UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI

Instrumente Structurale 2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Testarea Sistemelor

2. Tipuri de simulare, valoarea logică necunoscută

Tipuri de Simulare

Simularea logică este unul dintre mijloacele de testare cele mai eficiente a verificării proiectării, considerând atât avantajele cât şi limitările acesteia. Principalele tipuri de simulatoare sunt simulatoarele compilate şi simulatoarele conduse prin evenimente. Există anumite elemente cheie pot influenţa atât acurateţea cât şi performanţele simulatoarelor. Printre acestea un rol deosebit îl au modul specific de tratare al valorilor logice neprecizate, modelarea întârzierilor, detectarea hazardurilor, controlul oscilațiilor şi sistemul valorilor necesare pentru circuitele logice cu trei stări ale circuitele MOS. Tehnicile de evaluare a elementelor simulate dar şi algoritmii de simulare conduşi prin evenimente la nivel logic ori prin extensie la niveluri RTL dar şi comportamental sunt tot atâtea trăsături definitorii. Simularea logică în contextul simulărilor digitale este apreciată ca fiind printre domeniile cele mai dezvoltate şi, implicit, evoluate din domeniul ingineriei calculatoarelor.

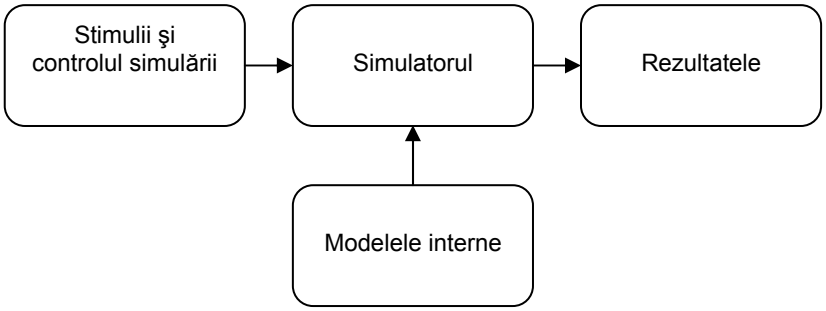

Figura 1. Fluxurile proceselor de simulare.

Aplicaţii tipice

Simularea logică este forma cea mai larg utilizată pentru *testarea verificării proiectării.* Simularea logică folosește un model al sistemului proiectat. În figura 1 este prezentată diagrama fluxului de informaţii în procesele de simulare. Simulatorul (programul de simulare) procesează o reprezentare a stimulilor de intrare şi determină evoluția în timp a semnalelor din modelul simulat.

Verificarea unui proiect logic încearcă să stabilească daca proiectul respectiv realizează comportamentul specificat, ceea ce include atât funcționalitatea cât şi duratele de funcționare specifice (*timing*-ul). Verificarea se face prin compararea rezultatelor obținute prin simulare cu valorile corespunzătoate așteptate așa cum sunt acestea definite prin specificația sistemului.
În plus, simularea logică poate fi folosită pentru verificarea faptului că operarea sistemului este:

- corectă şi independentă de starea inițială (de start);
- insensibilă la anumite variații în întârzierile interne ale unora dintre componente;
- lipsită de curse critice, de oscilații, de condiții de intrare *ilegale* şi de stări în care se poate bloca.

Alte aplicații ale simulării în procesul de proiectare sunt:

- *evaluarea alternativelor de proiectare,* în vederea creşterii raportului performanţă/cost;
- *evaluarea unor propuneri de schimbări* exprimate asupra unui proiect deja existent, în scopul verificării faptului că modificările intenționate vor produce schimbarea dorită fără efecte laterale nedorite;
- *documentarea proiectului* (generarea *diagramelor de timp,* etc.).

Proiectanții trebuie sa aibă la dispoziţie, tradițional, un prototip pentru verificarea unui nou proiect. Principalul avantaj al prototipului este acela că acesta lucrează la viteze de operare nominale dar, pe de-altă parte construcția unui prototip poate fi costisitoare şi consumatoare de timp. Prototipurile construite cu componente discrete nu au acuratețe, ca modele ale circuitelor integrate complexe. Simularea *înlocuieşte prototipul cu un model software,* ceea ce este uşor de analizat şi, mai ales, de modificat. Verificarea proiectării bazate pe simulare beneficiază de caracteristici suplimentare indisponibile procesului de verificare al proiectării bazat pe prototipuri:

- verificarea condițiilor de eroare (conflictele de magistrală, spre exemplu);
- abilitatea schimbării întârzierilor din model în scopul verificării condițiilor celor mai dificile din punctul de vedere al funcționarii cu considerarea relațiilor temporale;
- demararea circuitului simulat din orice stare se dorește;
- controlul precis al temporizării evenimentelor asincrone (întreruperile, spre exemplu);
- abilitatea furnizării unui mediu automat de testare pentru circuitul simulat, prin cuplarea acestuia cu un model LTR care pilotează şi/sau observă circuitul pe durata simulării.

Problemele specifice în verificarea proiectării bazate pe simulare

Există trei probleme intercorelate în testarea verificării proiectării bazate pe simulare (ca dealtfel, în egală măsura, în orice tip de experiment de testare) :

- cum trebuiesc generați stimulii de intrare ? (problema *generării testului*);
- cum se recunosc rezultatele corecte ?
- cât de "buni" sunt stimulii de intrare aplicați, adică cât de *complet* este acest experiment de testare ? (*evaluarea* sau *calificarea testului*).

Stimulii de intrare sunt, în mod obişnuit, organizați ca o secvență de *cazuri de test,* unde un caz de test este desemnat să verifice un anumit aspect al comportamentului modelului. Rezultatele sunt considerate corecte atunci când acestea coincid, se potrivesc, rezultatelor aşteptate, rezultatele aşteptate fiind furnizate prin specificația de proiectare. Inițial, specificaţia constă dintr-un model informal (mental sau scris) al comportamentului intenţionat al sistemului.
Pe durata procesului de proiectare *de –sus – în – jos*, modelul formal de la nivelul cel mai ridicat (acesta este, de regulă, un model LTR) este verificat în raport cu modelul informal inițial.

După aceasta, orice model de nivel ridicat defineşte specificaţia de implementare pentru implementarea sa la nivelul imediat inferior. Verificarea faptului că implementarea satisface specificaţia se reduce la aplicarea aceloraşi cazuri de test

ambelor modele şi folosind rezultatele obţinute de la nivelul mai înalt ca rezultate aşteptate pentru nivelul mai coborât.
Este important să se înțeleagă diferenţele dintre generarea testului pentru verificarea proiectării, unde obiectivul este găsirea erorilor de proiectare, comparativ cu generarea testului pentru detectarea defectelor fizice dintr-un sistem fabricat, manufacturat.

Cele mai multe tipuri de defecte fizice pot fi reprezentate prin defecte logice, ale căror efecte asupra comportamentului sistemului sunt bine definite. În baza diferenţei dintre comportamentul în prezenţa unui anumit defect şi comportamentul în absenţa oricărui defect (sau în prezenţa *defectului vid*, cum se mai spune), se poate deduce, se poate deriva, un test pentru acel defect.
Multe modele de defecte logice permit enumerarea defectelor posibile. Existenţa unei mulțimi de defecte enumerabile (mulţimi, mai precis finite, de regulă) permite determinarea calităţii unui test prin calculul *acoperirii defectelor,* care este raportul dintre numărul de defecte detectate prin respectivul test şi numărul total de defecte din model.

În contrast cu cele arătate la erorile de manufacturare, spațiul erorilor de proiectare nu este la fel de bine definit iar mulțimea erorilor de proiectare este ne-enumerabilă. Pe cale de consecinţă, este imposibil de dezvoltat algoritmi pentru generarea testelor, sau să se definească măsuri riguroase privitor la calitatea testelor de verificare a proiectării.

Chiar dacă mulțimea defectelor de proiectare este ne-enumerabilă, experiența a demonstrat că majoritatea erorilor de proiectare sunt legate mai degrabă de secvențierea transferurilor și transformărilor, decât de operațiile propriu-zise asupra datelor. Motivul este că operațiile (aritmetice și logice) asupra datelor sunt mai *locale* și au un grad mai mare de *regularitate*, Aceasta în timp ce controlul și temporizarea acestor operații pot depinde de alte operații care au loc în mod concurent.
Din acest motiv, strategia obișnuită în verificarea proiectării este exercitarea cu precădere a *semnalelor de control* (pe scurt a *controlului*). Cazurile de test minime pentru setul de instrucțiuni ale unui procesor, spre exemplu, constau din executarea fiecărei instrucțiuni din repertoriu. Suplimentar, proiectantul testului trebuie să considere secvențe de instrucțiuni care sunt relevante pentru operarea procesorului, interacțiuni dintre aceste secvențe și întreruperi, ș.a.m.d.

Verificarea proiectării prin simulare este limitată în anumite privințe. Deoarece nu există nici un procedeu formal de producere a testelor, generarea stimulilor se face, în general, printr-un proces euristic ce se bazează în mod special pe intuirea şi înțelegerea, de proiectantul verificării, a sistemului aflat în testare. Un sistem ce trece un test se dovedește a fi corect în raport cu cazurile de test aplicate, obținându-se, în consecinţă, doar dovada unei funcționări, de cele mai multe ori, parțial corecte. Mai mult, completitudinea testelor nu poate fi riguros determinată, în marea majoritate a situaţiilor de interes.

Cu toate aceste limitări, simularea este o tehnica eficientă de verificare a proiectării şi experiența a dovedit că simularea ajută să se descopere cele mai multe dintre erori devreme în procesul de proiectare. Pentru circuitele LSI / VLSI / ULSI, unde erorile de proiectare sunt extrem de costisitoare iar prototipurile sunt nepractice, simularea logică este deosebit de importantă.

Tipurile de simulare

Simulatoarele pot fi clasificate în raport cu tipul modelului intern pe care acestea î1 procesează. Un simulator care execută un model cu cod compilat este referit ca fiind un *simulator cu cod compilat* (*compiler-driven simulator*) sau, pe scurt, *simulator compilat* (*compiled simulator*). Codul compilat este generat dintr-un model LTR, dintr-un model funcțional scris într-un limbaj de programare convențional, sau dintr-un model structural.

Un simulator care interpretează un model bazat pe structuri de date se spune că este un simulator *pilotat* (*condus*) *prin tabele* (*table-driven*). Structurile de date sunt determinate dintr-un model LTR sau dintr-un model structural. Interpretarea modelului este controlată prin stimulii aplicați, aceştia conducând la apelul unor rutine care implementează operatorii primitivi (corespunzător modelor LTR) sau implementează componentele primitive (pentru modelele structurale).
Se consideră un circuit în operare şi se va concentra atenția asupra semnalelor ce-şi schimbă valoarea la un moment dat.

Aceste semnale sunt numite *semnale active.* Raportul dintre numărul de semnale active şi numărul total de semnale dintr-un circuit este numit *activitate.* Activitatea unui circuit, în general, este cuprinsă între 1 şi 5 procente. Acest rezultat pragmatic constituie baza *simulării orientate pe activitate* (*activity-directed simulation*). Această simulare este efectiva numai asupra părţii *active* din circuit.

Un *eveniment* reprezintă o schimbare a valorii semnalului unei linii. Când are loc un asemenea eveniment pe linia *i,* elementele având la intrare linia *i* (acestea mai sunt referite şi prin denumirea de elemente alimentate de linia *i*) sunt *activate.*

Procesul de determinare al valorilor de ieşire ale unui element se numeşte *evaluare.* Simularea orientată pe activitate evaluează numai elementele activate. Anumite elemente activate pot, la rândul lor să-şi schimbe valorile de ieşire, generând astfel noi evenimente. Deoarece activitatea este cauzată de evenimente, simularea orientată pe activitate este deasemenea numită ca fiind *simularea pilotată* (*condusă*) *prin evenimente* (*event-driven simulation*). Pentru a propaga evenimente de-a lungul interconexiunilor trecând prin elemente, un simulator pilotat prin evenimente necesită un model structural al circuitului. Din acest motiv simularea condusă prin evenimente este de regulă pilotată prin tabele.

Simularea compilată este în mod deosebit orientată spre verificarea funcțională şi nu se focalizează asupra temporizării circuitului. Acest aspect face ca această simulare să fie aplicabilă mai mult circuitelor sincrone, pentru care temporizarea se poate verifica separat. Spre deosebire de această simulare, trecerea timpului este un obiectiv central în simulatoarele conduse prin evenimente, unde se lucrează cu modele temporale precise. Astfel, simularea condusă prin evenimente este mai generală ca orizont, fiind deasemenea aplicabilă şi circuitelor asincrone.

Simularea condusă prin evenimente poate procesa *semnale în timp real,* adică poate procesa intrări a căror momente de schimbare sunt independente de activitatea din circuitul simulat. Acest fapt constituie o caracteristică importantă privitor la testarea

verificării proiectării, deoarece permite simularea unor evenimente nesincronizate, cum ar fi întreruperile sau cererile concurente de utilizare ale unor magistrale.

Simularea compilată admite schimbarea intrărilor numai atunci când circuitul este stabil. Aceasta tehnica este adecvată atunci când stimulii de intrare sunt *vectori* aplicați cu frecvență fixă. Se poate remarca faptul că intrările în timp real includ vectorii aplicați cu frecvență fixă, ca un caz particular.

Adesea cele doua tipuri de simulări sunt combinate, astfel încât un algoritm condus prin evenimente propagă evenimente prin componente, iar componentele activate sunt evaluate prin modele cu cod compilat.

Nivelul de simulare corespunde nivelului de modelare folosit pentru reprezentarea sistemul simulat.

Astfel, se poate întâlni:

- *simulare la nivel registru,* pentru sisteme modelate în întregime în LTR sau ca interconexiuni de componente modelate în LTR;
- *simulare la nivel funcțional,* pentru sistemele modelate ca interconexiuni de blocuri funcționale primitive (uneori acest termen este deasemenea folosit atunci când componentele sunt modelate în LTR);
- *simulare la nivel de poartă;*
- *simulare la nivel de tranzistor* (se va considera doar nivelul logic şi nu simularea la nivel analogic a circuitului respectiv);
- *simularea cu nivele mixte.*

Valoarea logică necunoscută

Răspunsul unui circuit cu stări la o secvență de intrare, în general, depinde de starea sa inițială. Totuşi, atunci când un circuit este pus sub tensiune, starea inițială a elementelor cu memorie, cum ar fi bistabilii şi memoriile cu acces aleator, este imprevizibilă.

ŞI	0	1	*u*
0	0	0	0
1	0	0	*u*
u	0	*u*	*u*

SAU	0	1	*u*
0	0	1	*u*
1	1	1	1
u	*u*	1	*u*

NU	0	1	*u*
	1	0	*u*

Figura 2. Tabelele de adevăr pentru logica 3-valorică.

Acesta este motivul pentru care înainte să înceapă operarea normală, se aplica o secvență de inițializare cu scopul să se aducă circuitul într-o stare cunoscută, inițială (*reset*, *homing sequence*). Pentru procesarea stării inițiale necunoscută, algoritmii de simulare folosesc o valoare logică distinctă, notată prin *u,* pentru ca să se indice o *valoare logică necunoscută***.** Valoarea logică *u* este procesată împreună cu valorile logice binare pe durata simulării. Extensia operatorilor Booleeni la logica cu trei valori se bazează pe următorul raționament. Valoarea *u* reprezintă o valoare din mulțimea, setul, {0,1}.
Similar se pot trata valorile 0 şi 1 ca fiind mulțimile {0} şi respectiv {1}. Un operator Boolean *B* între *p* si *q,* unde *p* şi *q* aparțin {0, l, u}, este considerat ca un operator între mulțimile de valori care sunt reprezentate de *p* şi *q,* şi este definit ca mulțimea reuniunii tuturor rezultatelor posibile atunci când se aplică operatorul *B* între

componentele celor două mulţimi. Astfel, se poate proceda din aproape în aproape şi se pot deduce aceste relaţii:

$$\text{ŞI}(0\ ,\ u) = \text{ŞI}(\{0\},\{0,1\}) = \{\ \text{ŞI}(0,0),\ \text{ŞI}(0\ ,\ 1)\} = \{0\ ,\ 0\} = \{0\} = 0.$$

$$\text{SAU}(0,\ u) = \text{SAU}(\{0\},\{0,1\}) = \{\text{SAU}(0,0),\ \text{SAU}(0,1)\} = \{0,1\} = u.$$

Similar, rezultatul unui NU(q), unde q aparţine mulţimii $\{0,1,u\}$, este definit ca fiind reuniunea mulţimilor rezultatelor operatorului NOT asupra elementelor mulţimii care corespunde argumentului q:

$$\text{NU}(u) = \text{NU}(\{0,1\}) = \{\text{NU}(0),\ \text{NU}(1)\} = \{1,\ 0\} = u.$$

Un procedeu general de determinare a valorii unei funcţii combinaţionale $f\,(x_1, x_2, \ldots, x_n)$ pentru o combinaţie dată a intrărilor $(v_1, v_2, \ldots, v_n)$ din valorile posibile 0,1, u, ar fi următoarea:

1. se formează cubul $(v_1, v_2, \ldots, v_n \mid x)$;
2. folosind operatorul intersecţiei ($\cap$) modificat ca în figura 3, se intersectează acest cub cu cuburile primitive ale funcţiei f ; dacă se găseşte o intersecţie consistentă, atunci valoarea funcţiei f este obţinută ca valoarea din extrema dreaptă; altfel $f = u$.

$\cap$	0	1	x	u
0	0	$\varnothing$	0	$\varnothing$
1	$\varnothing$	1	1	$\varnothing$
x	0	1	x	u
u	$\varnothing$	$\varnothing$	u	u

Figura 3. Operatorul intersecţiei ($\varnothing$) modificat.

Pentru înţelegerea acestui procedeu, este util de reamintit că o valoare x într-un cub primitiv denotă o valoare neprecizată. Cu alte cuvinte, o valoare necunoscută este consistentă cu un x dintr-un cub primitiv. O valoare de intrare binară specificată dintr-un cub primitiv este o valoare impusă, aşa că o valoare u ca o valoare de intrare nu poate genera valoarea de ieşire corespunzătoare, totuşi. Pentru o poartă ŞI cu două intrări, spre exemplu, cubul (u 0 | x) se potriveşte unui cub primitiv, dar (u 1 | x) nu se potriveşte nici unui cub primitiv.

Există o pierdere de informaţie asociată folosirii logicii cu trei valori. Acest fapt se poate deduce din tabelul de adevăr al funcţiei de negaţie (NU). În cazul în care atât intrarea cât şi ieşirea au valoarea u, se pierde relaţia de complementaritate dintre acestea.
Acelaşi efect are loc între ieşirile unui bistabil a cărui stare este u. În prezenţa unei ramificaţii reconvergente cu număr inegal de parităţi ale inversiunii, această pierdere de informaţie poate conduce la rezultate excesiv de pesimiste. O astfel de situaţie este ilustrată în figura 4 unde ieşirea porţii ŞI-NU este actualmente 1, dar aceasta este calculată ca fiind u potrivit regulilor logicii cu trei valori (aşa cum se poate vedea din figura 3).

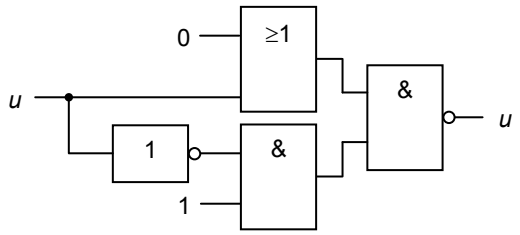

Figura 4. Rezultat pesimist într-o simulare 3-valorică.

Aparent folosirea valorilor necunoscute complementare *u* si *u′,* împreună cu regulile $uu' = 0$ şi $u+u' = 1$, ar rezolva problema. Acest fapt e adevărat, însă numai pentru cazul în care apare doar o singură variabilă de stare cu valoarea *u.*
În figura 5 se ilustrează modul în care folosirea valorilor *u* şi *u'* poate să conducă, totuşi, la rezultate incorecte.
Întrucât este mai bine ca o soluţie să fie pesimistă decât incorectă, folosirea valorilor *u* şi *u′* nu este recomandată ca soluţie satisfăcătoare. O soluţie corectă ar fi folosirea câtorva semnale necunoscute distincte $u_1, u_2, ..., u_k$ (câte una pentru fiecare variabilă de stare) şi cu regulile:

$$u_i.u_i' = 0,\ u_i+u_i'=1, \text{ pentru } i = 1, 2, ..., k.$$

Din nefericire această tehnică devine greoaie pentru circuitele mari, deoarece valorile anumitor linii ar fi reprezentate de lungi expresii Booleene în variabilele u_i .

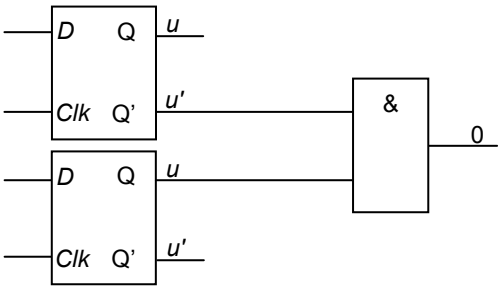

Figura 5. Rezultat incorrect, urmare a utilizării valorilor *u* şi *u'*.

Operarea unui element funcţional este determinata, adesea, prin decodificarea valorilor unui grup de linii de control. Apare o problemă în simulare atunci când un element funcţional trebuie să fie evaluat şi unele dintre liniile sale de control au valori *u*.
Dacă *k* linii de control au valori *u,* elementul poate executa una din cele 2^k operaţii posibile, în genere.

O soluţie precisă (dar potenţial costisitoare) ar fi realizarea tuturor celor 2^k operaţii şi să se ia ca rezultat reuniunea mulţimii rezultatelor respective. Astfel, dacă o variabilă ia valoarea 0 în anumite operaţii şi 1 în altele, valoarea sa rezultantă va fi $\{0,1\} = u$. Bine înţeles, aceasta soluţie este practică doar dacă 2^k reprezintă, totuşi, o valoare mică.

Într-un circuit asincron, prezenţa valorilor *u* poate indica o oscilaţie, aşa cum se poate vedea din figura 6.

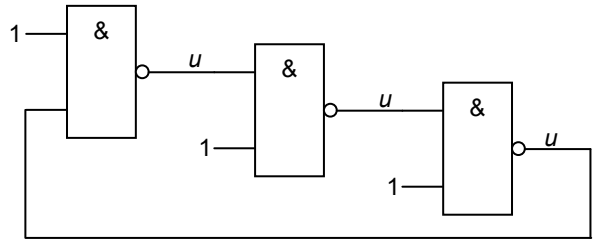

Figura 6. Valori *u* care arată prezenţa unor oscilaţii.

Semnale implicate într-o oscilaţie de frecvenţă ridicată, adesea iau valori între nivelele corespunzând valorilor logice 0 şi 1.
Astfel, suplimentar valorii statice necunoscute, *u* poate să semnifice deasemenea o valoare dinamică necunoscută sau o valoare logică nedeterminată.

Simularea compilată

În simularea compilată modelul cu cod compilat devine parte integrantă a simulatorului. În cazul simplificat (extrem) simulatorul este chiar modelul prin cod compilat.
Astfel, acest cod citeşte vectorii de intrare şi scrie rezultatele. În general, modelul cu cod compilat este legat (în sensul de editării legăturilor) cu nucleul simulatorului, în ale cărui sarcini sunt incluse citirea vectorilor de intrare, executarea modelului pentru fiecare vector în parte şi extragerea rezultatelor corespunzătoare.
Se va ilustra operarea unui simulator compilat folosind circuitul din figura 7. Acest circuit este sincron, controlat de semnalul de ceas *CLK*. Se presupune că după ce un nou vector de intrare este aplicat, există suficient timp pentru ca datele de intrare ale bistabilului să devină stabile cu cel puţin t_{setup} timp înainte ca bistabilul sa fie acţionat, unde t_{setup} este timpul de stabilire al intrărilor în bistabil. Această ipoteză poate fi verificată independent, printr-un program de verificare a temporizărilor. Dacă ipoteza este validă, atunci simularea poate ignora întârzierile individuale ale porţilor, ca şi cum temporizarea exacta a semnalelor în logica combinaţională nu ar mai avea importanţă. În continuare, pentru fiecare vector, simularea necesită doar sa calculul valorii statice a semnalului *F* şi să transfere această valoare liniei *Q*.

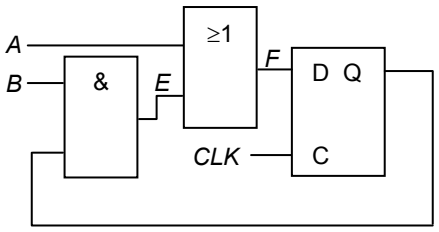

Figura 7. Circuit sincron.

Modelul cod este generat astfel încât calculul valorilor să urmeze nivel, după nivel. Aceasta asigură faptul că ori de câte ori codul evaluează o poartă, porţile ce alimentează respectiva poartă au fost deja evaluate. Valorile intrărilor primare *A* şi *B* (care au nivel 0) sunt citite dintr-un fişier de stimuli.
Singurul alt semnal de nivel 0 este variabila de stare *Q*; întâi se presupune că valoarea sa iniţială este cunoscută. Atunci simulatorul trebuie sa proceseze numai valori binare, care sunt stocate în variabilele *A, B, Q, E* şi *D*.
Iată cum ar arăta modelul în cod de asamblare al circuitului:

```
LDA   B
AND   Q
INV
STA   E
OR    A
STA   F
STA   Q
```

Este de remarcat faptul că simulatorul compilat evaluează toate elementele din circuit pentru fiecare vector de intrare.
Dacă valoarea iniţiala a liniei *Q* este necunoscută, atunci simulatorul trebuie sa proceseze valorile 0,1 şi *u*.
Aceste valori sunt codificate prin vectori cu câte doi biţi fiecare:

0 - 00
1 - 11
u - 01

Se poate uşor observa ca s-au ales codificările astfel încât o operaţie AND (OR) între vectorii de doi biţi este corect calculată prin operaţia AND (OR) între biţi (consideraţi individual).
Operaţia NOT nu poate fi făcută, totuşi, numai prin complementarea biţilor, deoarece complementul valorii *u* ar conduce la codul ilegal 10. Pentru NOT, soluţia este ca după complementare să se inter-schimbe între ei cei doi biţi.

Se consideră evaluarea unei porţi ŞI cu intrările *A* şi *B* şi ieşirea *C*, prin operaţia $C = A \cdot B$, unde „·” reprezintă instrucţiunea maşină a calculatorului pe care se face simularea, numit pe scurt calculatorul gazdă.

Dacă evaluarea se mărgineşte la logica bi-valorică, este nevoie de numai un singur bit pentru reprezentarea valorii unui semnal. În *evaluarea paralelă* („*parallel-pattern evaluation*”) se foloseşte o locaţie de memorie cu *W* biţi, a calculatorului gazdă pentru memorarea valorilor aceluiaşi semnal din *W* vectori diferiţi.

Dacă instrucţiunea „·” lucrează pe operanzi cu *W* biţi, atunci $C = A \cdot B$ calculează simultan valorile pentru semnalul *C* în *W* vectori. Acest lucru este valabil, bineînţeles, numai în circuitele combinaţionale în care ordinea de aplicare a vectorilor este lipsită de relevanţă. Această metodă grăbeşte simularea compilata bi-valorică cu un factor egal cu *W* (tipic *W* este 32). Pentru logici tri-valorice, doar W/2 vectori pot fi simulaţi concurent.

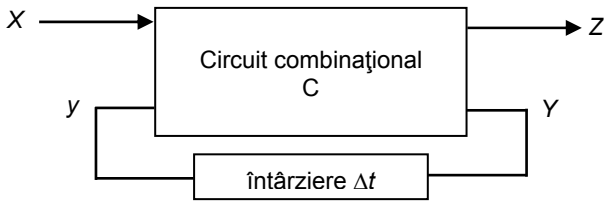

Figura 8. Modelul unui circuit secvenţial asincron.

Simularea compilata poate fi folosită deasemenea pentru circuite asincrone, folosind modelul arătat în figura 8, ceea ce presupune ca întârzierile sunt prezente doar pe liniile

de reacţie. Ca răspuns la un vector de intrare *X*, circuitul poate sa treacă printr-o serie de tranziţii de stare, reprezentate de schimbările variabilelor de stare *y*. Se presupune că vectorul de intrare este aplicat numai atunci când circuitul este stabil, adică atunci când *y* = *Y*. Liniile de reacţie, care au nivel 0, trebuie sa fie identificate înainte ca modelul cod pentru circuitul combinaţional *C* să fie generat. În figura 9 se prezentat schematic procedeul general pentru simularea unui circuit asincron. Execuţia modelului calculează valorile liniilor *z* şi *Y* în baza valorilor *X* şi *y*.

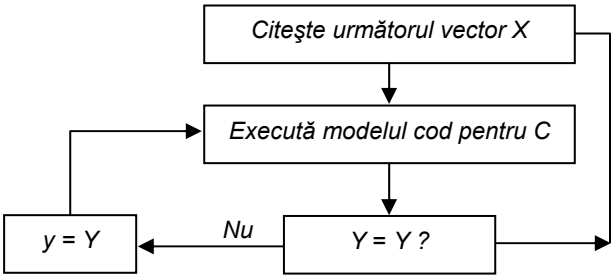

Figura 9. Simularea unui circuit asincron
cu model cod compilat.

Acest tip de simulare nu este precis pentru circuitele asincrone a căror operare este bazată pe anumite valori de întârziere. Circuitul din figura 10(a), spre exemplu, poate fi folosit ca generator de impulsuri.
Dar, fără o modelare atentă, acest impuls nu poate fi realizat de un simulator compilat, care tratează numai comportamentul static al circuitului (static semnalul *C* este întotdeauna 0).
Pentru a lua în consideraţie întârzierea semnalului *B*, care este esenţiala pentru operaţia intenţionată a circuitului, semnalul *B* trebuie tratat ca o linie de reacţie, aşa cum este arătat în figura 10(b). Cu acest model, circuitul poate fi corect simulat.

De remarcat faptul că deducerea unui astfel de model *corect* nu poate fi făcută automat, cerând totuşi intervenţia utilizatorului.

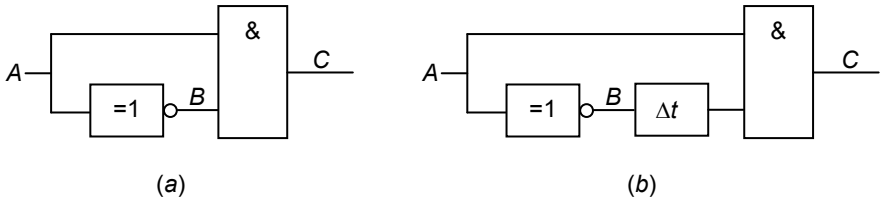

Figura 10.
(a) Circuit utilizat ca generator de impuls.
(b) Modelul corect pentru simularea compilată.

Simularea pilotată (condusă) prin evenimente

Un simulator pilotat sau condus (cei doi termeni vor fi folosiţi alternativ) foloseşte un model structural al unui circuit pentru propagarea evenimentelor. Schimbările valorilor intrărilor primare sunt definite în fişierul de stimuli.

Evenimente pe alte linii sunt produse prin evaluări ale elementelor activate. Un eveniment are loc la un anumit moment (simulat).

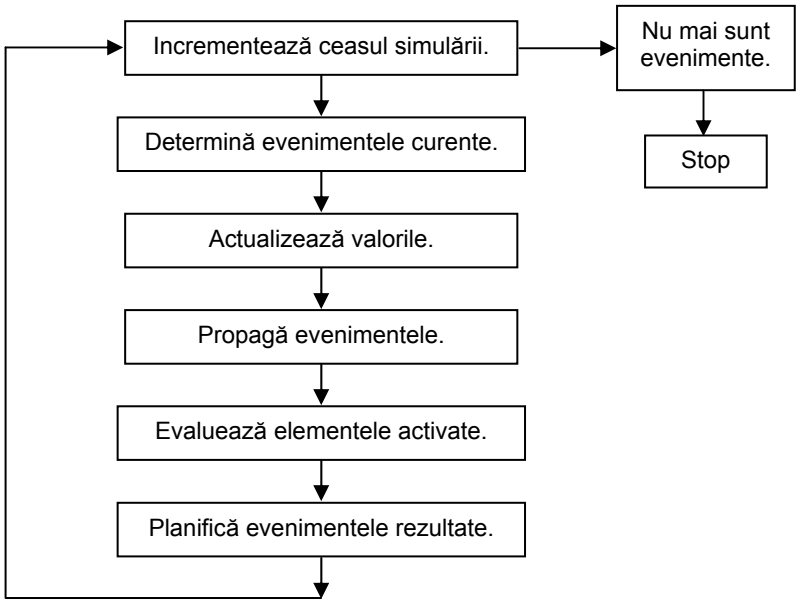

Figura 11. Fluxul principal al simulării conduse prin evenimente.

Simularea *mecanismului de scurgere a timpului* manipulează evenimentele astfel încât acestea vor apare într-o ordine cronologică corectă. Stimulii aplicaţi sunt reprezentaţi prin secvenţe de evenimente a căror momente sunt predefinite.
Evenimentele planificate să apară într-un viitor apropiat (relativ la momentul de timp curent simulat) se spune că sunt în aşteptare, suspendate, şi sunt menţinute într-o structură de date numită *lista evenimentelor*.

În figura 11 este descris principalul flux al simulării conduse prin evenimente. Timpul simulat este avansat la momentul următor atâta vreme cât sunt elemente suspendate; acest moment devine timpul de simulare curent.
În continuare simulatorul recuperează din lista de evenimente, evenimentele prevăzute sa aibă loc la momentul curent şi actualizează valorile semnalelor active.
Lista de răspândire („*fanout*") a semnalelor este apoi urmărită pentru determinarea elementele activate; acest proces paralelizează propagarea schimbărilor din circuitul real.
Evaluarea elementelor activate poate conduce la noi evenimente. Aceste evenimente sunt prevăzute, planificate, să apară într-un moment viitor în raport cu întârzierile asociate cu operarea elementelor.
Simulatorul înserează noile evenimente generate în lista evenimentelor. Simularea continuă atâta vreme cât exista activitate logică în circuit; cu alte cuvinte până când lista de evenimente devine vidă.

Evaluarea unui element M modelat printr-un LTR neprocedural poate genera *un eveniment stare* ce denota o schimbare în valoarea unei variabile interne de stare a elementului M. (În LTR procedurale, schimbările variabilelor de stare apar imediat, fără vre-o prelucrare prin lista de evenimente.)
Când un astfel de eveniment de stare are loc, acesta activează numai elementul M care l-a generat; adică are drept efect reevaluarea elementului M.

Pentru simplitate, în descrierea anterioara s-a presupus că toate evenimentele ce definesc stimulii aplicaţi au fost înserate în lista evenimentelor înainte de simulare. În fapt, simulatorul trebuie sa citească periodic fişierul de stimuli şi să adauge evenimentele intrărilor primare evenimentelor generate intern.

În afara de evenimentele ce poartă actualizări ale valorilor semnalelor, un simulator condus prin evenimente poate deasemenea procesa *evenimente de control*, ceea ce oferă o manieră convenabilă de iniţiere a activităţilor diferite la anumite momente. Acţiunile tipice legate de evenimentele de control sunt:

- extragerea, tipărirea, valorilor unor anumite semnale;
- verificarea valorilor estimate pentru anumite semnale (şi, posibil, să oprească simularea dacă se dovedeşte o nepotrivire între valorile estimate şi cele obţinute pentru respectivele semnale);
- încetarea simulării.

Simularea logică este apreciată, alături de alte domenii ale ingineriei calculatoarelor, ca fiind *state-of-the-art*.
Actualul context impune verificări ale proiectării cu complexităţi mult crescute comparativ cu perioada timpurie a tehnologiilor submicronice.

Modele ale întârzierilor

Există mai multe variante diferite ale fluxului de evaluare descris în figura 11. Diferenţele rezidă, în principiu, din modelele de întârziere asociate comportamentului componentelor din modelul respectiv.
Modelele întârzierilor prin porţi joacă un rol esenţial. Acestea cuprind o seamă de aspecte dominante ale funcţionării porţilor:
Întârzierea de transport – orice poartă introduce o întârziere a semnalelor fizice ce se propagă prin respectiva poartă.
Întârzierea inerţială – toate circuitele necesită energie pentru ca să comute, să-şi schimbe starea.

Evaluarea elementelor

Evaluarea unui element combinaţional cuprinde procesul de calcul prin care se evaluează ieşirea acestuia având ca intrări valorile curente ale liniilor sale de intrare. Cu cât rutinele de evaluare sunt mai rapide cu atât este mai performantă simularea acelui element de circuit.
Una dintre cele mai răspândite metode de evaluare este utilizarea *Tabelelor de Adevăr*. Presupunând doar valori binare ale procesului de simulare, pentru un element evaluat cu n linii de intrare se va utiliza un tabel cu 2^n linii. Componentele unei intrări din tabelele de adevăr reflectă valori ale liniilor de ieşire şi stări (depinzând de nivelul de simulare ţintit). Mecanismele implementate prin tabele de adevăr permit introducerea unor modele logice multi-valorice corespunzătoare unor modele de mare fineţe capabile să detalieze funcţionarea la nivelul unor fracţiuni de cuante de timp pe durata simulării.

Detecţia hazardurilor

Hazardurile statice impun analiza comportamentului tranzitoriu pentru semnalele prezente la intrarea unui element de circuit simulat.
Divizarea intervalului de timp pe durata căruia se efectuează evaluarea liniilor de intrare şi, corespunzător modelului evoluţia liniei de ieşire a elementului de circuit evaluat, joacă rolul central în simularea hazardurilor statice.
Hazardurile dinamice pun în evidenţă apariţia unor pulsuri tranzitorii ale semnalului la ieşirea unei porţi (ori alt element de circuit simulat) pe durata evaluării respectivei linii.
Sunt mult utilizate semnale modelate prin logici cu valori multiple (utilizând patru biţi, spre exemplu) capabile să surprindă varietatea de fenomene tranzitorii dinamice.
Cele mai dificile circuite sunt cele asincrone, dar şi cele mai performante. Există numeroase arhitecturi logice care-şi cresc performanţele printr-o organizare mixtă, comportament global asincron dar local sincron, prin mecanisme cu semnale de ceas multiple asociate fiecărei subsecţiuni funcţionale. Astfel de arhitecturi necesită simulări capabile să surprindă atât din punctul de vedere al variaţiei tehnologice cât şi al multitudinii de cazuri logice funcţionale, toate situaţiile care pot interveni în dinamica utilizării respectivului dispozitiv.
Proiectarea logică este de cele mai multe ori pilotată prin astfel de simulări capabile să valideze soluţiile constructive cu cele eficiente costuri ulterioare (măştile de integrare, procesele tehnologice de metalizare etc.).