



2. Verificarea și eliminarea erorilor

Orice proiectare a unui circuit oricără de complex presupune parcurgerea următorilor pași (reprezentați și în Figura 2.1):

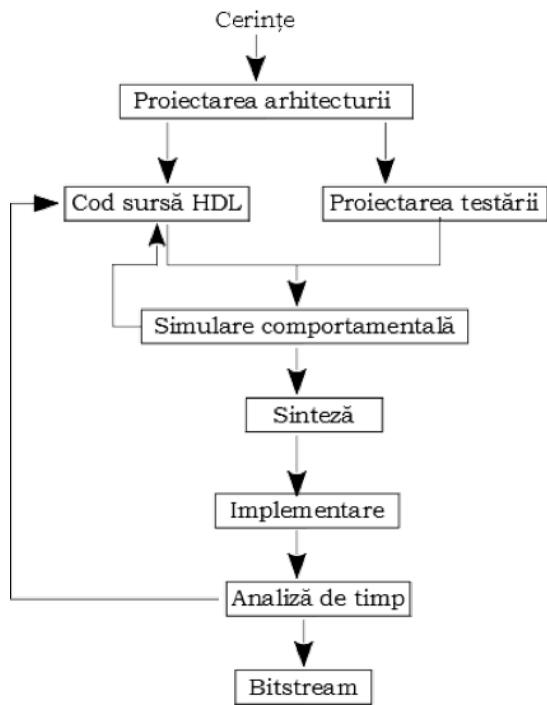


Figura 2.1: Proiectarea unui circuit

- Analiza cerințelor proiectului; descompunerea problemei și simularea funcțională
- Descrierea arhitecturii în limbaj HDL
- Scrierea de teste și modele comportamentale

- Verificarea corectitudinii HDL prin compararea ieșirilor modelului HDL cu modelul comportamental
- Conversia descrierii HDL într-un netlist de către un sintetizator
- Se verifică proiectul implementat că satisfac constrângerile de timp

Trebuie menționat faptul că faza de verificare, eliminare de erori și optimizare până se îndeplinește condițiile initiale necesită cam 70-75% din efortul de realizare a unui circuit.

Pentru a reliefa faza de debugging a unui circuit vom lua în considerare mai multe metode.

2.1 Metoda 1. VIO (Virtual I/O Core) pentru testare hardware

Pentru exemplificarea acestei metode vom folosi tot proiectul din aplicația 1. Deschidem proiectul și apoi deschidem fișierul **design_1.bd** (Block Diagram) printr-un simplu dublu-click.

Vom mai adăuga un IP nou diagramei noastre denumit VIO. Acest nou IP ne va permite eliminarea switch-urilor ca și metodă de I/O și vom folosi probe_outs / probe_ins pentru capturarea rezultatelor. Acest lucru este deosebit de util în cazul în care porturile de I/O existente pe plăcuță nu mai sunt suficiente pentru a simula comportarea circuitului proiectat.

După cum se știe deja, circuitul nostru are 3 intrări și o singură ieșire. IP-ul VIO introdus de noi are o intrare de clk, o intrare **probe_in0** și o ieșire **probe_out** (fiecare dintre ele pe mai mulți biți cu excepția semnalului clk). Vom deschide IP-ul VIO pentru a realiza niște setări conform I/O ale circuitului nostru printr-un dublu-click pe acest IP.

Se observă că avem mai multe tab-uri unde putem seta anumite valori. În conformitate cu proiectarea noastră vom avea setările prezentate în figura 8 și apoi selectăm butonul **OK**. De remarcat că după efectuarea acestor setări în tab-ul Probe_OUT Ports o să avem 3 porturi. Se mai poate observa că tab-ul PROBE_OUT Ports este acum 0 ... 2.

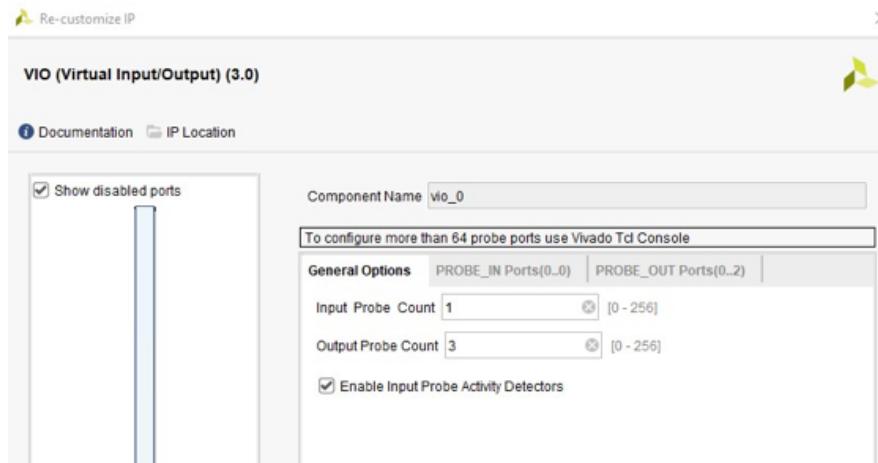


Figura 2.2: Virtual Input / Output

Vom elimina porturile vechi (A, B, C) din diagramă și le vom conecta la semnalele probe_out0, probe_out1, probe_out2. Ieșirea noastră D, o vom conecta la probe_in0. Momentan preferam să avem și ieșirea D care va rămâne conectată la led. Pentru realizarea conexiunilor este suficient să selectăm portul de început și portul de final, Vivado construind traseele firelor într-un mod optim. Dacă dorim o redesenare nu avem decât să apăsăm butonul „Regenerate Layout” și vom obține o altă amplasare a modulelor și a traseelor care fac legătura între porturi.

Ne mai rămâne portul **clk** care, aşa cum se poate intui ușor, se va face un port extern și va fi conectat la portul de ceas al placii în fișierul de constrângeri. Fișierul de constrângeri de data aceasta nu va mai contine maparea porturilor necesare pentru switch-uri, deci aceste linii le vom comenta.

NU UITAȚI !!!!! – Denumirile porturilor din fișierul de constrângeri trebuie să fie identice cu denumirile porturilor din proiectare (se face verificare la sinteză, avem cazul de CASE SENZITIVE deci denumirea unui port din proiectare trebuie să fie IDENTICĂ cu denumirea din fișierul de constrângeri)

Proiectarea finală precum și fișierul de stimuli pot fi observate în figura 2.3.

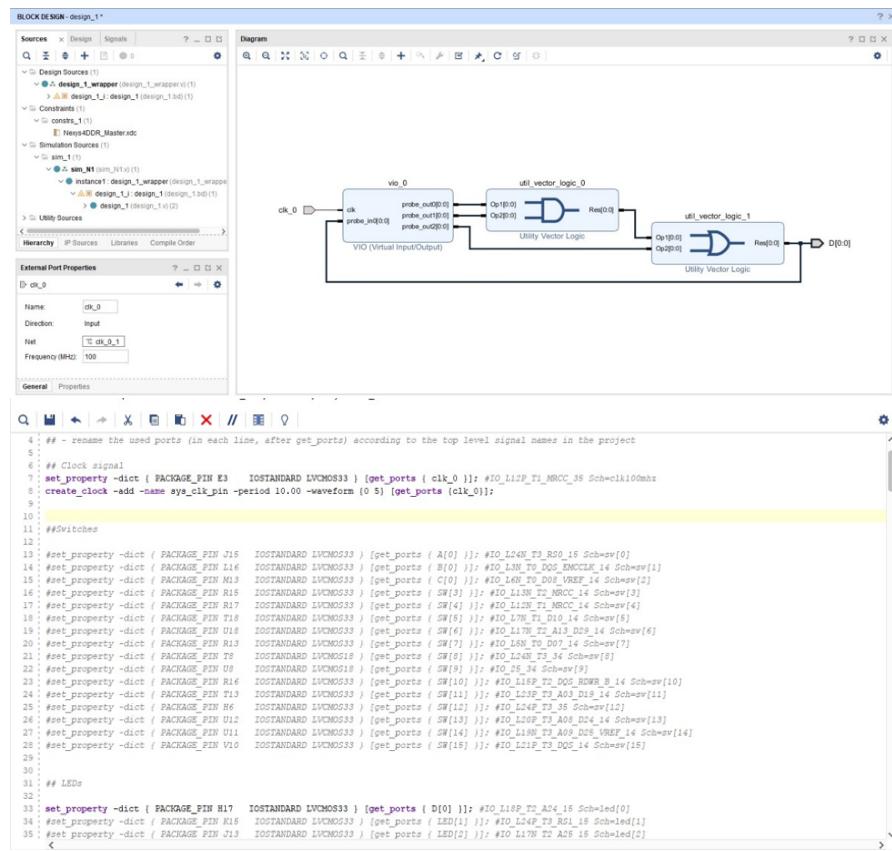


Figura 2.3: Proiectare finală

Vom selecta acum opțiunea „**Generate Bitstream**” din cadrul ferestrei „**Flow Navigator**” pentru a genera un nou fișier bitstream. Odată generat fișierul bitstream vom programa plăcuța cu noul bitstream nu înainte de a observa în fereastra „**Program device**” că pe lângă linia ce indică fișierul .bit mai avem o linie nouă care indică către fișierul de probe *.ltx.

După programarea plăcuței, în fereastra „**hw_vios**” vom adăuga toate semnalele care apar atunci când selectăm simbolul „+”. Acum vom selecta valori pentru intrări și vom observa ieșirea (1 va fi reprezentat cu săgeată albastră cu vârful în sus iar pentru valoarea 0 săgeata va fi orientată invers) atât în fereastra **hw_vios** cât și pe placă. Rezultatul final poate fi observat în figura 2.4.

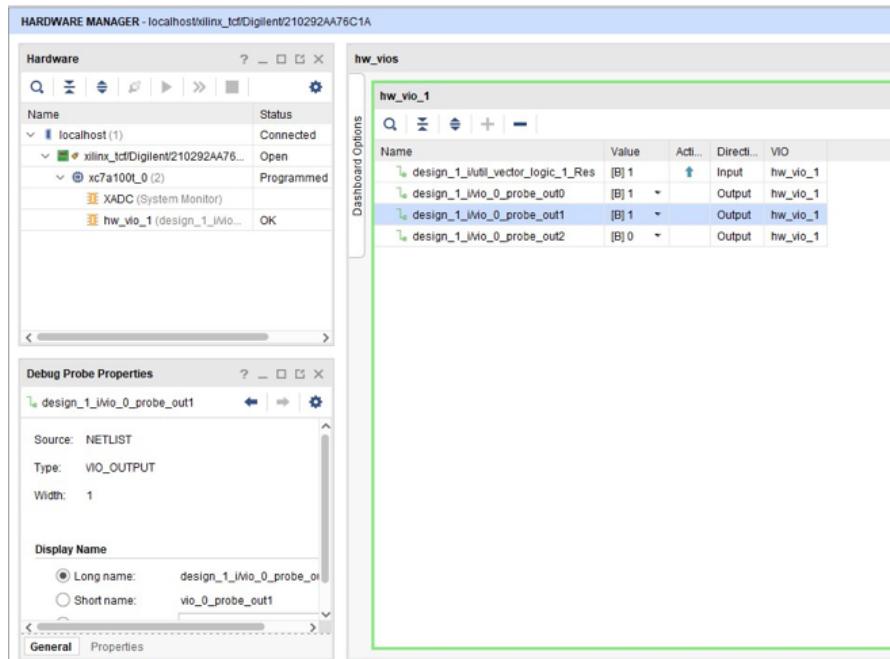


Figura 2.4: Rezultatul final

2.2 Metoda 2. Simularea circuitelor

Pentru a exemplifica această metodă, vom redeschide proiectul realizat în cadrul aplicației 1.

Pentru a simula un fișier de tip .bd (Block Diagram), acesta trebuie transformat într-un fișier Verilog (sau VHDL) – această transformare nu mai este necesară în cazul în care avem implementarea descrisă într-un limbaj HDL. Vom verifica existența fișierului design_1_wrapper.v în fereastra **Sources – Design Sources**.

Următorul pas este scrierea unui fișier de simulare în care să precizăm valorile de intrare. Astfel click dreapta în **Simulation Sources**, selectăm **Add Sources** și apoi **Add or Create Simulation Sources** (Figura 2.5).

Dăm un nume acestui fișier (de exemplu sim_N1.v), apoi OK. Fișierul cu date de simulare se scrie concret. De exemplu, pentru modulul design_1_wrapper.v provenit din design_1.bd, descris mai sus, modulul Verilog pentru simulare este descris în Figura 2.6.

Editarea unui fișier de simulare se poate face și după apăsarea click dreapta în fereastra surselor și selectarea opțiunii **Edit Simulation Sets** (Figura 2.7)

Pentru a rula simularea se selectează fișierul de simulare și se apasă Run simulation în fereastra Flow Navigator. Rezultatul va fi vizibil în dreapta, datele de ieșire generate conform designului descris de noi putând fi astfel analizate (Figura 2.8).

Este util să ne uităm și în fereastra Tcl Console pentru a observa ieșirea, aşa cum se poate observa în Figura 2.9.

Trebuie subliniat faptul că dacă linia 32 din modulul de test nu ar avea precizată o întârziere #55, atunci în **Tcl Console** ne-ar fi apărut ca și rezultat x deoarece valoarea semnalului era de la momentul de timp de simulare anterior lui #50.

Observații deosebit de utile

- Pentru adăugarea unor semnale în secțiunea de semnale afișate (simularea poate să nu conțină

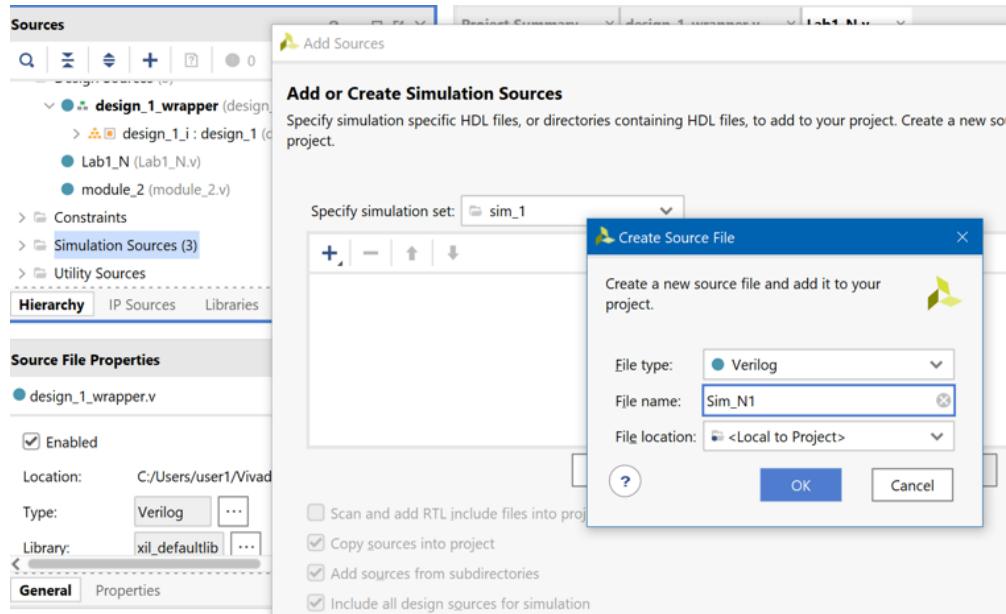


Figura 2.5: Adăugare fișier sursă simulare

toate semnalele folosite în cadrul unei implementări), se caută în fereastra **Objects** din partea stângă a ferestrei de simulare semnalul dorit, se selectează și se folosește metoda drag and drop pentru a fi adăugat în fereastra de simulare. Fiind un semnal nou se adăugat se resetează simularea.

- Pentru a șterge un semnal sau mai multe semnale din cadrul unei simulări, se selectează semnalul/semnalele în cauză și se apasă tasta **Delete**.
- Pentru a vedea valorile semnalelor din cadrul unui simulări într-o anumită bază, selectam toate semnalele prin **CTRL+A**, și apăsam click dreapta. Din meniu afișat vom selecta opțiunea **RADIX** și selectam din noul meniu baza dorită.
- Tab-ul **Messages**, de lângă tab-ul **Tcl Console** este utilizat întotdeauna în cazul în care primim eroare de simulare sau de orice alt fel. În acest tab vom primi informații legate de erorile găsite, locul unde au fost găsite (fișierul sursă) și numărul liniei unde a fost depistată eroarea.

Se poate observa că testul scris mai sus nu este unul exhaustiv. Completarea fișierului de test astfel încât circuitul să fie testat pentru orice combinație de intrare este o mică temă de făcut în cadrul laboratorului.

Metoda folosită până acum de introducere a stimulilor pentru variabilele de intrare este una deosebit de utilă în cazul proiectării unor circuite care au intrări puține și deci nu este necesar să avem multe combinații de test pentru a obține un test exhaustiv al circuitului. În cazul în care avem mai multe intrări vom prefera să avem un fișier prin care să specificăm toate combinațiile de intrare. În esență o să avem nevoie să parcurgem 2 pași.

Pasul 1 – crearea unui fișier **input.txt** (cu ajutorul programului Notepad spre exemplu) în care să specificăm valorile de intrare și rezultatul așteptat.

Pasul 2 – Scrierea fișierului de test va implica utilizarea noțiunilor de Verilog prezentate la curs. Un exemplu de astfel de fișier poate fi observat în Figura 2.10b.

Rezultatele simulării și valorile din cadrul lui Tcl Console pot fi observate în figura 2.11.

```

1  `timescale 1ns / 1ps
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

```

```

module sim_N1();
    reg A, B, C; //intotdeauna la simulare intrarile vor fi declarate ca si reg
    wire D; //intotdeauna iesirea in cazul simularii va fi de tip wire

    design_1_wrapper instance1(A, B, C, D); //instantierea modulului care ne va furniza rezultatul dorit

    initial begin
        A = 1'bX; B = 1'bX; C = 1'bX;
        #50 A = 1; B = 0; C = 0;
        #55 $display("Pentru valorile A=%d, B=%d, C=%d iesirea este %d", A, B, C, D);
        #50 A = 0; B = 0; C = 1;
        #50 A = 1'bX; B = 1'bX; C = 1'bX;
        #50 $stop;
        #50 $finish;
    end
endmodule

```

Figura 2.6: Fișier de simulare

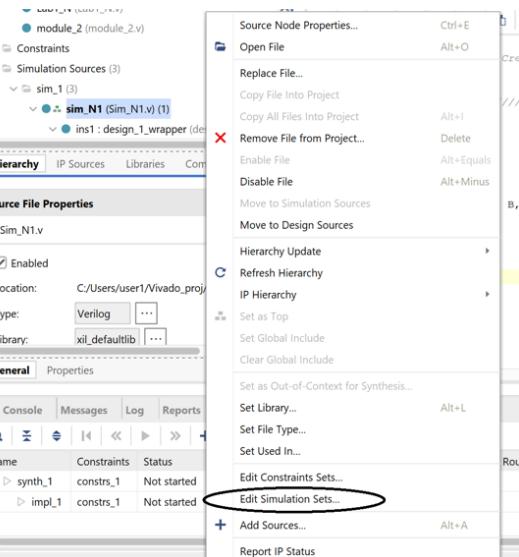


Figura 2.7: Editarea unui fișier de simulare



Figura 2.8: Rezultatul simulației

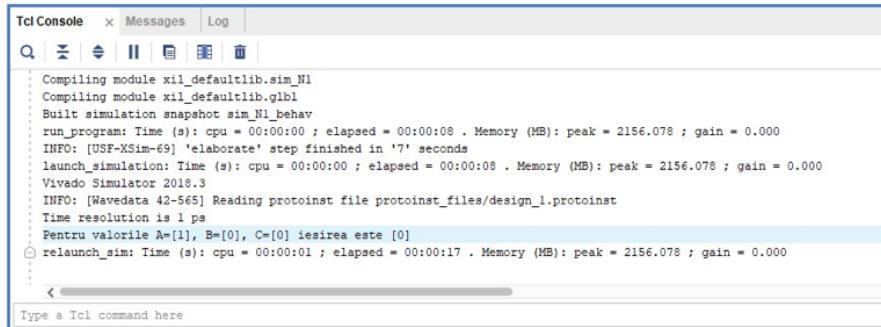


Figura 2.9: Fereastra Tcl Console

(a) Fișierul input.txt

```

1 : timescale 1ns / 1ps
2 : //intotdeauna la simulare intrarile vor fi declarate ca si reg
3 : 
4 : module sim_N1();
5 :   reg A, B, C; //intotdeauna iesirea in cazul simularii va fi de tip wire
6 :   reg [8#100:1] clinie;
7 :   integer fd, count, status, i_A, i_B, i_C, i_D, erori;
8 : 
9 :   design_1_wrapper instance(A, B, C, D); //instantierea modulului care ne va furniza rezultatul dorit
10:
11:   initial begin
12:     A = 1'bX; B = 1'bX; C = 1'bX;
13:     fd = $fopen("../..../input.txt", "r");
14:     if (fd==0)
15:       fd = $fopen("../..../input.txt", "r");
16:     count = 1;
17:     #100
18:     erori = 0;
19:     while ($fgets(clinie, fd))
20:     begin
21:       status = $sscanf(clinie, "%d %d %d %d", i_A, i_B, i_C, i_D);
22:       A = i_A; B = i_B; C = i_C;
23:       #50
24:       if (i_D == D)
25:         $display("%d@%t ok, A=%d, B=%d, C=%d, D=%d\n", count, $time, A, B, C, D); // $time in ps
26:       else begin
27:         erori = erori+1;
28:         $display("%d@%t ERORARE, A=%d, B=%d, C=%d, D(actual)=%d, D(correct)=%d", count, $time,
29:                 A, B, C, D, i_A);
30:       end
31:       count = count + 1;
32:     end
33:     #50 A = 1'bX; B = 1'bX; C = 1'bX;
34:     #50 $stop;
35:     #50 $finish;
36:   end
37: endmodule

```

(b) Exemplu fișier test

Figura 2.10: Simularea

2.3 Metoda 3. ILA (Integrated Logic Analyzer)

Această metodă este similară cu metoda 2 doar că prin această metodă putem verifica dacă circuitul nostru atinge o anumite valoare, caz în care simularea se oprește.

Vom deschide din nou proiectul din cadrul laboratorului 1, iar în cadrul diagramei vom introduce IP-ul ILA. A, B și C vor rămâne porturi externe legate la switch-uri prin intermediul fișierului de constrângeri.

Vom configura IP-ul ILA efectuând un dublu-click pe el și vom verifica dacă opțiunea „Native” este selectată. Având în vedere că avem o singură ieșire, în câmpul „Number of Probes” vom trece valoarea 1. Modalitatea de setare a IP-ului ILA și diagrama finală se pot observa în figura 2.12.

Se impun câteva explicații suplimentare:

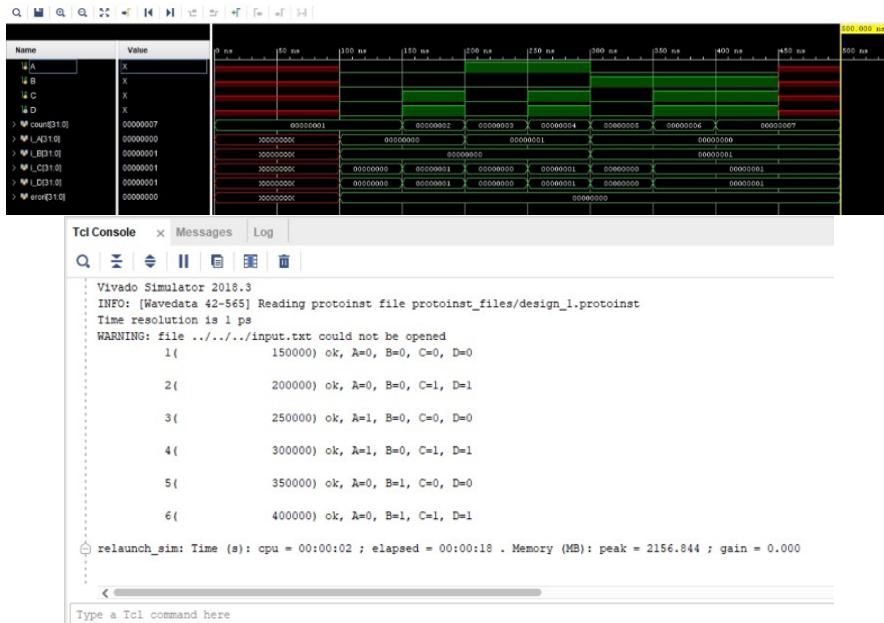


Figura 2.11: Rezultate simulare

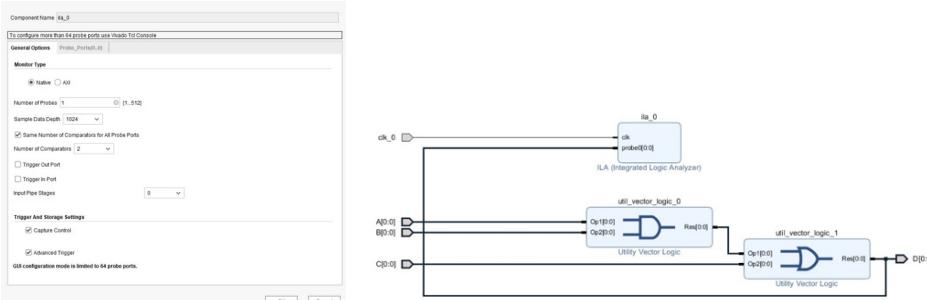


Figura 2.12: Setarea IP-ului ILA

- Opțiunea „Native” trebuie să fie activată deoarece avem ca și sursă un cod HDL. Despre monitorizarea AXI se va discuta în detaliu la CN2. Pentru moment este suficient să stim că AXI este magistrala de comunicare între mai multe dispozitive conectate la această magistrală (spre exemplu comunicarea dintre un procesor ARM și un modul Verilog).
 - „Sample Data Path” specifică cantitatea de memorie pe care o vom utiliza pentru colectarea sample-urilor. Placa hardware are în interiorul ei o memorie DRAM de 64 (Nexys4 DDR) sau 512MB (Nexys A7). În cazul în care această memorie ar lipsi, atunci se va crea automat din resursele plăcii cu ajutorul unor bistabile o memorie. Evident că această dimensiune poate fi un punct critic dacă placă de dezvoltare nu posedă suficientă memorie. În cazul nostru am păstrat valoarea implicită de 1024 dar ea poate fi ușor modificată.
 - „Same Numbers of Comparators for all Probe Ports” specifică cum putem compara valoările probelor.
 - „Trigger Out Port” este folosit de regulă pentru hardware-software codesign și aceste aspecte vor fi tratate la master-ul de AAC. Practic se compară electric portul de ieșire cu ieșirea

implementării din MATLAB spre exemplu.

- „**Trigger In Port**” folosit pentru compararea intrării observabile pe un dispozitiv (de exemplu dispozitivul Logic Analyzer de la Digilentinc existent și în cadrul laboratorului) cu intrarea generată de către FPGA.
- „**Capture Control**” ne permite să specificăm ce date intră în memorie
- „**Advanced Trigger**” ne permite vizualizarea completă a unui semnal (inclusiv fronturile coborâtoare sau crescătoare – în cazul BASIC se poate observa doar 0 sau 1) sau a lungimii unui puls.

Fisierul de constrângeri se poate observa în figura 2.13.

```

Diagram × Nexys4DDR_Master.xdc ×
C:/Users/Decebal/Desktop/labcn1/labcn1.srcc/constrs_1/imports/Vivadopl/Nexys4DDR_Master.xdc
Search | Open | Save | Print | Find | Replace | Close | Help | Exit | X | // | { } | Q |
4: ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5:
6: ## Clock signal
7: set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMS33 } [get_ports { clk_0 }]; #IO_L18P_T1_MRCC_35 Sch=clk100mhz
8: create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clk_0 }];
9:
10:
11: ##Switches
12:
13: set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMS33 } [get_ports { A[0] }]; #IO_L24N_T3_RS0_18 Sch=sv[0]
14: set_property -dict { PACKAGE_PIN J16      IOSTANDARD LVCMS33 } [get_ports { B[0] }]; #IO_L3N_T0_DQS_ENCLK_14 Sch=sv[1]
15: set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMS33 } [get_ports { C[0] }]; #IO_L6N_T0_DQS_VREF_14 Sch=sv[2]
16: set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
17: set_property -dict { PACKAGE_PIN R17      IOSTANDARD LVCMS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
18: set_property -dict { PACKAGE_PIN T18      IOSTANDARD LVCMS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sv[5]
19: set_property -dict { PACKAGE_PIN U18      IOSTANDARD LVCMS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sv[6]
20: set_property -dict { PACKAGE_PIN R13      IOSTANDARD LVCMS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D0T_14 Sch=sv[7]
21: set_property -dict { PACKAGE_PIN T19      IOSTANDARD LVCMS18 } [get_ports { SW[8] }]; #IO_L24N_T3_94 Sch=sv[8]
22: set_property -dict { PACKAGE_PIN U13      IOSTANDARD LVCMS18 } [get_ports { SW[9] }]; #IO_L26_34 Sch=sv[9]
23: set_property -dict { PACKAGE_PIN R11      IOSTANDARD LVCMS33 } [get_ports { SW[10] }]; #IO_L18P_T2_DQS_RDW_B_14 Sch=sv[10]
24: set_property -dict { PACKAGE_PIN V10      IOSTANDARD LVCMS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sv[11]
25: set_property -dict { PACKAGE_PIN H6      IOSTANDARD LVCMS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sv[12]
26: set_property -dict { PACKAGE_PIN U12      IOSTANDARD LVCMS33 } [get_ports { SW[13] }]; #IO_L0P_T3_A08_D24_14 Sch=sv[13]
27: set_property -dict { PACKAGE_PIN U11      IOSTANDARD LVCMS33 } [get_ports { SW[14] }]; #IO_L18N_T3_A09_D25_VREF_14 Sch=sv[14]
28: set_property -dict { PACKAGE_PIN J13      IOSTANDARD LVCMS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQG_14 Sch=sv[15]
29:
30:
31: ## LEDs
32:
33: set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMS33 } [get_ports { D[0] }]; #IO_L18P_T2_A24_18 Sch=led[0]
34: set_property -dict { PACKAGE_PIN K15      IOSTANDARD LVCMS33 } [get_ports { LED[1] }]; #IO_L24P_T3_RSL_15 Sch=led[1]
35: set_property -dict { PACKAGE_PIN J13      IOSTANDARD LVCMS33 } [get_ports { LED[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
<

```

Figura 2.13: Fisierul de constrângeri

Vom genera din nou fisierul bitstream și vom programa plăcuța cu noul fisier generat. După acest pas se vor deschide 3 ferestre:

- Fereastra „Waveform-hw_il_1” este fereastra care ne va arăta când se va atinge condiția noastră
- Din fereastra „Settings-hw_il_1” vom seta modul de capturare pe BASIC și apoi în fereastra „Status-hw_il_1” vom putea porni execuția
- Din fereastra „Trigger setup-hw_il_1” vom putea seta condiția care trebuie îndeplinită de către ieșire pentru a verifica că o anumită valoare este luată sau nu

Rezultatele simulării pot fi observate în figura 14 și pentru a le obține este nevoie să modificăm poziția switch-urilor.

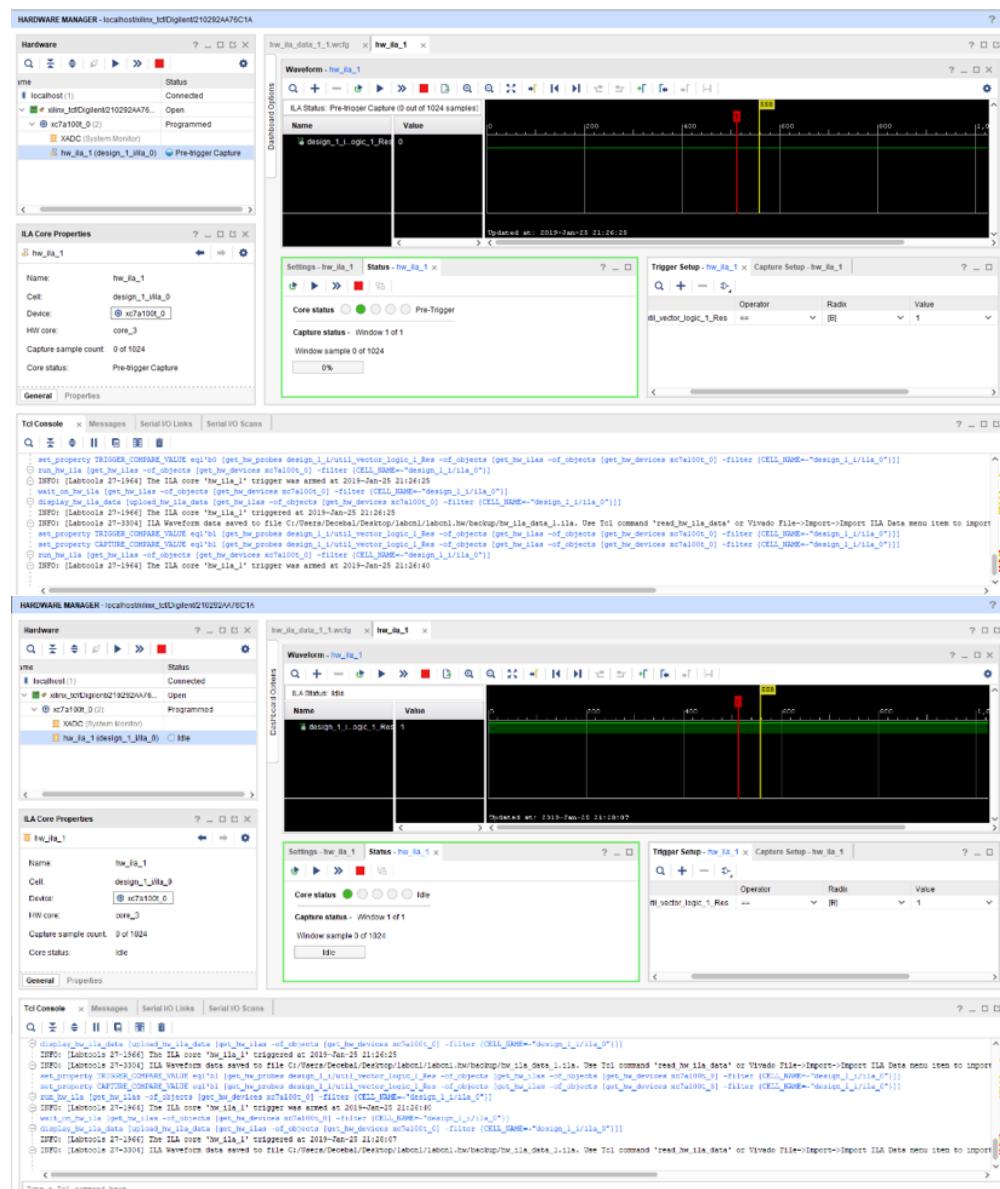


Figura 2.14: Rezultatele simulării