CAPITOLUL 1

PROIECTAREA CU CIRCUITE FPGA

În prima parte a acestui capitol sunt prezentate principalele etape care trebuie parcurse pentru proiectarea cu circuite logice programabile în general și circuite FPGA în particular. În continuare, sunt prezentate principalele componente ale mediului de proiectare Xilinx Vivado. În restul capitolului se prezintă un exemplu de utilizare a mediului de proiectare Xilinx Vivado pentru proiectarea și implementarea unui ceas de timp real.

1.1 Proiectarea cu circuite logice programabile

1.1.1 Etape de proiectare

Pentru proiectarea sistemelor digitale utilizând circuite logice programabile PLD (*Programmable Logic Device*), cum sunt circuitele FPGA, se utilizează pachete de programe de proiectare asistată de calculator CAD (*Computer Aided Design*). Aceste pachete de programe asistă proiectantul în toate etapele procesului de proiectare. Astfel, majoritatea pachetelor CAD pentru circuitele programabile asigură următoarele funcții principale:

- *Descrierea* sistemului digital;
- *Sinteza* descrierii, care constă în transformarea descrierii într-o listă de conexiuni conținând porți logice elementare și interconexiunile dintre ele;
- *Simularea funcțională* a sistemului pe baza listei de conexiuni obținute, înainte de implementarea într-un anumit circuit;
- *Implementarea* sistemului într-un circuit prin adaptarea listei de conexiuni pentru o utilizare eficientă a resurselor disponibile ale circuitului;

- *Configurarea* (numită și programarea) circuitului pentru ca acesta să realizeze funcția dorită.

Figura 1.1 ilustrează etapele din procesul de proiectare a sistemelor digitale utilizând circuite logice programabile. Modulele IP (*Intellectual Property*) reprezintă module hardware complexe care sunt testate în mod extensiv și care pot fi utilizate în diferite proiecte pentru a reduce în mod semnificativ timpul de proiectare.

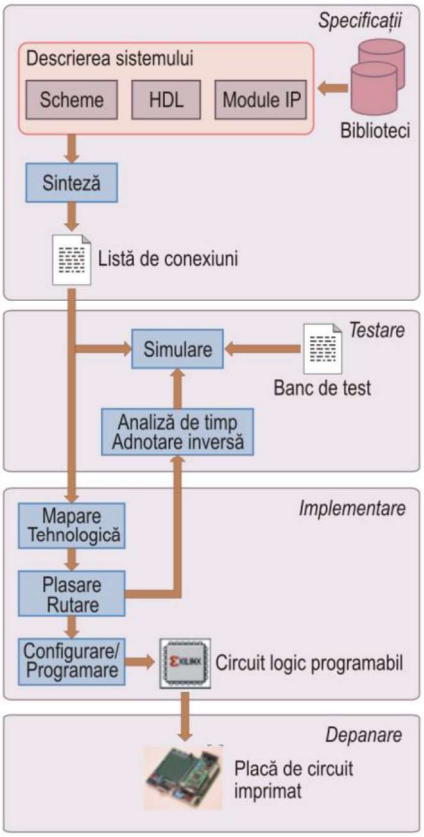

Figura 1.1. Etapele de proiectare a sistemelor digitale utilizând circuite logice programabile.

În continuare sunt descrise mai detaliat principalele etape de proiectare.

1.1.2 Descrierea sistemului

Există diferite metode pentru descrierea sistemelor digitale. Principalele metode de descriere sunt descrierea prin scheme logice, prin limbaje de descriere hardware HDL (*Hardware Description Language*) și prin diagrame de stare.

În mod tradițional, sistemele digitale sunt descrise prin scheme logice. Pentru această descriere se utilizează un editor schematic, care permite specificarea componentelor care trebuie utilizate și a modului în care acestea trebuie interconectate. Această metodă este ilustrată în figura 1.2. Circuitul din această figură detectează secvența binară 1010 aplicată la intrarea *X*. La detectarea acestei secvențe, ieșirea *Z* va fi setată la 1 logic.

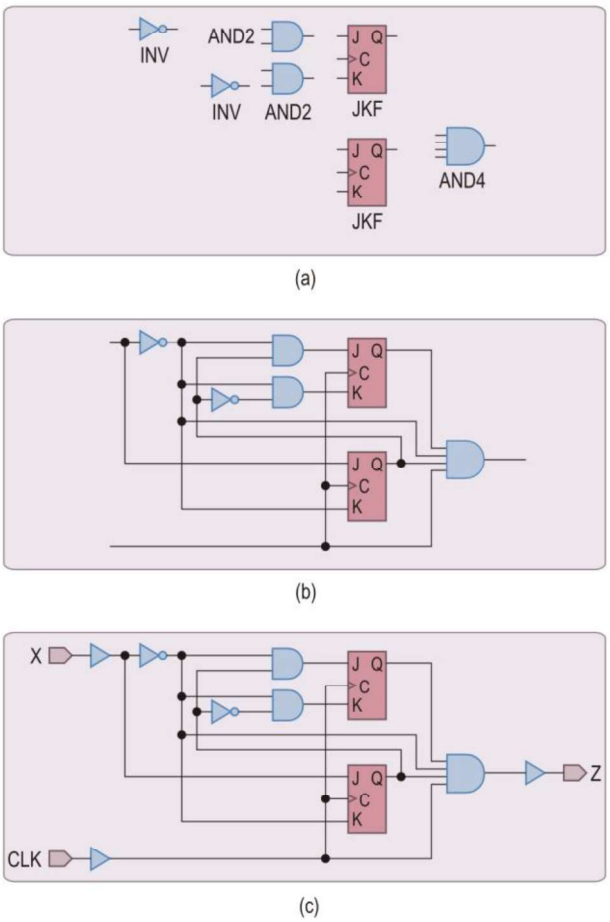

Figura 1.2. Etapele descrierii unui sistem digital utilizând scheme: (*a*) selectarea și amplasarea componentelor; (*b*) conectarea componentelor; (c) adăugarea porturilor de I/E.

La utilizarea schemelor logice, principalele etape pentru proiectarea unui sistem digital sunt prezentate în continuare.

1. Într-un editor schematic se selectează componentele necesare dintr-o bibliotecă de componente. Exemple de asemenea componente sunt porți elementare, multi-

plexoare, decodificatoare, numărătoare și unități aritmetice. În funcție de circuitul programabil care se va utiliza, proiectantul trebuie să selecteze o anumită bibliotecă de componente, deoarece există biblioteci care sunt specifice diferiților producători de circuite logice programabile și diferitelor familii de circuite. Circuitele dintr-o anumită familie diferă prin capacitatea lor, viteza circuitelor și capsula utilizată. În această etapă, nu este necesară specificarea exactă a circuitului care va utilizat dintr-o anumită familie.

2. Componentele selectate și plasate în cadrul unei scheme se interconectează prin fire de legătură. Proiectantul realizează interconectarea componentelor pentru a obține configurația necesară pentru o anumită aplicație.
3. Se adaugă și se etichetează porturile de I/E. Aceste porturi definesc intrările și ieșirile sistemului, permițând aplicarea semnalelor la pinii de intrare ai sistemului digital și preluarea semnalelor de ieșire generate de sistem la pinii de ieșire. Semnalele de intrare sunt aplicate la intrările sistemului digital prin intermediul unor buffere de intrare, iar semnalele de ieșire sunt preluate de la sistem prin intermediul unor buffere de ieșire. Aceste buffere izolează sistemul digital față de exterior. În unele cazuri, bufferele de I/E sunt adăugate în mod automat de sistemul CAD.

Pe lângă schemele logice, o altă posibilitate pentru descrierea sistemelor digitale este utilizarea unor limbaje de descriere hardware (HDL). Aceste limbaje sunt utilizate actualmente pe scară largă, fiind preferate pentru descrierea sistemelor cu complexitate ridicată, datorită următoarelor avantaje:

- Posibilitatea unei descrieri funcționale a sistemelor, aceasta fiind o descriere de nivel mai înalt, fără detalierea structurii la nivelul porților elementare. Astfel, timpul necesar pentru descrierea sistemelor complexe se reduce în mod semnificativ.
- Independența descrierilor HDL față de diferite tipuri de circuite. În timp ce schemele logice sunt realizate cu componente de bibliotecă specifice unei anumite familii de circuite programabile, descrierile HDL sunt complet independente de un anumit circuit, astfel încât aceeași descriere se poate utiliza pentru implementarea sistemului într-un anumit circuit FPGA, dar și într-un alt tip de circuit logic programabil, de exemplu, într-o rețea logică programabilă.
- Posibilitatea modificării mai simple a descrierii HDL a unui sistem, datorită faptului că o asemenea descriere reprezintă și o documentare a sistemului.

Există diferite limbaje de descriere hardware, dar cele mai utilizate dintre acestea sunt limbajele VHDL (VHSIC *Hardware Description Language*) și Verilog. Aceste limbaje sunt standardizate de organizația IEEE (*Institute of Electrical and*

Electronics Engineers). Figura 1.3 prezintă o descriere posibilă în limbajul VHDL a circuitului ilustrat în figura 1.2.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity detect is
    Port ( CLK : in STD_LOGIC;
           X   : in STD_LOGIC;
           Z   : out STD_LOGIC);
end detect;

architecture automat_stare of detect is
    type TIP_STARE is (A, B, C, D);
    signal Stare: TIP_STARE;
begin
    tranz_stare: process (CLK)
    begin
        if RISING_EDGE (CLK) then
            case Stare is
                when A =>
                    if (X = '1') then Stare <= B; end if;
                when B =>
                    if (X = '0') then Stare <= C; end if;
                when C =>
                    if (X = '0') then Stare <= A;
                    else Stare <= D; end if;
                when D =>
                    if (X = '0') then Stare <= C;
                    else Stare <= B; end if;
            end case;
        end if;
    end process tranz_stare;
    Z <= '1' when Stare = D else '0';
end automat_stare;
```

Figura 1.3. Descrierea în limbajul VHDL a circuitului ilustrat în figura 1.2.

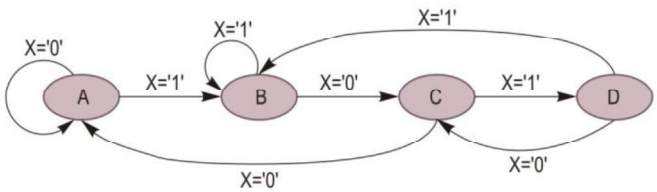

Figura 1.4. Diagrama de stare echivalentă a circuitului reprezentat prin schema din figura 1.2 și descrierea în limbajul VHDL din figura 1.3.

Pentru descrierea automatelor cu stări finite se utilizează pe scară largă diagramele de stare. Sistemele CAD pentru proiectarea cu circuite logice programabile pot conține editoare pentru diagramele de stare, care permit specificarea sub formă grafică a stărilor sistemului, a tranzițiilor între stări și a semnalelor de ieșire care trebuie generate în fiecare stare. O diagramă de stare va fi compilată de către sistemul CAD

într-o reprezentare internă sau într-o descriere HDL, care poate fi simulată și utilizată apoi pentru implementarea automatului într-un anumit circuit. Figura 1.4 prezintă o diagramă de stare echivalentă cu circuitul ilustrat în figura 1.2 și descrierea în limbajul VHDL din figura 1.3.

1.1.3 Sinteza sistemului

După descrierea sistemului digital, etapa următoare a procesului de proiectare este cea de sinteză a sistemului. Sinteza constă în translatarea schemei logice, a descrierii HDL sau a diagramei de stare într-o *listă de conexiuni*. Această translatare se realizează cu ajutorul unui program de sinteză al sistemului CAD. Lista de conexiuni ("*netlist*") este o descriere compactă a sistemului digital sub formă textuală, în care sunt specificate componentele sistemului, interconexiunile dintre acestea și pinii de intrare/ieșire. Lista de conexiuni este prelucrată de celelalte module ale sistemului CAD pentru realizarea etapelor următoare din procesul de proiectare.

Există diferite formate pentru listele de conexiuni. Formatul EDIF (*Electronic Digital Interchange Format*) este cel mai utilizat și reprezintă un standard industrial. Pe lângă acest format standard, se pot utiliza diferite formate care sunt specifice anumitor producători de circuite. Un exemplu este formatul XNF (*Xilinx Netlist Format*), care este un format proprietar al firmei Xilinx. O altă posibilitate este utilizarea unui limbaj de descriere hardware ca format pentru lista de conexiuni. De exemplu, sistemul CAD poate utiliza o reprezentare structurală a sistemului într-un limbaj de descriere hardware specificat de proiectant.

Relația dintre schema logică a unui circuit simplu și un format posibil al unei liste de conexiuni este ilustrată în figura 1.5. În prima parte a listei de conexiuni sunt declarate componentele din cadrul schemei, iar în a doua parte sunt specificate conexiunile dintre componente. Numele componentelor sunt G1 .. G7, iar numele conexiunilor sunt N1 .. N10. Aceste nume sunt fie cele specificate de proiectant, fie cele asignate în mod automat de sistemul CAD.

În circuitul ilustrat în figura 1.5, există două inversoare (G1 și G2), două porți ȘI cu două intrări (G3 și G4), o poartă ȘI cu patru intrări (G7) și două bistabile JK (G5 și G6). Inversoarele au un pin de intrare IN, un pin de ieșire OUT, un pin de alimentare și un pin de masă. Pinii de alimentare și de masă nu sunt specificați în mod explicit. Similar, porțile ȘI cu două intrări au doi pini de intrare IN1 și IN2, un pin de ieșire OUT, un pin de alimentare și un pin de masă. Bistabilele au doi pini pentru intrările de date J și K, un pin pentru intrarea de ceas C și un pin pentru ieșirea Q, pe lângă pinii de alimentare și de masă. O conexiune este reprezentată prin listarea tuturor pinilor care sunt conectați împreună. Semnalele de intrare *X* și *CLK* sunt conectate la pinii de intrare cu aceleași nume ai circuitului, iar semnalul de ieșire *Z* este conectat la pinul de ieșire al circuitului.

Proiectantul poate specifica diferite criterii de optimizare de care să se țină cont în timpul procesului de sinteză. Exemple de asemenea criterii de optimizare sunt

minimizarea numărului de porți elementare necesare, obținerea vitezei maxime de funcționare a circuitului, sau minimizarea puterii consumate. Proiectantul poate experimenta cu diferite criterii de optimizare pentru a obține soluția cea mai convenabilă pentru aplicația respectivă.

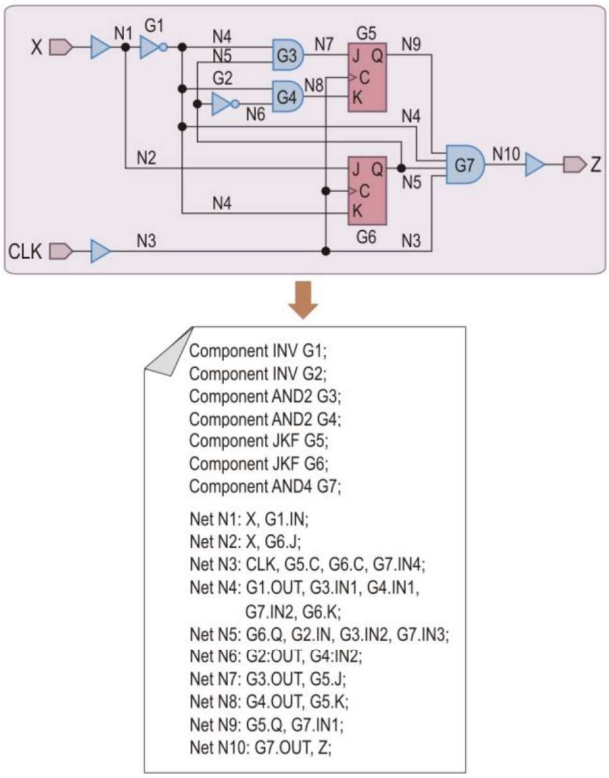

Figura 1.5. Relația dintre schema logică și lista de conexiuni pentru circuitul din figura 1.2.

1.1.4 Simularea funcțională

În această etapă se utilizează un program simulator pentru verificarea funcționării sistemului proiectat înainte de implementarea acestuia într-un circuit logic programabil. Această verificare se referă doar la aspectele funcționale ale sistemului, fără a se lua în considerare întârzierile semnalelor, care vor fi cunoscute numai după implementare. Pentru verificarea funcțională, proiectantul furnizează simulatorului mai multe combinații ale valorii semnalelor de intrare; o asemenea combinație este numită *vector de test*. De asemenea, proiectantul poate specifica valorile semnalelor de ieșire care trebuie generate de sistem pentru fiecare vector de test.

Simulatorul aplică vectorii de test la intrările sistemului unul câte unul, determină semnalele de ieşire care sunt generate de sistem şi le compară cu valorile semnalelor care au fost specificate de proiectant. Dacă apar diferenţe, simulatorul afişează mesaje care indică diferenţele apărute. Proiectantul va efectua modificările necesare ale descrierii sistemului pentru a corecta erorile apărute, va efectua sinteza descrierii modificate şi va executa din nou simularea funcţională. Aceste etape vor fi repetate până când sistemul va funcţiona conform cerinţelor.

Figura 1.6 ilustrează modul în care pot fi vizualizate pe ecranul calculatorului semnalele de intrare şi de ieşire ale detectorului de secvenţă utilizat ca exemplu în secţiunile precedente.

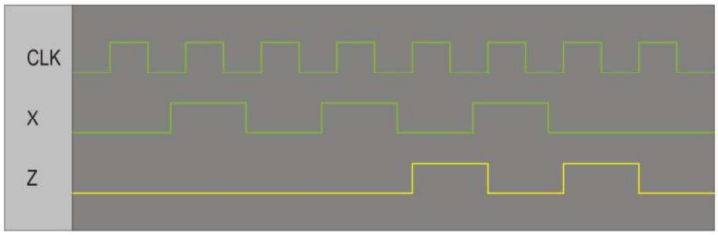

Figura 1.6. Semnalele de intrare şi de ieşire ale detectorului de secvenţă afişate la simularea funcţională a circuitului.

1.1.5 Maparea tehnologică

Etapele următoare din procesul de proiectare realizează implementarea sistemului proiectat într-un circuit logic programabil. Prima etapă a implementării este cea de *mapare tehnologică*. Această etapă constă dintr-o serie de operaţii care procesează lista de conexiuni şi o adaptează la particularităţile şi resursele disponibile ale circuitului utilizat pentru implementare. Operaţiile executate în această etapă variază în funcţie de sistemul de sinteză. Cele mai obişnuite operaţii sunt adaptarea la elementele fizice ale circuitului, optimizarea şi verificarea regulilor de proiectare (de exemplu, testarea depăşirii numărului pinilor de I/E disponibili în cadrul circuitului). În timpul acestei etape, proiectantul selectează tipul circuitului logic programabil care va fi utilizat, capsula circuitului integrat, viteza şi alte opţiuni specifice circuitului respectiv.

În urma execuţiei operaţiilor din etapa de mapare tehnologică, se generează un raport detaliat al rezultatelor obţinute. Pe lângă mesaje de eroare şi de avertizare, se creează şi o listă cu resursele utilizate din cadrul circuitului.

Figura 1.7 ilustrează etapa de mapare tehnologică pentru circuitul utilizat ca exemplu. După cum se observă, schema circuitului a fost modificată pentru a utiliza bistabile D în locul bistabilelor JK, iar porţile ŞI au fost înlocuite cu porţi ŞI-NU. Se menţionează că aceste transformări sunt efectuate asupra listei de conexiuni care s-a obţinut în urma etapei de sinteză, schema din figura 1.7 fiind doar ilustrativă.

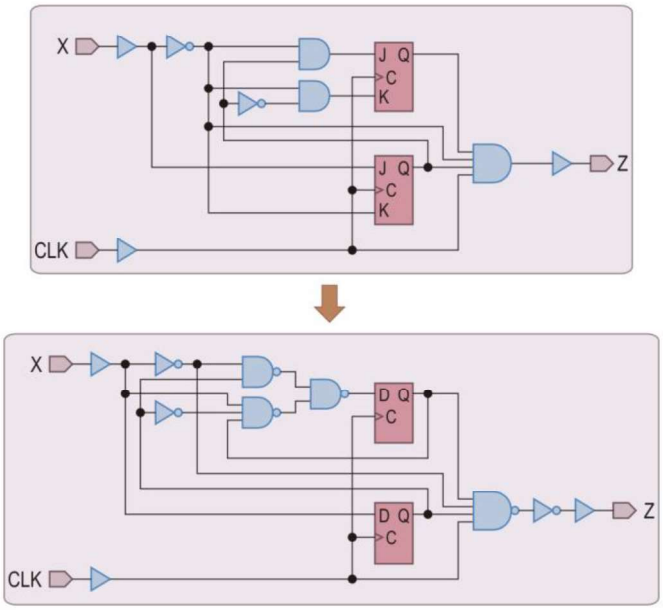

Figura 1.7. Ilustrarea etapei de mapare tehnologică pentru circuitul din figura 1.2.

1.1.6 Plasarea și rutarea

Operațiile de plasare și rutare sunt executate atunci când se utilizează un circuit FPGA pentru implementare. Pentru proiectarea cu alte circuite programabile, operația echivalentă este numită adaptare ("*fitting*"). *Plasarea* este procesul de selectare a unor module sau blocuri logice ale circuitului logic programabil care vor fi utilizate pentru implementarea diferitelor funcții ale sistemului digital. *Rutarea* constă în interconectarea acestor blocuri logice utilizând resursele de rutare disponibile ale circuitului.

Majoritatea sistemelor CAD realizează operațiile de plasare și rutare în mod automat, astfel încât utilizatorul nu trebuie să cunoască detaliile arhitecturii circuitului. Anumite sisteme permit utilizatorilor experți plasarea și rutarea manuală a unor porțiuni critice ale sistemului digital pentru a obține performanțe superioare.

Figura 1.8 ilustrează plasarea și rutarea listei de conexiuni obținute în urma mapării tehnologice a circuitului utilizat ca exemplu. După selectarea blocurilor logice care vor fi utilizate pentru implementarea circuitului, acestea se configurează pentru implementarea unor porțiuni ale circuitului. Pentru interconectarea semnalelor generate de diferitele blocuri logice se utilizează resursele de rutare disponibile. Aceste resurse sunt indicate în figură prin linii orizontale și verticale. Intrările și ieșirile utilizate ale blocurilor logice se conectează la liniile de rutare prin puncte de conexiune

programabile (reprezentate în figură prin cercuri), iar liniile de rutare sunt interconectate prin comutatoare programabile.

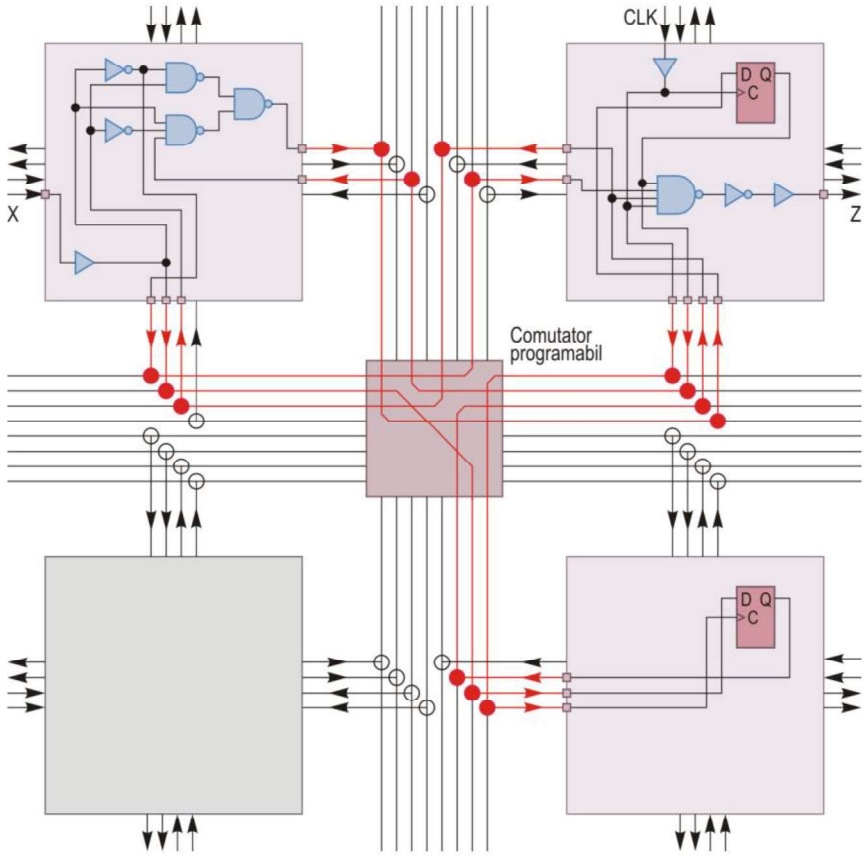

Figura 1.8. Ilustrarea etapelor de plasare și rutare pentru circuitul din figura 1.7.

Operațiile de plasare și rutare pot necesita un timp ridicat pentru execuție în cazul sistemelor digitale complexe, deoarece sunt necesare operații complexe pentru determinarea și configurarea blocurilor logice necesare din circuitul programabil, interconectarea corectă a acestora și verificarea faptului că sunt asigurate cerințele de performanță specificate în timpul proiectării.

1.1.7 Analiza de timp

Pachetele de programe CAD pentru proiectarea sistemelor digitale conțin, de obicei, un program numit analizor de timp, care poate furniza informații despre întârzierile semnalelor. Aceste informații se referă atât la întârzierile introduse de blocurile

logice, cât și la întârzierile datorate interconexiunilor. Analizorul de timp poate afișa aceste informații în diferite moduri, de exemplu, prin ordonarea conexiunilor în ordinea descrescătoare a întârzierilor semnalelor. Proiectantul poate utiliza informațiile despre întârzierile semnalelor pentru a realiza o nouă simulare a sistemului, în care să se țină cont de aceste întârzieri. Operația prin care se furnizează simulatorului informații detaliate despre întârzierile semnalelor se numește *adnotare inversă* ("*back-annotation*").

1.1.8 Configurarea sau programarea circuitului

Operația de *configurare* se referă la circuitele programabile bazate pe memorii volatile RAM statice și constă din încărcarea informațiilor de configurare în memoria circuitului. Operația de *programare* se referă la circuitele logice programabile bazate pe memorii nevolatile (cum sunt circuitele care conțin anti-fuzibile). Această operație se execută similar cu cea de configurare, dar informațiile de configurare sunt păstrate și după întreruperea tensiunii de alimentare.

La sfârșitul operațiilor de plasare și rutare, se generează un fișier care conține toate informațiile necesare pentru configurarea circuitului. Aceste informații se referă atât la configurarea blocurilor logice ale circuitului, cât și la specificarea interconexiunilor dintre blocurile logice. Fișierul în care se înscriu aceste informații conține un șir de biți ("*bitstream*"), fiecare bit indicând starea închisă sau deschisă a unui comutator. Circuitele logice programabile conțin un număr mare de asemenea comutatoare. Un comutator poate fi implementat printr-un tranzistor sau o celulă de memorie. Un bit de 1 din șirul de biți va determina închiderea unui comutator și, deci, stabilirea unei conexiuni. Biții din fișierul de configurare sunt aranjați într-un anumit format pentru a realiza o corespondență între un bit și comutatorul corespunzător.

Conținutul fișierului de configurare se transferă la circuitul logic programabil, aflat, de obicei, pe o placă de circuit imprimat sau o placă de dezvoltare împreună cu alte circuite. Comutatoarele circuitului se închid sau rămân deschise în funcție de valorile biților din șirul de configurare.

Din cauza utilizării unei memorii volatile, circuitul trebuie configurat din nou după fiecare întrerupere a tensiunii de alimentare. Informațiile de configurare pot fi păstrate însă într-o memorie nevolatilă, iar circuitul poate fi configurat în mod automat din această memorie nevolatilă la aplicarea tensiunii de alimentare.

Configurarea sau programarea se pot realiza de la un calculator utilizând mai multe tipuri de interfețe. De exemplu, se pot utiliza interfețe seriale cum sunt SPI (*Serial Peripheral Interface*) sau USB (*Universal Serial Bus*). Placa cu circuitul logic programabil trebuie să conțină un conector pentru interfața utilizată.

O altă posibilitate este utilizarea unui cablu special și a unei metodologii de configurare propusă de organizația JTAG (*Joint Test Advisory Group*). Această metodologie, cunoscută sub numele de "*Boundary-Scan*", a fost standardizată de organizațiile IEEE și ANSI ca standardul 1149.1, reprezentând un set de reguli de proiectare

care facilitează configurarea sau programarea circuitelor, testarea și depanarea acestora. Un capăt al cablului JTAG se conectează la un port USB al calculatorului, iar celălalt capăt se conectează la un număr de cinci pini speciali de pe placa de dezvoltare. Informațiile de configurare sunt transferate serial la circuitul logic programabil. Un asemenea cablu permite și testarea sistemului digital implementat prin citirea unor informații (valori ale semnalelor sau conținutul unor locații de memorie) de la circuitul logic programabil în timpul funcționării, transferul acestor informații la calculator și afișarea lor pe ecran.

Figura 1.9 prezintă conectarea unui cablu JTAG în modul "*Boundary-Scan*" la un sistem de dezvoltare conținând unul sau mai multe circuite logice programabile. Firele de legătură se conectează cu un capăt la pinii JTAG ai cablului, iar cu celălalt capăt la pinii JTAG corespunzători ai plăcii de dezvoltare. Un asemenea cablu poate fi utilizat fie pentru configurarea unui singur circuit programabil, fie a mai multor circuite conectate într-un lanț "*Boundary-Scan*".

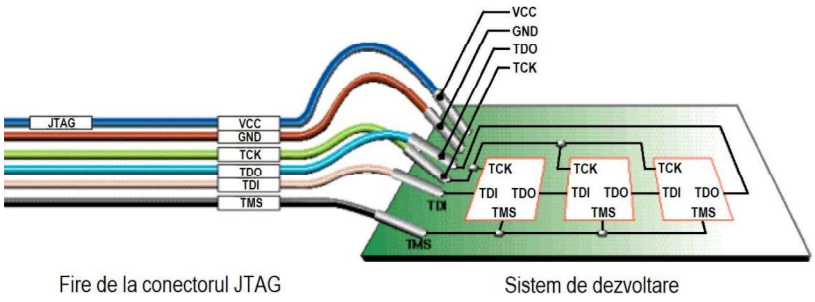

Figura 1.9. Conectarea unui cablu JTAG la un sistem de dezvoltare în modul "*Boundary-Scan*".

Tabelul 1.1 prezintă denumirea și semnificația semnalelor JTAG care sunt utilizate pentru configurarea circuitelor logice programabile.

Tabelul 1.1. Denumirea și semnificația semnalelor JTAG utilizate pentru configurarea circuitelor logice programabile.

Denumire	Semnificație
VCC	Alimentare - Tensiunea de alimentare (5 V, 3,3 V sau 2,5 V)
GND	Masă - Referința pentru masa electrică
TCK	*Test Clock* - Semnal de ceas pentru circuitele logice programabile
TDO	*Test Data Out* - Semnal pentru citirea datelor de la circuitele logice programabile
TDI	*Test Data In* - Semnal pentru transmiterea instrucțiunilor și datelor la circuitele logice programabile
TMS	*Test Mode Select* - Semnal decodificat de controlerul JTAG pentru controlul operațiilor

1.1.9 Verificarea și depanarea sistemului

În această ultimă etapă a procesului de proiectare, se verifică funcționarea sistemului digital în condiții reale. O funcționare necorespunzătoare a sistemului se poate datora nerespectării specificațiilor de proiectare, a specificațiilor circuitului utilizat pentru implementare, a unor aspecte legate de întârzierile semnalelor etc. Depanarea poate fi simplificată dacă circuitul se configurează astfel încât să conțină unele module speciale care permit citirea valorii unor semnale în timpul funcționării și transferul acestor informații la calculator, utilizând un cablu JTAG și un program special pentru afișarea formei de undă a semnalelor dorite.

1.2 Mediul de proiectare Xilinx Vivado

Mediul de proiectare Xilinx Vivado Design Suite integrează toate utilitarele necesare pentru proiectarea sistemelor digitale utilizând circuitele FPGA Xilinx din seria 7, UltraScale, UltraScale+, ca și circuitele SoC (*System-on-Chip*) din seriile Zynq-7000 și Zynq UltraScale+. Pentru generațiile precedente de circuite FPGA Xilinx, trebuie să se utilizeze pachetul software Xilinx ISE Design Suite. Mediul Vivado Design Suite pune la dispoziție o interfață grafică numită Vivado Integrated Design Environment (IDE). Această interfață grafică permite invocarea utilitarelor de proiectare pentru executarea operațiilor necesare în diferitele etape ale procesului de proiectare cu circuite FPGA, cum sunt descrierea sistemului, sinteza, implementarea, analiza proiectului, configurarea circuitului, verificarea și depanarea. Facilitățile de analiză a proiectului cuprind simularea logică în diferitele etape ale procesului de proiectare, definirea constrângerilor, verificarea regulilor de proiectare, analiza de timp, analiza puterii consumate și vizualizarea structurii proiectului.

Există două moduri care se pot utiliza pentru invocarea utilitarelor de proiectare individuale integrate în mediul Vivado Design Suite. Primul mod constă în invocarea acestor utilitare din interfața grafică Vivado IDE; aceasta presupune crearea de către utilizator a unui proiect cu ajutorul interfeței grafice Vivado IDE și includerea fișierelor sursă în proiect. Interfața grafică Vivado IDE gestionează apoi fișierele sursă incluse în proiect, memorează rezultatele rulării, generează rapoarte ale proiectului și salvează în mod automat starea proiectului. Al doilea mod constă în utilizarea unor comenzi Tcl (*Tool Command Language*) sau a unor fișiere script Tcl. În acest mod, utilizatorul are un control total asupra procesului de proiectare, dar diferitele utilitare de proiectare nu gestionează în mod automat fișierele sursă și nu salvează în mod automat starea proiectului. Comenzile Tcl individuale pot fi introduse în panoul *Tcl Console* din interfața grafică Vivado IDE, iar fișierele script Tcl conținând secvențe de comenzi Tcl pot fi lansate în execuție din același panou. Este posibil să se utilizeze și interpretorul de comenzi Vivado Tcl, care se deschide în afara interfeței grafice Vivado IDE, pentru a introduce comenzi Tcl sau pentru a rula fișiere script Tcl.

Aceste fișiere se pot utiliza pentru a executa întregul proces de proiectare sau pentru a executa părți ale procesului de proiectare.

Figura 1.10 ilustrează procesul de proiectare utilizând mediul de proiectare Xilinx Vivado Design Suite.

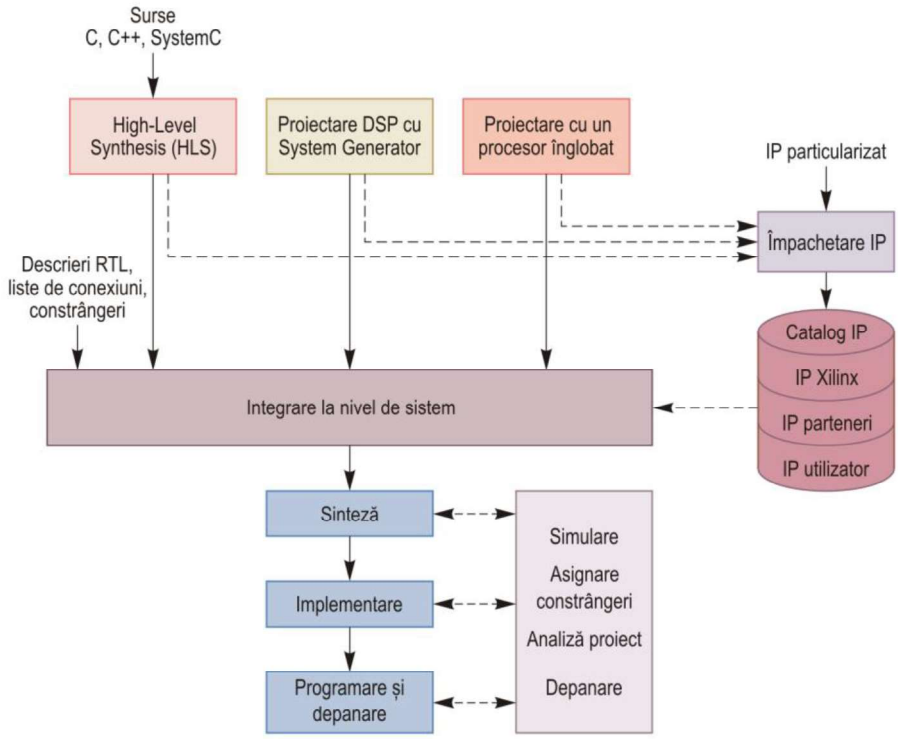

Figura 1.10. Procesul de proiectare utilizând mediul de proiectare Xilinx Vivado Design Suite.

Mediul de proiectare Xilinx Vivado Design Suite permite utilizarea unor fișiere sursă de diferite tipuri conținând descrierea sistemului digital care trebuie implementat. Unul din tipurile de fișiere sursă conține descrierea sistemului digital sau a unei părți a acestuia la nivelul transferurilor între registre (RTL – *Register Transfer Level*). La acest nivel, sistemul constă din componente interconectate, cum sunt multiplexoare, decodificatoare, sumatoare, bistabile, registre, numărătoare și memorii. Descrierile RTL pot fi specificate în limbajele de descriere hardware VHDL, Verilog sau SystemVerilog.

Un alt tip de fișier sursă este reprezentat de o listă de conexiuni sintetizată, specificată fie în formatul standard EDIF, fie ca o descriere structurală Verilog. Un asemenea fișier poate fi creat cu un utilitar de sinteză al unui producător terț sau cu un alt utilitar de sinteză Xilinx, cum este utilitarul XST (*Xilinx Synthesis Technology*) al mediului de proiectare Xilinx ISE Design Suite. Fișierele de constrângeri pentru

specificarea cerințelor și a restricțiilor de proiectare pot fi, de asemenea, incluse ca fișiere sursă într-un proiect Vivado. Aceste fișiere pot fi specificate în formatul SDC (*Synopsys Design Constraints*), care este un standard industrial, și în formatul propriu al Xilinx, XDC (*Xilinx Design Constraints*), care este o extensie a formatului SDC.

Mediul de proiectare Vivado Design Suite permite adăugarea unor module IP (*Intellectual Property*) la proiect. Aceste module pot fi adăugate din catalogul IP pus la dispoziție de mediul Vivado Design Suite. Acest catalog conține module IP furnizate de firma Xilinx și partenerii săi și poate fi extins cu module IP particularizate create de utilizator și împachetate într-un format standard cu ajutorul utilitarului Vivado *IP Packager*. Se pot utiliza și module create cu utilitarul CORE Generator al mediului de proiectare Xilinx ISE Design Suite. Modulele IP ale firmei Xilinx, ale partenerilor și ale utilizatorului pot fi particularizate cu un utilitar de configurare. Un alt utilitar Vivado pus la dispoziție pentru a lucra cu module IP este *IP Integrator*, care permite crearea unor subsisteme complexe, numite proiecte bloc. Aceste proiecte bloc sunt subsisteme IP conținând module IP configurate de utilizator și conexiunile dintre aceste module IP.

În mediul de proiectare Vivado Design Suite sunt incluse și utilitare de proiectare de nivel mai înalt. Unul dintre acestea permite crearea unor sisteme înglobate bazate pe circuite SoC cum sunt Zynq-7000 și Zynq UltraScale+, sau pe procesorul Xilinx MicroBlaze. Se poate folosi utilitarul *IP Integrator* pentru instanțierea și configurarea modulului procesorului, selectarea dispozitivelor periferice, configurarea setărilor hardware și conectarea acestor componente pentru a crea un sistem înglobat complex. Pentru interconectarea modulelor IP se utilizează magistrala AMBA AXI (*Advanced eXtensible Interface*). Magistrala AXI face parte din familia de magistrale ARM AMBA (*Advanced Microcontroller Bus Architecture*) dezvoltată de ARM Ltd. pentru microcontrolere. Această familie de magistrale este utilizată și pentru interconectarea blocurilor funcționale din circuitele SoC. Versiunea magistralei AXI care este utilizată este AXI4, care este inclusă în specificațiile AMBA 4.0. După implementarea proiectului hardware, acesta este exportat în mediul de dezvoltare software Vitis pentru dezvoltarea și depanarea aplicației software care rulează pe procesorul înglobat.

Un alt utilitar care este integrat în mediul de proiectare Vivado Design Suite este *System Generator*, care permite utilizarea mediului de modelare și simulare Simulink al firmei MathWorks pentru proiectarea unor module de procesare digitală a semnalelor (DSP – *Digital Signal Processing*) implementate în circuite FPGA. Pentru aceste module, se poate utiliza limbajul de programare MATLAB (*Matrix Laboratory*), care este deosebit de util pentru calcule numerice. Într-un proiect Vivado se pot adăuga ca și fișiere sursă fișiere existente conținând modele de proiecte *System Generator*, iar din interfața grafică Vivado IDE pot fi create noi module DSP, pentru care se va lansa în execuție utilitarul *System Generator*. Un modul DSP poate fi împachetat ca și modul IP și poate fi adăugat la catalogul IP, putând fi integrat ulterior în proiect sau utilizat în proiecte viitoare.

Utilitarul *High-Level Synthesis* (HLS) permite descrierea unor algoritmi folosind limbajele C, C++ sau SystemC. După descrierea unui algoritm într-unul din aceste limbaje, acesta poate fi compilat, poate fi executat pentru validarea faptului că algoritmul este corect din punct de vedere funcțional, iar apoi poate fi sintetizat într-un modul RTL. Modulul implementat poate fi analizat și depanat, iar apoi poate fi instanțiat în proiect sau poate fi împachetat într-un modul IP pentru a fi utilizat în proiect sau în proiecte viitoare. Utilitarul HLS pune la dispoziție mai multe facilități pentru generarea unei implementări optime a algoritmului.

1.3 Exemplu de utilizare a mediului Xilinx Vivado

Această secțiune prezintă un exemplu de proiectare pentru care se utilizează limbajul VHDL pentru descrierea sistemului și mediul de proiectare Xilinx Vivado Design Suite pentru implementare. Exemplul ilustrează etapele procesului de proiectare cu circuite FPGA și demonstrează principalele funcții oferite de mediul de proiectare Xilinx Vivado.

1.3.1 Prezentarea proiectului

Proiectul prezentat ca exemplu reprezintă un ceas de timp real care păstrează și afișează ora, minutul și secunda, dar nu ține evidența zilei, a lunii și a anului. Proiectul este destinat implementării pe o placă de dezvoltare având patru butoane și un afișaj cu șapte segmente conținând opt cifre. Ceasul de timp real are două moduri de funcționare, modul normal și modul de setare a orei. În modul normal, se afișează ora, minutul și secunda sub forma hh-mm-ss, caracterele "-" pâlpâind o dată în fiecare secundă. În modul de setare a orei, se poate seta în mod independent ora, minutul și secunda.

Ceasul de timp real pornește în modul normal de funcționare, cu ora setată la 12-00-00 și actualizată în fiecare secundă. Ceasul trece în modul de setare a orei prin apăsarea butonului *Mode*, când cele două cifre ale orei vor pâlpâi. Prin apăsarea butonului *Up* sau *Down*, ora va fi incrementată, respectiv decrementată. După setarea orei, prin apăsarea butonului *Mode* cele două cifre ale minutului vor pâlpâi și se poate seta minutul cu butonul *Up* sau *Down*. Prin apăsarea din nou a butonului *Mode*, cele două cifre ale secundei vor pâlpâi și se poate seta secunda cu butonul *Up* sau *Down*. Apăsând butonul *Mode* încă o dată, ceasul de timp real va părăsi modul de setare a orei și va trece în modul normal de funcționare. Ceasul poate fi resetat prin apăsarea butonului *Rst*.

Schema bloc a ceasului de timp real este ilustrată în figura 1.11. Modulul control generează semnalele de comandă necesare pentru actualizarea ceasului în modul normal de funcționare și pentru incrementarea sau decrementarea orei, a minutului și a secundei în modul de setare a orei. Modulul time_cnt utilizează semnalele generate de modulul control pentru actualizarea orei și setarea orei. Modulul displ7seg_blink

este un multiplexor pentru afișajul cu șapte segmente. La intrarea acestui modul se aplică informațiile care trebuie afișate, iar modulul va genera semnalele pentru selecția cifrei active a afișajului și pentru aprinderea segmentelor cifrei active. Opțional, acest modul poate realiza pâlpâirea oricărei cifre afișate. Modulul debounce conține o logică simplă pentru filtrarea oscilațiilor unui buton; se utilizează trei asemenea module, câte unul pentru fiecare din butoanele *Mode*, *Up* și *Down*. Modulul principal clock conține instanțierea celorlalte module și aplică semnalele la intrările multiplexorului pentru afișajul cu șapte segmente.

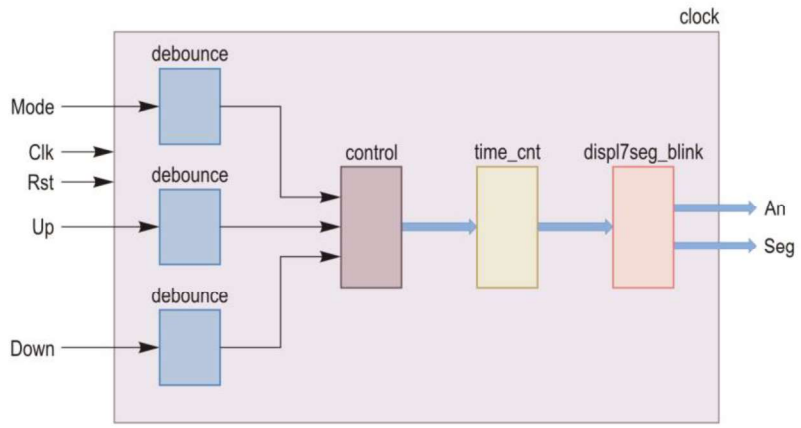

Figura 1.11. Schema bloc a ceasului de timp real.

Semnalele de intrare ale ceasului de timp real sunt următoarele:

- *Clk*: Semnalul de ceas al sistemului, cu frecvența de 100 MHz.
- *Rst*: Semnal de resetare; inițializează ceasul și resetează ora la 12-00-00.
- *Mode*: Semnal de mod; comută ceasul între modul normal și modul de setare a orei. În modul de setare a orei, prin apăsarea repetată a acestui buton este posibilă setarea orei, a minutului și a secundei.
- *Up*: Semnal de incrementare; incrementează ora, minutul sau secunda în modul de setare a orei.
- *Down*: Semnal de decrementare; decrementează ora, minutul sau secunda în modul de setare a orei.

Semnalele de ieșire ale ceasului de timp real sunt următoarele:

- *An*(7:0): Semnale pentru selecția anodului activ al afișajului.
- *Seg*(7:0): Semnale pentru selecția catozilor (segmentelor) cifrei active a afișajului.

1.3.2 Codul sursă în limbajul VHDL

În această secțiune se prezintă codul sursă în limbajul VHDL al modulului de comandă, al modulului de actualizare a ceasului de timp real, al multiplexorului pentru afișajul cu șapte segmente și al modulului principal. Modulul pentru filtrarea oscilațiilor semnalelor de intrare va fi creat pe parcursul etapelor de proiectare și implementare.

1.3.2.1 Modulul de comandă

Pe lângă semnalele *Clk* și *Rst*, modulul de comandă control are ca intrări semnalele de la butoanele *Mode*, *Up* și *Down*, denumite *ModeIn*, *UpIn* și *DownIn*. Modulul contorizează ciclurile de ceas și activează semnalul *IncTime* după o secundă; acest semnal este utilizat în modul normal de funcționare pentru actualizarea ceasului de către modulul time_cnt. În modul de setare a orei, modulul de comandă generează semnalele pentru incrementarea și decrementarea orei, a minutului și a secundei. De exemplu, semnalele *IncHour* și *DecHour* sunt utilizate pentru incrementarea sau decrementarea orei. Acest modul generează și semnalele de comandă *BlinkHour*, *BlinkMin* și *BlinkSec* pentru a realiza pâlpâirea cifrelor orei, a minutului sau secundei în modul de setare a orei.

Modulul de comandă este implementat ca un automat de stare. Starea `run` corespunde modului normal de funcționare al ceasului de timp real. Automatul va trece în stările `set_hour`, `set_min` și `set_sec` atunci când butonul *Mode* este apăsat în mod repetat. Pentru fiecare din aceste stări, există stări separate pentru incrementarea sau decrementarea orei, a minutului și a secundei, în care se trece atunci când se apasă butonul *Up* sau *Down* (de exemplu, `set_sec_inc`, `set_sec_dec`).

Descrierea modulului de comandă este prezentată în Exemplul 1.1.

Exemplul 1.1

```
----------------------------------------------------------------------------
-- Nume proiect: clock
-- Nume modul:   control
-- Descriere:    Modul de comanda pentru ceasul de timp real
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control is
   Port ( Clk       : in  STD_LOGIC;
          Rst       : in  STD_LOGIC;
          ModeIn    : in  STD_LOGIC;
          UpIn      : in  STD_LOGIC;
          DownIn    : in  STD_LOGIC;
          IncTime   : out STD_LOGIC;
          IncHour   : out STD_LOGIC;
          DecHour   : out STD_LOGIC;
          BlinkHour : out STD_LOGIC;
```

```
          IncMin    : out STD_LOGIC;
          DecMin    : out STD_LOGIC;
          BlinkMin  : out STD_LOGIC;
          IncSec    : out STD_LOGIC;
          DecSec    : out STD_LOGIC;
          BlinkSec  : out STD_LOGIC);
end control;

architecture Behavioral of control is

constant CLK_RATE : INTEGER := 100_000_000; -- frecventa semnalului Clk
constant CNT_1HZ  : INTEGER := CLK_RATE;    -- divizor pentru o secunda
type CTRL_TYPE is (run, set_hour, set_hour_inc, set_hour_dec, set_min,
                   set_min_inc, set_min_dec, set_sec, set_sec_inc,
                   set_sec_dec);
signal State : CTRL_TYPE;

begin
   ctrl: process (Clk)
   variable Count : INTEGER range 0 to CNT_1HZ - 1 := 0;
   begin
      if RISING_EDGE (Clk) then
         if (Rst = '1') then
            State <= run;
            Count := 0;
         else
            IncTime <= '0';
            case State is
               when run =>
                  if Count = CNT_1HZ - 1 then -- a trecut o secunda
                     IncTime <= '1';     -- incrementeaza secunda
                     Count := 0;         -- reseteaza numaratorul
                  else
                     Count := Count + 1;
                  end if;
                  if (ModeIn = '1') then
                     State <= set_hour;  -- altfel ramane in starea run
                  end if;
               when set_hour =>
                  if (UpIn = '1') then
                     State <= set_hour_inc;
                  elsif (DownIn = '1') then
                     State <= set_hour_dec;
                  elsif (ModeIn = '1') then
                     State <= set_min;   -- altfel ramane in starea set_hour
                  end if;
               when set_hour_inc =>
                  State <= set_hour;
               when set_hour_dec =>
                  State <= set_hour;
               when set_min =>
                  if (UpIn = '1') then
                     State <= set_min_inc;
                  elsif (DownIn = '1') then
                     State <= set_min_dec;
                  elsif (ModeIn = '1') then
                     State <= set_sec;   -- altfel ramane in starea set_min
                  end if;
               when set_min_inc =>
```

```
                State <= set_min;
             when set_min_dec =>
                State <= set_min;
             when set_sec =>
                Count := 0;
                if (UpIn = '1') then
                   State <= set_sec_inc;
                elsif (DownIn = '1') then
                   State <= set_sec_dec;
                elsif (ModeIn = '1') then
                   State <= run;       -- altfel ramane in starea set_sec
                end if;
             when set_sec_inc =>
                State <= set_sec;
             when set_sec_dec =>
                State <= set_sec;
          end case;
       end if;
    end if;
 end process ctrl;

 -- Asignarea semnalelor de comanda
 IncHour   <= '1' when State = set_hour_inc else '0';
 DecHour   <= '1' when State = set_hour_dec else '0';
 BlinkHour <= '1' when State = set_hour else '0';
 IncMin    <= '1' when State = set_min_inc else '0';
 DecMin    <= '1' when State = set_min_dec else '0';
 BlinkMin  <= '1' when State = set_min else '0';
 IncSec    <= '1' when State = set_sec_inc else '0';
 DecSec    <= '1' when State = set_sec_dec else '0';
 BlinkSec  <= '1' when State = set_sec else '0';

end Behavioral;
```

1.3.2.2 Modulul de actualizare a ceasului

Modulul de actualizare a ceasului time_cnt utilizează ca intrări semnalele de comandă generate de modulul de comandă pentru actualizarea ceasului în modul normal de funcționare și pentru setarea orei în celălalt mod. Ieșirile acestui modul sunt ora, minutul și secunda, fiecare reprezentată pe două cifre în cod binar-zecimal (BCD). Modulul conține trei procese separate pentru actualizarea secundei, a minutului și a orei. Sunt definite semnale interne pentru păstrarea valorii curente a secundei, minutului și orei (de exemplu, *SecHi_int* și *SecLo_int* pentru păstrarea cifrei mai semnificative și a celei mai puțin semnificative a secundei). Aceste semnale interne sunt asignate la semnalele de ieșire utilizând instrucțiuni concurente de asignare.

În procesul pentru actualizarea secundei, dacă semnalul *IncTime* este activat (în modul normal de funcționare) sau dacă semnalul *IncSec* este activat (în modul de setare a orei), cifra mai puțin semnificativă a secundei este incrementată, cu excepția cazului în care această cifră este 9, când ea devine 0. În acest caz, cifra mai semnificativă a secundei este incrementată, cu excepția cazului în care această cifră este 5, când ea devine 0. Astfel, dacă cele două cifre ale secundei sunt 59, acestea devin 00; în acest caz, dacă ceasul este în modul normal de funcționare, semnalul *TcSec* este

activat. Acest semnal este utilizat de procesul care actualizează minutul pentru a-l incrementa atunci când ceasul este în modul normal de funcționare. În același proces pentru actualizarea secundei, dacă semnalul *DecSec* este activat (în modul de setare a orei), cifra mai puțin semnificativă a secundei este decrementată, cu excepția cazului în care această cifră este 0, când ea devine 9. În acest caz, cifra mai semnificativă a secundei este decrementată, cu excepția cazului în care această cifră este 0, când ea devine 5. Astfel, dacă cele două cifre ale secundei sunt 00, acestea devin 59.

Procesul pentru actualizarea minutului este similar cu procesul pentru actualizarea secundei, exceptând faptul că în modul normal de funcționare incrementează minutul atunci când semnalul *TcSec* (și nu semnalul *IncTime*) este activat, iar în modul de setare a orei incrementează sau decrementează minutul atunci când semnalul *IncMin*, respectiv *DecMin* este activat. Dacă cele două cifre ale minutului sunt 59, acestea devin 00, și dacă ceasul este în modul normal de funcționare, semnalul *TcMin* este activat. Acest semnal este utilizat de procesul care actualizează ora pentru a o incrementa atunci când ceasul este în modul normal de funcționare.

Procesul pentru actualizarea orei incrementează ora dacă semnalul *TcMin* este activat (în modul normal de funcționare) sau dacă semnalul *IncHour* este activat (în modul de setare a orei). Dacă cele două cifre ale orei sunt 23, acestea devin 00. Dacă semnalul *DecHour* este activat (în modul de setare a orei), ora este decrementată, cu excepția cazului în care cele două cifre ale orei sunt 00, când ele devin 23.

Modulul de actualizare a ceasului este descris în Exemplul 1.2.

Exemplul 1.2

```
----------------------------------------------------------------------------
-- Nume proiect: clock
-- Nume modul:   time_cnt
-- Descriere:    Modul de actualizare a ceasului
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity time_cnt is
   Port ( Clk     : in  STD_LOGIC;
          Rst     : in  STD_LOGIC;
          IncTime : in  STD_LOGIC;
          IncHour : in  STD_LOGIC;
          DecHour : in  STD_LOGIC;
          IncMin  : in  STD_LOGIC;
          DecMin  : in  STD_LOGIC;
          IncSec  : in  STD_LOGIC;
          DecSec  : in  STD_LOGIC;
          HourHi  : out STD_LOGIC_VECTOR (3 downto 0);
          HourLo  : out STD_LOGIC_VECTOR (3 downto 0);
          MinHi   : out STD_LOGIC_VECTOR (3 downto 0);
          MinLo   : out STD_LOGIC_VECTOR (3 downto 0);
          SecHi   : out STD_LOGIC_VECTOR (3 downto 0);
          SecLo   : out STD_LOGIC_VECTOR (3 downto 0));
end time_cnt;
```

```
architecture Behavioral of time_cnt is

signal SecHi_int  : STD_LOGIC_VECTOR (3 downto 0) := x"0";
signal SecLo_int  : STD_LOGIC_VECTOR (3 downto 0) := x"0";
signal MinHi_int  : STD_LOGIC_VECTOR (3 downto 0) := x"0";
signal MinLo_int  : STD_LOGIC_VECTOR (3 downto 0) := x"0";
signal HourHi_int : STD_LOGIC_VECTOR (3 downto 0) := x"1";
signal HourLo_int : STD_LOGIC_VECTOR (3 downto 0) := x"2";
signal TcSec      : STD_LOGIC := '0';
signal TcMin      : STD_LOGIC := '0';

begin

   update_sec: process (Clk)      -- actualizarea secundei
   begin
      if RISING_EDGE (Clk) then
         if (Rst = '1') then
            SecHi_int <= x"0";
            SecLo_int <= x"0";
         else
            TcSec <= '0';
            if ((IncTime = '1') or (IncSec = '1')) then
               if (SecLo_int = x"9") then
                  SecLo_int <= x"0";
                  if (SecHi_int = x"5") then
                     SecHi_int <= x"0";
                     if (IncTime = '1') then
                        TcSec <= '1';
                     end if;
                  else
                     SecHi_int <= SecHi_int + 1;
                  end if;
               else
                  SecLo_int <= SecLo_int + 1;
               end if;
            end if;
            if (DecSec = '1') then
               if (SecLo_int = x"0") then
                  SecLo_int <= x"9";
                  if (SecHi_int = x"0") then
                     SecHi_int <= x"5";
                  else
                     SecHi_int <= SecHi_int - 1;
                  end if;
               else
                  SecLo_int <= SecLo_int - 1;
               end if;
            end if;
         end if;
      end if;
   end process update_sec;

   update_min: process (Clk)      -- actualizarea minutului
   begin
      if RISING_EDGE (Clk) then
         if (Rst = '1') then
            MinHi_int <= x"0";
            MinLo_int <= x"0";
```

```
      else
         TcMin <= '0';
         if ((TcSec = '1') or (IncMin = '1')) then
            if (MinLo_int = x"9") then
               MinLo_int <= x"0";
               if (MinHi_int = x"5") then
                  MinHi_int <= x"0";
                  if (TcSec = '1') then
                     TcMin <= '1';
                  end if;
               else
                  MinHi_int <= MinHi_int + 1;
               end if;
            else
               MinLo_int <= MinLo_int + 1;
            end if;
         end if;
         if (DecMin = '1') then
            if (MinLo_int = x"0") then
               MinLo_int <= x"9";
               if (MinHi_int = x"0") then
                  MinHi_int <= x"5";
               else
                  MinHi_int <= MinHi_int - 1;
               end if;
            else
               MinLo_int <= MinLo_int - 1;
            end if;
         end if;
      end if;
   end if;
end process update_min;

update_hour: process (Clk)      -- actualizarea orei
begin
   if RISING_EDGE (Clk) then
      if (Rst = '1') then
         HourHi_int <= x"1";
         HourLo_int <= x"2";
      else
         if ((TcMin = '1') or (IncHour = '1')) then
            if ((HourHi_int = x"2") and (HourLo_int = x"3")) then
               HourHi_int <= x"0";
               HourLo_int <= x"0";
            elsif (HourLo_int = x"9") then
               HourHi_int <= HourHi_int + 1;
               HourLo_int <= x"0";
            else
               HourLo_int <= HourLo_int + 1;
            end if;
         end if;
         if (DecHour = '1') then
            if ((HourHi_int = x"0") and (HourLo_int = x"0")) then
               HourHi_int <= x"2";
               HourLo_int <= x"3";
            elsif (HourLo_int = x"0") then
               HourHi_int <= HourHi_int - 1;
               HourLo_int <= x"9";
            else
```

```
                HourLo_int <= HourLo_int - 1;
            end if;
          end if;
        end if;
      end if;
   end process update_hour;

   -- Asignarea semnalelor interne la semnalele de iesire
   HourHi <= HourHi_int;
   HourLo <= HourLo_int;
   MinHi  <= MinHi_int;
   MinLo  <= MinLo_int;
   SecHi  <= SecHi_int;
   SecLo  <= SecLo_int;

end Behavioral;
```

1.3.2.3 Multiplexorul pentru afișajul cu șapte segmente

Într-un afișaj cu șapte segmente, fiecare cifră este formată din segmentele A, B, C, D, E, F, G și un punct zecimal opțional. Segmentele și punctul zecimal conțin câte o diodă electroluminiscentă (LED). Diodele segmentelor pot fi iluminate în mod individual prin aplicarea unei tensiuni între anodul și catodul corespunzător. Modulul multiplexor descris presupune un afișaj cu un număr de opt cifre și anod comun, în care anozii diodelor corespunzătoare fiecărei cifre sunt conectați împreună, iar catozii diodelor sunt separați. Semnalele anodului comun pentru fiecare cifră reprezintă opt intrări ale afișajului cu opt cifre, denumite AN0 .. AN7. Catozii segmentelor similare ale tuturor cifrelor sunt conectați împreună; de exemplu, cei opt catozi ai segmentelor A de la cele opt cifre sunt conectați la o singură intrare denumită CA. Există semnale de intrare similare pentru catozii celorlalte segmente (CB, CC, CD, CE, CF, CG) și pentru catodul punctului zecimal (DP). Această conexiune creează un afișaj multiplexat, în care o singură cifră este activă (iluminată) la un moment dat, cea pentru care semnalul anodului este activat. Semnalele catodului sunt comune pentru toate cifrele.

Pentru iluminarea unui segment al unei cifre, trebuie să se aplice un semnal cu nivel logic ridicat (1 logic) pe anodul segmentului și un semnal cu nivel logic scăzut (0 logic) pe catodul său. La unele plăci de dezvoltare, de exemplu, Nexys4 DDR sau Nexys A7, semnalele anodului comun sunt inversate. De aceea, pentru iluminarea unui segment al unei cifre trebuie să se aplice semnale cu valoarea logică 0 atât anodului cât și catodului segmentului.

Modulul multiplexor displ7seg_blink prezentat în continuare este destinat unui afișaj cu opt cifre și anod comun, la care trebuie să se aplice un semnal cu valoarea logică 0 pe anodul comun al cifrei active. Sunt necesare modificări ale multiplexorului dacă afișajul are un număr diferit de opt cifre, dacă este cu catod comun, sau dacă pentru activarea unei cifre trebuie să se aplice un semnal cu valoarea logică 1 pe anodul comun al cifrei respective.

Modulul multiplexor pentru afișajul cu șapte segmente generează semnalele de comandă pentru anozi și semnalele corespunzătoare pentru catozii fiecărei cifre în

mod repetat, cu o anumită rată de reîmprospătare. Pentru un afișaj cu opt cifre, fiecare cifră va fi iluminată doar pentru a opta parte din timp. Din cauza inerției ochiului uman, acesta nu va percepe stingerea unei cifre înainte ca aceasta să fie iluminată din nou dacă rata de reîmprospătare este în jur de 60 Hz sau mai mare. Dacă însă rata de reîmprospătare este mai mică de 60 Hz, poate fi percepută pâlpâirea afișajului. Pentru ca fiecare din cele opt cifre ale afișajului să apară iluminată continuu, fiecare cifră trebuie selectată cel puțin o dată la fiecare 16 ms, ceea ce corespunde unei rate de reîmprospătare de 62,5 Hz. Pentru un interval de reîmprospătare de 16 ms, fiecare cifră va fi iluminată pentru o perioadă de 2 ms. Modulul multiplexor trebuie să aplice la catozii unei anumite cifre semnalele corecte atunci când cifra respectivă este selectată cu semnalul aplicat anodului său.

Modulul multiplexor conține un proces care implementează un numărător necesar pentru a asigura o anumită rată de reîmprospătare pentru afișaj. Se utilizează trei biți ai acestui numărător pentru a selecta, la un moment dat, o cifră din cele opt cifre ale afișajului și pentru a selecta semnalul aplicat catodului cifrei selectate. De aceea, numărătorul ar trebui declarat ca un vector cu o dimensiune care depinde de valoarea maximă a numărătorului. Presupunând un semnal de ceas cu frecvența de 100 MHz, pentru o rată de reîmprospătare de 100 Hz numărătorul trebuie să numere până la 1.000.000 – 1. Pentru determinarea dimensiunii vectorului, considerăm valoarea cea mai apropiată de această valoare maximă care este și o putere a lui 2. Cea mai apropiată valoare este 1.048.576 = 2^{20}, astfel că dimensiunea vectorului trebuie să fie 20. Prin alegerea unui vector de 20 de biți, întregul afișaj va fi reîmprospătat în 2^{20} cicluri de ceas. Cu o frecvență a semnalului de ceas de 100 MHz (cu o perioadă de 10 ns), intervalul de reîmprospătare al întregului afișaj va fi de 10,48576 ms, iar fiecare cifră va fi activă pentru un timp de 10,48576 ms / 8 = 1,31072 ms. Rata de reîmprospătare efectivă a afișajului va fi de 1 / 10,48576 ms ≅ 95,36 Hz.

În descrierea VHDL a modulului multiplexor, prezentată în continuare, numărătorul (*Count*) este declarat ca un semnal de tip întreg și se utilizează funcția **CONV_STD_LOGIC_VECTOR** pentru conversia acestuia într-un vector (*CountVect*):

```
CountVect <= CONV_STD_LOGIC_VECTOR (Count, 20);
```

Al doilea parametru al funcției reprezintă numărul de biți ai vectorului la care va fi convertit semnalul numărătorului. Această funcție de conversie este declarată în pachetul **STD_LOGIC_ARITH** din biblioteca IEEE. Cei trei biți mai semnificativi ai vectorului *CountVect* sunt asignați semnalului *LedSel*, care va fi utilizat pentru selecția cifrei active a afișajului și a semnalului care fi aplicat la catozii cifrei active.

Intrarea principală a modulului multiplexor este vectorul *Data* de 64 biți, care conține opt octeți cu configurația segmentelor (catozilor) pentru fiecare cifră. Octetul cel mai semnificativ (biții 63..56) trebuie să conțină configurația pentru cifra din stânga, iar octetul cel mai puțin semnificativ (biții 7..0) trebuie să conțină configurația pentru cifra din dreapta. Modulul nu decodifică vectorul de intrare, ci aplică octeții individuali ai acestui vector direct la segmentele (catozii) cifrelor. Aceasta permite

afișarea pe o cifră a oricărei configurații a segmentelor și nu doar a cifrelor hexazecimale. Modulul multiplexor nu controlează punctele zecimale ale cifrelor, astfel încât acestea vor fi stinse în permanență. În schimb, modulul permite afișarea cu pâlpâire a informației pe oricare cifră dacă bitul cel mai semnificativ al octetului corespunzător acelei cifre este 1. De exemplu, dacă biții 63 și 55 ai vectorului *Data* sunt setați, pe cele două cifre din stânga informația va fi afișată intermitent, producând efectul de pâlpâire. Această facilitate este utilizată de ceasul de timp real pentru pâlpâirea cifrelor orei, minutului sau secundei în modul de setare a orei. Pentru implementarea funcției de pâlpâire se utilizează un proces separat. Acest proces utilizează un numărător pentru comutarea stării semnalului *BlinkOn* la un interval de 0,5 s. Atunci când acest semnal este activat și pâlpâirea unei cifre este validată, cifra respectivă este stinsă.

Multiplexorul pentru afișajul cu șapte segmente este descris în Exemplul 1.3.

Exemplul 1.3

```
--------------------------------------------------------------------------------
-- Nume proiect: clock
-- Nume modul:   displ7seg_blink
-- Descriere:    Multiplexor pentru afisajul cu sapte segmente, cu
--               posibilitatea palpairii cifrelor. Datele de intrare nu
--               sunt decodificate, ci sunt aplicate direct la segmentele
--               afisajului. Pentru afisarea valorilor hexazecimale,
--               codul fiecarei cifre trebuie aplicat la intrarea Data
--               prin intermediul functiei de conversie HEX2SSEG. Pentru
--               afisarea unei cifre cu palpaire, bitul 7 al codului cifrei
--               trebuie setat la 1.
--               Codificarea segmentelor (biti 7..0): 0GFE DCBA
--                   A
--                  ---
--               F |   | B
--                  ---    <- G
--               E |   | C
--                  ---
--                   D
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity displ7seg_blink is
   Port ( Clk  : in  STD_LOGIC;
          Rst  : in  STD_LOGIC;
          Data : in  STD_LOGIC_VECTOR (63 downto 0);
                 -- date de afisat (cifra 1 din stanga: biti 63..56)
          An   : out STD_LOGIC_VECTOR (7 downto 0);
                 -- semnale pentru anozi (active in 0 logic)
          Seg  : out STD_LOGIC_VECTOR (7 downto 0));
                 -- semnale pentru segmentele (catozii) cifrei active
end displ7seg_blink;

architecture Behavioral of displ7seg_blink is
```

```
constant CLK_RATE  : INTEGER := 100_000_000;  -- frecventa semnalului Clk
constant CNT_100HZ : INTEGER := 2**20;        -- divizor pentru rata de
                                              -- reimprospatare de ~100 Hz
constant CNT_500MS : INTEGER := CLK_RATE / 2; -- divizor pentru 500 ms
signal Count       : INTEGER range 0 to CNT_100HZ - 1 := 0;
signal CountBlink  : INTEGER range 0 to CNT_500MS - 1 := 0;
signal BlinkOn     : STD_LOGIC := '0';
signal CountVect   : STD_LOGIC_VECTOR (19 downto 0) := (others => '0');
signal LedSel      : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
signal Digit1      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit2      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit3      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit4      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit5      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit6      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit7      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit8      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');

begin
   -- Proces pentru divizarea frecventei ceasului
   div_clk: process (Clk)
   begin
      if RISING_EDGE (Clk) then
         if (Rst = '1') then
            Count <= 0;
         elsif (Count = CNT_100HZ - 1) then
            Count <= 0;
         else
            Count <= Count + 1;
         end if;
      end if;
   end process div_clk;

   CountVect <= CONV_STD_LOGIC_VECTOR (Count, 20);
   LedSel <= CountVect (19 downto 17);

   -- Proces pentru functia de palpaire
   blink: process (Clk)
   begin
      if RISING_EDGE (Clk) then
         if (Rst = '1') then
            CountBlink <= 0;
            BlinkOn <= '0';
         elsif (CountBlink = CNT_500MS - 1) then
            CountBlink <= 0;
            BlinkOn <= not BlinkOn;
         else
            CountBlink <= CountBlink + 1;
         end if;
      end if;
   end process blink;

   -- Date pentru segmentele fiecarei cifre
   Digit8 <= x"FF" when (BlinkOn = '1') and (Data(7)  = '1') else
             '1' & Data (6 downto 0);
   Digit7 <= x"FF" when (BlinkOn = '1') and (Data(15) = '1') else
             '1' & Data (14 downto 8);
   Digit6 <= x"FF" when (BlinkOn = '1') and (Data(23) = '1') else
             '1' & Data (22 downto 16);
```

```
    Digit5 <= x"FF" when (BlinkOn = '1') and (Data(31) = '1') else
              '1' & Data (30 downto 24);
    Digit4 <= x"FF" when (BlinkOn = '1') and (Data(39) = '1') else
              '1' & Data (38 downto 32);
    Digit3 <= x"FF" when (BlinkOn = '1') and (Data(47) = '1') else
              '1' & Data (46 downto 40);
    Digit2 <= x"FF" when (BlinkOn = '1') and (Data(55) = '1') else
              '1' & Data (54 downto 48);
    Digit1 <= x"FF" when (BlinkOn = '1') and (Data(63) = '1') else
              '1' & Data (62 downto 56);

    -- Semnal pentru selectarea cifrei active (anozi)
    An <= "11111110" when LedSel = "000" else
          "11111101" when LedSel = "001" else
          "11111011" when LedSel = "010" else
          "11110111" when LedSel = "011" else
          "11101111" when LedSel = "100" else
          "11011111" when LedSel = "101" else
          "10111111" when LedSel = "110" else
          "01111111" when LedSel = "111" else
          "11111111";

    -- Semnal pentru segmentele cifrei active (catozi)
    Seg <= Digit8 when LedSel = "000" else
           Digit7 when LedSel = "001" else
           Digit6 when LedSel = "010" else
           Digit5 when LedSel = "011" else
           Digit4 when LedSel = "100" else
           Digit3 when LedSel = "101" else
           Digit2 when LedSel = "110" else
           Digit1 when LedSel = "111" else
           x"FF";

end Behavioral;
```

1.3.2.4 Modulul principal

Modulul principal clock conține instanțierea modulului de comandă, a modulului de actualizare a ceasului și a multiplexorului pentru afișaj. Modulul principal conține și funcția `HEX2SSEG`, care convertește codul de patru biți al unei cifre hexazecimale în codul de opt biți care trebuie aplicat la catozii afișajului. Bitul 7 al acestui cod corespunde punctului zecimal; acest bit este setat întotdeauna în codul generat de funcția `HEX2SSEG`, astfel încât punctul zecimal va fi stins întotdeauna. Biții 0 până la 6 corespund segmentelor A până la G, după cum rezultă din comentariile descrierii în limbajul VHDL a modulului principal din Exemplul 1.4.

Modulul clock setează octeții individuali (*Digit1* până la *Digit8*) reprezentând configurațiile segmentelor pentru cele opt cifre. Pentru cifrele orei, minutului și secundei se utilizează funcția `HEX2SSEG` pentru a obține codul care trebuie aplicat la segmentele afișajului. Bitul 7 al fiecărui octet de cod este resetat și apoi este setat la 1 dacă semnalul corespunzător de pâlpâire (*BlinkHour*, *BlinkMin* sau *BlinkSec*) este activat. Pentru cele două cifre separatoare "-", octetul de cod este definit direct pentru a aprinde segmentul G. Bitul 7 al acestor doi octeți de cod este resetat la 0 dacă oricare

semnal de pâlpâire este activat și este setat la 1 în caz contrar. Astfel, cele două cifre separatoare nu vor pâlpâi dacă ceasul este în modul de setare a orei și vor pâlpâi dacă ceasul este în modul normal de funcționare. Cei opt octeți de cod sunt apoi concatenați într-un vector de 64 de biți care este aplicat la intrarea *Data* a multiplexorului pentru afișajul cu șapte segmente.

Descrierea modulului principal este prezentată în Exemplul 1.4.

Observație

Descrierea modulului principal trebuie completată parcurgând etapele descrise în secțiunea 1.3.4 pentru instanțierea modulului de filtrare a oscilațiilor butoanelor și conectarea sa la celelalte module.

Exemplul 1.4

```
----------------------------------------------------------------------------
-- Nume proiect: clock
-- Nume modul:   clock
-- Descriere:    Modul principal pentru ceasul de timp real
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clock is
   Port ( Clk  : in  STD_LOGIC;
          Rst  : in  STD_LOGIC;
          Mode : in  STD_LOGIC;
          Up   : in  STD_LOGIC;
          Down : in  STD_LOGIC;
          An   : out STD_LOGIC_VECTOR (7 downto 0);
          Seg  : out STD_LOGIC_VECTOR (7 downto 0));
end clock;

architecture Behavioral of clock is

-- Functie de conversie pentru un afisaj cu sapte segmente cu anod comun
-- Intrare: Cifra hexazecimala
-- Iesire:  Codul pentru segmente
-- Codificarea segmentelor (biti 7..0): HGFE DCBA (H - punct zecimal)
--      A
--     ---
--  F |   | B
--     ---    <- G
--  E |   | C
--     ---  . <- H
--      D

function HEX2SSEG (Hex : in STD_LOGIC_VECTOR (3 downto 0))
                  return STD_LOGIC_VECTOR is
   variable Sseg : STD_LOGIC_VECTOR (7 downto 0);
begin
   case Hex is
      when "0000" => Sseg := "11000000";  -- 0
```

```
      when "0001" => Sseg := "11111001";  -- 1
      when "0010" => Sseg := "10100100";  -- 2
      when "0011" => Sseg := "10110000";  -- 3
      when "0100" => Sseg := "10011001";  -- 4
      when "0101" => Sseg := "10010010";  -- 5
      when "0110" => Sseg := "10000010";  -- 6
      when "0111" => Sseg := "11111000";  -- 7
      when "1000" => Sseg := "10000000";  -- 8
      when "1001" => Sseg := "10010000";  -- 9
      when "1010" => Sseg := "10001000";  -- A
      when "1011" => Sseg := "10000011";  -- b
      when "1100" => Sseg := "11000110";  -- C
      when "1101" => Sseg := "10100001";  -- d
      when "1110" => Sseg := "10000110";  -- E
      when "1111" => Sseg := "10001110";  -- F
      when others => Sseg := "11111111";
   end case;
   return Sseg;
end function HEX2SSEG;

signal Data      : STD_LOGIC_VECTOR (63 downto 0) := (others => '0');
signal Digit1    : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit2    : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit3    : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit4    : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit5    : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit6    : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit7    : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit8    : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');

signal IncTime   : STD_LOGIC := '0';
signal IncHour   : STD_LOGIC := '0';
signal DecHour   : STD_LOGIC := '0';
signal BlinkHour : STD_LOGIC := '0';
signal IncMin    : STD_LOGIC := '0';
signal DecMin    : STD_LOGIC := '0';
signal BlinkMin  : STD_LOGIC := '0';
signal IncSec    : STD_LOGIC := '0';
signal DecSec    : STD_LOGIC := '0';
signal BlinkSec  : STD_LOGIC := '0';
signal HourHi    : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
signal HourLo    : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
signal MinHi     : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
signal MinLo     : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
signal SecHi     : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
signal SecLo     : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');

begin
   -- Setarea configuratiei segmentelor pentru fiecare cifra
   -- Cifrele 1-2: ora
   Digit1 <= (HEX2SSEG (HourHi) and x"7F") or (BlinkHour & "0000000");
   Digit2 <= (HEX2SSEG (HourLo) and x"7F") or (BlinkHour & "0000000");
   -- Cifra 3: separator; fara palpaire in modul de setare a orei,
   --          altfel cu palpaire
   Digit3 <= b"0011_1111" when (BlinkHour or BlinkMin or BlinkSec) = '1'
                          else b"1011_1111";
   -- Cifrele 4-5: minutul
   Digit4 <= (HEX2SSEG (MinHi) and x"7F") or (BlinkMin & "0000000");
   Digit5 <= (HEX2SSEG (MinLo) and x"7F") or (BlinkMin & "0000000");
```

```
    -- Cifra 6: separator; fara palpaire in modul de setare a orei,
    --          altfel cu palpaire
    Digit6 <= b"0011_1111" when (BlinkHour or BlinkMin or BlinkSec) = '1'
                           else b"1011_1111";
    -- Cifrele 7-8: secunda
    Digit7 <= (HEX2SSEG (SecHi) and x"7F") or (BlinkSec & "0000000");
    Digit8 <= (HEX2SSEG (SecLo) and x"7F") or (BlinkSec & "0000000");

    Data   <= Digit1&Digit2&Digit3&Digit4&Digit5&Digit6&Digit7&Digit8;

    control_i: entity WORK.control port map (
                       Clk => Clk,
                       Rst => Rst,
                       ModeIn => Mode,
                       UpIn => Up,
                       DownIn => Down,
                       IncTime => IncTime,
                       IncHour => IncHour,
                       DecHour => DecHour,
                       BlinkHour => BlinkHour,
                       IncMin => IncMin,
                       DecMin => DecMin,
                       BlinkMin => BlinkMin,
                       IncSec => IncSec,
                       DecSec => DecSec,
                       BlinkSec => BlinkSec);

    time_cnt_i: entity WORK.time_cnt port map (
                       Clk => Clk,
                       Rst => Rst,
                       IncTime => IncTime,
                       IncHour => IncHour,
                       DecHour => DecHour,
                       IncMin => IncMin,
                       DecMin => DecMin,
                       IncSec => IncSec,
                       DecSec => DecSec,
                       HourHi => HourHi,
                       HourLo => HourLo,
                       MinHi  => MinHi,
                       MinLo  => MinLo,
                       SecHi  => SecHi,
                       SecLo  => SecLo);

    displ7seg_blink_i: entity WORK.displ7seg_blink port map (
                       Clk => Clk,
                       Rst => Rst,
                       Data => Data,
                       An => An,
                       Seg => Seg);

end Behavioral;
```

1.3.3 Crearea unui nou proiect

Lansați în execuție aplicația *Vivado*. În continuare sunt prezentate operațiile necesare pentru a crea un nou proiect.

Observații

- Etapele descrise în continuare pentru exemplificarea utilizării mediului de proiectare Xilinx Vivado presupun utilizarea versiunii 2010.1 a acestui mediu. Pot fi necesare modificări ale acestor etape dacă se utilizează o versiune diferită.
- Etapele care se referă la o anumită placă de dezvoltare presupun utilizarea plăcii Nexys4 DDR sau Nexys A7-100T (care sunt identice funcțional). Va fi necesară specificarea unui alt circuit FPGA și a unui alt fișier de constrângeri dacă se utilizează o altă placă de dezvoltare.

1. În fereastra aplicației *Vivado*, secțiunea *Quick Start*, selectați **Create Project**. Se afișează fereastra de dialog *Create a New Vivado Project*.
2. Selectați butonul **Next** în fereastra *Create a New Vivado Project*. Se afișează fereastra de dialog *Project Name*.
3. În câmpul *Project name* introduceți numele proiectului (de exemplu, **clock**).
4. În câmpul *Project location* selectați butonul  pentru a naviga la directorul în care trebuie creat proiectul. Selectați directorul dorit, iar apoi butonul **Select**. Verificați ca opțiunea **Create project subdirectory** să fie bifată și apoi selectați butonul **Next**. Se afișează fereastra de dialog *Project Type*.
5. Selectați tipul proiectului **RTL Project** dacă nu este selectat, bifați opțiunea **Do not specify sources at this time**, iar apoi selectați butonul **Next**. Se afișează fereastra de dialog *Default Part*.
6. Selectați panoul **Boards**, iar în lista afișată selectați linia corespunzătoare plăcii de dezvoltare (de exemplu, **Nexys4 DDR**, **Nexys A7-100T** sau **Nexys A7-50T**), evitând însă selecția legăturii de culoare albastră. Prin aceasta, implementarea se va realiza pentru circuitul FPGA al plăcii selectate. Dacă placa dorită nu apare în lista afișată, reveniți în panoul **Parts**, iar în coloana *Part* selectați circuitul care se află pe placa de dezvoltare. Pentru plăcile Nexys4 DDR și Nexys A7-100T, circuitul FPGA este xc7a100tcsg324-1 din familia Artix-7. Selectați apoi butonul **Next**.
7. În fereastra *New Project Summary*, revizuiți setările cu care se va crea noul proiect și selectați butonul **Finish** pentru crearea proiectului.

1.3.4 Descrierea proiectului

În această etapă a procesului de proiectare, se vor adăuga la proiect fișierele sursă ale exemplului de proiectare, se va crea un nou modul VHDL și acesta se va conecta la celelalte module.

1.3.4.1 Adăugarea fișierelor sursă și de constrângeri

Executați operațiile următoare pentru a crea fișierele sursă ale exemplului de proiectare și pentru adăugarea lor și a fișierului de constrângeri la proiect.

1. Într-un director de lucru, creați fișiere text cu extensia .vhd pentru modulele ceasului de timp real (control, time_cnt, displ7seg_blink) și pentru modulul principal (clock) pe baza codului sursă prezentat în secțiunea 1.3.2.
2. Descărcați fișierul de constrângeri (cu extensia .xdc) al plăcii de dezvoltare și copiați acest fișier în același director ca și fișierele .vhd create anterior. Pentru placa Nexys4 DDR, fișierul este Nexys4DDR_Master.xdc, iar pentru placa Nexys A7-100T, fișierul este Nexys-A7-100T-Master.xdc.
3. În panoul *Flow Navigator* din partea stângă a ferestrei *Vivado* selectați opțiunea **Add Sources** sau selectați iconița **Add Sources**  din panoul *Sources*. Se afișează fereastra de dialog *Add Sources*.
4. Selectați opțiunea **Add or create design sources** și selectați butonul **Next**. Se afișează fereastra de dialog *Add or Create Design Sources*.
5. Selectați butonul **Add Files**, navigați la directorul de lucru, selectați toate fișierele cu extensia .vhd create anterior și apoi selectați butonul **OK**.
6. În aceeași fereastră de dialog *Add or Create Design Sources*, verificați ca opțiunea **Copy sources into project** să fie bifată și selectați butonul **Finish**. Fișierele selectate vor fi adăugate în proiect și se va actualiza ierarhia fișierelor sursă.
7. Selectați din nou opțiunea **Add Sources** din panoul *Flow Navigator* pentru a adăuga fișierul de constrângeri la proiect. În fereastra de dialog *Add Sources*, selectați opțiunea **Add or create constraints** și selectați butonul **Next**. Se afișează fereastra de dialog *Add or Create Constraints*.
8. Selectați butonul **Add Files**, selectați fișierul cu extensia .xdc din același director de lucru și apoi selectați butonul **OK**.
9. În aceeași fereastră de dialog *Add or Create Constraints*, verificați ca opțiunea **Copy constraints files into project** să fie bifată, iar apoi selectați butonul **Finish**. Fișierul selectat va fi adăugat în proiect și se va actualiza ierarhia fișierelor de constrângeri.
10. În panoul *Sources*, expandați ierarhia fișierelor sursă și a fișierelor de constrângeri din secțiunile *Design Sources*, respectiv *Constraints*, pentru a vizualiza fișierele care au fost adăugate în proiect. În acest panou se afișează și unitățile de proiectare care sunt adăugate în acest moment în proiect, împreună cu numele entității asociate. După cum se poate observa în figura 1.12, fiecare unitate de

proiectare este reprezentată sub forma *nume_instanțiere* : *nume_entitate* (*nume_arhitectură*) (*nume_fișier*).

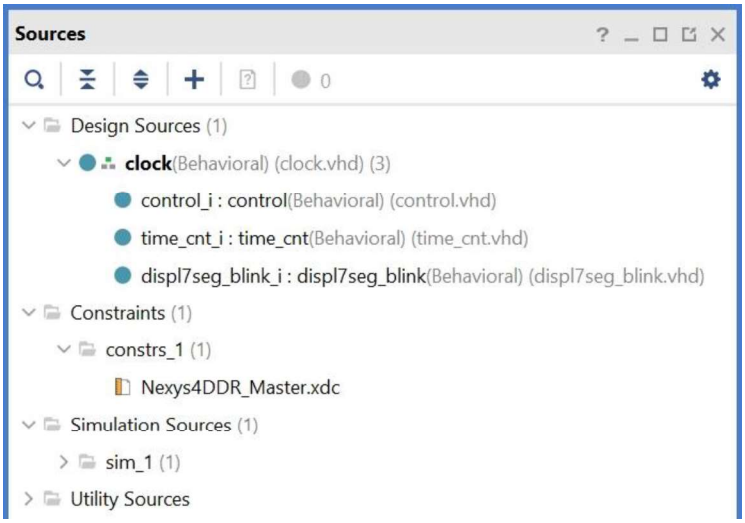

Figura 1.12. Ierarhia proiectului după adăugarea fișierelor sursă și a fișierului de constrângeri.

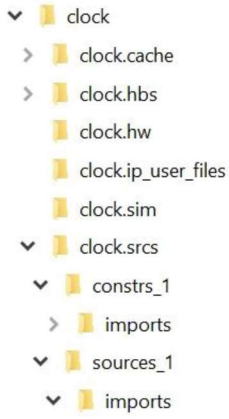

Figura 1.13. Structura de directoare generată la crearea proiectului și adăugarea fișierelor.

11. Folosiți utilitarul *File Explorer* pentru a vizualiza conținutul directorului în care a fost creat proiectul. Se poate observa că în acest director au fost create mai multe subdirectoare, printre care subdirectorul clock.srcs. Fișierele sursă adăugate în proiect pot fi regăsite în subdirectorul clock.srcs\sources_1\imports\clock\, iar fișierul de constrângeri adăugat poate fi regăsit în subdirectorul clock.srcs\constrs_1\imports\clock\, presupunând că numele directorului de lucru este clock (figura 1.13).

1.3.4.2 Crearea unui nou modul VHDL

În proiectul inițial al ceasului de timp real, semnalele de intrare *Mode*, *Up* și *Down* sunt conectate direct de la butoanele de pe placa de dezvoltare la modulul de comandă control. Pentru eliminarea oscilațiilor semnalelor de intrare *Mode*, *Up* și *Down* generate de butoanele plăcii, se va crea un nou modul VHDL. Pentru a crea acest modul, parcurgeți etapele următoare.

1. Selectați opțiunea **Add Sources** din panoul *Flow Navigator*. În fereastra de dialog *Add Sources*, selectați opțiunea **Add or create design sources** și selectați butonul **Next**. Se afișează fereastra de dialog *Add or Create Design Sources*.
2. Selectați butonul **Create File**; se va afișa caseta de dialog *Create Source File*. În câmpul *File type* selectați opțiunea **VHDL** dacă nu este selectată deja, în câmpul *File name* introduceți numele fișierului (de exemplu, debounce), iar apoi selectați butonul **OK**.
3. În fereastra de dialog *Add or Create Design Sources* selectați butonul **Finish**. Se va afișa fereastra de dialog *Define Module*.
4. În câmpul de sub linia *Port Name* introduceți numele portului de intrare **Clk**. Introduceți în mod similar numele porturilor **Rst**, **Din** și **Qout**. Modificați direcția portului *Qout* la **out**, după care selectați butonul **OK**. Se va crea fișierul sursă debounce.vhd, care conține declarația de entitate și de arhitectură a modulului debounce.
5. Deschideți fișierul sursă debounce.vhd în fereastra de editare executând un clic dublu pe numele acestui fișier în panoul *Hierarchy* din fereastra *Sources*.

1.3.4.3 Adăugarea unui șablon al limbajului

Editorul de texte al mediului de proiectare Xilinx Vivado pune la dispoziție construcții ale limbajelor VHDL și Verilog pentru simulare și sinteză, reprezentând componente digitale utilizate pe scară largă. În acest exemplu de proiectare se va utiliza construcția pentru sinteză *Debounce circuit*. Executați următoarele operații pentru a selecta șablonul necesar și pentru a adăuga acest șablon în fișierul sursă al modulului debounce.

1. Selectați iconița *Language Templates*  din meniul ferestrei de editare. Se va afișa fereastra *Language Templates*.
2. Expandați intrările *VHDL*, *Synthesis Constructs*, *Coding Examples* și *Misc* din fereastra *Language Templates*, iar apoi selectați șablonul numit *Debounce circuit*. Conținutul șablonului este afișat în panoul *Preview* al ferestrei *Language Templates*.

3. Selectați întregul text afișat în panoul *Preview*, executați un clic dreapta pe textul selectat și selectați opțiunea **Copy**.
4. Închideți fereastra *Language Templates*.
5. În fereastra de editare a fișierului debounce.vhd, inserați textul copiat anterior sub linia `begin`.
6. În fișierul debounce.vhd mutați linia cu definițiile semnalelor *Q1*, *Q2*, *Q3* între cuvintele cheie `architecture` și `begin`.
7. Înlocuiți `<clock>` (inclusiv parantezele <>) cu **Clk** în lista de sensibilitate a procesului și în instrucțiunea `if`. Înlocuiți `<reset>` cu **Rst** în instrucțiunea `if`. Înlocuiți `D_IN` cu **Din** și `Q_OUT` cu **Qout**.
8. Salvați fișierul sursă selectând iconița *Save File* din meniul ferestrei de editare. Fișierul va fi salvat în subdirectorul clock.srcs\sources_1\new\.

1.3.4.4 Instanțierea modulului VHDL

În continuare, modulul VHDL debounce creat anterior va fi instanțiat în modulul de nivel superior și va fi conectat la modulele existente ale proiectului. Pentru instanțierea și conectarea modulului debounce, executați operațiile următoare.

1. În panoul *Hierarchy* al ferestrei *Sources* executați un clic dublu pe linia în care apare numele fișierului clock.vhd pentru a deschide fișierul în fereastra de editare.
2. În fișierul clock.vhd declarați trei noi semnale de tip `STD_LOGIC` după ultima declarație `signal` și denumiți aceste semnale **ModeD**, **UpD** și **DownD**. Inițializați fiecare din aceste semnale cu '0'. Aceste semnale vor reprezenta ieșirile modulelor pentru eliminarea oscilațiilor care vor fi instanțiate în modulul de nivel superior.
3. Instanțiați entitatea debounce de trei ori la sfârșitul fișierului (înainte de linia `end Behavioral`) și conectați porturile în modul ilustrat în figura 1.14.
4. În același fișier clock.vhd, în declarația de instanțiere a entității control, modificați semnalele *Mode*, *Up* și *Down* (care apar după indicatorul =>) în **ModeD**, **UpD**, respectiv **DownD** (nu modificați numele porturilor *ModeIn*, *UpIn* și *DownIn* care apar la stânga indicatorului =>). Prin aceste modificări, vor fi eliminate oscilațiile semnalelor de intrare înaintea conectării lor la modulul control.
5. Salvați fișierul clock.vhd selectând iconița *Save File*. În panoul *Hierarchy* al ferestrei *Sources* se poate observa că instanțierile modulului debounce sunt adăugate în ierarhia proiectului (figura 1.15).

```
    mode_i: entity WORK.debounce port map (
                        Clk => Clk,
                        Rst => Rst,
                        Din => Mode,
                        Qout => ModeD);

    up_i: entity WORK.debounce port map (
                        Clk => Clk,
                        Rst => Rst,
                        Din => Up,
                        Qout => UpD);

    down_i: entity WORK.debounce port map (
                        Clk => Clk,
                        Rst => Rst,
                        Din => Down,
                        Qout => DownD);
end Behavioral;
```

Figura 1.14. Instanțierea entității debounce.

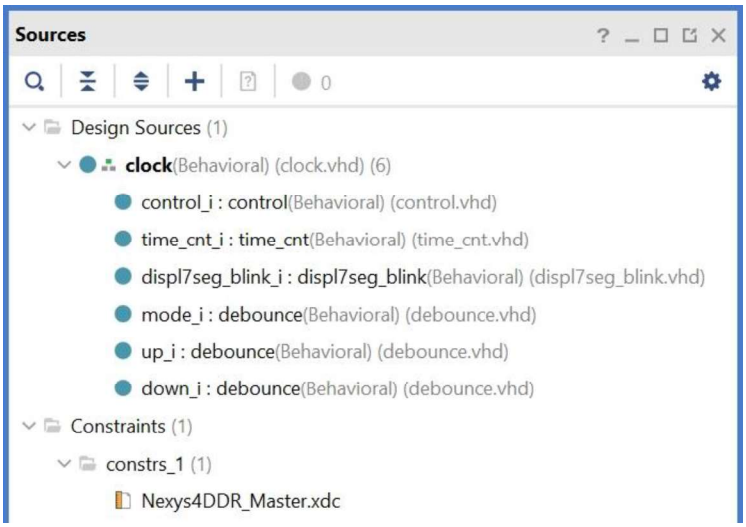

Figura 1.15. Ierarhia proiectului după adăugarea și instanțierea modulului debounce.

1.3.5 Elaborarea proiectului

În etapa de elaborare a proiectului, mediul de proiectare Vivado compilează fișierele sursă ale proiectului, execută optimizări de nivel înalt, generează lista de conexiuni a proiectului la nivelul RTL (*Register Transfer Level*), construiește structurile de date necesare etapelor următoare de proiectare și, opțional, aplică unele constrângeri de proiectare. Elaborarea se realizează în mod automat în timpul sintezei proiectului, dar există posibilitatea invocării explicite a acestei etape pentru compilarea și

verificarea sintactică a fișierelor sursă, explorarea listei de conexiuni și a schemei la nivelul RTL, verificarea unor reguli de proiectare și specificarea unor constrângeri.

Executați operațiile următoare pentru compilarea fișierelor sursă, vizualizarea schemei RTL și specificarea constrângerilor de plasare a porturilor de I/E.

1. În panoul *Flow Navigator*, selectați opțiunea **Open Elaborated Design** din secțiunea RTL ANALYSIS.
2. Dacă se afișează caseta de dialog *Elaborate Design*, selectați butonul **OK**. După terminarea elaborării proiectului, dacă nu sunt erori sintactice în fișierele sursă, se va afișa schema bloc a modulului de nivel superior al proiectului. Schema se poate afișa și dacă se selectează opțiunea *Tools* → *Schematic* din meniul principal sau se apasă tasta **F4**.
3. În fereastra *Schematic*, selectați iconița *Zoom In* pentru a putea distinge detaliile schemei. Se pot observa modulele care au fost instanțiate în modulul de nivel superior clock și unele componente suplimentare care au fost adăugate. Executați un clic dublu pe unul din modulele debounce pentru afișarea schemei interne a acestui modul.
4. Selectați iconița *Previous* pentru a reveni la schema modulului de nivel superior. Selectați unul din module (de culoare albastră), executați un clic cu butonul din dreapta în interiorul modulului și selectați opțiunea **Go to Source** (sau apăsați tasta **F7**) pentru a deschide fișierul sursă în care este instanțiat modulul respectiv. Reveniți apoi în fereastra *Schematic*, selectați unul din modulele RTL_ROM (de culoare galbenă) și apăsați tasta **F7** pentru a deschide fișierul sursă din care s-a generat acest modul.
5. Selectați panoul *Messages* din partea de jos a ecranului și parcurgeți mesajele generate în urma analizei fișierelor sursă și a elaborării proiectului. În acest panou se afișează și eventualele mesaje de eroare în urma compilării fișierelor sursă. Se poate selecta oricare mesaj de avertisment sau de eroare pentru a deschide fișierul sursă corespunzător în fereastra de editare.
6. Închideți fereastra *Schematic*.
7. În panoul *Flow Navigator* din secțiunea RTL ANALYSIS, selectați opțiunea **Report Noise**.
8. În caseta de dialog *Report Noise*, selectați butonul **OK** pentru generarea raportului ssn_1. Raportul va fi afișat în partea de jos a ecranului, secțiunea *I/O Bank Details* a raportului fiind selectată în mod automat.
9. În secțiunea *I/O Bank Details*, textul Unplaced Ports este afișat cu culoarea roșie (figura 1.16), motivul fiind faptul că nu s-a specificat asignarea porturilor la pinii circuitului. Se observă că sunt raportate doar porturile de ieșire *An* și *Seg*,

deoarece analiza se realizează doar pentru porturile de ieșire. Cu toate acestea, este necesară și specificarea asignării porturilor de intrare la pinii circuitului.

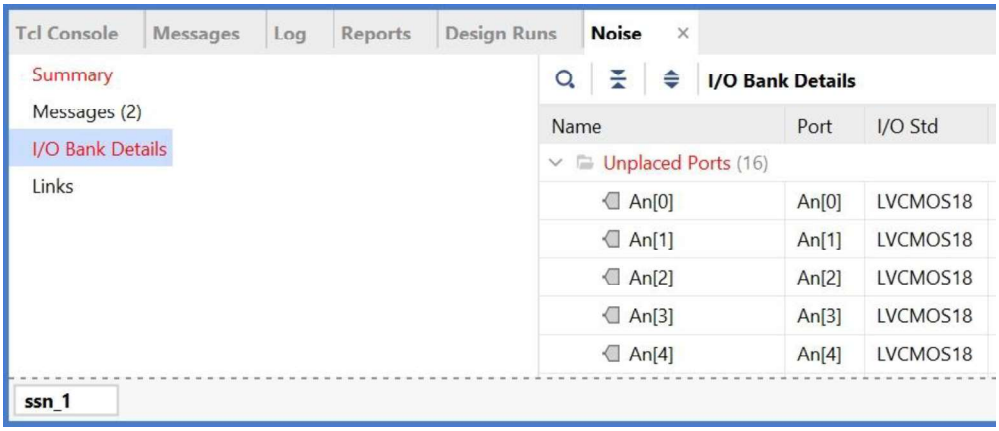

Figura 1.16. Panoul *Noise*.

10. În meniul principal Vivado, selectați opțiunea *Window* → *Sources*, expandați secțiunea *Constraints* din fereastra *Sources* și deschideți pentru editare fișierul de constrângeri (cu extensia .xdc).

11. În fișierul deschis se observă că toate liniile sunt comentate (fiecare linie începe cu caracterul #), acesta fiind un fișier general de constrângeri pentru placa de dezvoltare. Pentru specificarea asignării porturilor, trebuie șterse caracterele # de la începutul liniilor corespunzătoare porturilor utilizate în proiect și trebuie modificate numele porturilor astfel încât acestea să corespundă cu numele utilizate în proiect. Mai întâi, ștergeți caracterele # de la începutul celor două linii care se referă la semnalul de ceas și înlocuiți în ambele linii numele portului `CLK100MHZ` (care apare după comanda `get_ports`) cu **Clk**. Apoi, ștergeți caracterele # de la începutul liniilor corespunzătoare porturilor indicate în tabelul 1.2 și modificați numele acestor porturi conform acestui tabel.

Observație

În fișierul de constrângeri se ține cont de tipul literelor, mari sau mici, din numele porturilor.

12. Salvați fișierul de constrângeri selectând iconița *Save File*, iar apoi închideți acest fișier. Se observă că sub bara de titlu a ferestrei *ELABORATED DESIGN* se afișează un mesaj indicând faptul că proiectul elaborat nu este la zi deoarece constrângerile au fost modificate. Selectați legătura **Reload** pentru procesarea constrângerilor introduse și reîncărcarea proiectului elaborat.

Tabelul 1.2. Numele porturilor care trebuie modificate în fișierul de constrângeri.

Nume inițial al portului	Nume modificat al portului
CLK100MHZ	Clk
CA	Seg[0]
CB	Seg[1]
CC	Seg[2]
CD	Seg[3]
CE	Seg[4]
CF	Seg[5]
CG	Seg[6]
DP	Seg[7]
AN[0] .. AN[7]	An[0] .. An[7]
BTNU	Up
BTNL	Mode
BTNR	Rst
BTND	Down

13. În panoul *Flow Navigator* selectați din nou opțiunea **Report Noise**, iar în caseta de dialog *Report Noise* selectați butonul **OK** pentru regenerarea raportului ssn_1. În raportul afișat nu ar mai trebui să apară erori (texte afișate cu culoarea roșie).

14. Închideți proiectul elaborat selectând opțiunea *File → Close Elaborated Design* din meniul principal și confirmați închiderea selectând butonul **OK** în caseta de dialog *Confirm Close*.

1.3.6 Setarea opțiunilor pentru sinteză și implementare

Opțiunile pentru sinteză și implementare permit modificarea comportamentului utilitarelor care execută sinteza și implementarea proiectului pentru a efectua optimizări conform cu anumite criterii. Exemple de criterii sunt timpul de execuție redus, performanța crescută a sistemului implementat sau spațiul redus ocupat în circuit. Aceste opțiuni pot fi specificate sub forma unor strategii de rulare pentru sinteză și implementare. O *strategie de rulare* reprezintă un set memorat de parametri, set care va fi utilizat în diferitele etape ale procesului de sinteză sau de implementare. Există un număr de strategii predefinite, separat pentru sinteză și pentru implementare, dar utilizatorul poate defini strategii proprii care vor fi folosite la rulările ulterioare.

Procesele de sinteză și de implementare vor fi executate pe baza unei *rulări de sinteză* și a unei *rulări de implementare*, care sunt create automat de mediul de proiectare Vivado și care utilizează strategiile de rulare selectate anterior. Aceste rulări create automat sunt afișate în fereastra *Design Runs* din partea de jos a ecranului. Utilizatorul poate crea rulări suplimentare de sinteză și implementare, care utilizează strategii de rulare diferite cu scopul de a crea mai multe versiuni ale proiectului implementat. Rulările de sinteză și implementare pot fi lansate în execuție în mod secvențial sau, dacă sunt disponibile mai multe nuclee de procesoare, în mod concurent.

Executați operațiile următoare pentru specificarea opțiunilor care vor fi utilizate pentru sinteza și implementarea proiectului.

1. În panoul *Flow Navigator*, selectați opțiunea **Settings**. Se va afișa panoul *General* din zona *Project Settings* a ferestrei de dialog *Settings*.
2. În panoul *General*, în dreptul liniei **Target Language**, selectați opțiunea **VHDL** dacă nu este selectată deja.
3. În partea stângă a ferestrei de dialog *Settings*, selectați linia **Synthesis** din zona *Project Settings*.
4. În partea dreaptă a aceleiași ferestre, în zona *Settings*, vizualizați opțiunile disponibile pentru categoria *Strategy*. După vizualizarea acestora, selectați strategia **Flow_RuntimeOptimized**. Se poate observa că o parte a parametrilor afișați în zona *Synth Design (vivado)* sunt modificați în mod automat; aceștia sunt indicați printr-un asterisc. Strategia de rulare selectată permite reducerea timpului de execuție a procesului de sinteză, dar vor fi executate mai puține optimizări.
5. Revizuiți parametrii afișați în zona *Synth Design (vivado)*. Parametrul *–flatten_hierarchy* indică modul în care este controlată ierarhia diferitelor module ale proiectului în timpul sintezei. Observați că opțiunile disponibile pentru acest parametru sunt *full*, *none* și *rebuilt*. Opțiunea *full* indică eliminarea completă a ierarhiei modulelor, păstrând doar modulul de nivel superior. Opțiunea *none* indică păstrarea ierarhiei proiectului; astfel, optimizarea va fi efectuată asupra modulelor separate, ceea ce va reduce complexitatea și va reduce timpul de procesare necesar utilitarului de sinteză. Opțiunea *rebuilt* indică utilitarului de sinteză să elimine ierarhia proiectului, să execute sinteza, iar apoi să reconstruiască ierarhia inițială a modulelor. Această opțiune permite efectuarea unor optimizări, având și avantajul că ierarhia inițială este păstrată, ceea ce permite o analiză mai simplă a proiectului. Pentru reducerea timpului de execuție, păstrați neschimbată opțiunea selectată pentru parametrul *–flatten_hierarchy* (**none**).
6. Parametrul *–directive* permite rularea procesului de sinteză ținând cont de diferite optimizări; parametrul a fost modificat automat în *RuntimeOptimized* atunci când s-a selectat strategia de rulare **Flow_RuntimeOptimized**. Păstrați neschimbat acest parametru.
7. În partea stângă a ferestrei de dialog *Settings*, selectați linia **Implementation** din zona *Project Settings*. Se vor afișa opțiunile disponibile pentru implementarea proiectului.
8. În partea dreaptă a aceleiași ferestre, în zona *Settings*, vizualizați opțiunile disponibile pentru categoria *Strategy*. După vizualizarea acestora, selectați strategia **Flow_Quick**; observați că o parte a parametrilor afișați pentru diferitele etape ale procesului de implementare sunt modificați în mod automat. Strategia

de rulare *Flow_Quick* permite reducerea timpului de execuție a procesului de implementare, dar nu ține cont de constrângerile temporale.

9. Parcurgeți parametrii care sunt setați pentru diferitele etape de implementare, dar nu modificați acești parametri. Selectați butonul **OK** pentru a închide fereastra de dialog *Settings*.

1.3.7 Sinteza proiectului

După setarea opțiunilor pentru sinteză și implementare, se poate lansa în execuție una din comenzile următoare:

- *Run Synthesis*, prin care se va executa doar procesul de sinteză;
- *Run Implementation*, prin care se va executa mai întâi procesul de sinteză dacă acesta nu s-a executat în prealabil, după care se va executa procesul de implementare;
- *Generate Bitstream*, prin care se vor executa mai întâi procesele de sinteză și de implementare dacă acestea nu s-au executat în prealabil, după care se va genera fișierul conținând șirul de biți pentru configurarea circuitului FPGA.

În acest exemplu de proiectare, cele trei comenzi vor fi lansate în execuție separat.

În etapa de sinteză a proiectului, se execută mai întâi elaborarea proiectului, chiar dacă etapa de elaborare a fost executată anterior (deoarece rezultatele elaborării proiectului nu sunt salvate). În continuare, lista de conexiuni RTL generată la elaborarea proiectului, care este o listă de conexiuni generică și nu este specifică unei anumite arhitecturi, este transformată într-o listă de conexiuni care este optimizată pentru arhitectura circuitului FPGA care va fi utilizat pentru implementare.

1.3.7.1 Lansarea în execuție a sintezei proiectului

Pentru lansarea în execuție a procesului de sinteză a proiectului, în panoul *Flow Navigator* selectați opțiunea **Run Synthesis** din secțiunea SYNTHESIS (sau selectați opțiunea *Flow* → *Run Synthesis* din meniul principal). Dacă se afișează fereastra de dialog *Launch Runs*, selectați butonul **OK**. La terminarea sintezei, se va afișa caseta de dialog *Synthesis Completed*, cu trei opțiuni disponibile. Selectați opțiunea *Open Synthesized Design*, iar apoi selectați butonul **OK**.

1.3.7.2 Analiza proiectului sintetizat

După deschiderea proiectului sintetizat, se afișează ierarhia listei de conexiuni generate în fereastra *Netlist* și schema circuitului sintetizat în fereastra *Schematic*. Executați operațiile următoare pentru analiza proiectului sintetizat.

1. Folosiți utilitarul *File Explorer* pentru a verifica faptul că a fost creat subdirectorul clock.runs în directorul proiectului, iar în acest subdirector a fost creat subdirectorul synth_1 conținând fișierele generate în urma sintezei.
2. În fereastra *Schematic*, selectați iconița *Zoom In*  pentru a putea distinge detaliile schemei. În schema generată se pot regăsi modulele care au fost instanțiate în modulul principal clock. Se poate observa însă că schema conține resurse disponibile în circuitul FPGA și nu module generice RTL. De exemplu, memoriile RTL_ROM din schema generată la elaborarea proiectului au fost înlocuite cu blocuri LUT (*Look-Up Table*) din circuitul FPGA (LUT4 reprezintă blocuri LUT cu câte patru intrări). De asemenea, au fost adăugate în mod automat buffere de intrare IBUF, un buffer pentru semnalul de ceas BUFG și buffere de ieșire OBUF.
3. După vizualizarea schemei, închideți fereastra *Schematic*.
4. Selectați panoul *Reports* din partea de jos a ecranului. Dacă acest panou nu este vizibil, selectați opțiunea *Window* → *Reports* din meniul principal.
5. În panoul *Reports*, executați un clic dublu pe linia **synth1_synth_synthesis_report_0** pentru deschiderea raportului de sinteză.
6. Parcurgeți raportul de sinteză și urmăriți care sunt componentele RTL generate în urma sintezei. Închideți apoi raportul de sinteză.
7. În panoul *Flow Navigator*, selectați opțiunea **Report Timing Summary** din secțiunea *Open Synthesized Design*. În fereastra de dialog *Report Timing Summary*, selectați butonul **OK** pentru generarea raportului de timp timing_1. Se va deschide panoul *Timing* în partea de jos a ecranului.
8. În partea din stânga a panoului *Timing*, expandați intrarea *Check Timing*. Se poate observa că sunt afișate avertismente referitoare la faptul că nu sunt specificate întârzieri pentru semnalele aplicate la intrările circuitului FPGA și pentru semnalele de ieșire generate de circuitul FPGA. Aceste avertismente pot fi ignorate pentru prezentul exemplu de proiectare.
9. Închideți panoul *Timing*.
10. În panoul *Flow Navigator*, selectați opțiunea **Report Power** din secțiunea *Open Synthesized Design*. În fereastra de dialog *Report Power*, păstrați opțiunile implicite și selectați butonul **OK** pentru generarea raportului power_1.
11. În panoul *Power* din partea de jos a ecranului se va afișa un raport cu puterea consumată estimată (figura 1.17). În secțiunea *Summary* a raportului se afișează un sumar cu puterea consumată de diferite elemente ale circuitului: semnale de ceas, alte semnale, blocuri logice și blocuri de I/E. De notat că valorile afișate reprezintă doar estimări realizate după sinteză și ele se pot modifica după

implementare. Selectați unele categorii din partea stângă a panoului *Power* pentru a vizualiza detalii despre acestea.

12. Închideți panoul *Power*.

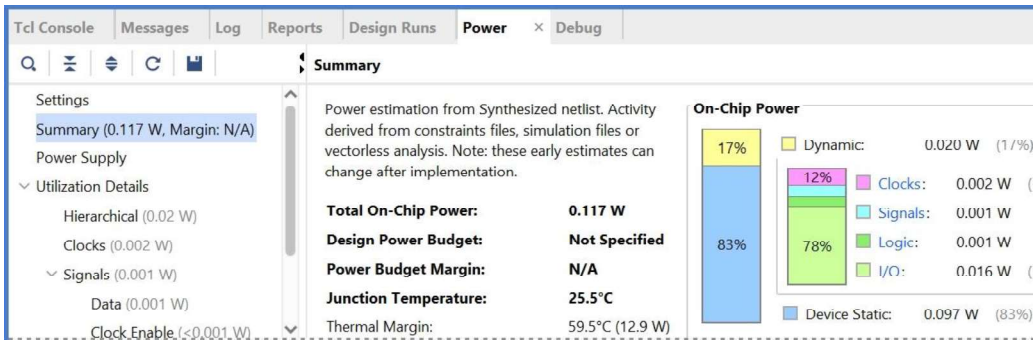

Figura 1.17. Panoul *Power*.

1.3.8 Implementarea proiectului

Procesul de implementare realizează plasarea și rutarea proiectului pe baza listei de conexiuni generate în urma sintezei. Mai întâi, se execută un sub-proces de optimizare a proiectului, după care se execută sub-procesele de plasare și de rutare. Opțional, după optimizarea proiectului se poate executa o optimizare a elementelor de proiectare pentru a se reduce puterea consumată de circuitul FPGA. De asemenea, în mod opțional, după plasare se poate executa o optimizare suplimentară pentru a se reduce puterea consumată și o optimizare a plasării ținând cont de întârzierile estimate pe baza plasării anterioare. După rutare se poate executa și o optimizare a logicii, a plasării și a rutării ținând cont de întârzierile efective de rutare.

1.3.8.1 Lansarea în execuție a implementării proiectului

Pentru lansarea în execuție a procesului de implementare a proiectului, în panoul *Flow Navigator* selectați opțiunea **Run Implementation** (sau selectați opțiunea *Flow → Run Implementation* în meniul principal). Dacă se afișează fereastra *Launch Runs*, selectați butonul **OK**. În timpul procesului de implementare, se pot urmări mesajele afișate selectând panoul *Log* din partea de jos a ecranului. La terminarea implementării, se afișează caseta de dialog *Implementation Completed*, cu trei opțiuni disponibile. Selectați opțiunea *Open Implemented Design*, iar apoi selectați butonul **OK**.

1.3.8.2 Analiza proiectului implementat

După deschiderea proiectului implementat, se afișează ierarhia listei de conexiuni în fereastra *Netlist* și structura circuitului FPGA în fereastra *Device*. Executați operațiile următoare pentru analiza proiectului implementat.

1. Folosiți utilitarul *File Explorer* și verificați că în subdirectorul clock.runs a fost creat subdirectorul impl_1 conținând fișierele generate în urma implementării.
2. În panoul *Flow Navigator*, selectați opțiunea **Report Utilization** din secțiunea *Open Implemented Design*. În caseta de dialog *Report Utilization*, selectați butonul **OK** pentru generarea raportului utilization_1. Se va deschide panoul *Utilization* în partea de jos a ecranului.
3. Selectați categoria *Slice LUTs* din partea stângă a panoului *Utilization* pentru a vizualiza numărul total de blocuri LUT utilizate și numărul de blocuri LUT utilizate de diferitele module ale proiectului. Selectați categoria *Slice Registers* pentru a vizualiza numărul total de registre utilizate și numărul de registre utilizate de diferitele module.
4. Închideți panoul *Utilization*.
5. În panoul *Flow Navigator*, selectați opțiunea **Report Timing Summary** din secțiunea *Open Implemented Design*. În fereastra de dialog *Report Timing Summary*, selectați butonul **OK** pentru generarea raportului de timp timing_1. Se va deschide panoul *Timing* în partea de jos a ecranului.
6. În partea stângă a panoului *Timing*, expandați intrarea *Intra-Clock Paths*, apoi expandați intrarea *sys_clk_pin* și selectați linia *Setup*. În partea dreaptă a panoului *Timing*, selectați una din căile afișate, de exemplu, **Path1** (figura 1.18). Calea selectată va fi evidențiată în fereastra *Device* (selectați iconița *Zoom In* pentru a putea distinge detaliile).

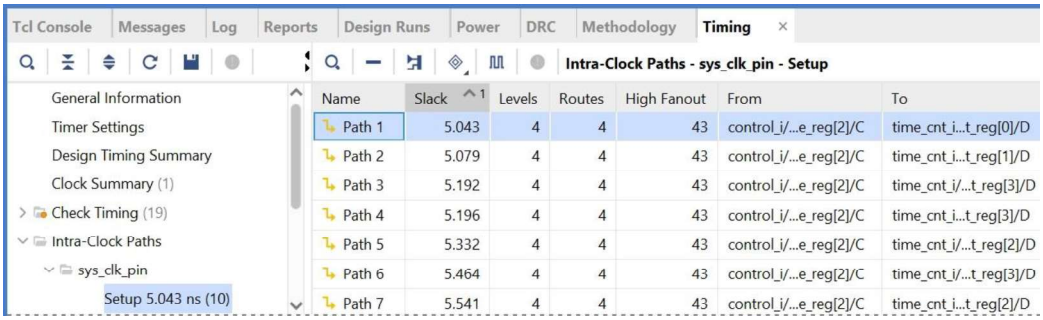

Figura 1.18. Panoul *Timing* cu calea de date Path1 selectată.

7. Selectați iconița *Routing Resources* din meniul ferestrei *Device*, dacă nu este selectată deja. Astfel se vor afișa și conexiunile de rutare în fereastra *Device*. Urmăriți rutarea căii selectate în această fereastră.
8. În timp ce una din căile din panoul *Timing* este selectată, executați un clic cu butonul din dreapta în fereastra *Device* și selectați opțiunea *Schematic*. Se va

deschide fereastra *Schematic*, în care se afișează schema căii selectate, în modul ilustrat în figura 1.19.

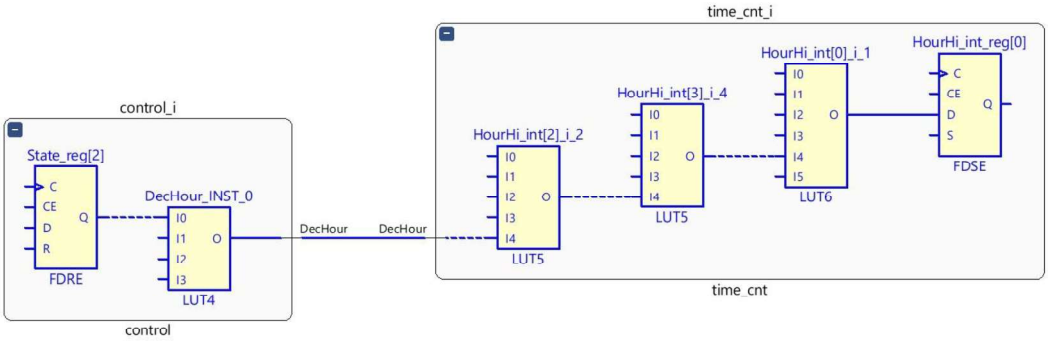

Figura 1.19. Schema căii de date Path1 selectate în panoul *Timing*.

9. În fereastra *Schematic*, fără a deselecta calea selectată, executați un clic cu butonul din dreapta într-o zonă liberă și selectați opțiunile *Expand Cone* → *To Flops or I/Os* pentru a se afișa un context mai larg al căii selectate. Urmăriți blocurile logice din calea selectată.
10. Închideți proiectul implementat selectând *File* → *Close Implemented Design* din meniul principal. Selectați butonul **OK** în caseta de dialog *Confirm Close*.

1.3.9 Generarea fișierului de configurare

Pentru a genera fișierul cu șirul de biți necesar pentru configurarea circuitului FPGA, executați operațiile descrise în continuare.

1. În panoul *Flow Navigator*, selectați opțiunea **Generate Bitstream** din secțiunea PROGRAM AND DEBUG. Dacă se afișează fereastra *Launch Runs*, selectați butonul **OK**.
2. În timpul generării fișierului, selectați panoul *Tcl Console* din partea de jos a ecranului. Se poate observa că a fost lansată în execuție comanda `launch_runs` pentru rularea de implementare impl_1 și comanda `write_bitstream`.
3. La terminarea generării fișierului, se afișează caseta de dialog *Bitstream Generation Completed*. Selectați butonul **Cancel** pentru închiderea casetei de dialog.

1.3.10 Configurarea circuitului și testarea funcționării

Pentru configurarea circuitului FPGA și testarea funcționării ceasului de timp real, executați operațiile descrise în continuare.

1. Conectați o placă de dezvoltare la un port USB al calculatorului și mutați comutatorul de alimentare de pe placă în poziția ON.
2. În panoul *Flow Navigator*, selectați opțiunea **Open Hardware Manager** din secțiunea PROGRAM AND DEBUG. Dacă se afișează fereastra de dialog *Windows Security Alert*, selectați butonul **Allow access**; poate fi necesară introducerea parolei de administrator. Se va deschide fereastra HARDWARE MANAGER indicând starea neconectată a plăcii.
3. Selectați iconița *Auto Connect*  din panoul *Hardware*. Dacă se afișează fereastra de dialog *Windows Security Alert*, selectați butonul **Allow access**; poate fi necesară introducerea parolei de administrator. Starea afișată în partea de sus a panoului ar trebui să se modifice în *Connected*.
4. Selectați legătura *Program device* din partea de sus a ferestrei HARDWARE MANAGER.
5. În caseta de dialog *Program Device*, verificați ca în câmpul *Bitstream file* să fie afișat fișierul cu extensia .bit din directorul proiectului, iar apoi selectați butonul **Program**. Fișierul de configurare va fi transmis la circuitul FPGA.
6. La terminarea configurării, se va aprinde dioda LED DONE de pe placă. Ceasul ar trebui să afișeze ora 12-00-00.
7. Apăsați butonul BTNL pentru a trece ceasul în modul de setare. Cifrele orei trebuie să pâlpâie. Setați ora utilizând butonul BTNU (pentru incrementare) sau BTND (pentru decrementare).
8. Apăsați din nou butonul BTNL. Setați minutul utilizând butonul BTNU sau BTND. Procedați în mod similar pentru setarea secundei.
9. Apăsați butonul BTNL pentru a reveni în modul de funcționare normală.
10. După terminarea testării, închideți fereastra HARDWARE MANAGER și confirmați închiderea în caseta de dialog *Confirm Close*. Închideți fereastra *Vivado* și confirmați închiderea în caseta de dialog *Exit Vivado*.
11. Opriți alimentarea plăcii de dezvoltare și deconectați placa de la calculator.

Aplicații

1.1 Răspundeți la următoarele întrebări:

a. Care sunt avantajele utilizării limbajelor de descriere hardware pentru proiectarea sistemelor digitale?

b. Care este operația principală executată în etapa de sinteză a proiectului?

c. Ce reprezintă maparea tehnologică?

d. Care sunt operațiile executate la plasarea și rutarea proiectului?

e. Care sunt utilitarele de proiectare de nivel înalt incluse în mediul de proiectare Xilinx Vivado Design Suite?

1.2 Executați etapele descrise în secțiunea 1.3 pentru exemplul de proiectare al ceasului de timp real.

1.3 Creați un nou proiect pentru un modul numit displ7seg, care este un multiplexor modificat pentru afișajul cu șapte segmente. Acest modul diferă de modulul displ7seg_blink prin faptul că nu permite pâlpâirea cifrelor afișate și are ca date de intrare valori hexazecimale pe care le decodifică în configurația necesară a segmentelor pentru afișarea valorilor respective. Intrarea *Data* a noului modul este un vector de 32 de biți, care specifică opt cifre hexazecimale de câte patru biți. Prima cifră afișată corespunde celor patru biți mai semnificativi ai vectorului de intrare (biții 31..28). După crearea modulului multiplexorului, creați un modul principal în care instanțiați entitatea displ7seg și aplicați la intrarea de date a acesteia opt caractere hexazecimale oarecare. Folosiți ca și fișier de constrângeri același fișier utilizat în exemplul de proiectare al ceasului de timp real, în care comentați liniile corespunzătoare butoanelor, cu excepția butonului de resetare.