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-şi „reveni” după apariția unui eveniment de tip metastabilitate.

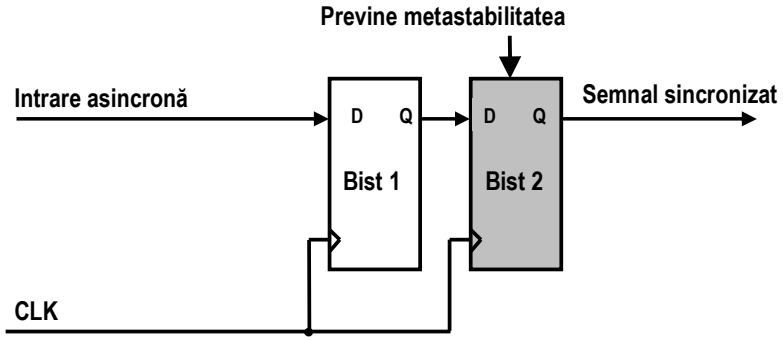

Figura 13.9 *Circuitul de sincronizare 1*

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.

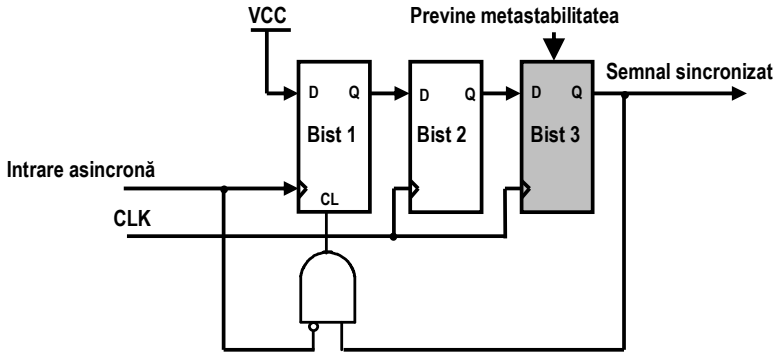

Figura 13.10 *Circuitul de sincronizare 2*

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 şi eficiente de scriere a codului şi 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 şi 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 utilitarelor 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 cascadate.
- 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 şi 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), deşi nu acesta era efectul dorit;
- Folosiți instrucțiuni de tip *case* pentru decodificatoare de mari dimensiuni, şi nu instrucțiuni de tipul *if-then-else*;
- Ordonați şi grupați funcțiile şi operatorii aritmetici şi logici – de exemplu, scrieți „A <= (B + C) + (D + E)”, şi nu „A <= B + C + D + E”;
- Evitați orice inferență nedorită a unui bistabil – trebuie acoperite toate ieşirile posibile pe fiecare ramură de cod. Acest lucru este uşor de realizat specificând instrucțiuni de atribuire de valori implicite înaintea instrucțiunilor *if-then-else* şi *case*;
- Unele resurse trebuie să fie instanțiate 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 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-şi 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 şi tranziții de la o stare la alta;
- La ieşiri: semnale de ieşire, de control şi 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 şi de generare a ieşirilor specifice, fără nici o interferență cu alte module ale sistemului (trebuie să fie „pur”, complet separat de partea de execuție).

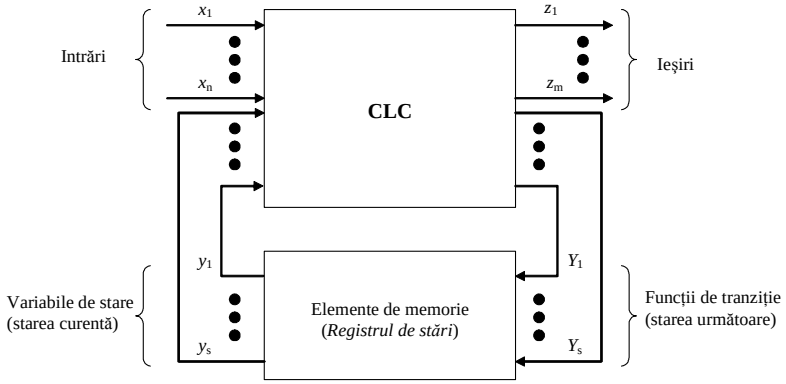

Figura 13.11 *Schema generică a unui automat finit*

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 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 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 ieşirile automatului finit prin registre, pentru a-i creşte performanța.

c) Instanțierea şi 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 greşit. 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 doreşte creşterea performanței sistemului sau diminuarea suprafeței de logică activă ocupată în cip.

Este indicat să limităm localizarea componentelor instanțiate 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 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 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*.

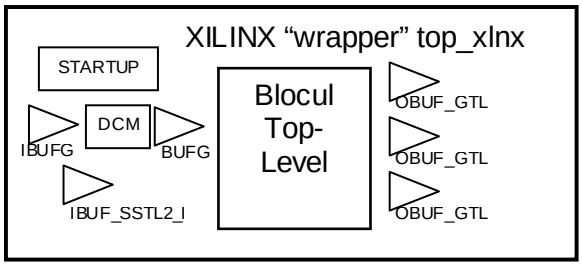

Figura 13.12 *Schema de principiu a unui* wrapper

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

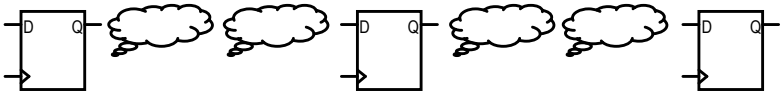

Figura 13.13 *Operaţia de* retiming

3. Desfăşurarea lucrării

1. Se editează şi se simulează numărătorul Moebius din figura 13.1 şi se verifică obținerea corectă a formelor de undă din figura 13.2.
2. Se parcurg toți paşii necesari în vederea implementării numărătorului Moebius în cipul FPGA de pe placa NEXYS-2, folosindu-se şi utilitarele din pachetul Foundation Series menționate în secțiunea 2.2.
3. Se editează, se simulează şi se parcurgând toți paşii necesari în vederea implementării unui numărător zecimal şi a unei unități 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 toți 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 condiții?
6. Fiind dat următorul circuit (figura 13.14):

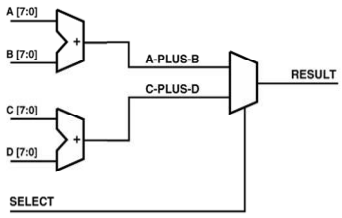

Figura 13.14 *Circuitul inițial*

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

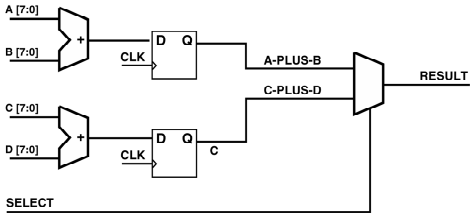

Figura 13.15 *Circuitul* pipeline-*izat*

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 componente principale similare cu cele ale plăcii Spartan3 Starter Kit care a fost prezentată în Anexa lucrării 12.

Principalele deosebiri (din punct de vedere al desfăşurării lucrărilor practice) constau în dispozitivul FPGA existent pe placă (Spartan3E-500 FG320) şi în modul său de programare, care se realizează 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 funcționare al acestor afişaje este similar celui al afişajelor de pe placa Spartan3 Starter Kit. Pinii dispozitivului FPGA la care sunt legați pinii afişajelor sunt redați 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
A	L18
B	F18
C	D17
D	D16
E	G14
F	J17
G	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 şi principiul de funcționare al acestor comutatoare este similar celui al comutatoarelor de pe placa Spartan3 Starter Kit. 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 şi principiul de funcționare al acestor comutatoare de tip „push button” este similar celui al comutatoarelor de tip „push button” de pe placa Spartan3 Starter Kit. 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 şi principiul de funcționare al acestor LED-uri este similar celui al LED-urilor de pe placa Spartan3 Starter Kit. 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 Spartan3*

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