C

APPENDICE

II Nozioni di base di logica Design

Ho sempre amato quella parola,
Boolean.

Claude Shannon
IEEE Spectrum, Lapril 1992
(Di Shannon tesi di laurea è emerso che l'algebra inventata da George Boole nel 1800 potrebbe rappresentare il funzionamento di interruttori elettrici.)

C.1 Introduzione

Questa appendice fornisce una breve discussione le basi della progettazione logica. Essa non sostituisce un corso di progettazione logica, né le permetterà di progettare importanti sistemi logici di lavoro. Se si dispone di poca o nessuna per la progettazione logica, tuttavia, questa appendice fornisce di base sufficienti a comprendere tutto il materiale in questo libro. Inoltre, se si sta cercando di capire alcune delle motivazioni che stanno dietro come i computer sono implementati, questo materiale servirà da utile introduzione. Se la vostra curiosità è eccitato, ma non sen vien satollo questa appendice, i riferimenti alla fine fornire una serie di fonti di informazioni aggiuntive.

Sezione C.2 introduce gli elementi di base della logica, vale a dire, *cancelli*.Sezione C.3 utilizza questi elementi per stabilire semplice *combinatoria* sistemi di logica, che non contengono memoria. Se avete avuto qualche esperienza con logica o sistemi digitali, si sarà probabilmente familiarità con il materiale contenuto in queste prime due sezioni. Sezione C.5 viene illustrato come utilizzare i concetti di punti C.2 e C.3 per la progettazione di un ALU per il processore MIPS. Sezione C.6 mostra come fare un sommatore veloce,

e può essere tranquillamente ignorato se non siete interessati a questo argomento. Sezione C.7 è una breve introduzione al tema del clock, che è necessario per discutere di come il lavoro di memoria elementi. Sezione C.8 introduce elementi di memoria, e la Sezione C.9 estende a concentrarsi sulle memorie ad accesso casuale, ma descrive sia le caratteri-stiche che sono importanti per capire come vengono utilizzati nel Capitolo 4, e lo sfondo che motiva molte delle aspetti della progettazione di memoria gerarchia nel Capitolo 5. Sezione C.10 descrive la progettazione e l'uso di macchine a stati finiti, che sono blocchi logici sequenziali. Se avete intenzione di leggere l'Appendice D, è necessario comprendere a fondo il materiale nelle sezioni C.2 attraverso C.10. Se avete intenzione di leggere solo il materiale per il controllo nel Capitolo 4, è possibile scorrere le appendici, tuttavia, si dovrebbe avere una certa familiarità con tutto il materiale eccetto la sezione C.11. Sezione C.11 è destinato a coloro che vogliono una più profonda comprensione delle metodologie e dei tempi di clock. Spiega le basi di come edge-triggered opere clock, introduce un altro schema di clock, e descrive brevemente il problema di sincronizzare gli ingressi asincroni.

In tutta questa appendice, in cui è appropriato, anche i segmenti di dimostrare come la logica possa essere rappresentato in Verilog, che si introduce nella sezione C.4. Un tutorial più approfondite e complete Verilog appare altrove sul CD.

C.2 Gates, tavole di verità, ed equazioni logiche

L'elettronica all'interno di un moderno computer sono *digitale*.Elettronica digitale funzionare con due soli livelli di tensione di interesse: una tensione alta e bassa tensione. Tutti i valori di tensione gli altri sono temporanei e si verificano durante la transizione tra i valori. (Come vedremo più avanti in questa sezione, un trabocchetto possibile nella progettazione digitale è il campionamento di un segnale quando chiaramente non sia alta o bassa.) Il fatto che i computer sono digitali è anche una delle ragioni principali che usano numeri binari, dal momento che un sistema binario partite l'astrazione sottostante inerente l'elettronica. Nelle famiglie logiche diverse, i valori e le relazioni tra i due valori di tensione diversi. Così, invece di fare riferimento ai livelli di tensione, si parla di segnali che sono (logicamente) vero, o 1, o sono **affermato**;o segnali che sono (logicamente) false, o 0, o sono **deasserito**.I valori
0 e 1 sono chiamati *complementi* o *inversi* uno dall'altro.

asserted segnale LaSIgnal che è (logicamente) vero, o 1.

segnale deasserito Un segnale che è (logicamente) falso, o 0.

Blocchi logici vengono classificati in due tipi, a seconda che essi contengono memoria. I blocchi senza memoria sono chiamati *combinational;* l'uscita di un blocco combinatorio dipende solo dalla corrente di ingresso. In blocchi con memoria, le uscite possono dipendere entrambi gli ingressi e il valore memorizzato nella memoria, che è chiamata *stato* del blocco logico. In questa sezione e la successiva, ci concentreremo

solo **cologica mbinational**. Dopo l'introduzione di elementi di memoria diversi nella Sezione C.8, descriveremo come **sequenziale logica**, Che è la logica tra cui lo stato, è stato progettato.

cologica mbinational Un sistema logico cui blocchi non contengono memoria e quindi calcolare l'uscita stessa data lo stesso ingresso.

logica sequenziale LagrOUP elementi logici che contengono memoria e quindi il cui valore dipende ingressi e la corrente contents della memoria.

Truth Tabelle

Poiché un blocco di logica combinatoria contiene alcuna memoria, può essere completamente specificato definendo i valori delle uscite per ogni possibile insieme di valori di ingresso. Tale descrizione è normalmente dato come *truth tavolo*. Per un blocco logico con *n*ingressi, ci sono due^{n}voci nella tabella di verità, poiché ci sono che molte combinazioni possibili di valori di ingresso. Ciascuna voce indica il valore di tutte le uscite di questa combinazione particolare ingresso.

Truth Tabelle

ESEMPIO

Consider una funzione logica con tre ingressi, *La*,*B*, E *C*, E tre uscite, *D*,*E*,e *F*.La funzione è definita come segue: *D*è vero se almeno un ingresso è vera, *E*è vera se esattamente due ingressi sono vere, e *F*è vero solo se tutti e tre gli ingressi sono vere. Mostra la tabella di verità per questa funzione.

RISPOST

La tabella di verità conterrà 23 =8 voci. Eccolo:

Ingres			Uscite		
L	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Ttabelle uth può descrivere completamente qualsiasi funzione logica combinatoria, ma crescono in dimensioni rapidamente e può non essere facile da capire. A volte si vuole costruire una funzione logica, che sarà pari a 0 per le combinazioni di input che, e usiamo una scorciatoia di specificare solo le voci della tabella di verità per le uscite non nulle. Questo approccio viene utilizzato nel capitolo 4 e l'Appendice D.

Algebra booleana

Lanother approccio è di esprimere la funzione logica con equazioni logiche. Questo viene fatto con l'uso di *BoOlean algebra* (Dal nome di Boole, un 19 ° secolo mate-matician). In algebra booleana, tutte le variabili hanno i valori 0 o 1 e, in formulazioni tipiche, ci sono tre operatori:

- L'operatore OR è scritto come +, come in *La*+*B*. Il risultato di un operatore OR è
  1 se una delle variabili è 1. L'operazione di OR è chiamato anche *logico somma*,
  poiché il risultato è 1 se uno dei due operandi è 1.
- L'operatore AND viene scritto come ·, come in *La*·*B*. Il risultato di un operatore AND è 1 solo se entrambi gli ingressi sono 1. L'operatore AND viene anche chiamato *logico prodotto*, Dal momento che il suo risultato è 1 solo se entrambi gli operandi sono 1.
- L'operatore unario NON è scritto come $\overline{La}$. Il risultato di un operatore NOT è 1 solo se l'ingresso è 0. Applicando l'operatore non ad un valore logico risultati in una inversione o negazione del valore (cioè, se l'ingresso è 0 l'uscita è 1, e viceversa).

Ci sono diverse leggi dell'algebra booleana che sono utili per manipolare equazioni logiche.

- Iodentity legge: *La*+0=*La*e *La*·1=*La*.
- ZerO e uno leggi: *La*+1=1 e *La*·0=0.
- INVERSE leggi: $\overline{La}$+*La*=1 e *La*·$\overline{La}$=1.
- Coleggi mmutative: *La*+*B*=*B*+*La*e *La*·*B*=*B*·*La*.
- Laleggi ssociative: *La*+(*B*+*C*)=(*La*+*B*)+*C*e *La*·(*B*·*C*)=(*La*·*B*) · *C*.
- Leggi distributive: *La*·(*B*+*C*)=(*La*·*B*)+(*La*·*C*) E
  La+(*B*·*C*)=(*La*+*B*) · (*La*+*C*).

IoInoltre n, ci sono due altri teoremi utili, chiamati DeMorgan le leggi, che vengono discussi in modo più approfondito negli esercizi.

Unyset di funzioni logiche può essere scritta come una serie di equazioni con un'uscita sulla sinistra di ciascuna equazione e una formula basata variabili e tre operatori sopra sul lato destro.

Logic Equazioni

ESEMPIO

Mostra le equazioni logiche per le funzioni logiche, *D*,*E*,e *F*,descritto nel precedente esempio.

RISPOSTA

Qui 's l'equazione per *D:*

$$D=La+B+C$$

*F*è altrettanto semplice:

$$F=La\cdot B\cdot C$$

*E*è un po 'difficile. Pensate in due parti: ciò che deve essere vero per *E*to essere vero (due dei tre ingressi deve essere vero), e ciò che non può essere vero (tutti e tre non può essere vero). Così siamo in grado di scrivere *E*come

$$E=((La\cdot B)+(La\cdot C)+(B\cdot C))\cdot\overline{(La\cdot B\cdot C)}$$

We può anche derivare *E*by rendersi conto che *E*è vero solo se esattamente due degli ingressi sono vere. Allora possiamo scrivere *E*come un OR di tre termini possibili che hanno due ingressi veri e un ingresso false:

$$E=(La\cdot B\cdot\overline{C})+(La\cdot C\cdot\overline{B})+(B\cdot C\cdot\overline{La})$$

Provzione che queste due espressioni sono equivalenti è esplorato negli esercizi.

Ion Verilog, si descrive la logica combinatoria, quando possibile utilizzando l'istruzione di assegnazione, che è descritta a partire da pagina C-23. Siamo in grado di scrivere una definizione di *E*utilizzando l'operatore OR esclusivo Verilog come assegnare E = (A ^ B ^ C) * (A + B + C) * (A * B * C), che è un altro modo per descrivere questa funzione. *D*e *F*hanno anche semplici rappresentazioni, che sono proprio come il corrispondente codice C: D = A | B | C e F = A & B & C.

Gates

gate Un dispositivo che implementa funzioni logiche di base, come AND o OR.

Logica blocchi sono costruiti **gAtes** che implementano funzioni logiche di base. Per esempio, una porta AND implementa la funzione AND e una porta OR implementa la funzione OR. Dal momento che sia gli operatori AND e OR sono commutative e associative, un AND o una porta OR può avere più ingressi, con l'uscita pari alle AND e OR di tutti gli ingressi. La funzione logica non è implementata con un inverter che ha sempre un singolo ingresso. La rappresentazione standard di questi tre blocchi logici costruzione è mostrata in Figura C.2.1.

Invece di disegnare inverter in modo esplicito, una pratica comune è quella di aggiungere "bolle" per gli ingressi o le uscite di una porta per causare il valore logico su tale linea d'ingresso o di linea da invertire. Ad esempio, la Figura C.2.2 mostra il diagramma logico
la funzione $La+B$, Utilizzando gli inverter esplicite per gli ingressi sinistro e bolliti e
uscite a destra.

Unfunzione y logica può essere costruito utilizzando porte AND, OR porte, e l'inversione, per cui molti degli esercizi che danno la possibilità di provare l'attuazione di alcune funzioni logiche comuni con cancelli. Nella prossima sezione, vedremo come l'implementazione di una funzione logica può essere costruito utilizzando questa conoscenza.

Porta NOR Lan invertito Porta OR.

NPorta AND Lan invertito Porta AND.

IoInfatti n, tutte le funzioni logiche possono essere costruiti con un solo tipo sola porta, se il cancello è invertente. Le due porte invertenti comuni sono chiamati **NOR** e **NE** e correspond a invertito OR e porte AND, rispettivamente. Porte NAND e NOR sono chiamati *universale*,poiché ogni funzione logica può essere costruito usando questo tipo di gate. Gli esercizi esplorare ulteriormente questo concetto.

Contro llare Ynoi stessi

Arposta i seguenti due espressioni logiche equivalenti? In caso contrario, trovare una impostazione delle variabili per dimostrare che non lo sono:

- $(La \cdot B \cdot \overline{C}) + (La \cdot C \cdot B) + \overline{(\overline{B} \cdot C \cdot La)}$
- $B \cdot \overline{(La \cdot \overline{C} + C \cdot La)}$

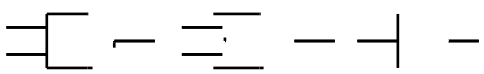


FIGURA C.2.1 disegno standard per una porta AND, OR e un inverter, mostrata da sinistra a destra. I segnali a sinistra di ogni simbolo sono gli ingressi, mentre l'uscita appare sulla destra. Le porte AND e OR hanno entrambi due ingressi. Gli inverter hanno un singolo ingresso.

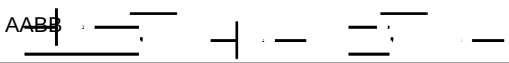


FIGURA C.2.2 attuazione porta logica di *La*+*Butilizzando* inverte esplicite per gli ingressi sinistro e bolle e le uscite a destra. Questa funzione logica può essere semplificata $La \cdot B$o in

Verilog, A & ~ B.

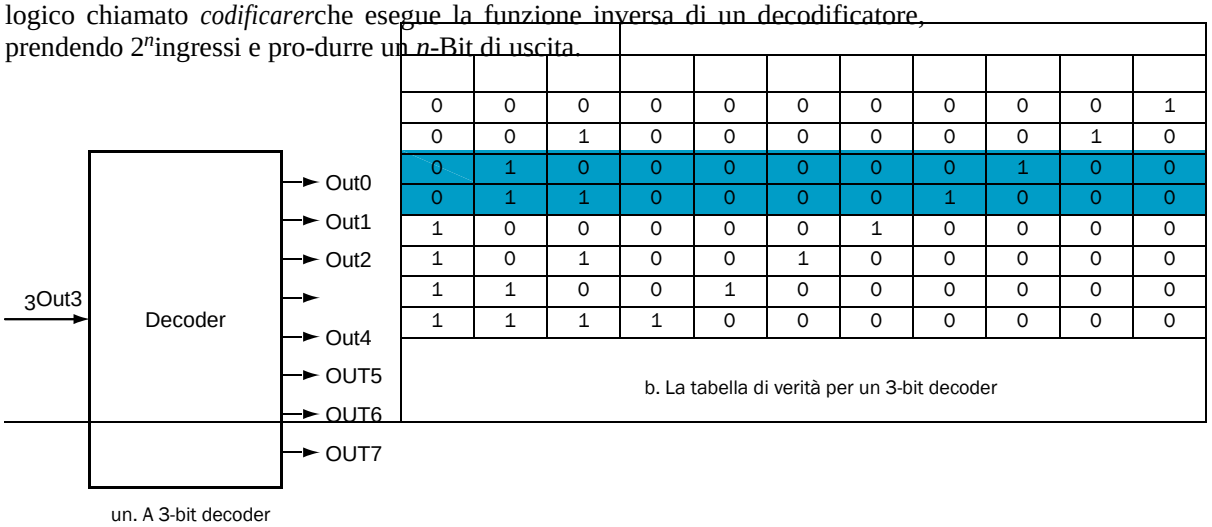
C.3 Logica combinatoria

Ion questa sezione, guardiamo un paio di isolati più grandi logici di costruzione che usiamo pesantemente, e si discute il disegno di logica strutturata che può essere eseguita automaticamente da una equazione logica o tabella di verità da un programma di traduzione. Infine, si discute la nozione di un array di blocchi logici.

Decoder

Un blocco di codice che useremo nella costruzione di componenti più grandi è un **decoder**.Il tipo più comune di decoder ha un *n*Bit di ingresso e 2^nuscite, in cui si afferma una sola uscita per ogni combinazione di ingresso. Questo decoder traduce il *n*Bit ingresso in un segnale che corrisponde al valore binario della *n*Bit di ingresso. Il
uscite sono quindi solitamente numerati, per esempio, Out0, Out1,. . . , Out2^n-1.Iof il valore di
l'ingresso è *Io*, Allora Out*Io*wmalato essere vero e tutte le altre uscite sarà false. Figura C.3.1
mostra un 3-bit decoder e la tabella di verità. Questo è chiamato un decodificatore *3-to-8 decoder* poiché vi sono tre ingressi e 8 (23) uscite. Vi è anche un elemento logico chiamato *codificarer*che esegue la funzione inversa di un decodificatore, prendendo 2^ningressi e pro-durre un *n*-Bit di uscita.

decoder Una logica blocco che ha un *n*Bit di ingresso e 2^nuscite, in cui si afferma una sola uscita per ogni combinazione di ingresso.



un. A 3-bit decoder

0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

b. La tabella di verità per un 3-bit decoder

Figura C.3.1 A 3-bit decoder dispone di 3 ingressi, chiamate 12, 11, e 10, e 23 = 8 uscite, chiamati a Out0 OUT7. Solo l'uscita corrispondente al valore binario di ingresso è vera, come mostrato nella tabella di verità. L'etichetta 3 sul ingresso al decodificatore dice che il segnale di ingresso è di 3 bit.

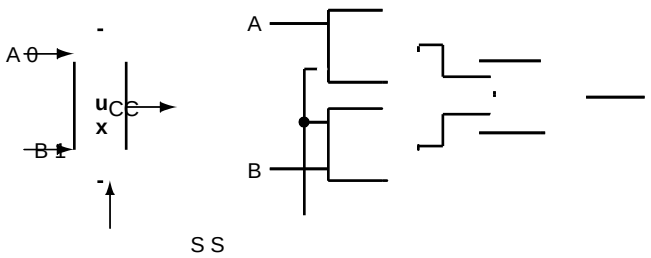


FIGURA C.3.2 a due ingressi multiplexor sulla sinistra e la sua applicazione con porte sulla destra. Il multiplexer dispone di due ingressi (dati*La*e *B*), Che sono etichettati *0*e *1*,ed un selettore d'ingresso (*S*), Nonché un'uscita *C*. IoTTUAZIONE multiplexer in Verilog richiede un lavoro un po 'più, soprattutto quando sono più larghe di due ingressi. Mostriamo come farlo a partire da pagina C-23.

Multiplexer

valore del selettore Chiamato anche control valore. Il segnale di controllo che permette di selezionare uno dei valori di ingresso di un multiplexer come l'uscita del multiplexer.

Una funzione logica di base che usiamo molto spesso nel capitolo 4 è il *multiplexer*. Un multiplexer potrebbe essere più propriamente chiamato *selector*,poiché la sua uscita è uno degli ingressi che viene selezionato da un controllo. Si consideri il multiplexer a due ingressi. Il lato sinistro della Figura C.3.2 mostra questa multiplexor ha tre ingressi: due valori di dati e una **selettore (orcontrol) Valore**. Il valore del selettore determina quale degli ingressi diventa l'output. Possiamo rappresentare la funzione logica calcolata da due in_put multiplexor, mostrato in forma cancello sul lato destro della Figura C.3.2, come $C=(La \cdot S)+(B \cdot S)$.

Multiplexors può essere creato con un numero arbitrario di ingressi dati. Quando ci sono solo due ingressi, il selettore è un singolo segnale che seleziona uno degli ingressi se è vero (1) e l'altro se è falso (0). Se ci sono *n*input di dati, ci sarà devono essere $\lceil \log_2 n \rceil$ ingressi di selezione. In questo caso, il multiplexer consiste essenzialmente tre parti:

1. Un decoder che genera *n*segnali, ciascuno indica un valore diverso di ingresso
2. Un array di *n*Porte AND, ciascuna combinazione degli ingressi con un segnale dal decodificatore
3. Un unico grande porta OR che incorpora le uscite delle porte AND

To associare gli ingressi con valori di selettore, spesso etichettare gli ingressi dati numericamente-mente (ad esempio, 0, 1, 2, 3, ..., *n*- 1) e interpretare i dati in ingresso di selezione come un binario nterra d'ombra. A volte, ci avvaliamo di un multiplexer con non decodificati i segnali di selezione.

Multiplexors sono facilmente rappresentato combinationally in Verilog utilizzando *se* eXpressions. For multiplexer grandi, *caso* dichiarazioni sono più convenienti, ma bisogna fare attenzione per sintetizzare logica combinatoria.

Two livello Logica e PLA

Las ha sottolineato nella sezione precedente, ogni funzione logica può essere implementata solo con funzioni AND, OR e NOT. In effetti, un risultato molto più forte è vero. Ogni funzione logica può essere scritta in una forma canonica, dove ogni input è una variabile reale o integrate e ci sono solo due livelli di porte- uno è E e l'altra o, con una possibile inversione del risultato finale. Tale rappresentazione è chiamato *two a livello di rappresentanza*, e ci sono due forme, chiamate **somma dei prodotti** e *prodotto of somme*.Una somma di prodotti di rappresentazione è una somma logica (OR) di prodotti (termini con l'operatore AND), un prodotto di somme è esattamente l'opposto. Nel nostro precedente esempio, abbiamo avuto due equazioni per l'uscita *E:*

somma dei prodotti Una forma di rappresentazione logica che utilizza una logica somma (OR) di prodotti (termini uniti mediante l'operatore AND).

$$E=((La\cdot B)+(La\cdot C)+(B\cdot C))\cdot(\overline{La\cdot B\cdot C})$$

e

$$E=(La\cdot\overline{B}\cdot C)+(La\cdot C\cdot\overline{B})+(B\cdot \mathrm{C}\cdot \mathrm{A})\ \overline{\ }$$

Thiequazione di s seconda è in una somma di prodotti di forma: ha due livelli di logica e le inversioni solo sono variabili individuali. La prima equazione ha tre livelli di logica.

Elaborazione: We può anche scrivere *E*come prodotto di somme:

$$E=\overline{(\overline{La}+B+C)\cdot(\overline{La}+C+B)\cdot(B+\mathrm{C}+La)}$$

To derivare questa forma, è necessario utilizzare *DeMorgan teoremi*, Che sono discusse negli esercizi.

Ion questo testo, utilizzare la somma di prodotti di forma. È facile vedere che qualsiasi funzione logica può essere rappresentata come una somma di prodotti da costruzione di una tale rappresenta-zione dalla tabella di verità per la funzione. Ogni voce della tabella di verità per cui la funzione è vera corrisponde a un termine prodotto. Il termine prodotto consiste in un prodotto logico di tutti gli ingressi o complementi degli ingressi, a seconda se la voce nella tabella di verità ha un 0 o 1 corrispondente a questa variabile. La funzione logica è la somma logica dei termini di prodotto in cui la funzione è vera. Questo è più facilmente visibile con un esempio.

Somma dei prodotti

ESEMPIO

Mostra la somma di prodotti rappresentanza per la seguente tabella di verità per *D*.

Ingressi di uscita ABCD			
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

RISPOSTA

Ci sono quattro termini di prodotti, poiché la funzione è vera (1) per quattro combinazioni di input diversi. Questi sono:

il primo
logica di fase della matrice logica programmabile (PLA).

logica di matrice programmabile (PLA) Un elemento logico-strutturato composto da un insieme di ingressi e complementi d'ingresso corrispondente e due fasi della logica: i primi termini di generazione di prodotti degli ingressi e complementi d'ingresso, e la seconda generazione termini di somma dei prospetti informativi. Quindi, PLA implementare funzioni logiche come somma di prodotti.

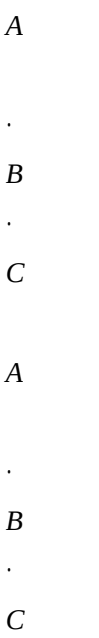
mintermini Anche detto termini di prodotto. Un insieme di ingressi logici uniti da congiunzione (AND operazioni), i cui termini di prodotto costituiscono

$L \quad \cdot C)+(La \cdot B \cdot C)+(La \cdot B \cdot \overline{C})+(\overline{La} \cdot B \cdot C)$

Nota che solo le voci della tabella di verità per cui la funzione è vera generare termini dell'equazione.

We possibile utilizzare questo rapporto tra una tabella di verità e su due livelli rap-presentazione per generare un cancello a livello di attuazione di qualsiasi serie di funzioni logiche. Un insieme di funzioni logiche corrisponde a una tabella di verità con colonne di output, come abbiamo visto nell'esempio a pagina C-5. Ogni colonna di output rappresenta una funzione diversa logica, che può essere costruito direttamente dalla tabella di verità.

La somma di prodotti rappresentazione corrisponde a un comune strutturato-logica di implementazione chiamata **programmabile logica di array (PLA)**.Un PLA ha una serie di ingressi e complementi ingresso corrispondente (che può essere implementato con una serie di invertitori), e due fasi di logica. La prima fase è una matrice di porte AND che formano un insieme di **termini di prodotto** (A volte chiamato **mintermini**); Ogni termine prodotto può contenere qualsiasi degli ingressi o loro complementi. La seconda fase è una matrice di porte OR, ciascuna delle quali forma una somma logica di qualsiasi numero di termini di prodotto. Figura C.3.3 mostra la forma di base di un PLA.



Quindi, possiamo scrivere la funzione di *D*come somma di questi termini:

$D=(La \cdot B$

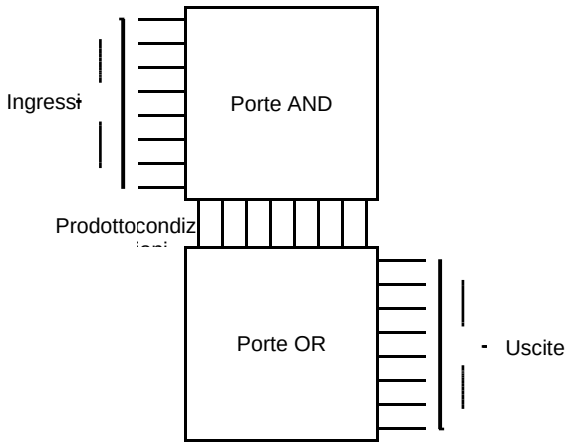


FIGURA C.3.3 La forma di base di una PLA costituito da una matrice di porte AND seguita da una matrice di porte OR. Ogni voce della AND gate array è un termine prodotto costituito da un numero di ingressi o ingressi invertiti. Ogni voce della OR gate array è un termine somma composta da un numero qualsiasi di questi termini di prodotto.

Un PLA può attuare direttamente la tabella di verità di un insieme di funzioni logiche con più ingressi e uscite. Poiché ogni voce in cui l'uscita è vera richiede un termine prodotto, ci sarà una riga corrispondente nella PLA. Ogni uscita corrisponde a una riga potenziale di porte OR nel secondo stadio. Il numero di porte OR corrisponde al numero di voci della tabella di verità per cui l'uscita è vera. La dimensione totale di una PLA, come quello mostrato in Figura C.3.3, è pari alla somma delle dimensioni del AND gate array (chiamato *E piano*) E la dimensione della matrice porta OR (chiamato *O in aereo*). Guardando la Figura C.3.3, possiamo vedere che la dimensione della AND gate array è uguale al numero di ingressi volte il numero di termini di prodotto, e la dimensione della matrice porta OR è il numero di uscite per il numero dei termini di prodotto.

Il PLA ha due caratteristiche che contribuiscono a rendere un modo efficace per attuare una serie di funzioni logiche. Prima, solo le voci della tabella di verità che producono un valore reale per almeno una uscita qualsiasi porte logiche ad essi associati. In secondo luogo, ogni differente termine prodotto avrà una sola voce nel PLA, anche se il termine prodotto viene utilizzato in più uscite. Vediamo un esempio.

PLA

Consider l'insieme delle funzioni logiche definite nell'esempio a pagina C-5. Mostra un'implementazione PLA di questo esempio per *D*,*E*,e *F*.

RISPOST

Suoe è la tabella di verità abbiamo costruito in precedenza:

Ingressi Uscite *ABCDEF*					
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Since vi sono sette termini prodotto unico con almeno un valore true nella sezione di uscita, ci saranno sette colonne e piano. Il numero di righe nel piano E è di tre (dato che ci sono tre ingressi), e ci sono anche tre righe in sala operatoria piano (dato che ci sono tre uscite). Figura C.3.4 mostra la risultante PLA, con i termini di prodotto corrispondente a verità le voci della tabella dall'alto verso il basso.

Invece di disegnare tutte le porte, come facciamo in figura C.3.4, i progettisti spesso mostrano solo la posizione di porte AND e OR porte. Puntini vengono utilizzati sulla intersezione di una linea di segnale di termine prodotto e una linea di ingresso o di una linea di uscita quando una porta corrispondente porta AND oppure OR è richiesto. Figura C.3.5 mostra come il PLA di figura C.3.4 apparirebbe quando viene disegnata in questo modo. I contenuti di un PLA sono fissati quando il PLA viene creato, anche se ci sono anche forme di PLA strutture simili, chiamato *PAls*, Che può essere programmato elettronicamente quando un progettista è pronto per l'uso.

read-only memory (ROM) Una memoria i cui contenuti sono designate al momento della creazione, dopo di che l' contents possono essere solo letti. ROM viene usato come logica strutturata per implementare una insieme di funzioni logiche utilizzando i termini in funzioni logiche come input di indirizzo e le uscite come bit in ogni parola della memoria.

ROM programmabile (PROM) Laforma di memoria di sola lettura programmabile quando un progettista conosce il suo contenuto.

ROM

Lanother forma di logica strutturata che può essere utilizzato per implementare un insieme di logica fun-zioni è un **read-solo memory (ROM)**.Una ROM è chiamato memoria perché ha un insieme di posizioni che possono essere letti, tuttavia, il contenuto di queste posizioni sono fissati, solitamente al momento è fabbricato il ROM. Ci sono anche **ROM programmabili (PROM)** programmabile elettronicamente, quando un progettista conosce il contenuto. Ci sono anche PROM cancellabili; questi dispositivi richiedono un lento processo di cancellazione con luce ultravioletta, e quindi sono utilizzati come memorie a sola lettura, tranne che durante il processo di progettazione e messa a punto.

Una ROM ha un insieme di linee di indirizzo di ingresso e una serie di uscite. Il numero di voci nella ROM indirizzabili determina il numero di linee di indirizzo: se l'

ROM contains 2^mvoci indirizzabili, detto il *altezza*,poi ci sono *m*linee di ingresso. Il numero di bit in ciascuna voce indirizzabile è pari al numero di bit di uscita ed è talvolta chiamato l' *wIDþ* della ROM. Il numero totale di bit nella ROM è uguale ai tempi altezza della larghezza. L'altezza e la larghezza sono talvolta collettivamente denominati *forma* della ROM.

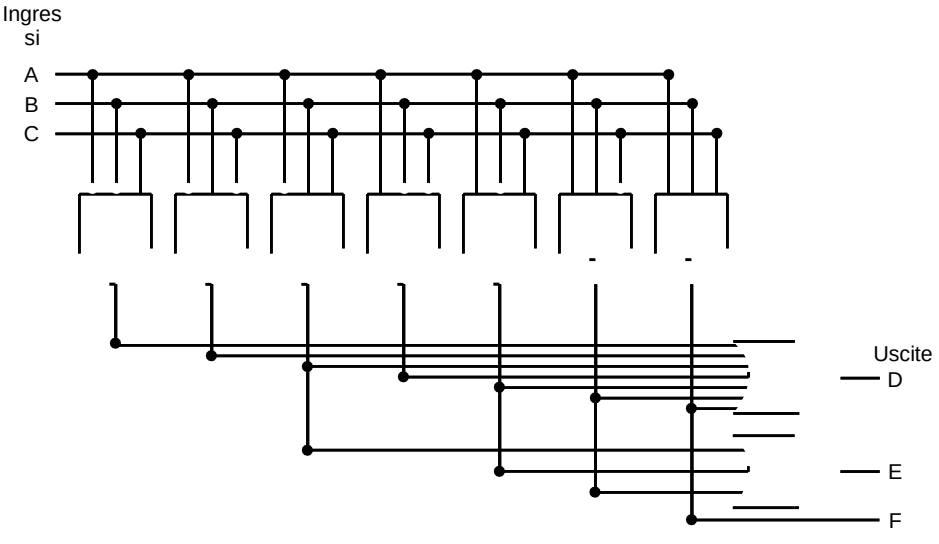


FIGURA C.3.4 Il PLA per la realizzazione della funzione logica descritta nell'esempio.

Una ROM può codificare un insieme di funzioni logiche direttamente dalla tabella di verità. Ad esempio, se vi sono *n*funzioni con *m*ingressi, abbiamo bisogno di una ROM con *m*linee di indirizzo (e 2^mvoci), con ogni voce di essere *n*bit. Le voci nella porzione di ingresso della tabella di verità rappresentano gli indirizzi delle voci nella ROM, mentre il contenuto della porzione di uscita della tabella di verità costituiscono il contenuto della ROM. Se la tabella di verità è organizzato in modo che la sequenza di voci nella porzione di ingresso costituisce una sequenza di numeri binari (come tutte le tabelle di verità abbiamo mostrato finora), quindi la porzione di uscita dà il contenuto ROM per pure. Nell'esempio a pagina C-13, c'erano tre ingressi e tre
uscite. Questo porta ad una ROM con 23 =8 voci, ogni 3 bit. Il contenuto di
tali voci in ordine crescente per indirizzo sono direttamente in porzione di uscita della tabella di verità che appare a pagina C-14.

ROM e PLA sono strettamente correlati. Una ROM è completamente decodificato: contiene una parola piena potenza per ogni combinazione possibile input. Un PLA è solo parzialmente decodificato. Ciò significa che una ROM conterrà sempre più voci. Per la tabella di verità in precedenza a pagina C-14, la ROM contiene le voci per tutti gli otto ingressi possibili, mentre il PLA contiene solo i termini di prodotto che lavorano sette. Poiché il numero di ingressi cresce,

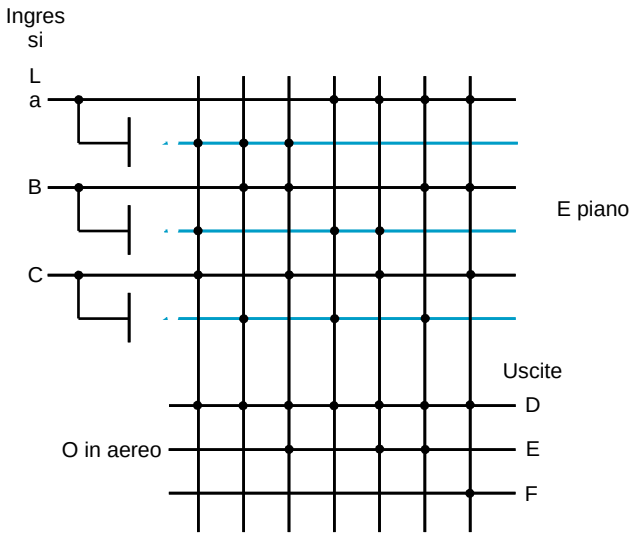


Figura C.3.5 Un PLA disegnato con puntini per indicare i componenti dei termini di prodotto e termini somma nella matrice. Invece di inverter uso sulle porte, di solito tutti gli ingressi vengono gestiti la larghezza del piano e sia vera e forme del complemento. Un punto nel piano e indica che l'ingresso, o il suo inverso, avviene nel termine prodotto. Un punto nel piano O indica che il termine prodotto corrispondente appare l'uscita corrispondente.

il numero di voci nella ROM cresce in modo esponenziale. Al contrario, per la maggior parte delle funzioni logiche reali, il numero di termini prodotto cresce molto più lentamente (vedi gli esempi in **Appendic D**). Questa differenza rende PLA generalmente più efficiente per realizzare funzioni logiche combinatorie. ROM hanno il vantaggio di essere in grado di realizzare qualsiasi funzione logica con il corrispondente numero di ingressi e uscite. Questo vantaggio rende più facile modificare il contenuto ROM se i cambiamenti di funzione logica, poiché la dimensione della ROM non deve cambiare.

Ioltre a ROM e Plas, moderni sistemi di sintesi logica sarà anche piccoli blocchi trans-fine della logica combinatoria in una collezione di porte che possono essere posizionate e cablate automaticamente. Sebbene alcune piccole collezioni di porte di solito non efficiente zona, per le funzioni logiche piccole hanno meno risorse rispetto alla struttura rigida di una ROM e PLA e così sono preferiti.

For progettazione logica al di fuori di un circuito personalizzato o semicustom integrato, una scelta comune è un dispositivo di programmazione campo, abbiamo descritto questi dispositivi nella sezione C.12.

Non Cares

Spesso, in attuazione di alcune logica combinatoria, ci sono situazioni in cui non ci interessa quale sia il valore di alcuni di uscita è, o perché un'altra uscita è vera o perché un sottoinsieme delle combinazioni di input determina i valori delle uscite. Tali situazioni sono indicati come *non si preoccupa*. Non cure sono importanti perché facilitare ottimizzare l'attuazione di una funzione logica.

Ci sono due tipi di non importa: uscita non importa e ingresso non importa, entrambi i quali possono essere rappresentati in una tabella di verità. *Di uscita non si preoccupa* sorgono quando non si preoccupano del valore di un uscita per una combinazione di ingresso. Appaiono come Xs nella porzione di uscita di una tabella di verità. Quando un'uscita è un non si cura per una combinazione di input, l'ottimizzazione progettista o logica del programma è libero di rendere l'output vero o falso per quella combinazione di input. *Iongresso non si preoccupa* sorgono quando un'uscita dipende solo alcuni degli ingressi, e sono anche mostrati come Xs, se nella porzione di ingresso della tabella di verità.

Non Cares

ESEMPIO

Consider una funzione logica con ingressi *La*,*B*, E *C*definito come segue:

- Iof*La*o *C*è vero, allora l'uscita *D*è vero, qualunque sia il valore di *B*.
- Iof*La*o *B*è vero, allora l'uscita *E*è vero, qualunque sia il valore di *C*.
- Produzione *F*è vero se esattamente uno degli ingressi è vero, anche se non si preoccupano del valore di *F*,whenever *D*e *E*sono entrambe vere.

Mostra la tabella di verità completa per questa funzione e la tabella di verità con non importa. Quanti termini di prodotto sono necessari in un PLA per ciascuno di questi?

RISPOSTA

Qui 's la tabella di verità completa, senza non si preoccupa:

Ingres			Uscite		
L	*B*	*C*	*D*	*E*	*F*
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	1	1	0
1	0	0	1	1	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	0

Ciò richiede sette termini di prodotto, senza ottimizzazione. La tabella di verità scritto con uscita non si prende cura è simile al seguente:

Ingres			Uscite		
L	*B*	*C*	*D*	*E*	*F*
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	1	1	X
1	0	0	1	1	X
1	0	1	1	1	X
1	1	0	1	1	X
1	1	1	1	1	X

Iof usiamo anche l'ingresso non se ne frega, questa tabella di verità può essere ulteriormente semplificato per ottenere quanto segue:

Ingressi Uscite *ABCDEF*					
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	1	1
X	1	1	1	1	X
1	X	X	1	1	X

This tabella di verità semplificata richiede una PLA con quattro mintermini, oppure può essere implementato in porte discrete con uno a due ingressi porta AND e OR tre porte (due con tre ingressi e una con due ingressi). Questo confronto con la tabella di verità originale che aveva sette mintermini e avrebbe richiesto quattro porte AND.

Logicminimization è fondamentale per ottenere implementazioni efficienti. Uno strumento utile per ridurre al minimo la mano di logica casuale è *Karnaughmappe*. Mappe di Karnaugh rappresentare graficamente la tabella di verità, in modo che termini prodotto che possono essere combinati sono facilmente visibili. Tuttavia, l'ottimizzazione mano di funzioni logiche significative utilizzando Karnaugh mappe è impraticabile, sia a causa delle dimensioni delle mappe e la loro complessità. Fortunatamente, il processo di minimizzazione logica è altamente meccanica e può essere eseguita da strumenti di progettazione. Nel processo di minimizzazione, gli strumenti di sfruttare il non importa, così indicando loro è importante. I riferimenti libri di testo alla fine della presente appendice fornisce ulteriori discussioni sulla minimizzazione logica, mappe di Karnaugh, e la teoria che sta dietro tali algoritmi di minimizzazione.

Array di elementi logici

Uomoy delle operazioni combinatorie da eseguire sui dati devono essere fatto per un'intera parola (32 bit) di dati. Così spesso si vuole costruire un array di elementi

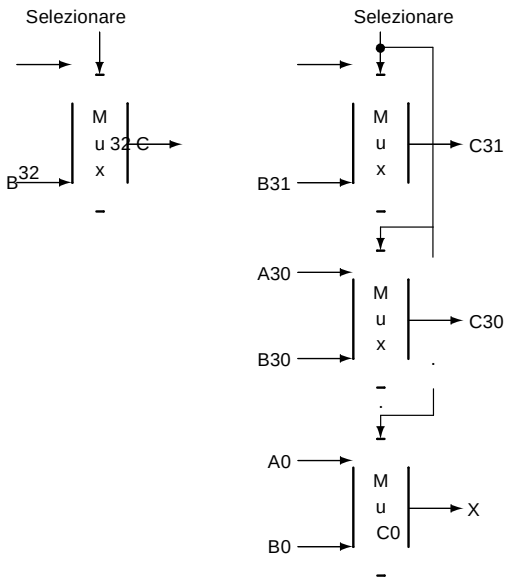
logici,

which possiamo rappresentare semplicemente mostrando che una determinata operazione sarà di un'intera collezione di ingressi. Per esempio, abbiamo visto a pagina C-9 che un 1-bit multiplexor sembrava, ma all'interno di una macchina, la maggior parte del tempo che vogliamo scegliere tra una coppia di *autobus*.La**bus**Ioraccolta sa di linee dati trattati insieme come un unico segnale logico. (Il termine *bus*Ios anche utilizzared per indicare un insieme condiviso di linee con più fonti e impieghi, in particolare nel capitolo 6, in cui si è discusso di I / O bus.)

autobus Ion progettazione logica, un insieme di linee di dati che vengono trattati insieme come un unico segnale logico, inoltre, una raccolta condivisa di linee con più fonti e impieghi.

For esempio, nel set di istruzioni MIPS, il risultato di una istruzione che viene scritto in un registro può provenire da una delle due fonti. Un multiplexer permette di scegliere quale dei due bus (ogni 32 bit di larghezza) verrà scritto nel registro dei risultati. L'1-bit multiplexer, che abbiamo mostrato in precedenza, sarà necessario replicare 32 volte.

We indicano che il segnale è un bus piuttosto che una singola linea 1-bit mostrandola con una linea spessa in figura. La maggior parte degli autobus sono 32 bit, quelli che non sono esplicitamente etichettati con la loro larghezza. Quando mostriamo una unità logica di cui ingressi e uscite sono autobus, questo significa che l'unità deve essere replicata un numero sufficiente di volte per accogliere la larghezza dell'ingresso. La figura mostra come si C.3.6 disegnare un multiplexer che seleziona tra una coppia di 32-bit bus e come questa si espande in termini di
1 bit-widposta multiplexer. A volte abbiamo bisogno di costruire un array di elementi logici in cui i mezzi di produzione per alcuni elementi della matrice sono uscite da elementi precedenti. Ad esempio, questo è il modo in cui una multibit livello ALU è costruito. In tali casi, si deve esplicitamente mostrano come creare ampie matrici, poiché i singoli elementi della matrice non sono più indipendenti, come sono, nel caso di un 32-bit di ampiezza multiplexer.



un. A 32 bit b larga 2-a-1 multiplexer. Il 32-bit multiplexor gamma è in realtà una serie di 32 a 1 bit

multiplexer

Figura C.3.6 Un multiplexer è schierato 32 volte per eseguire una selezione tra due 32-bit ingressi. Nota che c'è ancora un solo segnale di selezione dei dati utilizzati per tutti i 32 1 bit multiplexer.

Controllare Ynoi stessi

Parietà è una funzione in cui l'uscita dipende dal numero di 1 in ingresso. Per una funzione di parità pari, l'uscita è 1 se l'ingresso ha un numero pari di quelle. Supponiamo che un ROM viene utilizzata per implementare una funzione di parità pari con un 4-bit di ingresso. Che di A, B, C, o D rappresenta il contenuto della ROM?

Indirizzo	L	B	C	D
0	0	1	0	1
1	0	1	1	0
2	0	1	0	1
3	0	1	1	0
4	0	1	0	1
5	0	1	1	0
6	0	1	0	1
7	0	1	1	0
8	1	0	0	1
9	1	0	1	0
10	1	0	0	1
11	1	0	1	0
12	1	0	0	1
13	1	0	1	0
14	1	0	0	1
15	1	0	1	0

C.4 Utilizzando un linguaggio di descrizione dell'hardware

Hardware linguaggio di descrizione Un linguaggio di programmazione for hardware descrivendo, utilizzato per generare simulazioni di un design hardware e anche come input per gli strumenti di sintesi che possono generare l'hardware reale.

VerILOG Una delle due lingue hardware più comuni descrizione.

VHDL Una delle due lingue hardware più comuni descrizione.

Adisegno, giorno, la maggior parte digitale di processori e sistemi hardware relativi è fatto usando un **hardware linguaggio di descrizione**. Tale linguaggio serve a due scopi. In primo luogo, fornisce una descrizione astratta di hardware per simulare ed eseguire il debug del progetto. In secondo luogo, con l'impiego di sintesi logica e strumenti di compilazione hardware, questa descrizione può essere compilato nel implementazione hardware.

Ion questa sezione, si introduce il linguaggio di descrizione hardware Verilog e mostrare come può essere utilizzato per la progettazione combinatoria. Nel resto della appendice, abbiamo espandere l'uso di Verilog per includere progettazione di logica sequenziale. Nelle sezioni facoltative di cui al capitolo 4 che appaiono sul CD, che usiamo per descrivere Verilog implementazioni del processore. Nella sezione facoltativa dal capitolo 5 che appare sul CD, usiamo un sistema di Verilog per descrivere le implementazioni del controller della cache. Sistema Verilog aggiunge strutture e alcune altre caratteristiche utili per Verilog.

VerILOG è una delle due lingue principali di descrizione hardware, l'altro è **VHDL**.VerILOG è alquanto più ampiamente utilizzato nell'industria ed è basato su

C, al contrario di VHDL, che si basa su Ada. Il lettore in genere familiarità con C sarà

trovare le basi di Verilog, che usiamo in questa appendice, facili da seguire. I lettori hanno già familiarità con VHDL dovrebbe trovare i concetti semplici, purché siano stati esposti alla sintassi di C.

VeILOG è possibile specificare sia un comportamento e una definizione strutturale di un sistema digitale. La **specifiche comportamentali** descrive il funzionamento di un sistema digitale di funzionalmente aziona. La **specifica strutturale** descrive l'organizzazione dettagliata di un sistema digitale, generalmente utilizzando una descrizione gerarchica. Una specifica strutturale può essere usato per descrivere un sistema hardware in termini di una gerarchia di elementi di base come porte e interruttori. Così, potremmo usare Verilog per descrivere il contenuto esatto delle tabelle di verità e datapath dell'ultima sezione.

Won l'arrivo di **hardware sintesi** aols, la maggior parte dei progettisti ora utilizzare Verilog o VHDL per descrivere strutturalmente solo l'unità di elaborazione, basandosi sulla sintesi logica per generare il controllo da una descrizione comportamentale. Inoltre, la maggior parte dei sistemi CAD offrono ampie librerie di componenti standardizzati, come ALU, multiplexer, registrare i file, ricordi, e blocchi logici programmabili, così come cancelli di base.

Obtaining un risultato accettabile usando le librerie e la sintesi logica richiede che la specifica essere scritto con un occhio verso la sintesi finale e il risultato desiderato. Per i nostri disegni semplici, questo significa in primo luogo chiarire che cosa ci aspettiamo di essere implementato in logica combinatoria e ciò che ci aspettiamo di richiedere logica sequenziale. Nella maggior parte degli esempi che usiamo in questa sezione e il resto di questa appendice, abbiamo scritto il Verilog con la sintesi finale in mente.

Tipi di dati e operatori in Verilog

Ci sono due tipi di dati primari in Verilog:

1. La **wira** specifica un segnale combinatoria.
2. La **reg**(Registro) contiene un valore, che può variare con il tempo. Un reg non deve necessariamente corrispondere a un registro reale in una implementazione, anche se spesso sarà.

Un registro o filo, denominato X, che è 32 bit è dichiarato come un array: reg [31:0] X o filo [31:0] X, che stabilisce anche l'indice da 0 a designare il bit meno significativo del registro. Dato che spesso si vuole accedere ad un sottocampo di un registro o di filo, è possibile fare riferimento a un insieme contiguo di bit di un registro o di filo con la notazione [bit di partenza: fine bit], in cui entrambi gli indici devono essere valori costanti.

Lan matrice di registri è utilizzata per una struttura come un file di registro o memoria. Pertanto, la dichiarazione

```
reg [31:0] registerfile [0:31]
```

spécifiésregisterfile una variabile che è equivalente a una registerfile MIPS, dove registrati
0 è la prima. Quando si accede a una matrice, si può fare riferimento a un singolo elemento, come in C, utilizzando la notazione registerfile [regnum].

specifiche comportamentali Descrive come un sistema digitale opera funzionalmente.

specifica strutturale Descrive come un sistema digitale è organizzato in termini di una connessione gerarchica di elementi.

strumenti hardware di sintesi Computer-assistita progettazione del software che può generare un cancello a livello di progettazione sulla base delle descrizioni di comportamento di un sistema digitale.

wira Ion Verilog, specifica un segnale combinatoria.

regIon Verilog, un registro.

I valori possibili per un registro o filo in Verilog sono

- 0 o 1, che rappresenta false logica o vero
- X, che rappresenta sconosciuto, il valore iniziale dato a tutti i registri e qualsiasi filo non collegato a qualcosa
- Z, che rappresenta la stato di alta impedenza per cancelli tri-state, che non discuteremo in questa appendice

CoI valori nstant possono essere specificati come numeri decimali così come binario, ottale o esadecimale. Noi spesso vogliono dire esattamente quanto grande di un campo costante è in bit. Questo occorre anteporre il valore con un numero decimale che specifica la dimensione in bit. Per esempio:

- 4'b0100 specifica un 4-bit costante binaria con il valore 4, come fa 4'd4.
- - 8 'h4 specifica un 8-bit con il valore costante -4(In due la rappresentazione in complemento)

Valori possono anche essere concatenate ponendoli all'interno di {} separati da com-mas. La notazione {{x}} campo di bit replica bit x volte sul campo. Per esempio:

- {16}} {2'b01 crea un valore a 32 bit con il modello 0101. . . 01.
- {A [31:16], B [15:0]} crea un valore cui 16 bit superiori vengono da A e la cui 16 bit inferiori provengono da B.

VeILOG offre l'insieme completo di operatori unari e binari da C, inclusi gli operatori aritmetici (+, -, *. /), Gli operatori logici (&, |, ~), il confronto (operatori= =, !=, >, <, <=, > =), gli operatori (spostamento<<, >>), e C condizionale
operatore (che viene utilizzato nella condizione forma expr1:?? espr2 e ritorna expr1 se la condizione è vera e expr2 se è falsa). Verilog aggiunge un insieme di operatori unari riduzione logici (&, |, ^) che producono un singolo bit applicando l'operatore logico a tutti i bit di un operando. Ad esempio, & A restituisce il valore ottenuto come AND tutti i bit di un insieme, e ^ A restituisce la riduzione ottenuta mediante OR esclusivo su tutti i bit di A.

Controllare Ynoi stessi

Which delle seguenti definire esattamente lo stesso valore?

1. 8'b11110000
2. 8'hF0
3. 8'd240
4. {{{4 1'b1}}, {4 {1'b0}}}
5. {4'b1, 4'b0}

Struttura di un programma Verilog

A Verilog programma è strutturato come un insieme di moduli che possono rappresentare qualsiasi cosa, da un insieme di porte logiche di un sistema completo. I moduli sono simili a
classi in C+ +, anche se non così potente. Un modulo di ingresso e ne specifica il porte di uscita, che descrivono le connessioni in entrata e in uscita di un modulo. Un modulo può anche dichiarare variabili aggiuntive. Il corpo di un modulo è costituito da:

- iniziale constructs, in grado di inizializzare le variabili reg
- Cocontinua che le assegnazioni, che definiscono solo la logica combinatoria
- sempre constructs, che può definire sia la logica sequenziale o combinatorio
- Ionstances di altri moduli, che vengono utilizzati per implementare il modulo essendo definita

Rappresentare Complesso logica combinatoria in Verilog

Un compito continuo, che viene indicato con la parola chiave assegnare, agisce come una funzione logica combinatoria: l'uscita viene assegnato il valore continuamente, e un cambiamento dei valori di ingresso si riflette immediatamente nel valore di uscita. I fili possono essere assegnati valori con le assegnazioni continui. Utilizzo di assegnazioni di continuo, siamo in grado di definire un modulo che implementa un mezzo sommatore, come figura C.4.1 spettacoli.

Lassign dichiarazioni sono un modo sicuro per scrivere Verilog che genera logica combinatoria. Per le strutture più complesse, invece, le dichiarazioni assegnano può essere imbarazzante o noioso da usare. È anche possibile utilizzare la sempre a blocchi di un modulo per descrivere un elemento combinatorio logica, anche se è necessario prestare attenzione. Utilizzando sempre un blocco consente l'inserimento di costrutti di controllo Verilog, come *if-then-else, caso* dichiarazioni, *per* dichiarazioni, e *retorba* dichiarazioni, da utilizzare. Queste istruzioni sono simili a quelli in C con piccole modifiche.

Lan sempre blocco specifica una lista opzionale di segnali su cui il blocco è sen-sibili (in un elenco che inizia con @). Il blocco viene sempre rivalutato se una delle elencato

```
Modulo half_adder (A, B, Somma, Carry);
    ingresso A, B, / / due 1-bit input
    Somma di uscita, avanti, / / due 1-bit output
    assegnare Somma = A ^ B; / / somma è A xor B
    assegnare Carry = A & B / / Carry è A e B
endmodule
```

FIGURA C.4.1 Un modulo Verilog che definisce un mezzo sommatore utilizzando assegnazioni continue.

segnali di cambiamenti valore, se la lista è omesso, il blocco è sempre costantemente rivalutata. Quando un blocco è sempre specifica logica combinatoria, la **sensibilità lista** dovrebbero includere tutti i segnali di ingresso. Se ci sono più Verilog-lancio da eseguire sempre in un blocco, sono circondati dalle parole chiave di inizio e fine, che prendono il posto della {e} in C. Un blocco sembra sempre così in questo modo:

sensibilità lista L'elenco dei segnali che specifica quando un blocco deve sempre essere rivalutata.

```
sempre @ (elenco dei segnali che causano rivalutazione)
cominciare
    Dichiarazioni Verilog tra assegnazioni e altri terminali
di controllo dichiarazioni
```

Variabili Reg può essere assegnato solo all'interno di un blocco di sempre, con una dichiarazione di procedura di assegnazione (in quanto distinto da cessione continuo abbiamo visto in precedenza). Vi sono, tuttavia, due diversi tipi di assegnazioni procedurali. Il

Operatore di assegnazione =executes come in C, il lato destro è valutata, e il lato sinistro è assegnato il valore. Inoltre, esegue come la normal istruzione di assegnamento C: che è, si completa prima che l'istruzione successiva viene eseguita. Di conseguenza, l'assegnazione operatore =ha il nome **blocco di assegnazione**.

blocco di assegnazione In VerILOG, un incarico che completa prima l'esecuzione dell'istruzione successiva.

Questo blocco può essere utile per la generazione di logica sequenziale, e torneremo ad esso breve. L'altra forma di assegnazione (**non bloccante**) È indicata da <=. In assegnazione non bloccante, tutti a destra i lati delle assegnazioni in un gruppo sempre vengono valutate e le assegnazioni sono fatte contemporaneamente. Come primo esempio di logica combinatoria implementata utilizzando sempre un blocco, figura C.4.2 mostra l'implementazione di un 4-a-1 multiplexor, che utilizza un costrutto caso per rendere più facile scrivere. Il costrutto caso si presenta come una istruzione switch C. Figura C.4.3 mostra la definizione di un ALU MIPS, che utilizza anche una dichiarazione caso.

Assegnazione non bloccante Lan assegnazione che continua dopo la valutazione del lato destro, assegnando a sinistra lato il valore solo dopo che tutti i lati a destra vengono valutati.

Since solo le variabili reg può essere assegnato all'interno di blocchi di sempre, quando si vuole descrivere la logica combinatoria utilizzando un blocco di sempre, la cura deve essere adottate per assicurare che il reg non sintetizzare in un registro. Una varietà di trappole sono descritti nella elaborazione sottostante.

Elaborazione: Istruzioni di assegnazione continui sempre resa logica combinatoria, ma altre strutture Verilog, anche se in sempre blocchi, può produrre risultati imprevisti durante la sintesi logica. Il problema più comune è la creazione di logica sequenziale implica-zione l'esistenza di un latch o di registro, che si traduce in una implementazione che è sia più lento e più costoso forse destinato. Per garantire che la logica che si intende essere combinatoria è sintetizzato in questo modo, assicurarsi di effettuare le seguenti operazioni:

1. Inserire tutta la logica combinatoria di un incarico o di un continuo sempre bloccare.
2. Assicurarsi che tutti i segnali utilizzati come ingressi vengono visualizzati nell'elenco sensibilità di un sempre bloccare.

3. Assicurarsi che ogni percorso attraverso un sempre blocco assegna un valore alla stessa coppia di bit.

L'ultimo di questi è il più facile da trascurare, per leggere attraverso l'esempio della Figura C.5.15 per convincere te stesso che questa proprietà è rispettato.

```
Modulo Mult4to1 (IN1, IN2, IN3, IN4, Sel, Out);
    ingresso [31:0] IN1, IN2, IN3, IN4, / quattro
    ingressi a 32-bit input [1:0] Sel; / / selettore di
    segnale
    uscita reg [31:0] Out ;/ / 32-bit di
    uscita sempre @ (IN1, IN2, IN3, IN4,
    Sel)
    caso (Sel) / / a 4 -> 1 multiplexor
        0: Out <= In1;
        1: Out <= In2;
        2: Out <= In3;
        di default: Out <= In4;
    endcase
endmodule
```

FIGURA C.4.2 Una definizione Verilog di un 4-a-1 con multiplexor 32-bit input, utilizzando un caso dichiarazione. La dichiarazione caso si comporta come un'istruzione switch C, tranne che in Verilog solo il codice associata con la selezione corrente viene eseguito (come se ogni stato caso aveva una pausa alla fine) e non vi è caduta attraverso l'istruzione successiva .

```
modulo MIPSALU (ALUctl, A, B, ALUOut, Zero);
    ingresso [3:0] ALUctl;
    ingresso [31:0] A, B;
    uscita reg [31:0] ALUOut;
    uscita Zero;
    assegnare Zero = (ALUOut == 0) / / Zero è vero se ALUOut è 0; va da nessuna parte sempre @
    (ALUctl, A, B) / / rivalutare se questi cambiamenti
        caso (ALUctl)
            0: ALUOut <= A e B;
            1: ALUOut <= A | B;
            2: ALUOut <= A + B;
            6: ALUOut <= A - B;
            7: ALUOut <= A <B? 01:00;
            12: ALUOut <= ~ (A | B), / / risultato è né
            di default: ALUOut <= 0; / / default a 0, non dovrebbe accadere;
        endcase
endmodule
```

FIGURA C.4.3 Una definizione comportamentale Verilog di un ALU MIPS. Questo potrebbe essere sintetizzato utilizzando una libreria di moduli contenente aritmetiche e logiche.

Controllare Ynoi stessi

Lassuming tutti i valori sono inizialmente zero, quali sono i valori di A e B dopo l'esecuzione di questo codice Verilog sempre all'interno di un blocco?

```
C =
  1;
La<= C, B
= C;
```

*ALU n. [**La**rthritic **L**OGIC **U**nit o (raro) **La**rithmetic **L**OGIC **U**nit] Un generatore di numeri casuali fornito di serie con tutti i sistemi informatici.*

Stan Kelly-Bootle, *Il Devil DP di Dictionary*, 1981

C.5 Costruzione di una logica di base aritmetica Unità

Il **ica aritmetica** (**ALU**)è la soppressata del computer, il dispositivo che esegue le operazioni aritmetiche, come l'addizione e la sottrazione o operazioni logiche come AND e OR. In questa sezione costruisce una ALU da blocchi hardware quattro edifici (porte AND e OR, inverter, e multiplexer) e illustra come funziona la logica combinatoria. Nel prossimo paragrafo, vedremo come più può essere accelerato attraverso disegni più intelligenti.

Perché la parola MIPS è di 32 bit, abbiamo bisogno di una a 32 bit a livello di ALU. Supponiamo che collegherà 32 1-bit ALU per creare le ALU desiderati. Ci quindi iniziare con la costruzione di un 1-bit ALU.

A 1-bit ALU

Le operazioni logiche sono più facili, perché mappare direttamente sui componenti hardware nella figura C.2.1.

Il 1 bit unità logica per AND e OR si presenta come figura C.5.1. Il multiplexer a destra seleziona quindi *un*E *b*o *un*O *b*, A seconda che il valore di *Operazione* è 0 o 1. La linea che controlla il multiplexor appare a colori ad essa dis-differenziare gli da le righe che contengono dati. Si noti che abbiamo rinominato le linee di controllo e di uscita del multiplexer per dare loro nomi che riflettono la funzione della ALU.

La prossima funzione da includere è aggiunta. Un sommatore deve avere due ingressi per gli operandi e un singolo bit di uscita per la somma. Ci deve essere una seconda uscita di trasmettere il riporto, chiamata *Carryout*. Dal momento che il Carryout dal sommatore prossimo deve essere contenuto in un ingresso, abbiamo bisogno di un terzo ingresso. Questo ingresso è chiamato *Carryin*. Figura C.5.2 mostra gli ingressi e le uscite di un sommatore a 1 bit. Poiché sappiamo che cosa oltre che dovrebbe fare, è possibile specificare le uscite di questa "scatola nera" in base alle sue entrate, come dimostra la figura C.5.3.

We può esprimere il Carryout uscita funzioni e Somma come equazioni logiche, e queste equazioni possono a loro volta essere implementato con porte logiche. Facciamo Carry-Out. Figura C.5.4 mostra i valori degli ingressi quando Carryout è un 1.

We può trasformare questa tabella di verità in una equazione logica:

$$\text{Carryout} = (B \cdot \text{Carryin}) + (A \cdot \text{Carryin}) + (A \cdot b) + (A \cdot b \cdot \text{Carryin})$$

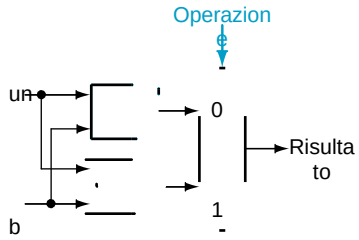
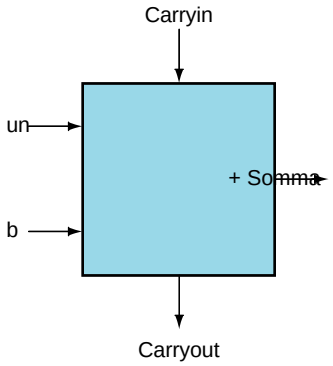


Figura C.5.1 Il 1 bit unità logica per AND e OR.



C.5.2 FIGURA A 1-bit adder. Questa tolleranza è chiamato un sommatore, ma è anche chiamato (3,2) sommatore perché ha 3 ingressi e 2 uscite. Un sommatore con solo gli ingressi A e B è chiamato a (2,2) sommatore o half-adder.

Ingres			Uscite		
u	b	Carryin	Carryout	Som	Commenti
0	0	0	0	0	0 + 0 + 0 = 00two
0	0	1	0	1	0 + 0 + 1 = 01two
0	1	0	0	1	0 + 1 + 0 = 01two
0	1	1	1	0	0 + 1 + 1 = 10two
1	0	0	0	1	1 + 0 + 0 = 01two
1	0	1	1	0	1 + 0 + 1 = 10two
1	1	0	1	0	1 + 1 + 0 = 10two
1	1	1	1	1	1 + 1 + 1 = 11two

FIGURA C.5.3 di ingresso e le specifiche di uscita per un sommatore a 1 bit.

Iofa · b · Carryin è vero, allora tutti gli altri tre termini deve essere anche vero, in modo che possiamo lasciare fuori questo termine ultimo corrispondente alla quarta riga della tabella. Possiamo quindi semplificare l'equazione di

$$\text{Carryout} = (B \cdot \text{Carryin}) + (A \cdot \text{Carryin}) + (A \cdot b)$$

Figura C.5.5 mostra che l'hardware all'interno della casella nera per sommatore Carryout è costituito da tre porte AND e una porta OR. Le tre porte AND corrispondono esattamente ai tre termini tra parentesi della formula sopra per Carryout, e la porta OR somma i tre termini.

Ingres		
u	**b**	**Carryin**
0	1	1
1	0	1
1	1	0
1	1	1

FIGURA C.5.4 valori degli ingressi quando Carryout è un 1.

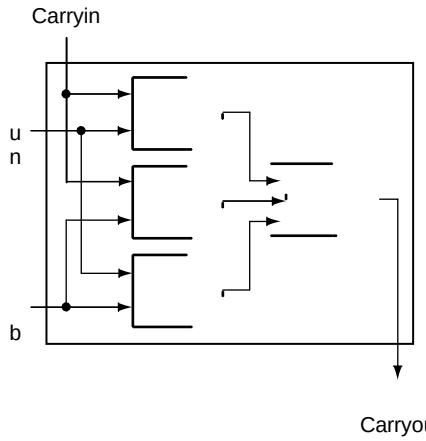


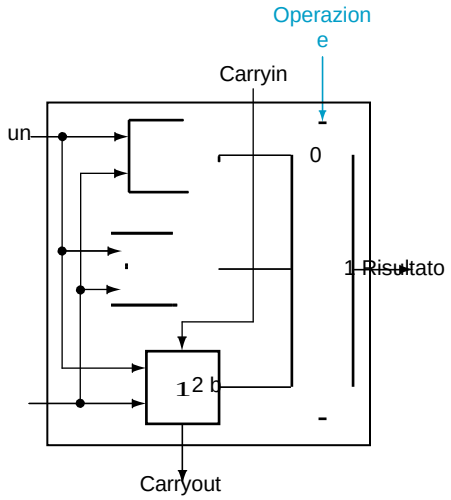
FIGURA C.5.5 hardware Adder per il segnale Carryout. Il resto dell'hardware sommatore è la logica per l'uscita Sum dato nell'equazione di questa pagina.

Il bit viene impostato quando Sum esattamente un ingresso è 1 o quando tutti e tre gli ingressi sono 1. Il
Srisultati um in una complessa equazione booleana (ricōrdiamo che un mezzo, non a):

$$\text{Sum} = (A \cdot \bar{b} \cdot \overline{\text{Carryin}}) + (\bar{A} \cdot b \cdot \overline{\text{Carryin}}) + (A \cdot \bar{b} \cdot \text{Carryin}) + (A \cdot b \cdot \text{Carryin})$$

Il disegno della logica per il bit Somma nella casella Black Adder è lasciato come esercizio per il lettore.

Figura C.5.6 mostra un 1-bit ALU derivati combinando il sommatore con i componenti precedenti. A volte i progettisti vogliono anche la ALU per eseguire alcuni più le semplici operazioni, come la generazione 0. Il modo più semplice per aggiungere una operazione è quello di ampliare il multiplatore controllato dalla linea di funzionamento e, per questo esempio, 0 per connettersi direttamente all'ingresso del multiplatore nuova espanso.



C.5.6 FIGURA A 1-bit ALU che esegue AND, OR, e l'aggiunta (vedi figura C.5.5).

A 32-bit ALU

Now che abbiamo completato le 1-bit ALU, la piena a 32-bit ALU viene creato di collegare adiacenti "scatole nere". Utilizzo *xIo*to intende il *Io*esimo bit di *x*, Figura C.5.7 mostra un 32-bit ALU. Proprio come una pietra può causare increspature di irradiare sulle sponde di un lago tranquillo, un singolo effettuare il bit meno significativo (Result0) possono propagarsi tutto il percorso attraverso la vipera, causando un procedere del bit più significativo (Result31). Quindi, il sommatore creato collegando direttamente le trasporta di sommatori a 1 bit è chiamato *ripple trasportare* sommatore. Vedremo un modo più veloce per collegare le 1-bit adder a partire da pagina C-38.

Subtraction è la stessa aggiungendo la versione negativa di un operando, e questo è come sommatori eseguire sottrazione. Ricordiamo che il collegamento per la negazione di un numero di complemento a due è quello di invertire ogni bit (talvolta chiamata *proprio complemento*) E si aggiunge 1. Per invertire ogni bit, bisogna semplicemente aggiungere un 2:1 multiplexer che sceglie
tra B e B, come figura C.5.8 spettacoli.

Suppose noi connect 32 di questi a 1-bit ALU, come abbiamo fatto nella figura C.5.7. Il multiplexer aggiunto dà la possibilità di b o il suo valore invertito, a seconda Binvert, ma

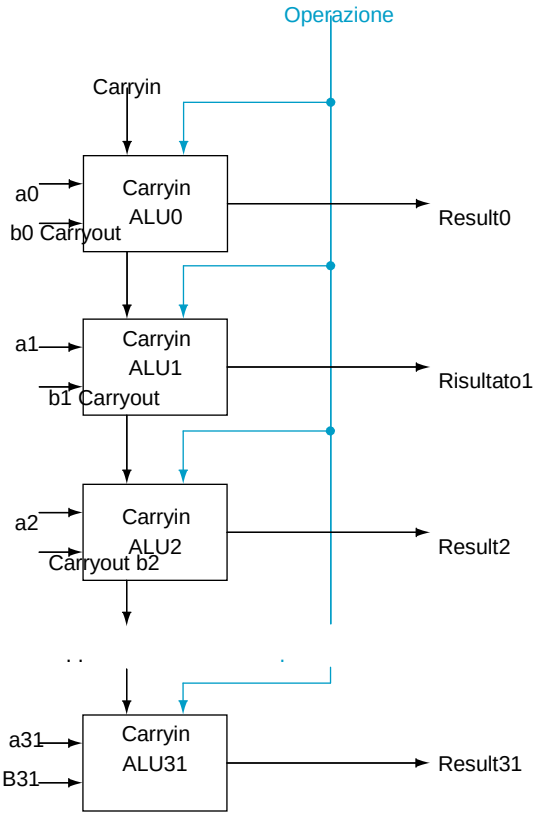
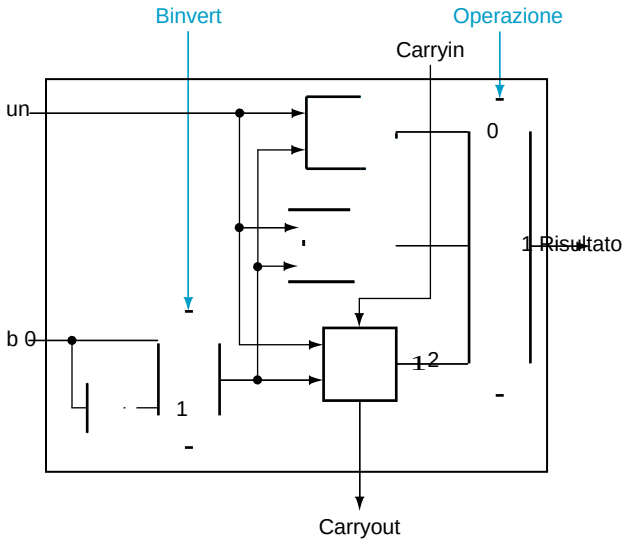


FIGURA C.5.7 A a 32 bit ALU costruite da 32 a 1-bit ALU. Carryout del bit meno significativo è collegato al Carryin del bit più significativo. Questa organizzazione si chiama ripple carry.

questo è solo un passo nel negare a due numero di complemento. Si noti che il bit meno significativo ha ancora un segnale di carry, anche se è necessario per l'aggiunta. Che cosa succede se si imposta questa carry a 1 invece di 0? Il sommatore calcola quindi
un+b+1. Selezionando la versione invertita di b, si ottiene esattamente quello che vogliamo:

$$un+\bar{b}+1=un+(B\ \bar{}+1) =un+(-b) =un- b$$

Thsemplicità e della progettazione hardware di un sommatore a due complemento aiuta a spiegare perché due la rappresentazione in complemento è diventato lo standard universale per l'aritmetica del computer intero.



C.5.8 FIGURA A 1-bit ALU che esegue AND, OR, e l'aggiunta di A e B o a e b. Da selezione b (Binvert =1) e l'impostazione carry a 1 il bit meno significativo della ALU, otteniamo complemento a due sottrazione di b da un posto di aggiunta di b ad un.

Una ALU MIPS ha anche bisogno di una funzione NOR. Invece di aggiungere una porta separata per NOR, siamo in grado di riutilizzare gran parte dell'hardware già in ALU, come abbiamo fatto per il sub-tratto. L'intuizione viene dalla verità NOR seguenti informazioni:

$$\overline{(A + b)} = \overline{a} \cdot \overline{b}$$

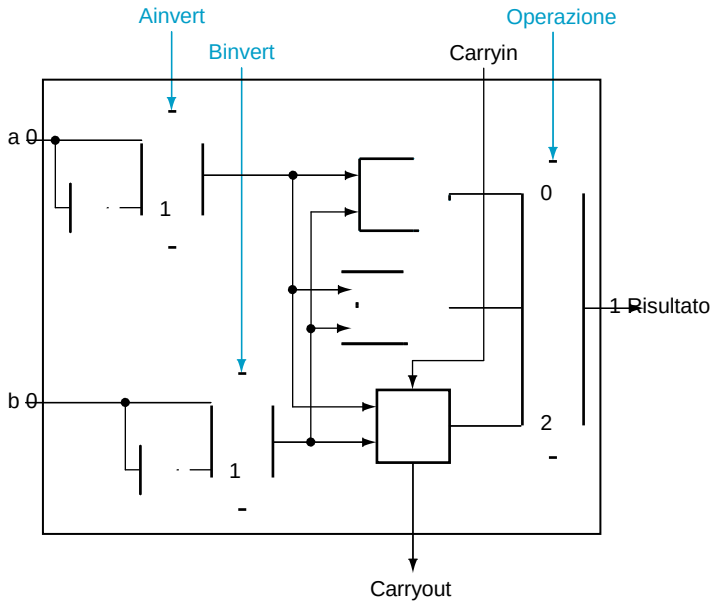
Che non è, (A o B) è equivalente a NOT A e B non. Questo fatto è chiamato DeMorgan teorema e viene esplorato negli esercizi in modo più approfondito.

Since abbiamo AND e NOT b, abbiamo solo bisogno di aggiungere un NOT alle ALU. Figura C.5.9 dimostra che i cambiamenti.

Tailoring a 32-bit ALU per MIPS

Queste quattro operazioni, addizione, sottrazione, AND, OR-si trovano nei ALU di quasi tutti i computer, e le operazioni di maggior parte delle istruzioni MIPS può essere eseguita da questo ALU. Ma il design della ALU è incompleta.

Suistruzione e che ha ancora bisogno di sostegno è il set con meno di istruzioni (slt). Ricordiamo che l'operazione produce 1 se rs < rt, E 0 altrimenti. Di conseguenza, TAS
set tutti ma il bit meno significativo a 0, con il bit meno significativo impostato secondo il confronto. Per la ALU per eseguire slt, abbiamo prima bisogno di espandere la tre ingressi



C.5.9 FIGURA A 1-bit ALU che esegue AND, OR, e l'aggiunta di A e B o a e b. Da selezione ūn (Ainvert =1) e b (Binvert =1), si ottiene una b NOR al posto di a e b.

multiplexor in Figura C.5.8 per aggiungere un ingresso per il risultato slt. Noi chiamiamo questo nuovo input
*Less*und utilizzarle esclusivamente per slt.

Il disegno superiore della Figura C.5.10 mostra i nuovi 1-bit ALU con il multiplexor espanso. Dalla descrizione di slt sopra, si deve collegare l'ingresso da 0 a meno per i 31 bit superiori delle ALU, dal momento che tali bit sono sempre impostati a 0. Ciò che rimane da considerare è come confrontare e impostare il *meno significativo bit* per set con meno di istruzioni.

Wcappello succede se sottraiamo b da un? Se la differenza è negativa, allora <b da

$$(A - b) < 0 \Rightarrow ((A - b) + b) < (0 + b)$$
$$\Rightarrow un < b$$

We desidera il bit meno significativo di un insieme meno di operazione da un 1 se un <b;
cioè un 1 se un - b è negativo e 0 se è positivo. Tale risultato corrisponde exactly to il segno di valori di bit: 1 significa negativo e 0 indica positivi. Seguendo questo
argomento, è sufficiente collegare il bit del segno dal sommatore di uscita per il bit meno significativo per ottenere impostare con meno di.

Unfortunately, l'uscita Result dal bit più significativo ALU in cima Figura C.5.10 per l'operazione è slt *non* l'uscita del sommatore; ALU l'uscita per l'operazione SLT è ovviamente il valore di ingresso minore.

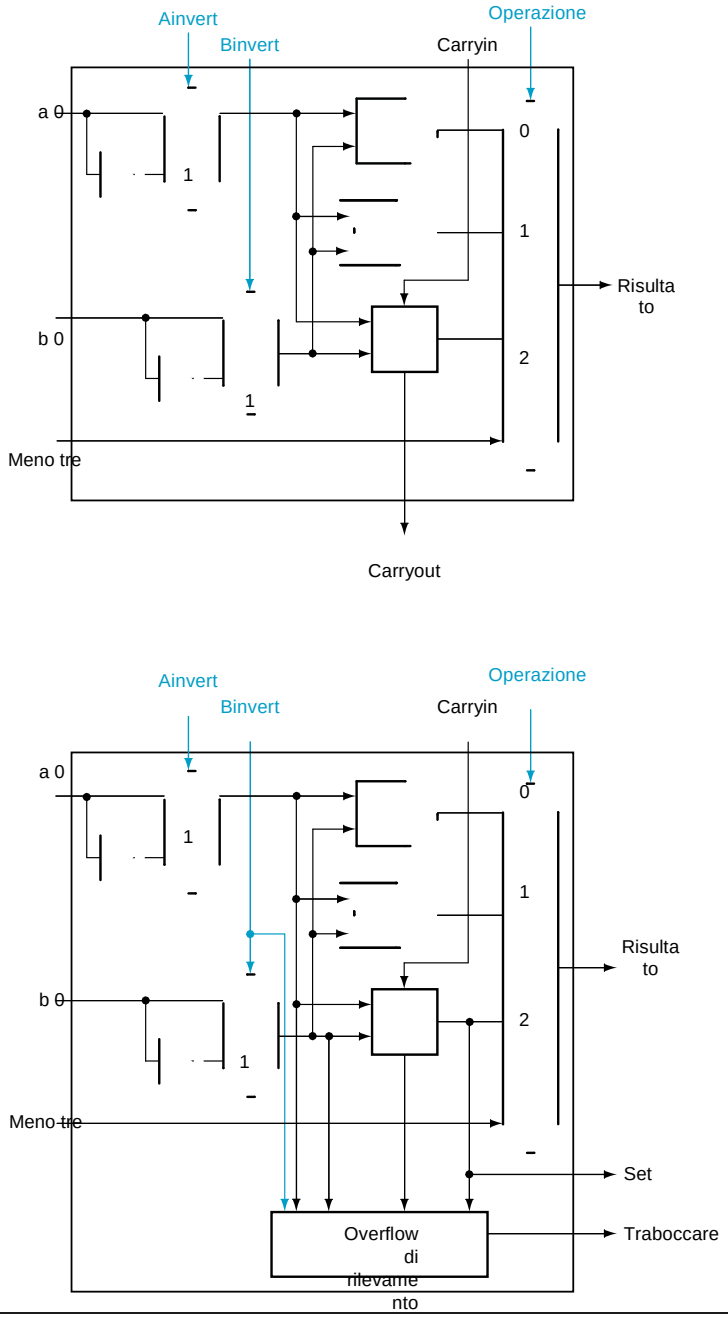


FIGURA C.5.10 (Top) A 1-bit ALU che esegue AND, OR, e l'aggiunta di A e B o b, e (in basso) a 1-bit ALU per il bit più significativo. Il disegno superiore include un ingresso diretto che è collegato per eseguire il set con meno di funzionamento (vedi Figura C.5.11), il fondo ha un'uscita diretta dal sommatore di meno di confronto chiamato Set. (Si veda l'Esercizio C.24 alla fine di questa appendice per vedere come calcolare troppo pieno con meno ingressi.)

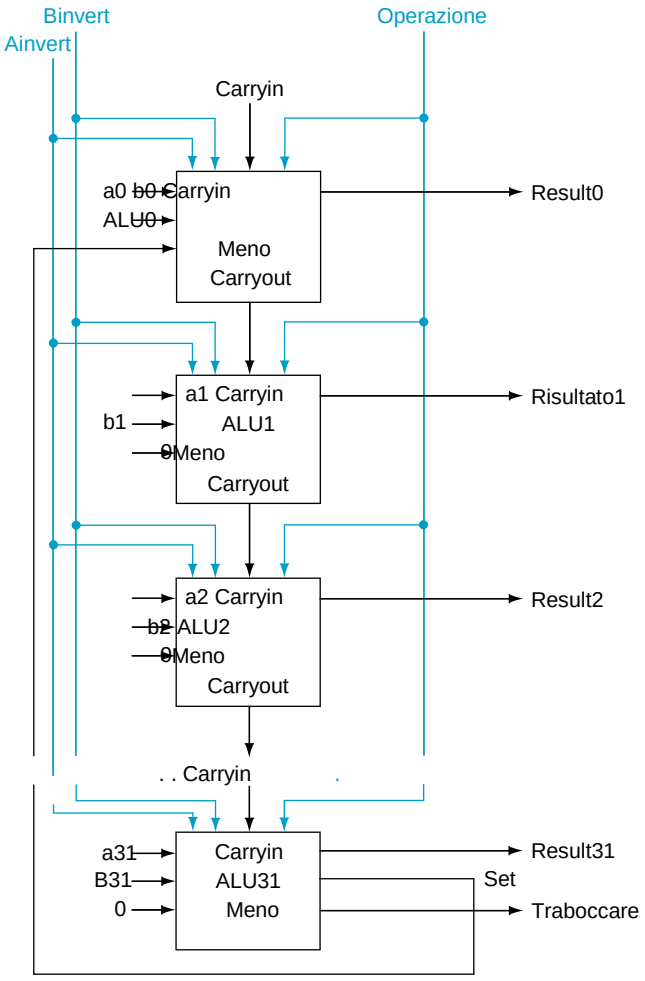


FIGURA C.5.11 A a 32-bit ALU costruiti a partire dalle 31 copie dei 1-bit ALU nella parte superiore della figura C.5.10 e una 1-bit ALU nella parte inferiore di quella figura. Gli ingressi sono collegati Meno a 0 tranne per il bit meno significativo, che è collegato all'uscita Set del bit più significativo. Se l'ALU
esegue - b e selezionare l'ingresso 3 del multiplexor nella figura C.5.10, poi Risultato =0. . . 001 se un <b, e risultato =0. . . 000 in caso contrario.

Quindi, abbiamo bisogno di un nuovo 1-bit ALU per il bit più significativo che ha un po 'più di uscita: l'uscita sommatore. Il disegno inferiore della Figura C.5.10 mostra il design, con questa nuova linea di uscita sommatore chiamato *Set,*und utilizzato solo per slt. Finché abbiamo bisogno di un ALU speciali per il bit più significativo, abbiamo aggiunto la logica di rilevamento di overflow poiché è anche associata a tale bit.

Ahimè, la prova di meno è un po 'più complicato di quanto appena descritto, a causa di un overflow, mentre esploriamo negli esercizi. Figura C.5.11 mostra i 32-bit ALU.

NOtice che ogni volta che vogliamo la ALU per sottrarre, abbiamo impostato sia Carryin e Binvert a 1. Per aggiungere o operazioni logiche, vogliamo entrambe le linee di controllo a 0. Possiamo quindi semplificare il controllo della ALU combinando il carry e Binvert di una linea di controllo singolo chiamato *Bnegate*.

To personalizzare ulteriormente le ALU per il set di istruzioni MIPS, dobbiamo sostenere condi-zioni istruzioni di salto. Questi ramo istruzioni o se i due registri sono uguali o se sono uguali. Il modo più semplice per verificare l'uguaglianza con l'ALU è quello di sottrarre b da a e quindi verificare se il risultato è 0, dal momento che

$$(A - b = 0) \Rightarrow un = b$$

Quindi, se si aggiungono componenti hardware per verificare se il risultato è 0, siamo in grado di verificare l'uguaglianza. Il modo più semplice è quello di O tutte le uscite insieme e poi inviare il segnale tramite un inverter:

$$Zero = \overline{(Result31 + Result30 + \ldots + Result2 + Result1 + Result0)}$$

Figura C.5.12 mostra le riviste a 32 bit ALU. Si può pensare alla combinazione del-1 bit Ainvert linea, le linee 1-bit Binvert linea, e il 2-bit Funzionamento come 4-linee di bit di controllo per la ALU, dicendo di eseguire addizioni, sottrazioni, AND, OR, o impostare con meno. Figura C.5.13 mostra le linee di controllo ALU e il funzionamento corrispondente ALU.

Infine, ora che abbiamo visto quello che è all'interno di una ALU a 32 bit, useremo il simbolo universale per un ALU completo, come mostrato nella Figura C.5.14.

Definizione della ALU MIPS in Verilog

Figura C.5.15 mostra come una combinatoria MIPS ALU può essere specificato in Verilog; tale specifica sarebbe probabilmente essere compilato utilizzando una libreria standard parti fornito un sommatore, che potrebbe essere istanziata. Per completezza, mostriamo il controllo ALU per MIPS in figura C.5.16, che viene utilizzato nel capitolo 4, dove costruire una versione Verilog del datapath MIPS.

La domanda successiva è: "Quanto tempo questo ALU aggiungere due operandi a 32 bit?" Siamo in grado di determinare gli ingressi A e B, ma l'ingresso Carryin dipende dal funzionamento nell'adiacente sommatore a 1 bit. Se si traccia tutto il percorso attraverso la catena di dipendenze devono, si collega il bit più significativo al bit meno significativo, per cui il bit più significativo della somma deve attendere la *sequential* evalutazione di tutte le 32 a 1 bit sommatori. Questa reazione a catena sequenziale è troppo lento per essere utilizzati in tempo critico hardware. La sezione successiva esplora il modo per accelerare aggiunta. Questo argomento non è cruciale per comprendere il resto

del l'appendice e può essere saltata.

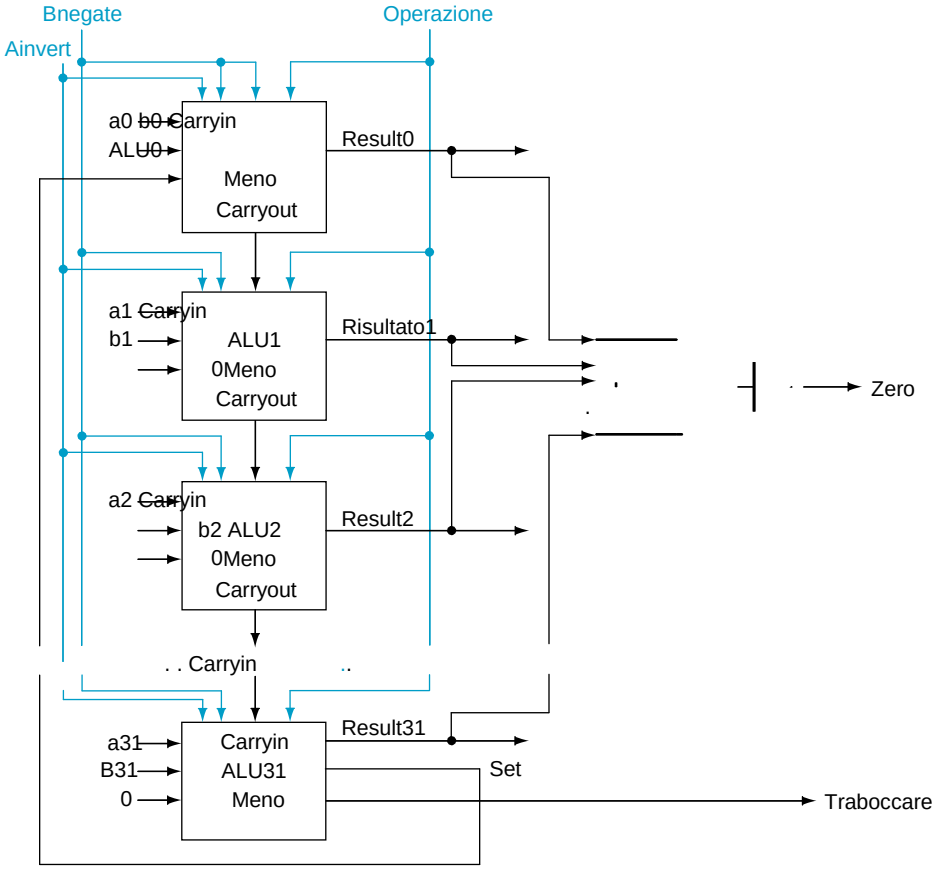


FIGURA C.5.12 Gli ultimi 32 bit ALU. Questo aggiunge un rivelatore Zero alla figura C.5.11.

ALU controllo delle linee	Funzione
0000	E
0001	O
0010	aggiungere
0110	sottrarre
0111	impostato con meno
1100	NOR

Figura C.5.13 I valori delle tre linee di controllo ALU, Bnegate, e del funzionamento e le operazioni corrispondenti ALU.

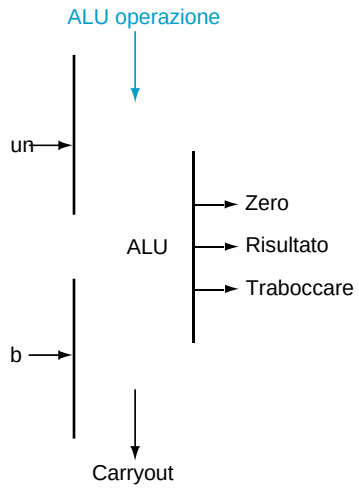


FIGURA C.5.14 Il simbolo comunemente usato per rappresentare una ALU, come mostrato in figura C.5.12. Questo simbolo è anche usato per rappresentare un sommatore, quindi è normalmente etichettati sia con ALU o sommatore.

```
modulo MIPSALU (ALUctl, A, B, ALUOut, Zero);
    ingresso [3:0] ALUctl;
    ingresso [31:0] A, B;
    uscita reg [31:0] ALUOut;
    uscita Zero;
    assegnare Zero = (ALUOut == 0) / / Zero è vero se è 0 ALUOut sempre @
    (ALUctl, A, B) begin / / rivalutare se questi cambiamenti
        caso (ALUctl)
            0: ALUOut <= A e B;
            1: ALUOut <= A | B;
            2: ALUOut <= A + B;
            6: ALUOut <= A - B;
            7: ALUOut <= A <B? 1: 0;
            12: ALUOut <= ~ (A | B), / / risultato è né predefinito:
            ALUOut <= 0;
        endcase
     fine
endmodule
```

FIGURA C.5.15 Una definizione comportamentale Verilog di un ALU MIPS.

```
Modulo ALUControl (ALUOp, FuncCode, ALUCtl);
    ingresso [01:00] ALUOp;
    ingresso [05:00] FuncCode; uscita
    [3:0] reg ALUCtl;
    sempre case (FuncCode)
    32: ALUOp <= 2; / / aggiungere
    34: ALUOp <= 6; / / sottrazione
    36: ALUOP <= 0, / / e
    37: ALUOp <= 1; / / o
    39: ALUOp <= 12; / / né
    42: ALUOp <= 7; / / slt
    di default: ALUOp <= 15; / / non dovrebbe accadere
    endcase
endmodule
```

Figura C.5.16 Il MIPS ALU di controllo: un semplice pezzo di logica di controllo combinatoria.

Controllare Ynoi stessi

Suppose si vuole aggiungere l'operazione di NOT (A e B), chiamato NAND. Come potrebbe cambiare la ALU per sostenerlo?

1. Nessun cambiamento. È possibile calcolare NAND rapidamente utilizzando i ALU correnti, in quanto
   $(\overline{un \cdot b}) = un + b$e abbiamo già NON è un NON, b, e OR.
2. È necessario ampliare il multiplexor grande per aggiungere un altro ingresso, e quindi aggiungere nuova logica per calcolare NAND.

C.6 Faster Aggiunta: Carry lookahead

La chiave per accelerare Inoltre è determinante il riporto per i bit più presto. Ci sono una varietà di schemi di anticipare il riporto in modo che il caso peggiore è funzione della log2 del numero di bit nel sommatore. Questi segnali anticipatori sono più veloci perché passano attraverso minor numero di porte in sequenza, ma ci vogliono molti più porte di anticipare il riporto corretto.

Una chiave to comprensione rapida carry regimi è quello di ricordare che, a differenza del software, l'hardware viene eseguito in parallelo, ogni volta che gli ingressi cambiano.

Fast Carry Utilizzando hardware "Infinito"

Las abbiamo accennato in precedenza, ogni equazione può essere rappresentato in due livelli di logica. Dal momento che gli unici ingressi esterni sono i due operandi e il carry al minimo

bit significativo del sommatore, in teoria potremmo calcolare i valori Carryin per tutti i rimanenti bit del sommatore a due soli livelli di logica.

For esempio, il Carryin per bit 2 del sommatore è esattamente la Carryout di bit 1, quindi la formula è

CarryIn2 =(B1. CarryIn1) +(A1. CarryIn1) +(A1. B1)

Analogamente, è definito come CarryIn1

CarryIn1 =(B0. CarryIn0) +(A0. CarryIn0) +(A0. B0)

Ucantare la sigla più breve e più tradizionale di c*Io*per Carryin*Io*, Possiamo riscrivere le formule come

c2 =(B1. C1) +(A1. C1) +(A1. B1)
c1 =(B0. C0) +(A0. C0) +(A0. B0)

Substituting la definizione di c1 per i risultati prima equazione in questa formula:

c2 =(A1. A0. B0) +(A1. A0. C0) +(A1. B0. C0)

+(B1. A0. B0) +(B1. A0. C0) +(B1. B0. C0) +(A1. B1)

You può immaginare come l'equazione espande come si arriva a più bit del sommatore; cresce rapidamente con il numero di bit. Tale complessità si riflette nel costo dell'hardware per trasportare veloce, rendendo questo semplice schema proibitivo per sommatori larghe.

Fast Carry Con il primo livello di astrazione: Propagare e generare

Most fast-portano regimi limitare la complessità delle equazioni di semplificare l'hardware, pur rendendo miglioramenti sostanziali velocità oltre ripple carry. Uno schema di questo tipo è un *carry-lookahead adder*.Ion Capitolo 1, abbiamo detto sistemi informatici far fronte alla complessità, utilizzando livelli di astrazione. Una ricerca in avanti di sommatore si basa su diversi livelli di astrazione nella sua attuazione.

Let 's fattore nostra equazione originale, come primo passo:

$$c_{Io+1} = (B_{Io} \cdot c_{Io}) + (A_{Io} \cdot c_{Io}) + (A_{Io} \cdot b_{Io})$$
$$= (A_{Io} \cdot b_{Io}) + (A_{Io} + b_{Io}) \cdot c_{Io}$$

Iof dovessimo riscrivere l'equazione per c2 utilizzando questa formula, vedremmo alcuni modelli ripetuti:

c2 =(A1. B1) +(A1 +b1). ((A0. B0) +(A0 +b0).) c0

Nota la comparsa ripetuta di (un*Io*· b*Io*) E (a*Io*+b*Io*) Nella formula precedente.

Queste due fattori importanti sono tradizionalmente chiamati *generare* (G*Io*) E *propagare* (P*Io*):

$$g_{Io} = un_{Io} \cdot b_{Io}$$
$$p_{Io} = un_{Io} + b_{Io}$$

Ucanto loro di definire c_{Io+1}, Si

ottiene

$$c_{Io+1} = g_{Io} + p_{Io} \cdot$$

$$c_{Io}$$

To vedere dove i segnali di ottenere i loro nomi, si supponga che g_{Io} è 1. Poi

$$c_{Io+1} = g_{Io} + p_{Io} \cdot c_{Io} = 1 + p_{Io} \cdot c_{Io} = 1$$

Cioè, il sommatore *genera* un Carryout (c_{Io+1}) Indipendente dal valore di Carryin (C_{Io}). Supponiamo ora che g_{Io}p è 0 e $_{Io}$è 1. Poi

$$c_{Io+1} = g_{Io} + p_{Io} \cdot c_{Io} = 0 + 1. \; c_{Io} = c_{Io}$$

Cioè, il sommatore *propagares*Carryin ad un Carryout. Mettendo le due cose insieme, Carryin$_{Io+1}$è 1 se uno dei due g_{Io}è 1 o entrambi p_{Io}è 1 e Carryin$_{Io}$è 1.

Las analogia, immaginate una fila di domino fissati sul bordo. Il domino finale può essere ribaltato premendo uno lontano, se non ci sono spazi vuoti tra i due. Allo stesso modo, un procedere può essere fatto da un vero generare lontano, a condizione che tutti si propaga tra di loro sono vere.

Relying sulle definizioni di propagarsi e generare il nostro primo livello di astrazione, possiamo esprimere i segnali Carryin economicamente più. Guardiamo in spettacolo per 4 bit:

$$c1 = g0 + (P0. \; C0)$$

$$c2 = g1 + (. \; P1 \; g0) + (P1. \; P0. \; C0)$$

$$c3 = g2 + (P2. \; G1) + (P2. \; P1. \; G0) + (P2. \; P1. \; P0. \; C0)$$

$$c4 = g3 + (P3. \; G2) + (P3. \; P2. \; G1) + (P3. \; P2. \; P1. \; G0) + (P3. \; P2. \; P1. \; P0. \; C0)$$

Queste equazioni rappresentano solo il buon senso: Carryin$_{Io}$è 1 se un po 'di tolleranza in precedenza genera un riporto e tutti i sommatori intermedi propagare un riporto. Figura C.6.1 utilizza idraulico per cercare di spiegare portare lookahead.

Evit questa forma semplificata porta a equazioni di grandi dimensioni e, quindi, la logica considerevole anche per un sommatore a 16 bit. Cerchiamo di passare a due livelli di astrazione.

Fast Carry Con il secondo livello di astrazione

In primo luogo, consideriamo questo sommatore a 4 bit con la sua ricerca in avanti di logica come un unico blocco di costru-zione. Se li collegare in ripple portare la moda in modo da formare un sommatore a 16 bit, il componente aggiuntivo sarà più veloce rispetto all'originale con l'hardware un po 'più.

To andare più veloce, abbiamo bisogno di portare lookahead ad un livello superiore. Per eseguire portare guardare avanti per 4-bit adder, abbiamo bisogno di diffondere e generare segnali a questo livello più alto. Qui sono per i quattro blocchi di 4 bit adder:

P0 =p3. p2. p1. p0

P1 =p7. p6. p5. p4

P2 =p11. p10. p9. p8

P3 =p15. p14. p13. p12

Cioè, il "super" propagare il segnale 4-bit astrazione (P*Io*) È vera solo se ciascuno dei bit del gruppo si propaga un riporto.

For il "super" Generazione di segnale (G*Io*), Ci interessa solo se vi è un procedere del bit più significativo del gruppo 4-bit. Ciò si verifica se, ovviamente, generare è vero per quel po 'più significativo, ma si verifica anche se un precedente generazione è vero *e* tutte le propaga intermedi, tra cui quella del bit più significativo, è anche vero:

G0 =g3 +(P3. G2) +(P3. P2. G1) +(P3. P2. P1. G0)

G1 =g7 +(P7. G6) +(P7. P6. G5) +(P7. P6. P5. G4)

G2 =g11 +(P11. G10) +(P11. P10. G9) +(P11. P10. P9. G8)

G3 =g15 +(P15. G14) +(P15. P14. G13) +(P15. P14. P13. G12)

Figura C.6.2 aggiornamenti nostra analogia idraulica per mostrare P0 e G0.

Poi le equazioni a questo livello più alto di astrazione per il trasporto, per ciascun 4-bit del gruppo 16-bit adder (C1, C2, C3, C4 in figura C.6.3) sono molto simili a portare il out equazioni per ogni bit del sommatore a 4 bit (c1, c2, c3, c4) a pagina C-40:

C1 =G0 +(P0. C0)

C2 =G1 +(. P1 G0) +(P1. P0. C0)

C3 =G2 +(P2. G1) +(P2. P1. G0) +(P2. P1. P0. C0)

C4 =G3 +(P3. G2) +(P3. P2. G1) +(P3. P2. P1. G0)
+(P3. P2. P1. P0. C0)

Figura C.6.3 mostra 4-bit adder collegati con una ricerca in avanti di unità. Gli esercizi esplorare le differenze di velocità tra questi strumenti presentano, notazioni diverse per multibit propagarsi e generare i segnali, e la progettazione di un sommatore a 64 bit.

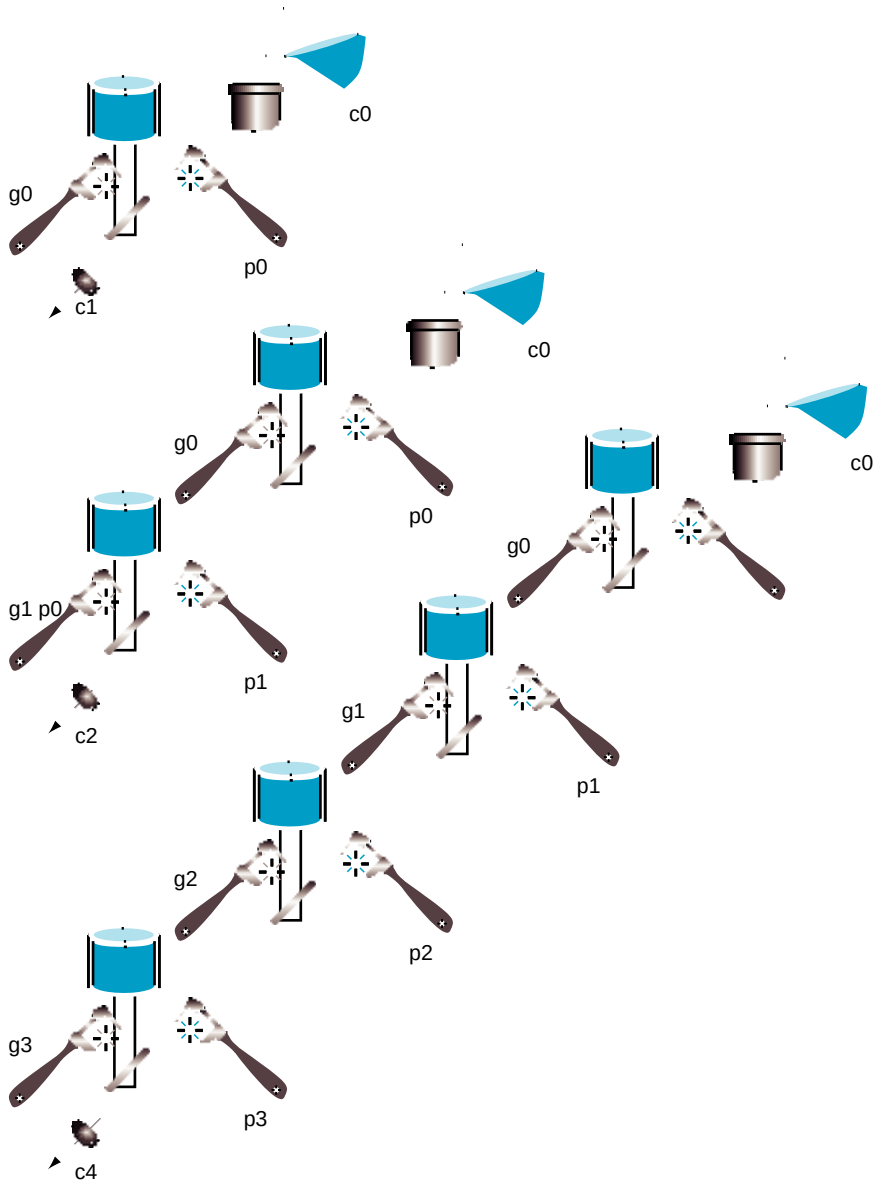


FIGURA C.6.1 Un'analogia idraulico per portare lookahead per 1 bit, 2 bit e 4 bit utilizzando tubi di acqua e valvole. Le chiavi sono rivolti alle valvole di apertura e chiusura. Acqua è mostrato in colori. Il
uscita del tubo (c*Io+ 1*) Sarà piena se sia il valore più vicino generate (g*Io*) È acceso o se la *Io*propagate valore (P*Io*) È acceso e c'è acqua più a monte, sia da una precedente o generare un propagano con acqua dietro. Carryin (c0) può provocare una realizzare senza l'aiuto di alcun genera, ma con l'aiuto di *tutti* propaga.

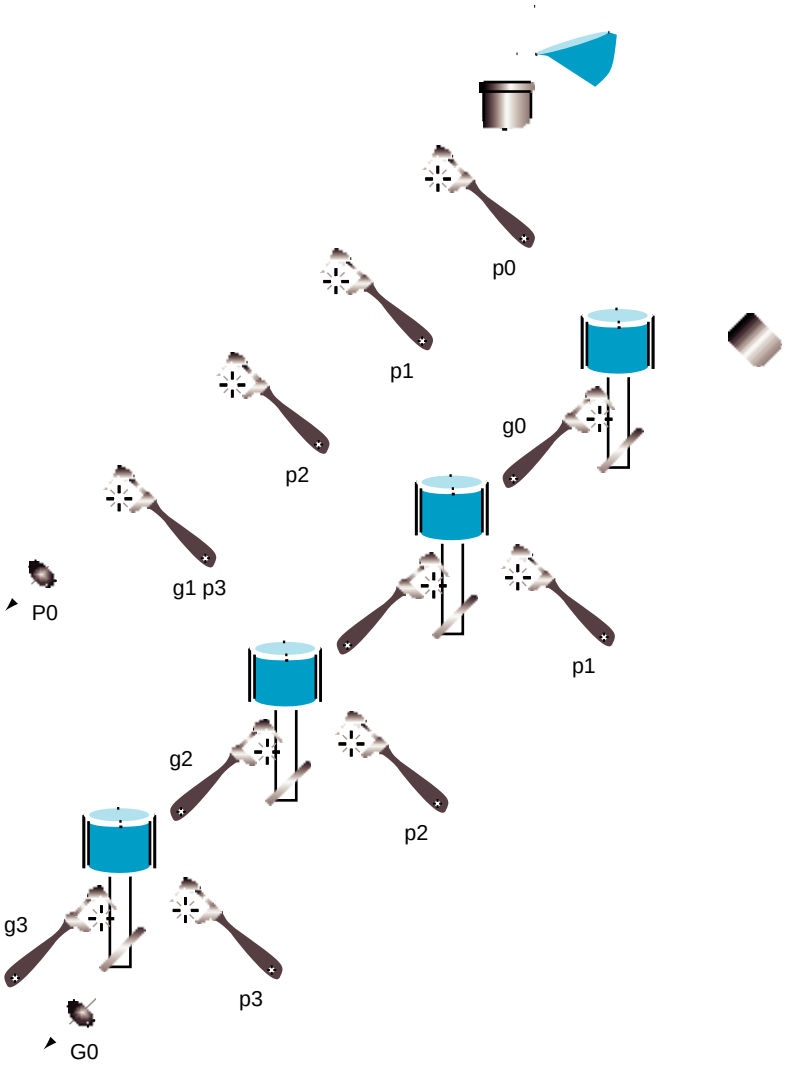


Figura C.6.2 Un'analogia idraulico per il prossimo livello di carry-lookahead segnali P0 e G0. P0 è aperto solo se tutti e quattro si propaga (p*Io*) Sono aperti, mentre l'acqua scorre in G0 solo se almeno uno di generare (g*Io*)è aperto e tutte le propaga a valle che generano sono aperti.

ESEMPIO

Entrambi i livelli di propagarsi e generare

Determinare la g*Io*,p*Io*, P*Io*E G*Io*valori di questi due numeri a 16 bit:

a: 0001 1010 0011 0011 due
b: 1110 0101 1110 1011 due

Inoltre, ciò è CarryOut15 (C4)?

RISPOST A

Allineamento i bit lo rende facile vedere i valori di generare g*Io*(A*Io*·b*Io*) E propagare p*Io*(A*Io*+b*Io*):

a:	0001	1010	0011	0011
b:	1110	0101	1110	1011
g*Io*:	0000	0000	0010	0011
p*Io*:	1111	1111	1111	1011

dove i bit sono numerate da 15 a 0 da sinistra a destra. Successivamente, la "super"
propaga (P3, P2, P1, P0) sono semplicemente l'AND di livello inferiore si

propaga: P3 =1. 1. 1. 1 =1

P2 =1. 1. 1. 1 =1

P1 =1. 1. 1. 1 =1

P0 =1. 0. 1. 1 =0

Il "super" genera sono più complesse, in modo da utilizzare le seguenti

equazioni: G0 =g3 +(P3. G2) +(P3. P2. G1) +(P3. P2. P1. G0)
=0+(1. 0) +(1. 0. 1) +(1. 0. 1. 1) =0+0+0+0=0
G1 =g7 +(P7. G6) +(P7. P6. G5) +(P7. P6. P5. G4)
=0+(1. 0) +(1. 1. 1) +(1. 1. 1. 0) =0+0+1+0=1
G2 =g11 +(P11. G10) +(P11. P10. G9) +(P11. P10. P9. G8)
=0+(1. 0) +(1. 1. 0) +(1. 1. 1. 0) =0+0+0+0=0
G3 =g15 +(P15. G14) +(P15. P14. G13) +(P15. P14. P13. G12)
=0+(1. 0) +(1. 1. 0) +(1. 1. 1. 0) =0+0+0+0=0

Infine, CarryOut15 è

C4 =G3 +(P3. G2) +(P3. P2. G1) +(P3. P2. P1. G0)
+(P3. P2. P1. P0. C0)
=0+(1. 0) +(1. 1. 1) +(1. 1. 1. 0) +(1. 1. 1. 0. 0)
=0+0+1+0+0=1

Eglince, ci *è* a svolgere durante l'aggiunta di questi due numeri a 16 bit.

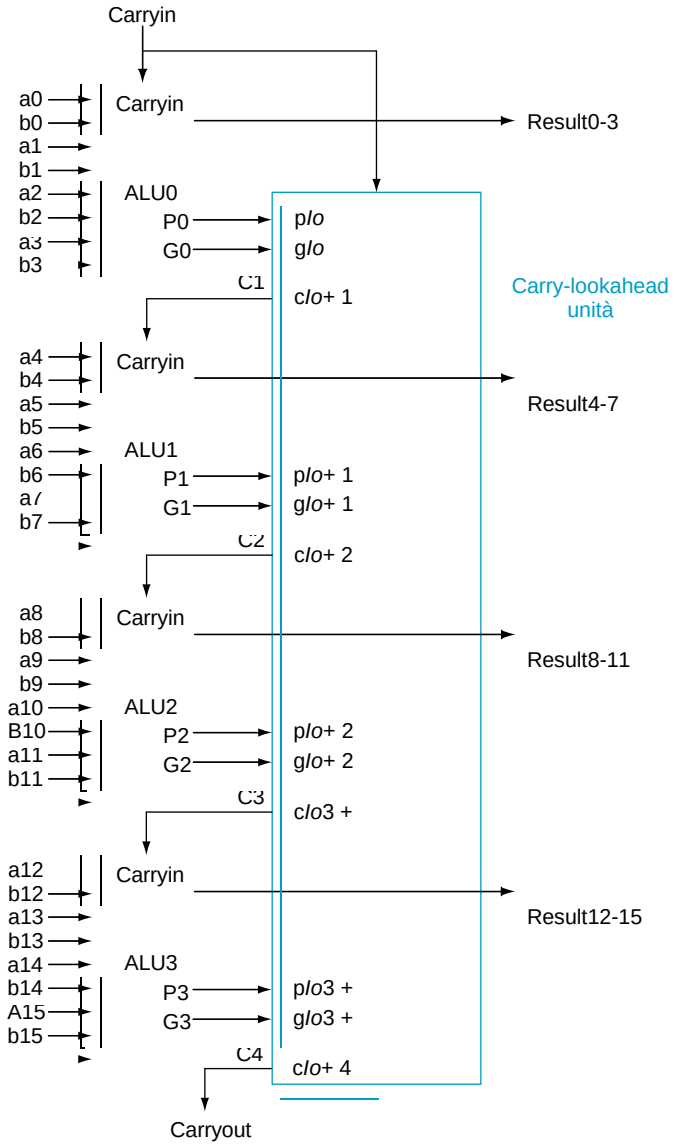


FIGURA C.6.3 Quattro 4-bit ALU utilizzando portare lookahead per formare un sommatore a 16 bit. Nota che la trasporta provengono dal gruppo di ricerca in avanti, non dai 4 bit ALU.

La ragione per trasportare lookahead può rendere più veloce è che trasporta tutte logico inizia valutando momento inizia il ciclo di clock, e il risultato non cambierà dopo l'uscita di ogni porta smette di cambiare. Prendendo la scorciatoia di passare attraverso porte meno per inviare il segnale di carry in, l'uscita delle porte non cambia più presto, e quindi il tempo per il sommatore può essere inferiore.

To apprezzare l'importanza di carry lookahead, abbiamo bisogno di calcolare il rapporto prestazioni-zione tra questo e sommatori carry ripple.

Velocità di Ripple Carry contro Carry lookahead

ESEMPIO

Un modo semplice per modellare il tempo per la logica è assumere ogni AND oppure OR impiega lo stesso tempo di un segnale di passare attraverso di essa. Tempo è stimato semplicemente contando il numero di porte lungo il percorso attraverso un pezzo di logica. Confrontare il numero di *cancello ritardi* per i percorsi di due sommatori a 16 bit, uno che utilizza ripple carry e uno con due livelli portano lookahead.

RISPOST A

Figura C.5.5 a pagina C-28 mostra che il segnale di uscita riporto prende due ritardi di gate per bit. Allora il numero di ritardi di gate tra un riporto in al minimo
significativo bit e il carry out dei più significativi è 16 x 2=32.

For trasportare lookahead, il procedere del bit più significativo è solo C4, definito nell'esempio. Occorrono due livelli di logica per specificare C4 in termini di P*Io*e G*Io*(L'OR di diverse e termini). P*Io*è specificato in un livello di logica (AND) utilizzando p*Io*,e G*Io*è specificato in due livelli con p*Io*e g*Io*,quindi il caso peggiore per questo successivo livello di astrazione è due livelli di logica. p*Io*e g*Io*sono ciascuno un livello di logica, definito in termini di una*Io*e b*Io*.Iof assumiamo un ritardo cancello
per ogni livello di logica in queste equazioni, il caso peggiore è 2 +2+1=5 cancello
ritardi.

Eglince, per il percorso da effettuare per svolgere, il 16-bit Inoltre da un sommatore di ricerca in avanti è sei volte più veloce, utilizzando questa stima molto semplice di velocità dell'hardware.

Riassunto

Carry lookahead offre un percorso più veloce in attesa della porta a propagarsi attraverso tutte le 32 a 1 bit sommatori. Questo percorso più veloce è pavimentato da due segnali, generare e propagare.

Il primo crea un riporto indipendentemente dall'ingresso riporto, e quest'ultimo passa a trasportare. Carry lookahead dà anche un altro esempio di come astrazione è importante nella progettazione di computer per far fronte alla complessità.

Controllare Ynoi stessi

Ucantare la semplice stima della velocità dell'hardware sopra con ritardi di gate, qual è la performance relativa di un ripple carry a 8 bit aggiungere rispetto a 64 bit aggiungere con ricerca in avanti di logica?

1. A 64 bit carry-lookahead adder è tre volte più veloce: 8-bit aggiunge ritardi sono 16 cancelli e 64-bit aggiunge sono 7 ritardi cancello.
2. Sono circa la stessa velocità, poiché 64-bit aggiunge richiedono più livelli di logica nel sommatore a 16 bit.
3. 8-bit aggiunge sono più veloci di 64 bit, anche con trasporto lookahead.

Elaborazione: We hanno rappresentato per tutti, ma una delle operazioni aritmetiche e logiche di base per il set di istruzioni MIPS: le ALU della Figura C.5.14 omette sostegno delle operazioni di scorrimento. Sarebbe possibile ampliare il multiplexor ALU per includere uno spostamento a sinistra di 1 bit o uno spostamento a destra di 1 bit. Ma i progettisti di hardware hanno creato un circuito chiamato *barrel shifter*, Che può passare 1-31 bit in più tempo di quello necessario per aggiungere due
Numeri a 32 bit, per cui lo spostamento avviene normalmente
al di fuori delle ALU.

Elaborazione: L'equazione logica per l'uscita Somma del sommatore a pagina C-28 può essere espresso più semplicemente utilizzando un cancello più potente di AND e OR. Un *esclusivo O* cancello è vero se i due operandi disaccordo, cioè,

$$x \neq y \Longrightarrow 1 \text{ e } x = = y \Longrightarrow 0$$

In alcune tecnologie esclusive, o è più efficiente di due livelli di AND e OR cancelli. Utilizzando il simbolo $\oplus$ per rappresentare OR esclusivo, ecco la nuova equazione:

$$\text{Somma} = a \oplus b \oplus \text{Carryin}$$

Inoltre, abbiamo disegnato le ALU modo tradizionale, utilizzando porte. I computer sono progettati oggi a transistor CMOS, che sono fondamentalmente interruttori. CMOS ALU e shifter a botte approfittare di questi interruttori e multiplexer hanno molti meno rispetto a quanto mostrato nei nostri disegni, ma i principi di progettazione sono simili.

Elaborazione: Utilizzo di maiuscole e minuscole per distinguere la gerarchia di generare e propagare i simboli si rompe quando si hanno più di due livelli. Una notazione alternativa che le scale è $g_{i \,..\, j}$ e $p_{i \,..\, j}$ per generare e propagare i segnali di bit *I*oa *j.* Così, g1 .. 1 è generato per bit 1, G4 .. 1 è per i bit 4-1, e G16 .. 1 è per bit 16 a 1.

C.7 Orologi

Prima di discutere gli elementi di memoria e logica sequenziale, è utile discutere brevemente il tema degli orologi. Questa breve sezione introduce l'argomento ed è simile alla discussione nella sezione 4.2. Maggiori dettagli sul clock e clock di metodo-logie sono presentate nella sezione C.11.

Orologi sono necessaria logica sequenziale per decidere quando un elemento che contiene lo stato dovrebbe essere aggiornato. Un orologio è semplicemente un free-running con un segnale fisso *ciclo time;* il *Frequenza di clock* è semplicemente l'inverso del tempo di ciclo. Come mostrato in Figura C.7.1, la *clock time* o *periodo di clock* è suddivisa in due porzioni: quando il clock è alto e quando il clock è basso. In questo testo, usiamo solo **edge-triggered clocking**. Ciò significa che tutti i cambiamenti di stato avvengono su un fronte di clock. Usiamo un edge-triggered metodologia perché è più semplice da spiegare. A seconda della tecnologia, che può o non può essere la scelta migliore per un **clocking metodologia**.

edge-triggered clocking Un clock regime in cui tutti i cambiamenti di stato avvengono su un fronte di clock.

clocking metodologiadologia L'approccio utilizzato per determinare quando i dati sono validi e stabile relativo al clock.

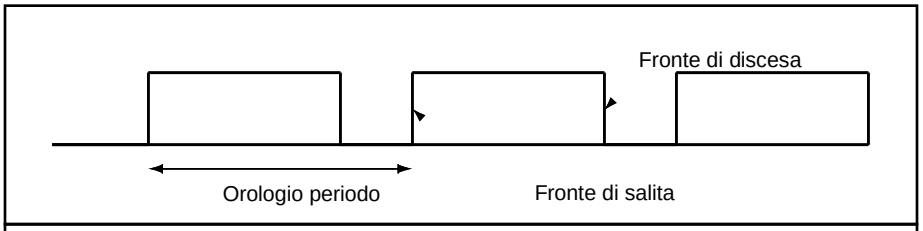


FIGURA C.7.1 Un segnale di clock oscilla tra valori alti e bassi. Il periodo di clock è il tempo per un ciclo completo. In un edge-triggered progettazione, sia il fronte di salita o di discesa del clock è attivo e provoca lo stato di essere cambiato.

Ion un edge-triggered metodologia, il fronte di salita o al fronte di discesa del clock è *attivo* e provoca cambiamenti di stato che si verifichi. Come vedremo nel prossimo sec-zione, la **elementi stato** in un edge-triggered progettazione vengono implementate in modo che il contenuto degli elementi di stato cambia solo sul fronte di clock attiva. La scelta di quale lato attivo è influenzato dalla tecnologia di implementazione e non influenza i concetti coinvolti nella progettazione logica.

stato elemento Una memoria elemento.

sistema sincrono Un sistema di memoria che impiega orologi e in cui i segnali di dati vengono letti solo quando l'orologio indica che i valori dei segnali sono stabili.

Il fronte di clock agisce come un segnale di campionamento, causando il valore dell'ingresso dati di stato di un elemento da campionare e memorizzati nell'elemento stato. Utilizzando un trigger bordo significa che il processo di campionamento è essenzialmente istantanei, eliminando i problemi che potrebbero verificarsi se i segnali sono stati campionati in tempi leggermente diversi.

Il vincolo principale di un sistema di clock, chiamato anche **sistema sincrono**, È che i segnali che sono scritte in elementi statali devono essere *valido* quando attivo

fronte di clock si verifica. Un segnale è valido se è stabile (cioè, non cambia), e il valore non cambierà nuovamente fino al cambio ingressi. Dal momento che i circuiti combinatori non può avere un feedback, se gli ingressi ad una unità logica combinatoria non sono cambiati, le uscite finirà per diventare valido.

Figura C.7.2 mostra la relazione tra gli elementi di stato e le combi-nazionali blocchi logici in un sincrono, progettazione logica sequenziale. Lo stato elementi, le cui uscite cambiare solo dopo che il fronte del clock, fornire input validi al blocco logica combinatoria. Per garantire che i valori scritti negli elementi di stato sul fronte di clock attiva sono valide, l'orologio deve avere un periodo abbastanza lungo in modo che tutti i segnali del blocco di logica combinatoria stabilizzare, i campioni del segnale di clock tali valori per memorizzazione dello stato elementi. Questo vincolo pone un limite inferiore alla lunghezza del periodo di clock, che deve essere sufficientemente lungo per gli ingressi degli elementi tutte le statali a essere valido.

Ion il resto di questa appendice, così come nel Capitolo 4, di solito si omette il segnale di clock, in quanto si presuppone che tutti gli elementi di stato vengono aggiornati sul bordo stesso clock. Alcuni elementi di stato sarà scritto su ogni fronte di clock, mentre altre saranno scritti solo in determinate condizioni (ad esempio, un registro in fase di aggiornamento). In questi casi, si avrà un segnale esplicito scrittura per quell'elemento stato. Il segnale di scrittura deve essere ancora gated con l'orologio in modo che l'aggiornamento si verifica solo sul fronte del clock, se il segnale di scrittura è attivo. Vedremo come questo è fatto e utilizzato nella sezione successiva.

Un altro vantaggio di un edge-triggered metodologia è che è possibile avere un elemento di stato che viene utilizzato sia come ingresso e uscita per la stessa combi-nazionale blocco logico, come mostrato nella Figura C.7.3. In pratica, la cura deve essere presa per evitare gare in tali situazioni e per garantire che il periodo di clock è abbastanza lungo, questo argomento è discusso ulteriormente nella sezione C.11.

Now che abbiamo discusso di come clock viene utilizzato per aggiornare gli elementi di stato, possiamo discutere come costruire gli elementi di stato

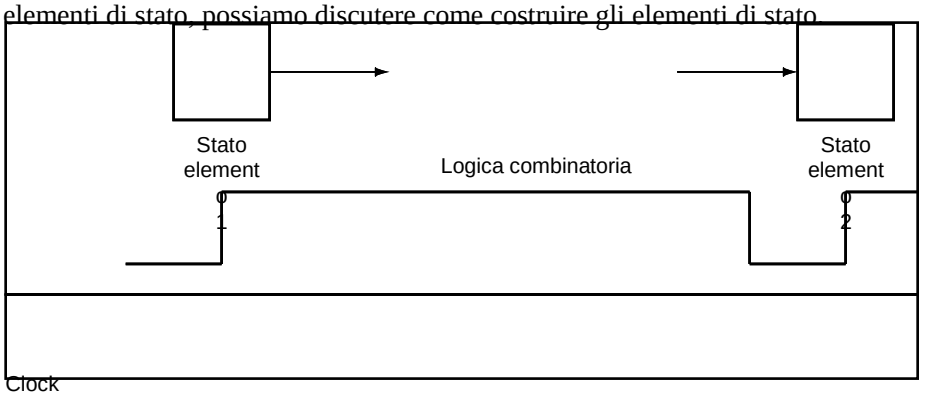


Figura C.7.2 Gli ingressi di un blocco di logica combinatoria provengono da un elemento di stato, e le uscite sono scritti in un elemento di Stato. Il fronte di clock determina quando il contenuto degli elementi di stato vengono aggiornati.

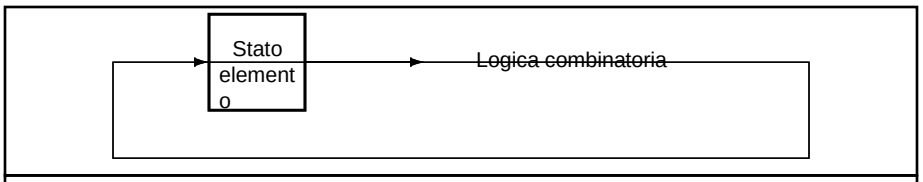


Figura C.7.3 Un edge-triggered metodologia permette un elemento di stato di essere letti e scritti nel ciclo di clock stesso senza creare una gara che potrebbe portare a indeterminati i valori dei dati. Naturalmente, il ciclo di clock deve essere ancora abbastanza lungo in modo che i valori di ingresso sono stabili quando il fronte di clock attivo avviene.

Elaborazione: Occasionalmente, i progettisti trovare utile avere un piccolo numero di elementi che cambiano di stato sul fronte del clock opposto dalla maggioranza dello stato elementi. Ciò richiede una cura estrema, perché tale approccio ha effetti su entrambi gli ingressi e le uscite dello stato dell'elemento. Perché, allora, avrebbe progettisti mai fare questo? Si consideri il caso in cui la quantità di logica combinatoria prima e dopo un elemento di stato è abbastanza piccolo in modo che ciascuno possa operare in una metà del ciclo di clock, piuttosto che il ciclo di clock più usuale piena. Quindi l'elemento di stato può essere scritta sul fronte di clock corrispondente a un ciclo di clock mezzo, poiché gli ingressi e le uscite saranno entrambi utilizzabile dopo metà ciclo di clock. Un luogo comune in cui viene utilizzata questa tecnica è in registrare i file, Dove la semplice lettura o la scrittura del file di registro può spesso essere fatto in metà del normale ciclo di clock. Capitolo 4 fa uso di questa idea per ridurre il sovraccarico pipelining.

regIster file di
LaStato elemento costituito
di una serie di registri che possono essere letti e scritti fornendo un numero di registro a cui accedere.

C.8 Elementi di memoria: infradito, Chiavistelli, e Registri

Ion questa sezione e l'altra, discutiamo i principi di base della memoria ele-menti, a partire da flip-flop e fermi, di passare a registrare i file, e finitura di memorie. Tutti gli elementi di memoria negozio stato: l'uscita da qualsiasi elemento di memoria dipende sia degli ingressi e il valore che è stato memorizzato all'interno dell'elemento di memoria. Così tutti i blocchi che contengono un elemento di memoria contenere dello stato-e sono sequenziali.

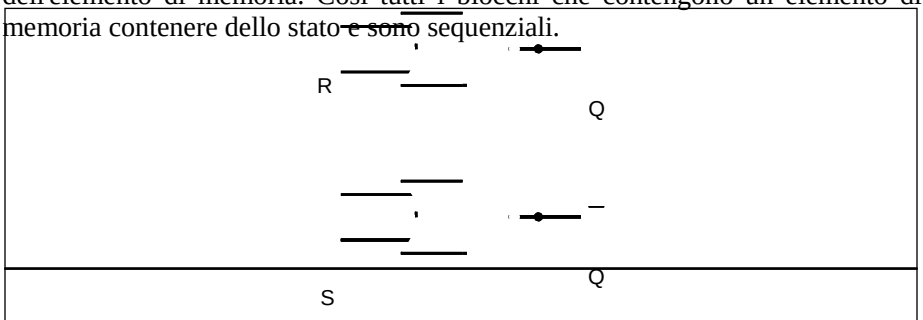


Figura C.8.1 Un paio di cross-coupled porte NOR in grado di memorizzare un valore interno. Il valore memorizzato in uscita *Q*è riciclato invertendo ottenere *Q*e poi invertente *Q*to ottenere *Q*. Se uno dei due *R*o *Q*si afferma, *Q*wmalato essere deasserito e viceversa.

Il tipo più semplice di elementi di memoria sono *unclocked;*cioè, non hanno alcun ingresso di clock. Anche se usiamo solo elementi di memoria con clock in questo testo, un latch unclocked è l'elemento più semplice di memoria, quindi diamo un'occhiata a questo primo circuito. La figura mostra un C.8.1 *S-R latch* (Set-reset fermo), costruito da una coppia di porte NOR (OR porte con uscite invertite). Le uscite *Q*e *Q*represent il valore dello stato memorizzato e il suo complemento. Quando nessuno dei due *S*né *R*si affermano, il cro ss-coupled porte NOR agiscono come inverter e memorizzare i valori precedenti di *Q*e *Q*.

For esempio, se l'uscita, *Q*, È vero, allora l'inverter inferiore produce un'uscita falso (che è *Q*), Che diventa l'ingresso all'invertitore superiore, che produce un output vero, che è *Q*, E così via. Se *S*è asserito, l'uscita *Q* sarà affermata e *Q*wmalato essere deasserito, mentre se *R*è asserito, l'uscita *Q*wmalato be affermata e *Q*wmalato essere deasserito. Quando *S*e *R*sono entrambi deasseriti, gli ultimi valori di *Q*e *Q*will continuare ad essere memorizzati nella struttura ad accoppiamento incrociato. Affermare *S*e *R*allo stesso tempo può provocare un funzionamento non corretto: a seconda di come *S*e *R*sono deasserito, il fermo può oscillare o diventare metastabile (questo è descritto in dettaglio nella sezione C.11).

Questo accoppiamento incrociato struttura è la base per più elementi di memoria complessi che ci permettono di memorizzare i segnali di dati. Questi elementi contengono porte aggiuntive utilizzate per memorizzare valori di segnale e di provocare lo stato di essere aggiornati solo in combinazione con un orologio. La sezione successiva mostra come questi elementi sono costruiti.

Flip-flop e Chiusure

Flip-flop e **fermi** sono gli elementi semplici di memoria. In entrambi flip-flop e fermi, l'uscita è uguale al valore dello stato memorizzato all'interno dell'elemento. Inoltre, a differenza del latch SR sopra descritto, tutte le serrature e flip-flop useremo da questo punto in poi sono clock, il che significa che essi hanno un ingresso di clock e il cambiamento di stato è attivato da tale orologio. La differenza tra un flip-flop e un latch è il punto in cui l'orologio provoca l'effettiva modifica di stato. In un latch clock, lo stato viene cambiato ogni volta corrispondenti ingressi cambiano e l'orologio è asserito, mentre in un flip-flop, lo stato viene modificato solo su un fronte di clock. Dal momento che in tutto il testo si usa un edge-triggered metodologia tempi in cui lo stato viene aggiornato solamente sui fronti di clock, è necessario utilizzare solo flip-flop. Flip-flop sono spesso costruiti da fermi, quindi cominceremo a descrivere il funzionamento di un dispositivo di chiusura semplice clock e poi discutere il funzionamento di un flip-flop costruito da tale chiusura.

uscite sono semplicemente il valore dello stato interno (*Q*) E

Foapplicazioni informatiche r, la funzione sia di flip-flop e latch è quello di memorizzare un segnale. La *D latch* o **D flip-flop** memorizza il valore del segnale dati in ingresso nella memoria interna. Anche se ci sono molti altri tipi di latch e flip-flop, il tipo D è l'unico elemento di base che abbiamo bisogno. Un latch D ha due ingressi e due uscite. Gli ingressi sono il valore di dati da memorizzare (chiamato *D*) E un segnale di clock (chiamato *C*) Che indica quando il dispositivo di chiusura deve leggere il valore sul *D*input e memorizzarlo. Le

flip-flop Una memoria elemento per cui l'uscita è uguale al valore dello stato memorizzato all'interno dell'elemento e per cui lo stato interno è cambiato solo su un fronte di clock.

chiavistello Laelemento di memoria in cui l'uscita è uguale al valore dello stato memorizzato all'interno dell'elemento e lo stato viene cambiato ogni volta la modifica appropriata ingressi e l'orologio è asserito.

D flip-flop Laflip-flop con un ingresso di dati che memorizza il valore del segnale di ingresso nella memoria interna, quando il bordo clock.

suo complemento $\overline{(Q)}$. Quando il clock di ingresso *C*è asserito, il latch è detto essere *Open*, E il valore dell'uscita (*Q*) Diventa il valore dell'ingresso *D*. Quando il clock di ingresso *C*è deasserito, il latch è detto essere *chiuso,*e il valore della potenza (*Q*)è qualsiasi valore è stato memorizzato l'ultima volta che il fermo era aperta.

Figura C.8.2 mostra come un latch D può essere implementato con due porte addizionali aggiunti al cross-coupled porte NOR. Da quando il coperchio è aperto il valore di *Q*cHanges come *D*cHanges, questa struttura è talvolta chiamato *trfermo ansparent*. Figura C.8.3 mostra come funziona latch D, supponendo che l'uscita *Q*è inizialmente falsa e che *D*cHanges prima.

Las accennato in precedenza, si usa flip-flop come il blocco di base, piuttosto che fermi. Flip-flop non sono trasparenti: le loro uscite cambia *solo* sul fronte di clock. Un flip-flop può essere costruito in modo che innesca né sulla salita (positivo) o di discesa (negativo) fronte di clock, per i nostri progetti possono utilizzare entrambi i tipi. Figura C.8.4 mostra come un fronte di discesa D flip-flop è costruito da una coppia di D latch. In un D flip-flop, l'uscita viene memorizzato quando il bordo clock. Figura C.8.5 mostra come questo flip-flop opera.

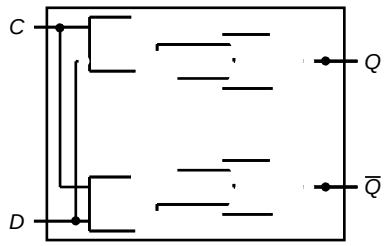


FIGURA C.8.2 fermo AD realizzato con porte NOR. LaPorta NOR comporta come un invertitore se l'altro ingresso è 0. Così, l'accoppiamento incrociato coppia di porte NOR agisce per memorizzare il valore di stato a meno del clock, *C*, Si afferma, nel qual caso il valore dell'ingresso *D*replaces il valore di *Q*e viene memorizzato. Il valore dell'ingresso *D*must stabile quando il segnale di clock *C*cHanges da affermato a deasserito.

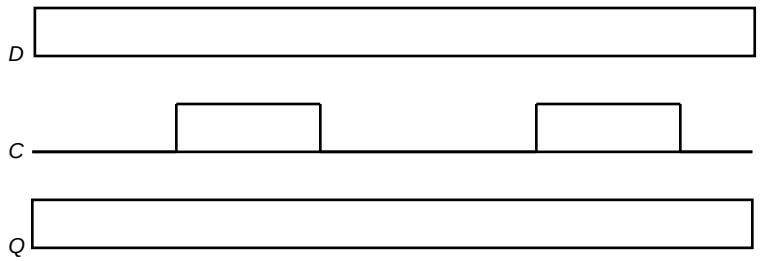
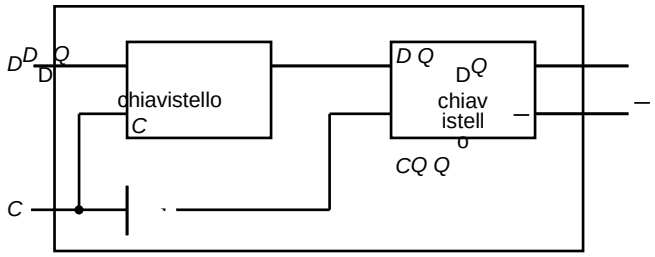


FIGURA C.8.3 Funzionamento di un latch D, supponendo l'uscita è inizialmente deasserito. Wgallina l'orologio, *C*, È asserito, il coperchio è aperto e la *Q*uscita assume immediatamente il valore del *D*ingresso.



C.8.4 FIGURA AD flip-flop con un fronte di discesa di trigger. Primo latch, denominato master, è aperta e segue l'ingresso *D*quando il clock in ingresso, *C*,si afferma. Quando il clock in ingresso, *C*, Cadute, primo latch è chiusa, ma il fermo secondo, detto slave, è aperto e riceve l'input dal uscita del latch principale.

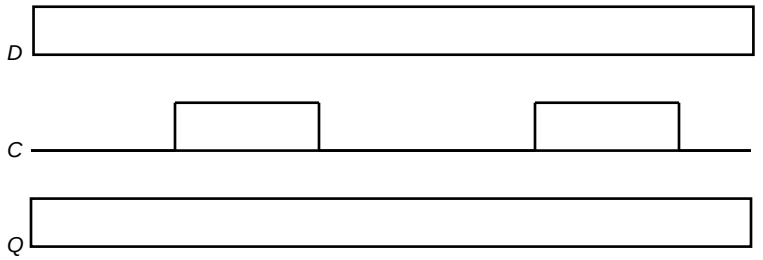


FIGURA C.8.5 Funzionamento di un D flip-flop con un fronte di discesa di trigger, assumendo l'uscita è inizialmente deasserito. Wuando il clock di ingresso (*C*) Passa da asserito per deasserito, il *Q*uscita memorizza il valore del *D*ingresso. Confronta questo comportamento a quello del latch D clock mostrato in Figura C.8.3. In un latch clock, il valore memorizzato e l'uscita, *Q*, ogni volta che sia il cambiamento *C*è alto, contro soltanto quando *C*transitions.

Suoe una descrizione Verilog di un modulo per un fronte di salita D flip-flop, assumendo che C è l'ingresso di clock e D è l'ingresso di dati:

```
Modulo DFF (orologio, D, Q, Qbar);
    ingresso di clock, D;
    uscita reg Q / / Q è un reg in quanto viene assegnato in un blocco di sempre
    uscita Qbar;
    assegnare Qbar ~ = Q / / Qbar è sempre solo l'inverso di Q
    sempre @ (orologio posedge) / / eseguire azioni ogni volta che
l'orologio si erge
        Q= D;
endmodule
```

Becausethe*D*input è campionato sul fronte di clock, esso deve essere valido per un periodo di tempo immediatamente prima ed immediatamente dopo il fronte di clock. Il tempo minimo che l'ingresso deve essere valido prima del fronte del clock è chiamato **setup tempo**; L'

tempo di setup Il tempo minimo che l'ingresso di un dispositivo di memoria deve essere valido prima del fronte di clock.

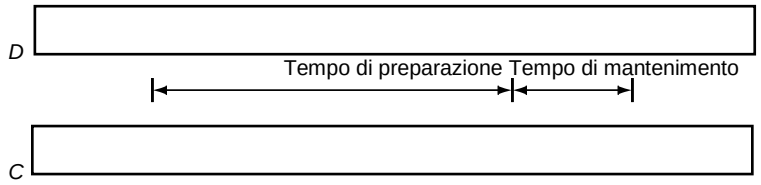


FIGURA C.8.6 Installazione e tenere requisiti di tempo per un flip-flop D con un fronte di discesa di trigger. L'ingresso deve essere stabile per un periodo di tempo prima del fronte di clock, e dopo il fronte di clock. Il tempo minimo del segnale deve essere stabile prima che il fronte di clock è chiamato il tempo di installazione, mentre il tempo minimo del segnale deve essere stabile dopo il fronte di clock è chiamato tempo di attesa. Mancato rispetto di questi requisiti minimi può comportare una situazione in cui l'uscita del flip-flop può non essere prevedibile, come descritto nella Sezione C.11. Contenere i tempi di solito non sono 0 o molto piccolo e quindi una causa di preoccupazione.

tempo di attesa Il tempo minimo durante il quale l'ingresso deve essere valido dopo il fronte di clock.

minimum Tempo durante il quale deve essere valido dopo che il fronte del clock è chiamato il **tempo di attesa**. Gios gli ingressi a qualsiasi flip-flop (o qualcosa di costruito con flip-flop) deve essere valido nel corso di una finestra che inizia al momento t_{setup}before il fronte di clock e termina a t_{hold}after il fronte di clock, come mostrato nella Figura C.8.6. Sezione C.11 parla di clock e tempistica vincoli, tra cui il ritardo di propagazione attraverso un flip-flop, in modo più dettagliato.

We può utilizzare una matrice di D flip-flop di costruire un registro che può contenere un dato multibit, come un byte o parola. Abbiamo usato registri tutta la nostra unità di elaborazione nel Capitolo 4.

Registrare i file

Una struttura che è al centro della nostra unità di elaborazione è un *regIster file di*.Un file di registro è costituito da un insieme di registri che possono essere letti e scritti fornendo un numero di registro a cui accedere. Un file di registro può essere implementato con un decoder per ogni lettura o scrittura di porta e una serie di registri costruito da D flip-flop. Poiché la lettura di un registro non cambia qualsiasi stato, abbiamo bisogno solo di fornire un numero di registro come input e l'output solo saranno i dati contenuti nel registro. Per la scrittura di un registro dovremo tre ingressi: un numero di registro, i dati da scrivere, e un orologio che controlla la scrittura nel registro. Nel Capitolo 4, abbiamo utilizzato un file di registro che dispone di due porte di lettura e una porta di scrittura. Questo file di registro viene disegnata come mostrato in Figura C.8.7. Le porte di lettura possono essere realizzati con una coppia di multiplexer, ciascuno dei quali è largo quanto il numero di bit in ciascun registro del file di registro. Figura C.8.8 mostra la realizzazione di due porte per leggere registro a 32-bit di ampiezza file di registro.

IoTTUAZIONE la porta di scrittura è leggermente più complessa, dal momento che può solo modificare il contenuto del registro designato. Possiamo farlo utilizzando un decodificatore per generare un segnale che può essere utilizzato per determinare quale registrati per scrivere. Figura C.8.9 mostra come implementare la porta di scrittura di un file di registro. È importante ricordare che

il flip-flop cambia stato solo sul fronte di clock. Nel Capitolo 4, abbiamo collegato i segnali per scrivere il file di registro in modo esplicito ed ha assunto l'orologio visualizzato in figura C.8.9 è collegato in modo implicito.

Wcappello succede se lo stesso registro è letto e scritto nel corso di un ciclo di clock? Perché la scrittura del file di registro avviene sul fronte di clock, il registro sarà

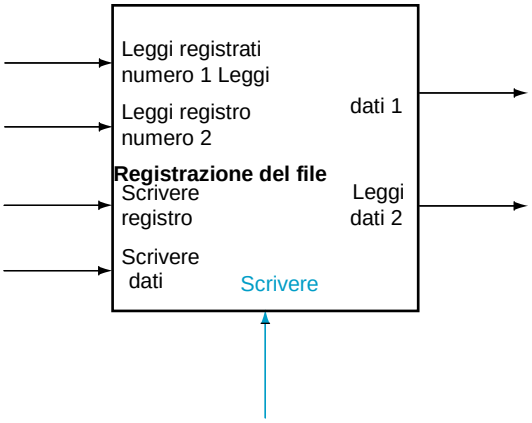


Figura C.8.7 Un file registro con due porte di lettura e una porta di scrittura ha cinque ingressi e due uscite. La scrittura ingresso di controllo è illustrato a colori.

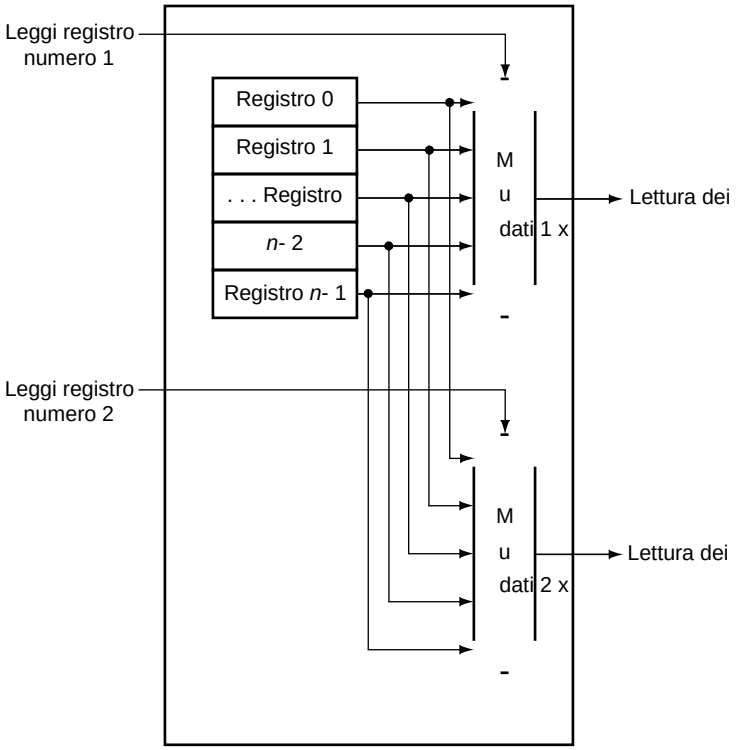


Figura C.8.8 L'attuazione di due porte di lettura di un file di registro con *n*registri può essere fatto con una coppia di *n*-A-1 multiplexer, ogni 32 bit. Il registro di lettura del segnale numero viene utilizzato come segnale multiplexor selettore. Figura C.8.9 mostra come la porta di scrittura è implementato.

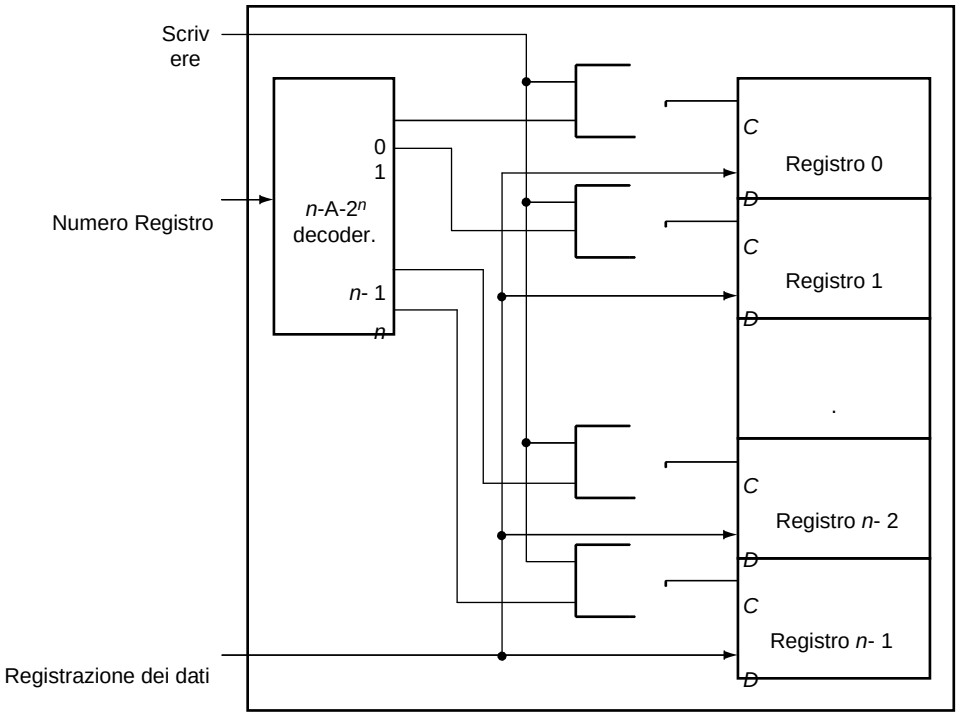


FIGURA C.8.9 La porta di scrittura per un file di registro viene implementato con un decoder che viene utilizzato con il segnale di scrittura per generare l' Cingresso ai registri. Tutti e tre ingressi (il numero di registro, i dati, ed il segnale di scrittura) avrà vincoli di installazione e di hold-time che assicurano che i dati corretti scritti nel file registro.

valido durante il tempo che viene letto, come abbiamo visto in precedenza in figura C.7.2. Il valore restituito sarà il valore scritto in un ciclo di clock precedente. Se vogliamo una legge per restituire il valore attualmente in fase di scrittura, logica aggiuntiva nel file di registro o al di fuori di esso è necessario. Capitolo 4 fa ampio uso di questa logica.

Specifica logica sequenziale in Verilog

To specificare logica sequenziale in Verilog, dobbiamo capire come generare un orologio, come descrivere quando un valore viene scritto in un registro, e come specificare il controllo sequenziale. Cominciamo specificando un orologio. Un orologio non è un oggetto predefinito in Verilog, invece, generiamo un orologio utilizzando la notazione Verilog # n prima che una dichiarazione, il che causa un ritardo di passi di simulazione in tempo n prima esecuzione dell'istruzione. Nella maggior parte dei simulatori Verilog, è anche possibile generare un orologio come ingresso esterno, consentendo all'utente di specificare al tempo di simulazione il numero di cicli di clock durante il quale eseguire una simulazione.

Il codice nella Figura C.8.10 implementa un semplice orologio che è alta o bassa per una unità di simulazione e passa quindi allo stato. Usiamo la funzione

di ritardo e l'assegnazione di blocco per l'attuazione del orologio.

```
orologio reg; / / orologio è un
registro sempre
    # 1 orologio = 1; # 1 orologio = 0;
```

FIGURA C.8.10 Una specifica di un orologio.

Next, dobbiamo essere in grado di specificare il funzionamento di un registro edge-triggered. In Verilog, questo viene fatto utilizzando l'elenco sensibilità sempre su un blocco e specificando come trigger il fronte positivo o negativo di una variabile binaria con la nota-zione posedge o negedge, rispettivamente. Quindi, il codice seguente Verilog provoca registro A essere scritto con il valore di b l'orologio fronte di salita:

```
    reg [31:0] A;
    filo [31:0] b;

    sempre @ (orologio posedge) A  <= B;
```

```
module registerfile (Leggi1, READ2, WriteReg, WriteData, RegWrite,
data1, data2, orologio);
   ingresso [05:00] READ1, READ2, WriteReg, / / i
numeri di registro per leggere o scrivere
   ingresso [31:0] WriteData; / / dati da
   scrivere RegWrite input, / / il controllo di
   scrittura
     orologio, / / l'orologio per attivare scrivere
   uscita [31:0] data1, data2, / / i valori di registro di lettura
   reg [31:0] RF [31:0], / / 32 registra ogni 32 bit di lunghezza

   assegnare Data1 RF = [Leggi1];
   assegnare Data2 RF = [READ2];

   iniziano sempre
      / / Scrivere il registro con nuovo valore se è RegWrite
alto
       @ (Orologio posedge) se (RegWrite) RF [WriteReg] <=
WriteData;
   fine
endmodule
```

FIGURA C.8.11 Un file di registro MIPS scritto in Verilog comportamentale. Questo file registro scrive sul fronte di clock di salita.

In questo capitolo e le sezioni Verilog del capitolo 4, si suppone un positive edge-triggered design. Figura C.8.11 mostra una specifica Verilog di un file MIPS registro che assume due operazioni di lettura e scrittura, uno con solo la scrittura viene cronometrato.

Controllare Ynoi stessi

Ion il Verilog per il file registro in Figura C.8.11, le porte di uscita corrispondenti ai registri in lettura sono assegnati utilizzando un'assegnazione continuo, ma il registro viene scritta non è sempre assegnato in un blocco. Quale dei seguenti è il motivo?

un. Non vi è alcun motivo particolare. E 'stato semplicemente conveniente.

b. Poiché Data1 e Data2 sono porte di uscita e WriteData è una porta di ingresso.

c. Perché la lettura è un evento combinatoria, mentre la scrittura è un evento sequenziale.

C.9 Elementi di memoria: SRAM e DRAM

memoria statica ad accesso casuale (SRAM)
Lamemoria in cui sono memorizzati i dati staticamente (come nel flip-flop) piuttosto che dinamicamente (come in DRAM). SRAM sono più veloci DRAM, ma meno denso e più costosi per bit.

Regstri e registrare file fornire gli elementi di base per le memorie di piccole dimensioni, ma grandi quantità di memoria sono costruiti utilizzando **SRAM (Statiche memorie ad accesso casuale)** o *DRAM* (Memorie dinamiche ad accesso casuale). Per prima cosa discutere SRAM, che sono un po 'più semplice, e poi girare a DRAM.

SRAM

SRAM sono semplicemente circuiti integrati che sono array di memoria con (di solito) una porta di accesso unico in grado di fornire sia una lettura o una scrittura. SRAM hanno un tempo fisso accesso a qualsiasi dato, anche se la lettura e scrittura caratteristiche di accesso spesso differiscono. Un chip SRAM ha una configurazione specifica in termini di numero di indirizzabile
posizioni, così come la larghezza di ogni locazione indirizzabile. Ad esempio, un 4M × 8
SRAM fornisce le voci 4M, ciascuno dei quali è di 8 bit. Così avrà 22 linee di indirizzo (da 4M =222), a 8 bit di uscita linea dati, e un 8-bit singolo ingresso linea dati.
Laswith ROM, il numero di posizioni indirizzabili è spesso chiamata la *altezza*, Con il numero di bit per unità chiamata *wIDb*. Per una serie di ragioni tecniche, le SRAM ultime e più veloci sono in genere disponibili in configurazioni strette: × 1 e × 4. Figura C.9.1 mostra i segnali di ingresso e di uscita per una 2M × 16 SRAM.

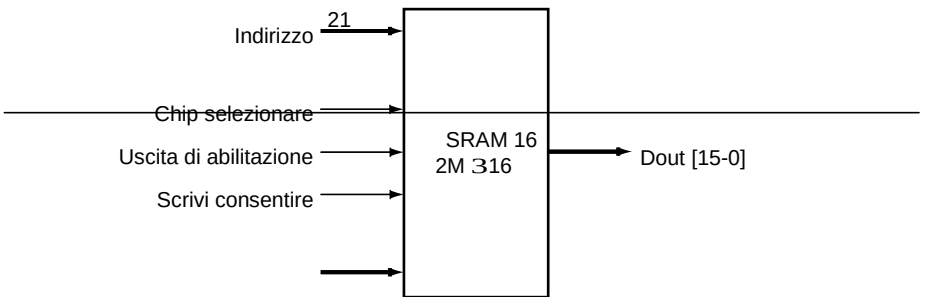


Figura C.9.1 A 32K ×8 SRAM che mostra le 21 linee di indirizzo (32K = 215) e 16 ingressi di dati, le 3 linee di controllo, e le 16 uscite di dati.

To avviare una lettura o scrittura, il segnale di selezione chip deve essere reso attivo. Per legge, dobbiamo anche attivare il segnale di abilitazione uscita che controlla se il dato selezionato dal indirizzo è effettivamente azionato sui perni. L'abilitazione uscita è utile per il collegamento di memorie multiple ad un singolo bus di uscita e l'utilizzo di uscita consentono di determinare quale guida il bus di memoria. L'accesso SRAM lettura di tempo di solito è specificato come il ritardo dal momento in cui l'uscita di abilitazione è vero e le linee di indirizzo sono valide fino al momento in cui i dati sono sulle linee di uscita. Tipica leggere i tempi di accesso per SRAM nel 2004 variava da circa 2-4 ns per le parti più veloci CMOS, che tendono ad essere un po 'più piccole e strette, a 8-20 ns per le parti più tipiche, che nel 2004 avevano più di 32 milioni di bit dei dati. La domanda di SRAM a bassa potenza per i prodotti di consumo ed elettrodomestici digitali è cresciuta notevolmente negli ultimi cinque anni; questi SRAM hanno molto più basso di stand-by e il potere di accesso, ma di solito sono 5-10 volte più lento. Più di recente, sincrono SRAM-simile alle DRAM sincrone, di cui parleremo nella prossima sezione, sono stati sviluppati.

For scrive, si devono fornire i dati da scrivere e l'indirizzo, nonché segnali di causare la scrittura a verificarsi. Quando sia la scrittura attivare e Chip select sono vere, i dati sulle linee di ingresso i dati vengono scritti nella cella specificata dall'indirizzo. Ci sono setup-tempo e hold-time requisiti per le linee di indirizzo e dei dati, così come ci sono stati per flip-flop D e si blocca. Inoltre, il segnale di abilitazione alla scrittura non è un fronte di clock, ma un impulso con un requisito di larghezza minima. Il tempo necessario per completare una scrittura è specificato dalla combinazione dei tempi di installazione, i tempi di attesa, e l'abilitazione di ampiezza di impulso Write.

SRAM di grandi dimensioni non può essere costruito nello stesso modo si costruisce un file di registro perché, a differenza di un file di registro in cui un 32-a-1 multiplexor potrebbe essere pratico, il 64K-a-1
multiplexor che sarebbero necessari per un 64K × 1 SRAM è del tutto impraticabile. Piuttosto
di utilizzare un multiplexer gigante, memorie di grandi dimensioni sono realizzati con una comune
linea di uscita, chiamata *linea di bit*,quali celle di memoria multipli nella matrice di memoria può affermare. Per consentire più fonti di guidare una singola linea, un *tre stato cuscinetto* (O *triState tampone*) Viene utilizzato. A tre stato cuscinetto ha due ingressi-un segnale di dati e una abilitazione uscita e una singola uscita, che è in uno dei tre stati: asserito, deasserito, o alta impedenza. L'uscita di un buffer tristate è uguale al segnale dati in ingresso, sia asserito o deasserito, se l'uscita di abilitazione è asserito, ed è altrimenti in un *stato di alta impedenza* che permette un altro three-state buffer il cui abilitazione uscita è asserito per determinare il valore di una uscita comune.

Figura C.9.2 mostra un insieme di buffer tri-state cablati per formare un multiplexer con un ingresso decodificato. E 'fondamentale che l'uscita consentono al massimo di uno dei buffer tri-state essere rivendicati, in caso contrario, i buffer tri-state può tentare di impostare la linea di uscita in modo diverso. Utilizzando buffer tri-state nelle singole celle della SRAM, ogni cella che corrisponde ad una particolare uscita possono condividere la stessa linea di uscita. L'uso di un

insieme di distribuiti buffer tri-state è una implementazione più efficiente di un multiplexer grande centralizzata. I buffer tri-state sono incorporati nel flip-flop che forma le cellule di base della SRAM. Figura C.9.3 mostra come una piccola 4× 2 SRAM potrebbe essere costruito, utilizzando D latch con un ingresso chiamato Abilita i controlli l'uscita a tre stati.

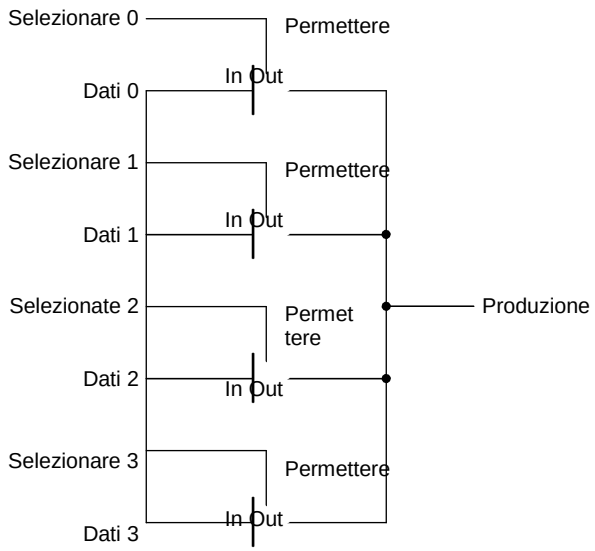


FIGURA C.9.2 Quattro buffer tri-state sono usati per formare un multiplexer. Solo uno dei quattro ingressi Select possono essere esercitati. A tre stato cuscinetto con una uscita di abilitazione deasserito ha una elevata impedenza di uscita che consente a tre stati cuscinetto di abilitazione uscita è asserito per pilotare la linea condivisa uscita.

Il disegno in figura C.9.3 elimina la necessità di un multiplexer enorme; tuttavia, esso richiede comunque un decoder molto grande e un numero più elevato di linee di parola. Ad esempio, in un 4M × 8 SRAM, avremmo bisogno di un 22-a-4M decoder
e linee di parola 4M (che sono le linee usate per consentire alla persona flip-flop)! Per ovviare a questo problema, i ricordi di grandi dimensioni sono organizzati come matrici rettangolari e
utilizzare un processo a due fasi di decodifica. Figura C.9.4 mostra come una 4M × 8 SRAM potrebbe essere
organizzate internamente usando due passaggi di decodifica. Come vedremo, i due livelli di decodifica
processo è molto importante per capire come funzionano le DRAM.

Recently abbiamo visto lo sviluppo di entrambe le SRAM sincrone (SSRAMs) e le DRAM sincrona (SDRAM). La capacità chiave fornita da RAM sincrona è la capacità di trasferire un *scoppio* di dati da una serie di indirizzi in sequenza all'interno di un array o una riga. Il burst è definito da un indirizzo di partenza, fornito nel modo usuale, e una lunghezza del burst. Il vantaggio della velocità di RAM sincrona deriva dalla capacità di trasferire i bit nel burst senza dover specificare ulteriori bit di indirizzo. Invece, un orologio è utilizzato per trasferire i bit successivi burst. L'eliminazione della necessità di specificare l'indirizzo per i trasferimenti all'interno del burst migliora significativamente il tasso di trasferimento del blocco di dati. A causa di questa capacità, SRAM e DRAM sincroni stanno rapidamente diventando la RAM di scelta per la costruzione di sistemi di memoria nel computer. Discutiamo l'uso di DRAM sincrone in un sistema di memoria in dettaglio nella sezione successiva e nel capitolo 5.

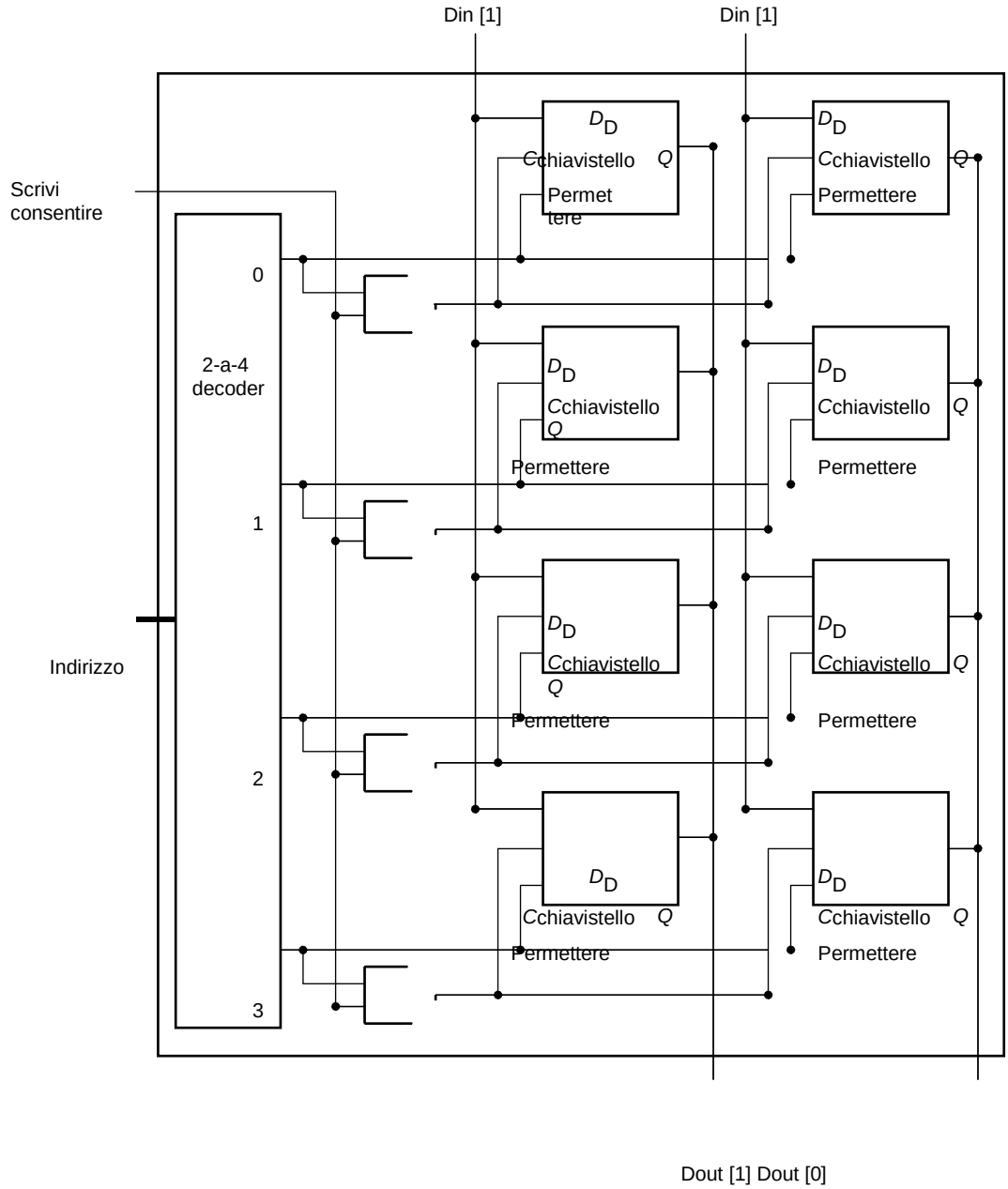


FIGURA C.9.3 La struttura di base di un 4 ×2 SRAM consiste di un decoder che seleziona quale coppia di celle per attivare. Le cellule attivate utilizzare tre stati uscita collegata alle bit verticali che forniscono i dati richiesti. L'indirizzo che seleziona la cella viene inviato su uno di un insieme di linee di indirizzo orizzontali, dette linee di word. Per semplicità, l'uscita attivare e Chip segnali di selezione sono stati omessi, ma potrebbero facilmente essere aggiunte con pochi porte AND.

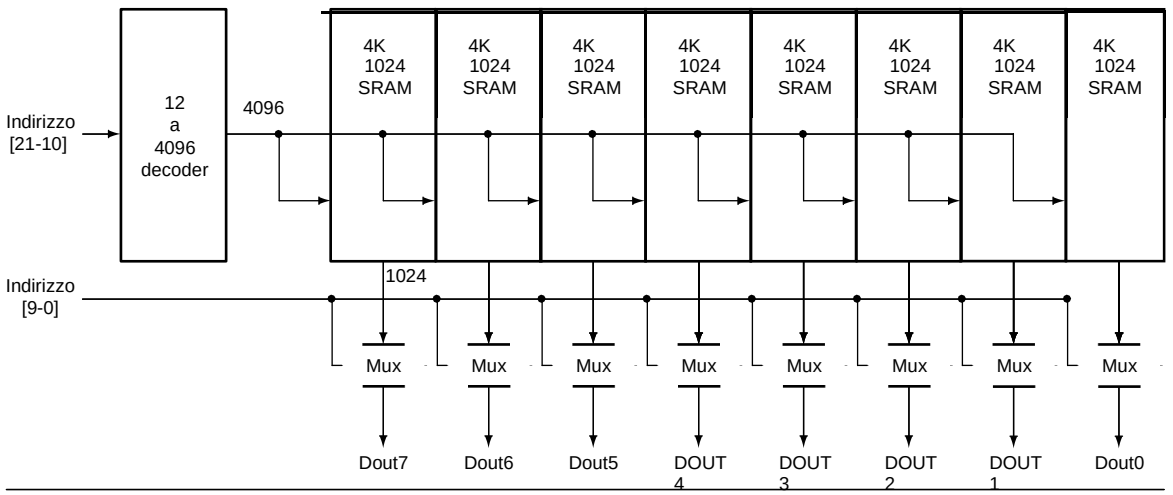


FIGURA C.9.4 organizzazione tipica di un 4M ×8 SRAM come un array di 4K ×1024 array. Il primo decoder genera gli indirizzi per otto 4K × 1024 array; poi una serie di multiplexer viene utilizzato per selezionare 1 bit per ogni 1024 bit di ampiezza array. Questo è molto più facile
progettazione di un singolo livello di decodifica, che avrebbe bisogno di un decoder o enorme o un multiplexer gigantesco. In pratica, una SRAM moderna di queste dimensioni probabilmente utilizzare un numero ancora maggiore di blocchi, ciascuno alquanto minore.

DRAM

Iona RAM statica (SRAM), il valore memorizzato in una cella è mantenuta su una coppia di porte invertenti, e fintanto che è alimentato, il valore può essere mantenuto indefinitamente. In una RAM dinamica (DRAM), il valore mantenuto in una cella viene memorizzata come una carica in un condensatore. Un singolo transistor viene poi utilizzato per accedere a questa carica immagazzinata, sia per leggere il valore o sovrascrivere la carica immagazzinata lì. Perché DRAM utilizzare un solo transistor-tore bit per di storage, sono molto più densi e più economico per bit. In confronto, SRAM richiedono 4-6 transistor per bit. Poiché DRAM immagazzinare la carica di un condensatore, non può essere mantenuto indefinitamente e devono essere periodicamente *refreshed*. Che Ecco perché questa struttura di memoria è chiamato *dynamic*,in contrapposizione alla memoria statica in una cella SRAM.

To aggiornamento della cella, ci si limita a leggere il suo contenuto e scrivere di nuovo. La carica può essere conservato per diversi millisecondi, che potrebbero corrispondere a quasi un milione di cicli di clock. Oggi, a chip singolo controller di memoria spesso gestire l'aggiornamento fun-zione indipendentemente dal processore. Se ogni bit doveva essere letti della DRAM e poi riscritte singolarmente, con DRAM di grandi dimensioni contenenti più mega-byte, ci sarebbe essere costantemente aggiornare la DRAM, che non lascia tempo per accedervi. Fortunatamente, DRAM anche utilizzare una struttura a due livelli di decodifica, e questo permette di aggiornare un'intera riga (che condivide una linea di parola) con un ciclo di lettura immediatamente seguito da un ciclo di scrittura. Tipicamente, le operazioni di aggiornamento consumare 1% al 2% dei cicli attivi della DRAM, lasciando il restante 98% al 99% dei cicli disponibili per la lettura e la scrittura di dati.

Elaborazione: Come fa un DRAM leggere e scrivere il segnale memorizzato in una cella? Il tran-sistore all'interno della cellula è un interruttore, chiamato *passare transistor*, Che permette il valore memorizzato nel condensatore a cui accedere per lettura o scrittura. Figura C.9.5 mostra come il singolo transistor cella sembra. Il transistore si comporta come un interruttore: quando il segnale sulla linea di parola è asserito, l'interruttore è chiuso, connettere il condensatore alla linea di bit. Se l'operazione di scrittura è, allora il valore da scrivere è posto sulla linea di bit. Se il valore è 1, il condensatore si carica. Se il valore è 0, il condensatore si scarica. Lettura è leggermente più complessa, in quanto la DRAM deve rilevare un costo molto piccolo immagazzinata nel condensatore. Prima di attivare la linea di parola per una lettura, la linea di bit è a carico del tensione che è a metà strada tra la bassa e alta tensione. Quindi, attivando la linea di parola, la carica sul condensatore viene letta sulla linea di bit. Ciò causa la linea di bit per spostare leggermente verso la direzione alta o bassa, e questo cambiamento viene rilevato con un amplificatore di rilevamento, che può rilevare piccole variazioni di tensione.

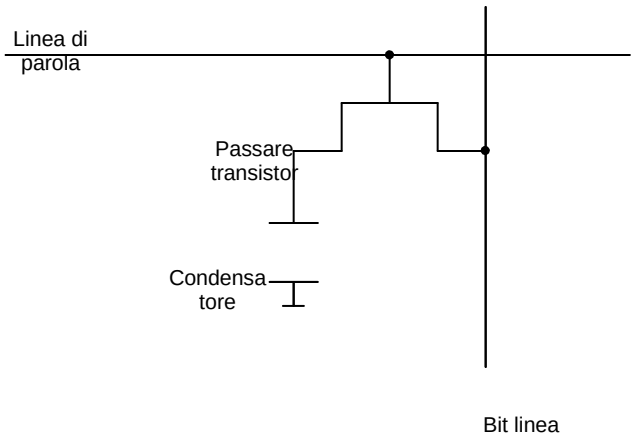


FIGURA C.9.5 Un singolo transistor cella DRAM contiene un condensatore che memorizza il contenuto della cella e un transistor utilizzato per accedere alla cella.

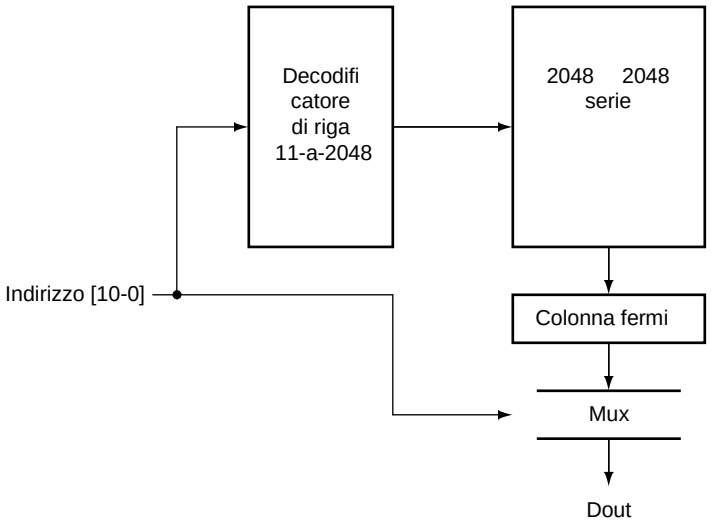


Figura C.9.6 A 4M ×1 DRAM è costruito con di 2048 ×2048 array. L'accesso riga utilizza 11 bit per selezionare una riga, che viene poi bloccato in 2048 1 bit fermi. Un multiplexer sceglie il bit di uscita da questi 2.048 fermi. Il controllo RAS e CAS segnali se le linee di indirizzo sono inviati al decodificatore di riga o colonna multiplexer.

DRAM utilizzare due livelli costituito da un decodificatore *row accesso* seguito by a *colonna di accesso*, come mostrato nella Figura C.9.6. L'accesso fila sceglie uno di un numero di righe e attiva la riga corrispondente parola. Il contenuto di tutte le colonne nella riga attiva vengono quindi memorizzati in una serie di fermi. L'accesso colonna seleziona i dati dalla colonna fermi. Per risparmiare perni e ridurre il costo del pacchetto, le linee di indirizzo stesse vengono utilizzate sia per l'indirizzo di riga e colonna; una coppia di segnali di chiamata RAS (Row Accesso Strobe) e CAS (Access Strobe colonna) sono utilizzati per segnalare la DRAM che o una riga o indirizzo di colonna viene fornita. Refresh viene eseguita semplicemente leggendo le colonne nella colonna fermi e poi scrivere gli stessi valori di ritorno. Così, l'intera riga viene aggiornata in un ciclo. Due livelli schema di indirizzamento, in combinazione con i circuiti interni, rendono DRAM tempi di accesso molto più lungo (di un fattore 5-10) di tempi di accesso SRAM. Nel 2004, i tempi di accesso tipici DRAM andata da 45 a 65 ns; DRAM 256 Mbit sono in piena produzione, ed i campioni dei clienti prima di 1 GB DRAM si sono resi disponibili nel primo trimestre del 2004. Il costo molto inferiore per bit rende la scelta DRAM per la memoria principale, mentre il tempo di accesso più veloce SRAM rende la scelta per cache.

You potrebbe osservare che un 64M × 4 DRAM accede effettivamente 8K bit su ogni roaccesso w e poi butta via tutto, ma 4 di quelli che nel corso di un accesso colonna. DRAM
progettisti hanno utilizzato la struttura interna della DRAM come un modo per fornire una maggiore larghezza di banda di una DRAM. Questo viene fatto consentendo l'indirizzo di colonna per cambiare senza cambiare l'indirizzo di riga, risultante in un accesso ad altri bit della colonna fermi. Per rendere questo processo più veloce e più preciso, gli ingressi di indirizzo sono stati clock, che porta alla forma dominante di DRAM in uso oggi: DRAM sincrona o SDRAM.

Since circa 1999, SDRAM sono il chip di memoria di scelta per la maggior parte di cache basati su sistemi di memoria principale. SDRAMs fornire un accesso veloce ad una serie di bit all'interno di una riga da trasferire sequenzialmente tutti i bit in un burst sotto il controllo di un segnale di clock. Nel 2004, DDRRAMs (Double Data RAM Rate), che sono chiamati double data rate, perché il trasferimento dei dati sia sul fronte di salita e discesa di un orologio fornito esternamente, sono stati la forma più usata di SDRAM. Come vedremo nel capitolo 5, i trasferimenti ad alta velocità possono essere utilizzate per aumentare la larghezza di banda disponibile di memoria principale per soddisfare le esigenze del processore e cache.

Correzione degli errori

A causa del rischio di danneggiamento dei dati nelle memorie di grandi dimensioni, maggior parte dei computer utilizzare una sorta di controllo degli errori di codice per rilevare il possibile danneggiamento dei dati. Un semplice codice che è molto utilizzato è un *parità di codice*.Iocodice di parità na il numero di 1 in una parola viene contato, la parola ha parità dispari se il numero di 1 è dispari e

evit altrimenti. Quando una parola è scritta in memoria, il bit di parità è anche scritto (1 per dispari, 0 per pari). Poi, quando la parola viene letto, il bit di parità è letto e controllato. Se la parità della parola di memoria e il bit di parità memorizzati non corrispondono, è verificato un errore.

La1-bit schema di parità può rilevare al massimo 1 bit di errore in un elemento di dati, se ci sono

2 bit di errore, poi un 1 bit schema di parità non in grado di rilevare eventuali errori, in quanto la par-lità corrisponderà i dati con due errori. (In realtà, un 1-bit schema di parità può rilevare qualsiasi numero dispari di errori, tuttavia, la probabilità di avere tre errori è molto inferiore alla probabilità di avere due, quindi, in pratica, un 1-bit codice di parità è limitata per rilevare un singolo bit di errore.) Naturalmente, un codice di parità non può dire che bit in un elemento di dati è in errore.

codice di rilevamento di errore
Lacode che permette la rilevazione di un errore di dati, ma non la posizione precisa e, quindi, la correzione dell'errore.

A 1-bit di parità è un regime **errore codice di rilevamento**, Ci sono anche *codici di correzione di errore* (ECC) che rileva e permette la correzione di un errore. Per grandi memorie principali, molti sistemi usano un codice che permette il rilevamento di fino a 2 bit di errore e la correzione di un singolo bit di errore. Questi codici funzionano utilizzando più bit per codificare i dati, ad esempio, i codici tipici utilizzati per memorie principali richiede 7 o 8 bit per ogni 128 bit di dati.

Elaborazione: A 1-bit di codice di parità è un *distanza 2 Codice*, Il che significa che, se guardiamo i dati più il bit di parità, 1 bit non il cambiamento è sufficiente a generare un altro legale combinazione dei dati di più di parità. Ad esempio, se si cambia un po nei dati, la parità sarà sbagliato, e viceversa. Naturalmente, se cambiamo 2 bit (qualsiasi 2 bit di dati o 1 bit di dati e bit di parità), la parità corrisponderanno ai dati e l'errore non può essere rilevato. Quindi, vi è una distanza tra due combinazioni legali di parità e dati.

To rilevare più di un errore o di correggere un errore, abbiamo bisogno di un *distanza-3 Codice*, Che ha la proprietà che qualsiasi combinazione valida di bit nel codice di correzione di errore e dei dati di almeno 3 bit differenti da qualsiasi altra combinazione. Supponiamo di avere un codice e abbiamo un errore nei dati. In tal caso, il codice più i dati sarà un po 'lontano da una combinazione giuridica, e siamo in grado di correggere i dati a quella legale combina-zione. Se abbiamo due errori, possiamo riconoscere che c'è un errore, ma non siamo in grado di correggere gli errori. Vediamo un esempio. Ecco le parole di dati e un codice di correzione della distanza-3 per errore a 4 bit di dati.

Dati Word	Codice bit	Dati	Codice bit
0000	000	1000	111
0001	011	1001	100
0010	101	1010	010
0011	110	1011	001
0100	110	1100	001
0101	101	1101	010
0110	011	1110	100
0111	000	1111	111

To vedere come funziona, cerchiamo di scegliere una parola di dati, ad esempio, 0110, la cui correzione è il codice di errore 011. Ecco i quattro a 1-bit possibilità di errore per questi dati: 1110, 0010, 0100, e 0111. Ora guarda l'elemento di dati con lo stesso codice (011), che è la voce con il valore di 0001. Se il decoder correzione di errore ricevuto una delle quattro parole di dati possibili con un errore, avrebbe dovuto scegliere tra la correzione a 0110 o 0001. Mentre queste quattro parole con errore hanno un solo bit modificato dal modello corretto
0110, hanno ciascuna due bit che sono diverse dalla correzione alternate di 0001. Quindi, il meccanismo di correzione di errore può scegliere facilmente correggere al 0110, poiché un singolo errore è una maggiore probabilità. Vedere che due errori possono essere rilevati, semplicemente notare che tutte le combinazioni con due bit modificate hanno un codice diverso. Il riutilizzo di uno stesso codice è con tre bit differente, ma se correggere un errore di 2-bit, verrà correggere il valore errato, poiché il decoder assumerà che solo un singolo errore si è verificato. Se vogliamo correggere 1 bit e rilevare gli errori, ma non erroneamente corretti, 2-bit errori, abbiamo bisogno di una distanza-4 del codice.

Anche se abbiamo distinto tra il codice e dati in nostro spiegazione, in realtà, un codice di correzione di errore tratta la combinazione di codice e di dati come una parola in un codice più grande (7 bit in questo esempio). Pertanto, si tratta di errori nei bit di codice nella stessa maniera errori nei bit di dati.

Mentre l'esempio precedente richiede n-1 bit per $n$bit di dati, il numero di bit necessari cresce lentamente, in modo che per una distanza-3 codice, una parola di 64 bit richiede 7 bit e
128-bit parola ha bisogno 8. Questo tipo di codice è chiamato *Codice di Hamming*, Dopo R. Hamming, che ha descritto un metodo per la creazione di tali codici.

C.10 Finite macchine a stati

macchina a stati finiti
Una funzione logica sequenziale costituito da un insieme di ingressi e uscite, una funzione dello stato successivo che mappa lo stato corrente e gli ingressi di un nuovo stato, e una funzione di uscita che associa lo stato corrente ed eventualmente gli ingressi di un insieme di uscite affermato.

stato successivo funzione
Lacombinational funzione che, dati gli ingressi e lo stato attuale, determina lo stato successivo di una macchina a stati finiti.

Las abbiamo visto in precedenza, i sistemi logici digitali possono essere classificati come combinatoria o sequenziale. Sistemi sequenziali contengono stato memorizzato in elementi di memoria interna al sistema. Il loro comportamento dipende sia l'insieme di ingressi forniti e sui contenuti della memoria interna, o stato del sistema. Così, un sistema sequenziale non può essere descritta con una tabella di verità. Invece, un sistema sequenziale viene descritto come **macchina a stati finiti** (O spesso solo *macchina a stati*). La macchina a stati finiti è un insieme di stati e due funzioni, chiamato **stato successivo funzione** e la *Funzione di uscita*. L'insieme degli stati corrisponde a tutti i possibili valori della memoria interna. Pertanto, se vi sono $n$bit di memoria, vi sono due^{n}stati. La funzione dello stato successivo è una funzione combinatoria che, dati gli ingressi e lo stato attuale, determina lo stato successivo del sistema. La funzione di uscita produce una serie di uscite dallo stato corrente e gli ingressi. C.10.1 figura mostra schematicamente questo.

Le macchine a stati di cui parliamo qui e nel capitolo 4 sono *sincrono*. Questo significa che lo stato cambia con il ciclo di clock, e un nuovo stato viene calcolato una volta ogni orologio. Così, gli elementi di stato vengono aggiornati solo sul fronte di clock. Usiamo questo metodo in questa sezione e in tutto il capitolo 4, e non lo facciamo

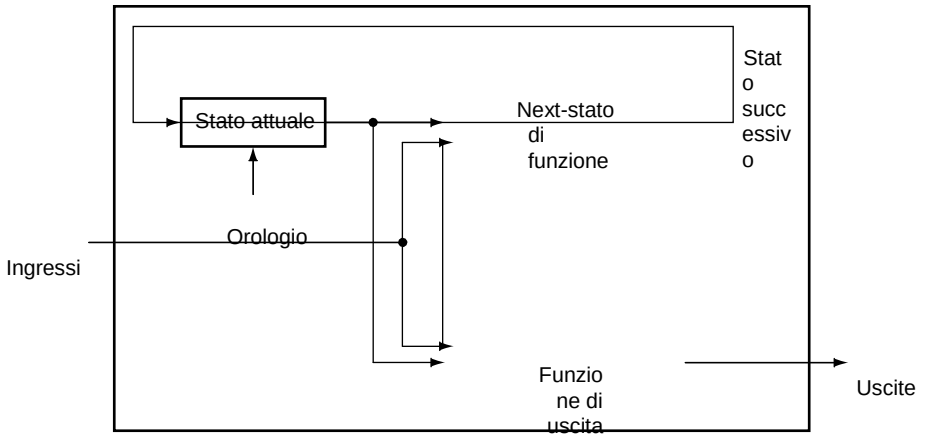


Figura C.10.1 Una macchina a stati è costituito da memoria interna che contiene lo stato e due funzioni combinatorie: la nuova funzione di stato e la funzione di uscita. Spesso, la funzione di uscita è limitata a prendere solo lo stato corrente come input; questo non cambia la capacità di una macchina sequenziale, ma influisce suoi interni.

di solito mostrano l'orologio in modo esplicito. Utilizziamo macchine a stati in tutto il Capitolo 4 per controllare l'esecuzione del processore e le azioni del datapath.

To illustrare come una macchina a stati finiti opera ed è progettato, diamo un'occhiata a un esempio semplice e classico: il controllo di un semaforo. (Capitoli 4 e 5 contengono esempi più dettagliati di utilizzo di macchine a stati finiti per il controllo del processore l'esecuzione.) Quando un macchina a stati finiti è utilizzato come un controller, la funzione di uscita è spesso limitata a dipendere soltanto lo stato corrente. Tale macchina a stati finiti è chiamato *Momentoore macchina*.Questo è il tipo di macchina a stati finiti che viene usato in questo libro. Se la funzione di uscita può dipendere sia lo stato corrente e la corrente di ingresso, la macchina viene chiamato *MEaly macchina*. Queste due macchine sono equivalenti nelle loro capacità, e può essere trasformato in altri meccanicamente. Il vantaggio fondamentale di una macchina di Moore è che può essere più veloce, mentre una macchina di Mealy può essere più piccola, in quanto può essere necessario un minor numero di stati di una macchina di Moore. Nel capitolo 5, discutiamo le differenze in dettaglio e mostrano una versione di Verilog a stati finiti di controllo utilizzando una macchina di Mealy.

Il nostro esempio riguarda il controllo di un semaforo in un incrocio di un asse nord-sud e est-ovest percorso. Per semplicità, si prenderà in considerazione solo le luci verde e rossa, aggiungendo la luce gialla è lasciato per un esercizio. Vogliamo le luci per passare non più veloce di 30 secondi in ogni direzione, quindi dovremo usare un orologio 0,033 Hz in modo che i cicli macchina tra gli stati, senza alcun più veloce di una volta ogni 30 secondi. Ci sono due segnali di uscita:

- *NSlite:* Wgallina questo segnale viene asserito, la luce sulla strada nord-sud è di colore verde, quando questo segnale viene deasserito, la luce sulla strada nord-sud è di colore rosso.
- *EWlite:* Wgallina questo segnale viene asserito, la luce sulla strada est-ovest è di colore verde; quando questo segnale viene deasserito, la luce sulla strada est-ovest è rosso.

Ion Inoltre, ci sono due ingressi:

- *NScar:* Iondicates che una macchina è il sensore posto nella massicciata di fronte alla luce sulla strada nord-sud (direzione nord o sud).
- *EWCAR:* Iondicates che una macchina è il sensore posto nella massicciata di fronte alla luce sulla strada est-ovest (in direzione est o ovest).

Il semaforo dovrebbe cambiare da una direzione all'altra solo se una macchina è in attesa di andare nella direzione opposta, altrimenti la luce dovrebbe continuare a mostrare verde nella stessa direzione come l'ultima vettura che ha attraversato l'incrocio.

To implementare questa luce semplice traffico abbiamo bisogno di due stati:

- *NSgreen:* Il semaforo è verde in direzione nord-sud.
- *EWgreen:* Il semaforo è verde in direzione est-ovest.

We anche bisogno di creare lo stato successivo funzione, che può essere specificato con una tabella:

	Ingres		
	NScar	EWcar	Stato
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

NOtice che non ha specificato nell'algoritmo di cosa succede quando si avvicina un auto da entrambe le direzioni. In questo caso, lo stato successivo funzione data sopra cambia lo stato di garantire che un flusso costante di auto da una direzione non può bloccare una macchina in direzione opposta.

La macchina a stati finiti è completata specificando la funzione di uscita.

Prima di esaminare come implementare questa macchina a stati finiti, diamo un'occhiata a una rappresentazione grafica, che viene spesso utilizzato per macchine a stati finiti. In questa rappresenta-zione, i nodi sono utilizzati per indicare gli stati. All'interno del nodo abbiamo posto un elenco delle uscite che sono attivi per quello stato. Archi diretti sono utilizzati per mostrare il prossimo-stato

	Uscite	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

funzione, con le etichette sugli archi specificando la condizione di ingresso come funzioni logiche. Figura C.10.2 mostra la rappresentazione grafica di questa macchina a stati finiti.

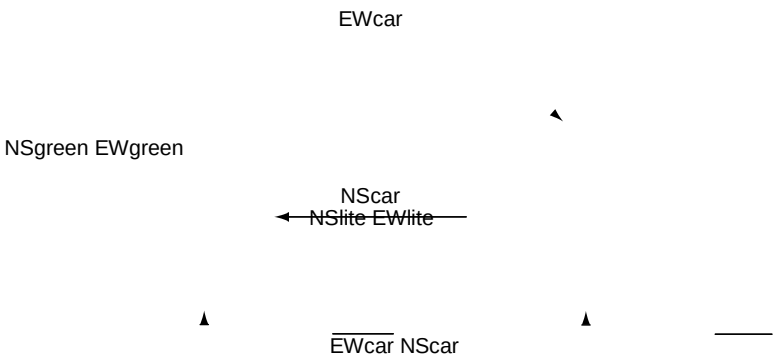


Figura C.10.2 La rappresentazione grafica dei due stati controllore del traffico leggero. Noi semplificata le funzioni logiche sulle transizioni di stato. Per esempio, la transizione da NSgreen a EWgreen
nel prossimo tavolo è stato (NScar · EWcar) +(NScar EWcar ·), che è equivalente a EWcar.

Una macchina a stati può essere implementato con un registro per mantenere lo stato corrente e un blocco di logica combinatoria che calcola lo stato successivo e la funzione di uscita. Figura C.10.3 mostra come una macchina a stati finiti con 4 bit di stato, e quindi fino a 16 stati, potrebbe apparire. Per implementare la macchina a stati finiti in questo modo, dobbiamo prima di assegnare i numeri di stato negli Stati Uniti. Questo processo è chiamato *stato di assegnazione*.For esempio, si potrebbe assegnare NSgreen allo stato 0 e EWgreen allo stato 1. Il registro di stato conterrebbe un singolo bit. La prossima funzione sarebbe stato dato come

$$\text{NextState} = (\overline{\text{CurrentState}} \cdot \text{EWcar}) + (\text{CurrentState} \cdot \overline{\text{NScar}})$$

dove CurrentState è il contenuto del registro di stato (0 o 1) e NextState è l'uscita dello stato successivo funzione che verrà scritto nel registro di stato alla fine del ciclo di clock. La funzione di uscita è anche semplice:

$$\text{NSlite} = \overline{\text{CurrentState}}$$

$$\text{EWlite} = \text{CurrentState}$$

Il blocco di logica combinatoria è spesso implementato utilizzando la logica strutturata, come una PLA. Un PLA può essere costruito automaticamente dalle tabelle di funzione dello stato successivo e uscita. In realtà, ci sono computer-aided design (CAD) programmi

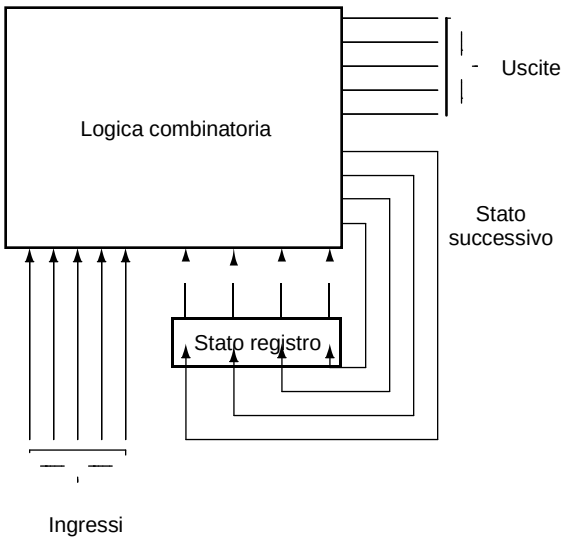


FIGURA C.10.3 Una macchina a stati finiti è implementato con un registro di stato che contiene lo stato corrente e un blocco di logica combinatoria per calcolare lo stato successivo e funzioni di uscita. Le ultime due funzioni sono spesso scisse e realizzato con due blocchi separati di logica, che possono richiedere un minor numero porte.

che prendere o una rappresentazione grafica o testuale di una macchina a stati finiti e produrre un'implementazione ottimizzata automaticamente. Nei capitoli 4 e 5, macchine a stati finiti sono stati usati per controllare l'esecuzione del processore. Appendice C discute l'applicazione dettagliata di questi controller con entrambe le PLA e ROM.

To mostrare come si potrebbe scrivere il controllo in Verilog, C.10.4 figura mostra una versione Verilog progettato per la sintesi. Si noti che per questa funzione di controllo semplice, una macchina Mealy non è utile, ma questo stile di specifica viene utilizzato nel capitolo 5 per implementare una funzione di controllo che è una macchina di Mealy e ha meno stati di controllore della macchina Moore.

```
Modulo TrafficLite (EWCar, NSCar, EWLite, NSLite, orologio);
    ingresso EWCar, NSCar, orologio;
uscita EWLite, NSLite;

reg Stato;

stato iniziale = 0, lo stato / / set iniziale

/ / Dopo due assegnazioni impostare l'uscita, che si basa solo sulla variabile di stato
assegnare NSLite = ~ Stato, / / NSLite su se stato = 0;
assegnare EWLite  = Stato; / / EWLite su se lo stato = 1

sempre @ (orologio posedge) / / tutti gli aggiornamenti di stato su un fronte di clock positivo
    caso (stato)
        0: Stato = EWCar; / / cambiamento di stato solo se EWCar

        1: Stato = NSCar; / / cambiamento di stato solo se NSCar

    endcase
endmodule
```

Figura C.10.4 Una versione Verilog del controllore semaforo.

Controll are Ynoi stessi

Wcappello è il più piccolo numero di stati in una macchina di Moore per il quale una macchina di Mealy potrebbe avere minor numero di stati?

un. Due, dato che ci potrebbe essere una macchina a stati Mealy che potrebbe fare la stessa cosa.

b. Tre, in quanto ci potrebbe essere una semplice macchina di Moore che è andato ad uno dei due stati diversi e sempre riportato allo stato originale dopo. Per una macchina semplice, a due stati della macchina di Mealy è possibile.

c. Avete bisogno di almeno quattro stati di sfruttare i vantaggi di una macchina di Mealy oltre una macchina di Moore.

C.11 TMetodologie IMing

In tutta questa appendice e nel resto del testo, usiamo un edge-triggered metodologia di temporizzazione. Questa metodologia di temporizzazione ha un vantaggio nel senso che è sim-accoppiatore di spiegare e comprendere che un livello-triggered metodologia. In questo sec-zione, si spiega questa metodologia di temporizzazione in un po 'più dettagliato e anche introdurre sensitivo del livello clock. Concludiamo questa sezione con una breve discussione il problema

di segnali asincroni e sincronizzatori, un problema importante per i progettisti digitali.

Lo scopo di questa sezione è quello di introdurre i concetti principali in clocking metodologia. La sezione fa alcune ipotesi semplificative importanti, se siete interessati a capire la metodologia di temporizzazione in modo più dettagliato, consultare uno dei riferimenti elencati alla fine di questa appendice.

We utilizzare un edge-triggered metodologia tempi perché è più semplice da spiegare e ha poche regole necessarie per la correttezza. In particolare, se si assume che tutti gli orologi arrivano allo stesso tempo, abbiamo la garanzia che un sistema con edge-triggered registri tra i blocchi di logica combinatoria in grado di funzionare correttamente senza gare, se ci limitiamo a rendere l'orologio abbastanza a lungo. La *rasso* si verifica quando il contenuto di un elemento di stato dipende dalla velocità relativa di elementi logici diversi. In un edge-triggered disegno, il ciclo di clock deve essere sufficientemente lunga per il percorso da un flip-flop attraverso la logica combinatoria a un altro flip-flop in cui deve soddisfare il requisito di setup-tempo. Figura C.11.1 mostra questa necessità di un sistema che utilizza il fronte di salita-triggered flip-flop. In tale sistema il periodo di clock (o tempo di ciclo) deve essere almeno grande quanto

$$t_{\text{professionistap}} + t_{\text{combinational}} + t_{\text{installazione}}$$

per il caso peggiore valori di questi tre ritardi, che sono definite come segue:

- $t_{\text{professionistap}}$ è il tempo di un segnale per propagarsi attraverso un flip-flop, ma è anche talvolta chiamato clock-to-*Q*.
- $t_{\text{combinational}}$ è il ritardo più lungo per qualsiasi logica combinatoria (che per definizione è circondato da due flip-flop).
- tsetup è il tempo prima che il fronte del clock di salita che l'ingresso di un flip-flop deve essere valido.

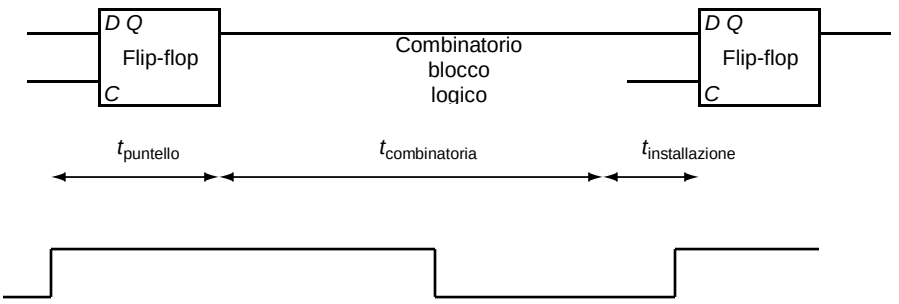


Figura C.11.1 In un edge-triggered di design, l'orologio deve essere sufficientemente lungo per consentire segnali sia valida per il tempo impostato richiesta prima della fronte di clock successivo. Il tempo per un flip-flop di ingresso per propagare al flip-flip uscite è t_{puntello}, Il segnale assume allora $t_{\text{combinational}}$ to viaggiare attraverso la logica combinatoria e deve essere valido $t_{\text{installazione}}$ prima del fronte di clock successivo.

We fare una ipotesi semplificatrice: i hold-time requisiti sono soddisfatti, che non è quasi mai un problema con la logica moderna.

clock skew La differenza di tempo assoluto tra i tempi in cui due elementi di stato vedere un fronte di clock.

Una complicazione aggiuntiva che deve essere considerata in edge-triggered disegni è **clock skew**. Clock skew è la differenza di tempo assoluto tra quando due elementi statali vedere un fronte di clock. Clock skew deriva dal fatto che il segnale di clock si usano spesso due percorsi diversi, con ritardi leggermente diverse, per raggiungere due elementi diversi di stato. Se il clock skew è abbastanza grande, può essere possibile per un elemento modifica di stato e causare l'ingresso ad un altro flip-flop di cambiare prima del fronte di clock è visto dal secondo flip-flop.

Figura C.11.2 illustra questo problema, ignorando i tempi di configurazione e flip-flop propagazione ritardo. Per evitare il funzionamento errato, il periodo di clock è aumentato per consentire il massimo ciclo skew. Pertanto, il periodo di clock deve essere più lungo

$$t_{\text{professionistap}} + t_{\text{combinational}} +$$

$$t_{\text{installazione}} + t_{\text{storto}}$$

Wesimo questo vincolo sul periodo di clock, i due orologi possono arrivare anche in ordine inverso, con il secondo clock in arrivo t_{storto} precedenza, e il circuito funzionerà

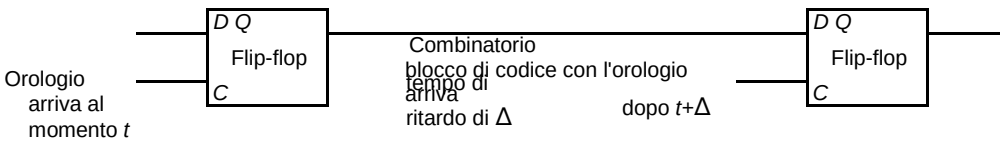


Figura C.11.2 Illustrazione di come skew può causare una gara, che porta ad un uso errato. Perché della differenza di quando i due flip-flop vedere l'orologio, il segnale che viene memorizzato nel primo flip-flop può correre avanti e cambiare l'ingresso del secondo flip-flop prima che l'orologio arriva al secondo flip-flop.

disegni.

sensitivo del livello clocking Una temporizzazione metodologia in cui si verificano i cambiamenti di stato sia a livello di clock alto o basso, ma non sono istantanei, in quanto tali modifiche sono in edge-triggered

corrrettamente. Progettisti a ridurre clock-skew problemi con cautela instradamento del segnale di clock per minimizzare la differenza nei tempi di arrivo. Inoltre, i progettisti intelligenti anche fornire un certo margine di fare il clock leggermente più lungo del minimo; questo consente variazione componenti così come l'alimentatore. Poiché clock skew può colpire anche i requisiti in tempo attesa, riducendo al minimo le dimensioni del clock skew è importante.

Edge-triggered disegni hanno due inconvenienti: richiedono logica supplementare e che a volte può essere più lenta. Solo guardando il flip-flop rispetto al livello sensibile fermo che abbiamo usato per costruire i flip-flop mostra che edge-triggered progettazione richiede più logica. Un'alternativa è quella di utilizzare **sensitivo del livello clocking**. Perché cambiamenti di stato di un livello sensibile metodologia non sono istantanei, un livello sensibile schema è leggermente più complesso e richiede cure aggiuntive per farlo funzionare correttamente.

Level-Sensitive Timing

Ion sensitivo del livello di temporizzazione, i cambiamenti di stato si verificano a livelli sia alti o bassi, ma non sono istantanei come sono in un edge-triggered metodologia. A causa del cambiamento di stato noninstantaneous, gare può facilmente verificarsi. Al fine di garantire un livello-sensitive design funzioni correttamente anche se l'orologio è abbastanza lento, progettisti utilizzano *two-fase clocking.*Two-clock di fase è un sistema che fa uso di due
segnali di clock non sovrapposti. Poiché i due orologi, tipicamente chiamate $\varphi_{1 \text{ e}}$ $\varphi_{2,}$
sono sovrapposte, al massimo uno dei segnali di clock è alta in un dato momento, come
Figura C.11.3 spettacoli. Siamo in grado di utilizzare questi due orologi per costruire un sistema che contiene il livello sensibile fermi, ma è esente da condizioni di gara, proprio come gli edge-triggered disegni erano.

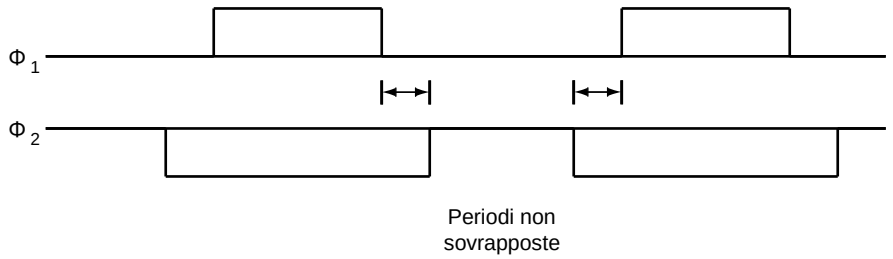


FIGURA C.11.3 in due fasi clocking schema che mostra il ciclo di ogni orologio e periodi non sovrapposti.

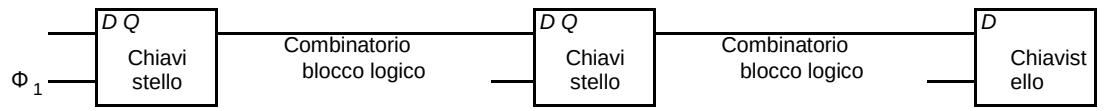


Figura C.11.4 in due fasi schema di temporizzazione con alternanza di fermi che mostra come il sistema funziona su entrambe le fasi di clock. L'uscita di un latch è stabile in fase opposta dal suo ingresso C. Così, il primo blocco di ingressi combinatori ha un ingresso stabile
durante $\varphi_{2,}$ e la sua uscita è bloccata da $\varphi_{2.}$Il secondo (più a destra) blocco combinatorio opera in un solo modo opposto, con ingressi stabili durante $\varphi_{1.}$Così, i ritardi attraverso i blocchi combinatori determinare il tempo minimo che i rispettivi clock deve essere rivendicato. Il dimensione del periodo non sovrapposte è determinata dal clock skew massimo e minimo ritardo di ogni blocco logico.

Un modo semplice per progettare un sistema simile è alternare l'utilizzo delle serrature che sono aperti $\varphi_{1}$with latch sono aperti $\varphi_{2.}$ Perché entrambi gli orologi non sono asseriti al tempo stesso, una corsa non può verificarsi. Se l'ingresso a un blocco combinatorio è un φ_{1}orologio, quindi la sua uscita è bloccata da un φ_{2}orologio, che è aperto solo durante $\varphi_{2}$when
il latch ingresso è chiuso e quindi ha una uscita valida. La figura mostra come C.11.4 un sistema con due fasi di temporizzazione e alternata fermi opera. Come in un

edge-triggered progettazione, dobbiamo prestare attenzione al clock skew, in particolare tra i due

orologio fasi. Da aumentando la quantità di nonoverlap tra le due fasi, si può ridurre il margine di errore potenziale. Così, il sistema è garantito per funzionare correttamente se ogni fase è sufficiente e se c'è abbastanza grandi nonoverlap tra le fasi.

Ingressi asincroni e Sincronizzatori

By usando un orologio unico o in due fase di clock, siamo in grado di eliminare le condizioni di gara se l'orologio-skew problemi sono evitati. Purtroppo, non è pratico per il funzionamento di un intero sistema con un solo orologio e di mantenere ancora la piccola inclinazione dell'orologio. Mentre la CPU può utilizzare un solo orologio, dispositivi di I / O probabilmente hanno il loro orologio. Capitolo 6 descritto come un dispositivo asincrono può comunicare con la CPU attraverso una serie di passi di handshaking. Per tradurre l'ingresso asincrono di un segnale sincrono che può essere utilizzato per modificare lo stato di un sistema, è necessario utilizzare un *sincronizzatore,*i cui ingressi sono il segnale asincrono e un orologio e la cui uscita è un segnale sincrono con il clock di ingresso.

Il nostro primo tentativo di costruire un sincronizzatore utilizza un edge-triggered flip-flop D, la cui *D*ingresso è il segnale asincrono, come figura C.11.5 spettacoli. Perché comunicare con un protocollo di sincronizzazione (come abbiamo visto nel capitolo 6), non importa se si rileva lo stato affermato del segnale asincrono su un orologio o il prossimo, dato che il segnale si terrà affermato fino a quando non viene riconosciuto. Così, si potrebbe pensare che questa semplice struttura è sufficiente per campionare il segnale con precisione, che sarebbe la causa, salvo per un piccolo problema.

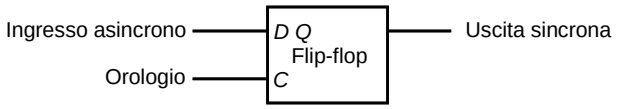


FIGURA C.11.5 Un sincronizzatore costruita da un D flip-flop è usato per campionare un segnale asincrono per produrre un output che è sincrono con il clock. Questo "sincronizzatore" sarà *non* wOrki correttamente!

metastabilità Una situazione che si verifica se il segnale è campionato quando non è stabile per la configurazione desiderata e tempi di mantenimento, causando il valore campionato a cadere in la regione indeterminata tra un valore alto e basso.

Il problema è una situazione chiamata **metastabilità**.Suppose il segnale asincrono è la transizione tra alto e basso quando il fronte di clock arriva. Chiaramente, non è possibile sapere se il segnale viene agganciato ad alto o basso. Tale pro-blema abbiamo potuto vivere. Purtroppo, la situazione è peggiore: quando il segnale che viene campionato non è stabile per la configurazione desiderata e tempo di attesa, il flip-flop può entrare in una *metastabile* stato. In tale stato, l'uscita non avrà un valore alto o basso legittima, ma sarà nella regione indeterminata tra loro. Inoltre, l'

flip-flop non è garantito per uscire da questo stato in una quantità limitata di tempo. Alcuni blocchi logici che guardano l'uscita del flip-flop possa vedere la sua uscita come 0, mentre altri lo vedono come 1. Questa situazione è definita **sincronizzatore fallimento**.

sincronizzatore fallimento
Lasituazione in cui un flip-flop entra in uno stato metastabile e dove alcuni blocchi logici leggendo l'output di il flip-flop vedere a 0 mentre per altri un 1.

Iona puramente sistema sincrono, il fallimento sincronizzatore può essere evitata garantendo che il setup e hold per un flip-flop o latch è sempre soddisfatto, ma questo è impossibile quando l'ingresso è asincrona. Invece, l'unica soluzione possibile è quella di attendere il tempo necessario prima di guardare l'uscita del flip-flop per assicurare che la sua uscita è stabile, e che è uscito stato metastabile, se mai entrato. Per quanto tempo è abbastanza lungo? Ebbene, la probabilità che il flip-flop rimarrà nello stato metastabile diminuisce esponenzialmente, quindi dopo un tempo molto breve la probabilità che il flip-flop è nello stato metastabile è molto bassa, ma non la probabilità raggiunge 0! Per questo i progettisti attendere abbastanza a lungo che la probabilità di un sincro-nizer fallimento è molto basso, e il tempo fra i guasti, le quali diverranno anni o addirittura migliaia di anni.

For più flip-flop disegni, in attesa di un periodo che è diverse volte più lungo del tempo di setup rende la probabilità di errore di sincronizzazione molto basso. Se la frequenza di clock è più lungo del periodo di metastabilità potenziale (che è probabile), quindi un sincronizzatore di sicurezza può essere costruita con due D flip-flop, come mostra la Figura C.11.6. Se siete interessati a saperne di più su questi problemi, esaminare i riferimenti.

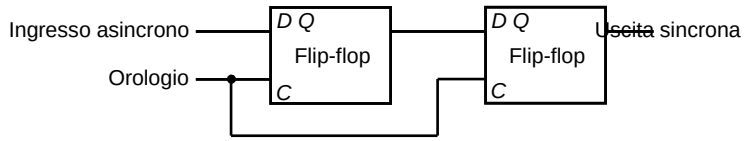


Figura C.11.6 Questo sincronizzatore funziona correttamente se il periodo di metastabilità che vogliamo evitare è inferiore al periodo di clock. Sebbene l'uscita del primo flip-flop può essere metastabile, non sarà visto da qualsiasi altro elemento logico che l'orologio secondo, quando il secondo flip-flop D campiona il segnale, che da quel momento non dovrebbe più essere in un metastabile stato.

Controllare Ynoi stessi

Suppose abbiamo un progetto con un orologio molto grande skew un tempo superiore al registro **propagazione time**. È sempre possibile per una tale progettazione per rallentare l'orologio giù sufficiente a garantire che la logica funziona correttamente?

tempo di propagazione
Il tempo richiesto per un ingresso a un flip-flop per propagare le uscite del flip-flop.

un. Sì, se l'orologio è sufficientemente lenta i segnali possono sempre propagare e la progettazione funzionerà, anche se l'inclinazione è molto grande.

b. No, poiché è possibile che due registri vedere il bordo stesso clock distanza sufficiente che un registro viene attivato, e le sue uscite propagato e visto da un secondo registro con il bordo stesso clock.

C.12 FDispositivi POSTO programmabili

dispositivi di campo programmabili (FPD) Un circuito integrato contenente logica combinatoria, ed eventualmente dispositivi di memoria, che sono configurabili all'utente finale.

proglogica del dispositivo rammable (PLD) Un integcircuito di valutazione contenente logica combinatoria cui funzione è configurata dall'utente finale.

campo gate array programmabile (FPGA) Laconfigurable circuito integrato contenente due blocchi logici combinatori e infradito.

semplice dispositivo logico programmabile (SPLD) Prlogica del dispositivo ogrammable, di solito contenente un singolo PAL o PLA.

matrice logica programmabile (PAL) Contains un programmabile e-seguita da un piano fisso o piano.

antifuse Lastruttura in un circuito integrato che quando programmato effettua una connessione permanente tra due fili.

Wntro un chip custom o semicustom, i progettisti possono sfruttare la flessibilità della struttura sottostante di implementare facilmente logica combinatoria o sequenziale. Come può un designer che non vuole utilizzare un costume o semicustom IC attuare un pezzo complesso di logica sfruttare i livelli molto elevati di integrazione a disposizione? La componente più popolare usato per la progettazione logica sequenziale e combinatoria al di fuori di un circuito integrato personalizzato o semicustom è un **campo dispositivo programmabile (FPD)**.Lan FPD è un circuito integrato contenente logica combinatoria, ed eventualmente dispositivi di memoria, che sono configurabili dall'utente finale.

FPD rientrano generalmente in due campi: **dispositivi logici programmabili** (PLD), che sono puramente combinatorio, e **Field Programmable Gate Array** (FPGA), che forniscono sia la logica combinatoria e flip-flop. PLD consistono in due forme: **PLD semplici** (SPLD), che di solito sono un PLA o **programmabile matrice logica (PAL)**,e PLD complessi, che consentono più di un blocco logico, nonché le interconnessioni tra blocchi configurabili. Quando si parla di una PLA in un PLD, si intende un PLA con l'utente programmabile e-plane e o-piano. Un PAL è come una PLA, tranne che il o-piano è fisso.

Prima di discutere FPGA, è utile parlare di come FPD sono configurati. La configurazione è essenzialmente una questione di dove fare o rompere i collegamenti. Strutture di gate e registro sono statici, ma le connessioni possono essere configurati. Si noti che con la configurazione dei collegamenti, un utente determina la logica funzioni sono implementate. Si consideri un configurabile PLA: individuando dove le connessioni sono in e-piano e il piano o, l'utente determina quali funzioni logiche sono calcolati in PLA. Connessioni in FPD sono permanenti o riconfigurabile. Connessioni permanenti comportare la creazione o distruzione di una connessione tra due fili. FPLDs attuali utilizzano tutti un **antifuse** technology, che permettono un collegamento da costruire in fase di programmazione che è quindi permanente. L'altro modo per configurare FPLDs CMOS è attraverso una SRAM. La SRAM è scaricato all'accensione, e il contenuto controllare l'impostazione degli interruttori, che a sua volta determina quali linee metalliche sono collegate. L'uso di SRAM controllo ha il vantaggio che la FPD può essere riconfigurato modificando il contenuto della SRAM. Gli svantaggi della SRAM-based controllo sono due: la configurazione è volatile e deve essere ricaricati accensione, e l'uso di transistori attivi per interruttori aumenta leggermente la resistenza di tali connessioni.

FPGA includono sia logica e dispositivi di memoria, strutturati in una matrice bidimensionale con i corridoi dividono le righe e colonne utilizzate per

gObbligazione Globale interconnessione tra le celle della matrice. Ogni cella è una combinazione di porte e flip-flop che possono essere programmati per eseguire una funzione specifica. Perché sono essenzialmente piccole, RAM programmabile, sono anche chiamati **tabelle di ricerca** (LUT). Post più recente FPGA contengono elementi più sofisticati come pezzi di vipere e blocchi di RAM che possono essere usati per costruire i file di registro. A FPGA pochi grandi contengono anche 32-bit core RISC!

Ioltre alla programmazione ogni cella per eseguire una funzione specifica, le interconnessioni tra le cellule sono anche programmabili, permettendo FPGA moderne con centinaia di blocchi e centinaia di migliaia di porte da utilizzare per le funzioni logiche complesse. Interconnessione è una sfida importante in chip custom, e questo è ancora più vero per FPGA, perché le cellule non rappresentano unità naturali di decomposizione-zione per la progettazione strutturata. In molti FPGA, il 90% della superficie è riservata per l'inter-connessione e solo il 10% è per la logica e blocchi di memoria.

Just come non si può progettare un chip custom o semicustom senza strumenti CAD, è anche bisogno di loro per FPD. Strumenti di sintesi logica sono stati sviluppati che tar-get FPGA, consentendo la generazione di un sistema che utilizza FPGA da Verilog strutturali e comportamentali.

Un testo generale sulla progettazione logica.

C.13 Osservazioni conclusive

Questo appendice introduce i concetti fondamentali della progettazione logica. Se si è digerito il materiale in questa appendice, si è pronti per affrontare il materiale nei capitoli 4 e 5, due settori che utilizzano i concetti discussi in questa appendice ampiamente.

Letture consigliate

Ci sono una serie di testi buoni progettazione logica. Qui ci sono un po 'di come si potrebbe approfondire.

Ciletti, M. D. [2002]. *Annuncioavanzato Digital Design won l'Verilog HDL,* Englewood Cliffs, NJ: Prinvogliare Hall.

Un libro completo sulla progettazione logica utilizzando Verilog.

KATZ, R. H. [2004]. *MLogic Design odern,*2nd ed, Reading, MA:. Addison-Wesley.

Un testo generale sulla progettazione logica.

Wakerly, J. F. [2000]. *Digital Design: Principles e pratiche,* 3rd ed, Englewood Cliffs, NJ:. Prentice Hall.

tabelle di ricerca (LUT) In un dispositivo di campo programmabile, il nome dato to le cellule perché sono costituiti da una piccola quantità di logica e RAM.

C.14 Esercizi

C.1 [10] <§ C.2> Oltre alle leggi fondamentali che abbiamo discusso in questa sezione, ci sono due importanti teoremi, chiamati DeMorgan teoremi:

$$\overline{La+B}=A\cdot\overline{B} \text{ e } A\cdot B=La+B$$

Prove di DeMorgan teoremi con una tabella di verità della forma

L	B	L	B	A + B	A · B	A · B	A + B
0	0	1	1	1	1	1	1
0	1	1	0	0	0	1	1
1	0	0	1	0	0	1	1
1	1	0	0	0	0	0	0

C.2 [15] <§ C.2> Dimostrare che le due equazioni per E nell'esempio che iniziano a pagina C-7 sono equivalenti utilizzando DeMorgan teoremi e gli assiomi indicati a pagina C-7.

C.3 [10] <§ C.2> Mostra che ci sono 2^nvoci in una tabella di verità per una funzione con
*n*ingress
i.

C.4 [10] <§ C.2> Una funzione logica che viene utilizzato per una varietà di scopi (compresa l'ai sommatori e per calcolare parità) è *esclusivo O*. L'output di una funzione a due ingressi OR esclusivo è vero solo se esattamente uno degli ingressi è vero. Mostra la tabella di verità per un due ingressi esclusiva funzione OR e implementare questa funzione con porte AND, OR porte, e inverter.

C.5 [15] <§ C.2> Dimostrare che la porta NOR è universale, mostrando come costruire AND, OR, NOT e le funzioni con un due ingressi porta NOR.

C.6 [15] <§ C.2> Dimostrare che la porta NAND è universale, mostrando come costruire gli operatori AND, OR e NOT funzioni utilizzando un NAND a due ingressi cancello.

C.7 [10] <§ § C.2, C.3> Costruire la tabella di verità per quattro ingressi odd-parity fun-zione (vedere a pagina C-65 per una descrizione di parità).

C.8 [10] <§ § C.2, C.3> Implementare il quattro ingressi dispari parità funzione con E
e porte OR con ingressi e uscite bolliti.

C.9[10] <§ § C.2, C.3> Implementare il quattro ingressi dispari parità funzione con un PLA.

C.10 [15] <§ § C.2, C.3> Dimostrare che un multiplexer a due ingressi è anche universale, mostrando come costruire la NAND (o NOR) porta con un multiplexer.

C.11 [5] <§ § 4.2, C.2, C.3> Si supponga che X è composto da 3 bit, x2 x1 x0. Scrivi quattro funzioni logiche che sono vere se e solo se

- X contiene solo 0
- X contiene un numero pari di 0s
- X quando interpretata come un numero binario senza segno è meno di 4
- X quando interpretato come un segno (complemento a due) il numero è negativo

C.12 [5] <§ § 4.2, C.2, C.3> attuare le quattro funzioni descritto nell'Esercizio Utilizzando un C.11 PLA.

C.13 [5] <§ § 4.2, C.2, C.3> Si supponga che X è composto da 3 bit, x2 x1 x0, e Y consists di 3 bit, y2 y1 y0. Scrivi funzioni logiche che sono vere se e solo se

- X<Y,dove X e Y sono pensati come numeri binari senza segno
- X<Y,dove X e Y sono pensati come firmati (complemento a due) numeri
- X= Y

Use un approccio gerarchico che può essere estesa ad un numero maggiore di bit. Mostra come si può estendere a 6-bit di confronto.

C.14 [5] <§ § C.2, C.3> Implementazione di una rete a commutazione che ha due ingressi dati (*La* e *B*Dati), due uscite (*C*e *D*), Ed un ingresso di controllo (*S*). Se *S*è uguale a 1, la rete è in modalità Passthrough, e *C*deve essere uguale *La*,e *D*deve essere uguale *B*.Se *S*uguale a 0, la rete è in modalità di attraversamento e *C*deve essere uguale *B*,e *D*dovrebbe pari *La*.

C.15 [15] <§ § C.2, C.3> Derive del prodotto-di-rappresentazione per somme *E*mostrato a pagina C-11 a partire dalla somma dei prodotti di rappresentazione. Avrete bisogno di usare DeMorgan teoremi.

C.16 [30] <§ § C.2, C.3> Dare un algoritmo per la costruzione della somma di prodotti di rappresentanza per un'equazione logica arbitraria composta da AND, OR e NOT. L'algoritmo dovrebbe essere ricorsiva e non deve costruire la tabella di verità nel processo.

C.17 [5] <§ § C.2, C.3> Mostra una tabella di verità per un multiplexer (ingressi *La*,*B*, E *S*;
produzione *C*), Utilizzando non si preoccupa di semplificare la
tabella, se possibile.

C.18 [5] <§ C.3> Qual è la funzione implementata dai seguenti moduli Verilog:

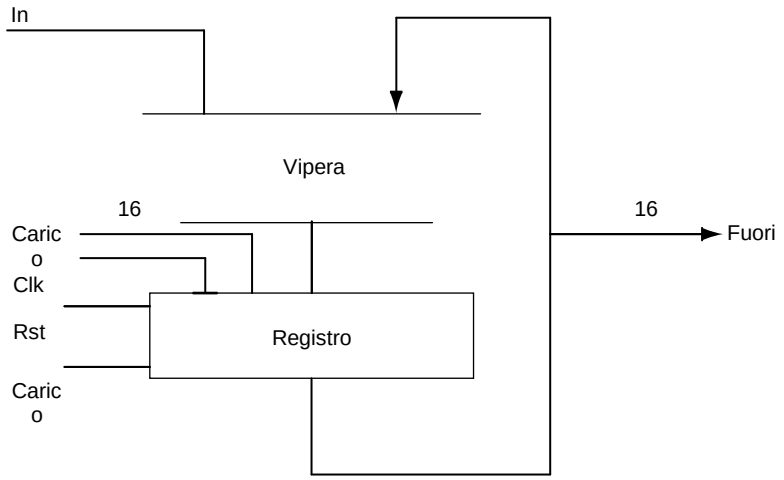
```
modulo FUNC1 (I0, I1, S, out);
ingresso I0,
I1, S
ingresso,
uscita fuori;
  fuori = S? I1: I0;
endmodule

modulo FUNC2 (out, ctl, clk, reset);
    uscita [7:0]
        fuori;
ingresso ctl, clk, reset;
   reg [7:0]
     fuori;
 sempre @ (posedge clk)
if (reset) inizia
     fuori <= 8'b0;
fine
else if (ctl)
     iniziano le <=
     Out + 1;
fine
 else begin
       fuori <= Out - 1;
fine
endmodule
```

C.19 [5] <§ C.4> Il codice Verilog a pagina C-53 è un D flip-flop. Visualizza VeriLOG code per un latch
D.

C.20[10] <§ § C.3, C.4> Scrivere un modulo Verilog attuazione di un 2-a-4 decoder (e / o encoder).

C.21 [10] <§ § C.3, C.4> Dato il seguente schema logico di un accumulatore, annotare l'implementazione del modulo Verilog di esso. Si supponga che un positive edge-triggered registro e Rst asincrona.



C.22 [20] <§ § C.3, C.4, C.5> Sezione 3.3 presenta il funzionamento di base e possibili implementazioni di moltiplicatori. Una unità di base di tali implementazioni è un'unità shift-and-add. Mostra una implementazione Verilog per questa unità. Mostra come si può utilizzare questa unità per costruire un moltiplicatore a 32 bit.

C.23 [20] <§ § C.3, C.4, C.5> Ripeti Esercizio C.22, ma per un divisore non firmato, piuttosto che un moltiplicatore.

C.24 [15] <§ C.5> La ALU supportato impostato con meno di (slt) utilizzando solo il bit del segno del sommatore. Proviamo un gruppo con meno di funzionamento con i valori -7ten e 6ten.

To rendere più semplice seguire l'esempio, cerchiamo di limitare le rappresentazioni binarie a
4 bit: 1001two e 0110two.

1001two - 0110two 1001two = + = 1010two 0011two

Thirisultato s suggerisce che -7>6, Che è chiaramente sbagliato. Quindi, si dovrà tener conto di overflow nella decisione. Modificare le 1-bit ALU in Figura C.5.10 a pagina C-33 per gestire correttamente slt. Apportare le modifiche una fotocopia di questa figura per risparmiare tempo.

C.25 [20] <§ C.6> Un semplice controllo di overflow durante Inoltre è vedere se il Carryin al bit più significativo non è la stessa della Carryout del bit più significativo. Dimostrano che questo controllo è la stessa in Figura 3.5 a pagina 232.

C.26 [5] <§ C.6> Riscrivere le equazioni a pagina C-44 per una ricerca in avanti di logica per un sommatore a 16 bit con una nuova notazione. Innanzitutto, utilizzare i nomi per i segnali Carryin dei singoli bit del sommatore. Cioè, usare c4, c8, c12,. . . invece di C1, C2, C3,. . . . Inoltre, sia $P_{Io,j}$significare una

propagazione del segnale per bit *Io*to*j*E $G_{Io,j}$significa generare un segnale per i bit *Io*to*j*.For esempio, l'equazione

$$C2 = G1 + (P1 \cdot G0) + (P1\ P0 \cdot \cdot$$

c0)

può essere riscritta come

$$c8 = G7, {}_4+ (P7, 4 \cdot G3, 0) + (P7, 4 \cdot P3, 0 \cdot c\ 0)$$

Questa notazione più generale è utile per creare più ampie sommatori.

C.27 [15] <§ C.6> Scrivere le equazioni per la ricerca in avanti di logica per un sommatore a 64 bit utilizzando la nuova notazione dell'Esercizio C.26 e l'utilizzo di 16 bit sommatori come build-zione blocchi. Includere un disegno simile alla figura C.6.3 nella soluzione.

C.28 [10] <§ C.6> Ora calcola la performance relativa di sommatori. Assumiamo che hardware corrispondente a qualsiasi equazione contenente solo OR o AND termini, quali le equazioni per p*Io*e g*Io*a pagina C-40, ha un tempo di unità Equazioni T. che consistono l'OR di diversi e termini, come ad esempio le equazioni di c1, c2, c3, e c4 a pagina C-40, avrebbe quindi due unità di tempo, 2T . Il motivo è ci vorrebbe per produrre le T e nei Termini e quindi un ulteriore T per produrre il risultato della OR. Calcolare i numeri e il rapporto prestazioni per 4-bit sommatori ripple sia per trasportare e portare lookahead. Se le condizioni di equazioni sono ulteriormente definiti da altre equazioni, quindi aggiungere i ritardi appropriate per tali equazioni intermedi, e continuare fino ricorsivamente i bit di ingresso reali del sommatore sono utilizzati in un'equazione. Includere un disegno di ogni sommatore etichettato con i ritardi calcolati e il percorso del caso peggiore ritardo evidenziato.

C.29 [15] <§ C.6> Questo esercizio è simile a Esercizio C.28, ma questa volta calcolare le velocità relative di un sommatore a 16 bit utilizzando ripple portare solo, ripple carry dei gruppi di 4 bit che portano lookahead uso, e il carry-lookahead schema a pagina C-39.

C.30 [15] <§ C.6> Questo esercizio è simile a Esercizi C.28 e C.29, ma questa volta calcolare le velocità relative di un sommatore a 64 bit con ondulazione portare solo, ripple carry di
Gruppi di 4 bit che utilizzano portare lookahead, ripple carry a 16 bit che utilizzano gruppi portano lookahead, e il carry-lookahead schema dell'Esercizio c.27.

C.31 [10] <§ C.6> Invece di pensare a un sommatore come un dispositivo che aggiunge due numeri di e poi collega la porta insieme, possiamo pensare al sommatore come un dispositivo hardware che può aggiungere tre ingressi insieme (un*Io*,b*Io*,c*Io*) E produrre due uscite
(S, *cIo+1*). Quando si aggiungono due numeri, c'è poco che possiamo fare con questo
osservazione. Quando abbiamo aggiunto più di due operandi, è possibile ridurre il costo del riporto. L'idea è di formare due somme indipendenti, chiamato S' (Bit somma) e C' (Portare bit). Alla fine del processo, è necessario aggiungere C' e S' together
utilizzando un sommatore normale. Questa tecnica di ritardare la propagazione portare fino alla fine
di una somma di numeri è chiamato *portare salvare oltre*.Il blocco di disegno in

basso a destra della Figura C.14.1 mostra l'organizzazione, con due livelli di effettuare salvare sommatori collegati da un unico sommatore normale.

Calcola i ritardi per aggiungere quattro numeri a 16 bit tramite la ricerca full carry-lookahead adder rispetto portare salvare con un carry-lookahead adder formare la somma finale. (Il T unità di tempo dell'esercizio C.28 è lo stesso.)

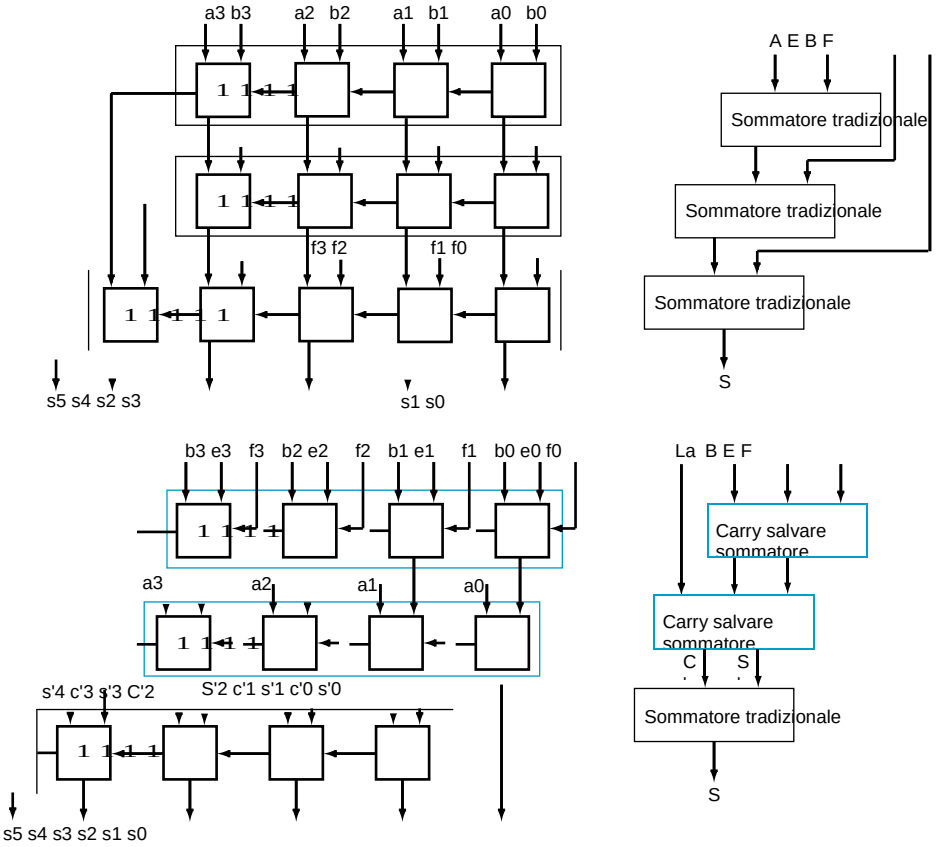


FIGURA C.14.1 ripple tradizionale trasportare e portare salvare aggiunta di quattro numeri a 4 bit. I dettagli sono mostrati a sinistra, con i singoli segnali in minuscolo, e le corrispondenti di livello più alto sono i blocchi a destra, con segnali collettivi in maiuscolo. Si noti che la somma di quattro *n*Bit numeri possono prendere *n*+ 2 bit.

C.32 [20] <§ C.6> Forse il caso più probabile di aggiungere numeri in una sola volta in un computer potrebbe essere quando si cerca di moltiplicarsi più rapidamente utilizzando sommatori molti di aggiungere molti numeri in un singolo ciclo di clock. Rispetto al l'algoritmo si moltiplicano nel capitolo 3, un riporto salvare schema con vipere molti potrebbero moltiplicarsi più di 10 volte più veloce. Questo esercizio stima il costo e la velocità di un moltiplicatore per moltiplicare due combinatorio positivi numeri a 16 bit. Si supponga di avere 16 intermedi termini M15, M14,. . . , M0, chiamato *prodotti parziali*,che contengono il multi-plicand AND con bit moltiplicatore m15, m14,. . ., M0. L'idea è quella di usare le vipere salvare per ridurre il *n*operandi in 2*n*/ 3 in gruppi paralleli di tre, e farlo più volte fino ad ottenere due grandi numeri di aggiungere insieme con un sommatore tradizionale.

Prima, mostra l'organizzazione blocco dei 16-bit portano salvare sommatori per aggiungere questi 16 termini, come mostrato a destra nella Figura C.14.1. Quindi calcolare i ritardi di aggiungere questi
16 numeri. Confronta questa volta per lo schema iterativo moltiplicazione nel Capitolo 3, ma solo supporre 16 iterazioni utilizzando un sommatore a 16 bit che ha piena portare lookahead la cui velocità è stata calcolata in Esercizio C.29.

C.33 [10] <§ C.6> Ci sono momenti in cui si desidera aggiungere un insieme di numeri. Si supponga di voler aggiungere quattro 4 bit numeri (A, B, E, F) con full adder a 1 bit. Ignoriamo portare lookahead per ora. Si sarebbe probabilmente collegare i sommatori a 1 bit nell'organizzazione nella parte superiore della Figura C.14.1. Sotto la tradizionale orga-nizzazione è un'organizzazione romanzo di full adder. Prova ad aggiungere quattro numeri che utilizzano entrambe le organizzazioni per convincere te stesso che si ottiene la stessa risposta.

C.34 [5] <§ C.6> Innanzitutto, visualizzare la struttura del blocco 16-bit portano salvare sommatori per aggiungere questi 16 termini, come mostrato nella Figura C.14.1. Si assuma che il ritardo attraverso ogni 1-bit sommatore è 2T. Calcolare il tempo di aggiunta di quattro 4-bit numeri all'organizzazione in alto rispetto dell'organizzazione al fondo della Figura C.14.1.

C.35 [5] <§ C.8> Molto spesso, ci si aspetterebbe che in un diagramma di distribuzione contenenti una descrizione dei cambiamenti che avvengono su un ingresso di dati *D*e un clock di ingresso *C*(Come nelle figure C.8.3 e C.8.6 alle pagine C-52 e C-54, rispettivamente), ci sarebbero differenze tra le forme d'onda di uscita (*Q*) Per un latch D e un D flip-flop. In una frase o due, descrivere le circostanze (ad esempio, la natura degli ingressi) per cui non ci sarebbe alcuna differenza tra le forme d'onda di uscita due.

C.36 [5] <§ C.8> Figura C.8.8 a pagina C-55 viene illustrata l'implementazione del file di registro per il datapath MIPS. Pretendere che un file di registro nuovo da costruire, ma che ci sono solo due registri e una sola porta di lettura, e che ogni registro ha solo due bit di dati. Ridisegna Figura C.8.8 in modo che ogni filo nel diagramma corrisponde a solo 1 bit di dati (a differenza del diagramma di figura C.8.8, in cui alcuni fili sono 5 bit e alcuni fili di 32 bit). Ridisegna i registri utilizzando flip-flop. Non c'è bisogno di mostrare come implementare un flip-flop o un multiplexer.

C.37 [10] <§ C.10> Un amico vorrebbe di creare un "occhio elettronico" per l'uso come dispositivo di sicurezza falso. Il dispositivo è costituito da tre luci allineati in una fila, controllato dal uscite di sinistra, centrale e destro, che, se affermati, indicano che la luce dovrebbe essere accesa. Solo una luce è accesa in un momento, e la luce "mosse" da sinistra a destra e poi da destra a sinistra, in modo da spaventare i ladri che credono che il dispositivo sta monitorando la loro attività. Disegnare la rappresentazione grafica per la macchina a stati finiti utilizzato per specificare l'occhio elettronico. Notare che la velocità di movimento dell'occhio sarà controllata dalla velocità di clock (che non deve essere troppo grande) e che vi sono essenzialmente nessun ingresso.

C.38 [10] <§ C.10> {Ex. C.37} Assegnare numeri statali per gli stati del

macchina a stati finiti che hai costruito per l'Esercizio C.37 e scrivere una serie di equazioni logiche per ciascuna delle uscite, comprese le next-bit di stato.

C.39 [15] <§ § C.2, C.8, C.10> Costruire un contatore a 3 bit con tre flip-flop D e una selezione di porte. Gli ingressi dovrebbe consistere in un segnale che resetta il contatore a 0, chiamato *reset*,e un segnale per incrementare il contatore, denominato *inc*. Le uscite devono essere il valore del contatore. Quando il contatore ha valore 7 ed è incre-mentato, dovrebbe avvolgere e diventano 0.

C.40 [20] <§ C.10> A *Gray codice* è una sequenza di numeri binari con la proprietà che non più di 1 bit passa a spostarsi da un elemento della sequenza all'altra. Ad esempio, qui è un 3-bit codice binario grigio: 000, 001, 011, 010, 110,
111, 101, e 100. Utilizzando tre D flip-flop e un PLA, la costruzione di un contatore a 3 bit codice Gray che ha due ingressi: *reset*,che imposta il contatore a 000, e *inc*, Il che rende il via contatore al valore successivo nella sequenza. Notare che il codice è ciclico, in modo che il valore dopo 100 nella sequenza è 000.

C.41 [25] <§ C.10> Vogliamo aggiungere una luce gialla al nostro esempio semaforo a pagina C-68. Faremo questo cambiando l'orologio di funzionare a 0,25 Hz (un 4-secondo tempo di ciclo), che è la durata di una luce gialla. Per evitare che le luci verdi e rosse di ciclismo troppo in fretta, si aggiunge un timer di 30 secondi. Il timer dispone di un ingresso singolo, chiamato *TimerReset*,che riavvia il timer, ed una singola uscita, chiamato *TimerSignal*, che indica che il 30 secondi di tempo è scaduto. Inoltre, dobbiamo ridefinire le indicazioni stradali per includere giallo. Facciamo questo la definizione di due segnali di uscita per ogni luce: verde e giallo. Se l'uscita NS verde si afferma, la luce verde è accesa, se il NSyellow uscita si afferma, la luce gialla è accesa. Se entrambi i segnali sono spenti, la luce rossa è accesa. Fare *non* affermano entrambi i segnali verdi e gialle, allo stesso tempo, dal momento che i piloti americani saranno certamente confuso, anche se i conducenti europei capire cosa significa! Disegnare la rappresentazione grafica per la macchina a stati finiti per questo controller migliorata. Scegliere nomi per gli stati che sono *diverso* dai nomi delle uscite.

C.42 [15] <§ C.10> {Ex. C.41} Scrivete le tabelle accanto a stato e di uscita, la funzione per il controller semaforo descritto nell'Esercizio C.41.

C.43 [15] <§ § C.2, C.10> {Ex. C.42} Assegnare numeri di stato agli stati nel traf-fico esempio luce di Esercizio C.41 e utilizzare le tabelle di esercizio C.42 a scrivere una serie di equazioni logiche per ciascuna delle uscite, comprese le uscite a stato successivo .

C.44 [15] <§ § C.3, C.10> {Ex. C.43} Implementare le equazioni logiche di esercizio
C.43 come
PLA.

§ C.2, pagina C-8: No. Se *La*=1, *C*=1, *B*=0, il primo è vero, ma la seconda è

falsa.
§ C.3, pagina C-20:
C.
§ C.4, pagina C-22: Sono tutti uguali.

§ C.4, pagina C-26: un =0, B =1. § C.5, pagin

a C-38: 2.
§ C.6, pagina C-47: 1.
§ C.8, pagina C-58: c.
§ C.10, pagina C-72: b.
§ C.11, pagina C-77: b.

Le risposte alle Controllare Yourself