CAPITOLUL 6

TESTAREA ȘI DEPANAREA PROIECTELOR VHDL

În acest capitol se prezintă modulele și utilitarele disponibile în mediul de proiectare Xilinx Vivado Design Suite pentru testarea funcționării și pentru depanarea proiectelor VHDL. Sunt descrise etapele necesare pentru utilizarea modulelor de depanare și a analizorului logic Vivado în scopul testării și depanării unui proiect al unui transmițător serial UART.

6.1 Prezentare generală

Mediul de proiectare Vivado Design Suite pune la dispoziție module de depanare care pot fi adăugate la un proiect prin configurarea și generarea modulelor, instanțierea lor în codul sursă și conectarea modulelor la proiect înaintea procesului de sinteză. În mod alternativ, este posibilă configurarea modulelor de depanare și inserarea lor în lista de conexiuni a proiectului cu ajutorul utilitarului de asistență *Set Up Debug* după procesul de sinteză. În ambele cazuri, proiectul este apoi implementat în circuitul FPGA, după care utilizatorul configurează circuitul FPGA și analizează proiectul folosind interfața grafică oferită de analizorul logic Vivado. Acesta este un analizor logic bazat pe software care permite monitorizarea stării semnalelor selectate dintr-un proiect pentru a detecta eventualele erori de proiectare.

Monitorizarea stării semnalelor este posibilă prin capturarea și memorarea valorii acestora atunci când se produc anumite evenimente. Volumul de date care poate fi memorat este limitat de memoria Block RAM (BRAM) disponibilă în circuitul FPGA. Utilizatorii pot seta condiții complexe de declanșare pentru a specifica momentul în care trebuie să înceapă capturarea datelor, ceea ce permite detectarea apariției anumitor evenimente care sunt importante pentru verificarea și depanarea proiectului. Condițiile de declanșare pot fi schimbate în timpul funcționării circuitului fără a fi necesară recompilarea proiectului.

Față de analizoarele logice bazate pe hardware, analizorul logic Vivado are mai multe avantaje. Unul din avantaje este că analizoarele logice bazate pe hardware sunt mult mai costisitoare. Un alt avantaj este acela că prin utilizarea analizorului logic Vivado este posibilă analiza unui număr mare de semnale (până la 1024), în timp ce analizoarele logice tradiționale au un număr limitat (de exemplu, 16) de canale de date, de multe ori insuficiente pentru depanarea proiectelor complexe. De asemenea, analizorul logic Vivado permite specificarea unor condiții și moduri de declanșare mult mai complexe.

6.2 Module de depanare

Principalele tipuri de module de depanare necesare pentru utilizarea analizorului logic Vivado sunt modulul ILA (*Integrated Logic Analyzer*) și distribuitorul de depanare *Debug Hub*. În mod opțional, se poate utiliza și modulul VIO (*Virtual Input/Output*). Pentru proiecte complexe, sunt disponibile și alte module de depanare, cum sunt analizorul serial IBERT (*Integrated Bit Error Ratio Tester*), care permite măsurarea și optimizarea calității semnalelor din legăturile seriale cu viteze ridicate, sau *JTAG-to-AXI Master*, care permite generarea unor tranzacții pe magistrala AXI (*Advanced eXtensible Interface*) pentru interacțiunea cu diferite module AXI într-un sistem complex. Aceste module de depanare mai complexe nu sunt descrise în acest capitol.

6.2.1 Distribuitorul de depanare *Debug Hub*

Toate modulele de depanare utilizează interfața JTAG (*Joint Test Action Group*) *Boundary Scan* a circuitului FPGA pentru a comunica cu calculatorul gazdă prin intermediul unui cablu de configurare JTAG (de exemplu, un cablu USB). Modulul *Debug Hub* pune la dispoziție o cale de comunicare între portul JTAG *Boundary Scan* al circuitului FPGA și celelalte module de depanare (ILA, VIO).

Modulul *Debug Hub* nu poate fi instanțiat de către utilizator, ci va fi configurat și inserat în mod automat de către mediul de proiectare Vivado în etapa de optimizare a implementării proiectului. Pentru comunicarea cu logica JTAG *Boundary Scan* a circuitului FPGA, modulul *Debug Hub* utilizează o componentă primitivă *Boundary Scan* (BSCAN).

Figura 6.1 ilustrează comunicația dintre modulele *Debug Hub*, ILA și VIO.

6.2.2 Modulul ILA

Modulul *Integrated Logic Analyzer* (ILA) este un analizor logic configurabil care poate fi utilizat pentru a monitoriza orice semnal intern al proiectului. Modulul ILA este sincron cu proiectul monitorizat, astfel încât toate constrângerile referitoare la semnalul de ceas care sunt aplicate proiectului respectiv sunt aplicate și componentelor din interiorul modulului ILA.

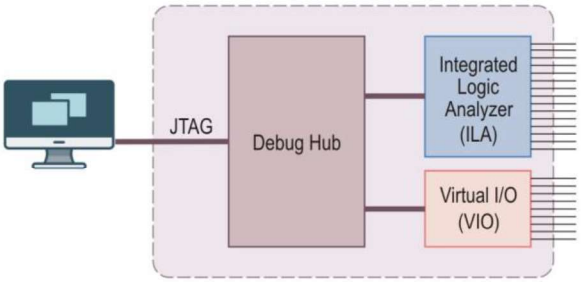

Figura 6.1. Comunicația dintre modulele *Debug Hub*, ILA și VIO.

Fiecare modul ILA poate avea până la 1024 porturi de intrare, pe care se conectează semnalele care trebuie monitorizate și cele care trebuie să declanșeze capturarea datelor. Fiecare port poate avea dimensiunea între 1 și 4096 biți. Modulul ILA permite detectarea unor evenimente care să declanșeze capturarea datelor. Pentru a detecta evenimente la un port de intrare, la configurarea modulului ILA utilizatorul poate specifica utilizarea a până la patru comparatoare pentru fiecare port, un comparator individual fiind numit *unitate de potrivire*. Astfel, este posibilă efectuarea unor comparații multiple asupra semnalelor portului de intrare.

Rezultatele mai multor unități de potrivire sunt combinate printr-o funcție booleană (AND, OR, NAND sau NOR) pentru a forma evenimentul global care va fi utilizat pentru a controla capturarea datelor. Selectarea unei singure unități de potrivire economisește resursele, permițând în același timp și o anumită flexibilitate în detectarea evenimentelor de declanșare. Selectarea a două sau a mai multor unități de potrivire permite o ecuație mai flexibilă a condiției de declanșare, dar crește utilizarea resurselor logice.

O *condiție de declanșare* este o combinație secvențială sau booleană de evenimente care este detectată de unitățile de potrivire ale porturilor de intrare ale modulului ILA. Condiția de declanșare este folosită pentru a specifica punctul inițial în fereastra de capturare a datelor.

O *condiție de calificare a memorării* este, de asemenea, o combinație booleană de evenimente care este detectată de unitățile de potrivire. Condiția de calificare a memorării diferă însă de condiția de declanșare prin faptul că evaluează evenimentele unităților de potrivire ale portului de intrare pentru a decide dacă trebuie sau nu capturat și memorat fiecare eșantion de date. Condiția de declanșare și condiția de calificare a memorării pot fi utilizate împreună pentru a defini momentul de început al procesului de capturare și datele care trebuie capturate.

Modulul ILA poate fi configurat pentru a utiliza fie un mod de declanșare de bază (BASIC), fie un mod de declanșare avansat (ADVANCED). În cazul modului de declanșare de bază, fiecare unitate de potrivire poate implementa un operator obișnuit de comparare (==, !=, <, <=, >, >=) și poate detecta tranziții ale semnalelor, cum sunt frontul crescător (R), frontul descrescător (F), ambele fronturi (B), sau lipsa

tranzițiilor (N). Selectarea funcției operatorului de potrivire se realizează prin interfața grafică a analizorului logic Vivado în timpul testării și nu în timpul configurării modulului ILA. În cazul modului de declanșare avansat, declanșarea se realizează de către un automat de stare specificat de utilizator. Automatul de stare poate conține până la 16 stări, iar fiecare stare poate conține ramificații cu până la trei alternative. Se pot utiliza până la patru numărătoare pentru a contoriza evenimente multiple.

Modulul ILA poate fi configurat pentru a se valida un port de intrare numit TRIG_IN. Dacă se validează acest port, condiția de declanșare poate fi setată ca fiind rezultatul funcției booleene OR între starea pinului TRIG_IN și condiția de declanșare rezultată din setarea modului de declanșare de bază sau al celui avansat. Declanșarea se poate realiza și atunci când semnalul de pe pinul TRIG_IN tranzitează din starea 0 logic în starea 1 logic.

Este posibilă și validarea unui port de ieșire TRIG_OUT al modulului ILA. Acest port poate fi conectat la un pin al circuitului pentru a declanșa echipamente de test externe, cum sunt osciloscoape și analizoare logice. Se poate specifica propagarea la acest port fie a condiției de declanșare, fie a semnalului de pe intrarea TRIG_IN, fie a rezultatului funcției booleene OR între cele două.

6.2.3 Modulul VIO

Modulul *Virtual Input/Output* (VIO) este un modul care poate monitoriza semnale ale proiectului testat și poate genera semnale pentru proiectul testat, în timp real, printr-un port JTAG. Numărul și dimensiunea porturilor de intrare și de ieșire ale modulului VIO sunt configurabile. Modulul poate fi configurat cu până la 256 de porturi de intrare și/sau de ieșire (din care primele 64 se pot configura din interfața grafică), iar fiecare port poate fi configurat cu o dimensiune de până la 256 de biți. Spre deosebire de modulul ILA, modulul VIO nu necesită resurse de memorare. Interacțiunea cu acest modul se poate realiza numai prin analizorul logic Vivado.

Modulul VIO eșantionează periodic starea semnalelor conectate la porturile sale de intrare, provenite de la proiectul testat, și le afișează în interfața grafică a analizorului logic Vivado sub formă textuală sau sub forma unor diode LED virtuale. Semnalele sunt eșantionate și memorate în mod sincron cu semnalul de ceas aplicat la intrarea de ceas a modulului VIO. Se recomandă ca acest semnal de ceas să fie sincron cu logica atașată la porturile de intrare ale modulului.

De asemenea, modulul VIO poate genera semnale prin porturile sale de ieșire, iar acestea pot fi utilizate în proiectul testat. Semnalele sunt definite de către utilizator în interfața grafică a analizorului logic Vivado utilizând butoane sau comutatoare virtuale. Aceste semnale sunt generate în mod sincron cu semnalul de ceas aplicat la intrarea de ceas a modulului, care ar trebui să fie sincron cu logica atașată la porturile de ieșire ale modulului.

Fiecare intrare a modulului VIO are celule adiționale pentru a captura prezența tranzițiilor la intrare. Deoarece este foarte probabil că ceasul proiectului este mult mai

rapid decât perioada de eşantionare a analizorului, este posibil ca semnalul monitorizat să tranziteze de mai multe ori între eşantioanele succesive. *Detectoarele de activitate* capturează acest comportament şi rezultatele sunt afişate în acelaşi timp cu valoarea semnalului în interfaţa grafică a analizorului.

6.3 Etapele necesare pentru testare şi depanare

În general, testarea şi depanarea unui proiect constă din trei faze distincte. În prima fază se selectează semnalele care vor fi monitorizate şi se alege metoda care se va utiliza pentru testare şi depanare. În faza a doua se adaugă la proiect modulele de depanare necesare, se conectează la acestea semnalele selectate pentru monitorizare şi se realizează implementarea proiectului. În faza a treia se configurează circuitul FPGA şi se realizează analiza funcţionării proiectului, urmărind starea semnalelor care au fost selectate pentru monitorizare.

Pentru testarea şi depanarea unui proiect se poate utiliza una din următoarele două metode principale: instanţierea modulelor de depanare în codul sursă sau inserarea modulelor de depanare în lista de conexiuni a proiectului sintetizat. Inserarea modulelor de depanare în lista de conexiuni nu se poate utiliza pentru un modul VIO, astfel încât pentru acest modul trebuie să se utilizeze metoda instanţierii în codul sursă. Cele două metode sunt ilustrate în figura 6.2.

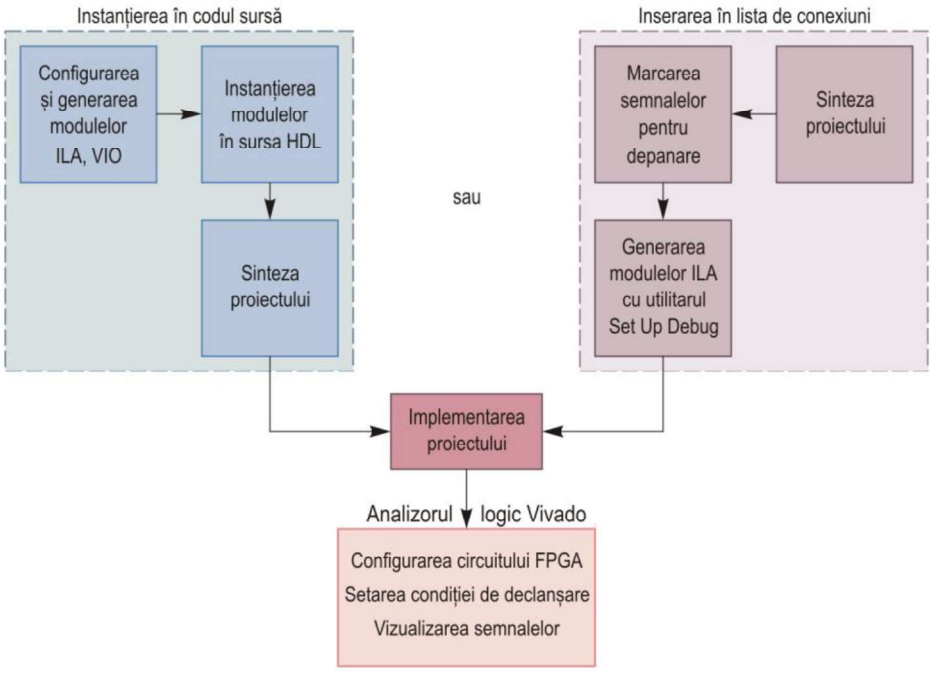

Figura 6.2. Metode de testare şi depanare care se pot utiliza cu analizorul logic Vivado.

Atunci când se utilizează metoda instanțierii modulelor de depanare în codul sursă, trebuie parcurse următoarele etape:

1. Identificarea semnalelor care vor fi monitorizate în timpul testării și, opțional, a semnalelor care vor fi generate cu ajutorul unui modul VIO, alegând acele semnale care sunt relevante pentru a urmări funcționarea circuitului sau a sistemului testat.
2. Configurarea modulelor de depanare care vor fi utilizate (ILA și, opțional, VIO), selectând numărul de porturi necesare și dimensiunea acestora, în funcție de semnalele selectate anterior.
3. Generarea listei de conexiuni a modulelor de depanare configurate.
4. Instanțierea în codul sursă a modulelor de depanare și conectarea la porturile acestora a semnalelor selectate pentru a fi monitorizate și a semnalelor care vor fi generate cu ajutorul modulului VIO.
5. Realizarea sintezei proiectului, a implementării acestuia și generarea șirului de biți pentru configurare.
6. Configurarea circuitului FPGA și utilizarea analizorului logic Vivado pentru a urmări funcționarea circuitului sau a sistemului.

Metoda inserării modulelor de depanare în lista de conexiuni se poate utiliza numai pentru modulul ILA. Dacă se utilizează această metodă, etapele care trebuie parcurse sunt următoarele:

1. Realizarea sintezei proiectului.
2. Selectarea semnalelor care vor fi monitorizate în timpul testării.
3. Marcarea semnalelor selectate prin specificarea atributului `mark_debug`. Acest atribut poate fi specificat direct în codul sursă, înaintea sintezei proiectului. Marcarea semnalelor se poate realiza și din interfața grafică, prin setarea opțiunii *Mark Debug* pentru fiecare semnal.
4. Lansarea utilitarului de asistență *Set Up Debug* pentru configurarea modulelor de depanare care se vor utiliza, conectarea semnalelor la porturile acestor module și generarea listei de conexiuni a modulelor.
5. Implementarea proiectului și generarea șirului de biți pentru configurare.
6. Configurarea circuitului FPGA și utilizarea analizorului logic Vivado pentru a urmări funcționarea circuitului sau a sistemului.

Atributul `mark_debug` se poate specifica în codul sursă astfel:

```
attribute mark_debug : STRING;
attribute mark_debug of <nume_semnal> : signal is "TRUE";
```

De multe ori, poate fi utilă specificarea atributului `keep` pentru ca utilitarul de sinteză Vivado să nu modifice numele semnalelor după realizarea sintezei. Acest atribut poate fi specificat în codul sursă astfel:

```
attribute keep : STRING;
attribute keep of <nume_semnal> : signal is "TRUE";
```

6.4 Exemplu de testare și depanare

Această secțiune prezintă etapele necesare pentru crearea unui proiect al unui transmițător serial asincron UART (*Universal Asynchronous Receiver-Transmitter*), adăugarea unui modul VIO la acest proiect și testarea funcționării transmițătorului. Se descrie apoi adăugarea unui modul ILA la proiect și utilizarea analizorului logic Vivado pentru depanarea transmițătorului.

6.4.1 Comunicația serială asincronă

În cazul comunicației seriale asincrone, fiecare caracter transmis este precedat de un bit de START, pentru care linia de comunicație are nivelul logic 0, și este urmat de cel puțin un bit de STOP, pentru care linia de comunicație are nivelul logic 1. Deci, biții de START și de STOP încadrează fiecare caracter transmis. Intervalul de timp între transmisia a două caractere succesive este variabil, pe durata acestui interval linia de comunicație având nivelul logic 1.

Atunci când receptorul detectează bitul de START care indică începutul unui caracter, pornește un oscilator de ceas local, care permite măsurarea intervalului de timp corespunzător unui bit, interval care depinde de debitul binar. Acest oscilator permite eșantionarea corectă a biților individuali ai caracterului. Eșantionarea biților se realizează aproximativ la mijlocul intervalului corespunzător fiecărui bit.

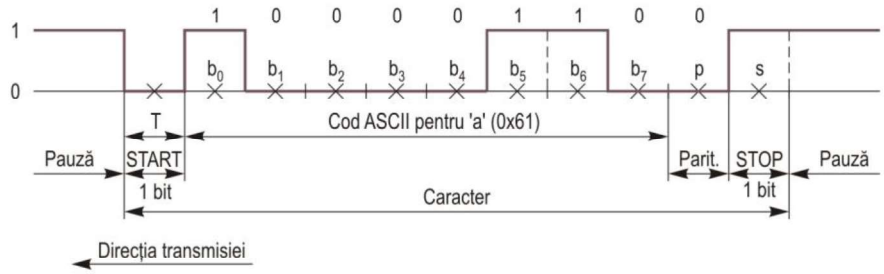

Figura 6.3. Transmisia serială asincronă a caracterului cu codul ASCII 0x61.

Figura 6.3 ilustrează transmisia serială asincronă a caracterului cu codul ASCII 0x61. După bitul de START, având durata T corespunzătoare unui bit, transmisia caracterului începe cu bitul cel mai puțin semnificativ b_0. Urmează transmisia biților următori până la bitul cel mai semnificativ b_7, după care se transmite un bit de paritate p; în acest exemplu, paritatea este impară (caracterul conține un număr impar de biți

de 1, excluzând biții de START și de STOP). Bitul de paritate este opțional, iar în cazul în care se adaugă la caracterul transmis, paritatea poate fi selectată pentru a fi pară sau impară. În exemplul ilustrat, la sfârșitul caracterului se transmite un bit de STOP *s*, după care linia rămâne la nivelul logic 1 un timp nedefinit. Acest timp corespunde unui interval de pauză între transmisia a două caractere succesive.

6.4.2 Descrierea transmițătorului serial UART

Modulul transmițătorului serial care se va realiza va permite specificarea debitului binar (a vitezei de comunicație) printr-un generic. Caracterele transmise vor fi de 8 biți, fără bit de paritate, și vor fi urmate de un singur bit de STOP. Acest modul se poate implementa cu ajutorul unui registru de deplasare cu încărcare paralelă și ieșire serială. Va mai fi necesară o unitate de control care generează semnalul de stare *TxRdy* și semnalele de comandă necesare registrului de deplasare. Unitatea de control va include și numărătoare pentru contorizarea biților transmiși și pentru măsurarea duratei unui bit, în funcție de debitul binar specificat.

Schema bloc a modulului transmițător este prezentată în figura 6.4. Intrările modulului sunt semnalul de ceas *Clk*, semnalul de resetare sincronă *Rst*, vectorul de opt biți *TxData*, reprezentând octetul care trebuie transmis, și semnalul *Start*, reprezentând comanda de începere a transmisiei octetului de la intrarea *TxData*. Ieșirile modulului sunt linia *Tx* pe care sunt transmise datele în mod serial și semnalul *TxRdy*, care indică prin starea sa activă faptul că transmisia octetului aplicat la intrare a fost terminată.

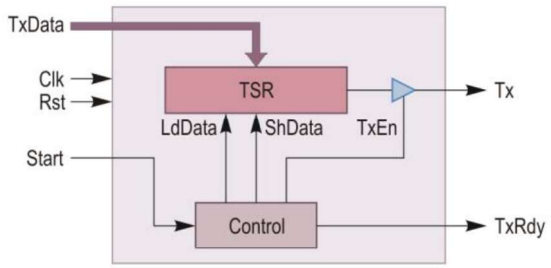

Figura 6.4. Schema bloc a transmițătorului serial UART.

Registrul de deplasare TSR este încărcat cu octetul care trebuie transmis, completat cu bitul de START ('0') și cu bitul de STOP ('1'); deci, registrul va avea 10 biți și se va deplasa la dreapta. Modulul transmițătorului conține și un buffer comandat de semnalul *TxEn*. Atunci când semnalul *TxEn* este activat, ieșirea serială a registrului de deplasare va fi transmisă pe linia *Tx*, iar în caz contrar linia *Tx* va avea nivelul logic 1, corespunzător intervalului de pauză.

Unitatea de control se poate implementa prin automatul de stare cu diagrama ilustrată în figura 6.5. Se utilizează un semnal *CntBit* pentru contorizarea biților

transmiși pe linia serială și un semnal *CntRate* pentru contorizarea ciclurilor de ceas în scopul măsurării intervalului de timp în care trebuie menținut fiecare bit pe linia serială. Atunci când semnalul *Start* devine activ, automatul trece în starea `load`, în care se încarcă registrul de deplasare cu octetul care trebuie transmis, completat cu biții de START și de STOP. În continuare, automatul trece în starea `send`, în care va reveni după transmisia unui bit. În starea `waitbit` se așteaptă trecerea intervalului de timp egal cu durata unui bit, incrementând contorul *CntRate*. Dacă acest contor ajunge la valoarea `T_BIT`, care reprezintă numărul ciclurilor de ceas corespunzători duratei unui bit, automatul trece în starea `shift`, în care se deplasează la dreapta registrul de deplasare și se incrementează contorul *CntBit*. Dacă nu s-au transmis toți cei 10 biți, automatul revine în starea `send`, iar în caz contrar acesta revine în starea `ready` în care se activează semnalul *TxRdy*, indicând terminarea transmiterii unui octet.

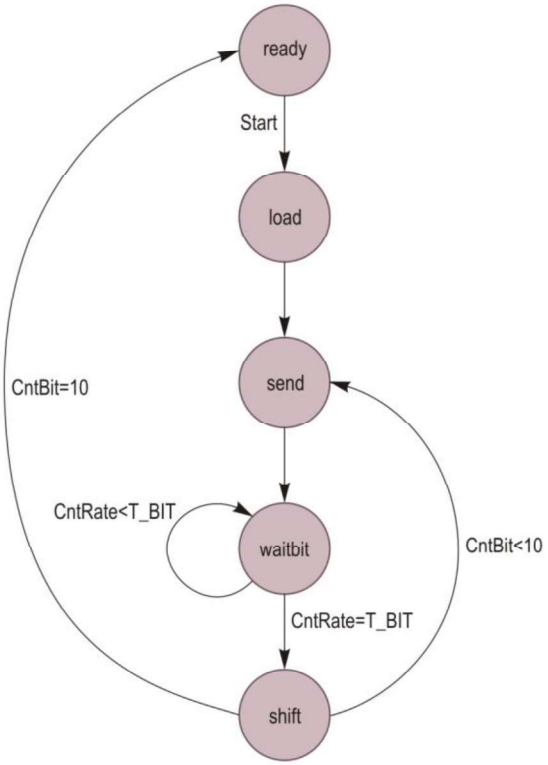

Figura 6.5. Diagrama de stare a unității de control pentru transmițătorul serial UART.

Descrierea în limbajul VHDL a automatului de stare care implementează unitatea de control este prezentată în Exemplul 6.1.

Exemplul 6.1

```
-- Automat de stare pentru unitatea de control a transmitatorului serial
   proc_control: process (Clk)
   begin
      if RISING_EDGE (Clk) then
         if (Rst = '1') then
            St <= ready;
         else
            case St is
               when ready =>
                  CntRate <= 0;
                  CntBit <= 0;
                  if (Start = '1') then
                     St <= load;
                  end if;
               when load =>
                  St <= send;
               when send =>
                  St <= waitbit;
               when waitbit =>
                  CntRate <= CntRate + 1;
                  if (CntRate = T_BIT) then
                     CntRate <= 0;
                     St <= shift;
                  end if;
               when shift =>
                  CntBit <= CntBit + 1;
                  if (CntBit = 10) then
                     St <= ready;
                  else
                     St <= send;
                  end if;
               when others =>
                  St <= ready;
            end case;
         end if;
      end if;
   end process proc_control;

-- Setarea semnalelor de comanda
   LdData <= '1' when St = load else '0';
   ShData <= '1' when St = shift else '0';
   TxEn   <= '0' when St = ready or St = load else '1';

-- Setarea semnalelor de iesire
   Tx    <= TSR(0) when TxEn = '1' else '1';
   TxRdy <= '1' when St = ready else '0';
```

6.4.3 Crearea modulului transmițătorului serial UART

Executați următoarele operații pentru crearea modulului VHDL al transmițătorului serial UART. La descrierea acestor operații, se presupune utilizarea plăcii de dezvoltare Nexys4 DDR sau Nexys A7-100T, dar se poate utiliza orice placă care conține o interfață serială RS-232, două butoane și o diodă LED.

1. În mediul de proiectare Vivado, creați un nou proiect pentru placa de dezvoltare utilizată.
2. Creați un fișier sursă VHDL pentru modulul transmițătorului; numele acestui fișier poate fi, de exemplu, uart_tx. Porturile de intrare și de ieșire ale modulului sunt cele ilustrate în figura 6.4.
3. În entitatea modulului creat, adăugați un generic de tip **INTEGER** pentru rata de biți și inițializați genericul cu valoarea 115_200 (pentru debitul binar de 115.200 biți pe secundă).
4. Copiați codul din secțiunea 6.4.2 în fișierul sursă al modulului și specificați utilizarea pachetului **STD_LOGIC_UNSIGNED**.
5. În partea declarativă a arhitecturii, declarați o constantă de tip **INTEGER** pentru frecvența semnalului de ceas și inițializați constanta cu 100_000_000 (pentru frecvența de 100 MHz).
6. Declarați constanta `T_BIT` de tip **INTEGER** și inițializați-o cu numărul ciclurilor de ceas corespunzători duratei unui bit. Acest număr se poate obține împărțind frecvența semnalului de ceas cu genericul reprezentând rata de biți.
7. Declarați un tip enumerat pentru stările automatului cu diagrama de stare din figura 6.5, declarați un semnal de stare *St* de acest tip și inițializați semnalul cu starea `ready`.
8. Declarați semnalele *CntBit* și *CntRate* de tip **INTEGER** și inițializați aceste semnale cu 0.
9. Declarați semnalele de comandă *LdData*, *ShData*, *TxEn* și inițializați aceste semnale cu '0'.
10. Declarați semnalul *TSR* de 10 biți pentru registrul de deplasare și inițializați semnalul cu (`others` => '0').
11. În același modul al transmițătorului, scrieți un proces pentru registrul de deplasare TSR. Registrul utilizează semnalul de ceas *Clk* și semnalul de resetare sincronă *Rst*. Atunci când semnalul *LdData* este activ, registrul se încarcă cu datele de intrare *TxData*, completate cu un bit de START ('0') în poziția bitului 0 și cu un bit de STOP ('1') în poziția bitului 9. Atunci când semnalul *ShData* este activ, conținutul registrului se deplasează la dreapta cu o poziție.

6.4.4. Crearea modulului principal

Pentru implementarea pe placa de dezvoltare, se va crea un modul principal în care se va instanția modulul transmițătorului serial. Semnalul de intrare *Start* al transmițătorului va fi generat prin butonul BTNU de pe placa de dezvoltare. Datele de

intrare *TxData* ale transmițătorului vor fi generate cu ajutorul unui modul VIO, care va fi creat și adăugat ulterior în proiect. Semnalul de stare *TxRdy* va fi conectat la o diodă LED a plăcii.

Executați următoarele operații pentru crearea modulului principal.

1. Creați un fișier sursă VHDL pentru modulul principal. Porturile de intrare ale acestui modul sunt *Clk*, *Rst* și *Start*, iar porturile de ieșire sunt *Tx* și *TxRdy*.
2. Adăugați la proiect fișierul sursă al unui modul pentru filtrarea oscilațiilor unui buton de pe placă (acest modul a fost utilizat la proiectul ceasului de timp real din capitolul 1) sau creați un nou modul.
3. În fișierul sursă al modulului principal, instanțiați entitatea transmițătorului serial și entitatea filtrului pentru buton, iar apoi declarați și inițializați semnalele necesare pentru conectarea acestor module.
4. Adăugați la proiect un fișier de constrângeri care a fost utilizat într-un proiect anterior și comentați liniile corespunzătoare elementelor care nu sunt necesare. Specificați conectarea portului *TxRdy* la una din diodele LED, a portului *Rst* la butonul BTND, a portului *Start* la butonul BTNU și a portului *Tx* la pinul UART_RXD_OUT.

Observație

Portul *Tx* nu trebuie conectat la pinul UART_TXD_IN.

6.4.5. Adăugarea unui modul VIO

Pentru a ilustra utilizarea unui modul VIO, se va configura un asemenea modul și se va adăuga apoi la proiect. Modulul VIO se va utiliza pentru setarea valorii octetului aplicat la intrarea *TxData* a transmițătorului prin intermediul interfeței grafice a analizorului logic Vivado, în locul conectării acestei intrări la comutatoarele plăcii de dezvoltare. Același modul se va utiliza pentru afișarea stării semnalelor *Tx* și *TxRdy* la două diode LED virtuale ale interfeței grafice a analizorului logic.

Un modul VIO poate fi adăugat la proiect doar prin metoda instanțierii în codul sursă. Parcurgeți etapele descrise în continuare pentru adăugarea unui modul VIO, conectarea semnalelor la acest modul și pentru implementarea proiectului.

1. În panoul *Flow Navigator*, secțiunea PROJECT MANAGER, selectați opțiunea **IP Catalog**.
2. În panoul *IP Catalog*, expandați secțiunea *Debug & Verification*, expandați secțiunea *Debug* și executați un clic dublu pe intrarea *VIO (Virtual Input/Output)*. Se va deschide fereastra de dialog *Customize IP*.
3. În panoul *General Options*, păstrați numele implicit **vio_0** al modulului VIO, păstrați neschimbat numărul porturilor de intrare din câmpul *Input Probe Count*

(1) și numărul porturilor de ieșire din câmpul *Output Probe Count* (1). Verificați să fie bifată opțiunea *Enable Input Probe Activity Detectors*.

4. Selectați panoul *PROBE_IN Ports(0..0)*, iar în acest panou setați la **2** dimensiunea portului de intrare PROBE_IN0. La acest port se vor conecta semnalele *Tx* și *TxRdy* pentru afișarea stării acestora.

5. Selectați panoul *PROBE_OUT Ports(0..0)*, iar în acest panou setați la **8** dimensiunea portului de ieșire PROBE_OUT0. La acest port se va conecta intrarea *TxData* a modulului transmițătorului.

6. Selectați butonul **OK** pentru generarea modulului VIO. În caseta de dialog *Create Directory*, selectați butonul **OK**. În caseta de dialog *Generate Output Products*, păstrați setările existente și selectați butonul **Generate**. Se va lansa în execuție generarea (sinteza) modulului. La terminarea sintezei modulului, selectați butonul **OK** în caseta de dialog care apare.

7. Închideți panoul *IP Catalog*. În fereastra *Sources*, selectați panoul *IP Sources* și expandați intrarea *vio_0*. Expandați apoi intrarea *Instantiation Template* și deschideți fișierul vio_0.vho în fereastra de editare.

8. Copiați șablonul de instanțiere al modulului VIO din fișierul vio_0.vho în modulul principal, după instanțierea filtrului pentru buton. Schimbați numele instanțierii componentei VIO și modificați instanțierea acestei componente într-o instanțiere a entității (eroarea indicată în linia cu instanțierea entității poate fi ignorată).

9. Declarați două semnale interne corespunzătoare porturilor de ieșire *Tx* și *TxRdy* ale modulului principal. Semnalele interne sunt necesare deoarece cele două porturi sunt de ieșire și ele nu pot fi citite. Inițializați semnalul corespunzător portului *Tx* cu '1', iar semnalul corespunzător portului *TxRdy* cu '0'.

10. În declarația de instanțiere a modulului VIO, conectați la portul `probe_in0` al acestui modul concatenarea semnalelor interne corespunzătoare porturilor *Tx* și *TxRdy* (în orice ordine), iar la portul `probe_out0` semnalul conectat la portul *TxData* al modulului transmițătorului.

Observație

Semnalele conectate la porturile modulului VIO trebuie să fie vectori. Dacă la aceste porturi trebuie conectate semnale individuale, acestea trebuie declarate ca vectori cu domeniul (`0 downto 0`). Referirea la un semnal declarat în acest fel se poate realiza prin elementul cu index 0 al vectorului. Această observație nu se referă la prezentul exemplu de proiectare, în care la portul de intrare al modulului VIO se conectează oricum un vector.

11. În declarația de instanțiere a modulului transmițătorului, conectați la porturile *Tx* și *TxRdy* semnalele interne corespunzătoare acestora.
12. La sfârșitul modulului principal, asignați la semnalele de ieșire *Tx* și *TxRdy* semnalele interne corespunzătoare acestora.
13. Salvați fișierul modulului principal. În panoul *Hierarchy* al ferestrei *Sources* verificați ca ierarhia fișierelor să fie corectă.
14. Realizați elaborarea proiectului selectând opțiunea **Open Elaborated Design** din panoul *Flow Navigator*. Corectați erorile, dacă există. După deschiderea proiectului elaborat, se afișează schema circuitului. Urmăriți conectarea modulului VIO la celelalte module, după care închideți proiectul elaborat (*File → Close Elaborated Design*).
15. Setați opțiunile pentru sinteza și implementarea proiectului. Pentru sinteză, selectați strategia de rulare *Flow_RuntimeOptimized*. Dacă în urma selecției acestei strategii parametrul *–fsm_extraction* va fi setat la *off*, modificați parametrul la **auto**. Pentru implementare, selectați strategia *Flow_Quick*.
16. Realizați sinteza și implementarea proiectului, iar apoi generați fișierul cu șirul de biți pentru configurarea circuitului FPGA.

6.4.6. Testarea funcționării proiectului

Pentru testarea funcționării transmițătorului serial, parcurgeți etapele descrise în continuare.

1. Descărcați și instalați o aplicație pentru emularea unui terminal, de exemplu, *PuTTY*, *Tera Term* sau *HyperTerminal Private Edition*.
2. Conectați o placă de dezvoltare la un port USB al calculatorului și configurați circuitul FPGA. După configurarea circuitului, în zona ferestrei editorului se deschide o fereastră a unui "tablou de bord" VIO (*VIO Dashboard*), tablou care inițial este gol, urmând ca în continuare să se adauge semnale în această fereastră (figura 6.6).
3. În fereastra *Hardware* din partea stângă, selectați modulul VIO **hw_vio_1** pentru afișarea proprietăților acestuia în fereastra *VIO Core Properties*. Dacă modulul VIO nu este vizibil în fereastra *Hardware*, executați un clic dreapta pe circuitul FPGA din această fereastră (**xc7a100t_0** pentru plăcile de dezvoltare Nexys4 DDR și Nexys A7-100T) și selectați opțiunea *Refresh Device*.
4. Selectați iconița *Add Probe(s)* + din meniul ferestrei "tabloului de bord". Se va deschide fereastra de dialog *Add Probes*, în care vor fi afișate semnalele care au fost conectate la modulul VIO. Selectați toate semnalele afișate în această fereastră, iar apoi selectați butonul **OK**.

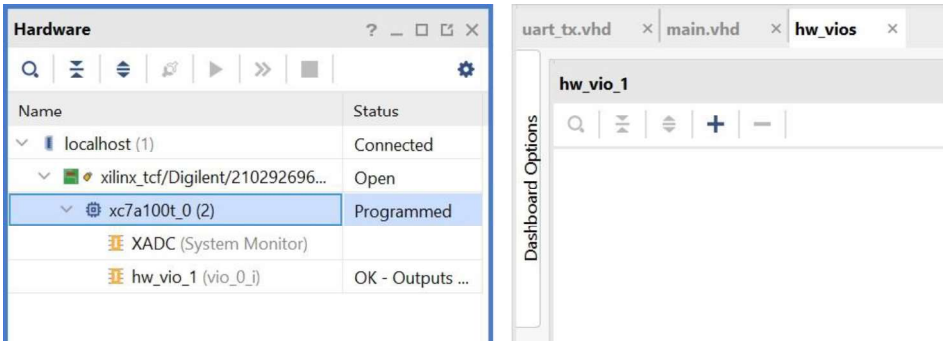

Figura 6.6. Ferestrele *Hardware* și *VIO Dashboard* după configurarea circuitului FPGA.

5. În fereastra "tabloului de bord", selectați semnalul intern care a fost declarat pentru portul *Tx*, executați un clic cu butonul din dreapta pe numele semnalului și alegeți opțiunea **LED**. În caseta de dialog *Select LED Colors*, selectați culoarea roșie în câmpul *High Value Color*, iar apoi selectați butonul **OK**. Prin aceasta, starea semnalului se afișează sub forma unei diode LED virtuale de culoare roșie atunci când semnalul are valoarea logică 1. Procedați în mod similar pentru afișarea stării semnalului intern declarat pentru portul *TxRdy*, sub forma unei diode LED virtuale de culoare verde.

6. Executați un clic cu butonul din dreapta pe vectorul *TxData*, selectați opțiunea *Toggle Button*, după care expandați vectorul *TxData* pentru a fi vizibile semnalele sale individuale. Astfel, va fi posibilă setarea valorii vectorului *TxData* aplicat la intrarea de date a transmițătorului serial într-un mod similar cu setarea prin comutatoarele plăcii de dezvoltare.

7. Setați vectorul *TxData* la o valoare corespunzătoare codului ASCII al unui caracter alfanumeric, de exemplu, 0x61 (litera 'a'). Fereastra *VIO Dashboard* ar trebui să arate similar cu fereastra din figura 6.7.

8. Lansați în execuție aplicația de emulare a unui terminal și creați o conexiune serială. De exemplu, dacă utilizați aplicația *PuTTY*, selectați opțiunea **Serial** pentru *Connection type*, în câmpul *Serial line* introduceți portul COM prin care se va realiza conectarea la placa de dezvoltare (acest port trebuie să fie diferit de COM1), iar în câmpul *Speed* introduceți **115200** (biți pe secundă) pentru viteza de comunicație. Verificați parametrii comunicației seriale selectând categoria **Serial** afișată în partea stângă. Acești parametri trebuie să fie **8** pentru numărul biților de date, **1** pentru numărul biților de STOP, **None** pentru paritate și **None** pentru controlul fluxului de date (*Flow control*). Deschideți conexiunea serială selectând butonul **Open**.

9. Apăsați butonul BTNU de pe placa de dezvoltare pentru activarea semnalului *Start* aplicat la intrarea transmițătorului serial, astfel încât acesta să transmită

caracterul aplicat la intrarea sa de date. În fereastra terminalului virtual ar trebui să se afișeze în mod corect caracterul al cărui cod ASCII s-a aplicat la intrarea de date a transmițătorului serial. Se poate observa însă că în această fereastră se afișează două caractere în loc de unul singur; al doilea caracter poate fi un spațiu.

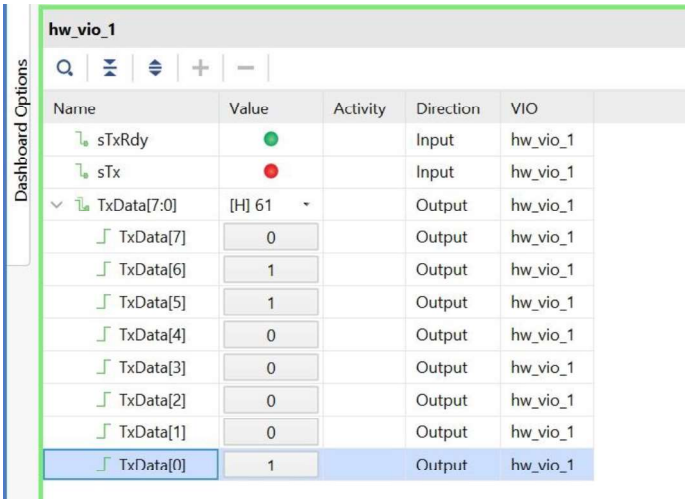

Figura 6.7. Fereastra *VIO Dashboard* după adăugarea semnalelor și configurarea modului de afișare al acestora.

10. Setați vectorul *TxData* la alte coduri ASCII ale altor caractere și verificați caracterele afișate în fereastra terminalului virtual la fiecare apăsare a butonului BTNU. Afișarea unui al doilea caracter indică o funcționare incorectă a transmițătorului serial.
11. Închideți conexiunea serială în fereastra terminalului virtual și închideți aplicația de emulare a terminalului.
12. Închideți fereastra *Hardware Manager* din mediul Vivado și opriți alimentarea plăcii de dezvoltare.

6.4.7. Adăugarea unui modul ILA

Pentru depanarea proiectului transmițătorului serial, se va adăuga un modul ILA la proiect și se vor conecta la acest modul semnalele selectate pentru a fi monitorizate în timpul funcționării. Modulul ILA va fi adăugat la proiect prin inserarea în lista de conexiuni, dar este posibilă adăugarea acestui modul și prin instanțierea în codul sursă, similar cu adăugarea modulului VIO exemplificată anterior.

Parcurgeți etapele descrise în continuare pentru adăugarea unui modul ILA la proiect și conectarea semnalelor la acest modul.

1. În fișierul sursă al modulului transmițătorului serial, modificați valoarea cu care este inițializat genericul pentru rata de biți de la 115_200 la 10_000_000. În acest fel, durata unui bit pentru frecvența semnalului de ceas de 100 MHz va fi de 10 cicluri de ceas, ceea ce va permite urmărirea mai simplă a semnalelor în timpul funcționării (pentru rata de 115.200 biți pe secundă, durata unui bit ar fi de 868 cicluri de ceas).
2. În același fișier sursă, specificați atributul `keep` pentru a evita modificarea numelor unor semnale după sinteza proiectului. Semnalele pentru care ar trebui specificat acest atribut sunt semnalul de stare *St*, contoarele *CntRate* și *CntBit*, și semnalul pentru registrul de deplasare *TSR*. Specificarea acestui atribut este exemplificată în secțiunea 6.3. Salvați fișierul sursă.
3. Realizați sinteza proiectului, fără a realiza și implementarea acestuia. După terminarea sintezei, selectați opțiunea **Open Synthesized Design** în caseta de dialog *Synthesis Completed* și apoi selectați butonul **OK** pentru a deschide proiectul sintetizat.
4. Afișați schema proiectului sintetizat cu tasta **F4** dacă nu se afișează în mod automat. Observați că anumite semnale sunt marcate deja pentru depanare cu iconița  .
5. În fereastra *Netlist*, expandați intrarea corespunzătoare modulului transmițătorului serial (*uart_tx_i*), iar apoi expandați linia *Nets*. Selectați semnalele următoare fie individual, fie mai multe în același timp, iar apoi executați un clic cu butonul din dreapta pe selecția semnalelor și selectați opțiunea *Mark Debug*: *CntBit*, *CntRate*, *St*, *TSR*, *TxData*, *Start*, *Tx*, *TxRdy* (este posibil ca numele unor semnale să nu fie identice cu acestea).

   ### Observație

   Semnalele pot fi marcate pentru depanare în mai multe moduri. De exemplu, ele pot fi marcate în fișierul sursă prin specificarea atributului `mark_debug`, sau în schema afișată după sinteza proiectului prin selectarea semnalelor dorite și specificarea opțiunii *Mark Debug* pentru acestea.

6. Afișați fereastra *Debug* selectând în meniul principal *Window* → *Debug*. În fereastra *Debug*, selectați iconița *Set Up Debug*  . Se va afișa fereastra de dialog *Set Up Debug*.
7. Selectați butonul **Next**. Se va afișa fereastra de dialog *Nets to Debug*.
8. În fereastra de dialog *Nets to Debug*, verificați dacă apar toate semnalele care au fost marcate anterior pentru depanare. Dacă lipsesc anumite semnale, acestea pot fi adăugate prin selectarea semnalelor în fereastra *Netlist* și selectarea iconiței *Add selected nets*  din fereastra de dialog *Nets to Debug*.

9. Selectați butonul **Next**. Se va afișa fereastra de dialog *ILA Core Options*.
10. În fereastra de dialog *ILA Core Options*, păstrați neschimbată valoarea implicită pentru numărul eșantioanelor de date care vor fi capturate (1024) și bifați opțiunea *Capture control*.
11. Selectați butonul **Next**. Se va afișa o pagină de sumar, indicând numărul modulelor de depanare care vor fi eliminate și create. Selectați butonul **Finish** pentru crearea comenzilor prin care se va genera modulul ILA și se vor conecta semnalele marcate pentru depanare la acest modul.
12. Creați un nou fișier sursă de constrângeri (verificați să fie selectată opțiunea *Add or create constraints* în fereastra de dialog *Add Sources*). Numele fișierului poate fi, de exemplu, debug.xdc.
13. Selectați fereastra *Sources*, executați un clic cu butonul din dreapta pe fișierul de constrângeri creat și selectați opțiunea *Set as Target Constraint File*. Prin aceasta, atunci când se vor salva constrângerile create, ele vor fi salvate în acest fișier.
14. Selectați iconița *Save Constraints*  de sub meniul principal pentru salvarea constrângerilor create. Selectați butonul **OK** în caseta de dialog *Out of Date Design*. Dacă se afișează caseta de dialog *Design Modified on Disk*, selectați butonul **Save**.
15. Deschideți fișierul de constrângeri creat anterior. Observați adăugarea unor comenzi pentru setarea proprietăților unor semnale, crearea modulului de depanare ILA, configurarea acestuia și conectarea semnalelor la porturile sale. Închideți fișierul de constrângeri.
16. Închideți proiectul sintetizat selectând *File → Close Synthesized Design* și confirmați închiderea în caseta de dialog *Confirm Close*.
17. Realizați din nou sinteza proiectului și verificați să nu apară erori.
18. Realizați implementarea proiectului și generați fișierul cu șirul de biți pentru configurarea circuitului FPGA.

6.4.8. Vizualizarea semnalelor în timpul funcționării

În continuare, se va utiliza analizorul logic Vivado pentru capturarea stării semnalelor care au fost selectate pentru monitorizare în timpul funcționării și apoi vizualizarea formei de undă a acestor semnale. Capturarea va fi declanșată de un eveniment care este relevant pentru funcționarea circuitului depanat.

Executați operațiile descrise în continuare pentru capturarea stării semnalelor și vizualizarea formei de undă a acestora.

1. Conectați o placă de dezvoltare la un port USB al calculatorului și configurați circuitul FPGA. După configurarea circuitului, se va detecta prezența unui modul ILA în circuitul FPGA configurat și se va deschide un "tablou de bord" ILA (*ILA Dashboard*). Acest tablou de bord conține mai multe ferestre în care se afișează toate informațiile de stare și de control care sunt relevante pentru un modul ILA.
2. În fereastra *Settings* (de sub fereastra *Waveform*), modul de declanșare (*Trigger mode*) este setat la BASIC_ONLY și această setare nu se poate modifica, pentru că la configurarea modulului ILA nu s-a specificat un mod de declanșare avansat. Modul de capturare (*Capture mode*) este setat la ALWAYS, ceea ce înseamnă că se va captura starea semnalelor în fiecare ciclu de ceas. Păstrați această setare pentru modul de capturare.
3. Opțiunea *Window data depth* indică numărul de eșantioane care se vor captura după îndeplinirea condiției de declanșare. Această opțiune este setată implicit la valoarea maximă de 1024 cu care s-a configurat modulul ILA. Pentru proiectul testat, se poate modifica această valoare la **256**, deoarece numărul de eșantioane va fi suficient pentru a urmări semnalele pe întreaga durată a transmiterii unui octet (durata unui bit va fi de 10 cicluri de ceas).
4. Pentru opțiunea *Trigger position in window*, modificați valoarea implicită la **10**, pentru a se afișa un număr de eșantioane și înaintea apariției evenimentului de declanșare.
5. În fereastra *Trigger Setup*, trebuie setată condiția de declanșare, care indică momentul în care va începe capturarea stării semnalelor. Mai întâi, trebuie să se adauge în această fereastră acele semnale care vor fi utilizate pentru stabilirea condiției de declanșare. Pentru proiectul testat, semnalul utilizat pentru stabilirea condiției de declanșare va fi *Start*, deoarece la activarea acestui semnal se va începe transmiterea unui octet. Pentru adăugarea semnalului *Start* în fereastra *Trigger Setup*, selectați iconița *Add probe(s)*  , selectați semnalul *Start* în fereastra afișată, iar apoi selectați butonul **OK**. Pentru semnalul adăugat se va afișa în mod implicit o valoare cu care se va compara acest semnal.

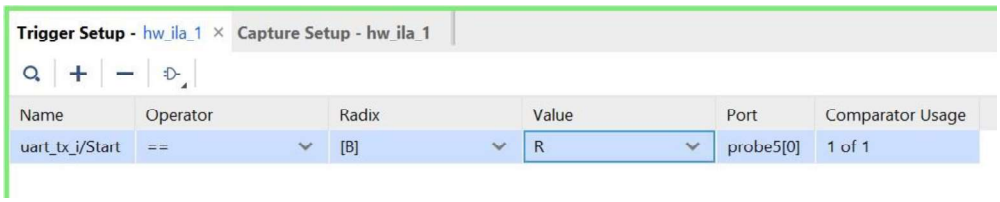

Figura 6.8. Fereastra *Trigger Setup* după setarea condiției de declanșare.

6. În aceeași fereastră *Trigger Setup*, în câmpul *Value* selectați **R (0-to-1 transition)**. Fereastra *Trigger Setup* ar trebui să arate ca în figura 6.8.

7. În fereastra *Waveform*, verificați să apară toate semnalele care au fost marcate pentru depanare.
8. Deschideți fereastra *VIO Dashboard* selectând **hw_vios** din bara aflată deasupra ferestrei *Waveform*. Fereastra conține semnalele care au fost adăugate la utilizarea anterioară a modulului VIO. În fereastra *Hardware*, verificați starea modulului *hw_vio_1*, afișată în coloana *Status*. Dacă starea afișată este *Outputs out-of-sync*, executați un clic cu butonul din dreapta pe modulul *hw_vio_1* și selectați opțiunea *Commit Output Values to VIO Core*; starea ar trebui să se modifice în *OK*. Prin aceasta, starea modulului VIO va corespunde cu starea semnalelor de ieșire afișate în interfața grafică.
9. Reveniți în fereastra *ILA Dashboard* (*hw_ila_1*), după care selectați iconița *Run trigger for this ILA core*  . Starea modulului ILA, afișată în fereastra *Status*, ar trebui să se modifice din *Idle* în *Waiting for Trigger*. Apăsați butonul BTNU de pe placa de dezvoltare pentru activarea semnalului *Start*. Se va realiza capturarea și memorarea stării semnalelor, starea modulului ILA va deveni din nou *Idle*, iar forma de undă a semnalelor capturate va fi afișată în fereastra *Waveform*.
10. Maximizați fereastra *Waveform*, selectați iconița *Zoom Fit*  și apoi iconița *Zoom In*  până când se pot distinge valorile afișate. Specificați baza binară pentru afișarea semnalului *TSR*. Se poate observa că după un ciclu de ceas de la apariția impulsului semnalului *Start*, starea liniei seriale *Tx* devine 0, ceea ce corespunde transmisiei bitului de START. Valoarea contorului de biți *CntBit* pentru acest prim bit transmis este 0. În continuare, se poate observa cum se transmit ceilalți biți ai octetului, până la bitul de STOP pentru care linia este în starea 1. Valoarea contorului de biți *CntBit* pentru bitul de STOP este 9. După transmiterea bitului de STOP, linia serială *Tx* ar trebui să rămână în starea 1 până când va începe transmisia unui alt caracter. Se poate observa însă că linia serială *Tx* trece în starea 0 după transmisia bitului de STOP (figura 6.9). Această trecere a liniei seriale *Tx* în starea 0 se va interpreta de către receptorul serial ca un bit de START pentru un nou caracter. Se explică astfel afișarea a două caractere în ecranul terminalului virtual, în locul unui singur caracter.
11. În fereastra *Waveform* observați că valoarea contorului de biți *CntBit* devine 10 ("a" în hexazecimal) după transmisia bitului de STOP și nu devine 0, cum ar fi fost corect. De aceea, în codul sursă va trebui revizuit modul în care se actualizează și se testează valoarea acestui contor.
12. În aceeași fereastră *Waveform*, măsurați durata unui bit, de exemplu, cea a bitului de STOP, pentru care contorul *CntBit* are valoarea 9. Pentru aceasta, executați un clic pe frontul crescător al semnalului *Tx*. Se va afișa marcajul de culoare galbenă în această poziție, iar în partea de sus a marcajului se va afișa valoarea 129. Selectați iconița *Add Marker*  pentru a adăuga un al doilea marcaj; acesta

se va afișa inițial în aceeași poziție cu primul marcaj. Mutați al doilea marcaj în poziția frontului descrescător al semnalului *Tx* cu ajutorul dreptunghiului de culoare albastră din partea de sus a marcajului; valoarea afișată în acest dreptunghi ar trebui să fie acum 142. Distanța dintre cele două marcaje este de 13 eșantioane (distanță indicată prin –13 în partea de jos a ferestrei), ceea ce înseamnă 13 cicluri de ceas. Durata corectă a unui bit, cu genericul setat la valoarea 10_000_000, ar trebui să fie de 10 cicluri de ceas. Aceasta reprezintă o a doua problemă, astfel încât codul sursă ar trebui revizuit și pentru a corecta această problemă (chiar dacă, la setarea genericului cu valoarea reală pentru rata de biți de 115.200, atunci când durata unui bit este de 868 cicluri de ceas, nu va avea o importanță prea mare dacă durata unui bit va fi cu 3 cicluri de ceas mai mare).

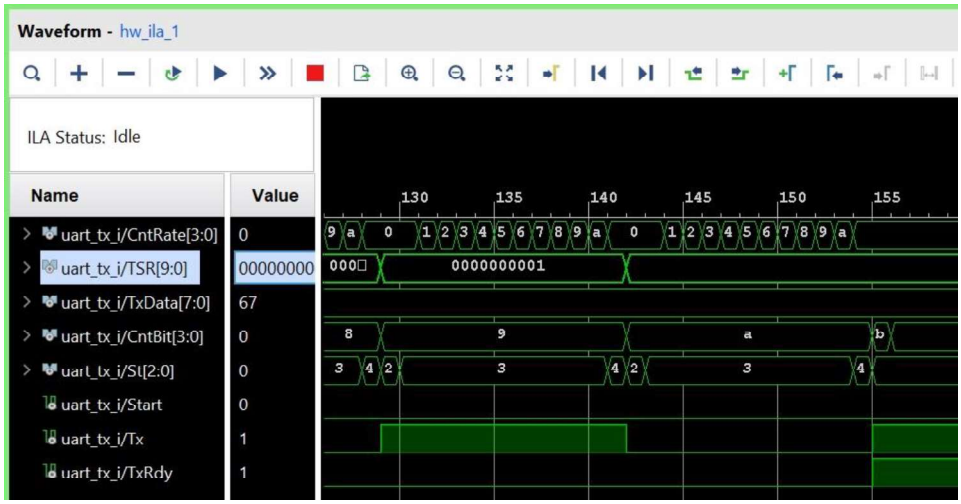

Figura 6.9. Fereastra *Waveform* indicând transmisia unui bit suplimentar de 0 după bitul de STOP.

13. Închideți fereastra *Hardware Manager* și deconectați placa de dezvoltare.

6.4.9. Corectarea erorilor și verificarea funcționării

Pentru corectarea celor două probleme detectate, trebuie revizuit automatul de stare din modulul transmițătorului serial. Prima problemă este legată de modul de actualizare al contorului de biți *CntBit*. Acest contor este incrementat în starea `shift` și este testat dacă a ajuns la valoarea maximă în aceeași stare. Deoarece contorul *CntBit* este un semnal, incrementarea valorii contorului nu se realizează imediat, ci la sfârșitul procesului. De aceea, se va testa, de fapt, valoarea anterioară a contorului, cea asignată la trecerea precedentă prin starea `shift`, iar o nouă testare a valorii contorului se va realiza doar la următoarea trecere prin starea `shift`, după ce s-a mai transmis un bit. Astfel se explică transmisia unui bit suplimentar.

Pentru corectarea acestei probleme, se poate muta incrementarea contorului *CntBit* într-o stare precedentă stării în care se testează valoarea contorului. De exemplu, incrementarea se poate realiza în starea `send`, care este parcursă o singură dată pentru fiecare bit transmis. O altă posibilitate este să se utilizeze o variabilă în locul unui semnal, actualizarea variabilei fiind realizată imediat și nu la sfârșitul procesului.

A doua problemă constă în durata mai mare cu trei cicluri de ceas a unui bit. Durata unui bit este controlată de contorul *CntRate*. Pentru acest contor se utilizează tot un semnal, iar acest semnal este incrementat și testat în aceeași stare, ca și semnalul utilizat pentru contorul *CntBit*. Aceasta explică însă doar creșterea cu un ciclu de ceas a duratei unui bit. Cele două cicluri de ceas suplimentare provin de la faptul că se mai parcurg încă două stări, `send` și `shift`, la transmisia fiecărui bit, fiecare cu durata unui ciclu de ceas.

Pentru a corecta a doua problemă, nu se poate muta incrementarea contorului *CntRate* într-o stare precedentă fără să se introducă o stare suplimentară. Se poate însă ține cont de diferența de trei cicluri de ceas, comparând valoarea contorului cu constanta `T_BIT-3` în loc de `T_BIT`.

Pentru corectarea erorilor și verificarea funcționării proiectului, parcurgeți etapele descrise în continuare.

1. În fișierul sursă al modulului transmițătorului serial, mutați incrementarea contorului *CntBit* din starea `shift` în starea `send`.
2. În starea `waitbit`, modificați valoarea cu care se compară contorul *CntRate* din `T_BIT` în `TBIT-3`.
3. În declarația entității, modificați valoarea cu care se inițializează genericul pentru rata de biți la 115_200 și salvați fișierul sursă.
4. Eliminați din proiect fișierul de constrângeri debug.xdc.
5. Realizați sinteza și implementarea proiectului și generați fișierul cu șirul de biți pentru configurarea circuitului FPGA.
6. Conectați o placă de dezvoltare la un port USB al calculatorului și configurați circuitul FPGA. După configurarea circuitului, ar trebui să se deschidă fereastra *VIO Dashboard*. Verificați ca vectorul *TxData* să fie inițializat cu codul ASCII al unui caracter alfanumeric.
7. În fereastra *Hardware*, executați un clic cu butonul din dreapta pe modulul *hw_vio_1* și selectați opțiunea *Commit Output Values to VIO Core*. Starea modulului VIO ar trebui să devină OK.
8. Lansați în execuție aplicația de emulare a unui terminal, creați o conexiune serială cu aceiași parametri ca și cei setați în secțiunea 6.4.6, iar apoi deschideți conexiunea serială.

9. Apăsați butonul BTNU de pe placa de dezvoltare. În fereastra terminalului virtual ar trebui să se afișeze doar caracterul cu codul ASCII setat în fereastra *VIO Dashboard*.
10. Setați vectorul *TxData* la alte coduri ASCII și verificați caracterele afișate în fereastra terminalului virtual la fiecare apăsare a butonului BTNU.
11. Închideți conexiunea serială și închideți fereastra aplicației de emulare a unui terminal. Închideți fereastra *Hardware Manager* din mediul Vivado, opriți alimentarea plăcii și deconectați placa de la calculator.

Aplicații

6.1 Răspundeți la următoarele întrebări:

a. Care sunt avantajele utilizării analizorului logic Vivado față de utilizarea unui analizor logic tradițional pentru depanarea proiectelor implementate în circuite FPGA?

b. Care sunt funcțiile modulelor ILA și VIO?

c. Ce reprezintă o condiție de declanșare?

d. Ce reprezintă o condiție de calificare a memorării?

6.2 Parcurgeți etapele descrise în secțiunea 6.4 pentru proiectarea și depanarea transmițătorului serial UART.

6.3 Creați un nou proiect pentru un modul `uart_send16` care transmite pe interfața serială un șir de 16 caractere în cod ASCII. Intrările modulului sunt semnalul de ceas *Clk*, semnalul de resetare sincronă *Rst*, vectorii *Data1* și *Data2* de câte 64 de biți conținând codurile ASCII ale caracterelor care trebuie transmise, și semnalul *Send*, care indică începerea transmisiei șirului de caractere de la intrarea *Data1* și apoi de la intrarea *Data2*. Ieșirile modulului sunt linia serială *Tx* pe care sunt transmise caracterele și semnalul *Rdy*, care indică prin starea sa activă terminarea transmisiei șirului de caractere. Caracterele trebuie transmise începând cu caracterul din octetul cel mai semnificativ al vectorului *Data1*. După transmisia șirului de caractere, modulul va transmite încă două caractere, CR (*Carriage Return*, cu codul x"0D") și LF (*Line Feed*, cu codul x"0A"). Pentru transmisia șirului de caractere, utilizați modulul transmițătorului serial UART care a fost proiectat și depanat în secțiunea 6.4. Scrieți un automat de stare ca un proces secvențial; pentru fiecare caracter din șir, automatul va activa semnalul *Start* pentru modulul transmițătorului serial și va aștepta terminarea transmisiei caracterului. Utilizați un semnal pentru contorizarea caracterelor transmise. Pentru aplicarea la intrarea *TxData* a transmițătorului serial a codului caracterului care trebuie transmis, utilizați o instrucțiune concurentă de asignare condițională

`when ... else` sau de asignare selectivă `with ... select` (echivalentă cu un multiplexor) în funcție de valoarea contorului din automatul de stare.

6.4 Completați proiectul creat pentru aplicația 6.3 cu un modul principal pentru testarea pe o placă de dezvoltare a modulului care transmite un șir de caractere pe interfața serială. Modulul va transmite pe interfața serială un text de 16 caractere atunci când se apasă un anumit buton de pe placa de dezvoltare și un alt text de 16 caractere atunci când se apasă un alt buton. Porturile de intrare ale modulului principal sunt *Clk*, *Rst* și intrările de la butoane, iar porturile de ieșire sunt linia serială *Tx* și semnalul de stare *Rdy* de la modulul care transmite un șir de caractere. Modulul principal va aplica la intrările *Data1* și *Data2* ale modulului testat vectorii corespunzători șirului de caractere care trebuie transmis și va activa semnalul *Send* dacă unul din cele două butoane este apăsat și dacă semnalul de stare *Rdy* indică terminarea unei transmisii inițiate anterior. Definiți cele două șiruri de caractere care trebuie transmise ca și constante de tip **`STRING`**. Pentru conversia șirurilor de caractere în vectori conținând codurile ASCII ale caracterelor, utilizați funcția `S8_TOASCII` definită în pachetul RISC_pkg din Anexa D. Utilizați două module pentru filtrarea oscilațiilor butoanelor de pe placa de dezvoltare. Verificați funcționarea proiectului pe placa de dezvoltare cu ajutorul unei aplicații de emulare a unui terminal.

6.5 Completați modulul principal din proiectul creat pentru aplicația 6.4 astfel încât acesta să transmită un șir de caractere conținând codul hexazecimal corespunzător stării comutatoarelor SW de pe placa de dezvoltare Nexys4 DDR sau Nexys A7-T100 atunci când se apasă un al treilea buton de pe placă. Pentru conversia vectorului de 16 biți conținând starea comutatoarelor într-un vector de 32 biți cu codurile ASCII corespunzătoare, utilizați funcția de conversie `B2_TOASCII` definită în pachetul RISC_pkg din Anexa D. Verificați funcționarea proiectului pe placa de dezvoltare cu ajutorul unei aplicații de emulare a unui terminal.

6.6 Pe baza modulului uart_send16 creat pentru aplicația 6.3, creați un nou proiect pentru un modul uart_send64 care transmite pe interfața serială un șir de 64 caractere în cod ASCII. Acest modul are aceleași intrări și ieșiri ca și modulul uart_send16, dar are ca intrări suplimentare vectorii *Data3*, *Data4*, *Data5*, *Data6*, *Data7* și *Data8* de câte 64 de biți conținând codurile ASCII ale celorlalte caractere care trebuie transmise. După transmisia șirului de 64 caractere, modulul va transmite caracterele CR și LF. Modulul uart_send64 va fi utilizat pentru implementarea pe o placă de dezvoltare a procesorului RISC din capitolul 8.