CAPITOLO 4: IL SISTEMA DI SVILUPPO

1. ITER DI PROGETTO

La realizzazione di una logica programmabile avviene mediante l'utilizzo di una serie di tool software che assistono il progettista in tutte le fasi necessarie alla produzione del file di programmazione.

Concettualmente l'iter di progetto può dividersi in una fase di Design Entry, una fase di Simulazione Funzionale (cioè a ritardi unitari), una fase di Sintesi (se si utilizzano linguaggi di descrizione ad alto livello), una di Fitter, una di Analisi Statica dei Ritardi, una di Simulazione Timing (non obbligatoria), eventualmente una di Download (ignorata se si utilizza la programmazione mediante microprocessore o mediante eprom) e una di verifica in-circuit. L'insieme dei due tipi di simulazione, della verifica in-circuit e dell'analisi statica dei ritardi forma le azioni di verifica sul progetto (Design Verification).

2. DESIGN ENTRY

Il Design Entry, partendo dalle specifiche di progetto, ha come obiettivo la realizzazione della rete logica in formato elettronico, cioè in uno o più file. Esistono tre modi per realizzare questa fase: schematico, equazioni, linguaggi di descrizione ad alto livello. A questi, in base alle necessità, è possibile affiancare una sintassi o un sistema grafico per la descrizione di macchine a stati.

Lo schematico raccoglie la modalità di funzionamento della rete in un formato grafico che risulta essere una collezione di simboli e connessioni e in un file che prende il nome di netlist. Tipici esempi sono il ViewDraw della ViewLogic, l'OrCAD, il Synario.

La modalitàad equazioni descrive la rete in una sintassi del tipo ABEL o molto vicina a questa. Anche se non risulta essere di immediata comprensione, questo è l'obiettivo finale a cui tendono tutti i Fitter e un eventuale riscontro a livello implementativo si ottiene in questo formato. Progettare ad equazioni per un progettista è simile a progettare in assembler per un programmatore.

L'utilizzo dei linguaggi di descrizione dell'hardware ad alto livello (HDL) ha consentito, negli ultimi anni, di incrementare significativamente sia la produttività in termini di velocità di progettazione, sia l'interoperabilità in quanto garantisce un'interfaccia standard tra i vari progettisti. Il progetto viene descritto da una sequenza di istruzioni, definite codice, e la sua verifica avviene mediante un simulatore che consente la validazione del circuito essenzialmente da un punto di vista virtuale, in quanto non è stato prodotto nessun tipo di gate, ma solo emulato. Il progetto può essere arricchito eventualmente da schematici che lo organizzano in blocchi di alto livello all'interno dei quali è presente il codice. Synario, Renoir della Mentor, Sge della Synopsys ne sono esempi. Il passaggio al mondo dei gate è subordinato alla presenza di un sintetizzatore di logica che, interpretando il codice, sotto opportune direttive di sintesi, produce la netlist necessaria al Fitter per compiere le successive operazioni. I linguaggi comunemente diffusi sono il Verilog e il VHDL. Tra i sintetizzatori è possibile citare il Design Compiler, l'FPGA Compiler, l'FPGA Express della Synopsys, il Sinplify della Sinplicity, il Leonardo, il Galileo, l'Autologic II della Exemplar-Mentor, il Concept della Cadence.

3. DESIGN VERIFICATION

La simulazione è uno dei processi di verifica della logica realizzata con una delle precedenti metodologie. È di tipo funzionale se determina la corretta evoluzione della logica sotto opportuni stimoli senza tenere conto dei ritardi, o meglio basandosi su ritardi unitari. È' di tipo timing se, riutilizzando gli stessi stimoli, il progetto continua a garantire integrità funzionale quando alla fine del processo di fitter sono tenuti in conto i ritardi delle net e dei blocchi sequenziali e combinatori. Tipicamente la simulazione funzionale si compie prima del processo di sintesi, quella timing dopo il processo di fitter. E' possibile compiere una simulazione timing sia dopo il processo di sintesi sia in una fase intermedia del processo di fitter (cioè dopo il placement e prima del routing). Queste possibilità rientrano nell'iter standard delle operazioni solo nel caso di una verifica più dettagliata, nell'eventualità in cui si sia presentato un problema di non facile individuazione nella verifica in-circuit. Esempi di simulatori sono il Vss della Synopsys, il V-System e il ModelSim della Model Technology, il Verilog-XL e il LeapFrog della Cadence, il Quicksim II della Mentor.

L'analisi statica dei ritardi è una metodologia di verifica che, se usata correttamente, evidenzia problemi non sempre visibili in una simulazione timing. E' compiuta alla fine della fase implementativa condotta dal Fitter senza l'ausilio di vettori di test, e divide il progetto in categorie di percorsi punto-punto ordinati in maniera decrescente a partire dal peggiore. Esistono quattro categorie di segnali del tipo pad-to-setup, clock-to-pad, clock-to-setup, pad-to-pad, e per ogni categoria ci possono essere più gruppi associati a diversi segnali di clock. La categoria pad-to-setup consente di stabilire il ritardo di attraversamento tra pad e flip flop raggiunti dai segnali che fluiscono negli stessi pad. La categoria clock-to-pad analizza il ritardo tra gli elementi sequenziali che generano segnali diretti a pin di uscita con gli stessi pin di uscita. La categoria clock-to-setup, per ciascun segnale di clock del circuito, valuta la frequenza operativa del sistema e lo skew che subisce un segnale di clock non trasportato da linee di clock dedicate. La categoria pad-to-pad valuta il ritardo di attraversamento del segnale quando questo incontra, nella logica programmabile, solo pad e logica combinatoria. Il mancato rispetto di requisiti timing avvia due possibili azioni: una condotta mediante l'alterazione delle strategie seguite dal Fitter; l'altra, più radicale, compiuta con un re-design delle parti coinvolte al fine di ridurre il numero di livelli di logica.

La verifica in-circuit è l'ultimo banco di prova che valida sia la funzionalità desiderata che le effettive performance in termini di velocità Se il circuito non è sottoposto a severe condizioni di funzionamento in termini di tensione e temperatura, la verifica in-circuit sarà però lontana dai risultati ottenibili con l'analisi statica.

4. DESIGN IMPLEMENTATION

In questa fase del flusso di progetto, utilizzando la netlist come file di ingresso, viene realizzata l'implementazione nella tecnologia scelta. La netlist viene adattata al tipo di componente (package, speed grade, densità, famiglia) e viene prodotto come risultato finale il file di programmazione, il database per la simulazione funzionale e quello per compiere un'analisi statica. Ogni vendor ha un proprio sistema di sviluppo, alternativamente chiamato Fitter, ma concettualmente tutti effettuano le stesse operazioni. Ci sono solo da distinguere i processi che si effettuano nella realizzazione di una PLD da quelli compiuti per un FPGA.

5. IL SISTEMA DI SVILUPPO DI UNA PLD

Il primo processo che il sistema di sviluppo effettua è una verifica sintattica del contenuto presente nella netlist e delle eventuali incongruenze circuitali quali pad non collegati o logica non usata. La successiva fase prende il nome di partitioner. In essa, inizialmente, si realizza la minimizzazione della logica che, sfruttando i teoremi dell'Algebra di Boole, ha lo scopo di produrre il minor numero possibile di product term. Questi, successivamente aggregati in somme di prodotti e in base a requisiti di area o di velocità, sono virtualmente raggruppati, con la restante logica sequenziale e le eventuali xor hardware, in uno dei blocchi tipici messi a disposizione dalla tecnologia come GLB, LAB, Function Block, Logic Block. Per netlist provenienti dai programmi di sintesi, tipicamente, il lavoro del minimizzatore è superfluo e si passa direttamente alla fase di aggregazione. Alla fase di partitioner segue quella di placement in cui i blocchi virtuali trovano una collocazione fisica. Prima che inizi il routing, che ha il compito di interconnettere i vari blocchi, il software, se necessario, cambia posizione ai blocchi. Questo viene effettuato per stabilire qual è il piazzamento ottimale che assicura la routability, cioè la completa interconnettività della PLD. Se il processo termina senza problemi si passa alla fase di fusemap in cui si crea il file necessario alla programmazione del componente. Se il routing non viene assicurato, vengono esaminate differenti strategie. Se anche queste non sortiscono l'effetto desiderato, il software prova a scambiare la logica tra i vari blocchi, effettua un cambio della posizione dei pin, effettua duplicazioni della logica dentro ai blocchi. Se anche questo tentativo non si conclude felicemente la PLD viene dichiarata unroutable. Azioni radicali al di fuori del sistema di sviluppo, come cambio del componente o re-design, potrebbero risolvere questo problema.

6. IL SISTEMA DI SVILUPPO DI UN FPGA

I sistemi di sviluppo delle PLD, grazie all'architettura che le caratterizza, hanno meno variabili da analizzare nel processo di placement che risulta invece critico nel caso degli FPGA a routing segmentato. I macroblocchi, come GLB, FB, LAB, LB, una volta personalizzati dalla logica di utente, non hanno particolari vincoli di piazzamento, se non per effetto di un eventuale pin locking o per ragioni di routability: saranno tutti raggiungibili allo stesso modo dalla struttura di routing. Il software chiamato a gestire tali componenti richiede contenute risorse di calcolo e garantisce prestazioni predicibili, anche per successive iterazioni. Le stesse considerazioni possono essere fatte per il sistema di sviluppo di un FPGA a routing del tipo FastTrack.

Per quanto riguarda gli FPGA a routing segmentato, il punto cruciale del sistema di sviluppo è proprio il piazzamento. In base alla qualità dell'algoritmo che implementa tale processo è possibile avere prestazioni più o meno elevate. Questi algoritmi si basano sul soddisfacimento di opportune funzioni di costo. Queste tendono a produrre un risultato ottimo sia in termini di tempo e risorse di calcolo, che in termini di minor lunghezza possibile dei collegamenti nell'ottica di una massimizzazione della frequenza di funzionamento. Nella maggior parte dei casi l'algoritmo di piazzamento è di tipo statistico, per cui a parità di progetto, due esecuzioni del piazzamento, possono produrre due risultati diversi. Proprio per ovviare a tale inconveniente è stato introdotto il file di guida. Una volta compiuta una prima iterazione, grazie alle informazioni di piazzamento ed eventualmente di routing contenute in questo file, nelle successive iterazioni si lascia il più possibile inalterato il lavoro già compiuto, ponendo cura alle parti che hanno subito variazioni, in termini di incrementi di logica o modifiche della logica già presente.

I processi coinvolti in un sistema di sviluppo di un FPGA sono: translation, partition, place, route e bitmap generation. A questi vengono affiancati processi di verifica come l'analisi statica dei ritardi, file di report, floorplanner e visualizzatori in forma grafica della logica di utente implementata nel componente, rispetto alle risorse dello stesso.

La netlist ottenuta da tool di sintesi, da schematici o da sistemi di descrizione dell'hardware ad equazioni, viene letta dal programma preposto alla translation. Questo, indipendentemente dal tipo di FPGA usato, effettua un controllo logico sulla netlist (Design Rule Check), e si interessa di raccogliere altre netlist, eventualmente evocate nella principale, al fine di creare un unico progetto in un unico livello gerarchico (Merge). Il logical DRC verifica che ogni simbolo presente nella netlist sia una primitiva della libreria tecnologica del Fitter, che non ci siano segnali non connessi come origine o destinazione, che i buffer di clock siano stati usati correttamente, che i nomi dei simboli non diano origine ad ambiguità, che i pad siano connessi nel modo giusto. La netlist flatten, restituita dal traduttore senza la segnalazione di errori, può essere letta dal programma di partizione che effettua la trasformazione dal dominio logico (LUT, flip flop, buffer di I/O) a quello fisico (Logic Cell e I/O Cell).

Il programma di partitioner (in certi casi si parla di mapping), associa i blocchi del dominio logico a quelli che compongono la tecnologia target (device, package, speed grade), detti per questo fisici. Vengono inoltre assegnate le risorse globali come clock e reset. Il logical DRC, presente anche in questo caso, assicura che le risorse utilizzate siano conformi con il target scelto (ad esempio non si siano usati buffer di clock in numero maggiore del consentito). Si fa inoltre carico di rimuovere la logica non connessa o non utilizzata presente nella netlist flatten.

Al mapping segue la nevralgica fase di place & route. Il progetto partizionato in mattoncini congrui con quelli messi a disposizione dal componente, deve essere allocato in un sottoinsieme di questi. L'allocazione viene realizzata in maniera tale che alla fine del piazzamento tutte le celle devono poter essere collegate e devono essere rispettati i requisiti timing del progetto. E' possibile individuare tre tipi di algoritmo di piazzamento: il mincut, il simulated annealing, il general force-directed relaxation algorithm (GFDR).

Nel piazzamento compiuto con l'algoritmo mincut, il progetto viene inizialmente diviso in due cluster di eguale numero di blocchi separati da una linea immaginaria. Successivamente questi blocchi vengono scambiati tra i due cluster al fine di ridurre al minimo il numero di connessioni che attraversano la linea. Ciascuno dei due cluster viene a sua volta diviso in due ripercorrendo la stessa modalità di interscambio di blocchi. Il processo si conclude quando il cluster ha le dimensioni di un blocco.

Il simulated annealing è stato colonna portante dei programmi storici di place & route ormai non più usati, come ad esempio l'APR della Xilinx. Esso basava il suo processing su tre fasi: annealing, quenching e routing. Nella fase di annealing venivano compiuti una serie di piazzamenti dei blocchi in maniera semicasuale. Il piazzamento che sembrava fornire una migliore qualità veniva scelto per la successiva fase. Il requisito di qualità era la minima temperatura raggiunta in un processo simile a quello del raffreddamento di un solido a partire da uno stato amorfo. Indicatori di processo erano la variazione percentuale della temperatura (velocità di raffreddamento), il cambiamento percentuale rispetto alle condizioni di partenza, il grado di miglioramento che si aveva una volta stabilizzata la temperatura. Nella fase di quenching i blocchi venivano mossi solo se si otteneva un miglioramento della qualità del progetto in termini di routability e di velocità La successiva fase di routing aveva il compito di fornire tutte le interconnessioni

necessarie, agendo con le opportune priorità, desunte da timing specification di utente o da file di guida riguardanti una precedente fase di routing.

L'algoritmo GFDR associa le connessioni tra i blocchi al modello fisico di una molla. La rigidità della molla rappresenta la criticità della net. I blocchi vengono scambiati di posizione o ruotati ciclicamente fino ad ottenere un centro di gravità blocco per blocco.

Tutti i programmi di piazzamento evoluti effettuano la procedura sia su fattori interni che esterni. Inizialmente viene compiuto un constructive placement nel quale vengono considerati fattori come constraint di locazione, lunghezza delle connessioni, risorse di routing disponibili. Successivamente viene effettuato un optimizing placement in cui viene compiuta una operazione di raffinazione. Se esistono dei constraint di tipo timing è possibile realizzare un timing driven placement che può essere prioritario rispetto ai vincoli di locazione presenti nella fase di constructive placement o essere evocato nella fase di optimizing placement nella quale si dovrà tenere conto anche dei vincoli di piazzamento di altri blocchi.

Il routing è il processo che si fa carico di connettere i blocchi precedentemente piazzati. Anch'esso è diviso in due fasi: constructive routing e cleanup routing. La prima realizza una procedura iterativa che converge verso due obiettivi: connettere tutte le net e soddisfare tutti i timing constraint. La seconda, a partire dai risultati ottenuti dalla prima, riconsidera alcune connessioni al fine di minimizzare i ritardi di tutte le net e di decrementare il numero delle risorse di routing utilizzate. Questa fase può, a sua volta, essere di due tipi: cost based e delay based. Nel primo caso, le operazioni vengono eseguite al fine di minimizzare il tempo di calcolo, nel secondo, al fine di ottenere i migliori risultati timing a scapito del tempo e delle risorse necessarie all'elaborazione.

Oltre alla possibilità di adoperare un file di guida nel caso di routing incrementale, quando cioè, consolidata una parte del progetto, se ne aggiunge un'altra, è possibile effettuare il routing rientrante. Se i timing constraint sono troppo stringenti o se il progetto non è completamente routed, è possibile riavviare la procedura di place & route alterando i parametri che coordinano le attività dei due processi per trovare, se esiste, un punto di accordo tra requisiti e risultati.

Il database creato dal place & route viene tradotto, da programmi di bitmap, in un file di programmazione downloadabile chiamato bitstream. Usando i programmi di makeprom, è possibile creare l'immagine di una eprom che fungerà da supporto fisico per i dati di programmazione.

7. METODOLOGIE AVANZATE: FLOORPLANNER

Il Floorplanner è un tool di piazzamento grafico che fornisce all'utente la possibilità di collocare manualmente function generator o flip flop all'interno dell'FPGA scelto. E' possibile selezionare in una finestra contenente simboli logici quello interessato e, mediante l'utilizzo del mouse, trasportarlo in una finestra contenente il prospetto dei siti occupabili nell'FPGA. Con una simile metodologia è possibile aumentare le prestazioni per quelle porzioni di progetto critiche da un punto di vista timing sia intervenendo su un progetto già piazzato che su uno ancora da piazzare. Esistono due tipi di Floorplanner. Uno consente la sola visualizzazione della logica piazzata. Un altro sia la visualizzazione che il piazzamento. Le strategie che si possono seguire per quest'ultima azione sono diverse.

Effettuando il floorplan prima della fase di place & route, si stabiliscono a priori le coordinate di piazzamento di alcuni o tutti i blocchi di logica. Queste coordinate sono memorizzate in un file di location constraint che il processo di placement

leggerà verificandone la validità Terminata l'esecuzione delle direttive contenute nel file, tale processo valuta il piazzamento di eventuali blocchi non inclusi nel file e poi passerà il controllo delle operazioni al processo di routing.

Il floorplan può essere compiuto dopo il place & route al fine di migliorare alcune locazioni che non hanno soddisfatto i timing constraint. Una volta compiute le opportune modifiche dovrà essere riavviato il routing.

E' possibile compiere un floorplan iterativo. Viene compiuto un piazzamento manuale di una porzione di logica, la memorizzazione in un file di location constraint, la fase di place & route. Si valutano i risultati a questo punto ottenuti, si effettuano le opportune azioni correttive, si compie il piazzamento di un'altra parte di logica salvaguardando quella già posizionata, e così via.

Nei progetti costruiti in maniera incrementale è possibile che si verifichi uno dei seguenti eventi: aggiunta di logica, rimozione di logica, modifica della logica esistente. La strada consigliata, nel caso di un design entry del tipo schematico, è l'adozione del file di guida per le successive iterazioni del place & route. Nel caso dei linguaggi di descrizione dell'hardware di alto livello che fanno uso di un sintetizzatore di logica il cambio, anche minimo, della descrizione della logica, porta uno stravolgimento delle etichette assegnate ai blocchi logici. Questo conduce ad una completa dissintonia del file di guida dalla nuova logica che potrebbe essere una fedele replica della precedente a meno, ad esempio, di un inverter. Mediante il Floorplanner è possibile ripristinare, per i nuovi blocchi rinominati, il piazzamento ottenuto nella prima iterazione, mediante una copia manuale dell'assegnamento posizionale. In questo caso occorrono due sessioni del programma in cui in una viene impostato il vecchio piazzamento, nell'altra quello che si ricostruisce.

8. METODOLOGIE AVANZATE: RPM

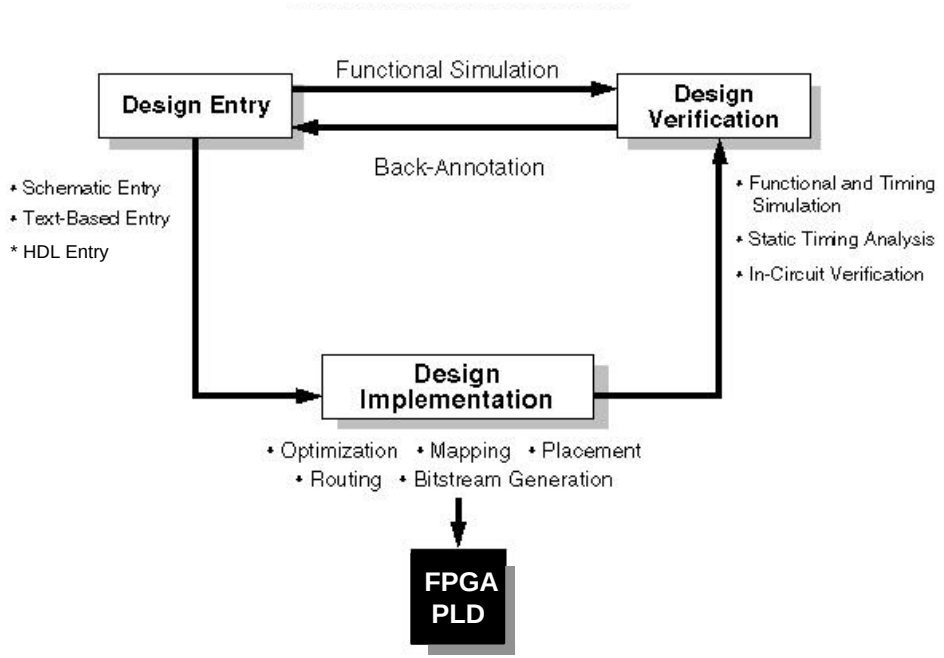
Le Relationally Placed Macro sono macro, disponibili in libreria tecnologica in fase di progettazione, che contengono la descrizione di una particolare funzionalità logica a cui sono associate delle informazioni di location constraint relative, assegnate ad ogni componente costituente la macro.

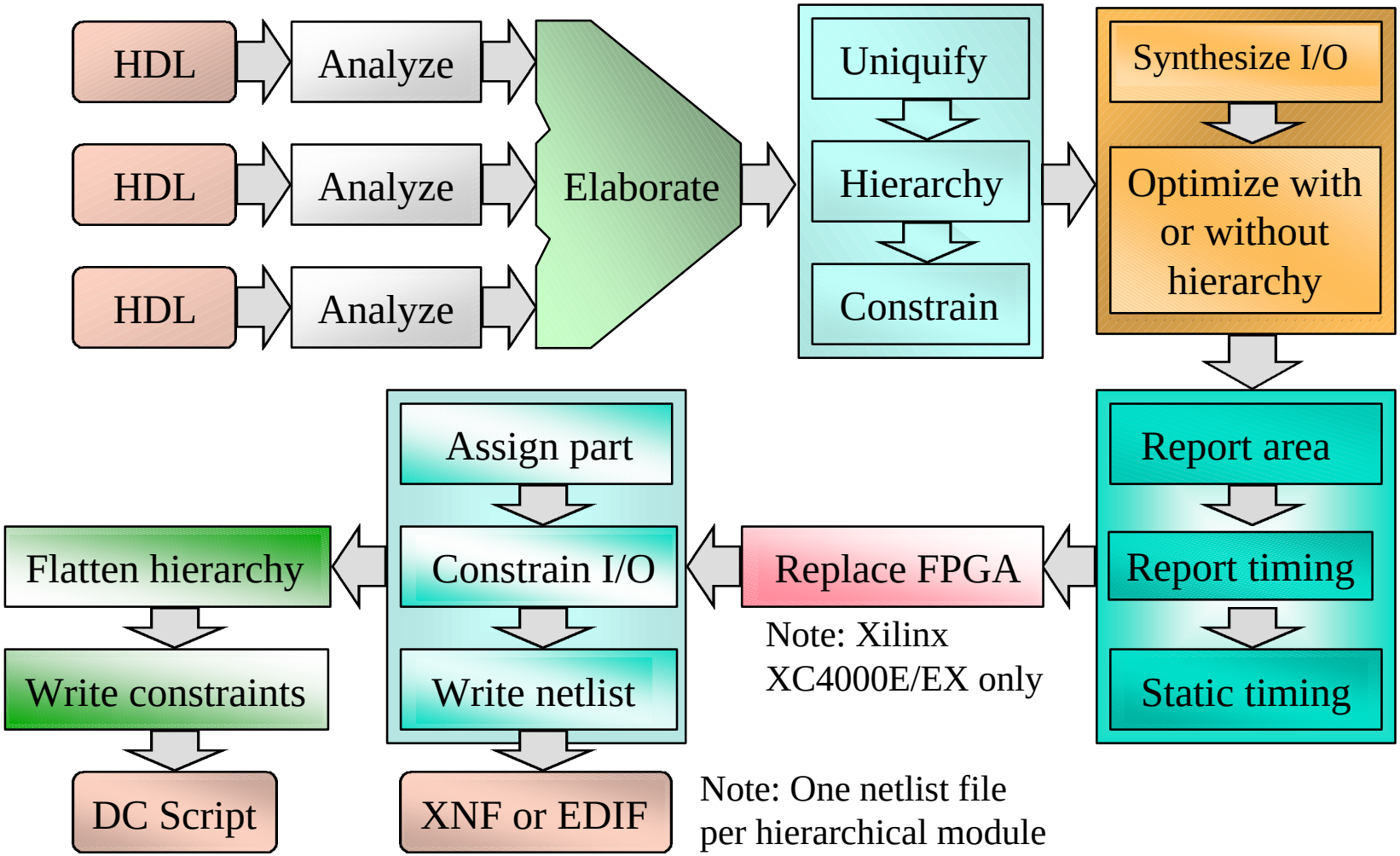
In un FPGA, un sito fisico, ha una posizione individuabile da una coordinata assoluta (ad esempio riga A, colonna B). Assegnare le coordinate relative tra un gruppo di blocchi all'interno di una macro, significa legare in maniera rigida il loro aspetto all'atto del piazzamento. Dove questo avverrà nel componente, sarà a discrezione del placer. Se, ad esempio, abbiamo una macro di due blocchi individuati dalle coordinate relative (riga_A,colonna_A) e (riga_A,colonna_B), questa macro può ritrovarsi piazzata in qualsiasi posizione individuata dalle coordinate assolute ad esempio (riga_C,colonna_D) e (riga_C,colonna_E).

Con le RPM è quindi possibile influenzare il piazzamento già in fase di progettazione al fine di ottenere elevate prestazioni in termini di timing. Infatti le coordinate relative sono scelte per minimizzare le interconnessioni e usare il più possibile le risorse di routing ai più bassi livelli gerarchici (Feedback, Direct Connect).

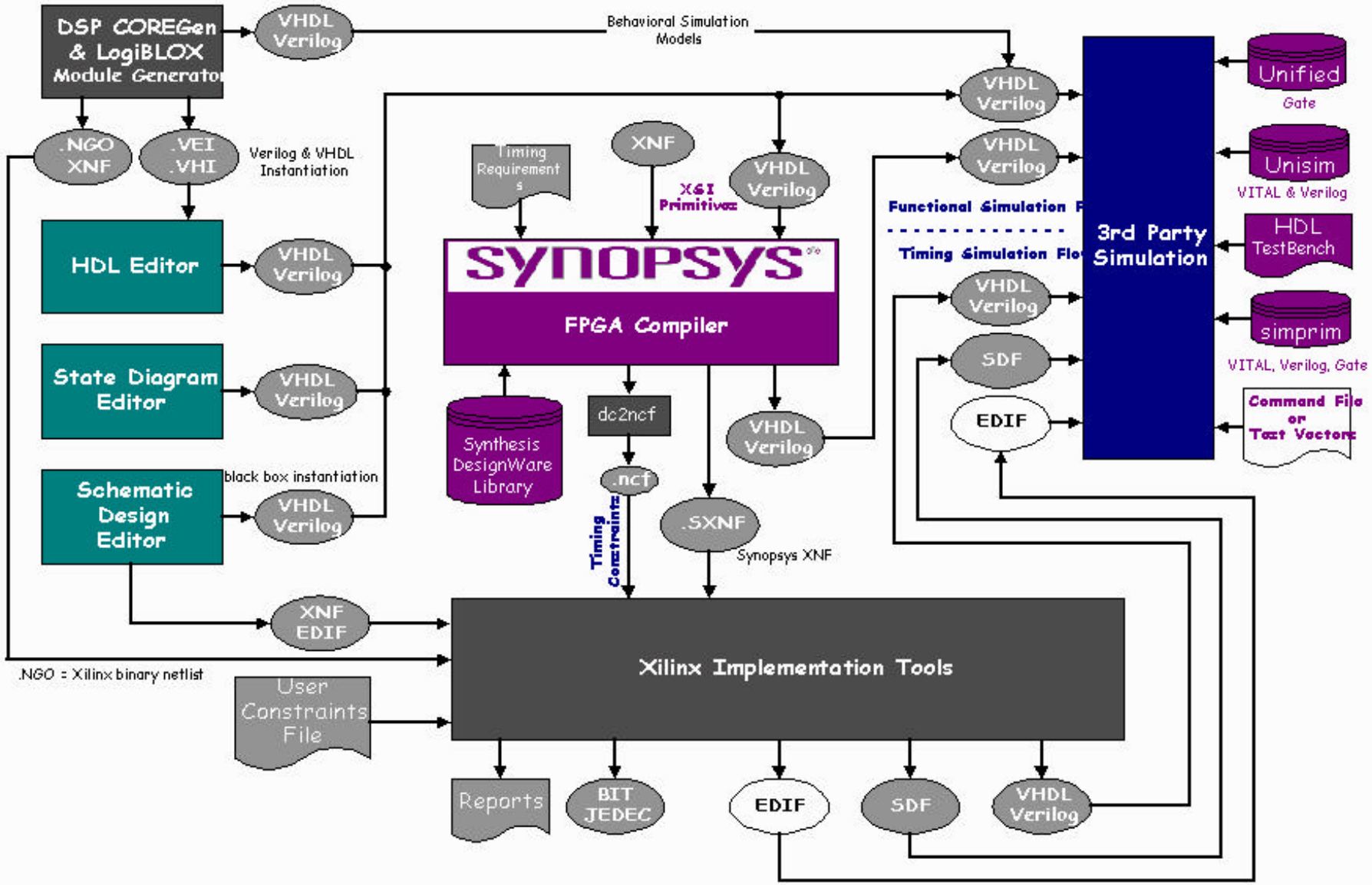
Una macro realizzata RPM può contenere flip flop, LUT, logica carry, memoria distribuita.

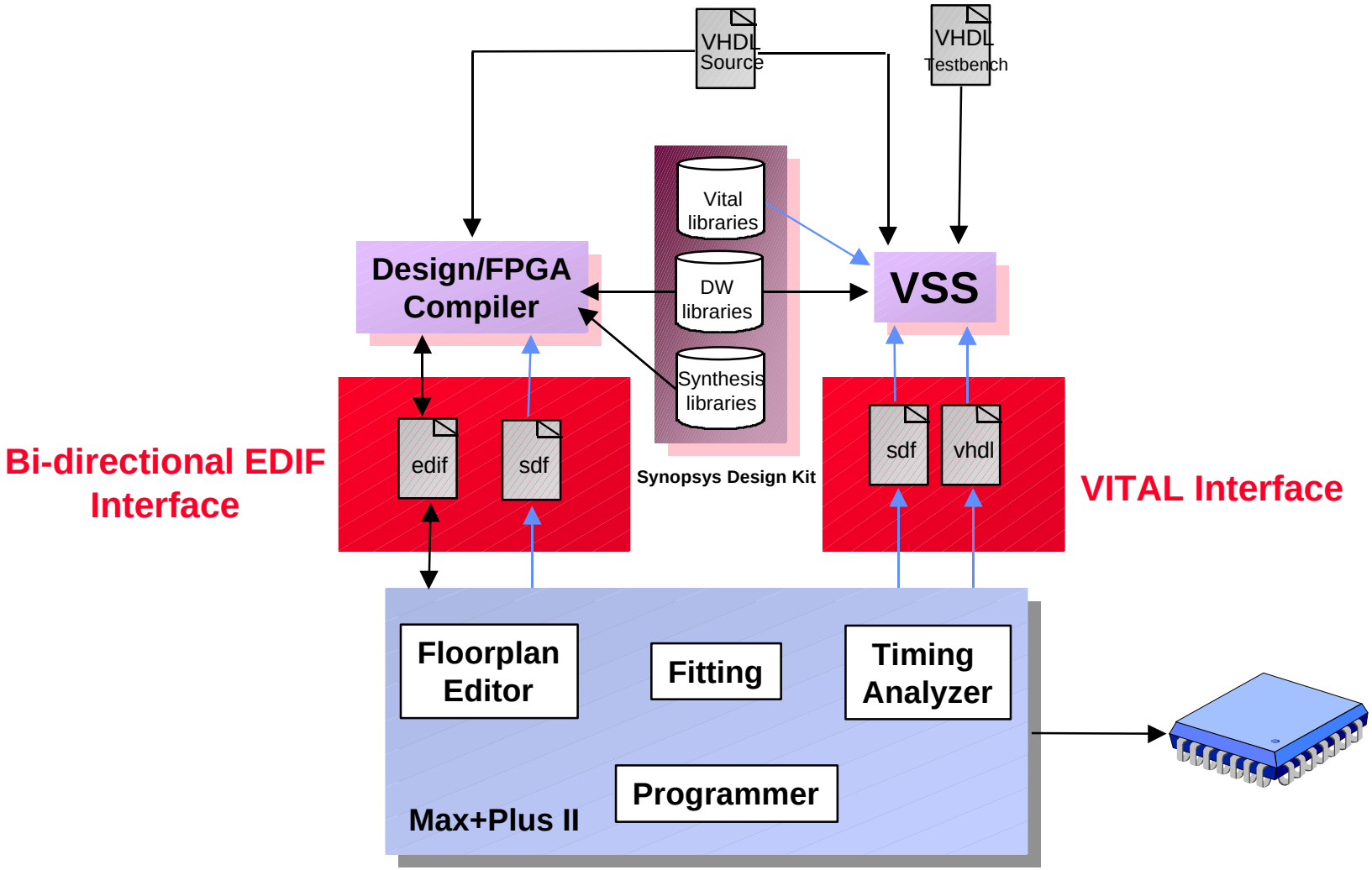
FIGURA 1: Principali fasi della progettazione, implementazione e verifica di una logica programmabile.

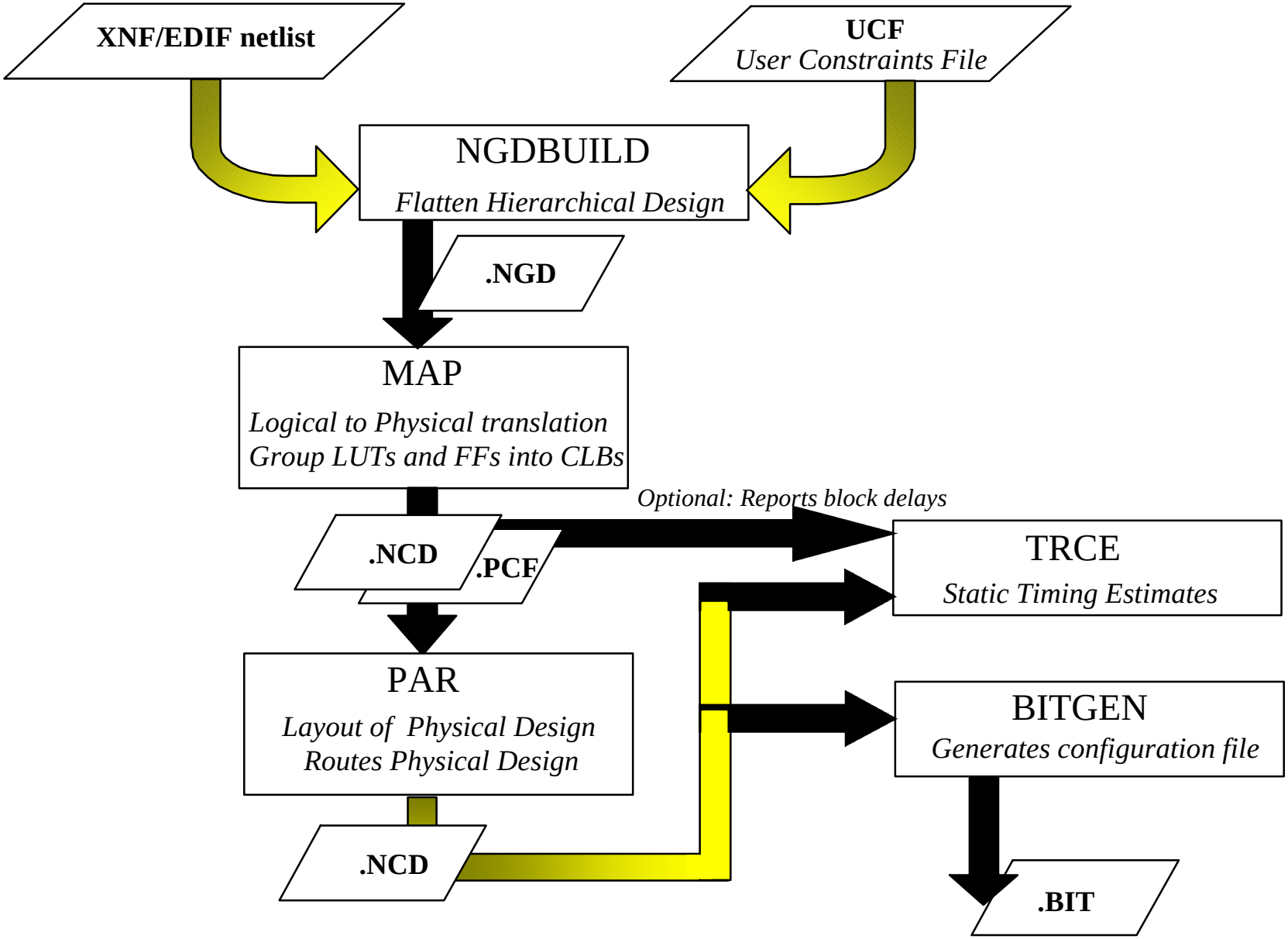



HDL
Analyze
HDL
Analyze
HDL
Analyze
Elaborate
Uniquify
Hierarchy
Constrain
Synthesize I/O
Optimize with or without hierarchy
Report area
Report timing
Static timing
Replace FPGA
Note: Xilinx XC4000E/EX only
Assign part
Constrain I/O
Write netlist
XNF or EDIF
Note: One netlist file per hierarchical module
Flatten hierarchy
Write constraints
DC Script
Note: Xilinx M1 only

Synopsys FPGA Compiler Implementation Flow Diagram



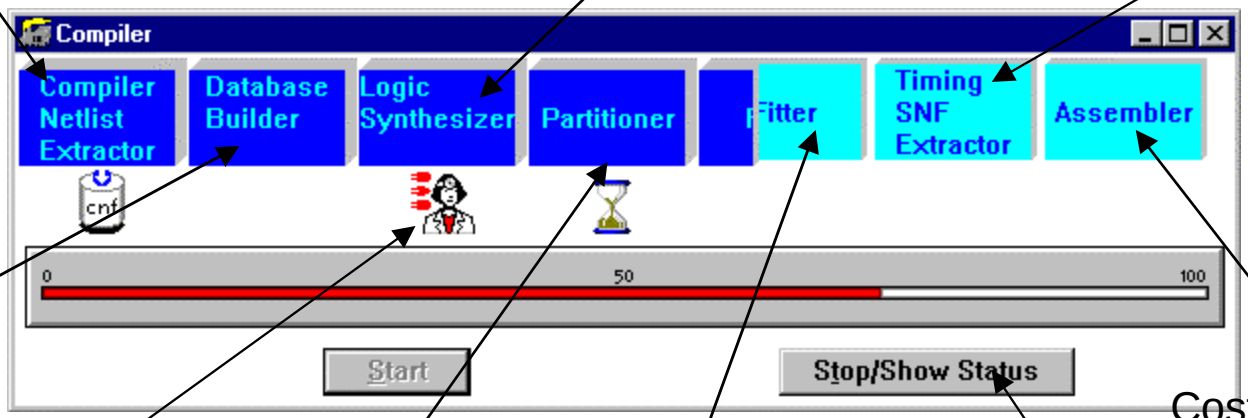

VHDL
Source
VHDL
Testbench
Vital
libraries
DW
libraries
Synthesis
libraries
Synopsys Design Kit
Design/FPGA
Compiler
VSS
edif
sdf
sdf
vhdl
Bi-directional EDIF
Interface
VITAL Interface
Floorplan
Editor
Fitting
Timing
Analyzer
Programmer
Max+Plus II


XNF/EDIF netlist
UCF
User Constraints File
NGDBUILD
Flatten Hierarchical Design
.NGD
MAP
Logical to Physical translation
Group LUTs and FFs into CLBs
Optional: Reports block delays
.NCD
.PCF
TRCE
Static Timing Estimates
PAR
Layout of Physical Design
Routes Physical Design
BITGEN
Generates configuration file
.NCD
.BIT

Estrae la netlist dell'intero progetto e controlla gli errori di sintesi

Esegue la sintesi logica utilizzando le opzioni di sintesi e di timing nel file .act

Costruisce il file per la simulazione e l'analisi delle temporizzazioni



Costruisce il database di sintesi

Costruisce i file di programmazione

Design Doctor controlla le violazioni del progetto

Controlla il fit nel dispositivo selezionato e - se necessario - partiziona il tutto su più dispositivi

L'utente può fermare la routine di place & route routine per esaminare lo stato

Esegue l'algoritmo di place & route e genera il file .rpt sommarizzando l'utilizzazione del dispositivo e i problemi di fitting

FIGURA 7: Floorplanner window (Xilinx Floorplanner Data Book).

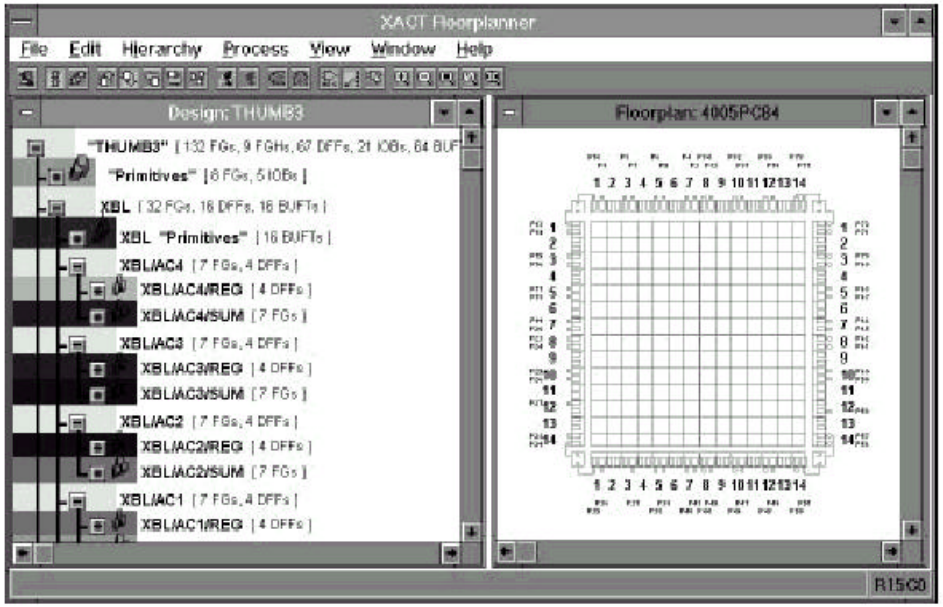


FIGURA 8: Floorplanner window (Altera).