SINTEZA CIRCUITELOR NUMERICE CU DISPOZITIVE PROGRAMABILE DE TIP FPGA

1. Scop

Se prezintă metode de sinteză specifice proiectării circuitelor numerice cu dispozitive FPGA. Se prezintă strategiile de implementare a componentelor elementare în dispozitivele FPGA XILINX. Se studiază modul de implementare a unui numărător Moebius specificat cu editorul schematic. Se prezintă principalele reguli de specificare a proiectelor în vederea sintezei lor în dispozitivele FPGA XILINX.

2. Considerații teoretice

Metodele de sinteză a circuitelor numerice nu diferă în cazul proiectării cu dispozitive FPGA de metodele clasice. Dimpotrivă, datorită specificului tehnicilor de proiectare cu aceste dispozitive (utilizarea instrumentelor software de susținere a proiectării), implementarea circuitelor ar trebui să fie mai simplă.

Totuşi, în cazul în care dorim să configurăm manual cipul FPGA (pentru a economisi timp de rulare a programelor software, pentru a atinge anumite obiective specifice sau din alte motive), este indicat să ținem seama de anumite tehnici de proiectare cu dispozitivele FPGA.

2.1 Strategii de proiectare cu FPGA

Reamintim că un circuit secvențial sincron este alcătuit din două părți principale:

- partea strict secvențială (bistabilele);
- partea combinațională (porți logice elementare) care contribuie în mare măsură la determinarea stării următoare a circuitului.

De aceea trebuie să discutăm despre tehnicile de proiectare cu FPGA-uri a principalelor elemente de circuit, atât a celor secvențiale cât şi a

celor combinaționale. Aceste tehnici sunt strict dependente de tipul de FPGA utilizat. În cele ce urmează vom discuta despre FPGA-urile XILINX.

Proiectarea cu LCA-ul necesită înțelegerea posibilităților şi limitărilor specifice *slice*-urilor şi CLB-urilor. Fiecare *slice* are o capacitate funcțională şi de stocare a datelor limitată, iar noi trebuie să determinăm *cum* pot fi folosite aceste capacități. Vom studia posibilitățile de construire a decodificatoarelor, multiplexoarelor, registrelor de deplasare şi a numărătoarelor (se subînțelege că implementarea codificatoarelor şi demultiplexoarelor se realizează analog cu cea a decodificatoarelor şi respectiv a multiplexoarelor).

Decodificatoarele se implementează prin cascadarea RAM-urilor funcționale ale *slice*-urilor, care sunt configurate fie ca funcții ŞI fie ca funcții ŞI-NU. Pentru a construi un decodificator 3-la-8 standard (cum ar fi 4138) sunt necesare cel puțin 8 *slice*-uri (care au 4 intrări şi o ieşire) numai pentru a genera ieşirile distincte pentru toate cele 8 combinații.

Multiplexoarele, ca şi decodificatoarele, sunt construite în interiorul *slice*-urilor configurând în mod adecvat RAM-urile interne ale acestora. Din nou, numărul limitat al intrărilor în RAM-uri necesită expandarea numărului de *slice*-uri pentru a forma funcții de mai multe intrări. Problemele care apar aici se referă la faptul că această abordare consumă rapid *slice*-urile şi scade viteza funcțiilor. În plus, bistabilele din *slice*-uri rămân de multe ori neutilizate.

Registrele de deplasare sunt eficient de implementat în *slice*-uri. Atribuind biții unor *slice*-uri sau CLB-uri adiacente, registrele de deplasare nu necesită interconectare globală şi nu blochează canalele de rutare. Clasa de aplicații care utilizează din plin registre de deplasare cuprinde echipamente de comunicații, dispozitive de recunoaştere a formelor, generatoare polinomiale etc. La fel cum decodificatoarele şi multiplexoarele lasă neutilizate bistabilele, registrele de deplasare nu folosesc blocurile logice combinaționale.

Implementarea *numărătoarelor* reprezintă o problemă interesantă în arhitecturile LCA. Metodele clasice sunt aplicabile numai pentru numărătoare mici. Partea combinațională din alcătuirea unui numărător (cea care determină starea următoare, deci funcțiile de tranziție) creşte foarte mult, în termeni de intrări necesare pentru porțile logice, pe măsură ce creşte numărul de biți (ordinul) numărătorului. Acest tip de abordare limitează prea mult posibilitățile LCA-ului, aşa că XILINX recomandă pentru numărătoare o abordare modulară şi cascadabilă. În această abordare, fiecare bistabil din numărător are în față un acelaşi tip de circuit logic combinațional. Vom numi un bistabil, împreună cu funcțiile de tranziție proprii, o „celulă de

numărare". Fiecare celulă de numărare semnalează celulei următoare faptul că şi-a încheiat ciclul de numărare, astfel fiind posibilă cascadarea acestor celule de numărare. Numărul de biți ai numărătorului este egal cu numărul de celule de numărare. Începând cu seria 4000, funcțiile de tranziție pot avea până la 4 intrări. Principalul neajuns al acestei abordări este viteza redusă. O plasare atentă a celulelor minimizează totuşi timpul de propagare al semnalelor de la o celulă de numărare la următoarea.

2.2 Introducerea proiectului prin editor schematic

În cazul introducerii proiectului prin editor schematic este necesară efectuarea unei sinteze manuale prealabile a proiectului. Se va utiliza apoi editorul schematic din pachetul ISE Foundation, pentru introducerea efectivă a proiectului. De exemplu, circuitul descris în schema din figura 13.1 reprezintă un numărător Moebius pe 4 biți:

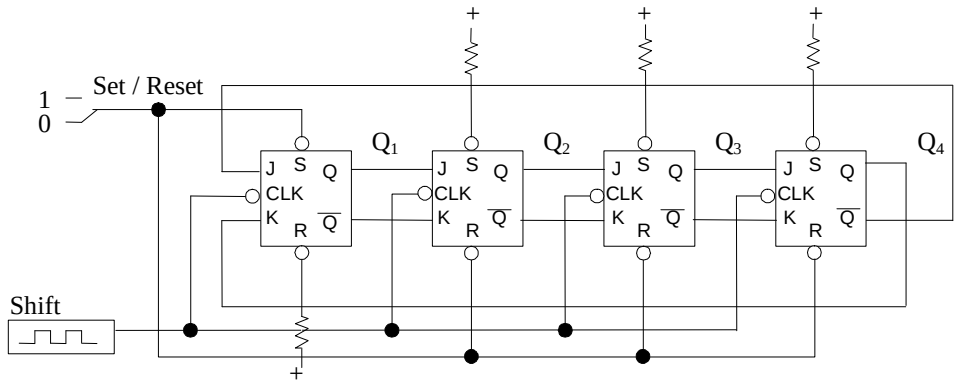

Figura 13.1 *Numărător Moebius pe 4 biți*

Acest numărător va avea formele de undă conform figurii 13.2:

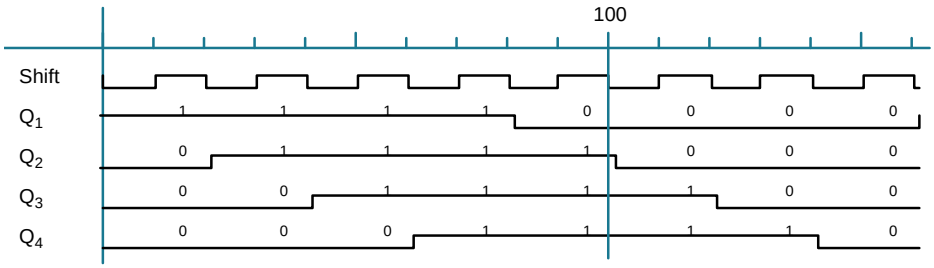

Figura 13.2 *Formele de undă ale numărătorului Moebius*

După editarea proiectului, acesta va fi simulat cu ajutorul simulatorului din ISE Foundation şi se vor obține formele de undă prezentate în figura 13.2. În continuare, putem parcurge celelalte etape din fluxul de proiectare, conform celor prezentate în Lucrarea 12.

2.3 Reguli şi metode generale pentru sinteza circuitelor numerice în FPGA

În cele ce urmează vom prezenta câteva reguli şi metode cu caracter general, dar extrem de utile pentru sinteza proiectelor digitale pentru dispozitive FPGA (şi nu numai). Aceste reguli şi metode permit creşterea performanței proiectului şi chiar a sistemului în care este integrat cipul FPGA gazdă.

a) Duplicarea bistabilelor

Există situații când un singur bistabil comandă foarte multe alte componente din sistem (figura 13.3). În acest caz, firul fn_1 are un fanout foarte mare.

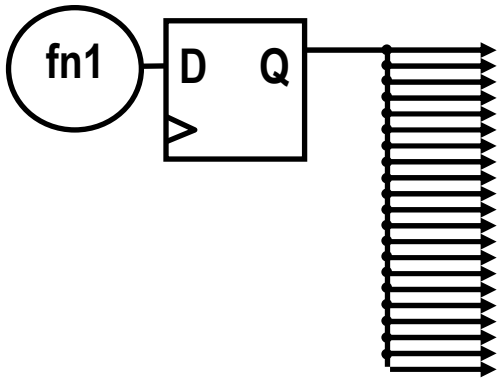

Figura 13.3 *Fir cu* fanout *mare: un singur bistabil comandă un număr foarte mare de alte resurse*

Firele care au un *fanout* mare sunt mai lente (semnalele se propagă mai lent) şi mai dificil de rutat în cip. Duplicarea bistabilelor poate rezolva ambele probleme:

- Reducerea fanout-ului micşorează întârzierile de propagare a semnalelor prin fir;

- Fiecare bistabil poate direcționa semnalul produs de el către regiuni fizice diferite ale cipului, reducându-se astfel congestia căilor de rutare.

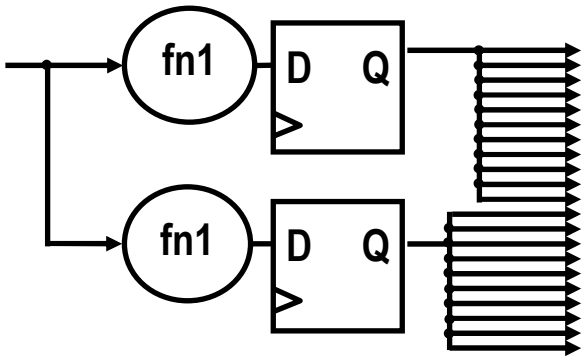

Figura 13.3 *Duplicarea bistabilului reduce* fanout*-ul şi congestia căilor de rutare*

Principalele rezultate ale aplicării acestei tehnici:

- Creşte rutabilitatea şi performanța;
- Creşte suprafața ocupată de către proiect în cip (logica activă + rețeaua de interconectare).

Se recomandă ca bistabilele duplicate să fie notate cu „_a", „_b" etc., nu „_1", „_2", deoarece bistabilele numerotate sunt mapate în acelaşi *slice* în mod implicit. Bistabilele duplicate trebuie să fie separate, mai ales dacă semnalele pe care le transmit sunt distribuite în regiuni diferite ale cipului.

Se recomandă ca bistabilele duplicate să fie create duplicate în codul HDL, deoarece deşi majoritatea instrumentelor de sinteză controlează în mod automat *fanout*-ul, ele nu determină întotdeauna diviziunea optimă a sarcinilor. În plus, aceste instrumente vor denumi bistabilele duplicate „_1", „_2" etc.

Trebuie de asemenea să setăm instrumentele de sinteză astfel ca să păstreze logica redundantă (de exemplu, se foloseşte atributul KEEP).

Nu trebuie duplicate bistabilele care primesc date de la semnale asincrone. Mai întâi semnalul respectiv trebuie sincronizat şi abia apoi semnalul sincronizat va fi distribuit bistabilelor duplicate.

b) Pipelining

Tehnica de *pipelining* are ca scop creşterea vitezei sistemelor de calcul. Regula generală este de a se introduce cel mult două niveluri de

logică combinațională între registrele tampon din interiorul sistemului – situația este ilustrată în figura 13.4.

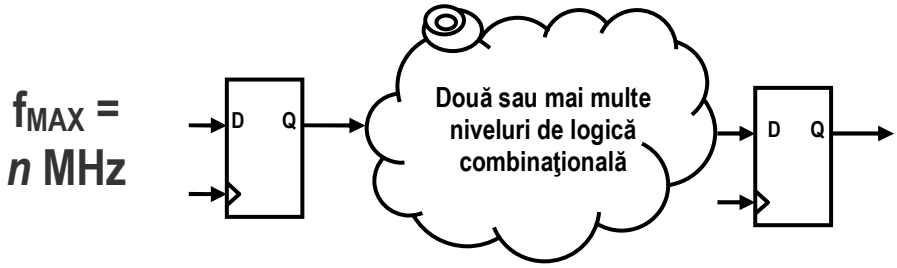

Figura 13.4 *Structura generală a unui sistem numeric*

Principalul factor care limitează viteza sistemului este, în acest caz, întârzierea de propagare a semnalelor prin blocul de logică combinațională. Cu cât numărul nivelurilor de logică combinațională dintre bistabile crește, cu atât viteza sistemului scade. De aceea, ideal pentru performanță ar fi ca numărul nivelurilor de logică combinațională dintre bistabile să fie minim (adică egal cu 1). Acest deziderat se poate atinge prin „spargerea" logicii combinaționale în blocuri mai mici între care se vor intercala alte registre tampon, ca în figura 13.5.

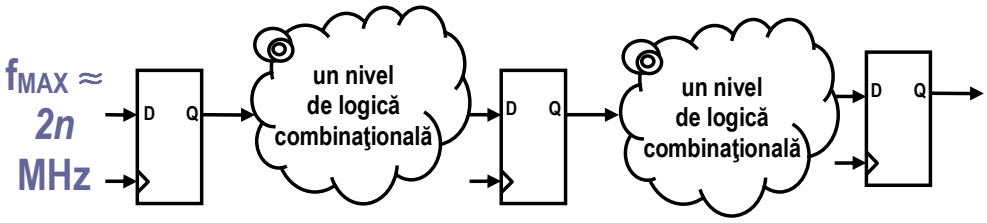

Figura 13.5 *Crearea unui* pipeline *cu două etaje pentru creşterea vitezei*

Atunci când recurgem la *pipelining*, trebuie să verificăm anumite aspecte.

În primul rând, trebuie să determinăm câte niveluri de logică combinațională există între bistabile în proiectul nostru. Dacă există un singur nivel logic între bistabile, tehnica de *pipelining* nu va îmbunătăți performanța. Acest fapt se poate determina citind raportul *Post-Map Static Timing Report* sau raportul *Post-Place & Route Static Timing Report*.

În al doilea rând, trebuie să ne întrebăm dacă avem la dispoziție în cipul FPGA suficiente bistabile pentru implementare. Acest fapt se poate

determina citind raportul produs de utilitarul MAP (MAP Report). În general, numărul de bistabile disponibile este suficient.

Următoarea problemă pe care trebuie să ne-o punem este aceea dacă sistemul poate tolera *latența*. Fiecare etaj al *pipeline*-ului introduce o întârziere egală cu o perioadă a impulsului de tact, înainte ca prima ieşire corectă să fie disponibilă. Acest fenomen se numeşte „umplerea *pipeline*-ului”. După ce *pipeline*-ul este „plin”, din acel moment încolo este disponibilă câte o nouă ieşire în fiecare ciclu de tact. Această problemă este ilustrată grafic în figura 13.6.

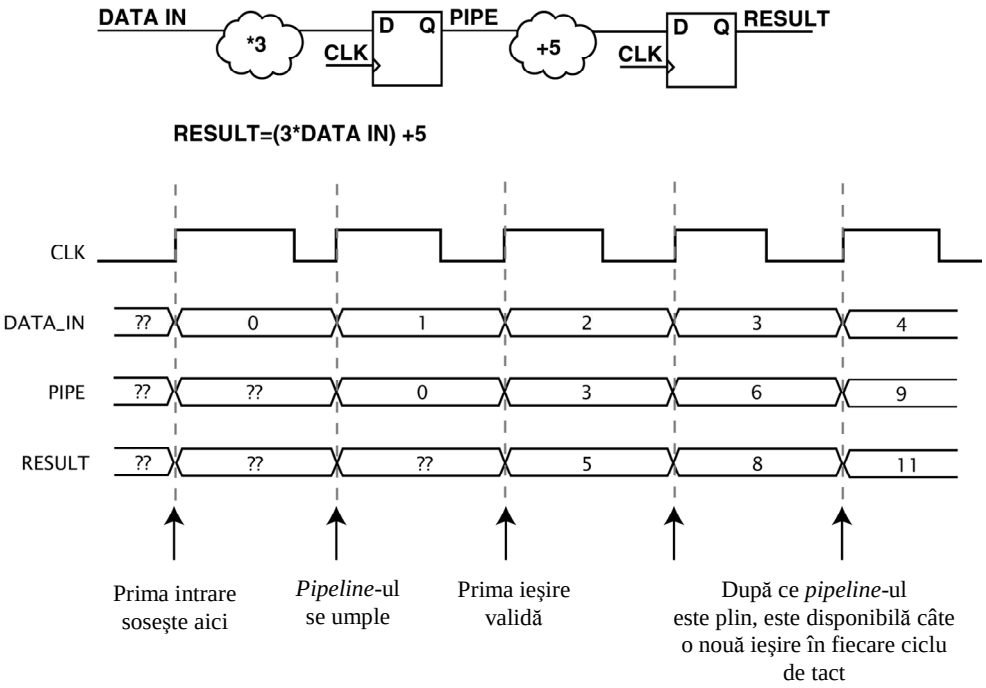

Figura 13.6 *Problema latenței în* pipeline

Un exemplu de *pipeline* este cel din figura 13.7 a). Se observă că există două niveluri de logică combinațională între bistabilul sursă şi cel destinație. Frecvența operațională este de aproximativ 233 MHz.

Prin introducerea unui nivel suplimentar de bistabile tampon, se creează un pipeline cu două etaje a cărui frecvență operațională ajunge la 385 MHz (figura 13.7 b)).

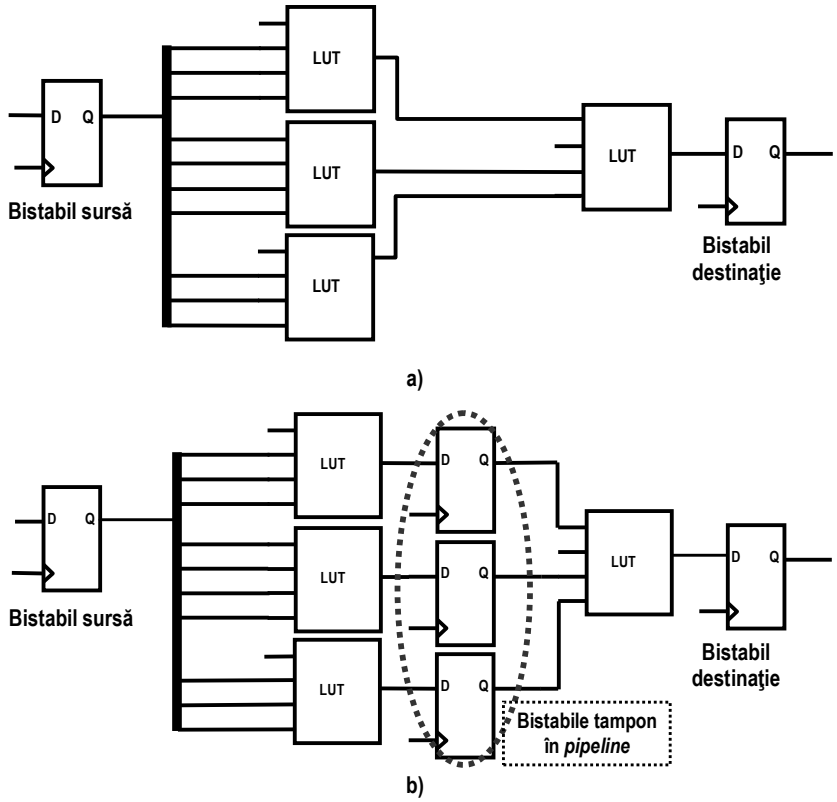

Figura 13.7 *a) Circuitul inițial cu două niveluri de logică combinațională între bistabilul sursă şi cel destinație; b) Circuitul* pipeline*-izat*

c) Folosirea bistabilelor din blocurile de intrare / ieşire

Fiecare bloc de intrare / ieşire (IOB) din dispozitivele FPGA XILINX conține mai multe bistabile (de regulă, 6 bistabile). Blocul IOB a fost prezentat în detaliu în Lucrarea 12.

Bistabilele din IOB-uri oferă valori garantate ale timpilor de *setup*, *hold* şi *clock-to-out*, atunci când semnalul de tact (*clock*) provine dintr-un *buffer* BUFG (Global Clock Buffer, o componentă primitivă din cipurile FPGA cu rol de repetor şi distribuitor al semnalului de tact în interiorul cipului). Folosirea acestor bistabile este foarte importantă mai ales în cazul blocurilor logice care realizează interfațarea cipului FPGA cu alte cipuri din sistemul în care este integrat. De asemenea, uneori – în cazul proiectelor de mari dimensiuni – bistabilele din *slice*-uri nu sunt suficiente, fapt pentru care este obligatoriu să se recurgă şi la cele din IOB-uri.

De regulă, instrumentele de sinteză tind să aloce bistabilele din *slice*-uri, nu din IOB-uri. Pentru a obține utilizarea bistabilelor din IOB-uri în decursul procesului de sinteză, putem recurge la specificarea unor constrângeri temporale care să forțeze plasarea bistabilelor în IOB-uri. Există instrumente de sinteză care oferă posibilitatea de a da valori unor atribute sau unor directive de sinteză cu ajutorul cărora bistabilele vor fi plasate într-un IOB.

De exemplu, în utilitarul Xilinx Constraint Editor trebuie selectat tab-ul **Misc** şi apoi se vor specifica registrele care ar trebui să fie plasate în IOB-uri. Este necesar să cunoaştem numele (eticheta) fiecărei instanțe de registru. Registrele fiind alcătuite din bistabile, acestea vor ocupa unul sau mai multe IOB-uri.

În timpul etapei MAP a procesului de implementare, în cutia de dialog „Map Properties”, opțiunea „Pack I/O Registers/Latches into IOBs” este selectată implicit. Folosirea constrângerilor temporale va avea de asemenea ca efect plasarea registrelor în IOB-uri, în cazul căilor critice.

În final, vom verifica raportul utilitarului MAP pentru a avea confirmarea faptului că au fost utilizate bistabilele din IOB-uri (în secțiunea „IOB Properties”).

d) Circuite de sincronizare şi metastabilitatea

Circuitele de sincronizare au rolul de a capta un semnal de intrare asincron şi de a-l transmite mai departe doar pe frontul ascendent sau descendent al semnalului de tact.

Aceste circuite sunt necesare pentru a preveni violarea timpilor de *setup* şi de *hold* ale bistabilelor şi pentru a asigura astfel siguranța şi fiabilitatea proiectului.

Circuitele de sincronizare sunt necesare atunci când cipul are intrări asincrone sau atunci când în cadrul aceluiaşi proiect există zone care funcționează sincron cu semnale de tact diferite şi total neînrudite (între care nu există nici o corelație), dar care schimbă informație prin intermediul unor semnale. Aceste semnale interzonale trebuie sincronizate cu ajutorul unor circuite de sincronizare. Dacă între semnalele de tact există o corelație, atunci nu este nevoie de nici un circuit de sincronizare (se pot folosi constrângeri de tip PERIOD impuse asupra semnalelor de tact, care vor rezolva toate problemele potențiale).

Violările timpilor de setup şi de hold constituie o problemă serioasă şi nepredictibilă în cazul intrărilor asincrone. Ele apar atunci când intrarea

de date a bistabilului se modifică într-un moment temporal prea apropiat de frontul tactului. Aceasta poate avea drept rezultat trei situații (figura 13.8):

- Bistabilul va fi înscris cu vechea valoare de pe intrarea sa de date;
- Bistabilul va fi înscris cu noua valoare de pe intrarea sa de date;
- Ieşirea bistabilului devine *metastabilă*.

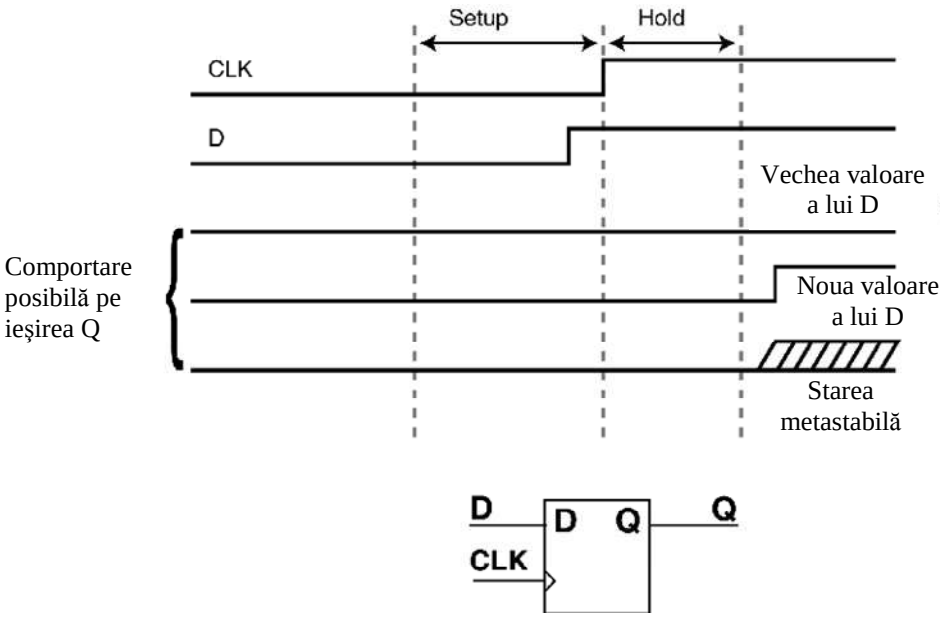

Figura 13.8 *Rezultatele posibile ale violării timpilor de setup şi de hold*

Metastabilitatea înseamnă că bistabilul intră într-o stare tranzitorie care nu este nici „0”, nici „1”, adică anumite circuite o pot interpreta drept „0”, iar altele o pot interpreta drept „1”. Bistabilul rămâne în această stare o perioadă de timp nepredictibilă, dar în cele din urmă se va stabiliza fie în starea „0”, fie în starea „1”.

Din motive statistice, apariția evenimentelor metastabile poate fi doar redusă, nu eliminată complet. Există un parametru funcțional care se numește Timpul Mediu Între Căderi (*Mean Time Between Failure* – MTBF) şi care depinde în mod exponențial de mărimea intervalului de timp acordat bistabilului pentru a-şi „reveni” din starea metastabilă. Dacă îi acordăm bistabilului câteva nanosecunde suplimentare pentru a-şi „reveni”, aceasta va reduce extrem de mult şansele apariției unui eveniment de acest tip. Circuitele din figurile 13.9 şi 13.10 permit acordarea unui „timp de recuperare” egal cu durata unui ciclu de tact complet.