# LUCRAREA NR. 13

## SINTEZA CIRCUITELOR NUMERICE CU DISPOZITIVE PROGRAMABILE DE TIP FPGA

### 1. Scopul lucrării

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. Consideraţii 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 susţinere 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 ţinem seama de anumite tehnici de proiectare cu dispozitivele FPGA.

#### 2.1 Strategii de proiectare cu FPGA

Reamintim că un circuit secvenţial sincron este alcătuit din două părţi principale:

- partea strict secvenţială (bistabilele);
- partea combinaţională (porţi 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 secvenţiale cât şi a celor combinaţionale. 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 înţelegerea posibilităţilor şi limitărilor specifice *slice*-urilor şi CLB-urilor. Fiecare *slice* are o capacitate funcţională şi de stocare a datelor limitată, iar noi trebuie să determinăm *cum* pot fi folosite aceste capacităţi. Vom studia posibilităţile de construire a decodificatoarelor, multiplexoarelor, registrelor de deplasare şi a numărătoarelor (se subînţelege că implementarea codificatoarelor şi demultiplexoarelor se realizează analog cu cea a decodificatoarelor şi respectiv a multiplexoarelor).

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

*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 funcţii de mai multe intrări. Problemele care apar aici se referă la faptul că această abordare consumă rapid *slice*-urile şi scade viteza funcţiilor. În plus, bistabilele din *slice*-uri rămân de multe ori neutilizate.

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

Implementarea *numărătoarelor* reprezintă o problemă interesantă în arhitecturile LCA. Metodele clasice sunt aplicabile numai pentru numărătoare mici. Partea combinaţională din alcătuirea unui numărător (cea care determină starea următoare, deci funcţiile de tranziţie) creşte foarte mult, în termeni de intrări necesare pentru porţile logice, pe măsură ce creşte numărul de biţi (ordinul) numărătorului. Acest tip de abordare limitează prea mult posibilităţile LCA-ului, aşa că XILINX recomandă pentru numărătoare o abordare modulară şi cascadabilă. În această abordare, fiecare bistabil din numărător are în faţă un acelaşi tip de circuit logic combinaţional. Vom numi un bistabil, împreună cu funcţiile de tranziţie 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 biţi ai numărătorului este egal cu numărul de celule de numărare. Începând cu seria 4000, funcţiile de tranziţie 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 biţi:

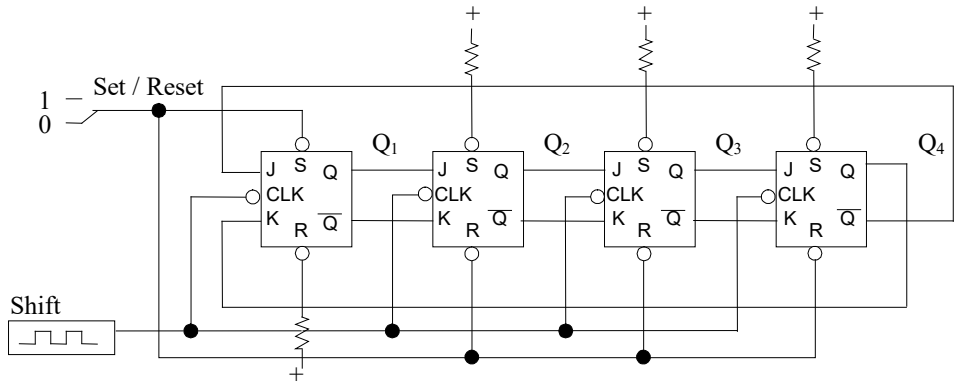


**Figura 13.1** *Numărător Moebius pe 4 biţi*

Acest numărător va avea formele de undă conform figurii 13.2:

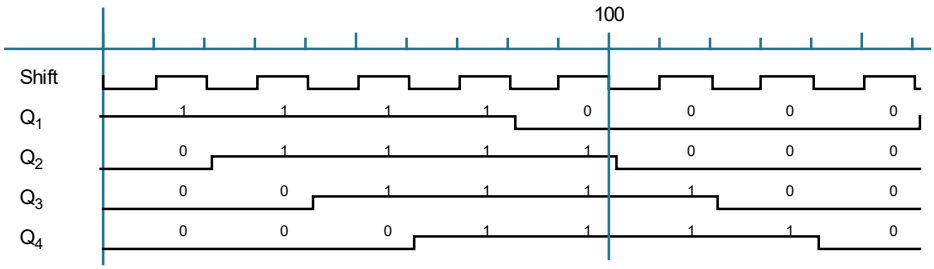


**Figura 13.2** *Formele de undă ale numărătorului Moebius*

După editarea proiectului, acesta va fi simulat cu ajutorul simulatorului din ISE Foundation şi se vor obţine 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 performanţei proiectului şi chiar a sistemului în care este integrat cipul FPGA gazdă.

***a) Duplicarea bistabilelor***

Există situaţii 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.

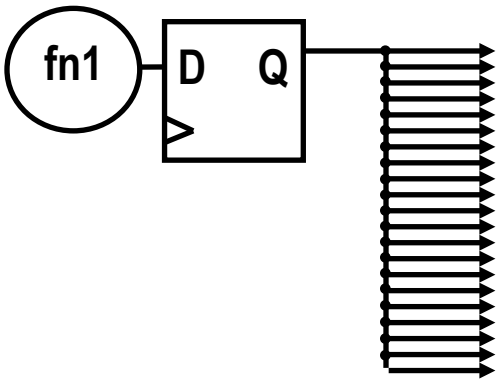


**Figura 13.3** *Fir cu fanout mare: un singur bistabil comandă un număr foarte mare de alte resurse*

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 direcţiona semnalul produs de el către regiuni fizice diferite ale cipului, reducându-se astfel congestia căilor de rutare.

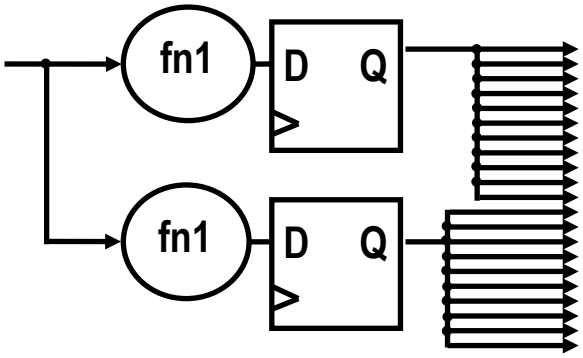


**Figura 13.3** *Duplicarea bistabilului reduce* fanout-*ul şi congestia căilor de rutare*

Principalele rezultate ale aplicării acestei tehnici:

- Creşte rutabilitatea şi performanţa;
- Creşte suprafaţa ocupată de către proiect în cip (logica activă + reţeaua de interconectare).

Se recomandă ca bistabilele duplicate să fie notate cu „_a", „_b" etc., nu „_1", „_2", deoarece bistabilele numerotate sunt amplasate în același *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 combinaţională între registrele tampon din interiorul sistemului – situaţia este ilustrată în figura 13.4.

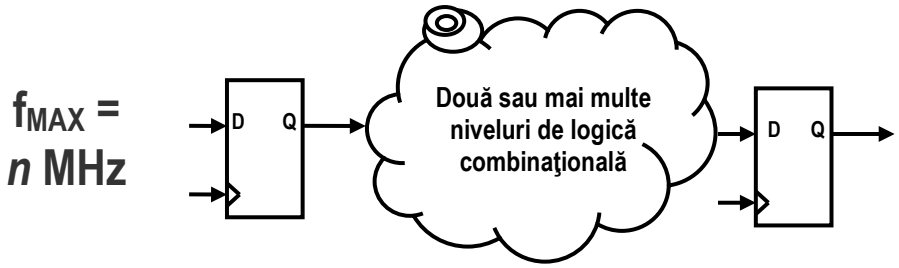


**Figura 13.4** *Structura generală a unui sistem numeric*

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

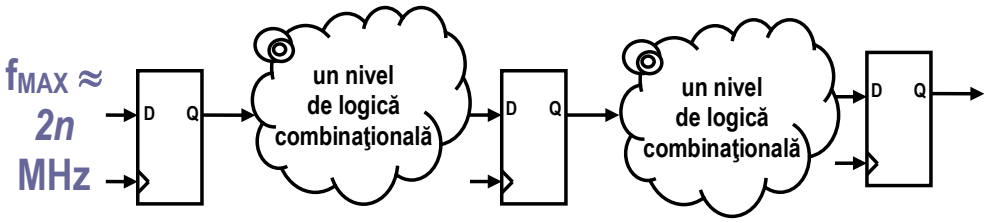


**Figura 13.5** *Crearea unui* pipeline *cu două etaje pentru creşterea vitezei*

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

În primul rând, trebuie să determinăm câte niveluri de logică combinaţională există între bistabile în proiectul nostru. Dacă există un singur nivel logic între bistabile, tehnica de *pipelining* nu va îmbunătăţi performanţa. 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 dispoziţie 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 *latenţa*. 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.

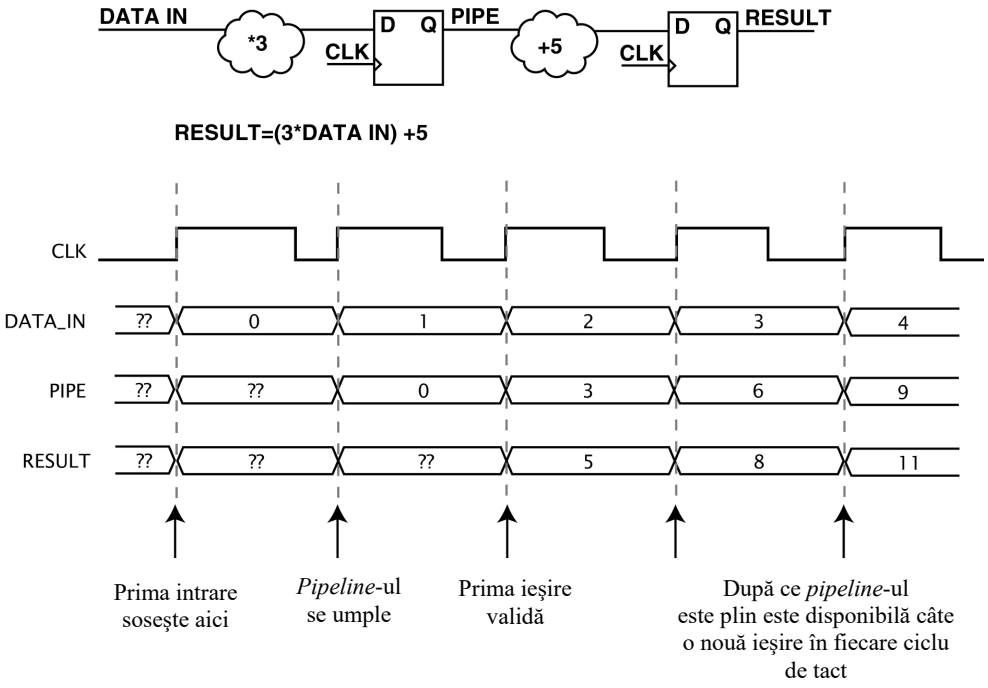


**Figura 13.6** *Problema latenţei în* pipeline

Un exemplu de *pipeline* este cel din figura 13.7 a). Se observă că există două niveluri de logică combinaţională între bistabilul sursă şi cel destinaţie. Frecvenţa operaţională este de aproximativ 233 MHz.

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

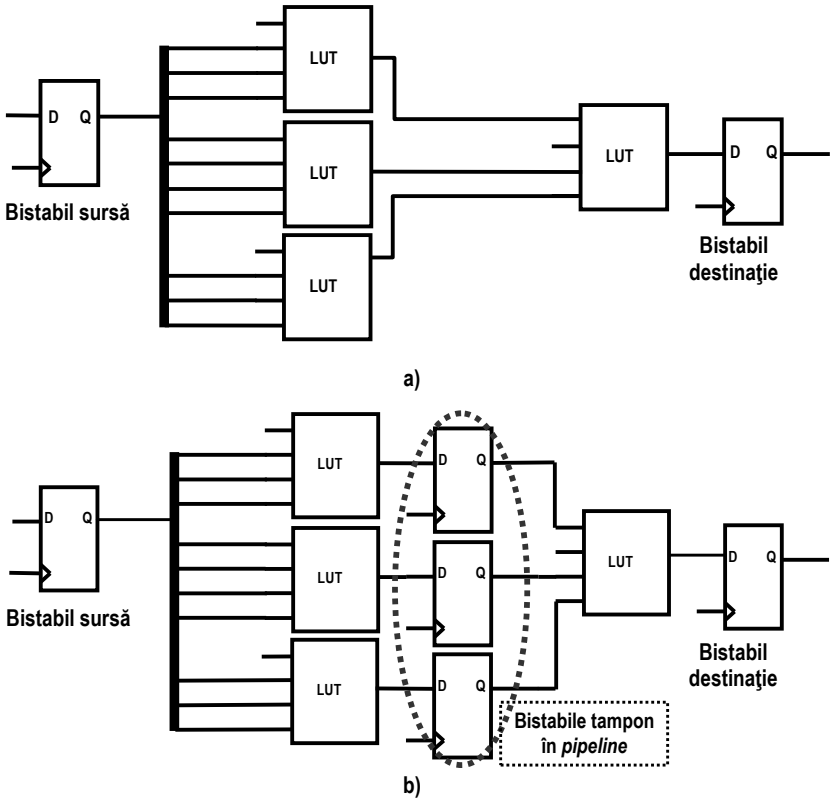


**Figura 13.7** *a) Circuitul iniţial cu două niveluri de logică combinaţională între bistabilul sursă şi cel destinaţie; b) Circuitul* pipeline*-izat*

### *c) Folosirea bistabilelor din blocurile de intrare / ieşire*

Fiecare bloc de intrare/ieşire (IOB) din dispozitivele FPGA XILINX conţine 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 interfaţarea 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 obţine utilizarea bistabilelor din IOB-uri în decursul procesului de sinteză, putem recurge la specificarea unor constrângeri temporale care să forţeze 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 instanţe 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", opţiunea „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 secţiunea „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 siguranţa şi fiabilitatea proiectului.

Circuitele de sincronizare sunt necesare atunci când cipul are intrări asincrone sau atunci când în cadrul aceluiaşi proiect există zone care funcţionează sincron cu semnale de tact diferite şi total neînrudite (între care nu există nici o corelaţie), dar care schimbă informaţie prin intermediul unor semnale. Aceste semnale interzonale trebuie sincronizate cu ajutorul unor circuite de sincronizare. Dacă între semnalele de tact există o corelaţie, 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 potenţiale).

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 situaţii (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*.

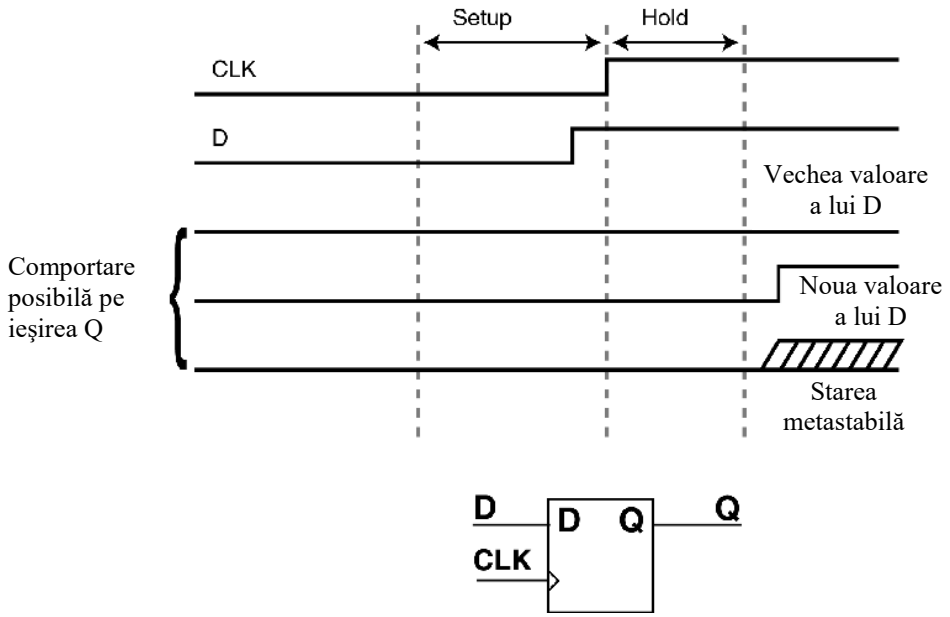


**Figura 13.8** *Rezultatele posibile ale violării timpilor de setup şi de hold*

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, apariţia evenimentelor metastabile poate fi doar redusă, nu eliminată complet. Există un parametru funcţional care se numeşte Timpul Mediu Între Căderi (*Mean Time Between Failure* – MTBF) şi care depinde în mod exponenţial 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 apariţiei 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.

Circuitul de sincronizare 1

Acesta poate fi folosit atunci când impulsurile de intrare au întotdeauna o durată cel puţin egală cu cea a perioadei semnalului de tact. Bistabilul „suplimentar" previne apariţia fenomenului de metastabilitate datorită faptului că îi oferă primului bistabil timpul necesar pentru a-şi „reveni" după apariţia unui eveniment de tip metastabilitate.

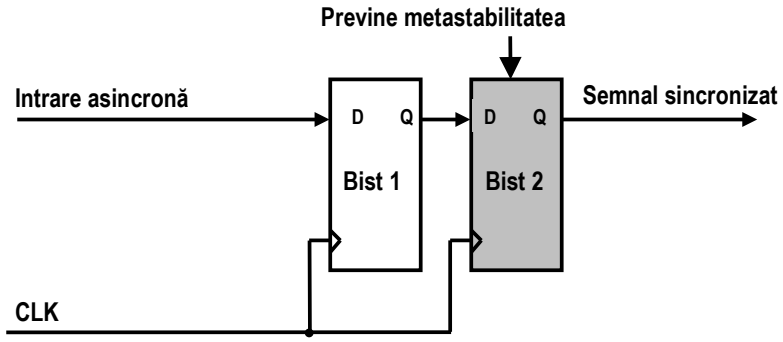


**Figura 13.9** *Circuitul de sincronizare 1*

Circuitul de sincronizare 2

Acesta poate fi folosit atunci când impulsurile de intrare pot avea o durată mai mică decât cea a perioadei semnalului de tact. Bistabilul 1 captează impulsurile scurte, care sunt apoi preluate în Bistabilul 2. Bistabilul 3 previne apariţia stării metastabile; dacă semnalul sincronizat a fost preluat corect, Bistabilul 1 este resetat pentru a fi pregătit să capteze următorul impuls.

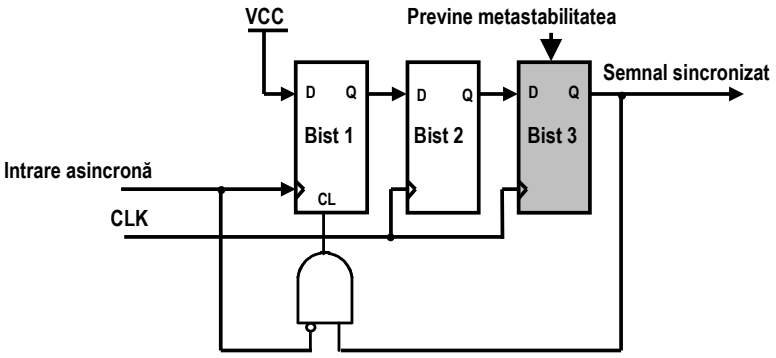


**Figura 13.10** *Circuitul de sincronizare 2*

## 2.4 Reguli de sinteză în cazul specificării proiectului prin limbaj de descriere hardware

### *a) Reguli generale de sinteză*

Prezentăm succint în continuare câteva reguli simple şi eficiente de scriere a codului şi de structurare a proiectului în cazul în care dorim implementarea acestuia într-un dispozitiv FPGA XILINX:

- Folosiţi tehnica de pipelining – cu ajutorul ei se va spori frecvenţa operaţională a sistemului proiectat şi implicit viteza;
- Folosiţi reset-ul sincron – astfel controlul sistemului de calcul proiectat va fi mai bun;
- Folosiţi optimizările automatelor finite – există astfel de setări care se pot face în diferitele meniuri ale utilitarelor din pachetul software de susţinere a proiectării;
- Folosiţi resurse care pot fi inferate de către instrumentul de sinteză, cum ar fi:
    - Multiplexor;
    - Shift Register LUT (SRL);
    - BlockRAM, LUT RAM;
    - Blocuri DSP cascadate.
- Evitaţi construcţiile de limbaj de nivel înalt (de exemplu, buclele) în cod – multe instrumente de sinteză produc implementări lente pe baza lor;
- Folosiţi constrângerile temporale:
    - Specificaţi constrângeri stricte, dar realiste asupra semnalelor de tact;
    - Plasaţi semnale de tact neînrudite în grupuri (*clock groups*) diferite.
- Folosiţi opţiunile şi atributele de sinteză cele mai adecvate:
    - Dacă doriţi să obţineţi o viteză cât mai mare, dezactivaţi opţiunea de partajare a resurselor (*resource sharing*). Când această opţiune este selectată, instrumentul de sinteză va permite ca funcţiile logice să aibă în comun anumite căi de semnal (de exemplu, două sumatoare separate pot fi autorizate să partajeze anumite blocuri logice). Folosirea acestei opţiuni generează de obicei module cu viteză mai mică, dar care ocupă mai puţin spaţiu (logică activă) în cip;

  - Mutaţi bistabilele din IOB-uri mai aproape de blocurile logice combinaţionale;
  - Activaţi opţiunea de optimizare a automatelor finite (*FSM optimization*) – implementarea automatelor finite este detaliată mai jos în cadrul acestei lucrări;
  - Folosiţi opţiunea de *retiming* – semnificaţia ei este detaliată mai jos în cadrul acestei lucrări.
- Evitaţi instrucţiunile *if-then-else* încuibările – majoritatea instrumentelor de sinteză le implementează în paralel, însă este posibil să se genereze un bloc logic pe bază de priorităţi (cum este de pildă codificatorul prioritar), deşi nu acesta era efectul dorit;
- Folosiţi instrucţiuni de tip *case* pentru decodificatoare de mari dimensiuni, şi nu instrucţiuni de tipul *if-then-else*;
- Ordonaţi şi grupaţi funcţiile şi operatorii aritmetici şi logici – de exemplu, scrieţi „A <= (B + C) + (D + E)”, şi nu „A <= B + C + D + E”;
- Evitaţi orice inferenţă nedorită a unui bistabil – trebuie acoperite toate ieşirile posibile pe fiecare ramură de cod. Acest lucru este uşor de realizat specificând instrucţiuni de atribuire de valori implicite înaintea instrucţiunilor *if-then-else* şi *case*;
- Unele resurse trebuie să fie instanţiate sau create / generate pe baza unui *IP core* (cu ajutorul utilitarelor *Architecture Wizard* şi *CORE Generator*). Exemple în acest sens: modulele FIFO16, ISERDES şi OSERDES (primitive care fac ca pinii de intrare / ieşire să comunice cu logica din exteriorul cipului FPGA la viteze mai mari), diverse resurse de gestionare a semnalelor de tact;
- Anumite resurse necesită o codificare specifică:
  - Registrele din primitiva DSP48 au numai *set* / *reset sincron*;
  - Memoriile RAM / ROM distribuite şi primitivele SRL nu sunt prevăzute cu funcţionalitatea de *set* sau *reset* după configurare.
- Este preferabilă folosirea *reset*-ului sincron în locul celui asincron – dispozitivele FPGA XILINX au un *reset* al configuraţiei (GSR – *Global Set-Reset*) care este utilizat pe durata etapei de configurare a cipului, pentru a aduce FPGA-ul într-o stare cunoscută. Semnalul GSR este de asemenea accesibil proiectantului după configurare, prin intermediul blocului

STARTUP. Nu este necesară iniţializarea asincronă la punerea sub tensiune a cipului.

### *b) Sinteza automatelor finite*

Sinteza eficientă a automatelor finite este un aspect deosebit de important, deoarece în multe aplicaţii proiectantul trebuie să-şi definească propriile automate finite, cu totul unice (în funcţie de cerinţele aplicaţiei). Întrucât de regulă automatele finite reprezintă „creierul" sistemului, este imperios necesar ca instrumentele de sinteză să genereze o structură optimă a lor.

Un automat finit trebuie să posede următoarele elemente:

- La intrări: semnale de intrare şi tranziţii de la o stare la alta;
- La ieşiri: semnale de ieşire, de control şi de validare pentru restul sistemului;
- Automatul **NU** trebuie să conţină logică aritmetică, unităţi de execuţie (căi de date) sau alte funcţii combinaţionale.

Cu alte cuvinte, automatul finit trebuie să conţină strict logica de trecere dintr-o stare în alta şi de generare a ieşirilor specifice, fără nici o interferenţă cu alte module ale sistemului (trebuie să fie „pur", complet separat de partea de execuţie).

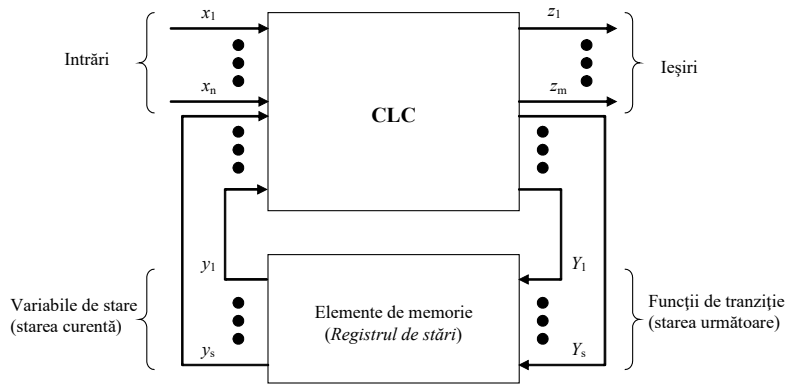


**Figura 13.11** *Schema generică a unui automat finit*

Când specificăm un automat finit într-un limbaj de descriere hardware, vom plasa logica responsabilă de generarea noii stări într-o instrucţiune *case*. Tot aici putem include registrul de stări; el poate fi specificat şi într-un proces separat. Această abordare previne partajarea resurselor (*resource sharing*), care poate afecta viteza sistemului.

În cazul în care folosim limbajul VHDL, vom folosi tipuri enumerate pentru definirea stărilor automatului, deoarece majoritatea instrumentelor de sinteză au comenzi pentru extragerea şi recodificarea stărilor automatelor descrise astfel. Acesta este un mare avantaj pentru proiectant, care nu trebuie să se mai preocupe de acest aspect (unde de altfel are destule şanse să greşească)!

În cazul în care urmărim să obţinem viteze foarte mari se recomandă folosirea metodei de sinteză cu un bistabil pe stare (one-hot encoding). Astfel vom folosi mai multe bistabile, dar se va simplifica logica de calcul al noii stări.

O ultimă sugestie este aceea de a trece ieşirile automatului finit prin registre, pentru a-i creşte performanţa.

### *c) Instanţierea şi inferenţa resurselor*

Se recomandă instanţierea unei componente atunci când suntem nevoiţi să impunem cu precizie ce resursă ne este necesară, iar instrumentul de sinteză fie nu poate infera resursa respectivă, fie o inferează greşit. Inferenţa este preferabilă ori de câte ori este posibil, deoarece face codul mult mai portabil.

Pe de altă parte, instanţierea este recomandată pentru a crea blocuri cu funcţionalitate mai complexă, cum ar fi Unităţi Aritmetico-Logice, multiplicatoare rapide, filtre cu răspuns finit (*Finite Impulse Response* – FIR). Pentru acestea se recomandă utilitarul CORE Generator. Se va folosi instanţierea numai atunci când este necesară accesarea anumitor caracteristici speciale ale dispozitivului FPGA, când se doreşte creşterea performanţei sistemului sau diminuarea suprafeţei de logică activă ocupată în cip.

Este indicat să limităm localizarea componentelor instanţiate la doar câteva fişiere sursă, pentru a facilita localizarea acestor componente atunci când vom dori să portăm codul pe un alt dispozitiv FPGA gazdă.

XILINX recomandă instanţierea următoarelor elemente:

- Resurse de memorie – mai ales BlockRAM-uri (se poate folosi utilitarul CORE Generator pentru a construi memorii de mari dimensiuni);
- Resursele standard SelectIO™ – cele cu ajutorul cărora se selectează standardul electric;
- Resursele de gestionare a semnalelor de tact – DCM, IBUFG, BUFG, BUFGMUX şi BUFGCE.

Raţiunea acestor sugestii o găsim în faptul că astfel este mai uşor să schimbăm sau să portăm proiectul nostru pe alte suporturi fizice, realizate în tehnologii noi (mai avansate). În felul acesta, sunt mai puţine constrângeri de sinteză şi atribute care trebuie transmise mai departe (cel mai simplu este să păstrăm majoritatea atributelor şi constrângerilor în fişierul *User Constraints File* (UCF), căci astfel informaţiile critice sunt grupate într-un singur fişier).

Se recomandă de asemenea să creăm un bloc ierarhic separat pentru instanţierea acestor resurse. Deasupra blocului de la nivelul ierarhic cel mai înalt din proiectul nostru (*top-level*), vom crea un „*wrapper*" care va conţine instanţierile componentelor specifice dispozitivelor FPGA XILINX. În cazul în care vom dori la un moment dat să portăm proiectul nostru pe un dispozitiv FPGA al altui producător, va fi suficient să efectuăm modificările în acest *wrapper*.

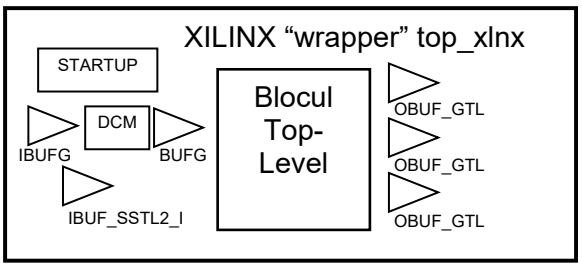


**Figura 13.12** *Schema de principiu a unui wrapper*

### *d) Operaţia de* Retiming

Această operaţie este suportată de principalii producători de software de susţinere a proiectării. Instrumentul de sinteză urmăreşte să distribuie automat blocurile de logică combinaţională dintre registre pentru a echilibra întârzierile pe căile de date combinaţionale. Figura 13.13 ilustrează acest concept.

Înainte de Retiming


După Retiming

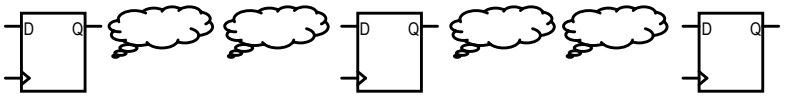


**Figura 13.13** *Operaţia de retiming*

### 3. Desfăşurarea lucrării

1. Se editează şi se simulează numărătorul Moebius din figura 13.1 şi se verifică obţinerea corectă a formelor de undă din figura 13.2.
2. Se parcurg toţi paşii necesari în vederea implementării numărătorului Moebius în cipul FPGA de pe placa NEXYS-2, folosindu-se şi utilitarele din pachetul Foundation Series menţionate în secţiunea 2.2.
3. Se editează, se simulează şi se parcurgând toţi paşii necesari în vederea implementării unui numărător zecimal şi a unei unităţi aritmetico-logice în cipul FPGA de pe placa NEXYS-2.
4. Se editează şi se simulează un dispozitiv universal numărător / registru de deplasare, parcurgându-se apoi toţi paşii necesari în vederea implementării în cipul FPGA de pe placa NEXYS-2.
5. Se va specifica numărătorul Moebius din figura 13.1 prin limbaj de descriere hardware, iar numărătorul zecimal şi unitatea aritmetico-logică vor fi specificate prin editorul schematic. Cum este mai simplu? Care variantă de implementare (cu editor schematic sau cu limbaj de descriere hardware) este mai avantajoasă şi în ce condiţii?
6. Fiind dat următorul circuit (figura 13.14):

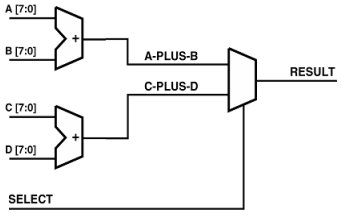


**Figura 13.14** *Circuitul iniţial*

s-a dorit *pipeline*-izarea lui, aşa că s-a creat noua schemă din figura 13.15

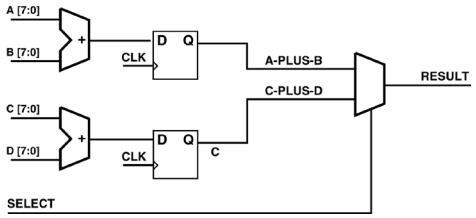


**Figura 13.15** *Circuitul* pipeline-*izat*

Care este problema cu noul circuit? Cum poate fi rezolvată această problemă?

Anexă

# Ghidul utilizatorului plăcii NEXYS-2 – elemente de bază –

Fiind produsă de aceeaşi companie (Digilent Inc.), placa NEXYS-2 are în principiu componentele principale similare cu cele ale plăcii NEXYS-3 care a fost prezentată în Anexa lucrării 12.

Principala deosebire (din punct de vedere al desfăşurării lucrărilor practice) constau în dispozitivul FPGA existent pe placă (Spartan3E-1200 FG320). Modul său de programare se realizează tot pe portul USB. Pentru programarea plăcii este necesară folosirea unui program suplimentar al firmei Digilent Inc., numit ADEPT.

## A.1 Cele patru afişaje cu LED-uri cu 7 segmente

Numărul şi principiul de funcţionare al acestor afişaje este similar celui al afişajelor de pe placa Nexys3. Pinii dispozitivului FPGA la care sunt legaţi pinii afişajelor sunt redaţi în tabelele 13.1 şi 13.2.

**Tabelul 13.1** *Conexiunile dintre dispozitivul FPGA şi afişajul cu 7 segmente (active pe 0)*

| Segmentul | Pinul dispozitivului FPGA |
|---|---|
| CA | L18 |
| CB | F18 |
| CC | D17 |
| CD | D16 |
| CE | G14 |
| CF | J17 |
| CG | H14 |
| DP | C17 |

**Tabelul 13.2** *Semnalele de control al anodului (active pe 0)*

| **Anode Control** | AN3 | AN2 | AN1 | AN0 |
|---|---|---|---|---|
| **FPGA Pin** | F15 | C18 | H17 | F17 |

### A.2 Cele opt comutatoare cu două poziţii

Numărul şi principiul de funcţionare al acestor comutatoare este similar celui al comutatoarelor de pe placa Nexys3. Pinii dispozitivului FPGA la care sunt legaţi pinii comutatoarelor sunt redaţi în tabelul 13.3.

**Tabelul 13.3** *Conexiunile comutatoarelor la pinii dispozitivului FPGA*

| **Comutator** | SW7 | SW6 | SW5 | SW4 | SW3 | SW2 | SW1 | SW0 |
|---|---|---|---|---|---|---|---|---|
| **Pin FPGA** | R17 | N17 | L13 | L14 | K17 | K18 | H18 | G18 |

### A.3 Cele patru comutatoare de tip „push button”

Numărul și principiul de funcționare al acestor comutatoare de tip „push button” este similar celui al comutatoarelor de tip „push button” de pe placa Nexys3. Pinii dispozitivului FPGA la care sunt legaţi pinii comutatoarelor de tip „push button” sunt redaţi în tabelul 13.4.

**Tabelul 13.4** *Conexiunile comutatoarelor de tip „push button” la pinii dispozitivului FPGA*

| ***Push Button*** | BTN3 | BTN2 | BTN1 | BTN0 |
|---|---|---|---|---|
| **Pin FPGA** | H13 | E18 | D18 | B18 |

### A.4 LED-urile

Numărul şi principiul de funcţionare al acestor LED-uri este similar celui al LED-urilor de pe placa Nexys3. Pinii dispozitivului FPGA la care sunt legaţi pinii LED-urilor sunt redaţi în tabelul 13.5.

**Tabelul 13.5** *Conexiunile LED-urilor la pinii dispozitivului FPGA Spartan3E – varianta 1200*

| **LED** | LD7 | LD6 | LD5 | LD4 | LD3 | LD2 | LD1 | LD0 |
|---|---|---|---|---|---|---|---|---|
| **Pin FPGA** | P4 | E4 | P16 | E16 | K14 | K15 | J15 | J14 |

### A.5 Programarea plăcii

Pentru programarea plăcii nu se poate folosi utilitarul IMPACT din mediul ISE Foundation, ci se va folosi programul ADEPT (mai precis modulul ExPort al acestuia). Interfaţa fiind foarte simplă şi intuitivă, nu va fi prezentată aici.