UNIVERSITATEA TEHNICĂ din CLUJ-NAPOCA
FACULTATEA de AUTOMATICĂ şi CALCULATOARE
CATEDRA de CALCULATOARE

Sistem CAD pentru sinteza sistemelor numerice

Referat de doctorat

Conducător ştiinţific,
Prof. Dr. Ing. PUSZTAI Kalman

Doctorand,
ş.l. ing. BARUCH Zoltan-Francisc

Cuprins

1. Metodologii de proiectare

În această secțiune, se vor prezenta conceptele de bază legate de metodologiile de proiectare pentru sinteza la nivel de sistem şi la nivel de cip. *Sinteza la nivel de sistem* porneşte de la o descriere funcțională a sistemului complet şi generează o altă descriere funcțională a aceluiaşi sistem, în care fiecare cip este descris separat. *Sinteza la nivel de cip* efectuează conversia descrierii funcționale a fiecărui cip într-o descriere structurală care utilizează componente la nivelul transferurilor între registre (RT). Lucrările legate de metodologiile de proiectare sunt rare, şi sistemele de sinteză de nivel înalt nu sunt utilizate pe scară largă în practică. Se vor prezenta în continuare cerințele pentru sistemele de sinteză de nivel înalt şi se vor discuta soluții posibile de realizare a acestor sisteme.

O metodologie de proiectare trebuie să specifice următoarele elemente:

(a) sintaxa şi semantica descrierii utilizate ca intrare şi a descrierii generate la ieşire;

(b) setul de algoritmi utilizați pentru translatarea descrierii de intrare în descrierea de ieşire;

(c) setul de componente care se vor utiliza pentru implementare;

(d) definiția şi domeniul restricțiilor de proiectare;

(e) metoda de selecție a stilurilor de proiectare, a arhitecturilor, topologiilor şi componentelor;

(f) strategiile de control (numite de obicei scenarii) care definesc diferitele etape ale sintezei şi ordinea în care operațiile respective sunt executate.

De obicei, cerințele (a) şi (b) sunt definite prin alegerea unui limbaj de descriere şi a unui set de utilitare pentru sinteză, iar cerințele (c) şi (d) reprezintă consecințele limbajului şi algoritmilor care s-au ales. De cele mai multe ori, cerințele (e) şi (f) nu sunt definite, şi se presupune că acestea reprezintă responsabilitatea proiectantului. Se va explica în continuare, pe baza unui exemplu simplu, semnificația cerințelor (a) - (f) din punctul de vedere al metodologiei de proiectare în sinteza de nivel înalt, şi se vor deduce trei sisteme simple de sinteză care să satisfacă aceste cerințe.

Se consideră descrierea simplă a unei bucle infinite care calculează în fiecare iterație următoarea expresie:

$y = (a + b) * (c - d)$

De exemplu, algoritmii DSP pot fi descrişi prin bucle infinite, deoarece semnalele continue sunt eşantionate cu o rată constantă, pentru fiecare eşantion fiind executate ace-

leaşi prelucrări. Se va presupune că descrierea de intrare constă numai din asignări ale variabilelor utilizând operatori aritmetici. Această descriere este compilată într-un graf al fluxului de date (GFD) care identifică în mod explicit dependențele de date (**Figura 1(a)**). Se presupune că apar noi date în fiecare ciclu de ceas, şi fiecare iterație a buclei se execută în două cicluri de ceas. Se presupune de asemenea că fiecare operație este executată într-un ciclu de ceas de o unitate funcțională cu o singură funcție, toate unitățile funcționale şi registrele au acelaşi număr de biți, iar numărul de unități funcționale nu este limitat.

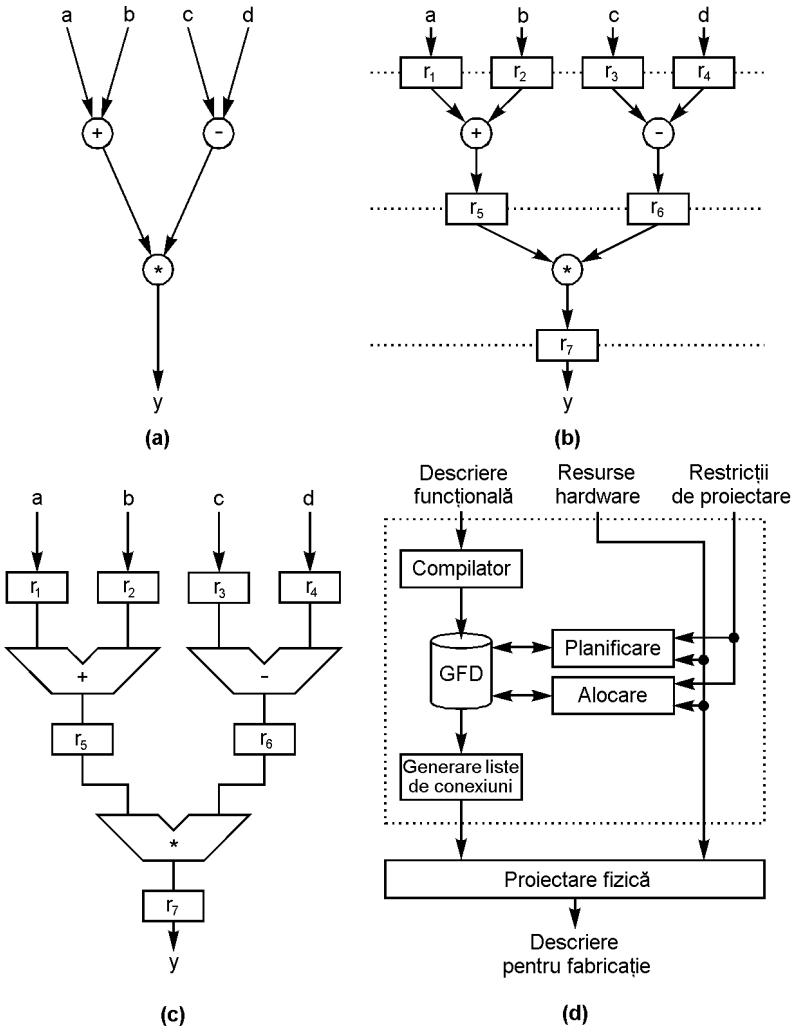


Figura 1. Sistem simplu de sinteză: (a) GFD; (b) GFD adnotat; (c) structura căii de date; (d) sistemul de sinteză.

Resursele hardware şi restricțiile de proiectare fiind specificate, se pot aplica algoritmii de planificare şi alocare. Aceşti algoritmi adnotează graful fluxului de date cu atribute suplimentare pentru a indica asignarea la stări şi componente. Fiecare operator este asignat la o stare de control şi la o unitate funcțională pentru a executa operația în ciclul de ceas dat. Fiecărei muchii din graful fluxului de date îi este asignat un latch, un registru sau o conexiune. Graful adnotat, indicat în **Figura 1(b)**, este utilizat pentru ge-

nerarea căii de date, prezentată în **Figura 1(c)**. De notat că graful adnotat şi calea de date obținută au aceeaşi topologie, ceea ce indică simplitatea arhitecturii destinație şi a algoritmilor de asignare.

Sistemul de sinteză pentru acest exemplu simplu (**Figura 1(d)**) va consta dintr-un compilator pentru conversia descrierii funcționale de intrare în reprezentarea GFD, un planificator pentru asignarea stărilor de control, un alocator pentru selecția şi asignarea unităților funcționale la operații şi a unităților de memorie la valori ale datelor, şi un generator al listelor de conexiuni pentru generarea listei de conexiuni a căii de date, care pot fi utilizate de alte sisteme. Lista de conexiuni a căii de date conține componente RT interconectate, şi anume un sumator, un scăzător, un circuit de înmulțire şi registre. Deoarece s-a presupus că fiecare unitate execută o singură funcție şi are acelaşi număr de biți, se poate presupune de asemenea că un singur proiectant va proiecta în prealabil toate unitățile funcționale şi de memorie pentru diferite tehnologii, acestea fiind memorate într-un set de fişiere accesibile de ceilalți proiectanți. Fiecare fişier va conține dimensiunile fizice ale unităților pentru amplasare şi poziția porturilor de I/E pentru rutare. Astfel, etapa de proiectare fizică va consta din executarea amplasării şi rutării pentru lista de conexiuni a căii de date, utilizând dimensiuni ale unităților şi locații ale porturilor predefinite.

Sistemul de sinteză prezentat în **Figura 1(d)** are numeroase limitări care îi restrâng utilitatea în situațiile practice. În primul rând, presupunerea că toate unitățile au acelaşi număr de biți şi aceeaşi întârziere de propagare este prea restrictivă. Un proiectant are de obicei posibilitatea de a alege între unități cu mod de funcționare, întârzieri de propagare şi număr de biți diferiți. În al doilea rând, stilul de proiectare de tipul fluxului de date, în care numărul unităților funcționale este egal cu numărul operatorilor din descrierea funcțională, este costisitor pentru proiectele obişnuite, cu excepția celor foarte simple. Pe de altă parte, o arhitectură destinație cu un număr foarte redus de unități funcționale necesită o unitate de control pentru a asigura secvențierea corectă a execuției operațiilor şi partajarea corespunzătoare a unităților funcționale. Pentru o asemenea arhitectură, trebuie să se utilizeze o metodologie de proiectare diferită. În al treilea rând, presupunerea că rata datelor de intrare şi de ieşire este aceeaşi cu rata de intrare şi de ieşire a căii de date, este de asemenea prea restrictivă. Datele pot proveni de la diferite surse, într-o ordine diferită şi momente de timp diferite, şi nu sunt sincronizate întotdeauna cu ceasul căii de date. Din acest motiv, sunt necesare cozi de intrare şi de ieşire sau memorii, pentru a sincroniza şirurile de date cu rate şi protocoale diferite.

În scopul prezentării unui sistem de sinteză pentru o clasă mai largă de arhitecturi, se va considera acelaşi exemplu, dar cu restricții de proiectare diferite. Se va impune ca operația $y = (a + b) * (c - d)$ să fie executată într-un timp de 300 ns, utilizând numai două unități funcționale, un circuit de înmulțire şi un sumator/scăzător, cu întârzieri de propagare mai mici de 100 ns fiecare.

Graful fluxului de date pentru calculul valorii y este indicat în **Figura 2(a)**. Graful poate fi planificat în trei stări de control, utilizând circuitele date. Deoarece se pot reutiliza registrele r_2 şi r_4, vor fi necesare în final numai cinci registre, în comparație cu şapte registre utilizate în **Figura 1(b)** şi **(c)**. Vor fi necesare în schimb patru multiplexoare 2:1. Graful adnotat este prezentat în **Figura 2(b)**, iar structura circuitului este prezentată în **Figura 2(c)**.

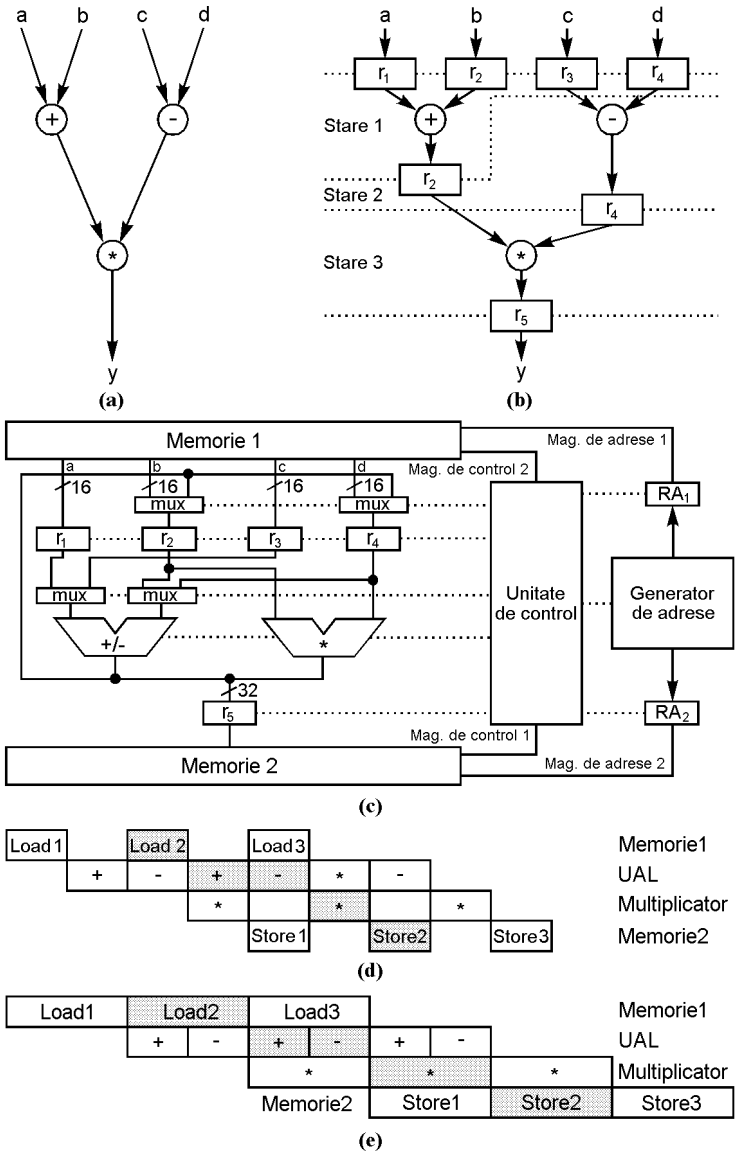


Figura 2. Un exemplu cu 3 operații: (a) GFD; (b) GFD adnotat; (c) implementarea ASFD; (d) utilizarea resurselor; (e) îmbunătățirea utilizării resurselor.

Pe lângă calea de date, circuitul final necesită o unitate de control pentru selecția şi încărcarea valorilor în registre şi pentru selecția operațiilor executate de unitățile funcționale. *Unitatea de control* are de asemenea rolul de a încărca datele din unitatea *Memorie1* şi de a memora datele în unitatea *Memorie2*, şi necesită o altă cale de date pentru generarea adreselor de încărcare şi memorare. Utilizarea resurselor este indicată în **Figura 2(d)**. Fiecare operație necesită pentru execuție un ciclu de ceas, inclusiv pentru încărcarea şi memorarea datelor, iar fiecare iterație necesită cinci cicluri de ceas. O nouă iterație este lansată la fiecare două cicluri de ceas. Astfel, sunt necesare $2(n\text{-}1)+5$ cicluri de ceas pentru a termina n iterații.

Se observă că unitatea aritmetică şi logică este o resursă critică, fiind utilizată în fiecare ciclu de ceas, în timp ce circuitul de înmulţire şi fiecare din cele două memorii sunt utilizate numai într-unul din două cicluri de ceas. Se poate realiza un compromis între gradul de utilizare al unor componente şi spaţiul de pe cip. De exemplu, se poate înlocui circuitul de înmulţire cu unul mai lent, care necesită două cicluri de ceas, dar un spaţiu mai redus. Similar, se pot utiliza memorii mai lente, sau acestea se pot înlocui cu memorii având un număr mai mic de porturi sau un număr mai mic de biţi pe cuvânt. De exemplu, unitatea *Memorie1*, cu patru porturi de câte 16 biţi, poate fi înlocuită cu o memorie cu două porturi de câte 16 biţi. Deci, vor fi necesare două cicluri de ceas pentru a furniza patru operanzi de câte 16 biţi. Se poate utiliza un singur port de 16 biţi pentru unitatea *Memorie2*, fiind necesare două cicluri de ceas pentru memorarea rezultatului de 32 biţi. Astfel, dacă se înlocuieşte circuitul de înmulţire şi ambele memorii cu componente care necesită două cicluri de ceas în loc de un ciclu, toate componentele vor fi utilizate complet (**Figura 2(e)**). Timpul total de execuţie a buclei va creşte numai cu trei cicluri de ceas, fiind de 2(*n*-1)+8 cicluri de ceas pentru *n* iteraţii.

Exemplul precedent demonstrează faptul că sinteza de nivel înalt generează o descriere structurală care conţine componente ca unităţi aritmetice şi logice, circuite de înmulţire, multiplexoare, registre, memorii şi controlere. Pentru fiecare din aceste componente, trebuie să se realizeze sinteza utilizând tehnici de sinteză logică sau secvenţială pe baza descrierii componentelor memorate într-o bază de date a componentelor (BDC). Această bază de date poate conţine de asemenea dimensiunea fiecărei componente necesară pentru operaţia de amplasare, şi poziţia fiecărui pin al componentelor necesară pentru rutarea în cadrul proiectării fizice. **Figura 3** prezintă o schemă-bloc a sistemului de sinteză care conţine o bază de date a componentelor.

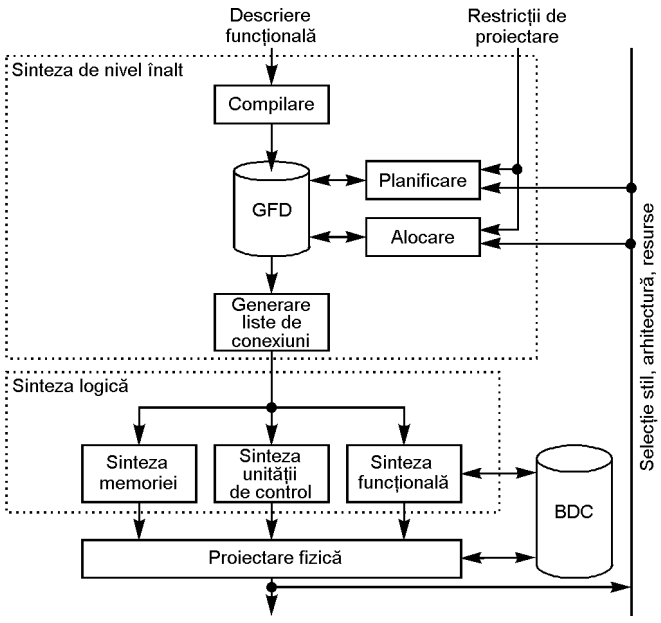


Figura 3. Sistem de sinteză cu o bază de date a componentelor.

Un proiect poate fi îmbunătăţit în mod iterativ prin alegerea unei alte arhitecturi destinaţie, a unei alte topologii sau a unor caracteristici diferite ale componentelor. În ultimul exemplu, îmbunătăţirea s-a realizat prin creşterea gradului de utilizare a unor componente. Proiectantul estimează de obicei calitatea proiectului şi efectuează modificări ale stilului de proiectare şi ale restricţiilor de proiectare pentru a îmbunătăţi proiectul în mod iterativ. Această operaţie poate fi automatizată, prin definirea unor indicatori de calitate şi dezvoltarea unei proceduri pentru estimarea calităţii, care realizează selecţia caracteristicilor proiectului (**Figura 4**).

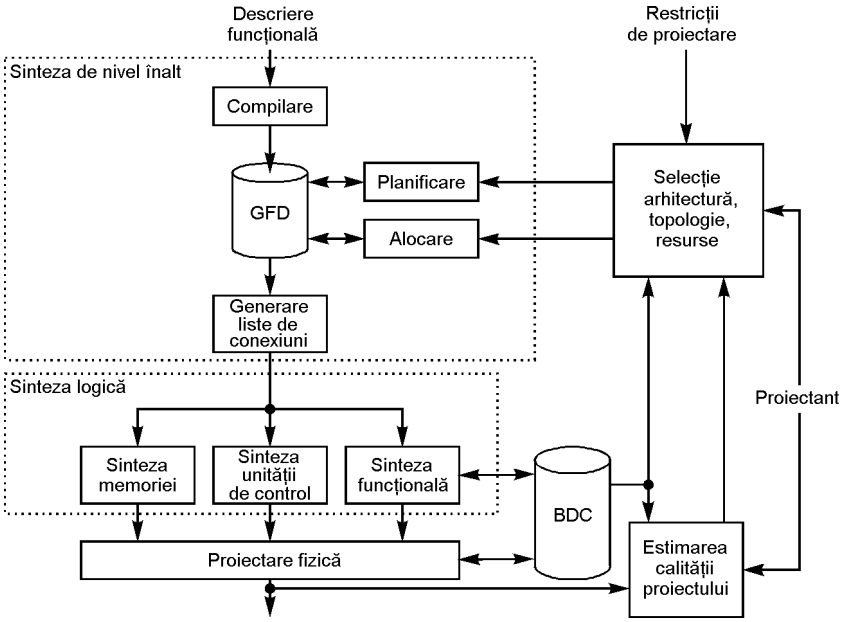


Figura 4. Sistem de sinteză utilizând îmbunătăţirea iterativă.

2. Sisteme de sinteză

2.1. Sistem generic de sinteză

În această secțiune se vor formula cerințele pentru procesul de proiectare şi se va descrie un sistem generic de sinteză ipotetic. Acest sistem este generic, deoarece reuneşte toate conceptele principale ale sintezei de nivel înalt. Sistemul este ipotetic, deoarece niciunul din sistemele de sinteză comerciale sau de cercetare nu satisfac toate criteriile prezentate în continuare:

(a) *Completitudine*. Sistemul trebuie să pună la dispoziție utilitare de sinteză pentru toate nivelele procesului de proiectare, de la specificație şi până la documentația de fabricație. Utilitarele de sinteză trebuie să se poată adapta la diferitele arhitecturi destinație.

(b) *Extensibilitate*. Sistemul trebuie să fie suficient de flexibil pentru a permite adăugarea unor noi algoritmi şi utilitare. Similar, sistemul trebuie să permită adăugarea unor noi stiluri arhitecturale, dacă este posibil, numai prin rescrierea strategiilor procesului de proiectare (a scenariilor). Sistemul trebuie să permită de asemenea adăugarea unor noi componente şi a unor algoritmi pentru adaptarea acestor componente la diferite tehnologii.

(c) *Controlabilitate*. Un proiectant trebuie să aibă posibilitatea de a controla tipul utilitarelor care se vor utiliza pentru o anumită descriere şi ordinea în care acestea vor fi executate. Acelaşi proiectant trebuie să aibă posibilitatea de a controla explorarea spațiului de proiectare, prin selecția diferitelor stiluri arhitecturale şi a diferitelor componente. În scopul asistării proiectantului în cadrul procesului de selecție, sistemul trebuie să pună la dispoziție diferiți indicatori de estimare ai calității.

(d) *Interactivitate*. Proiectantul trebuie să aibă posibilitatea de a interacționa cu utilitarele de sinteză, prin specificarea parțială a structurii proiectului, sau prin modificarea proiectului după sinteză. De asemenea, proiectantul trebuie să poată modifica fiecare asignare a obiectelor funcționale la obiectele structurale.

(e) *Îmbunătățirea metodologiei de proiectare*. Sistemul de sinteză trebuie să permită îmbunătățirea calității prin trecerea de la metodologia de proiectare bazată pe scheme electice şi simulare la metodologia bazată pe descriere şi sinteză, şi să permită posibilitatea utilizării ambelor strategii la fiecare nivel de abstractizare.

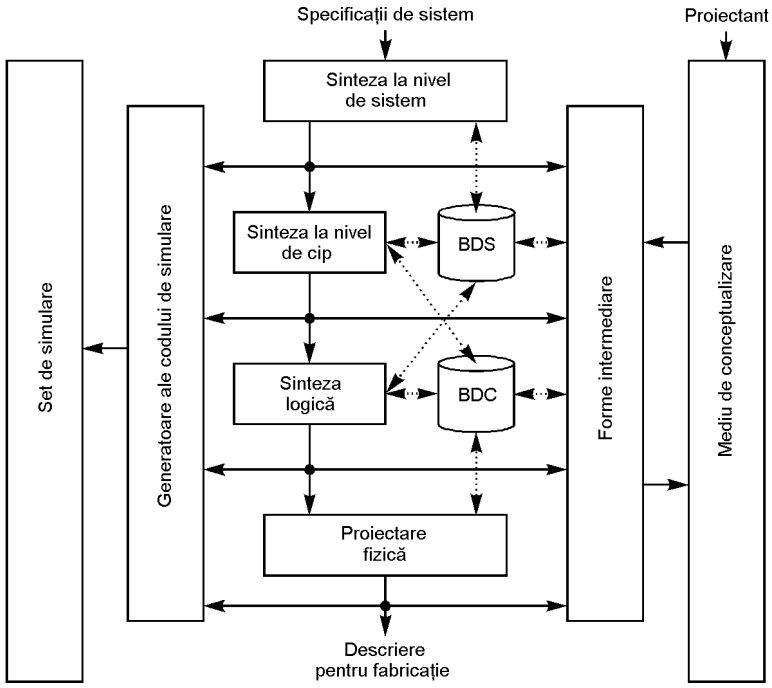


Figura 5. Un sistem de sinteză generic.

Schema-bloc a unui sistem de sinteză generic este prezentată în **Figura 5**. Sistemul dispune de un *Mediu de conceptualizare* pentru a permite interactivitatea şi controlabilitatea. Completitudinea sistemului este dată de faptul că acesta permite sinteza la nivel de sistem, de circuit, la nivel logic şi fizic. Extensibilitatea este asigurată prin existenţa a două baze de date. *Baza de date a componentelor* (*BDC*) permite adăugarea a unor noi componente pentru sinteză, în timp ce *Baza de date a sistemului* (*BDS*) permite adăugarea unor noi stiluri de proiectare, prin faptul că utilizează o reprezentare cuprinzătoare a proiectului şi facilitează adăugarea unor noi algoritmi, punând la dispoziţie diferite tipuri de informaţii pentru diferite nivele de abstractizare.

Sistemul permite utilizarea atât a metodologiei de proiectare bazate pe scheme electrice şi simulare, cât şi a metodologiei bazate pe descriere şi sinteză. Prima metodologie poate fi utilizată prin realizarea schemelor cu ajutorul *Mediului de conceptualizare* şi simularea acesteia printr-una din simulatoarele *Setului de simulare*. A doua metodologie poate fi utilizată prin reprezentarea descrierii proiectului într-una sau mai multe forme intermediare şi sinteza cu utilitarele corespunzătoare de sinteză.

Se utilizează diferite *Forme intermediare* pentru diferite aspecte de proiectare la acelaşi nivel sau pentru acelaşi aspect de proiectare la diferite nivele ale proiectării. În locul mai multor forme intermediare, s-ar putea defini un limbaj universal pentru toate nivelele şi stilurile de proiectare, dar un asemenea limbaj ar fi greoi şi ineficient. Ar fi necesar un timp considerabil pentru a se ajunge la un standard pentru un asemenea limbaj. Se poate utiliza în schimb un limbaj standard de simulare, cum este limbajul *VHDL*. Totuşi, un asemenea limbaj dispune mai ales de construcţii necesare pentru simulare şi

nu pentru sinteză. Mai mult, construcţiile acestui limbaj nu sunt adecvate pentru modelarea diferitelor stiluri de proiectare, şi proiectanţii nu sunt interesaţi de obicei în descrierea proiectelor în acest limbaj, ci în determinarea valorilor de ieşire pentru un set de vectori de test şi în observarea relaţiilor în timp între semnalele selectate. Deci, un raport de ieşire generat de un simulator va fi satisfăcător indiferent de sursa intrării pentru simulator, fie un alt limbaj, fie o altă formă de specificare a proiectului. De aceea, se poate utiliza orice formă intermediară pentru reprezentarea proiectului, cât timp există un generator al codului de simulare pentru translatarea formei intermediare într-o descriere care se poate utiliza pentru simulare. *Generatoarele codului de simulare* din **Figura 5** sunt utilizate în acest scop.

Generatoarele de cod se pot utiliza de asemenea în cadrul metodologiei bazate pe scheme şi simulare. De exemplu, o schemă poate fi considerată ca o formă intermediară care este translatată de către un generator al listelor de conexiuni într-o descriere utilizabilă pentru simulare. Similar, formele intermediare de la nivelul logic, secvenţial, RT şi sistem pot fi translatate în limbajul *VHDL* sau într-un alt limbaj de simulare. Formele intermediare pot fi utilizate pentru sinteza interactivă, pentru a realiza partiţionarea, planificarea şi asignarea manuală, ca şi transformarea şi verificarea proiectelor şi a descrierilor. Ele se pot utiliza de asemenea pentru estimarea calităţii proiectelor şi pentru estimarea efectelor modificării proiectelor asupra calităţii acestora.

2.2. Sinteza la nivel de sistem

Un sistem poate fi descris printr-un set de procese comunicante controlate de evenimente externe sau interne. Scopul sintezei la nivel de sistem este de a reprezenta o asemenea descriere a sistemului, de a evalua anumite caracteristici de bază ale acesteia, şi de a partiţiona descrierea într-o ierarhie de descrieri care reflectă entităţi fizice ca plăci, module multi-cip, cipuri sau macrocelule utilizate pentru fabricaţia sistemului.

Partiţionarea descrierii unui sistem în entităţi pentru care se poate realiza fabricaţia sau sinteza este o sarcină dificilă, în special dacă anumite entităţi pot fi asignate unor componente standard (ca procesoare şi memorii), în timp ce altele nu sunt definite complet sau sunt definite la un nivel de abstractizare diferit. Limbajul de descriere al sistemului trebuie să aibă posibilitatea specificării ierarhiei funcţionale şi structurale, să dispună de construcţii abstracte de comunicaţie şi de un stil de descriere care este păstrat pe parcursul mai multor nivele de partiţionare. Limbajul trebuie să permită reprezentarea informaţiilor structurale parţiale, ca cipuri, porturi şi conexiuni. Trebuie să permită de asemenea specificarea parţială a sistemului, astfel încât sinteza să poată fi realizată fără cunoaşterea detaliată a unor porţiuni ale sistemului. Descrierea trebuie să fie executabilă pentru a permite verificarea înainte şi după partiţionare.

O metodologie tipică pentru sinteza la nivel de sistem este prezentată în **Figura 6**. Descrierea de intrare este compilată într-o *Reprezentare a sistemului* (*RS*), utilizată de toate celelalte utilitare. Codul pentru simulare este generat de asemenea din această reprezentare. Descrierea sistemului este partiţionată în subdescrieri care se vor executa secvenţial pe anumite circuite sau vor exista concurent în acelaşi pachet. Două tipuri de obiecte sunt supuse partiţionării: structurile de date şi descrierile funcţionale.

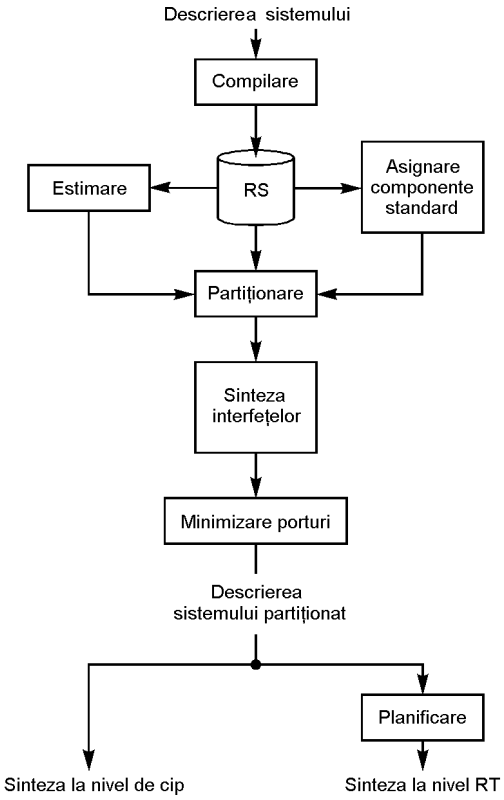


Figura 6. Metodologia de sinteză la nivel de sistem.

Structurile de date cuprind variabile globale şi locale, tablouri, liste, cozi şi înregistrări. Acestea sunt implementate de obici prin componente de memorie. De aceea, trebuie grupate structurile de date care pot fi implementate printr-o singură memorie, iar apoi trebuie partiționate acele memorii între cipuri sau alte pachete fizice.

Similar, descrierile funcționale pot fi partiționate între diferite unități de execuție sau procesoare. Se poate presupune că descrierea de intrare este bazată pe stări, tranzițiile între stări fiind determinate de evenimente, în fiecare stare fiind executate anumite prelucrări, cuprinzând instrucțiuni de asignare, funcții, proceduri sau procese. Partiționarea descrierilor funcționale poate fi considerată ca partiționare a stărilor. Dacă descrierea nu este bazată pe stări, aceasta se poate diviza în porțiuni mai mici prin partiționarea între limitele blocurilor, proceselor şi a procedurilor. Stările ale căror descrieri funcționale sunt executate secvențial pot fi grupate pentru execuție pe o singură arhitectură de tip ASFD, care poate fi apoi partiționată între pachetele fizice.

Pentru a evalua calitatea fiecărei partiții trebuie să se estimeze perioada ceasului, performanțele, spațiul ocupat, puterea consumată, costurile de testare şi încapsulare. Deoarece timpul total de execuție depinde de datele de intrare, estimarea perioadei ceasului şi a performanțelor pe baza descrierii funcționale este dificilă, deoarece datele nu sunt disponibile înaintea execuției. Algoritmii de estimare trebuie să utilizeze tehnici probabilistice pentru a defini numărul de parcurgeri ale fiecărei căi din descriere.

Dacă pentru implementare se vor utiliza componente standard, ca procesoare sau controlere, operația de partiționare devine foarte dificilă. Algoritmii de partiționare trebuie să fie capabili să recunoască părți ale descrierii care pot fi implementate prin componente standard. Astfel, partiționarea trebuie să fie precedată de o prepartiționare a descrierii în părți care sunt executabile sau neexecutabile pe anumite componente standard date.

Orice partiționare poate modifica unele caracteristici ale sistemului, în particular performanțele. De exemplu, mutarea locațiilor alocate pentru o structură de date de la un cip care utilizează acea structură de date într-un alt cip poate satisface cerințele de încapsulare, dar poate determina creşterea timpului de acces la acea structură de date. De asemenea, unirea mai multor structuri de date poate necesita modificarea protocolului de acces la acestea. Similar, unirea structurilor de date accesate de mai multe stări concurente necesită introducerea unor mecanisme de arbitraj. De aceea, sinteza interfețelor şi a logicii de arbitraj trebuie să fie realizată după fiecare iterație a partiționării.

După ce descrierea sistemului este partiționată în entități care satisfac restricțiile fizice, canalele de comunicație şi protocoalele lor pot fi unite pentru a minimiza numărul de porturi din fiecare partiție. Unirea canalelor poate fi considerată ca o multiplexare a mesajelor prin aceeaşi magistrală. Evident, această multiplexare este posibilă numai dacă două mesaje nu trebuie transmise concurent.

Sinteza la nivel de sistem are ca rezultat o structură de subdescrieri funcționale, fiecare din acestea definind un pachet sau modul a cărui sinteză trebuie realizată ulterior. Această descriere funcțională a fiecărui modul poate fi utilizată ca intrare pentru un utilitar de sinteză la nivel de cip, sau poate fi descompusă în continuare în cicluri de ceas prin planificarea operațiilor şi transmisă direct utilitarelor de sinteză la nivel RT.

2.3. Sinteza la nivel de cip

Sinteza la nivel de sistem produce o descriere la nivel de cip sub forma unor subdescrieri funcționale care pot fi implementate prin arhitecturi ASFD care comunică între ele. Aceste automate pot fi utilizate pentru a descrie procesoare, memorii cache, cozi, memorii, circuite de arbitraj a magistralelor şi interfețe. Sinteza la nivel de cip are ca scop transformarea unei descrieri funcționale ASFD într-o descriere structurală care utilizează componente RT. Un sistem generic de sinteză la nivel de cip este prezentat în **Figura 7**.

Sistemul de sinteză constă dintr-un compilator care generează o reprezentare internă, un set de utilitare pentru sinteza de nivel înalt, o bază de date a componentelor RT, un utilitar pentru adaptarea tehnologică şi un utilitar pentru optimizarea microarhitecturii. Descrierea funcțională de intrare este compilată într-o reprezentare internă (de exemplu, un graf al fluxului de control şi de date), care pune în evidență dependențele de control şi de date necesare pentru planificare şi alocare. Mai multe utilitare ale sintezei de nivel înalt adnotează această reprezentare cu informații necesare pentru sinteza logică şi secvențială.

Utilitarul de *Planificare* asignează operații stărilor de control. Utilitarele de *Alocare* a *unităților funcționale*, *de memorie* şi *de interconectare* alocă operatori unităților funcționale, variabile scalare şi tablouri registrelor şi memoriilor, şi conexiuni magis-

tralelor. Utilitarul de *Grupare a memoriilor* uneşte registrele în tablouri de registre, şi tablourile de registre în memorii. Acest utilitar determină de asemenea locaţia fiecărei variabile în cadrul memoriei grupate. *Selectorul de module* selectează componentele RT care vor fi utilizate pentru implementare. Acest utilitar poate partiţiona graful fluxului de control şi de date şi poate executa o amplasare preliminară pentru a determina combinaţia de componente RT care vor satisface restricţiile date.

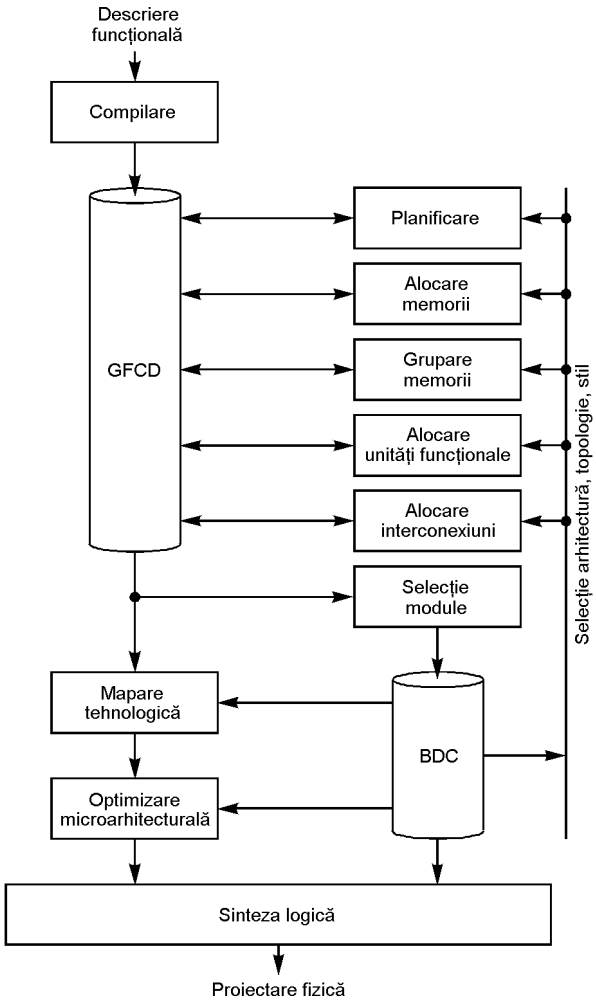


Figura 7. Sinteza la nivel de cip.

Baza de date a componentelor (*BDC*) memorează componentele RT care vor fi utilizate pentru sinteză şi furnizează informaţii despre caracteristicile acestora. Utilitarul de *Adaptare tehnologică* mapează componentele generice din descrierea pentru care se realizează sinteza la instanţieri ale componentelor memorate în *BDC*. Căile critice estimate şi întârzierile acestor căi se pot modifica după mapare. Utilitarul de *Optimizare microarhitecturală* elimină unele căi critice şi reduce întârzierile prin redistribuirea

componentelor, inserând componente mai rapide pe căile critice şi componente mai lente pe căile care nu sunt critice.

Prezența şi funcțiile ultimelor patru componente ale sistemului (*BDC*, *Selectorul de module*, utilitarele de *Adaptare tehnologică* şi *Optimizare microarhitecturală*) depind de metodologia de proiectare utilizată pentru proiectarea la nivel de cip. Se vor descrie trei metodologii diferite: "top-down", "meet-in-the-middle" şi "bottom-up".

Metodologia "top-down" porneşte de la componente RT generice presupunând anumite întârzieri nominale. Deoarece planificarea şi alocarea utilizează acele componente generice, ele vor apare în descrierea structurală rezultată. Optimizarea structurii RT este executată la nivelul logic. Înaintea executării optimizării logice, componentele RT sunt expandate în listele de conexiuni corespunzătoare la nivel de porți, obținute din *BDC*. Astfel, în cadrul acestei metodologii adaptarea tehnologică constă din expandarea definiției componentelor, în timp ce optimizarea logică este utilizată pentru finalizarea proiectării. Această metodologie este utilizată de numeroase sisteme de sinteză actuale, fiind utilizabilă pentru cipurile implementate cu rețele de porți şi celule standard.

Metodologia "meet-in-the-middle" porneşte atât de sus în jos, cât şi de jos în sus. Aceasta presupune că în prealabil componentele RT au fost proiectate manual sau că există generatoare de module pentru generarea planului de amplasare al componentelor RT. Componentele proiectate în prealabil şi generatoarele de module sunt memorate în *BDC*. Utilitarele sintezei de nivel înalt utilizează componente generice pentru sinteză. Adaptarea tehnologică pentru structura RT rezultată este realizată prin înlocuirea unei componente sau a unui grup de componente RT generice prin componente dependente de tehnologie din *BDC*. Din cauza diferențelor dintre componentele generice şi cele reale, optimizarea microarhitecturală este executată după adaptarea tehnologică. Această optimizare încearcă să reducă întârzierile de propagare pe căile critice şi să reducă spațiul ocupat pe căile care nu sunt critice. Adaptarea tehnologică şi optimizarea microarhitecturală sunt similare cu cele executate de utilitarele de sinteză logică. Această strategie este adecvată pentru proiectele specifice aplicațiilor, cu componente complexe ca circuite de înmulțire, unități de execuție şi memorii.

Metodologia "bottom-up" presupune de asemenea că în *BDC* sunt disponibile componente RT proiectate în prealabil. În plus, se presupune că *BDC* poate fi interogată despre disponibilitatea acelor componente şi caracteristicile lor. Algoritmii sintezei de nivel înalt trebuie modificați pentru a lucra cu o varietate de componente reale şi pentru a le selecta pe cele corespunzătoare. Cu alte cuvinte, selecția modulelor este executată de către utilitarele de planificare şi de alocare. În cazul celorlalte metodologii, selecția modulelor este executată înaintea planificării şi alocării. Mai mult, în cazul acestei strategii nu există adaptare tehnologică sau optimizare microarhitecturală. Singura operație executată după sinteza de nivel înalt este expandarea componentelor RT în componente la nivelul logic pentru implementarea prin rețele de porți sau celule standard.

Cele trei metodologii descrise sunt aplicabile de asemenea pentru cipurile care constau din mai multe automate ASFD. Unele din acestea, ca memoriile şi circuitele de înmulțire, sunt considerate ca automate ASFD standard sau macrocelule. Sinteza acestor macrocelule este realizată în timpul etapei de sinteză fizică.

2.4. Sinteza logică şi secvențială

Ieşirea sintezei de nivel înalt este o descriere structurală în care fiecare componentă este un automat ASFD descris printr-o tabelă de stări, o componentă a căii de date descrisă prin ecuații booleene, o componentă de interfață descrisă prin diagrame de timp sau forme de undă, sau o memorie specificată prin dimensiune, numărul şi tipul porturilor, şi protocolul de acces. Fiecare din aceste componente necesită tehnici diferite de sinteză, după cum se indică în **Figura 8** pentru un sistem ipotetic de sinteză logică.

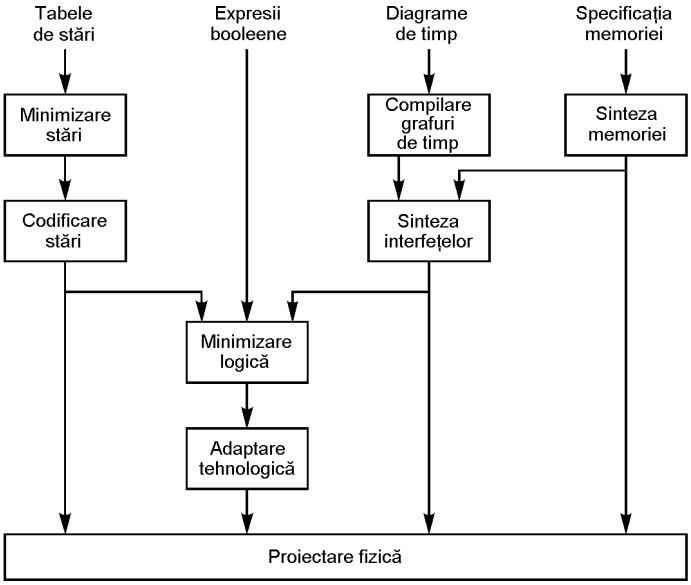


Figura 8. Sistem de sinteză logică.

O arhitectură ASFD constă dintr-o cale de date şi o unitate de control a cărei funcționare poate fi modelată ca un automat determinist cu stări finite. Un asemenea automat conține un registru de stare pentru memorarea stării prezente şi un circuit combinațional pentru evaluarea stării următoare şi a funcțiilor de ieşire. Un utilitar de sinteză a unității de control trebuie să translateze o asemenea descriere într-o structură hardware, executând minimizarea stărilor, codificarea stărilor, minimizarea logică şi adaptarea tehnologică.

Minimizarea stărilor reduce numărul stărilor unui automat cu stări finite prin eliminarea stărilor redundante, ale căror funcții pot fi executate de alte stări. Procedurile de minimizare a stărilor partiționează stările în clase de stări echivalente. Două stări s_i şi s_j sunt echivalente dacă nu există secvență de intrare care generează două secvențe diferite de ieşire, atunci când aceasta este aplicată automatului în stările s_i şi s_j. Deoarece toate stările echivalente generează aceleaşi ieşiri, fiecare clasă echivalentă poate fi înlocuită printr-o singură stare. Minimizarea stărilor este importantă deoarece numărul stă-

rilor determină dimensiunea registrului de stare şi a logicii de control. Minimizarea stărilor îmbunătăţeşte de asemenea testabilitatea automatelor cu stări finite, deoarece automatele fără stări redundante sunt mai uşor de testat.

Codificarea stărilor asignează coduri binare stărilor simbolice. Scopul codificării stărilor este de a minimiza logica necesară pentru implementarea automatului cu stări finite. Pentru satisfacerea acestui scop, se poate aplica implementarea prin două nivele logice sau implementarea printr-o logică multi-nivel.

În cazul implementării prin două nivele logice, se execută o minimizare logică cu valori multiple (simbolică) a logicii necodificate, prin gruparea setului de intrări care corespunde fiecărui simbol de ieşire. Această grupare stabileşte de asemenea un set de restricţii pentru codificarea fiecărui simbol. Simbolurile sunt codificare secvenţial utilizând codul cu lungimea minimă care satisface toate restricţiile.

În cazul implementării printr-o logică multi-nivel, procedura de codificare încearcă să minimizeze costul implementării prin minimizarea numărului de literale din expresiile booleene ale logicii stării următoare. Minimizarea numărului de literale este realizată prin maximizarea numărului şi dimensiunii subexpresiilor comune, ceea ce se realizează prin asignarea codurilor similare stărilor care generează ieşiri similare.

După codificarea tuturor intrărilor, ieşirilor şi stărilor simbolice, prin minimizarea logică se reduce costul implementării logicii stării următoare şi a logicii de control. Diferite implementări necesită concentrarea asupra unor obiective diferite. Pentru logica cu două nivele, obiectivul este de a se minimiza numărul termenilor produs ale ecuaţiilor booleene scrise sub forma sumelor de produse. Pentru logica multi-nivel, obiectivul este de a se minimiza numărul de literale din ecuaţiile booleene prin aplicarea mai multor tehnici.

Operaţia de *extragere* identifică divizorii comuni din expresiile booleene care se pot utiliza pentru a reduce numărul total de literale. Operaţia de *resubstituire* testează dacă o funcţie existentă este un divizor al altor funcţii. Operaţia de *asignare a fazei* realizează alegerea între implementarea funcţiei sau a complementului acesteia, în scopul minimizării numărului total de inversoare necesare pentru implementare. Operaţia de *factorizare a porţilor* factorizează ecuaţia logică pentru fiecare poartă complexă pentru a produce o reţea optimă pentru poarta respectivă. *Descompunerea porţilor* împarte funcţiile complexe în părţi mai simple. Prin *simplificare* se aplică minimizarea cu două nivele fiecărui nod din reţea.

Operaţiile logice menţionate sunt executate în ordinea specificată într-un scenariu, eficienţa acestora depinzând în mare măsură de caracteristicile logicii de intrare. Interacţiunea utilizatorului şi un scenariu cu autoadaptare pot fi utilizate pentru a creşte calitatea minimizării.

În timp ce logica cu două nivele este implementată de obicei printr-un circuit PLA, logica multi-nivel poate fi implementată prin biblioteci de celule standard. Adaptarea tehnologică transformă o reţea logică independentă de tehnologie generată prin minimizarea logică într-o listă de conexiuni a unor celule standard aflate într-o anumită bibliotecă. Dintre numeroasele metode care pot fi aplicate pentru adaptarea tehnologică se aminteşte partiţionarea reţelei logice într-o pădure de arbori, şi determinarea pentru fiecare arbore a unei acoperiri prin celule de bibliotecă, cu un cost minim.

Deoarece se presupune că fiecare proiect poate fi descris printr-un set de procese comunicante şi poate fi implementat printr-un set de automate ASFD care comunică între ele, proiectarea logicii de comunicaţie este o parte necesară a sintezei. Circuitul de interfaţă poate fi implementat ca o parte integrantă a fiecărui automat ASFD, sau ca un automat independent. Dacă un protocol de comunicaţie este o parte a descrierii ASFD, fiecare răspuns la un semnal de protocol necesită un număr întreg de stări. De exemplu, protocolul de cerere-confirmare se poate implementa utilizând o stare de aşteptare şi o stare de confirmare. Activarea semnalului de cerere determină ieşirea automatului din starea de aşteptare şi trecerea acestuia în starea de confirmare, în care este activat semnalul de confirmare. La dezactivarea cererii, automatul va ieşi din starea de confirmare. Pe de altă parte, dacă trebuie să se realizeze interfaţarea între două automate ASFD standard cu protocoale diferite, trebuie să se realizeze sinteza unui al treilea automat pentru conversia între cele două protocoale.

Sinteza interfeţelor este dificilă din cauza lipsei limbajelor de descriere a interfeţelor şi a metodelor de sinteză pentru logica combinată secvenţială sincronă şi asincronă. Protocoalele de comunicaţie sunt descrise de obicei prin diagrame de timp cu restricţii impuse de temporizare. Există trei tipuri de restricţii de temporizare: de ordonare, de simultaneitate şi de sincronizare. O restricţie de ordonare defineşte ordinea în care apar evenimentele într-un anumit interval de timp. O restricţie de simultaneitate defineşte toate evenimentele care apar împreună, iar o restricţie de sincronizare defineşte timpul de setare şi menţinere pentru logica sincronă.

O asemenea descriere trebuie convertită într-un graf de temporizare, în care fiecare nod reprezintă un eveniment, deci o tranziţie logică a unui semnal. Arcele grafului corespund restricţiilor de temporizare care există între două evenimente. Aceste grafuri pot fi construite în mod direct pe baza diagramelor de timp. Sinteza interfeţelor constă în implementarea grafului de temporizare cu un număr minim de latch-uri şi bistabile şi minimizarea logicii pentru setarea şi resetarea acestora. Logica de interfaţă poate fi sincronă sau asincronă.

Sinteza memoriei generează o descriere a memoriei pentru anumite cerinţe date, ca numărul de cuvinte, numărul de biţi pe cuvânt, numărul de porturi, tipul porturilor (citire, scriere, sau citire/scriere), rata de transfer, protocolul de acces, şi restricţiile de temporizare. Sinteza memoriilor poate fi complexă pentru anumite cerinţe. De exemplu, proiectarea unei memorii cu 4 porturi utilizând numai circuite de memorie cu un singur port poate fi simplă dacă datele pot fi partiţionate în patru grupuri, fiecare grup fiind accesat numai printr-un port (de exemplu, *Memorie1* din **Figura 2(c)**). Dacă însă aceleaşi date sunt accesate prin două porturi diferite, sau dacă două porturi accesează datele din acelaşi circuit de memorie cu un singur port, trebuie să se adauge o logică de rezolvare a conflictelor. Această logică are ca rezultat creşterea timpului de acces al memoriei. Deci, scopul utilitarului de grupare a memoriilor este de a partiţiona datele şi de a asigna fiecare partiţie unei memorii astfel încât numărul de conflicte şi costul logicii de generare a adreselor să fie minimizat.

2.5. Proiectarea fizică

Metodologia de proiectare fizică prezentată în **Figura 9** porneşte de la o listă de conexiuni RT care conţine trei tipuri de componente, acestea fiind caracterizate prin numărul dimensiunilor în care ele prezintă o regularitate. Componentele cu dimensiunile cele mai mari sunt tablourile bidimensionale, ca memoriile, circuitele de înmulţire şi grupurile de registre. Componentele cu dimensiuni medii sunt tablourile unidimensionale ca registrele, numărătoarele, unităţile aritmetice şi logice, driverele de magistrală şi unităţile logice. Porţile logice, latch-urile şi bistabilele sunt componente de dimensiuni mici, fără o regularitate.

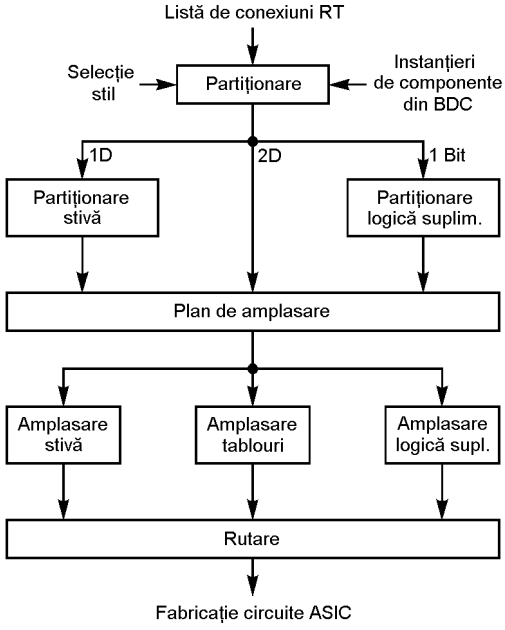


Figura 9. Metodologie de proiectare fizică.

Fiecare grup de componente are un stil de amplasare diferit. Componentele bidimensionale au de obicei dimensiuni mari şi constituie module separate. Componentele unidimensionale pot fi grupate împreună sub forma unei stive (**Figura 10(a)**), datele fiind rutate într-o direcţie şi semnalele de control în cealaltă direcţie, pe straturi metalice diferite. Ca un stil arhitectural eficient de amplasare, arhitectura de tip stivă reduce semnificativ spaţiul necesar şi lungimea conexiunilor. Totuşi, această amplasare poate avea ca rezultat o pierdere semnificativă de spaţiu datorită componentelor cu dimensiuni inegale. De exemplu, poate exista o unitate aritmetică şi logică de 16 biţi şi un numărător de 4 biţi, amplasate alăturat în stivă. În asemenea cazuri, spaţiul liber trebuie umplut prin amplasarea porţilor şi a bistabilelor, utilizând stilul de amplasare standard al celulelor, în care celulele sunt amplasate pe linii şi sunt conectate prin canale de rutare între linii. De exemplu, spaţiul liber de lângă stiva din **Figura 10(a)** este divizat în trei zone

dreptunghiulare. Două dintre aceste sunt umplute cu logica de control necesară componentelor din stivă, dar a treia este umplută cu componente care nu au legătură cu stiva.

După partiționarea componentelor în trei grupuri pe baza dimensiunii acestora, ele sunt partiționate în continuare pentru a satisface restricțiile spațiale impuse de planul de amplasare selectat (**Figura 10(b)**). Porțile şi bistabilele sunt partiționate pentru a umple spațiile dintre modulele de dimensiuni mari. Spațiul disponibil poate avea forma unui poligon arbitrar, şi astfel trebuie partiționat la rândul său în zone dreptunghiulare, cu capacități estimate ale tranzistoarelor. Porțile şi bistabilele sunt partiționate apoi în grupuri pentru care nu se depăşesc capacitățile tranzistoarelor zonelor care le sunt asignate. Similar, componentele unidimensionale pot fi partiționate în mai multe stive.

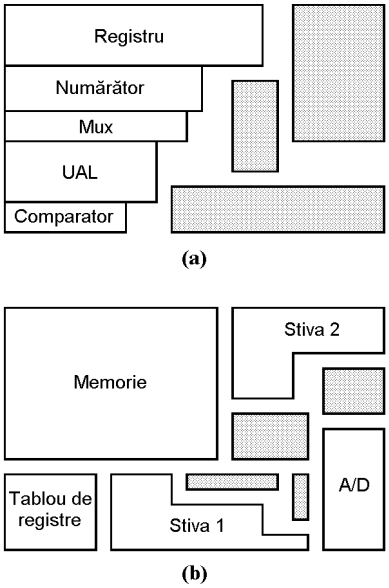


Figura 10. Planuri de amplasare generice.

Partiționarea şi amplasarea trebuie executate în mod iterativ, deoarece fiecare partiționare influențează amplasarea, şi invers, fiecare plan de amplasare influențează partiționarea. După terminarea amplasării, trebuie să se realizeze sinteza descrierii amplasării pentru fiecare componentă, utilizând una din cele trei arhitecturi de amplasare: tablou, stivă, sau celule standard. Operația finală din cadrul proiectării fizice este rutarea componentelor instanțiate.

2.6. Baza de date a sistemului

În mod tradițional, sistemele de sinteză sunt elaborate ca o colecție de utilitare de proiectare de sine stătătoare. În cazul acestor sisteme, există diferențe în ceea ce priveşte reprezentarea datelor, şi formatele de intrare şi de ieşire. A doua generație a unui asemenea sistem de sinteză poate fi dezvoltată fie ca un sistem puternic integrat, fie ca un sis-

tem integrat într-o măsură mai mică. Un sistem puternic integrat utilizează o reprezentare comună a datelor, fiecare utilitar utilizând aceleaşi proceduri pentru accesul la date. Un asemenea sistem este foarte eficient, dar extrem de rigid, deoarece o modificare a formatului influenţează toate utilitarele componente. Un sistem integrat într-o măsură mai mică separă datele de utilitare. Datele sunt memorate într-o bază de date şi fiecare utilitar face acces numai la informaţiile de care are nevoie, utilizând o reprezentare proprie pentru prelucrarea informaţiilor respective.

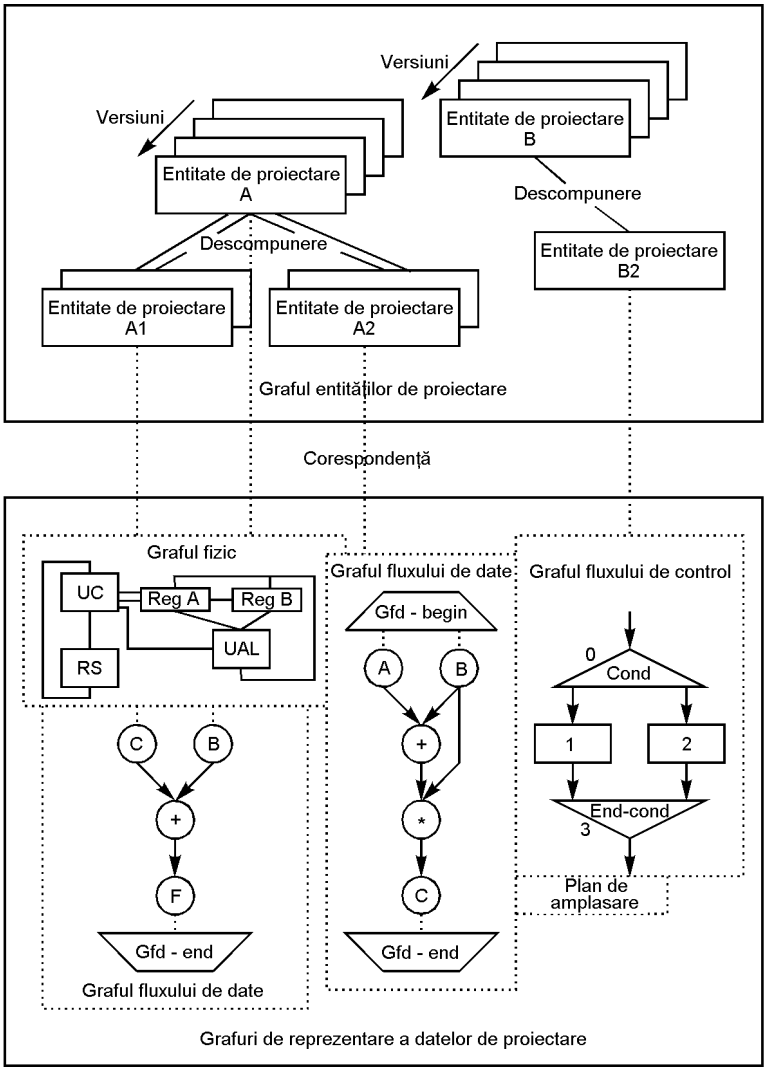


Figura 11. Model al datelor de proiectare.

O altă motivaţie pentru bazele de date este existenţa unor versiuni diferite ale aceluiaşi proiect în timpul explorării spaţiului de proiectare. În multe sisteme, proiectanţii sunt cei care gestionează ierarhia de proiectare, reviziile şi configuraţiile acesteia, deşi aceste sarcini pot fi asignate bazei de date. O bază de date poate memora şi gestiona

datele de proiectare utilizând un *model al datelor de proiectare* (MDP) care constă din două părți (**Figura 11**).

Primul nivel al MDP memorează organizarea administrativă globală a datelor de proiectare prin intermediul *Grafului entităților de proiectare*, care permite reprezentarea ierarhiei de proiectare, controlul versiunilor şi gestionarea configurațiilor. O entitate de proiectare este unitatea de bază a datelor care este gestionată de baza de date. Un mare număr de lucrări legate de bazele de date ale sistemelor CAD se concentrează asupra acestui nivel administrativ.

Al doilea nivel al MDP memorează datele de proiectare propriu-zise. Aceste date sunt modelate prin graful funcțional şi graful fizic. Modelul grafului funcțional este o reprezentare a descrierii ierarhice a funcționării la nivel de operații, la care se adaugă dependențe de control şi de date, restricții fizice şi de temporizare, şi asignări ale stărilor şi ale componentelor. Modelul grafului fizic este o reprezentare a structurii ierarhice a componentelor interconectate, la care se adaugă informații de temporizare şi de proiectare fizică.

După cum se indică în **Figura 11**, fiecare obiect din graful entităților de proiectare este o colecție de obiecte de proiectare asociate. Graful conține atribute specifice diferitelor domenii pentru estimarea calității datelor de proiectare. De exemplu, unele din atributele domeniului funcțional reprezintă numărul stărilor, cerințele de memorie şi de operatori, şi frecvența accesurilor la memorie, în timp ce atributele domeniului structural reprezintă spațiul estimat, întârzierea maximă şi numărul de componente. Fiecărui atribut îi este asociată o procedură specifică aplicației, care calculează automat valoarea atributului din datele de proiectare corespunzătoare. Aceasta asigură consistența între atributele păstrate în graful entităților de proiectare şi datele de proiectare din grafurile funcționale şi fizice.

O arhitectură tipică pentru o bază de date a sistemului (*BDS*) este prezentată în **Figura 12**. Utilitarul de *Vizualizare a schemelor* permite proiectantului să vizualizeze şi eventual să gestioneze entitățile de proiectare şi relațiile dintre ele. Utilitarul de *Gestiune a versiunilor* implementează procedura de derivare a versiunilor aleasă de proiectant, transmițând comenzi corespunzătoare utilitarului de *Gestiune a entităților de proiectare* pentru crearea şi gestionarea relațiilor între entitățile de proiectare. Utilitarul de *Gestiune a tranzacțiilor* controlează accesul la entitățile de proiectare, prevenind astfel accesul simultan la aceleaşi date de către mai multe utilitare de proiectare.

Pentru a permite utilizarea bazei de date de către utilitare din afara sistemului, *BDS* trebuie să pună la dispoziție interfețe adaptate pentru aceste utilitare. Aceste interfețe adaptate conțin un subset al informațiilor din MDP furnizate în formatul specificat de utilitarele respective. *BDS* pune la dispoziție un limbaj de descriere pentru specificarea acestor interfețe. Utilitarul de *Gestiune a interfețelor* compilează specificarea unei interfețe într-un set de comenzi pentru utilitarul de *Gestiune a datelor de proiectare* în scopul accesului la date. Această organizare asigură flexibilitate, deoarece pot fi create uşor noi interfețe adaptate, şi garantează extensibilitatea, deoarece pot fi adăugate noi informații la datele de proiectare fără a afecta interfețele existente.

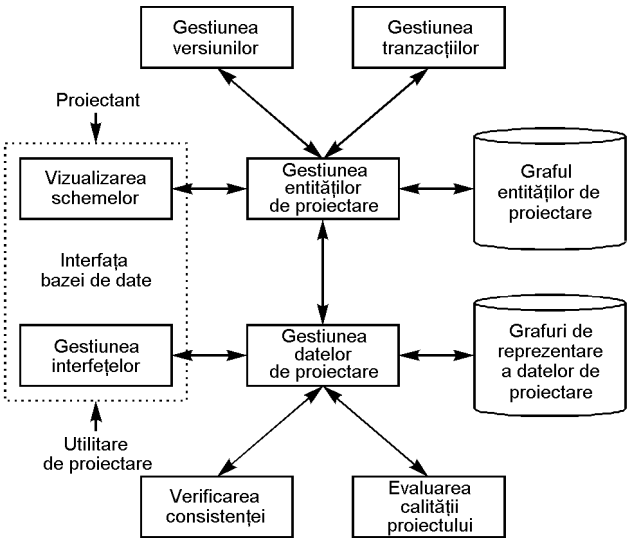


Figura 12. Arhitectura bazei de date a sistemului.

Verificarea consistenței este executată la două nivele diferite. În primul rând, utilitarul de *Verificare a consistenței* testează dacă utilitarele sunt apelate în ordinea corespunzătoare şi dacă datele solicitate de fiecare utilitar sunt disponibile în baza de date. În al doilea rând, verifică dacă datele noi adăugate sunt consistente cu datele originale. De exemplu, asignarea operatorilor la unitățile funcționale nu trebuie să modifice numărul paşilor de control, şi o modificare a numărului paşilor de control nu trebuie să modifice funcționarea inițială specificată. Utilitarul de *Evaluare a calității proiectului* pune la dispoziție indicatori de calitate sub forma unor atribute în graful entităților de proiectare după fiecare actualizare, astfel încât diferite versiuni ale proiectului pot fi regăsite pe baza unui indicator de calitate sau a unei combinații a unor asemenea indicatori.

2.7. Baza de date a componentelor

Utilitarele de sinteză funcțională generează un proiect microarhitectural din descrieri funcționale sau descrieri la nivelul transferurilor între registre. Microarhitectura generată constă din componente RT ca unități aritmetice şi logice, circuite de înmulțire, numărătoare, decodificatoare şi registre. Spre deosebire de componentele logice de bază, componentele RT au diferite opțiuni care pot fi parametrizate. Unul din parametri este dimensiunea componentei, ceea ce indică numărul de biți. Alți parametri sunt legați de aspectele funcționale sau proprietățile electrice şi geometrice ale componentei. De exemplu, numărătoarele pot avea opțiuni pentru incrementare sau decrementare, şi funcții de încărcare, setare şi resetare. De asemenea, fiecare componentă poate avea întârzieri diferite şi fiecare pin de ieşire poate fi încărcat în mod diferit. În plus, fiecare componentă poate avea mai multe opțiuni diferite pentru amplasare, ca şi pentru poziționarea porturilor de I/E în cadrul modulului.

Baza de date a componentelor (*BDC*) generează componente care satisfac cerinţele specifice de proiectare şi furnizează informaţii despre caracteristicile electrice şi de amplasare ale componentelor.

Deci, *BDC* trebuie să pună la dispoziţie două tipuri de informaţii: o estimare a caracteristicilor componentelor pentru fiecare tip de componentă disponibilă, şi o descriere a instanţierii componentelor pentru a fi inclusă în descrierile la nivel RT, logic şi la nivel de cip. Estimările caracteristicilor componentelor sunt utilizate de utilitarele sintezei de nivel înalt pentru planificare şi alocare, şi de utilitarele proiectării fizice pentru planul de amplasare. Descrierile componentelor sunt utilizate la trei nivele diferite. În primul rând, numele componentelor şi numele pinilor acestora sunt utilizate în descrierile RT generate de utilitarele sintezei de nivel înalt. În al doilea rând, descrierile de la nivelul logic sunt utilizate în timpul sintezei logice şi a optimizării. În al treilea rând, descrierile la nivel de cip sunt utilizate în timpul asamblării planurilor de amplasare. În plus faţă de descrierile componentelor, *BDC* trebuie să furnizeze trei modele diferite pentru simulare la nivel RT, la nivel logic şi la nivel de cip.

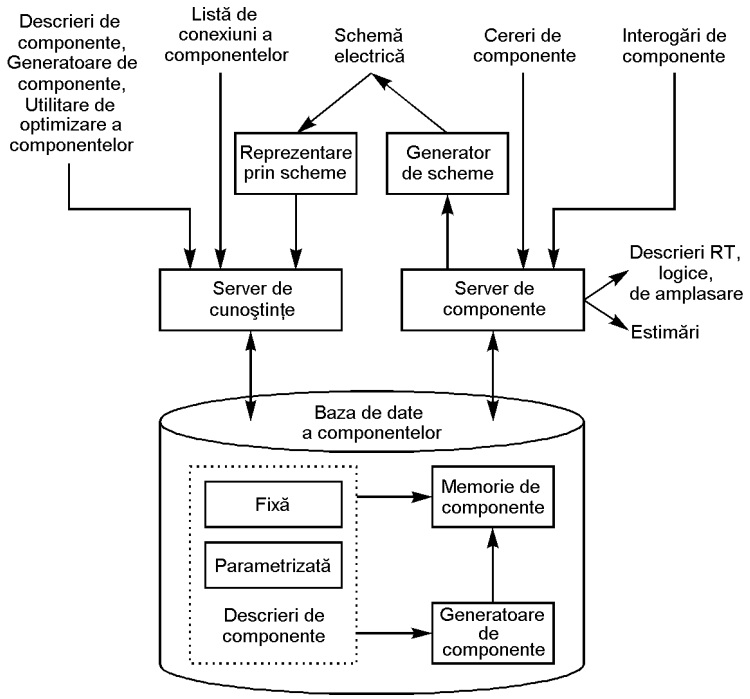


Figura 13. Baza de date a componentelor.

Există puţine lucrări legate de generarea componentelor şi bazele de date ale componentelor pentru sinteza de nivel înalt. Baza de date generică prezentată în **Figura 13** constă din două părţi: *Serverul de componente* şi *Serverul de cunoştinţe*. *Serverul de componente* furnizează estimări şi descrieri la nivel RT, la nivel logic şi la nivel de cip, atunci când indică răspunsuri la întrebări legate de instanţierile componentelor. *Serverul de cunoştinţe* permite adăugarea unor noi instanţieri de componente, a generatoarelor de

componente sau a utilitarelor de optimizare a componentelor. Acest server este utilizat pentru extinderea bibliotecilor de componente disponibile.

O componentă poate fi memorată ca o instanțiere atunci când este proiectată manual pentru o anumită implementare particulară, sau ca o descriere parametrizată pentru o clasă de instanțieri ale componentelor. *Generatoarele de componente* sunt seturi de utilitare care generează instanțieri de componente din descrieri parametrizate. Utilitarele de optimizare la nivel logic sau la nivel de cip pot fi utilizate ca o parte a procesului de generare a componentelor. Toate instanțierile proiectate manual sau generate sunt păstrate în *Memoria de componente* pentru a fi utilizate de un anumit proiectant sau un anumit utilitar de sinteză.

BDC este o componentă critică a metodologiei sintezei de nivel înalt, deoarece asigură o separare clară a sintezei la nivel de sistem şi la nivel de cip de sinteza logică şi fizică. Existența unei *BDC* permite ca proiectanții să se concentreze asupra proiectării sistemului, fără implicarea în proiectarea logică sau fizică. Permite de asemenea dezvoltarea rapidă a utilitarelor sintezei de nivel înalt, punând la dispoziție interfețe standard pentru sinteza componentelor şi izolarea față de modificările tehnologiilor de fabricație.

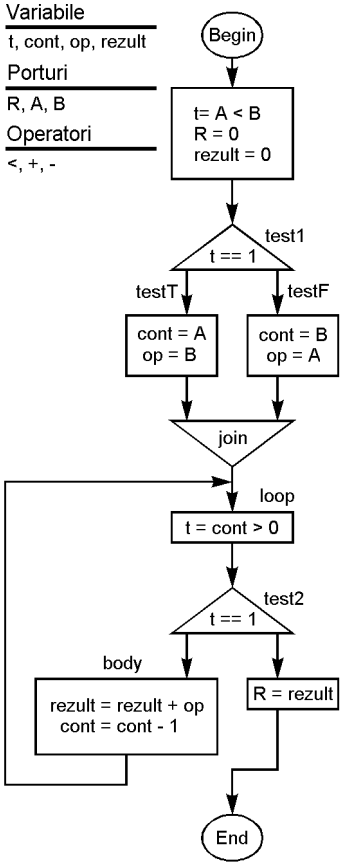
2.8. Mediul de conceptualizare

Automatizarea completă a procesului de proiectare pornind de la nivele mai înalte de descriere este un scop important, deşi nu este posibilă imediat. Chiar dacă toate utilitarele sintezei de nivel înalt ar fi disponibile în momentul de față, ar fi necesar ca proiectanții să devină familiarizați cu utilizarea lor. De asemenea, este necesar un timp pentru ca algoritmii de sinteză să asigure o calitate uniformă pentru diferite stiluri de proiectare. Astfel, este necesar să se asigure un mediu în care proiectanții pot verifica deciziile de proiectare şi pot modifica manual algoritmii de sinteză. Chiar dacă algoritmii sunt performanți, proiectanții trebuie să aibă posibilitatea de a lua decizii de nivel înalt, ca de exemplu selecția stilului de proiectare, a arhitecturii, a topologiei, a cerințelor şi a restricțiilor pentru componente. Deci, un mediu de conceptualizare trebuie să permită controlul deciziilor de proiectare şi a strategiei de proiectare în toate etapele procesului de proiectare, ceea ce necesită ca mediul să pună la dispoziție indicatori utilizabili de calitate la fiecare nivel de abstractizare.

Mediul de conceptualizare este strâns legat de baza de date a sistemului şi baza de date a componentelor. Aceste baze de date memorează toate versiunile datelor de proiectare pentru perioade lungi de timp, în timp ce mediul de conceptualizare face acces numai la porțiuni reduse ale datelor pentru modificare în timpul unor perioade scurte de timp.

Mediul de conceptualizare constă din cinci părți: gestiunea datelor şi a proiectului, vizualizarea şi editarea datelor, estimarea indicatorilor de calitate, algoritmii de sinteză şi verificarea consistenței proiectării. Calitatea proiectului este estimată pe baza datelor proiectului parțial sau complet. Indicatorii de calitate pot fi calculați relativ simplu pe baza proiectului final, dar trebuie să se utilizeze tehnici diferite de estimare la diferite nivele de abstractizare. Algoritmii de sinteză pot fi aplicați întregului proiect sau numai unor părți ale proiectului. Proiectantul trebuie să poată împărți sinteza în etape şi

să aplice algoritmii fiecărei etape separat. Prin verificarea consistenței proiectării se asigură faptul că modificările manuale ale proiectului nu produc nici o modificare în descrierea funcțională. De exemplu, dacă un transfer între registre de forma $y \leftarrow a+b+c$, care necesită două sumatoare, este împărțit de proiectant în două transferuri între registre, $y \leftarrow a+b$, $y \leftarrow y+c$, care necesită un singur sumator, dar două stări, prin verificarea consistenței trebuie să se asigure că cele două transferuri între registre generează același rezultat ca şi transferul original.



(a)

Stare	Condiție	Valoare Condiție	Acțiuni	Stare Urm.
Begin			t = (A<B), R = 0, rezult = 0	test1
test1	(t)	("1")		testT
		("0")		testF
testT			cont = A, op = B	join
testF			cont = B, op = A	join
join				loop
loop			t = (cont > 0)	test2
test2	(t)	("1")		body
		("0")		1
body			rezult = rezult + op, cont = cont - 1	loop
1			R = rezult	End
End				End

(b)

Figura 14. Descriere funcțională: (a) organigramă; (b) tabelă funcțională de stări.

Proiectul complet al unui cip poate fi reprezentat sub două forme: o tabelă funcțională de stări şi un plan de amplasare. Fiecare descriere funcțională exprimată sub forma instrucțiunilor "if", "case" şi "loop" poate fi partiționată după aceste instrucțiuni. De exemplu, testul condițional dintr-o instrucțiune "if" poate fi asignat unei stări, codul din fiecare ramură fiind asignat câte unei stări. Dacă există mai multe instrucțiuni de asignare a unor variabile în fiecare ramură, ele pot fi divizate în continuare manual sau automat în mai multe stări. De exemplu, o descriere funcțională sub forma unei organigrame este prezentată în **Figura 14(a)**. Tabela funcțională de stări care corespunde

acestei organigrame este prezentată în **Figura 14(b)**. De notat că nodul de unire din **Figura 14(a)** nu trebuie să ocupe neapărat o stare completă, dar astfel descrierea este mai lizibilă şi se permite verificarea modificărilor. În proiectul final, o asemenea descriere este compactată în tabela de stări alocată. Înțelegerea proiectului dintr-o tabelă de stări compactată şi nestructurată este însă dificilă.

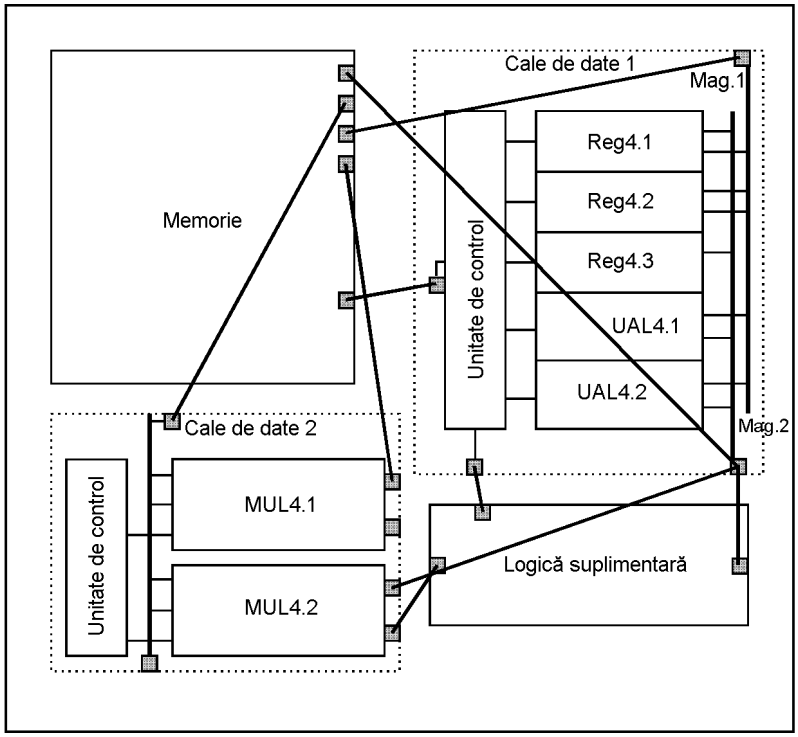


Figura 15. Plan de amplasare.

A doua reprezentare este planul de amplasare al proiectului, care indică poziția şi conexiunile fiecărei componente (**Figura 15**). Această reprezentare este ierarhică, deoarece poate fi de asemenea reprezentat planul de amplasare al fiecărei componente, ca de exemplu calea de date. Pentru fiecare componentă se reprezintă porturile acestuia. Conexiunile dintre porturi pot fi reprezentate fie ca linii drepte între porturi, fie ca şi conexiuni rutate între porturi.

Alte reprezentări facilitează selecția modulelor, amplasarea, alocarea şi planificarea. *Selecția modulelor* este realizată prin parcurgerea componentelor disponibile în *BDC*, pe categorii specifice ca de exemplu clasa (combinaționale, secvențiale), funcționarea (UAL, comparatoare, numărătoare), numărul maxim de biți şi stilul (de exemplu, cu transport succesiv sau anticipat). Informațiile fizice despre fiecare componentă sunt afişate după ce au fost specificate categoriile precedente.

Amplasarea este realizată pe baza planului de amplasare şi cu ajutorul unui editor. Proiectanții pot adăuga, şterge, roti şi repoziționa diferite module. De asemenea, se pot poziționa porturile modulelor. Similar, proiectanții pot adăuga sau şterge conexiuni,

sau le pot poziționa în cadrul canalelor din jurul modulelor. Calitatea amplasării, sub forma unor indicatori ca înălțimea, lățimea, spațiul de rutare, densitatea tranzistoarelor, lungimea conexiunilor şi frecvența ceasului, este afişată pentru fiecare amplasare.

Alocarea este executată prin selecția unităților şi asignarea obiectelor funcționale (de exemplu, variabile şi operații) la componentele RT, şi a conexiunilor la multiplexoare şi magistrale. Această asignare poate fi realizată utilizând o matrice, în care apar pe de o parte obiectele funcționale, iar pe de altă parte componentele. Calitatea unei asignări este estimată prin indicatori ca gradul de utilizare al componentelor, întârzierile de la registru la registru şi spațiul de rutare.

Planificarea este realizată prin repartiționarea descrierii funcționale inițiale în tabela funcțională de stări. Proiectantul poate adăuga sau şterge stări, poate diviza sau concatena expresii, condiții şi bucle, sau poate realoca asignările de la o stare la alta. Calitatea planificării este estimată prin indicatori ca numărul de componente necesare, suma spațiului necesar pentru toate componentele, întârzierile de la registru la registru, numărul de stări, şi timpul total de execuție.

Pentru a se permite o estimare rapidă a calității, descrierea funcțională şi planurile de amplasare sunt legate printr-o structură de date comună, astfel încât modificările din cadrul unei reprezentări sunt propagate în cealaltă reprezentare. Uneori propagarea unei modificări nu este posibilă în totalitate, şi ambele structuri trebuie actualizate. De exemplu, dacă două operații de adunare sunt planificate în acelaşi pas de control şi există un singur sumator, această modificare va genera un nou proiect, cu un sumator suplimentar, neconectat la nici o altă componentă. Acest sumator trebuie conectat manual pentru ca descrierea funcțională să fie corectă.

Mediul de conceptualizare permite utilizarea mai multor metodologii diferite. Proiectantul trebuie să aibă posibilitatea iterării unei anumite operații ori de câte ori este necesar pentru a satisface cerințele. **Figura 16** prezintă câteva scenarii posibile. Proiectantul poate porni de la o reprezentare a unei descrieri, care va fi partiționată în porțiuni mai mici. Deoarece funcționarea este specificată printr-o tabelă de stări, descrierea are întotdeauna o planificare inițială. După selecția componentelor şi asignare, dacă de exemplu planificarea violează anumite restricții de proiectare, proiectantul poate efectua o replanificare. După ce planificarea inițială devine satisfăcătoare, proiectantul poate vizualiza *BDC* pentru a identifica componentele de bază pentru proiectul final. Celelalte componente pot fi asignate ulterior, fie manual, fie automat. Proiectantul poate efectua o asignare prealabilă a unor obiecte funcționale importante la componentele selectate, înaintea executării alocării complete.

După selecția componentelor, proiectantul poate efectua amplasarea. Deoarece componentele de bază ocupă de obicei cea mai mare parte a spațiului, un plan de amplasare disponibil din fazele inițiale poate creşte viteza de explorare a spațiului de proiectare. Proiectantul poate specifica interconexiuni de bază, ca magistralele de date şi de adrese.

După amplasare, proiectantul poate reveni la planificare, deoarece restricțiile de resurse şi de temporizare sunt cunoscute mai exact. De obicei, se încearcă îmbunătățirea indicatorilor de calitate prin mai multe iterații, până la atingerea unei calități satisfăcătoare. Este de aşteptat ca proiectantul să ia cele mai multe din deciziile de nivel înalt, dar să efectueze sinteza manuală doar a unei porțiuni reduse a proiectului. Restul proiectului

va fi generat de către utilitarele de sinteză, care nu vor afecta de obicei în mod drastic deciziile luate de proiectant.

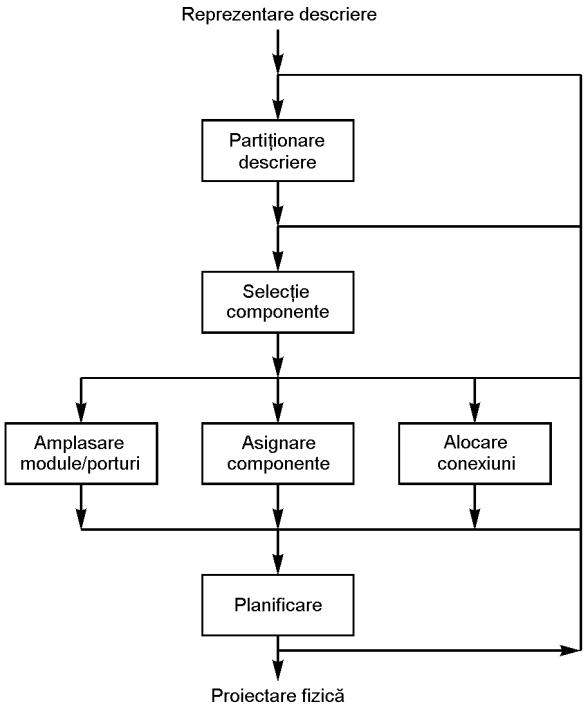


Figura 16. Scenarii posibile pentru sinteza de nivel înalt interactivă.

În concluzie, un mediu de conceptualizare permite proiectanților să execute manual aceleaşi operații pe care le execută proiectanții care lucrează la nivele mai înalte de abstractizare. Însă, un asemenea mediu permite de asemenea ca proiectanții să execute aceste operații parțial prin utilitarele de sinteză automată. Pe măsura îmbunătățirii algoritmilor pentru sinteza de nivel înalt, un număr mai mare de operații vor fi executate cu ajutorul utilitarelor de sinteză automată.

3. Rafinarea specificațiilor

3.1. Introducere

O specificație a unui sistem constă din *obiecte funcționale* ca variabile şi canale de comunicație. În timpul proiectării sistemului, aceste obiecte funcționale din cadrul specificațiilor sunt grupate într-un set de *componente de sistem*, ca circuite ASIC, memorii şi magistrale. În timp ce obiectele funcționale sunt lipsite de structură, componentele de sistem au o structură bine definită, ca numărul pinilor unui cip, numărul şi dimensiunea cuvintelor unei memorii, sau numărul liniilor unei magistrale. Actualizarea specificațiilor pentru reflectarea transformării obiectelor funcționale în componente de sistem reprezintă *rafinarea specificațiilor*.

Rafinarea specificațiilor este importantă în cadrul proiectării din mai multe motive. În primul rând, rafinarea determină ca specificația să devină consistentă din toate punctele de vedere, prin actualizarea acesteia pentru a reflecta deciziile de grupare luate în timpul partiționării sistemului. De exemplu, dacă variabilele sunt grupate pentru a fi asignate unei singure memorii, declarațiile şi referințele la aceste variabile trebuie actualizate. Rafinarea specificațiilor va înlocui declarațiile variabilelor cu o declarație a unui tablou de memorie, iar referințele la variabilele originale vor fi actualizate pentru a se accesa elemente de tablou. În al doilea rând, rafinarea permite ca specificația să poată fi simulată, permițând ca proiectantul să verifice corectitudinea funcțională a sistemului după o etapă de proiectare. În fine, specificația rafinată care este generată poate fi utilizată ca intrare pentru utilitarele de verificare, sinteză şi compilare care pot urma proiectării sistemului.

În continuare se va prezenta un set de operații de rafinare a specificațiilor. Întâi se vor descrie operațiile de rafinare asociate cu implementarea grupurilor de variabile şi canale. Se vor introduce apoi mecanisme pentru rezolvarea conflictelor de acces care pot apare atunci când se efectuează accesul concurent la variabile şi canale care au fost grupate împreună într-o memorie sau o magistrală.

3.2. Rafinarea grupării variabilelor

Memoria căreia i s-a asignat un grup de variabile este modelată în specificația rafinată ca un tablou cu un număr fix de cuvinte şi biți pe cuvânt. Pentru gruparea variabilelor trebuie efectuate două operații: amplasarea variabilelor în memorie şi translatarea adreselor de memorie.

3.2.1. Amplasarea variabilelor în memorie

Partiționarea sistemului asignează un grup de variabile la o memorie cu un număr fix de cuvinte şi o dimensiune a cuvântului fixă. Diferite variabile pot avea dimensiuni diferite. Amplasarea variabilelor în memorie se referă la asignarea biților unei variabile la biții fiecărui cuvânt din memorie.

Amplasarea variabilelor este simplă dacă dimensiunea variabilelor este mai mică decât dimensiunea cuvântului de memorie sau dacă este un multiplu exact al acestei dimensiuni. Considerăm declarațiile variabilelor din **Figura 17(a)**, care trebuie amplasate într-o memorie de 8 biți, după cum se indică în **Figura 17(b)**. Variabilelor *a* şi *b* li se asignează un cuvânt, respectiv două cuvinte de memorie.

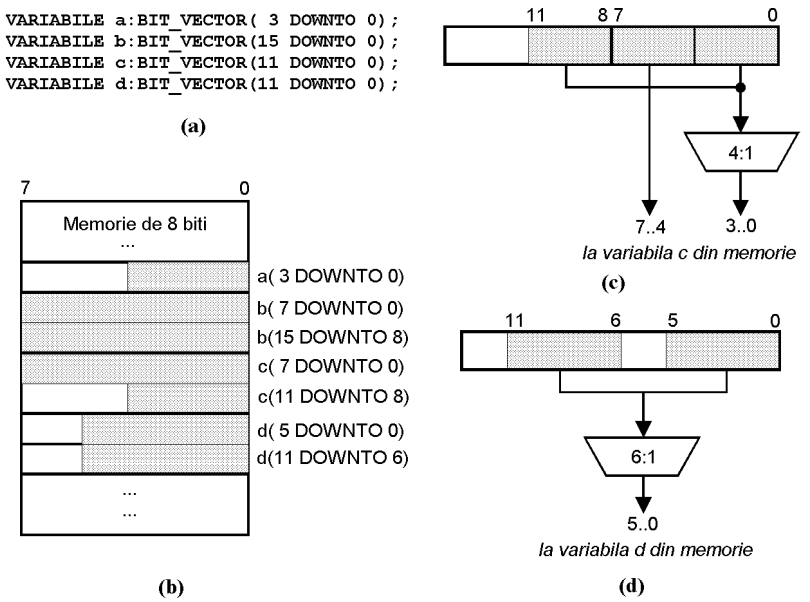


Figura 17. Amplasarea variabilelor în memorie: (a) declarații de variabile; (b) amplasarea într-o memorie de 8 biți; (c) înscrierea variabilei *c* cu o valoare de 12 biți; (d) înscrierea variabilei *d* cu o valoarea de 12 biți.

Dacă dimensiunea variabilelor nu este un multiplu întreg al dimensiunii cuvântului, variabilele pot fi amplasate într-unul din două moduri. Pentru variabila de 12 biți *c*, cei 8 biți mai puțin semnificativi sunt asignați unei adrese, iar cei 4 biți rămaşi sunt asignați următoarei adrese (**Figura 17(b)**). Variabila *d* este o altă variabilă de 12 biți, pentru care cei 6 biți mai puțin semnificativi sunt asignați unei adrese, iar ceilalți 6 sunt asignați următoarei adrese. Prima variantă necesită un multiplexor 4:1 pentru scrierea variabilei *c* în memorie, dar necesită conectarea a 8 drivere la magistrala de date a memoriei, după cum se indică în **Figura 17(c)**. A doua variantă necesită un multiplexor de 6:1, dar sunt necesare numai 6 drivere de magistrală (**Figura 17(d)**).

3.2.2. Translatarea adreselor de memorie

Asignarea adreselor la variabilele asignate la o memorie şi modificarea tuturor referinţelor la aceste variabile în cadrul specificaţiei este cunoscută ca *translatarea adreselor de memorie*.

Pentru asignarea adreselor de memorie, variabilele se pot considera în orice ordine. Numărul cuvintelor de memorie necesare pentru fiecare variabilă se poate determina prin amplasarea variabilelor în memorie. Elementelor unei variabile de tip tablou trebuie să li se asigneze adrese contigue în memorie.

Pentru variabile scalare, translatarea adreselor de memorie este relativ directă. Presupunem că o variabilă scalară *v* aparţine unui grup de variabile reprezentate de tabloul *Mem*, iar adresa variabilei *v* în memoria *Mem* este 30. Toate referinţele la variabila *v* în cadrul specificaţiei sunt înlocuite simplu prin referinţe la *Mem(30)*.

Pentru o variabilă de tip tablou care a fost grupată cu alte variabile, adresele care indexează variabila trebuie actualizate la rafinarea referinţelor la aceste variabile. De exemplu, pentru o variabilă de tip tablou *v(63 DOWNTO 0)* asignată adreselor de la 100 la 163 în memoria *Mem*, trebuie considerate mai multe aspecte pentru actualizarea referinţelor la variabila *v*.

În primul rând, dacă variabila de tip tablou *v* este indexată în totalitate prin adrese numerice, referinţele la aceasta sunt înlocuite în mod simplu prin referinţele corespunzătoare la memoria *Mem*. Astfel, referinţa *v(0)* va fi înlocuită prin *Mem(100)*, iar *v(36)* prin *Mem(136)*.

În al doilea rând, dacă tabloul este indexat printr-o expresie care implică variabile, la expresia respectivă trebuie să se adune un offset corespunzător adresei de memorie a primului element al variabilei. De exemplu, considerăm variabila de tip tablou *v*, care este indexată prin *j* sau *k* în cadrul codului din **Figura 18(a)**. Deoarece variabilei *v* i s-au asignat adrese începând cu 100 în memoria *Mem*, o referinţă *v(j)* va fi înlocuită prin *Mem(j+100)*, iar o referinţă *v(k)* va fi înlocuită prin *Mem(k+100)*, ca în **Figura 18(b)**.

În al treilea rând, dacă o variabilă este utilizată pentru a indexa un singur tablou, adunarea offsetului poate fi evitată prin iniţializarea corespunzătoare a variabilei de adresă. De exemplu, în **Figura 18(a)**, variabila *j* indexează numai variabila de tip tablou *v*. **Figura 18(c)** indică modul în care actualizarea tuturor iniţializărilor variabilei *j* (inclusiv a limitelor buclei **for**) elimină necesitatea adunării offseturilor. Totuşi, aceasta nu este posibilă întotdeauna dacă variabila *j* indexează mai multe tablouri. În acest caz, se poate evita adunarea offsetului pentru cel puţin o variabilă de tip tablou prin asignarea acesteia a adreselor de memorie începând cu 0.

```
VARIABILE j,k: INTEGER := 0;
VARIABILE v: IntArray(63 DOWNTO 0);
...
v(k) := 3;
x := v(36);
v(j) := x;
...
FOR j IN 0 TO 63 LOOP
  sum := sum + v(j);
END LOOP;
...
```

(a)

```
VARIABILE j,k: INTEGER := 0;
VARIABILE Mem: IntArray(255 DOWNTO 0);
...
Mem(k+100) := 3;
x := Mem(136);
Mem(j+100) := x;
...
FOR j IN 0 TO 63 LOOP
  sum := sum + Mem(j+100);
END LOOP;
...
```

(b)

```
VARIABILE j: INTEGER := 100;
VARIABILE k: INTEGER := 0;
VARIABILE Mem: IntArray(255 DOWNTO 0);
...
Mem(k+100) := 3;
x := Mem(136);
Mem(j) := x;
...
FOR j IN 100 TO 163 LOOP
  sum := sum + Mem(j);
END LOOP;
...
```

(c)

Figura 18. Translatarea adreselor de memorie: (a) specificație inițială; (b) specificație obținută prin înlocuirea expresiilor de index din referințele la variabila *v* cu un offset; (c) specificație cu actualizarea variabilei de index *j*.

3.3. Rafinarea canalelor şi a magistralelor

Obiectele funcționale concurente dintr-o specificație, numite de obicei procese, comunică între ele prin mesaje transmise prin canale de comunicație abstracte. Pentru minimizarea costului de interconectare, canalele din sistem sunt grupate astfel încât fiecare grup de canale este implementat printr-un mediu fizic comun numit magistrală. O magistrală constă dintr-un set de linii prin care are loc transferul datelor conform unui protocol de magistrală. Operația de generare a magistralelor şi a protocoalelor acestora pentru fiecare grup de canale este numită *rafinarea interfețelor*. Se vor descrie în continuare metode pentru rafinarea interfețelor.

3.3.1. Parametrii canalelor şi magistralelor

Pentru fiecare canal, un singur proces *master* inițiază şi controlează transferul datelor, şi unul sau mai multe procese *slave* răspund la comunicația inițiată de procesul master. Dacă procesul master transmite (sau recepționează) date prin canal, *direcția* asociată cu canalul este cea de scriere (sau citire). Canalele sunt de obicei unidirecționale, ceea ce implică faptul că dacă un proces citeşte şi scrie o variabilă a unui alt proces, sunt necesare canale distincte pentru fiecare direcție a transferului datelor.

Canalele sunt caracterizate prin patru parametri. **Dimensiunea în biți** a datelor transmise prin canal, *dim(C)*, reprezintă numărul de biți dintr-un singur mesaj transferat prin canalul *C*. Dimensiunea datelor include biții de adresă care pot fi necesari pentru accesarea variabilelor de tip tablou prin canal. **Numărul de accesuri**, *acces(P, C)*, reprezintă numărul de transferuri efectuate de procesul *P* prin canalul *C*, pe durata de viață a procesului. **Rata medie a canalului**, *rata_med(C)*, este rata la care datele sunt transmise prin canalul *C* pe durata de viață a proceselor care comunică prin canal. **Rata la vârf a canalului**, *rata_vârf(C)*, este rata la care este transferat un singur mesaj prin canalul *C*.

Orice implementare printr-o magistrală a unui canal sau grup de canale poate fi caracterizată prin patru parametri. **Dimensiunea magistralei**, *dim_bus(B)*, este numărul liniilor de date ale magistralei *B* prin care mesajele pot fi transferate între procese. Fiecărei magistrale i se asociază un protocol care defineşte secvența exactă de operații care implementează transferul mesajelor prin setul liniilor de date. **Întârzierea protocolului**, *prot_delay(B)*, este întârzierea totală a protocolului utilizat pentru un singur transfer prin magistrală. **Rata medie a magistralei**, *rata_med(B)*, este rata la care datele sunt transferate prin magistrală pe întreaga durată de viață a sistemului. **Rata la vârf a magistralei**, *rata_vârf(B)*, este rata maximă la care datele pot fi transferate prin magistrală. Între rata la vârf a magistralei şi dimensiunea magistralei există următoarea relație:

$$rata\_vârf(B) = \frac{dim\_bus(B)}{prot\_delay(B)} \qquad (3.1)$$

3.3.2. Definirea problemei

Fiind dat un grup de canale abstracte de comunicație, rafinarea interfețelor determină dimensiunea magistralei şi protocolul pentru magistrala care va implementa canalele. În cadrul acestei operații, există două cerințe contradictorii. Prima este cea de minimizare a costului de interconectare între componentele de sistem care utilizează o magistrală prin reducerea dimensiunii magistralei, *dim_bus(B)*. A doua cerință este de a se maximiza performanțele de comunicație prin magistrală, prin creşterea ratei la vârf a magistralei, *rata_vârf(B)*, şi în consecință prin creşterea *dim_bus(B)*.

Rafinarea interfețelor constă din două operații: generarea magistralei şi generarea protocolului. Fiind dat un set de restricții, **generarea magistralei** determină dimensiunea magistralei care va implementa grupul de canale. După ce a fost selectată dimensiunea dorită a magistralei, **generarea protocolului** selectează şi generează protocolul de comunicație care va implementa transferul datelor prin magistrală.

3.3.3. Generarea magistralei

Un caz simplu de determinare a dimensiunii magistralei este cel în care toate canalele unui grup au dimensiuni identice ale mesajelor. În acest caz, canalele sunt unite astfel încât toate canalele din fiecare grup sunt utilizate exclusiv pentru a comunica între aceleaşi două procese. În consecință, fiecare grup de canale este implementat cu o dimensiune a magistralei identică cu dimensiunea fiecărui canal.

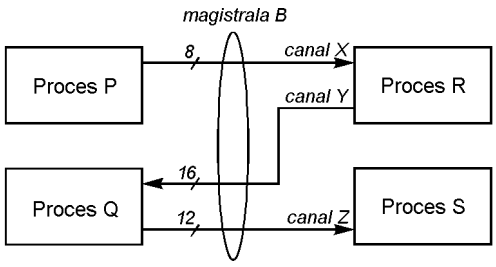


Figura 19. O magistrală tipică formată prin unirea canalelor care transferă date de diferite dimensiuni între diferite procese.

Într-un caz mai general, procesele comunicând între ele prin canale care au fost grupate împreună transferă date simultan prin mediul fizic partajat. În plus, diferite canale transferă mesaje cu dimensiuni diferite între procese. Un asemenea exemplu este prezentat în **Figura 19**. Procesele *P*, *Q*, *R* şi *S* comunică prin canalele *X*, *Y* şi *Z*, care au fost grupate prin partiționarea sistemului pentru a fi implementate printr-o singură magistrală, *B*. Cele trei canale transferă date de dimensiuni diferite, de 8, 16, respectiv 12 biți. În plus, este necesar ca procesul *P* să transfere date la procesul *R* în acelaşi timp în

care procesul *Q* trebuie să transmită date la procesul *S*. Se va descrie o metodă de generare a magistralei pentru un asemenea caz general.

3.3.3.1. Determinarea ratei magistralei

Se consideră două canale, *X* şi *Y*, care transferă mesaje de 8 biţi, respectiv 16 biţi, după cum se prezintă în **Figura 20**. Numărul de biţi asociat cu fiecare transfer de mesaj este indicat deasupra mesajului. Pentru simplitate, se presupune că cele patru intervale de timp indicate în figură reprezintă transferurile de date pe duratele de viaţă ale proceselor care comunică prin canalele *X* şi *Y*. Canalele *X* şi *Y* au ratele medii de 4 şi 12 biţi/s, respectiv. Dacă aceste canale sunt reunite într-o singură magistrală *B*, magistrala trebuie să transmită datele cu o rată de cel puţin 16 biţi/s pentru a putea satisface cerinţele pentru transferul datelor ale celor două canale originale.

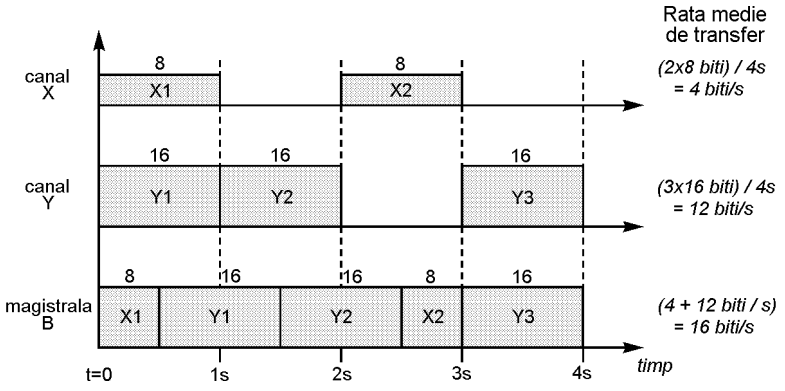


Figura 20. Unirea canalelor *X* şi *Y* sub forma magistralei *B*.

Mesajele individuale transferate prin canale au fost etichetate în figură pentru a putea fi asociate mai uşor cu datele transferate prin magistrala partajată. Considerăm mesajul etichetat cu *Y2* transferat la $t = 1$ s prin canalul original *Y*, care este transferat prin magistrala *B* la $t = 1.5$ s. În timp ce transferurile individuale de mesaje pot fi întârziate datorită conflictelor de acces la magistrală, numărul total de biţi transferaţi prin canalele individuale înaintea unirii canalelor este transmis prin magistrala partajată în acelaşi interval de timp.

Pentru sinteza magistralei *B* din **Figura 20**, se ţine cont de faptul că cele două canale individuale nu vor transfera date în permanenţă. Intervalele de timp în care un canal este inactiv sunt utilizate pentru transferul datelor celuilalt canal, realizându-se sinteza unei magistrale prin care datele sunt transferate întotdeauna cu o rată constantă.

Înainte de a fi unit într-o magistrală, dacă un canal transferă datele la o anumită rată medie, trebuie să fie capabil să transfere datele prin magistrală la aceeaşi rată medie. Aceasta se poate realiza dacă rata medie, *rata_med(B)*, a magistralei *B* este mai mare decât suma ratelor medii ale canalelor individuale. Deci

$$rata\_med(B) \geq \sum_{C \in B} rata\_med(C) \qquad (3.2)$$

Scopul generării magistralei este sinteza unei magistrale cu un număr minim de linii şi o rată medie dată de ecuaţia (3.2). Implementarea cea mai eficientă a magistralei va fi obţinută dacă magistrala nu este niciodată inactivă, şi dacă ea transferă în mod constant date cu o rată fixă. În asemenea condiţii ideale, rata la vârf şi cea medie a magistralei vor fi identice:

$$rata\_vârf(B) = rata\_med(B) \qquad (3.3)$$

3.3.3.2. Restricţii pentru generarea magistralei

Pentru un set dat de canale care au fost grupate împreună pentru a fi implementate printr-o singură magistrală, pot fi specificate restricţii pentru mai mulţi parametri ai magistralei şi ai canalelor.

O restricţie a **dimensiunii minime/maxime a magistralei** poate fi dedusă din restricţiile numărului de pini specificate pentru modulele sau cipurile cărora li s-au asignat procesele care comunică prin intermediul magistralei.

Rata medie a canalului poate fi restricţionată astfel încât să se asigure ca procesele să nu fie încetinite datorită întârzierilor de comunicaţie pe magistrală. Fiind date restricţii asupra timpului de execuţie al unui proces care comunică prin mai multe canale, proiectantul poate aloca sau calcula timpul necesar comunicaţiei prin diferite canale, din care se poate deduce o restricţie asupra ratei medii minime a canalului. O rată medie maximă a canalului poate fi specificată în cazurile în care unul din procesele care comunică prin canal reprezintă un dispozitiv lent care nu poate transmite sau recepţiona datele mai rapid decât o anumită rată.

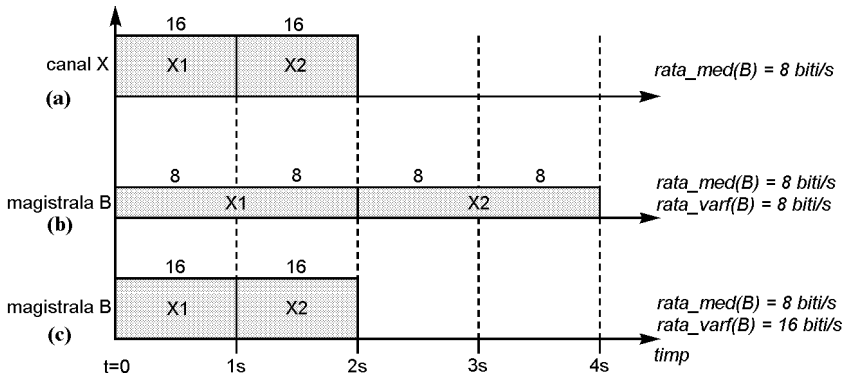


Figura 21. Restricţii asupra ratei la vârf a canalelor: (a) trasarea execuţiei pentru canalul *X*; (b) magistrala *B* implementată fără restricţii asupra ratei la vârf a canalelor; (c) magistrala *B* implementată cu o restricţie a ratei la vârf de minim 16 biţi/s.

În anumite cazuri, poate fi specificată o **rată la vârf** minimă, pentru a se asigura ca transferul unui singur mesaj prin canal să nu necesite un interval de timp excesiv. De exemplu, considerăm canalul *X* din **Figura 21(a)**, care transferă un mesaj de 16 biți în fiecare din primele două intervale de timp, $t = 0$ şi $t = 1$. Celelalte două intervale de timp sunt utilizate pentru a efectua calcule interne de către procesul care comunică prin canalul *X*. Dacă s-ar implementa canalul ca o magistrală, din ecuația (3.2), rata medie a magistralei ar fi de 8 biți/s, trasarea execuției fiind cea din **Figura 21(b)**. Însă, procesul ar necesita atunci patru intervale de timp numai pentru comunicația prin magistrală. Aceasta nu se poate accepta, deoarece vor fi necesare încă două intervale suplimentare de timp pentru execuția calculelor interne ale procesului (executate inițial în intervalele de timp $t = 2$ şi $t = 3$ în **Figura 21(a)**).

Dacă se specifică pentru canalul *X* o rată la vârf minimă de 16 biți/s, se va obține implementarea dorită a magistralei din **Figura 21(c)**, care nu necesită intervale de timp suplimentare. Deci, pentru toate canalele *C* cărora li se asociază o restricție minimă a ratei la vârf, există relația:

$$rata\_vârf(B) > rata\_vârf(C) \tag{3.4}$$

În cazul în care se specifică o rată la vârf minimă a canalului, magistrala rezultată poate fi inactivă în anumite intervale de timp.

3.3.3.3. Algoritm pentru determinarea dimensiunii magistralei

După ce s-au introdus restricțiile care pot fi specificate pentru generarea magistralei, se prezintă un algoritm pentru determinarea dimensiunii unei implementări a magistralei. Algoritmul presupune că liniile de control şi de date sunt disjuncte. În orice moment de timp, un singur canal poate transfera date prin magistrală. Dacă dimensiunea magistralei este mai mare decât numărul de biți de adrese şi de date, biții de adrese şi de date sunt transferați simultan prin magistrală, în caz contrar aceştia sunt transferați separat, în două transferuri distincte. În ultimul caz, biții de adresă trebuie memorați de procesul receptor. Dacă dimensiunea magistralei este mai mică decât dimensiunea mesajului care trebuie transmis prin magistrală, mesajul este transmis prin transferuri multiple.

Se presupune că toate procesele care comunică printr-una din canalele magistralei au o implementare sincronă. Transferul unui mesaj prin magistrală necesită deci un număr întreg de cicluri de ceas. O variabilă care se accesează prin magistrală este modelată printr-un proces separat care transmite şi recepționează valoarea acesteia prin magistrală ca răspuns la cererile de la alte procese. Deci, pentru calculul timpului de execuție al unui proces master care accesează o variabilă prin magistrală, se presupune că procesul slave care modelează acea variabilă este întotdeauna gata pentru transfer, adică nu apar întârzieri de sincronizare pentru accesul la variabile prin magistrală.

Intrarea pentru algoritmul de generare a dimensiunii magistralei constă dintr-un set de canale care trebuie implementate printr-o singură magistrală şi din restricții asupra ratelor de transfer prin canale şi asupra dimensiunii magistralei. Ieşirea algoritmului este dimensiunea în biți a magistralei care va implementa setul de canale.

Algoritmul examinează un domeniu al dimensiunilor posibile. Pentru fiecare dimensiune, se calculează rata la vârf a magistralei şi ratele medii individuale ale canalelor. Pentru sinteza unei magistrale care transferă date în mod constant, conform ecuațiilor (3.2) şi (3.3) rata la vârf a magistralei trebuie să fie mai mare decât suma ratelor medii ale canalelor. Fiecare dimensiune a magistralei pentru care această condiție este respectată reprezintă o implementare fezabilă a magistralei. Din setul de implementări fezabile ale magistralei, fiecare corespunzând unei dimensiuni diferite a magistralei, se selectează cea cu costul cel mai redus. În cazul în care nu sunt specificate restricții, se selectează o dimensiune unitară care corespunde unui transfer serial de date.

if nu există restricții **then**
    return (1)
end if
/* calculează domeniul dimensiunilor */
dim_min = 1
dim_max = *Max* (*biți*(*C*))
mincost = ∞
dim_mincost = ∞
for *dim_curentă* **in** *dim_min* **to** *dim_max* **loop**
    /* calculează rata la vârf a magistralei */
    rata_vârf (*B*) = *dim_curentă* / *prot_delay* (*B*)
    /* calculează suma ratelor medii ale canalelor pentru *dim_curentă* */
    suma_rata_med = 0;
    for toate canalele $C \in B$ **loop**

$$rata\_med(C) = \frac{acces(P,C) \times biti(C)}{timp\_calcul(P) + timp\_comunic(P)}$$

        suma_rata_med = *suma_rata_med* + *rata_med* (*C*)
    end loop
    if (*rata_vârf* (*B*) > *suma_rata_med*) **then**
        /* soluție fezabilă, determină costul minim */
        cost_curent = *CalculCost* (*dim_curentă*)
        if (*cost_curent* < *mincost*) **then**
            mincost = *cost_curent*
            dim_mincost = *dim_curentă*
        end if
    end if
end loop
if (*mincost* = ∞)
    then return (*eşec*)
    else return (*dim_mincost*)
end if

Figura 22. Algoritm pentru determinarea dimensiunii magistralei.

Algoritmul pentru determinarea dimensiunii magistralei este prezentat în **Figura 22**. Întâi se determină domeniul dimensiunilor examinate de algoritm. Dimensiunea ma-

ximă a magistralei, *dim_max*, este dimensiunea maximă a mesajelor transmise prin oricare canal. Dimensiunea minimă, *dim_min*, este 1.

Variabila *dim_curentă* reprezintă dimensiunea curentă care este evaluată de algoritm. Pentru fiecare valoare din domeniul (*dim_min*, *dim_max*), se calculează rata la vârf a magistralei cu ecuația (3.1).

Se notează cu *acces(P,C)* numărul mediu de accesuri efectuate de către procesul *P* la canalul *C*. Numărul de biți transmişi sau recepționați prin canalul *C* într-un singur mesaj se notează cu *biți(C)*. Dacă un proces face acces la o variabilă de tip tablou prin canal, dimensiunea adresei este de asemenea inclusă în *biți(C)*. De exemplu, dacă un proces face acces la o variabilă scalară de 16 biți *x* şi la o variabilă de tip tablou *y* de 32 cuvinte $\times$ 16 biți prin canalele *Cx*, respectiv *Cy*, atunci *biți(Cx)* este 16, iar *biți(Cy)* este 21 (5 biți de adresă şi 16 biți de date).

Pentru simplitate, se presupune că un proces *P* are un singur canal *C* prin care sunt transferate mesajele. În cazul în care dimensiunea curentă este mai mică decât numărul de biți ai mesajului, pot fi necesare mai multe transferuri (în număr de $\lceil biți(C) / dim\_curentă \rceil$) pentru implementarea transferului unui singur mesaj. Timpul necesar comunicației pentru procesul *P* este calculat astfel:

$$timp\_comunic(P) = acces(P,C) \times \left(\left\lceil \frac{biți(C)}{\dim\_curenta} \right\rceil \times prot\_delay(B) \right) \quad (3.5)$$

Utilizând valoarea calculată mai sus pentru *timp_comunic(P)*, se poate calcula rata medie pentru fiecare canal asignat magistralei. Suma ratelor medii a canalului pentru o valoare specifică a variabilei *dim_curentă* este memorată în *suma_rata_med*.

Dacă rata la vârf a magistralei, *rata_vârf(B)*, este mai mică decât suma ratelor medii a tuturor canalelor, atunci *dim_curentă* reprezintă o implementare a magistralei care nu este fezabilă. Se repetă calcularea ratei la vârf a magistralei şi a sumei ratelor canalelor pentru următoarea dimensiune în cadrul domeniului (*dim_min*, *dim_max*).

Dacă rata la vârf a magistralei este mai mare decât suma ratelor medii ale canalelor, atunci *dim_curentă* reprezintă o implementare fezabilă a magistralei. Pentru fiecare restricție asupra dimensiunii magistralei, a ratelor medii şi la vârf specificate pentru magistrală, procedura *CalculCost* calculează costul unei implementări fezabile ca suma pătratelor pentru toate violările restricțiilor, înmulțite cu ponderile relative specificate pentru restricțiile respective. De exemplu, presupunem că singura restricție specificată este cea a dimensiunii maxime a magistralei, reprezentată de *max_linii*. Fie *k* ponderea relativă specificată pentru această restricție. Pentru fiecare valoare a dimensiunii magistralei, *dim_curentă*, funcția de cost pentru magistrală va fi definită astfel:

$$\cos t = \begin{cases} (k(\dim\_curenta - \max\_linii))^2 & \text{daca } \dim\_curenta > \max\_linii \\ 0 & \text{altfel} \end{cases} \quad (3.6)$$

Pot fi specificate şi alte funcții de cost pentru evaluarea costului asociat cu o dimensiune specifică a magistralei. De exemplu, dacă se doreşte o implementare a magistralei cu un număr minim de linii, costul poate fi calculat simplu, ca fiind valoarea *dim_curentă*.

Dacă există mai multe soluţii fezabile pentru grupul de canale, pentru implementare se selectează dimensiunea care conduce la costul minim. Variabila *mincost* din cadrul algoritmului reprezintă costul minim calculat pentru toate implementările fezabile, iar variabila *dim_mincost* reprezintă dimensiunea care corespunde costului minim.

Dacă nu există soluţii fezabile pentru niciuna din dimensiunile examinate, atunci o implementare pentru grupul respectiv de canale nu este posibilă. Orice implementare a unui asemenea grup de canale va întârzia procesele care comunică prin magistrală. O asemenea situaţie poate apare dacă mai multe canale cu rate medii foarte ridicate sunt grupate pentru a fi implementate printr-o singură magistrală. O soluţie poate fi divizarea grupului de canale pentru a fi implementate ulterior prin mai multe magistrale. O altă posibilitate este selecţia dimensiunii nefezabile cu costul minim.

3.3.4. Generarea protocolului

După selecţia unei dimensiuni corespunzătoare a magistralei care va implementa un grup de canale, generarea protocolului defineşte mecanismul transferului de date prin magistrală. O magistrală constă din trei seturi de linii.

Liniile de **date** sunt utilizate pentru transmiterea datelor prin magistrală. Numărul necesar al liniilor de date poate fi determinat cu ajutorul algoritmului de determinare a dimensiunii magistralei, sau acest număr poate fi specificat de proiectantul de sistem.

Liniile de **control** sunt necesare pentru sincronizarea proceselor care comunică prin intermediul magistralei. Numărul liniilor de control necesare depinde de tipul protocolului selectat pentru implementarea transferurilor de date. De exemplu, un protocol standard cu confirmare necesită două semnale de control, *start* şi *done*. Semnalul *start* este setat/resetat de procesul master, iar semnalul *done* este setat/resetat de procesul slave asociat cu un canal. Setul liniilor de control este partajat de toate canalele care sunt asignate la aceeaşi magistrală.

Liniile de **identificare** sau de mod sunt necesare pentru identificarea canalului care transferă date prin magistrală în orice moment de timp. Deoarece semnalele de control ale magistralei sunt partajate de toate canalele, asemenea linii de identificare (ID) sunt esenţiale pentru a permite ca procesele să recunoască semnalele de control de pe magistrală care le sunt destinate. Fiecărui canal de pe magistrală i se asociază un identificator unic, care este utilizat ca adresă a acestuia. De fiecare dată când un proces master iniţiază un transfer de date prin magistrală, acesta plasează identificatorul corespunzător al canalului pe liniile ID ale magistralei, astfel încât numai procesul slave corespunzător va răspunde la semnalele de control. Liniile ID pot fi de asemenea codificate direct cu adresele datelor accesate prin magistrală. În asemenea cazuri, procesele slave trebuie să aibă un mecanism de detectare a adreselor care examinează fiecare adresă plasată pe magistrală, pentru a determina dacă ele trebuie să răspundă la semnalele de control setate de procesul master.

Se va ilustra generarea protocolului printr-un exemplu simplu, prezentat în **Figura 23**. Variabilele *x* şi *Mem* sunt accesate de procesele *P* şi *Q*. Liniile întrerupte indică asignarea proceselor şi a variabilelor la componentele sistemului. Canalele *C1*, *C2*, *C3*

şi *C4* sunt grupate într-o singură magistrală *B*, a cărei dimensiune s-a determinat ca fiind de 8 biţi. Generarea protocolului constă din mai multe etape:

- **Selecţia protocolului**. Pentru implementarea unei magistrale, pot fi selectate diferite protocoale de comunicaţie, ca de exemplu protocolul cu confirmare totală sau cel cu confirmare parţială. Fiecare protocol necesită un număr diferit de linii de control. Pentru magistrala *B* din **Figura 23**, se selectează un protocol cu confirmare totală. Se utilizează două semnale de control, *start* şi *done*, pentru implementarea protocolului.

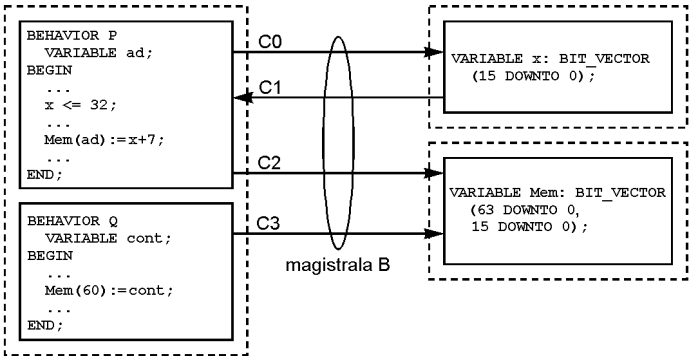


Figura 23. Accesul la variabile prin canale grupate într-o magistrală.

- **Asignarea identificatorului** (ID). Dacă *N* canale sunt implementate prin aceeaşi magistrală, vor fi necesare $log_2(N)$ linii pentru codificarea identificatorului canalului. Cele patru canale din **Figura 23** necesită două linii ID. Canalului *C0* i se asignează identificatorul "00", canalului *C1* "01", ş.a.m.d.
- **Definirea structurii magistralei**. Structura magistralei (liniile de date, de control şi de identificare) este definită în cadrul specificaţiei. Pentru fiecare canal asignat magistralei, se generează proceduri corespunzătoare de transmisie şi recepţie. **Figura 24** prezintă declaraţia unei magistrale de opt biţi, cu două linii de control şi două linii ID. Magistrala *B* este declarată ca o variabilă globală (un semnal în cazul limbajului *VHDL*), pentru a putea fi accesată de toate procesele. Procesul *P* înscrie variabila *x* de 16 biţi prin canalul *C0*. Deoarece dimensiunea magistralei este de 8 biţi, procedurile *SendC0* şi *ReceiveC0* din **Figura 24** transferă mesajul de 16 biţi pe magistrală în două transferuri de câte 8 biţi fiecare.
- **Actualizarea referinţelor la variabile**. Referinţele la o variabilă care a fost asignată unei alte componente a sistemului în urma partiţionării trebuie actualizate în procesele care iniţial utilizau referinţele directe la variabila respectivă. Accesele la variabile sunt înlocuite prin apelurile la procedurile de transmisie şi recepţie corespunzătoare canalului prin care se accesează variabila respectivă. De exemplu, în **Figura 23**, procesul *P* înscrie variabila *x* direct cu valoarea "32". Canalul *C0* reprezintă scrierea variabilei *x*. Instrucţiunea de asignare "`x <= 32`" este înlocuită prin apelul la procedura de transmisie "`sendC0(32)`", după cum se

indică în **Figura 25**. Instrucțiunea "`Mem(60) := cont`" din procesul *Q* este actualizată cu "`sendC3(60,cont)`", indicând faptul că valoarea *cont* se înscrie la adresa 60 a tabloului *Mem*.

```
TYPE HandShakeBus IS RECORD
  start, done: BIT;
  ID: BIT_VECTOR (1 DOWNTO 0);
  DATA: BIT_VECTOR (7 DOWNTO 0);
END RECORD;
SIGNAL B: HandShakeBus;
PROCEDURE ReceiveC0 (rxdata: OUT BIT_VECTOR) IS
BEGIN
  FOR j IN 1 TO 2 LOOP
    WAIT UNTIL (B.start = '1') AND (B.ID = "00");
    rxdata (8*j - 1 DOWNTO 8*(j-1)) <= B.DATA;
    B.done <= '1';
    WAIT UNTIL (B.start = '0');
    B.done <= '0';
  END LOOP;
END ReceiveC0;
PROCEDURE SendC0 (txdata: IN BIT_VECTOR) IS
BEGIN
  B.ID <= "00";
  FOR j IN 1 TO 2 LOOP
    B.DATA <= txdata (8*j - 1 DOWNTO 8*(j-1));
    B.start <= '1'
    WAIT UNTIL (B.done = '1');
    B.start <= '0'
    WAIT UNTIL (B.done = '0');
  END LOOP;
END SendC0;
```

Figura 24. Definirea magistralei *B* şi a protocoalelor pentru transmisie şi recepție pentru canalul *C0*.

- **Generarea proceselor pentru variabile**. În scopul obținerii unei specificații a sistemului care se poate utiliza pentru simulare, se crează un proces separat pentru fiecare grup de variabile care sunt accesate printr-un canal. În aceste procese sunt incluse proceduri corespunzătoare de transmisie şi recepție pentru a răspunde cererilor de acces la variabile prin magistrală. În **Figura 23**, variabilele *x* şi *Mem* sunt asignate unor componente diferite ale sistemului, după cum se indică prin liniile întrerupte. În **Figura 25**, au fost create procesele *xProc* şi *MemProc* pentru cele două variabile.

Generarea protocolului care a fost prezentată în această secțiune are mai multe avantaje. În primul rând, specificația obținută prin rafinare poate fi simulată, şi funcționarea poate fi verificată, după introducerea magistralelor şi a protocoalelor de comunicație. În al doilea rând, prin încapsularea transferului de date pe magistrală sub forma procedurilor de transmisie şi recepție, descrierea procesului rămâne mai ordonată decât în cazul în care s-ar insera asignările pentru liniile de control şi de date în fiecare punct

de comunicație din cadrul procesului. În sfârşit, dacă într-o etapă ulterioară se va selecta un alt protocol de comunicație, vor trebui modificate numai declarația magistralei şi procedurile de transmisie şi de recepție.

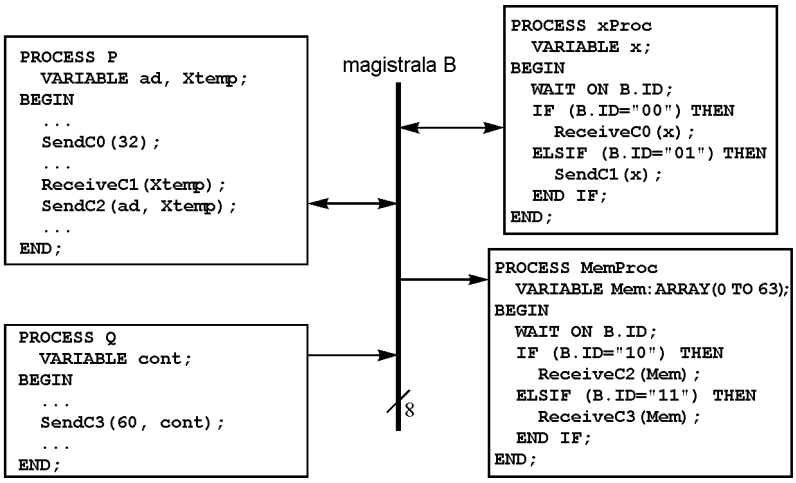


Figura 25. Specificație obținută după generarea protocolului.

4. Concluzii

În această lucrare, s-au introdus conceptele de bază legate de sistemele de sinteză de nivel înalt şi s-a prezentat o metodologie de proiectare pentru sinteza la nivel de sistem şi la nivel de cip, începând de la specificație şi până la amplasare şi rutare. Au fost prezentate mai întâi cerințele pentru sistemele de sinteză de nivel înalt şi soluțiile posibile de realizare ale acestor sisteme.

Descrierea inițială este compilată într-un graf al fluxului de date care identifică în mod explicit dependențele de date. Resursele hardware şi restricțiile de proiectare fiind specificate, se pot aplica algoritmii de planificare şi alocare. Aceşti algoritmi adnotează graful fluxului de date cu atribute suplimentare pentru a indica asignarea la stări şi componente. Graful adnotat este utilizat pentru generarea căii de date.

Sinteza de nivel înalt generează o descriere structurală care conține componente ca unități aritmetice şi logice, multiplexoare, registre, memorii şi controlere. Pentru fiecare din aceste componente, trebuie să se realizeze sinteza utilizând tehnici de sinteză logică sau secvențială pe baza descrierii componentelor memorate într-o bază de date a

componentelor. Această bază de date poate conține de asemenea dimensiunea fiecărei componente necesară pentru operația de amplasare, şi poziția fiecărui pin al componentelor necesară pentru rutarea în cadrul proiectării fizice.

S-au formulat cerințele pentru procesul de proiectare şi s-a descris un sistem generic de sinteză ipotetic, care satisface criteriile următoare: *completitudine*, *extensibilitate*, *controlabilitate*, *interactivitate*, *posibilitatea îmbunătățirii metodologiei de proiectare*. De asemenea, s-a descris un mediu pentru sinteză constând dintr-o *bază de date a sistemului*, o *bază de date a componentelor* şi un *mediu de conceptualizare* pentru sinteza interactivă.

Sinteza la nivel de sistem are ca scop reprezentarea descrierii sistemului ca un set de procese comunicante, evaluarea caracteristicilor de bază ale acesteia, şi partiționarea descrierii într-o ierarhie de descrieri care reflectă entități fizice ca plăci, module multi-cip, cipuri sau macrocelule utilizate pentru fabricația sistemului. Sinteza la nivel de sistem produce o descriere la nivel de cip sub forma unor subdescrieri funcționale care pot fi implementate prin arhitecturi ASFD care comunică între ele.

Sinteza la nivel de cip are ca scop transformarea unei descrieri funcționale ASFD într-o descriere structurală care utilizează componente RT.

Sinteza logică şi secvențială constă din operații ca minimizarea stărilor, codificarea stărilor, minimizarea logică, adaptarea tehnologică, sinteza interfețelor şi a memoriei.

Cercetarea în domeniul sintezei de nivel înalt se desfăşoară de mai mulți ani, dar nu s-a concentrat asupra sistemelor complete de sinteză. De o importanță particulară este sinteza sistemelor hardware/software complexe. Descrierile sistemelor nu fac distincție întotdeauna între implementările software şi hardware. Anumite părți ale descrierii pot fi implementate prin software, executat de un procesor standard, alte părți pot fi implementate prin componente de sistem, de exemplu controlere de dispozitive, iar altele pot fi implementate direct prin hardware specific aplicației.

Pentru a se permite implementarea descrierilor prin componente standard, trebuie să se dezvolte o tehnologie de mapare pentru aceste componente. Pentru implementări software, sunt necesare compilatoare generice, care vor compila descrierile de sistem în componente standard specificate prin structura acestora, şi nu printr-un set de instrucțiuni. Aceste compilatoare vor utiliza un set de transferuri între registre sau un set de microoperații în locul seturilor de instrucțiuni. Este necesar un progres în domeniul specificării formale a arhitecturilor destinație, pentru a fi posibilă optimizarea compilării pentru o arhitectură dată.

Deoarece utilitarele pentru sinteza la nivel de cip şi de sistem nu sunt disponibile încă, este justificată atenția acordată sistemelor hibride pentru sinteza interactivă. Asemenea sisteme trebuie să permită controlul manual al procesului de sinteză, cu ajutorul unor indicatori corespunzători de calitate. Automatizarea proiectării poate fi obținută numai după definiția indicatorilor de calitate ai proiectării şi înțelegerea influenței stilurilor, topologiilor arhitecturilor şi a metodologiei de proiectare asupra calității proiectelor. Astfel, sunt necesare cercetări asupra analizei comparative a calității proiectelor la toate nivelele de proiectare.

Bazele de date pentru proiectarea la nivel de sistem şi de cip nu sunt încă disponibile, deoarece s-a acordat o importanță redusă bazelor de date a sistemelor. Similar, în

scopul acceptării sintezei de nivel înalt de către proiectanți, sunt necesare cercetări asupra bazelor de date a componentelor şi a generatoarelor de componente pentru generarea instanțierilor de componente. Deci, sunt necesare cercetări nu numai pentru dezvoltarea algoritmilor, ci şi pentru dezvoltarea infrastructurii pentru sinteza la nivel de cip şi de sistem.

Au fost prezentate unele operații care trebuie executate pentru rafinarea specificației inițiale a sistemului după procesele de planificare şi alocare. S-a arătat importanța rafinării specificațiilor în cadrul proiectării sistemelor. S-au prezentat aspecte legate de gruparea variabilelor şi a canalelor, fiind descrisă o metodă pentru determinarea dimensiunii magistralei. Pentru o dimensiune selectată a magistralei, s-a arătat modul în care pot fi generate protocoalele de comunicație.

În cadrul rafinării specificațiilor, trebuie să se țină cont de câteva aspecte suplimentare. De exemplu, este necesară rezolvarea conflictelor de acces la o resursă partajată din cadrul sistemului. Sunt necesare tehnici pentru interfațarea protocoalelor fixe care sunt incompatibile. De asemenea, este necesară rafinarea comunicației între procesele pentru care s-au realizat implementări hardware şi software.

Operațiile prezentate pot fi extinse în diferite direcții. De exemplu, în timpul generării protocolului sunt necesari indicatori pentru a evalua diferite protocoale care pot fi selectate pentru implementarea transferurilor prin magistrală. În cadrul determinării dimensiunii magistralei, s-a presupus că întârzierile de arbitraj sunt neglijabile. Este necesară includerea efectului întârzierilor de arbitraj asupra timpului de execuție al proceselor şi a ratelor medii ale canalelor. De asemenea, este necesară studierea efectului diferitelor metode de arbitraj asupra performanțelor proceselor care comunică printr-o magistrală.

Pot fi aplicate diferite metode de optimizare asupra proceselor de interfață generate pentru compatibilizarea a două protocoale. Un exemplu de asemenea optimizare este minimizarea numărului şi a dimensiunii variabilelor utilizate de procesul de interfață în scopul reducerii complexității circuitelor care vor implementa procesul de interfață. Deşi descrierile prin limbaje de descriere hardware a proceselor de arbitraj şi de interfață permit simularea acestora împreună cu specificația sistemului, sinteza acestor procese prin metode de sinteză tradiționale bazate pe limbajele de descriere poate fi ineficientă. De exemplu, operații de sinteză ca planificarea aplicate procesului de arbitrare vor determina implementarea arbitrului de magistrală prin mai multe stări. Însă, aceste circuite sunt de obicei combinaționale. De aceea, este necesară dezvoltarea unor metode pentru sinteza eficientă a proceselor de interfață şi de arbitraj pe baza specificațiilor acestora prin limbaje de descriere.

Bibliografie

1. Giovanni De Micheli: *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.
2. P. Michel, U. Lauther, P. Duzy: *The Synthesis Approach to Digital System Design*. Kluwer Academic Publishers, 1992.
3. D. Patel, M. Schlag, M. Ercegovac: *An Environment for the Multi-level Specification, Analysis, and Synthesis of Hardware Algorithms*. Lecture Notes in Computer Science, Vol. 201: Functional Programming Languages and Computer Architecture, 1985, pp. 238-255.
4. D. D. Gajski, N. D. Dutt, C. H. Wu, Y. L. Lin: *Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.
5. D. D. Gajski, F. Vahid, S. Narayan, J. Gong: *Specification and Design of Embedded Systems*. P T R Prentice Hall, Englewood Cliffs, 1994.
6. P. Eles, K. Kuchinski, Z. Peng, A. Doboli: *Specification of Timing Constraints in VHDL for High-Level Synthesis*. Proceedings of ConTI '94-International Conference on Technical Informatics, vol. 5., pp. 26-36.
7. A. J. Martin: *Tomorrow's Digital Hardware will be Asynchronous and Verified*. Technical Report, Department of Computer Science, California Institute of Technology, Pasadena CA, 1993.
8. A. J. Martin: *Asynchronous Datapaths and the Design of an Asynchronous Adder*. Technical Report, Department of Computer Science, California Institute of Technology, Pasadena CA, 1991.
9. M. J. Pertel: *A Critique of Adaptive Routing*. Technical Report, Department of Computer Science, California Institute of Technology, Pasadena CA, 1992.
10. H. Barringer, G. Gough, B. Monahan, A. Williams: *The ELLA Verification Environment*. A Tutorial Introduction. Technical Report, Department of Computer Science, University of Manchester, 1994.
11. W. Rosenstiel, R. Camposano: *The Karlsruhe DSL Synthesis System*. Proceedings of the IFIP WG 10.2 Working Conference on From HDL Descriptions to Guaranteed Correct Circuit Designs, Elsevier Science Publishers, pp. 155-168.
12. Z. Navabi, C. H. Chiang, Fr. J. Hill: *Storage Logic Array Realization of RTL Descriptions*. Proceedings of the IFIP WG 10.2 International Symposium on Computer Hardware Description Languages and their Applications, North-Holland Publishing Company, pp. 153-163.
13. Yachyang Sun: *Algorithmic Results on Physical Design Problems in VLSI and FPGA*. PhD Thesis, University of Illinois, Urbana, 1994.
14. T-T. Hwang, R. M. Owens, M. J. Irwin, K. H. Wang: *Logic Synthesis for Field Programmable Gate Arrays*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. Vol. 13, No. 10, Oct. 1994, pp.1280-1287.
15. G. D. Chen, D. D. Gajski: *An Intelligent Component Database for Behavioral Synthesis*, Proceedings of the 27th Design Automation Conference, pp. 150-155, 1990.