CAPITOLO 5: STILI DI CODIFICA PER LOGICHE PROGRAMMABILI

1. DEFINIZIONE DI PROGETTO PORTABILE

Una logica programmabile può avere tre campi di applicazione: fornire uno strumento per la prototipazione di un Gate Array, rimediare ad un baco presente in uno già realizzato o correggere errori compiuti in fase di stesura delle specifiche di una piastra, essere parte integrante di un progetto.

Se l'obiettivo è la prototipazione, non è possibile fare uso di tutte le tecniche e di tutti gli strumenti messi a disposizione dalle logiche programmabili che, nei restanti casi, possono essere adoperati per realizzare il progetto in maniera più compatta ed efficiente.

Quando si effettua una prototipazione, lo scopo dell'operazione è la verifica del codice, non della tecnologia. Onde evitare di dedicare del tempo ai processi di fitting, sottraendolo a quello destinato ad altri scopi, è possibile, realizzando un breadboard di test, testare le parti veloci in una PLD le altre in un FPGA. Il codice, a sua volta, deve rispettare delle regole di portabilità Infatti non tutte le PLD hanno il segnale di preset, i bus tri-state interni sono disponibili solo per alcuni tipi di FPGA, i Gate Array non hanno look-up table che all'occorrenza diventano RAM 16x1.

Esistono quindi delle regole da rispettare in fase di progettazione di una rete logica. Queste consentono di avere uno stile di progettazione comune, di avere un progetto portabile su qualsiasi tecnologia, di realizzare una rete la cui frequenza di funzionamento sia determinata dalla tecnologia e non dal progetto.

Si definisce progetto portabile una rete logica in grado di esplicare la sua funzionalità sia mediante logica discreta TTL-CMOS-ECL, sia in una PLD, sia in un FPGA che in un Gate Array. E' possibile progettare con modalità non portabili ma questo dipende dal tipo di lavoro da compiere e di ottimizzazione da fare. Infatti allo scopo di sfruttare tutte le potenzialità di una logica programmabile è possibile realizzare un codice talmente ottimizzato da risultare praticamente illeggibile per un altro tipo di logica, anche dello stesso fornitore ma di famiglia diversa.

2. REGOLE DI PROGETTO PORTABILE

I segnali di clock di un progetto devono essere trattati con la massima attenzione. Un progetto che evolve con un unico segnale di clock è sincrono se questo alimenta tutti i flip flop. Se il segnale di uscita di un flip flop pilota il segnale di clock di un gruppo di altri, non è più possibile parlare di progetto sincrono evocando la discendenza dal primo segnale di clock, ma di due gruppi di segnali sincroni a due diversi segnali di clock. Il mancato rispetto di queste regole può provocare problemi quando si devono interscambiare segnali prodotti da differenti sorgenti di clock anche se accomunate nella genesi. La prima conseguenza è la riduzione dei margini di sicurezza soprattutto per applicazioni veloci. La seconda, nel caso degli FPGA, è l'introduzione di una variabile casuale nel comportamento della logica dovuta alla modalità di piazzamento non sempre riproducibile se non con l'utilizzo di opportuni vincoli. Una buona soluzione, per ottenere parti del circuito a frequenza scalata, è quella di utilizzare lo stesso clock di sistema e di produrre gli opportuni segnali di enable.

I glitch, che in tecnologie di altro tipo possono essere trascurati, nelle logiche programmabili rappresentano eventi anche di alcuni nanosecondi. E' quindi opportuno evitare che i segnali di gate dei latch o di abilitazione dei tri-state siano prodotti da logica combinatoria non opportunamente riletta.

Allo stesso modo sono da evitare i ripple clock e soprattutto i gating clock che in certi sistemi di sviluppo sono fortunatamente segnalati come errori. Le race condition, a loro volta, provocano degli inneschi di oscillazioni, di presenza più o meno evidente, in base alla tecnologia target. Per questo motivo vanno fortemente evitate o usate con cautela (tipico esempio il comparatore di fase di tipo D di un pll).

Una delle regole da rispettare, quando occorre trattare in un blocco dei segnali generati con un clock diverso, è la rilettura di questi con il nuovo clock, prima del loro utilizzo. Se uno di questi segnali esterni occupa il ruolo di enable di un contatore, può accadere che, a causa del piazzamento casuale compiuto dal Fitter e dell'asincronicità del clock e del segnale di enable, alcuni bit del contatore percepiranno l'enable attivo e altri no. Il risultato è un disastroso conteggio casuale del contatore. Ciò è eliminabile con la determinazione univoca del segnale di enable che avviene con la sua rilettura con il clock che muove lo stesso contatore. Il segnale così riletto può, a questo punto, interagire con il contatore senza generare problemi.

Un altro evento disastroso si ottiene quando, esaurite le linee di clock globali, si affida la distribuzione di un ulteriore segnale di clock ad una generica risorsa di routing. Il problema non è sempre evidenziabile con la simulazione timing, ma è ben visibile con una attenta analisi statica. Quando il processo di placement piazza i flip flop, si creano degli skew tra le linee di routing che fungono da clock. Questi, in certi casi, possono superare il ritardo accumulato dalla logica combinatoria e dalle net necessarie alla connessione dei vari flip flop. Il verificarsi di una simile condizione produce un non corretto funzionamento della logica. Questa modifica comportamentale può subire delle mutazioni se si effettuano iterazioni del processo di placement e alterare le funzionalità in maniera imprevedibile al punto di evocare presenze fantasma all'interno della logica. Una possibile risoluzione del problema, ove ci si trovi davanti ad un numero contenuto di elementi sequenziali, si ha con l'adozione delle Longline o FastTrack Interconnect come risorse di routing convoglianti i segnali di clock. La logica sequenziale, piazzata sulla medesima colonna o in certi casi riga, viene alimentata da una linea ad alto fan-out e ciò non produrrà skew apprezzabili.

L'utilizzo dei segnali di reset o preset asincroni è consigliabile solo in fase di inizializzazione e non nell'evoluzione della rete. I problemi potrebbero sorgere per due motivi. Il primo è il piazzamento che trasporterebbe il segnale asincrono in posti diversi in tempi diversi. Il secondo è che il tempo di reazione dei flip flop varia a seconda della tecnologia e questo può entrare in conflitto con la frequenza operativa del circuito. E' quindi consigliabile usare, nelle condizioni normali di funzionamento, dei reset sincroni con il clock di sistema.

3. STILI DI CODIFICA PER PLD

Al fine di meglio comprendere una delle maggiori potenzialità delle PLD qual è l'xor hardware, occorre fare utilizzo delle equazioni booleane.

Le equazioni booleane descrivono le variazioni dei segnali e i cambiamenti degli stati in risposta a tutti i possibili eventi e condizioni. Rappresentano l'ultimo step di progettazione o implementazione prima di avere la rete descritta in bit. ABEL e PALASM ne sono due esempi.

Il linguaggio produce un'elaborazione di variabili booleane mediante operatori logici rappresentati nella seguente convenzione: ! (not), # (or), & (and), $ (xor), $$ (xor hardware).

L'xor hardware è una xor già prevista nella macrocella della PLD a valle del Product Term Allocator (altrimenti chiamato Product Term Sharing Array) e a monte dell' elemento sequenziale. Consente una più potente integrazione ed è utilizzata dalle hard macro. Il progettista può evocarla con l'utilizzo dell'operatore logico $$, ove la sintassi del sistema di sviluppo lo consenta, oppure agendo opportunamente sulle direttive dei sistemi di sintesi.

E' possibile effettuare una breve dimostrazione al fine di porre in risalto i risultati che si ottengono con l'utilizzo dell'xor hardware in termini di risorse occupate. Per tale dimostrazione, ed esclusivamente a titolo di esempio, è stata scelta la famiglia 1000 della Lattice.

Volendo realizzare un contatore con più di 8 bit (ad esempio 12), non è possibile usare i 3 GLB che ipoteticamente sono necessari se si utilizzasse una xor hardware. Facciamo riferimento alla quarta uscita del contatore:

Q3 = Q3 $ (Q2&Q1&Q0) = (!Q3&Q2&Q1&Q0) # (Q3&!(Q2&Q1&Q0)) =
= (!Q3&Q2&Q1&Q0) # (Q3&(!Q2#!Q1#!Q0)) =
= !Q3&Q2&Q1&Q0 # Q3&!Q2 # Q3&!Q1 # Q3&!Q0

La descrizione con l'xor hardware è immediata:

```
Q3=Q3 $$ (Q2&Q1&Q0)
```

Come si vede, il numero di ingressi resta inalterato per l'And-Array. Nel primo caso il numero di product term cresce in maniera direttamente proporzionale al numero delle Q, mentre nel secondo caso resta uno per ogni Q.

Volendo mappare in un unico GLB la Q4, Q5, Q6, Q7, verranno richiesti 5, 6, 7, 8 = 26 product term, cosa impossibile perchè il massimo è 20 (per la famiglia scelta cioè la 1000). Saràpossibile mappare solo Q4, Q5, Q6 per un totale di 18 product term.
Utilizzando l'xor hardware i product term richiesti saranno 4+4=8 e quindi le quattro uscite saranno incluse in un unico GLB senza problemi.

Johnson Counter

È' possibile realizzare un contatore con frequenza di funzionamento superiore alla frequenza massima dichiarata per ciascuno speed grade. Si tratta di un contatore Johnson e può effettuare solo divisioni pari indicate con 2*n dove n rappresenta il numero dei flip flop da utilizzare. Segue un esempio di contatore modulo otto.

```
SIGTYPE [Q0..Q3] REG OUT;   //L`uscita è Q3 ed il contatore divide per 8
EQUATIONS
        Q0.CLK=CLOCK;
        Q0=!Q3;
        Q1=Q0;
        Q2=Q1;
        Q3=Q2;
END;
```

Consideriamo il componente 2032-150. La massima frequenza di conteggio per un contatore modulo 8 realizzato con le xor è 166.6 MHz. Con la precedente implementazione risulta essere 222.2 MHz. Tuttavia la soluzione adottata richiede un'area maggiore in termini di flip flop per produrre lo stesso modulo di conteggio rispetto ad una soluzione classica (4 contro 3).

4. STILI DI CODIFICA PER FPGA

One_Hot Encoding

Il metodo tradizionalmente usato per generare macchine a stati produce una grossa sezione combinatoria con l'utilizzo di pochi flip flop. Questo metodo di codifica è molto vantaggioso per le PLD che hanno blocchi di logica combinatoria molto potente e pochi flip flop. Negli FPGA un simile approccio è inefficiente in termini di densità e velocità La codifica One-Hot consiste nell'assegnare ad ogni stato della macchina a stati un flip flop. Questo è possibile grazie all'abbondanza di registri presenti in un FPGA e consente di realizzare una sezione combinatoria più snella. La minore area impiegata per il calcolo del singolo stato e il minor numero di livelli di logica consentono l'innalzamento della frequenza di funzionamento della macchina a stati.

Pipelining

Utilizzando l'abbondante numero di registri, al fine di aumentare la frequenza operativa del sistema, è consigliabile ridurre al minimo il numero di livelli di logica piazzando delle opportune riletture nei codici o negli schematici. Il ritardo di attraversamento del segnale nella logica ovviamente crescerà in misura proporzionale al numero degli stadi di rilettura.

LFSR Counter

Il Linear Feedback Shift Register Counter è un contatore realizzato mediante shift register e xor gate. Un CRC-16 è un esempio di contatore a 16 bit. Il vantaggio risiede nel limitato apporto di logica combinatoria e nelle limitate risorse di routing, in termini di numero e lunghezza necessarie, cosa che consente un'elevata frequenza di funzionamento. La lunghezza di conteggio di 2^n -1 al posto di 2^n e una sequenza di valori random sono le ricadute da tenere in considerazione.

FSM on ROM

Negli FPGA la memoria, distribuita o concentrata, può essere usata non solo come RAM, Single o Dual port, ma anche come ROM. Questa ROM può essere usata per realizzare una macchina a stati. L'indirizzamento alla memoria individuerà gli ingressi e lo stato presente; i dati estratti dalla memoria, le uscite e lo stato successivo.

Shift Register on LUT

La LUT, per certe tecnologie, può essere vista come una RAM 16x1. E' possibile realizzare uno shift register senza flip flop utilizzando una o più LUT come RAM e indirizzandole con un contatore esterno. I dati verranno scritti in una posizione ben precisa della memoria e al colpo di clock non saranno loro a muoversi bensì il contatore che li indirizza. Il risparmio di flip flop in applicazioni in cui è necessario creare elementi di ritardo è di un fattore sedici, anche se va tenuta in conto la presenza del contatore di indirizzamento.

Nella famiglia Virtex della Xilinx tale contatore non è più necessario perchè la LUT può essere direttamente utilizzata come elemento combinatorio, memoria o shift register di sedici bit.

DLL (Delay Locked Loop)

La distribuzione del clock e la minimizzazione dello skew su una linea di clock, al crescere delle dimensioni di un FPGA, possono essere un problema. Utilizzando dei Delay Locked Loop si riesce a contrastare questo duplice fenomeno. Il DLL è composto da una linea di ritardo programmabile e una logica di controllo. Questa, effettuando il confronto tra la fase del clock di ingresso e quella del clock distribuito all'interno del componente, imposta il corretto ritardo al fine di allineare i due segnali. Si agisce sul ritardo perchè una rotazione di fase di poco inferiore a 360° produce un anticipo della fase del segnale distribuito. Mediante i DLL, che vanno opportunamente evocati nel progetto logico, è anche possibile dividere il segnale in ingresso o raddoppiarne o quadruplicarne la frequenza. E' inoltre possibile effettuare rotazioni sulla fase con step di 90° e il clock mirror che consiste nell'allineamento del clock interno dell'FPGA con quello presente nel circuito stampato, al di fuori cioè del componente.

Mux mediante buffer tri-state interni

In un FPGA è possibile realizzare dei multiplexer sfruttando i buffer tri-state inglobati nell'architettura. Con un'opportuna codifica delle abilitazioni dei buffer interni è possibile associare ad ogni sorgente del multiplexer un segnale di alimentazione di ciascun buffer. In alcuni FPGA le linee tri-state interne sono reali in altri emulate.

5. LIBRARY OF PARAMETERIZED MODULES (LPM)

Un consorzio di venditori di logiche programmabili e venditori di software CAE (Computer Aided Engineering) ha stabilito uno standard tecnico per la progettazione digitale con l'intento di fornire un'interfaccia standard tra le parti di una logica programmabile dipendenti dalla tecnologia e quelle che non lo sono. Quest' interfaccia, chiamata libreria di moduli parametrizzabili (LPM), ha lo scopo di permettere ai progettisti l'accesso alle varie architetture senza una conoscenza dettagliata di ogni soluzione proposta dai vendor. E' composta da un insieme di 25 moduli specializzabili nelle dimensioni come contatori, addizionatori, sottrattori, multiplexer, moltiplicatori, comparatori, elementi di memoria come flip flop, latch, RAM e ROM.

Una versione preliminare dell'LPM è stata approvata nel marzo del 1991 dal comitato EDIF (Electronic Design Interoperability Format) che in fase successiva l'ha inglobata nello standard 2.0.0 (EIA-548-A).

L'LPM è supportato dai tool di sintesi e di simulazione e ogni vendor fornisce un proprio kit per la creazione di questi moduli nella propria tecnologia. E' possibile avere moduli sia in forma di schematico che in forma di codice. Esempi sono il MegaWizard Plug-In Manager dell'Altera o l'X-blox, adesso sostituito dal LogiBlox, della Xilinx.

Gli scopi dell'LPM sono di avere un design-entry indipendente dall'architettura e dal tool usato, un efficiente mapping del progetto, una completa copertura di tutti i moduli necessari alla progettazione digitale. Ma quando si passa attraverso i tool di sintesi non sempre si ha una soluzione efficiente per cui si preferisce utilizzarne direttamente la netlist evitando che il sintetizzatore interferisca. Tra i motivi può sicuramente contribuire la rivalità tra i fornitori dei tool di sintesi che, proteggendo i propri investimenti, non forniscono utili indicazioni al fornitore delle librerie.

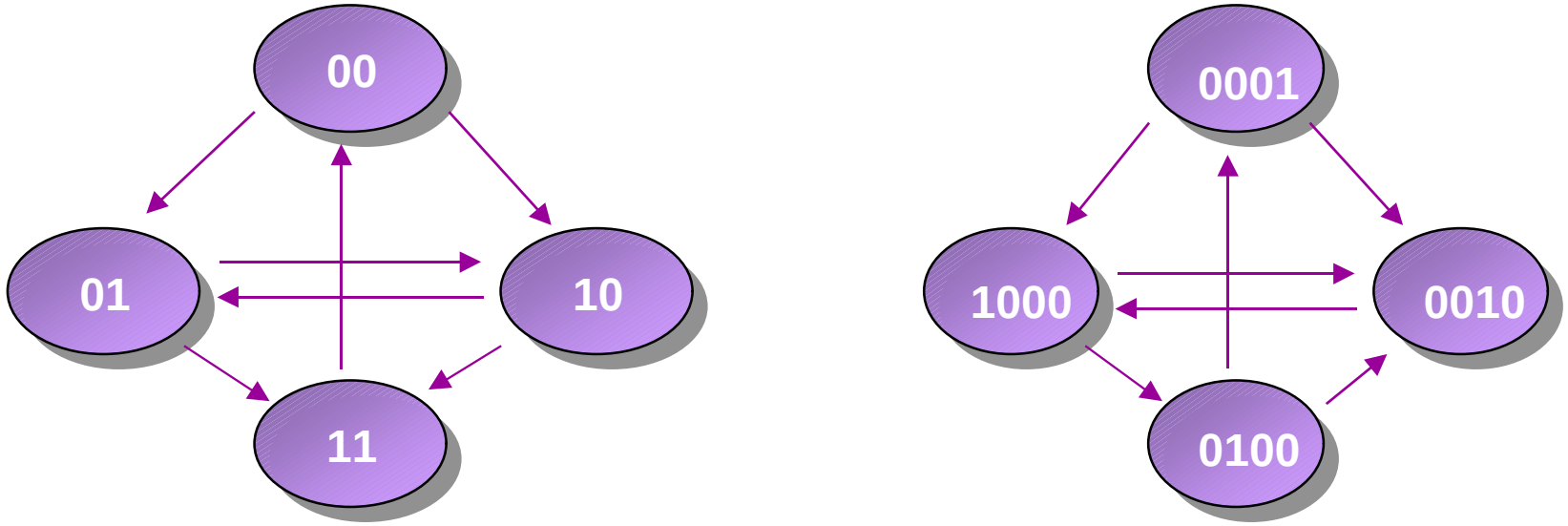
6. CORE

Con l'incremento delle densità e la diminuzione dei tempi di sviluppo può essere vantaggioso utilizzare macrofunzioni già realizzate e testate dal fornitore di logica programmabile o da ditte specializzate. Tipicamente è possibile acquistare, o avere libero accesso, al codice VHDL o Verilog della macro, al suo modello simulabile, alla netlist ottenuta alla fine di un processo di sintesi giàottimizzato.

Sono disponibili macrofunzioni del tipo Dual port RAM, FIR realizzati utilizzando l'aritmetica distribuita seriale o parallela, moduli di calcolo FFT o DFT, interfacce PCI, ALU, UART, DRAM Controller, Microprocessori, Moduli ATM, Codificatori e Decodificatori Reed-Solomon, Decodificatori basati sull'algoritmo di Viterbi.

L'Altera divide i suoi core in Megacore Macro Function che sono moduli base di libero accesso e in AMPP Megafunction (Altera Megafunction Partner Program) che sono core acquistabili. Una cosa analoga è compiuta dalla Xilinx con i suoi LogiCore e Alliance Core.

FIGURA 1: Binary Encoding vs One-Hot Encoding.



Binary Encoding

- n bits for 2^n states
- More encoding logic per bit
- Fewer flip-flops
- Best suited for combinatorial-intensive architectures like PLD

One-Hot Encoding

- One bit per state
- Less logic per bit
- More flip-flops
- Best suited for register-intensive architecture like FPGA

FIGURA 2: Esempio di pipelining.

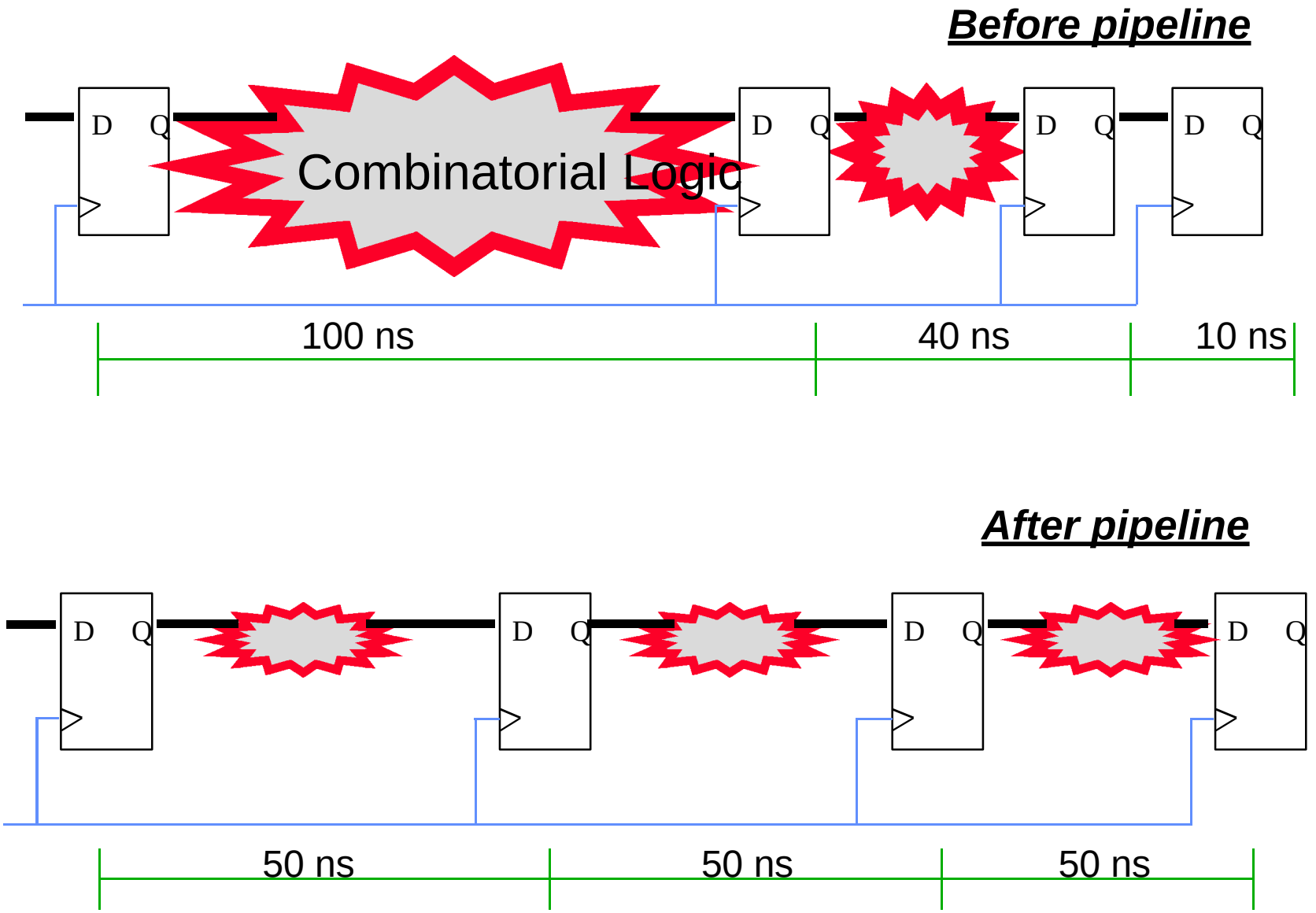


FIGURA 3: LFSR counter. Esempio realizzativo con un 16 bit CRC che produce un 16 bit counter.

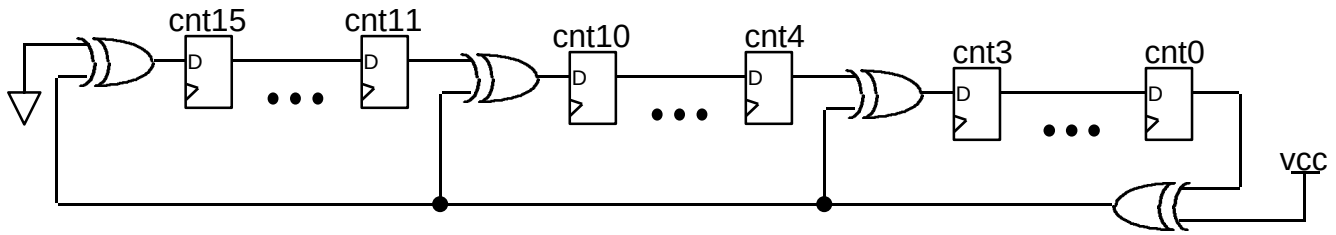
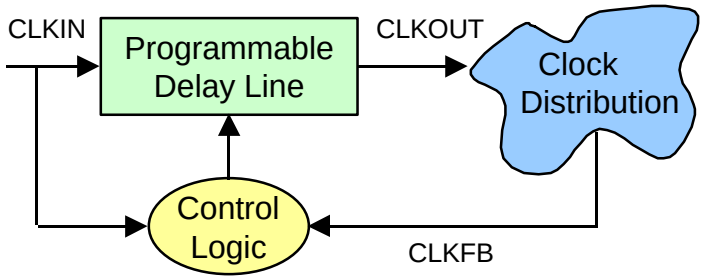
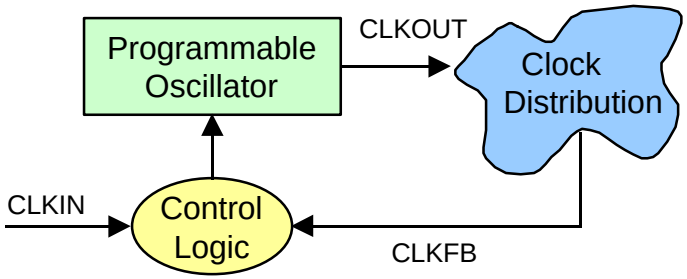


FIGURA 4: Implementazione del DLL per la riduzione dello skew e confronto con una realizzazione mediante PLL.



DLLs use Programmable Delay Line in Conjunction with Control Logic that Selects the Delay to Match the Distribution



PLLs use Programmable Oscillators in Conjunction with Phase Detectors & Filters to Phase Adjust the Clock

FIGURA 5: Implementazione di un multiplexer con un bus realizzato con buffer tri-state interni.