Un simulatore parallelo di circuiti elettronici

Circuiti sequenziali

di Giuseppe Cardinale Ciccotti

Nei precedenti appuntamenti con il progetto che stiamo portando avanti, la costruzione di un simulatore parallelo di circuiti digitali, abbiamo puntualizzato alcuni concetti base e abbiamo visto qual è la filosofia di base del nostro impianto simulatorio. Tuttavia ci siamo limitati alla simulazione di circuiti soltanto combinatori per semplicità. Occorre invece, a questo punto, introdurre altri nuovi concetti e le relative implementazioni per poter simulare circuiti sequenziali, diventa perciò indispensabile collegare i risultati ottenuti dagli elementi circuitali con il momento in cui vengono prodotti dalle porte

Parallelismo, determinismo e correttezza

L'impostazione data-flow che caratterizza l'implementazione del simulatore che stiamo descrivendo, non fa nessuna ipotesi sul momento in cui vengono eseguite le procedure e quindi prodotti i relativi risultati. La correttezza del programma è assicurata dal fatto che una procedura è attivata soltanto quando sono pronti i dati in ingresso ad essa. Per un circuito combinatorio ciò può andare molto bene in quanto ci interessa soltanto il risultato prodotto in funzione degli ingressi che forniamo.

Tuttavia quando vogliamo simulare circuiti che cambiano stato con una certa sequenza, per esempio un circuito con un generatore di clock connesso ad una rete combinatoria, allora è necessario considerare l'ordine con cui i risultati vengono prodotti per metterli in relazione agli ingressi che li hanno determinati. Bisogna fare in modo di mettere a punto un meccanismo tale per cui i risultati siano sincronizzati con i rispettivi ingressi.

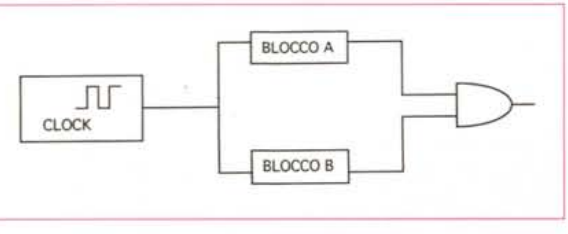

Figura 1 - Schema ideale per illustrare una generica situazione di indeterminatezza.

Un'altra importante ragione per la quale bisogna prendere tali accorgimenti è la correttezza dei risultati; in un ambiente parallelo la correttezza è molto più difficile da assicurare che in un ambiente seriale dove questa non dipende dal momento in cui vengono eseguite le singole istruzioni ma solo dalla relazione procedurale fra di esse. In un sistema parallelo come quello che stiamo impiantando dove tutte le istruzioni (processi) hanno la stessa probabilità di venire schedulate ed eseguite sempreché i dati in ingresso siano pronti, dobbiamo necessariamente far in modo che, qualsiasi ordine di scheduling venga eseguito, il risultato sia corretto. Se alcune sequenze di scheduling non assicurano la correttezza è necessario che non si verifichino.

È chiaro che nel caso combinatorio puro dei precedenti articoli, questa evenienza non si possa mai verificare perché calcoliamo soltanto un risultato e non una sequenza. È come se nel caso seriale eseguissimo una sola iterazione di un ciclo e quindi ottenessimo un solo risultato; volendone ottenere una sequenza potremmo eseguire più cicli. Il risultato sarà comunque corretto perché se una sola iterazione dà risultati corretti, coerentemente, a meno di errori procedurali, l'esecuzione della stessa sequenza di istruzioni più volte deve fornire tutte le volte risultati corretti.

In un programma parallelo strutturato come il nostro, dove a priori non c'è nessun controllo sulla sequenziazione delle istruzioni, immettere sequenze di valori di ingresso, non è detto che produca risultati corretti. Anzi si possono verificare due diverse situazioni: la prima è che esecuzioni successive con stesse sequenze di valori di ingresso producano risultati diversi, la seconda è che si possano verificare situazioni di blocco critico, in cui l'esecuzione si arresta perché i dati non pervengono al processo che quindi non viene eseguito.

Questa seconda ipotesi è sempre in agguato quando si programma con sistemi paralleli ed è il vero spauracchio dei programmatori. Nel nostro caso però l'Occam ci viene in aiuto, non permettendoci all'interno del costrutto par la condivisione di una stessa variabile e così evitiamo una delle principali fonti di stallo.

Per i lettori meno avvezzi alla programmazione parallela la prima ipotesi può sembrare un'eresia, ma purtroppo non lo è; facciamo un esempio per chiarirci meglio le idee: in figura 1 è riportato lo schema funzionale di un generico (alquanto strano) circuito. Nella sua semplicità però ci permette di evidenziare due blocchi indipendenti fra loro le cui uscite sono connesse in ingresso ad una semplice porta AND. Notate pure come le sequenze di ingresso siano le stesse in quanto entrambi i blocchi sono collegati al medesimo generatore di clock. Senza precisare nulla riguardo alla complessità dei blocchi A e B, potrebbero essere molto complicati o al limite soltanto una porta, forse anche uguali fra di loro, facciamo questa semplice osservazione: nel nostro sistema, i processi vengono eseguiti quando sono pronti i dati sui rispettivi ingressi; sia nel blocco A che in quello B ci sarà perlomeno un processo pronto ad essere eseguito non appena riceve i dati dal generatore di clock a monte. Supponiamo che venga eseguito il processo del blocco A e che fornisca i risultati ai processi ad esso connessi, allora almeno uno di questi diventerà pronto ad essere eseguito; supponiamo che venga eseguito, esso a sua volta, fornirà dei risultati che attiveranno altri processi e così via. Supponiamo ancora che vengano eseguiti tutti i processi del blocco A ed invece nessun processo del blocco B, sebbene pronto, venga schedula-

to: in uscita dal blocco A avremo perciò un risultato mentre dal blocco B nessuno.

Se dovessimo calcolare l'uscita del circuito in risposta ad una sola variazione degli ingressi, cioè il generatore di clock eseguisse un solo ciclo, non avremmo nessun problema in quanto saremmo sicuri che dopo aver eseguito tutti i processi del blocco A verrebbero schedulati ed eseguiti i processi del blocco B e quindi otterremmo il risultato corretto. Tuttavia in ingresso al circuito abbiamo connesso un generatore di clock che periodicamente ed inesorabilmente produce una variazione degli ingressi ai processi a cui è collegato. Nel caso peggiore potrebbe accadere che ad ogni ciclo di clock vengano sempre schedulati i processi del blocco A e mai quelli del blocco B. Vale a dire che all'uscita del blocco A si accumulerebbero tutti i risultati dei sucessivi cicli di clock. Come minimo bisognerebbe prevedere un buffer per ogni uscita, in realtà questa evenienza potrebbe verificarsi all'ingresso di qualsiasi porta del circuito e tale soluzione non è certo praticabile.

L'altra considerazione che va fatta, riguarda l'ordine di arrivo dei dati stessi, in quanto è possibile che per determinati circuiti il risultato sia funzione dell'ordine di arrivo dei dati; con il funzionamento appena descritto non è possibile tenere conto di questo vincolo. Si potrebbe ovviare facilmente, associando al valore trasmesso sul canale che connette i componenti anche un altro valore che indica il ritardo secondo un riferimento globale del valore trasmesso. Allora nella procedura che esegue l'operazione elementare bisognerà valutare quale dei dati è arrivato per primo e poi eseguire l'operazione corrispondente, infine sommare il tempo di ritardo del componente stesso.

In definitiva è necessario limitare l'indeterminatezza per assicurare la correttezza di un programma parallelo.

Tempo di ritardo

Qualsiasi componente sia esso una porta oppure un blocco più complesso, una volta ricevuti gli ingressi, fornirà dei risultati stabili dopo un certo tempo che dipende dalla complessità del circuito, dalla tecnologia con cui è realizzato, dal fan-out, cioè dal numero di porte a cui è connessa l'uscita, dalla temperatura e da altri fattori di minore importanza. Questi valori tipici di commutazione sono comunque noti sui data-sheet che descrivono il componente come tempi minimi e massimi necessari alla commutazione. Dalle considerazioni del paragrafo precedente, dobbiamo comunque introdurre questo ritardo nella nostra simulazione al fine di assicurare la fedeltà e la correttezza della simulazione stessa.

Quello che è necessario è un riferimento globale del tempo in modo tale che i tempi calcolati per ciascun componente siano coerenti fra loro; dal punto di vista della programmazione Occam, questo è un piccolo problema perché sappiamo che non possiamo condividere la stessa variabile in un par.

L'introduzione del tempo di ritardo si risolve semplicemente inizializzando un contatore locale al processo non appena sia arrivato almeno un ingresso, il risultato in uscita verrà fornito quando il contatore avrà raggiunto il numero proporzionale al tempo di ritardo del componente stesso.

Per esempio, mettiamo che di utilizzare una porta AND in tecnologia HCMOS, con un tempo di propagazione (tempo di ritardo, tempo di propagazione, tempo di porta, tempo di set-up sono dizioni diverse per indicare lo stesso fenomeno) di 50 ns e fissiamo che il nostro clock globale ci fornisca un impulso ogni 5 ns. Non appena verrà ricevuto un dato su uno degli ingressi, verrà inizializzato un contatore degli impulsi del clock globale, quando questo avrà raggiunto il valore 9 verrà spedito in sincrono con l'impulso successivo, il risultato dell'AND sul canale di uscita.

In effetti è come se campionassimo il comportamento della porta con una frequenza di 5 ns, dal punto di vista concettuale non è poi così sbagliato se considerate che una porta in realtà è un dispositivo analogico, che in qualsiasi condizione fornisce un output, che per nostra convenienza entro certi limiti e secondo determinate condizioni consideriamo costante. Chi dei lettori ha mai visto la traccia sull'oscilloscopio dell'uscita di una qualsiasi porta, avrà di certo notato come il valore stesso oscilli, con continuità, di 0.5 volt rispetto al valore dell'1 o dello 0. Nessun simulatore tuttavia riproduce così fedelmente la realtà, primo perché non è interessante per il funzionamento e in secondo luogo perché sarebbe necessario valutare il comportamento analogico della commutazione con tutto il problema dei transitori della commutazione di stato.

Se è necessario scendere a questo livello di dettaglio esiste tutta un'altra tipologia di prodotti che si occupa di valutare il comportamento fisico del componente in maniera così fine.

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

Figura 4 - Processo AND senza accorgimenti di tipo temporale.

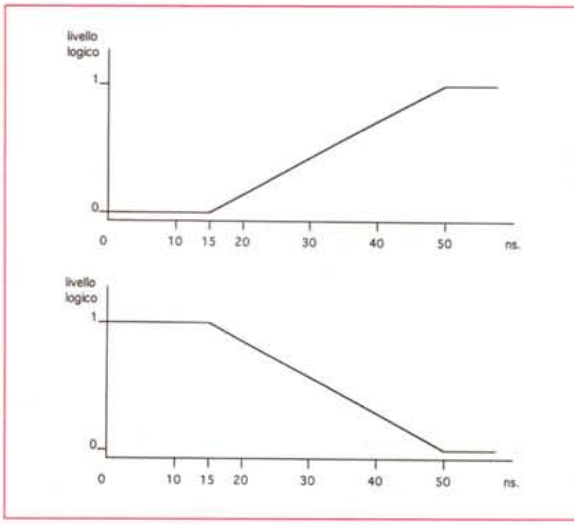

Figura 2 - Curve di commutazione da basso ad alto e da alto a basso di un dispositivo. La commutazione non è mai a gradino.

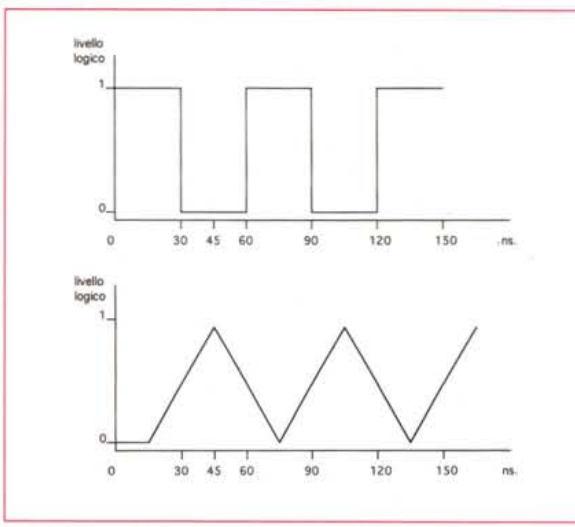

Figura 3 - Glitch di commutazione, dovuti ad una variazione rapida di uno degli ingressi (sopra) di un AND caratterizzato da un tempo di ritardo di 50 ns. In basso è l'uscita non stabile.

Glitch, spike e commutazioni

Anche se non abbiamo intenzione di andare ad «infognarci» nel problema della microsimulazione, tuttavia è ne-

```
PROC gAND (CHAN OF INT16 x, y, output, mclock, off)

  VAL ritardo IS 50 :
  VAL UnknowState IS #AAAA:

  INT16 dx,dy,ck,do,counter,any, mclockCycle :
  BOOL active :

  SEQ
    mclock ? mclockCycle
    active := TRUE
    counter := 0
    WHILE active
      ALT
        x ? dx
          WHILE (((counter*mclockCycle) < ritardo) AND active)
            ALT
              mclock ? ck
                counter := counter + 1
              x ? dx
                counter := 0
              y ? dy
                counter := 0
              off ? any
                active := FALSE
        y ? dy
          WHILE (((counter*mclockCycle) < ritardo) AND active)
            ALT
              mclock ? ck
                counter := counter + 1
              x ? dx
                counter := 0
              y ? dy
                counter := 0
              off ? any
                active := FALSE
        off ? any
          active := FALSE

      IF
        (NOT(dx = Uknowstate)) AND (NOT(dy = Uknowstate))
          do := (dx BITAND dy)
          output ! do
          counter := 0
        TRUE
          SKIP
:
```

Figura 5 - Processo AND che tiene conto del tempo di ritardo della porta ed è sincrono con il master clock. Il processo termina non appena riceve un dato sul canale OFF.

```
PROC MasterClock (CHAN OF[]INT16 mclock, CHAN OF[]INT16 off)

  VAL SimulationCycles IS 100 : --cicli complessivi di simulazione
  VAL mclockCycle IS 5:  --step singolo del master clock in ns
  VAL StopSignal IS #FFFF:  --segnale di arresto della simulazione
  VAL mclockSignal IS #FFFF: --segnale del clock

  INT16 i,counter :

  SEQ
    PAR i = 0 FOR 10   --10 è il numero dei componenti connessi
      mclock [i]! mclockCycle
    counter := 0
    WHILE (counter < SimulationCycles)
      SEQ
        PAR i = 0 FOR 10
          mclock [i]!  mclockSignal
        counter := counter + 1
    PAR i = 0 FOR 10
      off [i]!  StopSignal
    --off è un array di canali usato per terminare i processi
```

Figura 6 - Processo master clock. Questo è l'orologio utilizzato per sincronizzare tutti i componenti del simulatore. È usato anche per arrestare l'esecuzione della simulazione tramite l'array di canali OFF.

```
PROC ClockGen (CHAN OF INT16 clock, off,output)

  VAL DutyCycle IS 100:  --step singolo del clock in ns
  VAL HighClock IS #FFFF: --segnale del clock alto
  VAL LowClock IS #FFFF: --segnale del clock basso

  INT16 counter,ck,mclockCycle,any:
  BOOL active,clockstate:

  SEQ
    mclock ? mclockCycle
    counter := 0
    active := TRUE
    clockstate := FALSE
    WHILE active
      WHILE (active AND ((counter*mclockCycle) < DutyCycle))
        ALT
          mclock ? ck
            counter := counter +1
          off ? any
            active := FALSE
        IF
          clockstate
            output ! HighCLock
          NOT(clockstate)
            output ! LowClock
        clockstate := NOT(clockstate)
        counter := 0

  :
```

Figura 7 - Processo del generatore di clock. Anch'esso è subordinato al master clock e termina non appena riceve un segnale sul canale OFF. ►

cessario tenere presente almeno un'evenienza che può verificarsi durante la commutazione di un dispositivo e cercare di simularla in modo fedele. Poniamoci il problema di vedere cosa succede se gli ingressi di una porta, per esempio dell'AND del paragrafo precedente non rimangono stabili per tutto il tempo di commutazione della porta. Supponiamo per semplicità che un ingresso dell'AND rimanga stabile per tutto il tempo necessario mentre l'altro vari ogni 30 ns.

Dato che il tempo necessario alla commutazione è 50 ns, la porta non fa in tempo a fornire un'uscita stabile, tuttavia se guardate la figura 2 potete vedere come la commutazione dalla tensione associata a un livello all'altra non è istantanea (la commutazione istantanea come si dice a gradino è un'idealizzazione di una curva il cui tempo di salita è piccolo rispetto agli altri tempi in gioco), ma occupa un certo tempo, in figura posto pari a 35 ns. Perciò la porta dopo 15 ns inizia a commutare e dopo altri 35 ns stabilizza l'uscita. Se uno degli ingressi però varia dopo 30 ns, la porta inizia a commutare e dopo 15 ns deve ricommutare verso l'altro stato, supponendo che le curve da livello basso ad alto e da alto a basso siano uguali (non sempre è così), se attaccassimo all'uscita della porta un oscilloscopio vedremmo qualcosa di simile alla forma d'onda in figura 3. Una serie di oscillazioni molto strette e molto fastidiose chiamate glitch di commutazione, esse si verificano perché non stiamo rispettando le specifiche del dispositivo.

Tuttavia proprio perché molto spesso nei circuiti la fonte di problemi sono questi impulsi spuri, è necessario che il nostro simulatore li preveda.

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

Il problema esposto è molto più frequente di quanto non si creda infatti, basta un po' di rumore, dovuto per esempio a componenti passivi, ad alimentazioni non ben filtrate o a fattori esterni per far oscillare un ingresso e causare glitch. Precisiamo qui che oscillazioni spurie dovute a rumore vengono dette spike per distinguerle dai glitch dovuti come abbiamo visto alla commutazione.

Questo fenomeno rischia poi di diventare drammatico quando si è costretti ad adoperare porte con tempi di commutazione diversi nello stesso circuito, se utilizziamo dispositivi in tecnologia ECL caratterizzati da tempi di commutazione nell'ordine di 5-10 ns, vi rendete conto che un glitch diventa un segnale significativo. In genere si cerca di evitare di connettere direttamente dispositivi con tempi così diversi e chi lo fa, sa quali rischi di malfunzionamento può correre.

Come lo abbiamo esposto il problema dei glitch è però troppo semplice perché questi si possono verificare anche in altre situazioni e la loro ampiezza non è a priori così determinabile perché dipende da molti fattori diversi; allora nei sistemi di simulazione si fa ricorso al cosiddetto stato indeterminato che risolve in certo modo la questione. In parole povere quando non si sa che pesci prendere, per esempio quando gli ingressi di un dispositivo non rimangono stabili per un tempo sufficiente o bisogna determinare lo stato di un dispositivo quando si «accende» il simulatore, si stabilisce, onestamente, di non poter asserire nulla di preciso e ci si mette in uno stato indeterminato. Naturalmente questo stato non ha nessuna rispondenza nella realtà fisica poiché un dispositivo, una porta starà indifferentemente a livello alto oppure basso; tuttavia ciò non è predicibile a priori. Ci sono dei componenti che si pongono di preferenza o vengono forzati in uno stato piuttosto che in un altro per evitare le condizioni di indeterminatezza, questa caratteristica è illustrata dai data-sheet è ed naturalmente facilmente programmabile.

Sincronizzazione globale

Dopo aver evidenziato i problemi naturalmente dobbiamo risolverli: è indispensabile, ci siamo resi conto, sincronizzare le commutazioni sui vari dispositivi tenendo conto dei tempi di ritardo e degli stati indeterminati.

La soluzione è quella di predisporre un «orologio» globale uguale per tutti in modo da sincronizzare tutte le operazioni rispetto a questo riferimento assoluto. Come sappiamo però l'Occam non ci permette di condividere variabili fra processi in par così è necessario ricorrere alla soluzione di passare i segnali in uscita da questo orologio tramite un canale. Naturalmente questo orologio dovrà disporre di tanti canali quante sono i dispositivi nel circuito. Bisognerà poi scegliere quanto vale il ciclo di tale orologio, dovrà necessariamente essere più piccolo del minor tempo in gioco nel circuito.

Dato che tale step ci servirà anche come riferimento per le visualizzazioni delle forme d'onda che ci servono è utile sceglierlo abbastanza piccolo, un buon valore può essere 5 ns.

Vediamo ora il funzionamento di un dispositivo qualsiasi ad esempio di una porta, se ricordate, se non ve lo ricordate guardate figura 4, il listato di una porta AND prevedeva che l'operazione venisse computata non appena erano pronti i dati in ingresso perciò bastava mettere le due ricezioni in par.

Ora invece, partiamo dallo stato indeterminato che è individuato dall'intero binario a 16 bit 1010101010101010 pari a 10922 decimale o AAAA in esadecimale, ci conviene porre 0000 per lo zero logico e FFFF per l'uno logico visto che le operazioni logiche sono sui bit direttamente.

Lo stato indeterminato così sembra abbastanza riconoscibile.

Partire dallo stato indeterminato significa che qualsiasi operazione viene fatta con uno degli ingressi ad AAAA restituisce in uscita AAAA a meno di voler prevedere qualcosa di diverso, cosa che noi per ora non faremo.

L'altra cosa importante è di predisporre un canale per ricevere i segnali dal «master clock» e di conseguenza subordinare tutte le attività alla ricezione di questo segnale. Si tratterà perciò di attendere finché almeno uno degli ingressi abbia ricevuto un dato, allora campionare i segnali del master clock finché il contatore non raggiunge il tempo di ritardo della porta, quindi spedire il risultato sull'uscita.

Bisogna tuttavia stare attenti anche agli ingressi perché se prima che avvenga la commutazione, viene ricevuto un altro dato bisogna rinizializzare il contatore e mettere l'uscita in stato indefinito. Come si vede viene prodotto un dato in uscita solo quando c'è un cambiamento di stato effettivo, si potrebbe anche spedire lo stato ad ogni ciclo del master clock, anzi sarebbe più corretto dal punto di vista del campionamento, tuttavia questo ci farebbe consumare del tempo prezioso inutilmente, visto che i dati non cambiano, perciò spediremo i dati a valle solo quando cambiano da livello alto a basso o a stato indefinito e viceversa.

In figura 5 trovate il listato della porta AND così modificato ed in effetti, nostante abbiamo rivoluzionato concettualmente il nostro simulatore, potete vedere come il codice della porta non sia poi così complicato.

In figura 6 trovate il listato del processo master clock, molto semplice l'unica cosa furba da notare è che tutte le trasmissioni sono in par in modo tale che tutti i processi ricevano il medesimo clock prima che il master possa mandare un altro segnale. Purtroppo questo processo master clock, così com'è fatto non può essere un processo di libreria perché a priori non si sa quanti processi devono essere ad esso collegati, alternative ce ne sono, ma per ora ci accontenteremo di costruirlo volta per volta visto che il lavoro non è grave.

In ultimo in figura 7, trovate il listato del processo che simula un generatore di clock: è veramente elementare non fa niente altro che campionare il master clock e con la sua frequenza cambiare periodicamente lo stato dell'uscita, supponiamo che parta da uno stato di zero logico, senza alcuna perdita di generalità.

Conclusioni

Abbiamo introdotto dei concetti veramente importanti rivoluzionando il sistema che avevamo impiantato finora, tuttavia abbiamo fatto un grande passo in avanti, in quanto possiamo d'ora in poi lavorare con i circuiti sequenziali. Nei prossimi appuntamenti introdurremo componenti un po' più complessi delle semplici porte e inizieremo a fare delle applicazioni più probanti. Puntualizziamo che il simulatore è in fase di costruzione mentre scriviamo perciò eventuali soluzioni anche soltanto teoriche proposte dai lettori saranno bene accette e discusse su queste pagine. A risentirci.

MC