

# **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 numărator 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.

Totuși, 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 și 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 și limitărilor specifice *slice*-urilor și CLB-urilor. Fiecare *slice* are o capacitate funcțională și 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 și a numărătoarelor (se subînțelege că implementarea codificatoarelor și demultiplexoarelor se realizează analog cu cea a decodificatoarelor și respectiv a multiplexoarelor).

*Decodificatoarele* se implementează prin cascada RAM-urilor funcționale ale *slice*-urilor, care sunt configurate fie ca funcții ȘI fie ca funcții ȘI-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 și o ieșire) numai pentru a genera ieșirile distincte pentru toate cele 8 combinații.

*Multiplexoarele*, ca și 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 și scade viteza funcțiilor. În plus, bistabilele din *slice*-uri rămân de multe ori neutilizate.

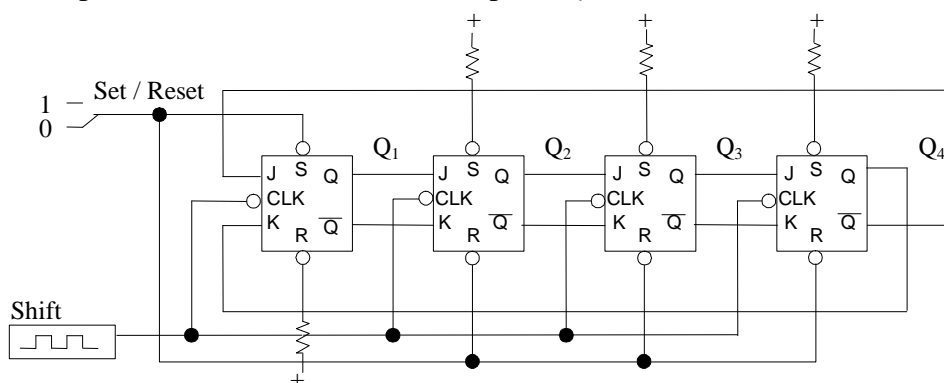
*Registrele de deplasare* sunt eficient de implementat în *slice*-uri. Atribuind biții unor *slice*-uri sau CLB-uri adiacente, registrele de deplasare nu necesită interconectare globală și nu blochează canalele de rutare. Clasa de aplicații care utilizează din plin registre de deplasare cuprinde echipamente de comunicații, dispozitive de recunoaștere a formelor, generatoare polinomiale etc. La fel cum decodificatoarele și 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) crește foarte mult, în termeni de intrări necesare pentru porțile logice, pe măsură ce crește numărul de biți (ordinul) numărătorului. Acest tip de abordare limitează prea mult posibilitățile LCA-ului, așa că XILINX recomandă pentru numărătoare o abordare modulară și cascabilă. În această abordare, fiecare bistabil din numărător are în față un același 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 și-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 totuși timpul de propagare al semnalelor de la o celulă de numărare la următoarea.

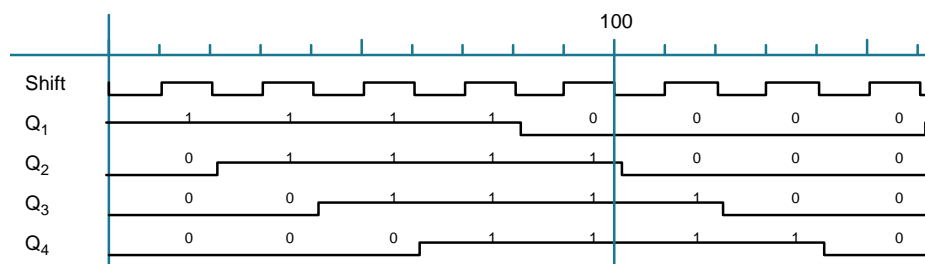
## 2.2 Introducerea proiectului prin editor schematic

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

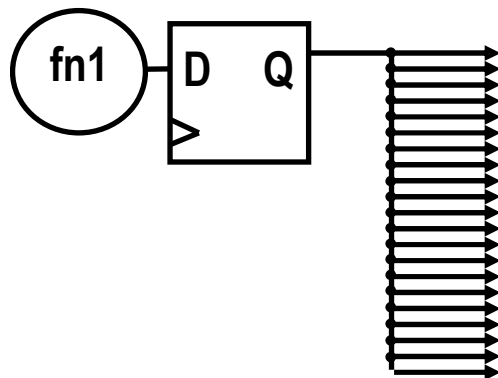
După editarea proiectului, acesta va fi simulat cu ajutorul simulatorului din ISE Foundation și se vor obține formele de undă prezentate în figura 13.2. În continuare, putem parcurge celelalte etape din fluxul de proiectare, conform celor prezentate în Lucrarea 12.

### 2.3 Reguli și metode generale pentru sinteza circuitelor numerice în FPGA

În cele ce urmează vom prezenta câteva reguli și metode cu caracter general, dar extrem de utile pentru sinteza proiectelor digitale pentru dispozitive FPGA (și nu numai). Aceste reguli și metode permit creșterea performanței proiectului și 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.

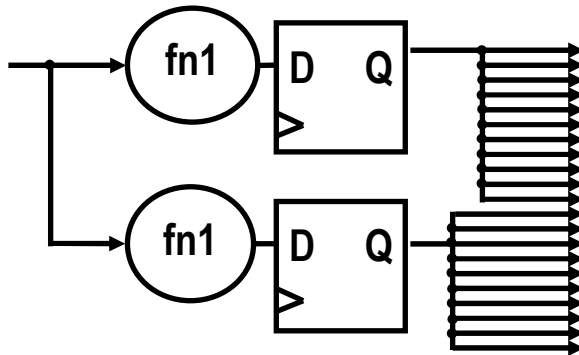


**Figura 13.3** *Fir cu fanout mare: un singur bistabil comandă un număr foarte mare de alte resurse*

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

- Reducerea fanout-ului micșorează 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 și congestia căilor de rutare

Principalele rezultate ale aplicării acestei tehnici:

- Crește rutabilitatea și performanța;
- Crește 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 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 deși 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.

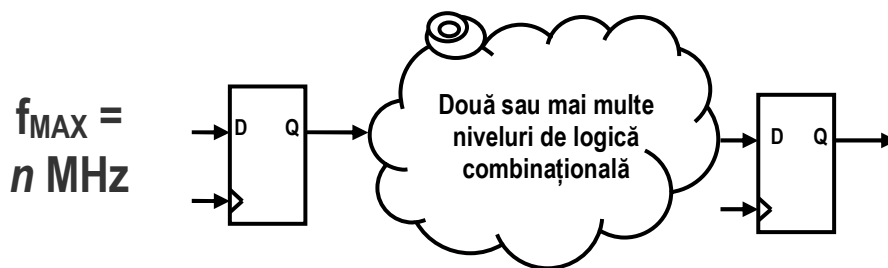
Trebuie de asemenea să setăm instrumentele de sinteză astfel ca să păstreze logica redundantă (de exemplu, se folosește atributul KEEP).

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

### **b) Pipelining**

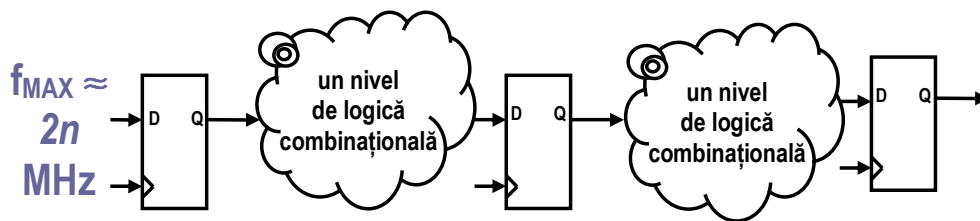
Tehnica de *pipelining* are ca scop creșterea vitezei sistemelor de calcul. Regula generală este de a se introduce cel mult două niveluri de

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 creșterea vitezei

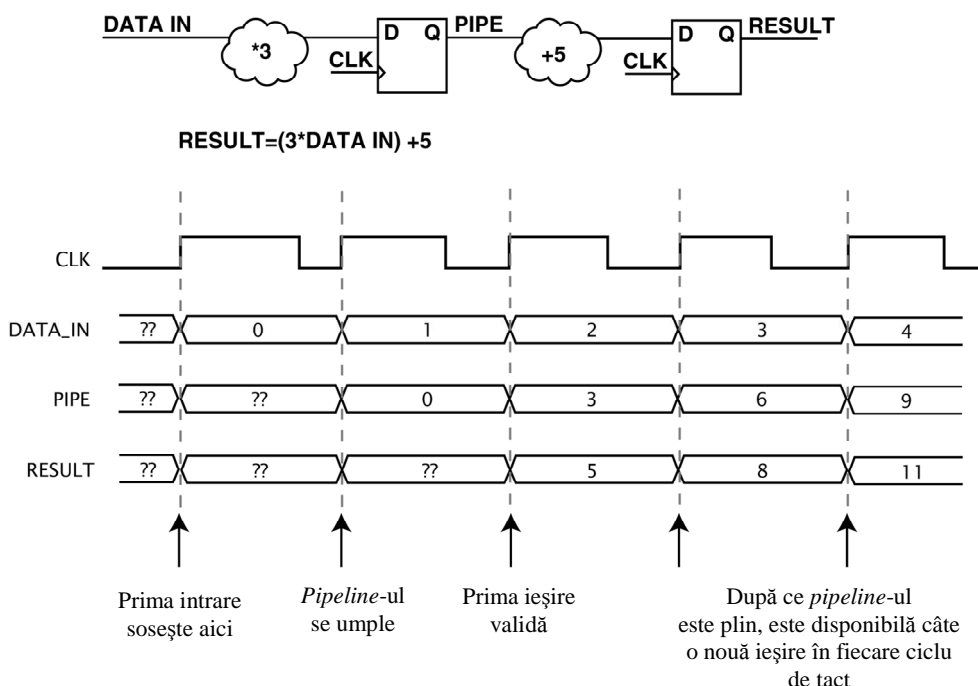
Atunci când recurgem la *pipelining*, trebuie să verificăm anumite aspecte.

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.

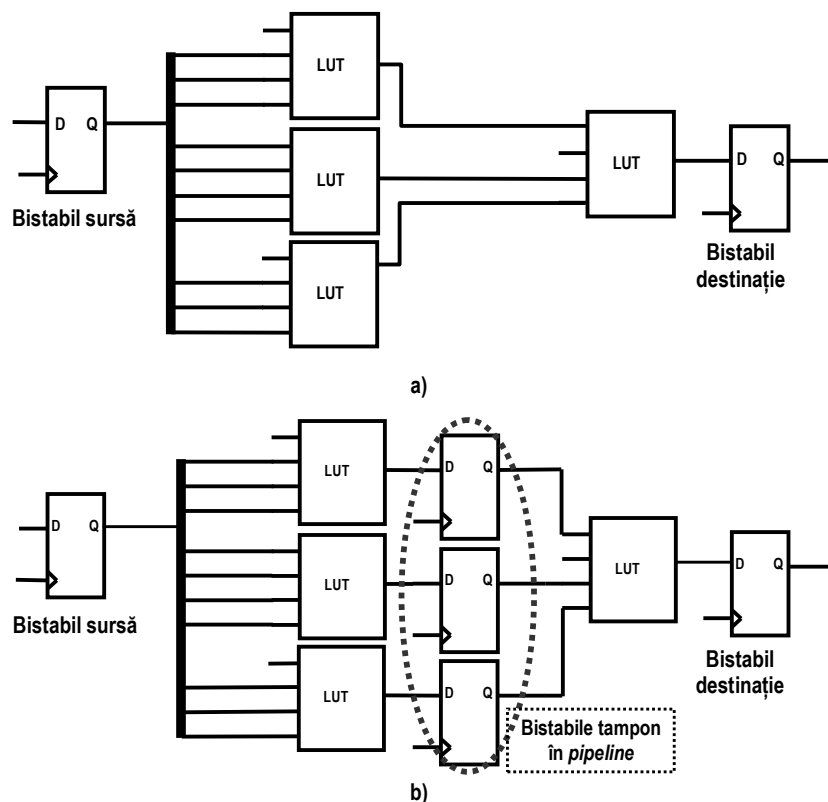
Următoarea problemă pe care trebuie să ne-o punem este aceea dacă sistemul poate tolera *latența*. Fiecare etaj al *pipeline*-ului introduce o întârziere egală cu o perioadă a impulsului de tact, înainte ca prima ieșire corectă să fie disponibilă. Acest fenomen se numește „umplerea *pipeline*-ului”. După ce *pipeline*-ul este „plin”, din acel moment încolo este disponibilă câte o nouă ieșire 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 și 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 combinatională între bistabilul sursă și cel destinație; b) Circuitul pipeline-izat

### c) Folosirea bistabilelor din blocurile de intrare / ieșire

Fiecare bloc de intrare / ieșire (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* și *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 și 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 și 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** și apoi se vor specifica registrele care ar trebui să fie plasate în IOB-uri. Este necesar să cunoaștem 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 și metastabilitatea***

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

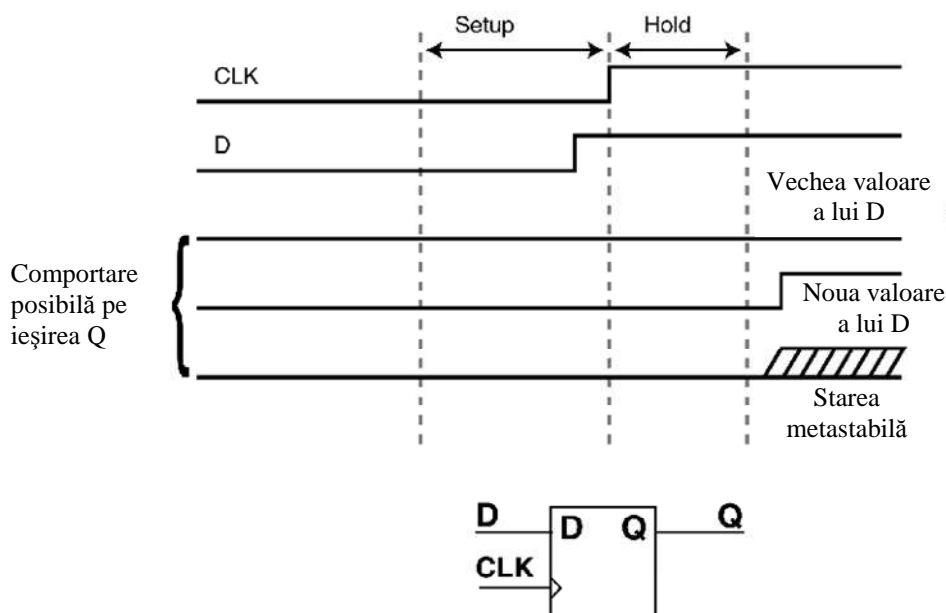
Aceste circuite sunt necesare pentru a preveni violarea timpilor de *setup* și de *hold* ale bistabilelor și pentru a asigura astfel siguranța și fiabilitatea proiectului.

Circuitele de sincronizare sunt necesare atunci când cipul are intrări asincrone sau atunci când în cadrul aceluiași proiect există zone care funcționează sincron cu semnale de tact diferite și 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 și de hold constituie o problemă serioasă și 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;
- ieșirea bistabilului devine *metastabilă*.



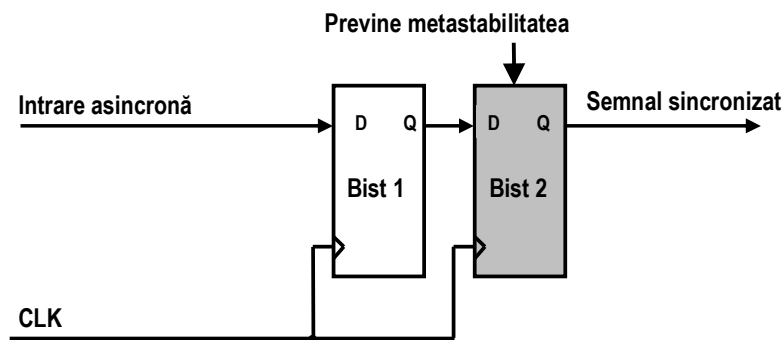
**Figura 13.8** Rezultatele posibile ale violării timpilor de setup și de hold

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*) și care depinde în mod exponențial de mărimea intervalului de timp acordat bistabilului pentru a-și „reveni” din starea metastabilă. Dacă îi acordăm bistabilului câteva nanosecunde suplimentare pentru a-și „reveni”, aceasta va reduce extrem de mult șansele apariției unui eveniment de acest tip. Circuitele din figurile 13.9 și 13.10 permit acordarea unui „timp de recuperare” egal cu durata unui ciclu de tact complet.

### Circuitul de sincronizare 1

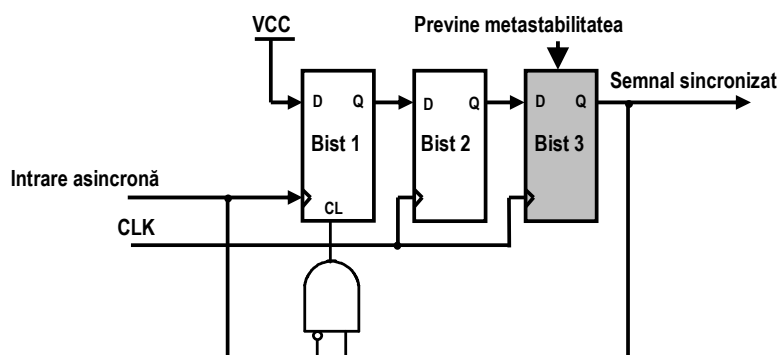
Acesta poate fi folosit atunci când impulsurile de intrare au întotdeauna o durată cel puțin egală cu cea a perioadei semnalului de tact. Bistabilul „suplimentar” previne apariția fenomenului de metastabilitate datorită faptului că îi oferă primului bistabil timpul necesar pentru a-și „reveni” după apariția 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 apariția 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 și eficiente de scriere a codului și de structurare a proiectului în cazul în care dorim implementarea acestuia într-un dispozitiv FPGA XILINX:

- Folosiți tehnica de pipelining – cu ajutorul ei se va spori frecvența operațională a sistemului proiectat și implicit viteza;
- Folosiți reset-ul sincron – astfel controlul sistemului de calcul proiectat va fi mai bun;
- Folosiți optimizările automatelor finite – există astfel de setări care se pot face în diferitele meniuri ale utilităților din pachetul software de susținere a proiectării;
- Folosiți resurse care pot fi inferate de către instrumentul de sinteză, cum ar fi:
  - Multiplexor;
  - Shift Register LUT (SRL);
  - BlockRAM, LUT RAM;
  - Blocuri DSP cascade.
- Evitați construcțiile de limbaj de nivel înalt (de exemplu, buclele) în cod – multe instrumente de sinteză produc implementări lente pe baza lor;
- Folosiți constrângerile temporale:
  - Specificați constrângeri stricte, dar realiste asupra semnalelor de tact;
  - Plasați semnale de tact neînrudite în grupuri (*clock groups*) diferite.
- Folosiți opțiunile și atributele de sinteză cele mai adecvate:
  - Dacă doriți să obțineți o viteză cât mai mare, dezactivați opțiunea de partajare a resurselor (*resource sharing*). Când această opțiune este selectată, instrumentul de sinteză va permite ca funcțiile logice să aibă în comun anumite căi de semnal (de exemplu, două sumatoare separate pot fi autorizate să partajeze anumite blocuri logice). Folosirea acestei opțiuni generează de obicei module cu viteză mai mică, dar care ocupă mai puțin spațiu (logică activă) în cip;

- Mutați bistabilele din IOB-uri mai aproape de blocurile logice combinaționale;
  - Activați opțiunea de optimizare a automatelor finite (*FSM optimization*) – implementarea automatelor finite este detaliată mai jos în cadrul acestei lucrări;
  - Folosiți opțiunea de *retiming* – semnificația ei este detaliată mai jos în cadrul acestei lucrări.
- Evitați instrucțiunile *if-then-else* încuibărite – majoritatea instrumentelor de sinteză le implementează în paralel, însă este posibil să se genereze un bloc logic pe bază de priorități (cum este de pildă codificatorul prioritar), deși nu acesta era efectul dorit;
- Folosiți instrucțiuni de tip *case* pentru decodificatoare de mari dimensiuni, și nu instrucțiuni de tipul *if-then-else*;
- Ordonăți și grupați funcțiile și operatorii aritmetici și logici – de exemplu, scrieți „ $A \leq (B + C) + (D + E)$ ”, și nu „ $A \leq B + C + D + E$ ”;
- Evitați orice inferență nedorită a unui bistabil – trebuie acoperite toate ieșirile posibile pe fiecare ramură de cod. Acest lucru este ușor de realizat specificând instrucțiuni de atribuire de valori implicite înaintea instrucțiunilor *if-then-else* și *case*;
- Unele resurse trebuie să fie instanțiate sau create / generate pe baza unui *IP core* (cu ajutorul utilităților *Architecture Wizard* și *CORE Generator*). Exemple în acest sens: modulele FIFO16, ISERDES și OSERDES (primitive care fac ca pinii de intrare / ieșire 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 și primitivele SRL nu sunt prevăzute cu funcționalitatea de *set* sau *reset* după configurare.
- Este preferabilă folosirea *reset*-ului sincron în locul celui asincron – dispozitivele FPGA XILINX au un *reset* al configurației (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 inițializarea asincronă la punerea sub tensiune a cipului.

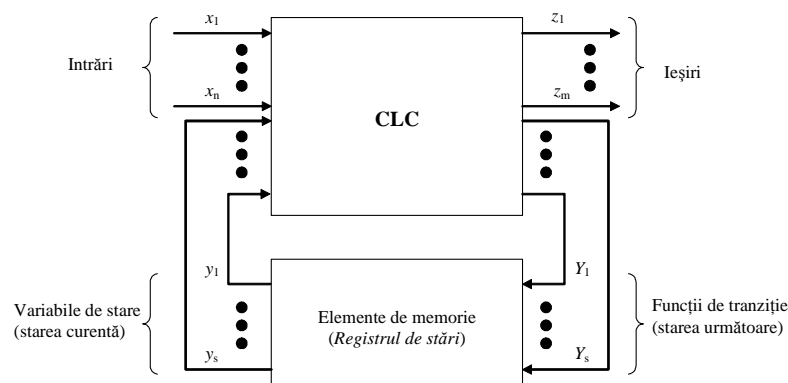
### b) Sinteza automatelor finite

Sinteza eficientă a automatelor finite este un aspect deosebit de important, deoarece în multe aplicații proiectantul trebuie să-și definească propriile automate finite, cu totul unice (în funcție de cerințele aplicației). 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 și tranziții de la o stare la alta;
- La ieșiri: semnale de ieșire, de control și de validare pentru restul sistemului;
- Automatul **NU** trebuie să conțină logică aritmetică, unități de execuție (căi de date) sau alte funcții combinaționale.

Cu alte cuvinte, automatul finit trebuie să conțină strict logica de trecere dintr-o stare în alta și de generare a ieșirilor specifice, fără nici o interferență cu alte module ale sistemului (trebuie să fie „pur”, complet separat de partea de execuție).



**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 instrucțiune *case*. Tot aici putem include registrul de stări; el poate fi specificat și 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 și 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 șanse să greșească)!

În cazul în care urmărim să obținem 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 ieșirile automatului finit prin registre, pentru a-i crește performanța.

### ***c) Instanțierea și inferența resurselor***

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

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

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

XILINX recomandă instanțierea 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 și BUFGCE.

Rațiunea acestor sugestii o găsim în faptul că astfel este mai ușor să schimbăm sau să portăm proiectul nostru pe alte suporturi fizice, realizate în tehnologii noi (mai avansate). În felul acesta, sunt mai puține constrângeri de sinteză și attribute care trebuie transmise mai departe (cel mai simplu este să păstrăm majoritatea atributelor și constrângerilor în fișierul *User Constraints File* (UCF), căci astfel informațiile critice sunt grupate într-un singur fișier).

Se recomandă de asemenea să creăm un bloc ierarhic separat pentru instanțierea acestor resurse. Deasupra blocului de la nivelul ierarhic cel mai înalt din proiectul nostru (*top-level*), vom crea un „wrapper” care va conține instanțierile 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*.

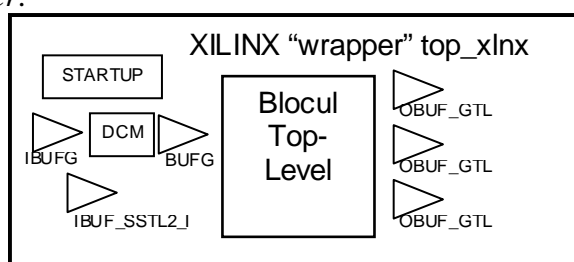


Figura 13.12 Schema de principiu a unui wrapper

#### d) Operația de Retiming

Această operație este suportată de principalii producători de software de susținere a proiectării. Instrumentul de sinteză urmărește să distribuie automat blocurile de logică combinațională dintre registre pentru a echilibra întârzierile pe căile de date combinaționale. Figura 13.13 ilustrează acest concept.

Înainte de Retiming



După Retiming

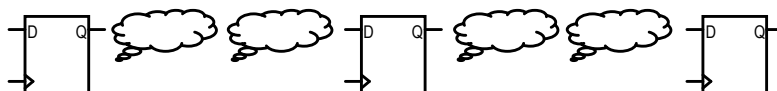
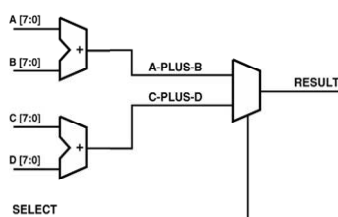


Figura 13.13 Operația de retiming



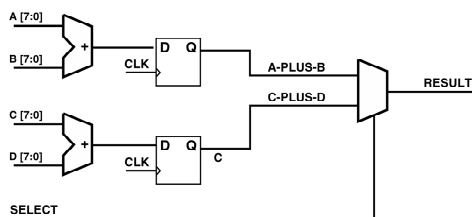
### 3. Desfășurarea lucrării

1. Se editează și se simulează numărătorul Moebius din figura 13.1 și se verifică obținerea corectă a formelor de undă din figura 13.2.
2. Se parcurg toți pașii necesari în vederea implementării numărătorului Moebius în cipul FPGA de pe placa NEXYS-2, folosindu-se și utilitarele din pachetul Foundation Series menționate în secțiunea 2.2.
3. Se editează, se simulează și se parcurgând toți pașii necesari în vederea implementării unui numărător zecimal și a unei unități aritmetico-logice în cipul FPGA de pe placa NEXYS-2.
4. Se editează și se simulează un dispozitiv universal numărător / registru de deplasare, parcurgându-se apoi toți pașii 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 și 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 și în ce condiții?
6. Fiind dat următorul circuit (figura 13.14):



**Figura 13.14** Circuitul inițial

s-a dorit *pipeline*-izarea lui, așa 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 aceeași 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 desfășurării 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 afișaje cu LED-uri cu 7 segmente

Numărul și principiul de funcționare al acestor afișaje este similar celui al afișajelor de pe placa Nexys3. Pinii dispozitivului FPGA la care sunt legați pinii afișajelor sunt redați în tabelele 13.1 și 13.2.

**Tabelul 13.1** *Conexiunile dintre dispozitivul FPGA și afișajul 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 poziții

Numărul și principiul de funcționare al acestor comutatoare este similar celui al comutatoarelor de pe placa Nexys3. Pinii dispozitivului FPGA la care sunt legați pinii comutatoarelor sunt redați 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 legați pinii comutatoarelor de tip „push button” sunt redați 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 și principiul de funcționare al acestor LED-uri este similar celui al LED-urilor de pe placa Nexys3. Pinii dispozitivului FPGA la care sunt legați pinii LED-urilor sunt redați 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). Interfața fiind foarte simplă și intuitivă, nu va fi prezentată aici.