costruiremo» perciò un simulatore digitale o forse anche analogico se l'intelletto ci assisterà, di circuiti, completamente parallelo.

Ci affideremo al nostro fidatissimo Occam che mai come in questo caso sarà decisivo.

Il nostro simulatore girerà su uno o più Transputer, proprio perché mai come in questo momento sono l'unica risorsa per sperimentare le delizie del parallelismo a basso costo.

Sono presenti sul mercato ormai delle schede con un T400 o un T805 semplici ed efficaci, che per prezzi di poco superiori al mezzo milione vi offrono un kit di base completo di compilatore. Ma c'è già chi sta progettando di costruire un parallel computer desktop... che ne direste di 4 o 5 Transputer a 5/6 milioni e capaci di prestazioni fino a 15 volte più elevate di un 80486 a 25 MHz?

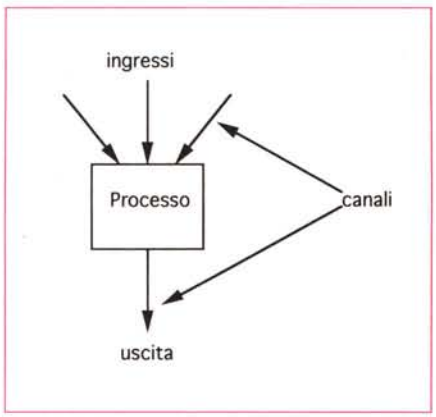


Figura 1 - I processi con cui vengono realizzate le porte logiche sono del tipo a n ingressi e una uscita.

La simulazione digitale

La simulazione è la riproduzione di uno o più eventi (per esempio il funzionamento di qualcosa) secondo un determinato modello del sistema simulato. Se è discreta significa che la valutazione dei parametri di interesse è campionata in determinati istanti piuttosto che a tempo continuo; è chiaro perciò che potremo avere una simulazione discreta di un sistema a tempo continuo. La simulazione discreta può essere analogica o digitale se il modello di calcolo che è utilizzato per la simulazione stessa è di tipo analogico o digitale.

Ci interesseremo di simulazione discreta digitale, mettendoci nell'ipotesi semplificativa, di simulare un sitema digitale o comunque rappresentabile tramite una combinazione di operazioni logiche.

Incominceremo con l'implementazione delle porte logiche elementari AND, OR, NOT, NAND, NOR e pochi altri elementi risolvendo semplici circuiti logici, aggiungeremo poi qualche semplice dispositivo che ci permetta di visualizzare il comportamento della rete ed avremo un nucleo attorno al quale costituire tutti gli altri elementi che simulano dispositivi reali.

La simulazione di un circuito elettronico

Implementare un simulatore digitale su un calcolatore seriale comporta te-

nere memorizzati in una matrice ingresso-uscita lo stato degli ingressi e delle uscite di ciascuna porta della rete logica. L'aggiornamento di questa matrice va fatto per ogni istante discreto in cui si calcola la simulazione, porta per porta. In pratica il nucleo di calcolo del simulatore non sarà nient'altro che un ciclo, iterato sugli istanti di simulazione, all'interno del quale vengono ricalcolati tutti gli ingressi e le uscite di ciascuna porta e quindi aggiornata la matrice che rappresenta il sistema. Per visualizzare la forma d'onda basterà pertanto disegnare a video i livelli corrispondenti al livello logico presente nei punti della rete che saranno stati indicati come output.

Nel caso invece di un'implementazione parallela, il programma di simulazione sarà composto da una serie di processi ognuno dei quali esegue una funzione logica elementare, indipendente l'uno dall'altro.

La filosofia dell'implementazione parallela

Per rispecchiare il più da vicino possibile il comportamento reale delle porte, predisporremo le cose in modo tale che i processi stessi siano accessibili soltanto tramite un passaggio esplicito di parametri, i terminali di ingresso e uscita della porta appunto.

In questa prima fase non ci occuperemo né di valutare i ritardi introdotti dalle porte che pure sono fondamentali in un simulatore degno di questo nome, né per questa prima puntata implementeremo meccanismi per ricostruire la forma d'onda nei punti in cui vogliamo visualizzarla.

Illustreremo di seguito, come implementare un solutore di espressioni booleane e la filosofia di base che permette di costruire una rete logica con processi Occam, sfruttando le peculiarità che tale linguaggio e l'implementazione hardware del task scheduler del Transputer offrono.

Ogni porta sarà perciò costituita da un processo che può essere visto come una procedura con un certo numero di ingressi ed una sola uscita, come potete vedere in figura 1.

I parametri di ingresso e di uscita saranno dei canali e precisamente dei canali che portano informazioni di tipo booleano; in questo modo, immessi in ingresso valori TRUE/FALSE, otteremo alle uscite o all'uscita della rete un valore TRUE/FALSE. Considerando poi che in Occam TRUE equivale a 1 e FALSE a 0 è facile ricondursi ai valori usuali dell'algebra binaria.

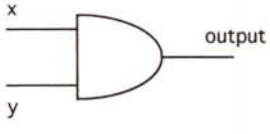
Senza complicarci troppo la vita e nell'ottica di fare un esempio didattico, utilizzeremo per ora solo porte a uno o due ingressi, anche perché questi sono i mattoni fondamentali sui quali si possono costruire qualsivoglia funzioni complesse.

Per evitare di incorrere nelle giuste ire del compilatore eviteremo di chiamare i processi come le funzioni che implementano perciò premetteremo una g (gate, porta in inglese) e i nostri processi si chiameranno gAND, gOR, gNOT.

La porta AND

Il primo processo che illustriamo in figura 2 è la porta AND a due ingressi; per i lettori che non conoscono le fun-

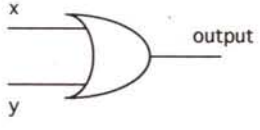
Figura 2 - Simbolo, tavola della verità e listato Occam della porta AND, la strategia dell'algoritmo rispecchia il funzionamento logico del dispositivo fisico.



x	y	output
F	F	F
F	T	F
T	F	F
T	T	T

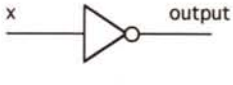
```
PROC gAND (CHAN OF BOOL x, y, output)
  BOOL dx,dy,do:
  SEQ
    PAR
      x ? dx
      y ? dy
    do := dx AND dy
    output ! do
:
```

Figura 3 - Simbolo, tavola della verità e listato Occam della porta OR, non c'è nessuna differenza rispetto alla porta AND tranne per l'operazione logica eseguita.



x	y	output
F	F	F
F	T	T
T	F	T
T	T	T

```
PROC gOR (CHAN OF BOOL x, y, output)
  BOOL dx,dy,do:
  SEQ
    PAR
      x ? dx
      y ? dy
    do := dx OR dy
    output ! do
```



x	output
F	T
T	F

```
PROC gNOT (CHAN OF BOOL x, output)
  BOOL dx,do:
  SEQ
    x ? dx
    do := NOT dx
    output ! do
:
```

Figura 4 - Porta NOT, simbolo, tavola della verità e listato del processo necessariamente sequenziale.

zioni elementari dell'algebra booleana (ma ce ne sono ancora?!) ricordiamo che la funzione AND restituisce sempre 0 (FALSE) in uscita a meno che gli ingressi non siano tutti a 1 (TRUE).

Come potete vedere il codice è più che elementare, notate però l'eleganza e la sintetica pulizia del processo e come sia intuibile il suo funzionamento. Il processo aspetta in parallelo di ricevere i valori sull'ingresso, in seriale esegue l'operazione logica e di seguito manda il risultato sul canale di uscita. È molto importante notare che le comunicazioni in attesa nel PAR non consumano tempo di CPU perciò il codice è abbastanza efficiente; inoltre, in qualche modo, rispecchia la realtà fisica dove effettivamente non si può predire quale degli ingressi arriverà prima. Per l'esecuzione della funzione logica ci affidiamo all'operatore AND dell'Occam.

La porta OR

In figura 3 trovate il codice, la tabella ed il simbolo della porta OR a due ingressi, ricordiamo che la funzione OR restituisce in uscita 1 (TRUE) se almeno uno degli ingressi è ad 1 (TRUE). Il codice segue la medesima filosofia di quello della porta AND mentre la funzione logica è eseguita dall'operatore OR dell'Occam.

La porta NOT

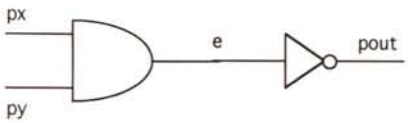
In figura 4 infine la terza porta fondamentale, questa volta ad un ingresso solo (ovviamente!), la cui funzione è semplicemente quella di invertire il valore ricevuto in ingresso. Notate come questa volta il codice sia necessariamente seriale.

Il risolutore di espressioni booleane

Adesso viene il bello! Costruiti con poca fatica i processi elementari, mettiamo insieme tutto e vediamo di fare calcolare il risultato di una funzione logica comunque complessa, costituita da AND, OR e NOT. Per inciso ricordiamo che con questi tre operatori fondamentali si possono costruire tutte le funzioni possibili.

Le più semplici ma anche le più utili sono le porte NAND e NOR. Si tratta nient'altro che di invertire le uscite di porte AND e OR rispettivamente, così ci basterà giustapporre per la porta NAND, un processo AND e uno NOT e per la porta NOR, un processo OR e uno NOT. In figura 5 vi mostriamo il codice per il circuito NAND; il programma

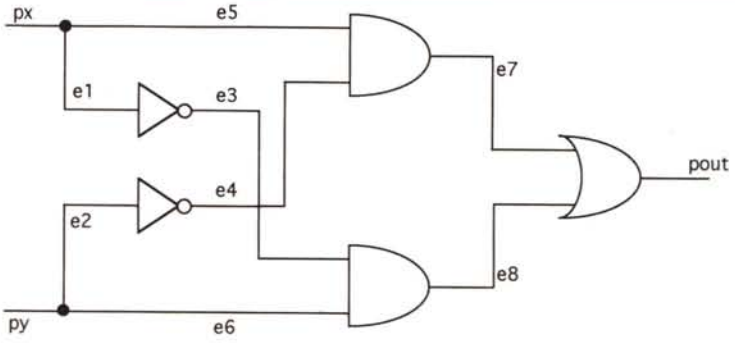
Figura 5 - Porta NAND realizzata con una porta AND invertita con una porta NOT. Il listato prevede stavolta due processi necessari a far partire e terminare il PAR. Una volta costruite le singole porte, è facile giustapporle per ricavare funzioni più complesse. ▶



px	py	pout
F	F	T
F	T	T
T	F	T
T	T	F

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

PROC gAND (CHAN OF BOOL x, y, output)
  BOOL dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := dx AND dy
    output ! do
:

PROC gNOT (CHAN OF BOOL x, output)
  BOOL dx,do :
  SEQ
    x ? dx
    do := NOT dx
    output ! do
:

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

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

PAR                          -- main process NAND
  kick (px, py, TRUE, FALSE)
  gAND (px, py, e)
  gNOT (e, pout)
  basket (pout)
:
```

Figura 6 - Circuito OR esclusivo e relativa tabella della verità. Notate la connessione sui terminali di ingresso e come debba essere implementata con un componente fittizio, visto che in Occam le connessioni sui canali sono soltanto punto-punto. ▼



px	py	pout
F	F	F
F	T	T
T	F	T
T	T	F

```
CHAN OF BOOL px, py, e1, e2, e3, e4, e5, e6, e7, e8, pout :

PROC gAND (CHAN OF BOOL x, y, output)
  BOOL dx,dy,do :
  SEQ
    PAR
      x ? dx
      y ? dy
    do := dx AND dy
    output ! do
:

PROC gNOT (CHAN OF BOOL x, output)
  BOOL dx,do :
  SEQ
    x ? dx
    do := NOT dx
    output ! do
:

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

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

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

PAR                          -- main process XOR
  kick (px, py, TRUE, FALSE)
  connect (px, e1, e5)
  connect (py, e2, e6)
  gAND (e5, e4, e7)
  gAND (e3, e6, e8)
  gNOT (e1, e3)
  gNOT (e2, e4)
  gOR (e7, e8, pout)
  basket (pout)
:
```

Figura 7 - Listato per la funzione XOR. Non appena la situazione si fa più complicata, tocchiamo con mano la potenza del parallelismo, dell'Occam e della strategia data-flow adottata!

inizia chiaramente con la dichiarazione dei canali utilizzati come parametri dei due processi, segue poi un PAR contenente le chiamate ai due processi. Come avviene il collegamento delle porte? Guardate le chiamate ai processi o meglio i parametri: il canale d'uscita della porta AND è passato come parametro d'ingresso della porta NOT. In questo modo il processo NOT aspetterà sul canale finché non riceverà il messaggio.

Il processo AND parimenti aspetterà sui due canali d'ingresso finché dall'esterno non gli vengano forniti dei valori booleani, dopodiché calcolatone l'AND, manderà sul canale d'uscita il risultato che sarà ricevuto, guarda caso, proprio dal NOT.

Per far partire la rete aggiungiamo un processo KICK, il cui compito non è niente altro che quello di spedire i due valori booleani all'AND. Per terminare il programma, ricordiamo che per terminare un PAR occore che siano terminati tutti i processi ivi contenuti, occorre anche un processo che assorba l'uscita del NOT, predisporremo perciò il processo BASKET.

Ed ora magia! Invertite all'interno del PAR l'ordine dei processi e tutto funzionerà allo stesso modo! I messaggi seguiranno il flusso stabilito lungo i canali e viaggeranno dal KICK all'AND al NOT fino al BASKET senza alcun problema.

Con questa strategia dell'algoritmo e grazie al meccanismo di scheduling del Transputer, sono i dati stessi a sequenziare l'ordine di esecuzione dei processi, in una sorta di data-flow.

In questo elementarissimo caso la rete è proprio sequenziale ma potete pensare facilmente che lo stesso meccanismo valga per un numero qualsiasi di porte. In quel caso, a meno di non avere costruito una catena, vi saranno effettivamente un certo numero di processi indipendenti che possono essere eseguiti non appena sono disponibili i dati sui canali, mentre gli altri in PAR attenderanno, senza consumare tempo (grazie Inmos!), finché i messaggi non gli saranno passati sui propri canali di ingresso.

Il programma principale sarà quindi costituito da una sfilza di chiamate ai processi per le varie porte, poste in qualsiasi, sì proprio qualsiasi, ordine. Bisognerà soltanto aver cura di connettere i canali in modo appropriato.

Un esempio più complesso: l'OR esclusivo

Dopo aver visto una realizzazione di porta NAND, tralasciamo quella della porta NOR perché basta sostituire il processo AND con quello OR, facciamo un ultimo esempio di rete logica, un tantino più complessa. In figura 6 trovate il circuito XOR con la relativa tavola della verità e l'implementazione Occam, secondo il nostro risolutore. Dobbiamo però risolvere una piccola complicazione: il fatto che lo stesso segnale va in ingresso a due porte. Come realizzare questa rete? Semplicemente implementando un processo CONNECT ad un ingresso e due uscite il cui compito è solo quello di replicare ciò che riceve dal canale d'ingresso sui due canali d'uscita. Come vedete la procedura non è niente altro che un SEQ della ricezione sull'ingresso e della spedizione del valore booleano sui canali di uscita.

Il programma main è l'insieme delle chiamate dei due NOT, i due AND, i due CONNECT e l'OR finale. Aggiustati bene i canali fra le porte, cioè connesse opportunamente le porte stesse per fare un paragone diretto con l'hardware, potete tentare di permutare i 9 processi nei 362880 ordini possibili e vi accorgerete che i segnali logici troveranno sempre la strada corretta. Potenza dell'Occam!

Per questa volta possiamo fermarci, nella prossima puntata implementeremo le porte NAND e NOR in maniera più maneggevole in modo da poter lavorare con circuiti NAND-NAND o NOR-NOR invece che con gli AON (AND OR NOT), come quasi sempre avviene nelle realizzazioni pratiche, e introdurremo il generatore di clock che insieme al registratore di livelli ci permetterà di costruire sulle basi del risolutore di espressioni booleane, un primo rudimentale simulatore. MC

Bibliografia

D. Pountain, D. May ***«A tutorial introduction to Occam programming»,*** BSP, 1988.
H. Mano, ***«Digital Design»,*** Prentice-Hall, 1989.