UNIVERSITATEA "POLITEHNICĂ" TIMIŞOARA
FACULTATEA DE ELECTRONICĂ ŞI TELECOMUNICAŢII
DEPARTAMENTUL DE ELECTRONICĂ APLICATĂ

Ing. PETRU DEMIAN

OPTIMIZAREA METODELOR DE TESTARE A ECHIPAMENTELOR ELECTRONICE NUMERICE

TEZĂ DE DOCTORAT

CONDUCĂTOR ŞTIINŢIFIC: Prof. Dr. Ing. TIBERIU MUREŞAN

TIMIŞOARA
2004

Cuprins

INTRODUCERE

Odată cu creşterea complexităţii circuitelor, echipamentelor şi a sistemelor electronice, testarea acestora a devenit tot mai greoaie şi mai mare consumatoare de timp, deci tot mai scumpă, ajungând să reprezinte circa 15% din preţul unui produs. În aceste condiţii, testarea utilizând JTAG şi testarea cu echipamente electronice configurabile au câştigat o pondere majoră la testarea circuitelor integrate, a plăcilor electronice şi a sistemelor electronice complexe, atât a celor utilizate în condiţii de laborator, cât şi în producţia de echipamente electronice în serii medii sau mari.

Din punct de vedere istoric testarea pe frontieră a devenit o necesitate începând cu anii '80, între 1985-1990 apărând unele standardizări în acest domeniu, iar în 1990 firme de prestigiu au standardizat metoda devenită IEEE 1149.1. Ulterior au mai apărut extensii ale standardelor, precum 1149.1a (1993), 1149.1b (1994), iar în 1995 a fost aprobat 1149.5, ceea ce a permis extinderea gradului de acoperire a testării prin JTAG.

În anii '90, caracterizaţi prin dezvoltarea spectaculoasă a tehnicilor de integrare pe scară largă a componentelor electronice, se remarcă dezvoltarea şi aplicarea practică a testării JTAG însoţită de dezvoltarea limbajelor de programare, pentru a creşte eficienţa testării şi pentru a permite o testare cu un grad cât avandat de automotizare. Astfel a apărut o nouă generaţie de echipamente de testare mult mai flexibile şi mai ieftine decât echipamentele predecesoare acestea mult mai bine adaptate la necesităţile din acel moment, iar complexitatea softului a crescut continuu ajungând treptat mai mare decât hardware-ul utilizat. Dificultăţile tehnice de realizare a unor echipamente de testare performante precum şi tendinţa de globalizare au determinat dispariţia firmelor incapabile să facă trecerea rapidă la o nouă generaţie de echipamente de testare rapide şi eficiente,

constatându-se că testabilitatea unui produs trebuie abordată cu seriozitate încă din fazele inițiale de dezvoltare ale acestuia.

O problemă specifică domeniului abordat în cadrul tezei este existența a numeroase surse bibliografice orientate preponderent spre aspectul aplicativ şi publicitar, deci adresate în special utilizatorului de asemenea sisteme. În aceste condiții, cercetătorul din acest domeniu este confruntat cu o lipsă acută de literatură care să aprofundeze aspectele teoretice fundamentale, să ofere informații funcționale detaliate şi să descrie metodele de analiză şi modelare, motiv pentru care autorul şi-a propus să găsească şi să analizeze cât mai multe surse bibliografice existente pe plan internațional, să sistematizeze şi să selecteze cele mai importante activități de cercetare-dezvoltare din domeniu, să verifice aspectele teoretice prin implementări practice şi să le pună la dispoziția celor interesați.

Ca urmare a acestei strategii teza se caracterizează prin următoarele obiective majore:

- asigură o bază teoretică referitoare la echipamentele de testare şi oferă detalii aferente testării utilizând JTAG;
- prezintă realizările autorului, caracterizate prin implementarea unor metode de testare, modelare şi generare automată a vectorilor de test, majoritatea acestora fiind implementate de autor pentru firme internaționale de prestigiu.

Capitolul 1 cuprinde o sistematizare a domeniului abordat cu o trecere treptată de la general la particular până la obiectivul concret al tezei de doctorat. Autorul consideră această prezentare necesară pentru cunoaşterea stadiului actual de dezvoltare a echipamentelor de testare, a problematicii şi constrângerilor specifice, precum şi a clasificării principalelor grupe de echipamente de testare. În finalul capitolului autorul defineşte direcțiile fundamentale de perfecționare constând în aprofundarea şi aplicarea testării JTAG la nivel de circuit, echipament şi sistem electronic.

Capitolul 2 conține o analiză succintă a principiilor şi metodelor aferente testării JTAG, în strânsă legătură cu obiectivele pe care autorul şi le-a propus în cercetarea posibilităților de optimizare a testării echipamentelor electronice. Autorul a introdus această analiză deoarece în literatura de specialitate se găsesc de obicei doar informații generale, iar detaliile tehnice sau exemplificările practice sunt rare.

Capitolul 3 prezintă posibilitățile de aplicare a metodelor de testare cu JTAG la nivel de placă sau sistem electronic, insistându-se pe metodele folosite la testarea unei unități de aviație Boeing 777 de control a încărcăturii. Din analizele şi implementările efectuate de

autor a rezultat că testarea pe frontieră este aplicabilă şi chiar foarte convenabilă şi la testarea sistemelor electronice complexe a căror siguranţă în funcţionare este esenţială. De asemenea, din analiza comparativă a testării JTAG şi a testării cu echipamente configurabile a rezultat că testarea JTAG este în general mai convenabilă, dar este necesar ca circuitul, placa sau sistemul electronic testat, să fie proiectat avându-se în vedere considerentele de testare.

Capitolul 4 conţine o analiză a principiilor de conectare a circuitelor JTAG insistându-se asupra performanţelor oferite de conectarea serie, stea şi ierarhică şi a recomandărilor de utilizare a acestora. Autorul a considerat necesară introducerea acestui capitol deoarece în literatura de specialitate, de obicei apare doar prezentarea sumară a conexiunii stea şi serie, fără o comparare a acestora şi fără o analiză matematică a eficienţei transmiterii vectorilor de test în cazul diferitelor conectări.

În capitolul 5 se analizează posibilităţile de modelare în VHDL a circuitelor, plăcilor şi sistemelor electronice, se prezintă modelarea în VHDL a părţii hardware JTAG şi se abordează problema generării automate a vectorilor de test pentru testarea JTAG pe baza modelării în VHDL. Autorul consideră că modelarea şi simularea în VHDL a circuitelor, echipamentelor şi sistemelor electronice au şanse reale de dezvoltare, existând deja o serie de programe complexe şi o tradiţie în acest domeniu, iar următoarea tendinţă va fi generarea automată a vectorilor de test pentru testarea reală.

Se poate aprecia că această teză de doctorat este o încercare de abordare multilaterală a domeniului testării echipamentelor electronice complexe, începând cu sistematizarea materialului bibliografic, completarea lacunelor existente în descrierea şi analizarea acestui domeniu, verificarea ipotezelor teoretice, prezentarea realizărilor practice ale autorului şi menţionarea domeniilor în dezvoltare şi care se mai pot aprofunda.

Lucrarea se caracterizează pe tot parcursul ei prin:

- sinteza unui vast volum de informaţii obţinut din multiple surse din literatura de specialitate şi de pe Internet, precum şi din diverse implementări practice personale ale autorului;
- numeroase clasificări, prelucrări şi analize ale rezultatelor practice;
- contribuţii originale ale autorului constând în expunerea rezultatelor cercetărilor personale şi a propriilor opinii.

Această teză de doctorat a fost elaborată sub atenta îndrumare a domnului prof. dr. ing. Tiberiu Mureşan, căruia autorul doreşte să îi mulţumească în mod deosebit pentru

coordonare, pentru sugestiile şi sfaturile deosebit de utile, pentru sprijinul acordat şi nu în ultimul rând pentru înțelegerea de care a dat dovadă.

Autorul îşi exprimă gratitudinea faţă de cadrele didactice ale Universităţii Politehnice din Timişoara, care au contribuit decisiv la formarea sa profesională, atât în timpul studenției, cât şi în timpul activității de cadru didactic la Facultatea de Electronică şi Telecomunicații, Catedra de Electronică Aplicată.

De asemenea, autorul rămâne recunoscător firmei Timteh Electronic din Timişoara şi în special domnului ing. Dan Simu pentru posibilitățile oferite de aplicare practică a conceptelor teoretice şi de perfecționare a activității în domeniul testării.

Nu în ultimul rând, autorul doreşte să mulțumească soției pentru înțelegerea şi sprijinul acordat pe parcursul elaborării tezei, activitate efectuată în afara atribuțiilor de serviciu într-un timp liber extrem de limitat.

CAPITOLUL 1

1. STADIUL ACTUAL AL TESTĂRII ECHIPAMENTELOR ELECTRONICE COMPLEXE

Parcurgând literatura de specialitate, apărută pe plan mondial în ultimii 10 ani, autorul a întâlnit diverse metode de testare a echipamentelor electronice, începând de la echipamentele clasice de testare cu ace, utilizate în producție şi sfârşind cu o serie de software-uri de analiză şi optimizare a testabilității echipamentelor electronice. În aceste condiții, s-a considerat necesar ca în acest prim capitol să se facă o analiză şi o clasificare generală a echipamentelor de testare utilizate în prezent, urmate de o prezentare a tendințelor generale, după care să se analizeze în detaliu metodele de testare, care prezintă potențial. Scopurile principale ale acestui capitol sunt: identificarea metodei sau a metodelor, care prezintă interes pentru testarea echipamentelor numerice complexe, localizarea aspectelor care merită să fie prezentate şi aprofundarea unora dintre acestea prin contribuții originale ale autorului.

1.1. Analiza stadiului actual de dezvoltare al echipamentelor de testare

Ca urmare a studiului literaturii de specialitate autorul a constatat că se folosesc următoarele categorii de echipamente:

- echipamente de testare *specializate*, utilizate de obicei pe liniile de producție ale echipamentelor electronice;
- echipamente de testare *configurabile*, bazate pe soluții standard şi modulare, utilizate în special la dezvoltarea produsului;
- echipamente de testare *bazate pe JTAG*, utilizate atât pe liniile de producție, la dezvoltarea produsului, cât şi la autotestarea produsului electronic.

Echipamentele de testare specializate ([40], [43], [44], [59], [66], [67], [68], [69], [72], [137], [145], [151], [181], [191], [193], [195], [196], [201], [203], [204], [205], [210], [221]): prezintă următoarele caracteristici principale:

- sunt foarte scumpe (> 100.000 $);
- sunt construite de firme de prestigiu în domeniu (Agilent, Teradyne, GenRad, Tektronix, Siemens);
- sunt foarte specializate şi cu un domeniu foarte strict de utilizare;
- de regulă există o firmă specializată, care asigură suportul acestor echipamente;
- reconfigurarea echipamentului pentru testarea unui alt produs este de regulă o operaţie complicată care nu se execută foarte des;
- de obicei pentru realizarea conexiunilor cu echipamentul testat se utilizează o matrice de ace, iar aplicarea vectorilor de test se face cu ajutorul unui echipament specializat;
- tot mai frecvent se realizează şi o inspecţie vizuală a conexiunilor sau testarea vizuală a unor poziţionări mecanice;
- permit şi programarea memoriilor Flash şi EEPROM a echipamentului testat pe linia de producţie (opţiune necesară tot mai frecvent);
- sunt utilizate numai la testarea echipamentelor electronice produse în serie mare şi foarte mare;
- conţin de regulă un echipament hard specializat, cel puţin un calculator relativ puternic şi software dedicat;
- dacă echipamentul testat conţine unul sau mai multe microcontrolere frecvent pe linia de producţie se încarcă şi execută în echipamentul testat un program de test sau diagnoză HW;
- întrucât siguranţa în funcţionare şi capacitatea de producţie a liniei de producţie sunt foarte importante este necesar ca echipamentele de testare utilizate să prezinte o fiabilitate ridicată şi cu timp de testare redus. Dacă totuşi nu este posibilă reducerea timpului de testare pe produs sub limite rezonabile se practică testarea în paralel a mai multor produse.

În categoria *echipamentelor de testare configurabile* ([39], [41], [43], [162], [165], [167], [172], [244], [247], [250], [252], [253]) intră de exemplu instrumentele hardware şi software produse de o serie de firme de prestigiu (National Instrument, Corelis sau Vector) prezentând următoarele caracteristici generale:

- sunt soluţii hard şi soft modulare uşor interconectabile şi extensibile;

- sunt disponibile şi programe de lucru integrate (de regulă soft pe PC), care asigură necesitățile utilizatorului, inclusiv suportul pentru testarea automată;
- prețul soluțiilor oferite nu este foarte scăzut, fiind de ordinul miilor sau zecilor de mii de dolari, dar cu cel puțin un ordin de mărime mai redus decât acela al echipamentelor de testare specializate utilizate la producție;
- se utilizează preponderent la testarea în faza de dezvoltare a produsului, fiind echipamente standard în laboratoarele firmelor medii şi mari.
- frecvent firmele producătoare de astfel de echipamente implementează facilități suplimentare ca urmare a cerințelor exprese ale utilizatorilor;
- frecvent echipamentele de testare configurabile se utilizează la dezvoltare întrucât se pot relativ uşor configura pentru necesitățile concrete de proiect iar costurile sunt acceptabile;
- deseori prin intermediul unor echipamente configurabile (HW şi software) se simulează şi mediul în care lucrează echipamentul testat reducându-se semnificativ costurile mediului de test;
- există o serie de magistrale suportate, cele mai importante fiind:
    - IEEE 488 (GPIB);
    - IEEE 485;
    - CAN;
    - IEEE 1149.1;
    - RS232;

Echipamentele de testare JTAG ([10], [38], [39], [42], [44], [45], [54], [58], [86], [139], [97], [98], [197], [237], [241], [243], [246]) au apărut după adoptarea standardului IEEE 1149.1 (în 1990) şi prezintă următoarele caracteristici:

- testarea se face prin utilizarea magistralei JTAG (o magistrală ieftină, sincronă cu 4 sau 5 fire);
- testarea se poate face la nivel de circuit, placă electronică sau sistem, în principiu cu acelaşi echipament;
- metoda este mai utilizată decât pare, multe dintre emulatoarele şi programatoarele existente pe piață folosesc magistrale şi facilități JTAG;
- prețul echipamentelor bazate pe JTAG şi a circuitelor cu facilități JTAG nu este prea ridicat (de exemplu, un emulator bazat pe JTAG costă in jur de 2.000 $, adică de 10 ori mai puțin decât un emulator activ);

- treptat s-au dezvoltat şi o serie de limbaje de programare şi au apărut o serie de programe utilizate la testarea JTAG. Chiar mai mult, limbajele BSDL şi HSDL utilizate în testarea JTAG sunt apropiate de limbajul VHDL (utilizat la proiectarea circuitelor electronice) şi ca atare testarea JTAG a pătruns adânc şi în proiectarea circuitelor numerice;
- există o serie de companii de prestigiu care produc echipamente de testare JTAG ([242], [245], [250]);
- există şi o serie de soft-uri care permit automatizarea testării generând automat vectorii de test pe baza modelelor în BSDL a circuitelor sau sistemelor [44], [203];
- există tendinţe de a extinde standardul IEEE 1149.1 pentru testarea părţii analogice şi la nivel de sistem (IEEE 1149.4, IEEE 1149.5 şi IEEE 1149.6) [31] .

1.2. Tendinţe generale în domeniul echipamentelor de testare

În urma studierii literaturii de specialitate, autorul a constatat următoarele tendinţe importante:

- echipamentele de testare cu ace au atins limite tehnologice datorită miniaturizării şi creşterii densităţii de integrare a componentelor electronice, iar testarea utilizând JTAG devine o alternativă convenabilă pentru tot mai multe firme de prestigiu ([7], [65], [93], [100], [132], [209]);
- testarea devine tot mai complexă şi mai scumpă, iar firmele care nu fac progrese importante în îmbunătăţirea metodei de testare utilizate vor da faliment;
- tot mai multe firme proiectează echipamentele electronice complexe ţinând seama de principiile de testabilitate încă de la începutul ciclului de dezvoltare al produsului;
- au apărut o serie de unelte HW şi SW pentru uşurarea testării la dezvoltarea produsului şi în producţie, iar multe dintre acestea devin soluţii standard;
- există tendinţa generală de a adăuga în circuitele şi în sistemele electronice hardware suplimentar pentru a îmbunătăţi testabilitatea produsului;
- se scrie relativ mult despre testarea automată, dar relativ rar testarea automată este complet realizabilă;
- se constată o creştere a importanţei softului utilizat la testare, a complexităţii acestuia precum şi a posibilităţii de conectare a diferitor module software;

- piața echipamentelor de testare este o piață în plină dezvoltare existând tendințe de standardizare şi interconectare uşoară a modulelor;
- exista tendința de utilizare mai frecventă a standardelor JTAG 1149.1, 1149.5, 1149.4 şi 1149.2;
- metoda JTAG este folosită la testarea circuitelor integrate, la testarea plăcii electronice şi pentru testarea la nivel de sistem electronic.

Analizând literatura de specialitate, autorul a constatat că testarea clasică cu ace este încă frecvent utilizată, dar există tot mai multe probleme şi limite de accesibilitate la testarea echipamentelor electronice complexe.

Din experiența proprie şi din analiza bibliografică autorul a ajuns la concluzia că utilizarea echipamentelor de testare configurabile este o soluție potrivită la dezvoltarea echipamentelor electronice sau pentru testarea în producție în serie mică, dar nu este foarte potrivită pentru testarea în producție datorită timpului de testare relativ mare. Din studiul bibliografic şi practic, autorul a constatat că, de regulă, la echipamentele de testare configurabile timpul de testare şi eficiența testării depind foarte mult de implementare, putând exista diferențe foarte mari între implementări. Autorul a mai observat că în situația utilizării echipamentelor de testare configurabile, de cele mai multe ori se insistă pe implementarea testării şi nu pe îmbunătățirea sau implementarea metodei optime de test. Se impune a se sublinia că utilizarea echipamentelor de testare configurabile are şanse reale de extindere, iar o exemplificare practică este prezentată în capitolul 3.

Din studiul bibliografic autorul a mai constatat că metoda de testare IEEE 1149.1 se utilizează la nivel de circuit integrat, placă electronică sau sistem electronic, costul echipamentului de testare JTAG fiind aproximativ cu un ordin de mărime mai mic decât costurile echipamentelor de testare configurabile şi aproximativ cu două ordine de mărime mai mic decât prețurile echipamentelor de testare clasice. Timpul de testare în cazul utilizării echipamentelor JTAG este frecvent comparabil cu timpul de testare de la utilizarea metodelor clasice, fiind foarte dependent de sistemul sau echipamentul electronic testat.

În aceste condiții, după părerea autorului, metoda de testare JTAG are şanse reale de extindere, atât la testarea în procesul de producție, cât şi la testarea pentru dezvoltarea produsului, ea putând fi utilizată la nivel de circuit, placă sau produs.

1.3. Direcții de aprofundare

Pe baza analizelor efectuate în secțiunile precedente autorul a constat unele direcții care merită aprofundate:

- *direcții de aprofundare la testarea cu echipamente configurabile:*
  - analiza soluțiilor tehnice utilizate la testare (mediu de testare utilizat, interfețe utilizate, cote din piață ale diferitelor soluții tehnice şi identificarea principalelor companii producătoare de astfel de echipamente);
  - analiza softurilor utilizate (medii soft disponibile, posibilitățile de import sau export în alte medii, facilitățile şi limitele softurilor existente pe piață);
  - analiza compatibilității (analiza compatibilității HW şi SW a echipamentelor de testare şi analiza posibilității de a înlănțui mai multe produse HW şi SW pentru realizarea testării);
  - propuneri de îmbunătățire a metodelor de testare HW şi SW cu echipamente configurabile;
  - analiza costurilor şi a performanțelor de testare.
- *direcții de aprofundare la testarea cu JTAG:*
  - sinteză bibliografică şi analiza noilor direcții de dezvoltare;
  - analiza posibilităților de testare cu JTAG, a domeniului de utilizare, a limitărilor existente şi a performanțelor testării;
  - analiza performanțelor la testarea JTAG şi propunerea unor metode şi soluții superioare;
  - analiza posibilității de utilizare a testării JTAG la nivel de circuit integrat, echipament electronic şi sistem electronic complex;
  - abordarea ierarhică a testării JTAG;
  - proiectarea circuitelor integrate şi a sistemelor numerice din considerente de testare;
  - generarea automată a vectorilor de test pentru testarea cu JTAG pe baza modelării în VHDL a circuitului, a echipamentului electronic sau a sistemului electronic.

Ca urmare a sintezei efectuate în acest capitol şi a activităților practice efectuate, autorul a constatat că testarea JTAG prezintă un potențial de dezvoltare mai mare, existând şi unele breşe insuficient aprofundate.

Având în vedere argumentele menționate anterior, autorul va aprofunda în cadrul tezei testarea JTAG, dar atunci când este necesar va face referiri la testarea clasică şi la testarea cu echipamente configurabile.

1.4. Concluzii

În cadrul acestui capitol introductiv autorul a făcut o analiză a echipamentelor utilizate în prezent pe plan mondial la testarea plăcilor şi sistemelor electronice complexe şi a propus o clasificare a principalelor grupe de echipamente de testare.

Din sinteza bibliografică şi din vizitele efectuate de autor în secțiile de producție de la Siemens VDO(din oraşele Babenhausen şi Karben) şi Ellf (Lugoj), autorul a constatat că echipamentele de testare dedicate producției sunt de obicei echipamente speciale adaptate pentru un anumit produs. Pe plan mondial există doar câteva firme producătoare de astfel de echipamente, acestea fiind în general foarte scumpe, iar în cazul testării unor plăci electronice complexe au apărut limitări tehnologice. Utilizarea echipamentelor de testare dedicate producției este justificată doar la producția în serie mare sau foarte mare (minim mii de teste pe zi) şi prezintă în general o flexibilitate foarte redusă.

Din experiența personală (activitate de proiectare şi dezvoltare în cadrul firmelor Siemens VDO şi Timteh-SRL) şi din bibliografia studiată, autorul a constatat că echipamentele de testare configurabile sunt foarte diverse, existând deja o serie de firme producătoare importante, fiind posibil ca segmentul de piață să se extindă în defavoarea echipamentelor de testare clasice.

Ca urmare a analizei bibliografice şi a implementărilor practice efectuate la firma Timteh, autorul a constatat că echipamentele de testare bazate pe JTAG au apărut doar după 1990 şi sunt utilizate tot mai frecvent, inclusiv în producție. În acest prim capitol, autorul doreşte să sublinieze că există deja tendința de a considera testarea bazată pe JTAG ca o soluție standard, iar tot mai des echipamentele electronice se proiectează ținând cont de considerentele de testare JTAG.

Din studiul bibliografic s-a constatat că testarea utilizând JTAG are şanse reale de dezvoltare, existând multe breşe insuficient aprofundate, în care se pot obține rezultate importante, atât teoretice, cât şi practice. În consecință, autorul şi-a propus să aprofundeze testarea JTAG, să insiste asupra metodei şi posibilităților de extindere a acesteia, să o aplice la testarea unor echipamente electronice şi la nivelul proiectării

circuitelor electronice, iar în final să o utilizeze la nivelul testării ierarhice a unui sistem electronic complex.

Ca urmare a analizei efectuate au rezultat următoarele direcții de aprofundare pe care autorul le va urmări în cadrul tezei:

- sinteză bibliografică şi analiza noilor direcții de dezvoltare la testarea JTAG şi la testarea cu echipamente configurabile;
- analiza posibilităților de testare cu JTAG, a domeniului de utilizare, a limitărilor existente şi a performanțelor testării;
- sinteza limbajelor utilizate la testarea IEEE 1149.1 şi exemplificarea practică a acestora în cazul unor proiecte reale;
- analiza performanțelor la testarea JTAG şi propunerea unor metode şi soluții superioare celor utilizate în prezent;
- analiza posibilităților de testare cu ajutorul echipamentelor de testare configurabile;
- compararea testării cu echipamente de testare configurabile şi a testării JTAG;
- analiza posibilității de utilizare a testării JTAG la nivel de circuit integrat, echipament electronic şi sistem electronic complex;
- abordarea ierarhică a testării JTAG şi analiza timpilor de testare;
- analiza proiectării circuitelor integrate şi a sistemelor numerice avându-se în vedere considerentele de testare;
- generarea automată a vectorilor de test pentru testarea cu JTAG pe baza modelării în VHDL a circuitului integrat, a echipamentului electronic sau a sistemului electronic.

În următorul capitol autorul îşi propune să realizeze o analiză succintă a principiilor şi metodelor aferente testării JTAG, să prezinte dezvoltările mai recente ale standardului IEEE 1149.1 (IEEE 1149.1b, IEEE 1149.4 şi IEEE 1149.5) şi să insiste pe posibilitățile de optimizare a testării echipamentelor şi sistemelor electronice cu JTAG.

CAPITOLUL 2

2. ANALIZA STRUCTURALĂ ŞI POSIBILITĂŢILE OFERITE DE STANDARDUL IEEE 1149.1 PENTRU OPTIMIZAREA TESTĂRII ECHIPAMENTELOR NUMERICE COMPLEXE

În cadrul capitolului precedent autorul a făcut o analiză a echipamentelor utilizate în prezent pe plan mondial la testarea plăcilor şi sistemelor electronice, a propus o clasificare a principalelor grupe de echipamente de testare şi a analizat în detaliu fiecare grupă. Din analiza efectuată a rezultat că este recomandabilă aprofundarea analizei echipamentelor de testare JTAG, existând direcţii care merită analizate în detaliu şi în care autorul poate avea contribuţii personale.

Continuând analiza făcută, capitolul curent îşi propune să realizeze o sinteză a principiilor şi metodelor aferente testării JTAG, în strânsă legătură cu obiectivele pe care le-a propus autorul în cercetarea posibilităţilor de optimizare a testării echipamentelor electronice complexe. La realizarea acestui capitol autorul a studiat o vastă literatură de specialitate în domeniul testării JTAG, a implementat practic unele metode de testare prin utilizarea JTAG şi a verificat rezultatele practice obţinute ([65], [86], [89], [91], [92], [93], [94], [98], [176], [177], [181], [182], [240]).

2.1. Generalităţi

În cadrul acestei secţiuni se prezintă pe scurt: cum a apărut IEEE 1149.x; ce trebuie să conţină circuitele integrate cu facilităţi JTAG; care este principiul de testare JTAG şi ce categorii de teste sunt posibile prin această metodă.

2.1.1. Istoric

JTAG (Joint Test Action Group) sau IEEE 11491. este o metodă de testare recunoscută, apărută ca o necesitate tehnică la cerinţele crescânde de miniaturizare şi de creştere a complexităţii circuitelor numerice [65], [131], [181], [182].

Caracteristici principale pentru IEEE 1149.1:

- metoda a fost dezvoltată de o serie de firme de prestigiu (Philips, British Telecom, Bull, Ericsson, Nixdorf Siemens şi alţii);
- istoric al IEEE1149.1:

  1985: Joint European Test Action Group (JTAG, Philips)

  1986: VHSIC Element-Test & Maintanance (ETM) standard bus (IBM)

  VHSIC Test & Maintanance TM Bus (IBM)

  1988: Joint Test Action Group (JTAG) a propus Boundary Scan Standard

  1990: Boundary Scan a fost standardizat şi aprobat ca IEEE 1149.1- 1990

  iar Boundary Scan Description Language (BSDL) a fost propus de HP

  1993: A fost aprobat 1149.1a - 1993, înlocuind 1149.1 - 1990

  1994: A fost aprobat ca standard 1149.1b şi BSDL

  1995: A fost aprobat 1149.5
- metoda urmăreşte asigurarea unor puncte de testare în circuit şi definirea în general a unui cadru de testare a echipamentelor electronice digitale;
- standardizarea impusă de această metodă de testare asigură compatibilitatea tuturor circuitelor cu facilităţi IEEE 1149.1.

2.1.2. Descrierea standardului JTAG

Schema bloc a părţii JTAG din interiorul circuitelor compatibile IEEE1149.1 este prezentată în figura 2.1 şi se pot face următoarele observaţii în legătura cu ea:

- la nivelul circuitului există numai 4 sau 5 conexiuni suplimentare (TDI, TDO, TMS, TCK şi opţional linia de reset hard TRST*);
- comunicaţia se face serial pe conexiunile de date intrare (TDI) şi date ieşire (TDO);
- liniile TDI, TDO şi TRST (dacă se utilizează) sunt prevăzute cu rezistenţe de pull-up;
- există un registru serial pentru date şi unul pentru instrucţiuni;
- intrarea TDI se citeşte pe frontul crescător al semnalului de tact TCK;
- ieşirea TDO se modifică pe frontul descrescător al semnalului de tact TCK;
- fiecare circuit cu facilităţi JTAG conţine:
  - TAP controller (Test Access Controller, care este un automat de stare);
  - un set de registre de instrucţiune;
  - câteva registre de date;

- pot exista suplimentar:
  - registre specifice de instrucțiune;
  - registre specifice de date.

În concluzie, circuitele cu facilități JTAG prezintă suplimentar 4-5 pini pentru magistrala JTAG şi o parte electronică, care se activează numai la anumite secvențe transmise pe magistrala JTAG. Automatul de stare TAP este comandat pe liniile TMS, TCK şi TDI, funcționând după o diagramă standardizată (2.10), fiind posibil controlul complet al circuitului numai prin magistrala JTAG.

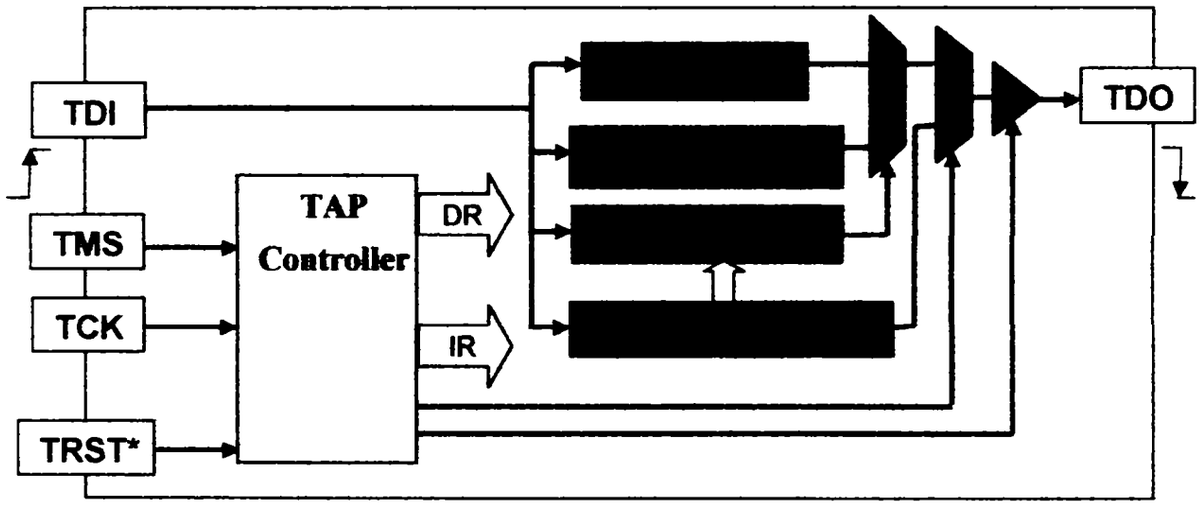


Figura 2.1. Schema bloc a părții JTAG

2.1.3. Testarea unei plăci electronice utilizând JTAG. Exemplu

În figura 2.2. este prezentată o placă electronică ce conține două circuite IEEE 1149.1 conectate din punct de vedere al testării JTAG în cascadă.

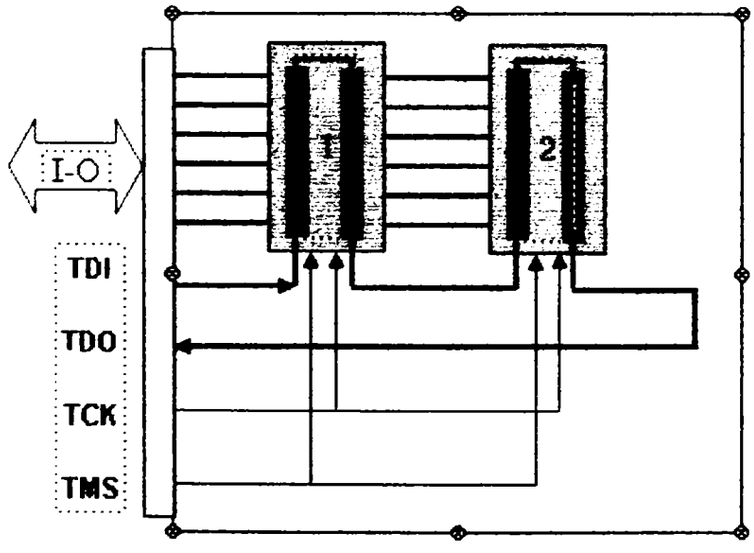


Figura 2.2. Arhitectura de testare pe frontieră. Exemplu

Din figura 2.2. se observă că, pentru fiecare pin de intrare/ieşire, circuitele compatibile IEEE 1149.1 prezintă celule de testare pe frontieră (Boundary-Scan Cell), iar prin intermediul magistralei JTAG valorile logice ale pinilor circuitului se pot citi şi programa. Astfel, prin plasarea unor circuite JTAG pe placa electronică a cărei testare se urmăreşte, se pot executa următoarele categorii de teste:

- teste de continuitate a cablajul plăcii testate;
- teste de scurt-circuit (între trasee, la masă sau la Vcc);
- testarea funcțională a circuitelor de pe placă;
- testarea funcțională a plăcii electronice supusă testului.

În concluzie, rezultă că pentru o testare relativ completă a plăcii electronice utilizând metoda JTAG, este foarte importantă proiectarea din considerente de testabilitate a acesteia, aspect care va fi analizat în detaliu în capitolul următor.

2.2. Problematica testării clasice şi a testării JTAG

2.2.1. De ce a apărut JTAG

JTAG a apărut ca o reacție la:

- *miniaturizare* (figura 2.3). Ca urmare a miniaturizării plăcilor electronice, lățimea traseelor şi distanța dintre trasee a scăzut, devenind tot mai dificilă amplasarea sondelor de test;
- *creşterea densității de integrare la nivelul circuitelor şi complicarea testabilității* (figura 2.4). Odată cu creşterea complexității funcționale şi a densității de integrare a circuitelor electronice, s-a redus şi numărul de noduri de test accesibile, întrucât acestea au devenit interne circuitului.

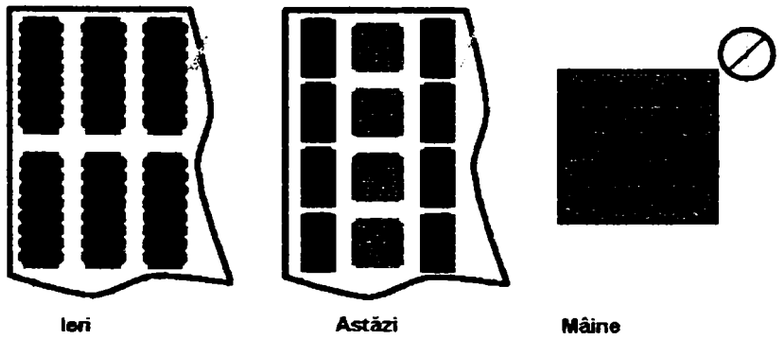


Figura 2.3. Evoluția miniaturizării

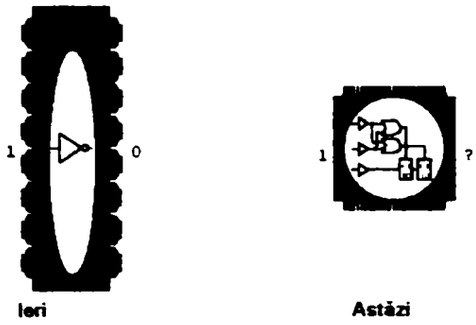


Figura 2.4. Evoluția densității de integrare a circuitelor integrate numerice

Având în vedere schimbările în echipamentele electronice menționate anterior, metodele clasice de testare au devenit prea scumpe, iar la echipamentele electronice complexe chiar ineficiente. În consecință, a devenit necesară standardizarea unei metode de testare practicabile în cazul echipamentelor electronice complexe şi al circuitelor integrate pe scară foarte largă.

2.2.2. Testarea funcțională

Testarea funcțională presupune:

- testarea la nivelul plăcii şi nu al structurii;
- generarea preponderent manuală a vectorilor de test;
- limitarea testării numai la nivelul I/O;
- considerarea plăcii testate ca o cutie neagră, făcând abstracție de funcțiile îndeplinite de circuitele individuale amplasate pe ea.

În figura 2.5. se prezintă ca exemplu metoda de testare funcțională a unei plăci electronice. Se poate observa că echipamentul electronic testat este privit ca o cutie neagră, testarea realizându-se numai la nivelul conectorilor, adică se aplică vectorii de test pe intrări şi se urmăreşte răspunsul plăcii testate pe ieşiri.

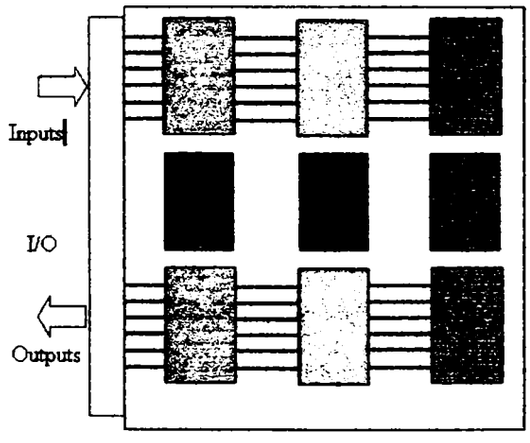


Figura 2.5. Exemplu de testare funcțională

Principalele avantaje ale testării funcționale sunt:

- se poate aplica la toate echipamentele, inclusiv la cele complexe;
- întrucât vectorii de test se aplică în principal pe conectori, costurile testării funcționale sunt mult mai mici decât cele ale testării clasice;
- este o metodă eficientă de testare dacă placa electronică testată a fost proiectată pe considerente de testare.

Dezavantajul principal al testării funcționale este că nu permite testarea individuală a circuitelor din interiorul "cutiei negre".

2.2.3. Testarea clasică în circuit

Testarea clasică se face de obicei cu sonde (sau ace) şi presupune:

- conectarea simultană a tuturor sondelor pe placă, metodă a cărei utilizare este limitată de complexitatea plăcii testate;
- costuri importante pentru testare şi ajustări (dependente de placa testată);
- testarea unor funcționalități poate fi ignorată pentru a reduce durata testului;
- verificarea continuității traseelor de interconectare;
- generarea automată a vectorilor de test pe baza unor modele de testare în circuit (care sunt foarte specifice);
- accesul limitat al sondelor de test datorită:
    - dimensiunilor pinilor capsulelor;
    - posibilității accesării doar a celor două fețe ale cablajului;
    - numărului limitat de puncte de acces pe cablaj.

În figura 2.6. este prezentată - ca exemplu - testarea în circuit a unui echipament electronic. Se poate observa că testarea în circuit presupune accesarea unor pini ai circuitelor amplasate pe placa testată pentru aplicarea stimulilor şi citirea răspunsului. În aceste condiții, metoda de testare în circuit se pretează la testarea echipamentelor electronice, care nu sunt foarte complexe şi care permit accesarea prin sonde a punctelor de test.

Din analiza bibliografică autorul a constatat că această metodă de testare este încă des utilizată, iar la echipamentele profesionale de testare în circuit există un număr de sonde cu poziție variabilă acționate de motoare pas cu pas de precizie ([67], [68], [72], [205]). Este de aşteptat că testarea clasică se va mai utiliza la testarea în producție, dar datorită limitărilor tehnologice răspândirea acesteia va scădea treptat.

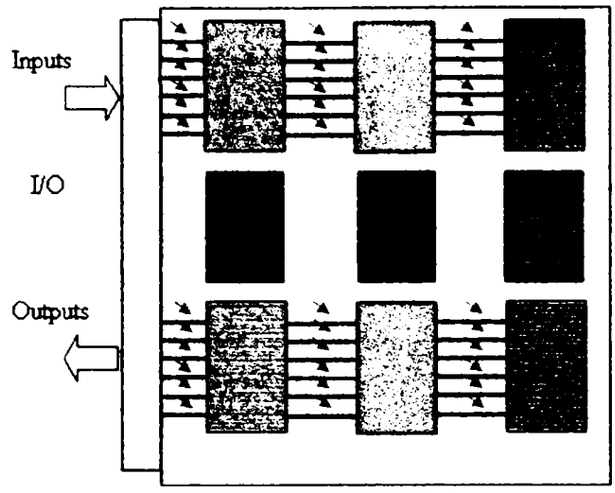


Figura 2.6. Testarea în circuit. Exemplu.

2.3. Elementele de bază ale testării pe frontieră

În figura 2.7. se prezintă o placă electronică ce cuprinde două circuite cu facilități IEEE1149.1. Se poate observa că:

- sondele de test de la testarea "în circuit" s-au mutat în circuitele integrate, creând o rețea virtuală de sonde;
- celulele de testare pe frontieră supraveghează fiecare nod permițând şi testarea continuității cablajului;
- celulele de observare/control sunt active doar la activarea modului de test, iar în funcționarea normală a echipamentului sunt transparente;
- scanarea asigură observarea rezultatelor testului şi posibilitatea încărcării vectorilor de test;
- metoda de scanare serială necesită un minimum de resurse (pini/trasee) pe circuite/placă(busul IEEE 1149.1. are 4-5 fire);
- testarea se face pe baza structurii plăcii electronice şi nu este limitată de circuit/funcție/complexitate;
- accesul pentru teste nu este limitat de factori fizici (lățimea traseelor, dimensiunea pad-urilor şi distanța între pinii circuitelor);
- permite testarea cablajului (conexiune corectă, circuit deschis, scurt între trasee sau la VCC respectiv masă)

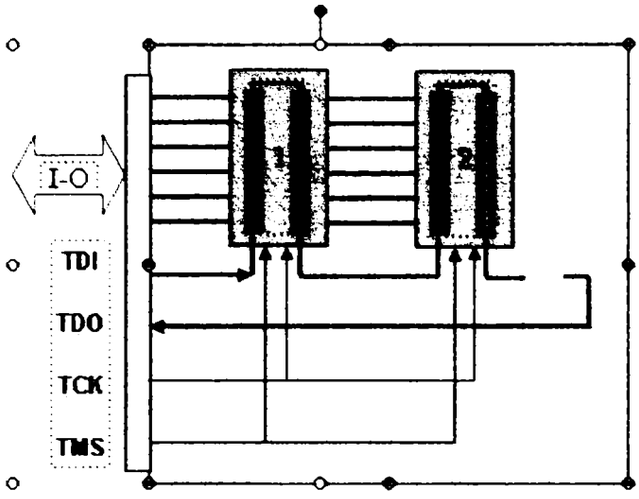


Figura 2.7. Conceptul de testare pe frontieră

Conceptul de testare pe frontieră a apărut în anul 1990 şi a câştigat teren treptat, fiind o soluție potrivită pentru testarea pe scară largă a sistemelor sau plăcilor electronice complexe şi chiar a circuitelor integrate.

2.4. Celula de scanare pe frontieră

Circuitele cu facilități IEEE 1149.1 prezintă obligatoriu pentru fiecare pin o celulă de scanare pe frontieră (figura 2.8.).

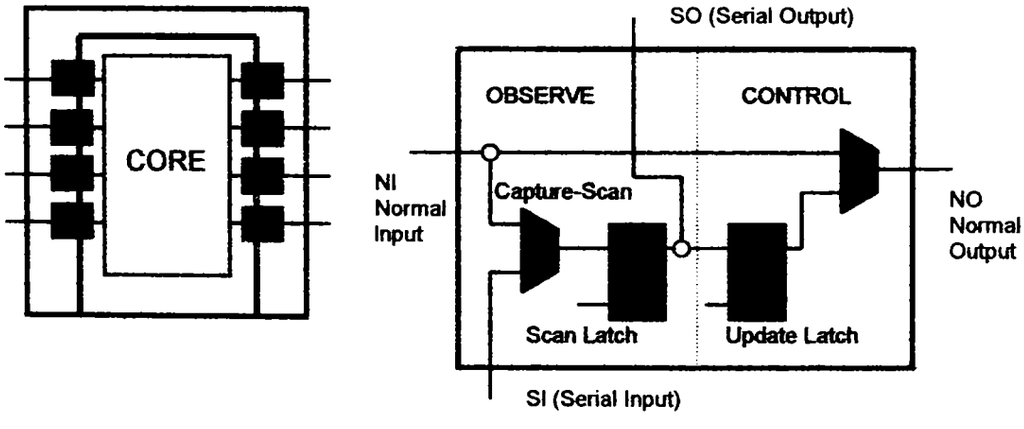


Figura 2.8. Celulele de scanare pe frontieră şi structura internă a unei celule

Din figura 2.8. se observă că celula de scanare poate fi încărcată pe intrarea serială sau pe intrarea normală, iar informația din celulă se poate citi pe ieşirea normală sau pe

ieşirea serială. De asemenea, există posibilitatea de a programa/citi, prin intermediul celulelor de scanare, valorile logice aplicate părții numerice interne a circuitului (CORE).

2.5. Structura bloc a părții IEEE1149.1

În figura 2.9 se prezintă structura internă a părții de scanare serială.

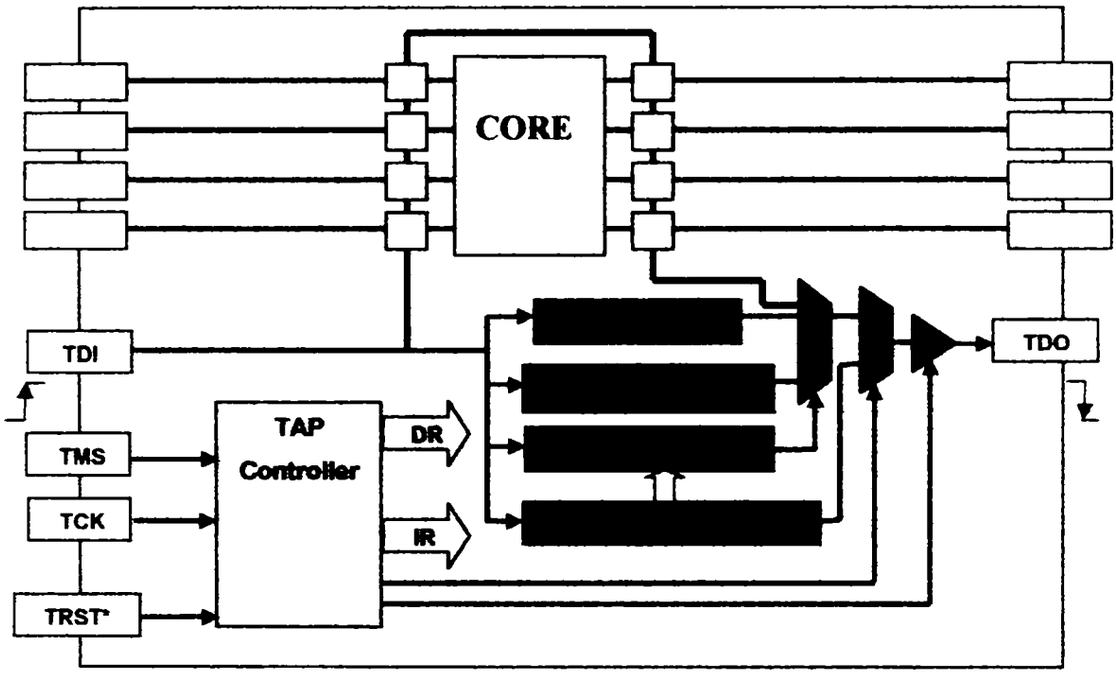


Figura 2.9. Structura bloc a părții IEEE1149.1

Din figura 2.9 se poate observa că:

- scanarea pe frontieră şi alte operații cu registrele de date se fac sub controlul controlerului TAP ca urmare a instrucțiunilor din registrul de instrucțiune;
- datele sunt deplasate de la TDI la TDO prin registrele de date sau de instrucțiune sub controlul TAP;
- TAP operează sincron cu TCK, iar pentru selecția stării se utilizează semnalul TMS;
- circuitul prezintă minimum două registre de date:
  - registrul de scanare pe frontieră, care cuprinde toate registrele celulelor de testare pe frontieră (cel puțin o celulă pe pin);
  - registrul BYPASS.

2.6. TAP (Test Access Port) controller

Pe baza semnalelor de intrare TDI, TMS şi de tact (TCK), TAP parcurge stările din figura 2.10 şi generează toate semnalele de control necesare funcționării circuitului conform specificațiilor JTAG. Valorile prezentate adiacent cu fiecare stare în figura 2.10 la fiecare tranziție reprezintă nivelul semnalului TMS pe frontul descrescător al semnalului de tact. Toate tranzițiile controllerului intervin pe frontul crescător al semnalului de tact, iar acțiunea corespunzătoare controllerului se produce pe frontul crescător sau descrescător al semnalului de tact. De exemplu, dacă controllerul TAP este în starea **TEST-LOGIC-RESET,** atâta timp cât TMS rămâne la nivelul "1", controllerul rămâne în aceeaşi stare. Pentru a părăsi această stare TMS trebuie pus pe "0" logic. Pe următorul front crescător al semnalului de tact se părăseşte starea TEST-LOGIC-RESET şi se intră în starea RUN-TEST-IDLE. Atâta timp cât TMS rămâne la nivel "0" logic, această stare se menține. Dacă TMS este trecut pe "1" logic, pe următorul front crescător al semnalului de tact se va intra în starea SELECT-DR-SCAN pe durata unei singure stări a semnalului de tact, după care se trece în starea SELECT-IR-SCAN sau CAPTURE-DR şi aşa mai departe.

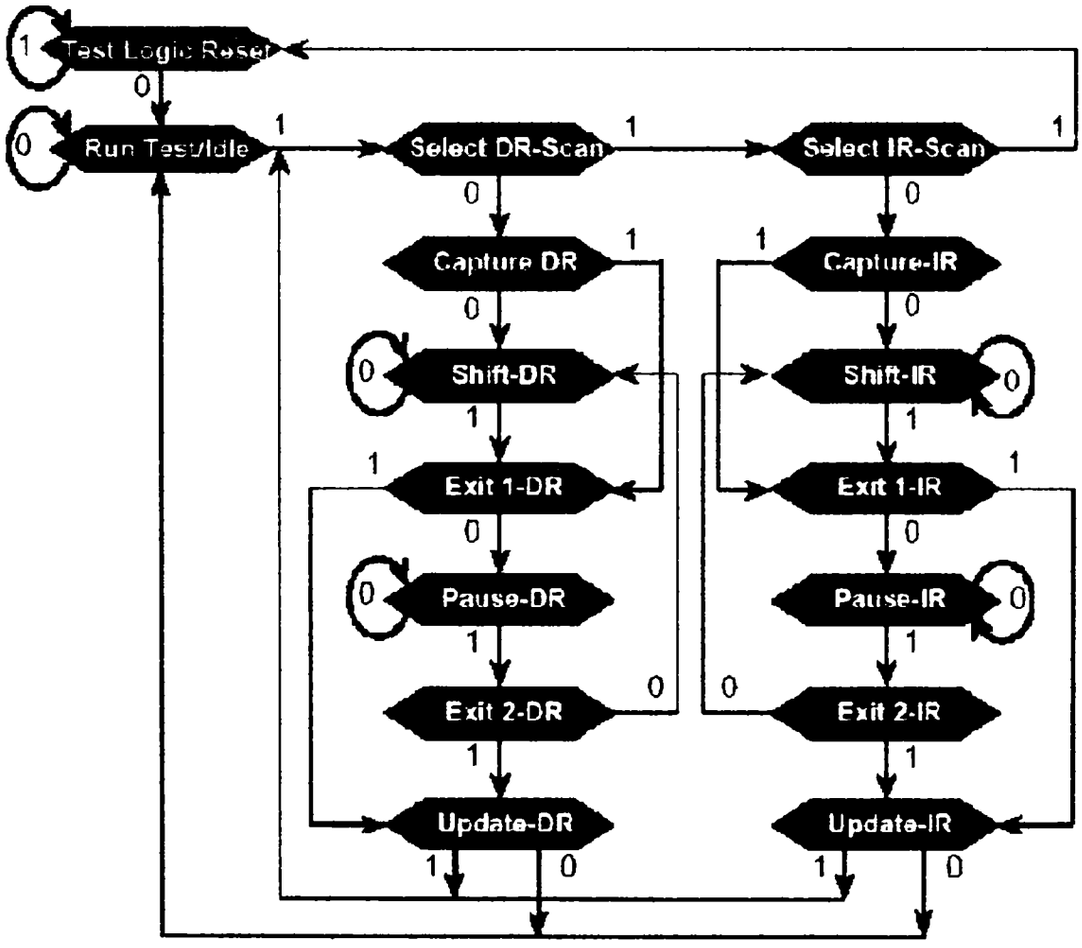


Figura 2.10. Diagrama de stare a controllerului TAP

În figura 2.11 se prezintă diagramele de timp asociate secvenței de instrucțiuni descrise anterior.

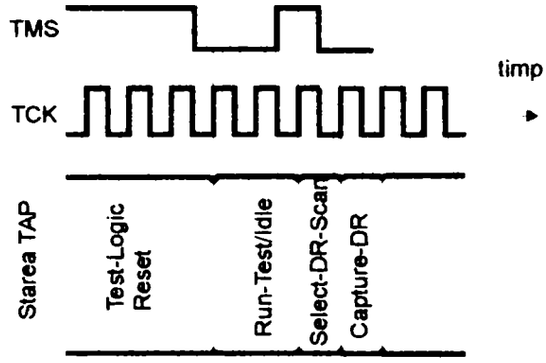


Figura 2.11. Diagramele de timp asociate la câteva stări ale controllerului TAP

Acțiunile executate în fiecare stare a controllerului sunt următoarele:

Test-Logic-Reset

În această stare toate activitățile de testare sunt resetate, iar circuitul operează normal. Această stare se obține prin menținerea semnalului TMS la nivel logic "1" pe durata a minimum 4 fronturi crescătoare ale semnalului de tact. Numărul exact de pulsuri ale semnalului de tact depinde de starea controllerului TAP. Semnalul TRST, care este opțional, realizează resetarea hard a circuitului şi aduce imediat TAP controllerul în starea Test-Logic-Reset.

Run-Test/Idle

Controllerul TAP se găseşte în această stare între operațiile de scanare şi va rămâne în starea Run-Test/Idle atâta timp cât semnalul TMS este ținut pe "0". Operațiile logice de test depind de instrucțiunea conținută în registrul de instrucțiune.

Capture-DR

Starea permite încărcarea datelor de pe intrările paralele în registrul de date selectat. Acțiunea se va petrece pe frontul crescător al semnalului de tact, iar starea controllerului TAP rămâne neschimbată.

Shift-DR

În această stare a controllerului datele capturate anterior sunt deplasate serial pe calea TDI/TDO, o singură deplasare are loc pe fiecare front crescător al semnalului de tact. Instrucțiunea nu schimbă starea curentă a controllerului.

Update-DR

Odată ce controllerul este în această stare, procesul de deplasare s-a încheiat. Datele conținute în registrele de date sunt memorate în registrele de ieşire paralelă şi se pot extrage ulterior din circuitul testat prin deplasare serială.

Capture-IR

Starea permite încărcarea instrucțiunii în registrul de instrucțiuni, iar acțiunea se produce pe frontul crescător al semnalului de tact. Registrele de date selectate de instrucțiunea curentă rămân în starea precedentă.

Shift-IR

În această stare a controllerului, datele capturate în starea Capture-IR sunt deplasate pe calea TDI/TDO, o deplasare pentru fiecare front crescător al semnalului de tact. Instrucțiunea curentă nu se schimbă în această stare a controllerului.

Update-IR

Instrucțiunea introdusă serial este memorată, ea fiind accesibilă pe ieşirile paralele ale registrului de instrucțiuni. Noua instrucțiune devine validă când controllerul TAP este în această stare. Toate registrele de deplasare selectate de această instrucțiune rămân în starea precedentă. Acțiunea se petrece pe frontul scăzător al semnalului de tact.

Pause-DR & Pause-IR

Stările Pause-DR şi Pause-IR sunt utilizate pentru oprirea temporară a procesului de deplasare a registrului de date, respectiv de instrucțiune. Aceste stări se mențin atâta timp cât semnalul TMS este ținut pe "0" şi sunt introduse pentru a permite echipamentelor automate de testare să citească datele din memorie.

2.7. Ieşirile TAP

TAP controllerul are rolul de circuit de comandă şi prezintă stările din figura 2.12.

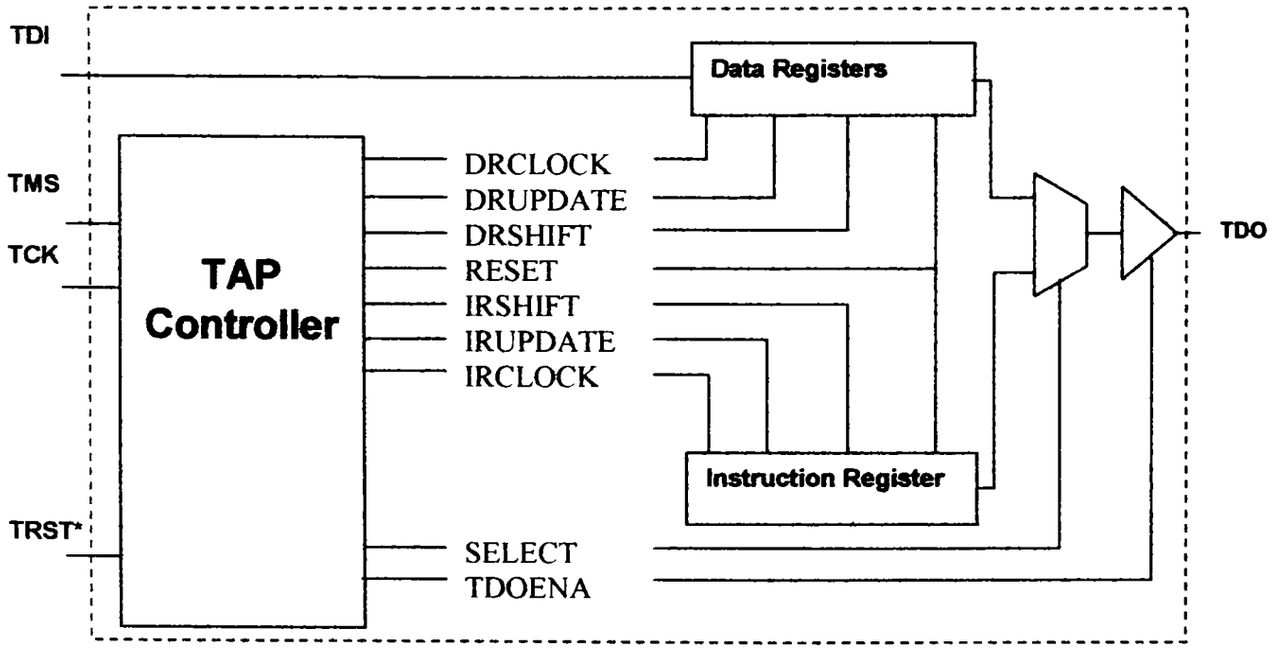


Figura 2.12. Ieşirile controllerului TAP

Se poate observa că TAP controllerul generează semnalele pentru:

- actualizarea registrului de instrucțiuni;
- actualizarea registrului de date;
- validarea/invalidarea ieşirilor şi selecția registrului de date sau instrucțiuni.

2.8. Instrucțiuni IEEE1149.1

În tabelul 2.1 se prezintă instrucțiunile IEEE1149.1.

Instrucțiune	Cod	Mod	Registru selectat de instrucțiune
EXTEST	0..0*	Test	Celulele de testare pe frontieră
SAMPLE-PRELOAD	Nestandard	Normal	Celulele de testare pe frontieră
BYPASS	1..1	Normal	celula bypass
Intest	Nestandard	Test	Celulele de testare pe frontieră
Runbist	Nestandard	Test	Nestandard
Idcode	Nestandard	Normal	Registrul de identificare (ID)
Usercode	Nestandard	Normal	Registrul de identificare (ID)
Clamp	Nestandard	Test	celula bypass
HighZ	Nestandard	Test	celula bypass
Instrucțiuni specifice	Nestandard	Nestandard	Nestandard

Tabelul 2.1. Instrucțiuni IEEE1149.1

Din tabelul 2.1. se observă că primele trei instrucțiuni sunt obligatorii conform cerințelor IEEE1149.1, iar restul instrucțiunilor utilizate depind de producătorul circuitului (uzual circuitele complexe prezintă mai multe instrucțiuni suplimentare). De obicei, producătorii de circuite integrate cu facilități de testare JTAG implementează mai multe instrucțiuni decât cele din tabelul 2.1., unele dintre ele fiind utilizate numai la testarea la producție.

2.8.1. Instrucțiunea EXTEST

Caracteristici:

- Instrucțiunea permite testarea interconexiunilor externe circuitului şi se realizează în doi paşi:
  - se execută prima dată instrucțiunea EXTEST pentru încărcarea celulelor de scanare cu valorile adecvate pentru pinii de ieşire;

- se execută încă o dată instrucțiunea EXTEST pentru citirea celulelor de scanare serială, adică a pinilor de intrare.

- scanarea serială permite ca răspunsul la vectorul de test să fie citit pe linia TDO simultan cu introducerea noului vector de test pe linia TDI;
- este o instrucțiune obligatorie conform specificațiilor IEEE1149.1;
- după operația de deplasare noii stimuli de test sunt transferați în latch-urile celulelor de testare pe frontieră.

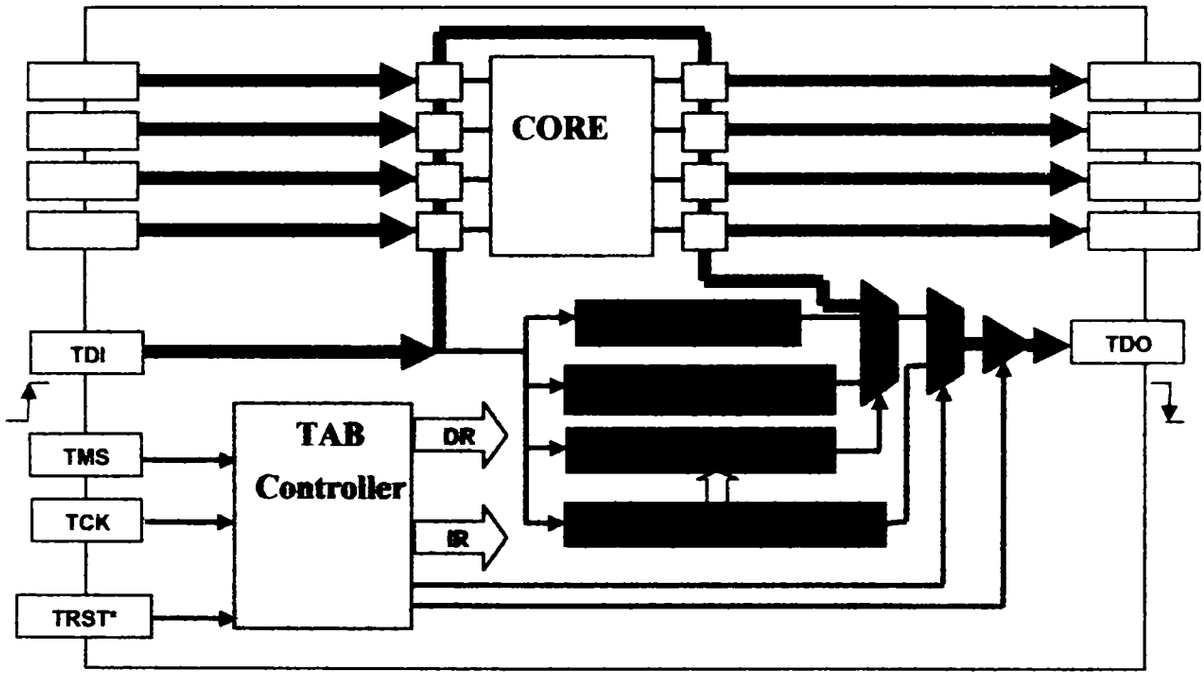


Figura 2.13. Instrucțiunea EXTEST

Această instrucțiune permite verificarea cablajului asociat circuitului respectiv, fiind frecvent utilizată. În capitolul 3 se va exemplifica modul de utilizare a acestei instrucțiuni la testarea unei unități de aviație de control al încărcăturii (CSC).

În noua versiune a standardului IEEE 1149.1. codul instrucțiunii EXTEST (care conținea numai "0") va fi modificat iar acest cod va fi utilizat pentru alte instrucțiuni. În felul acesta se evita situația în care un eventual scurtcircuit la masă pe calea TDI-TDO determină în prezent o intrare automată în executarea instrucțiunii EXTEST.

2.8.2. Instrucțiunea SAMPLE/PRELOAD

Caracteristici:

- instrucțiunea asigură încărcarea registrelor de scanare seriale înainte de intrarea în test mode:
  - intrările şi ieşirile operează în mod normal (nu în test mode);

- valorile logice ale pinilor de intrare şi ieşire (corespunzătoare nucleului circuitului) sunt memorate în celulele de testare pe frontieră (latch-uri);

- prin intermediul deplasării seriale răspunsul poate fi citit serial, în timp ce noii vectori de test sunt introduşi;
- după operaţia de deplasare, noii stimuli sunt transferaţi în latch-urile celulelor de testare pe frontieră;
- este o instrucţiune obligatorie conform specificaţiilor IEEE1149.1.

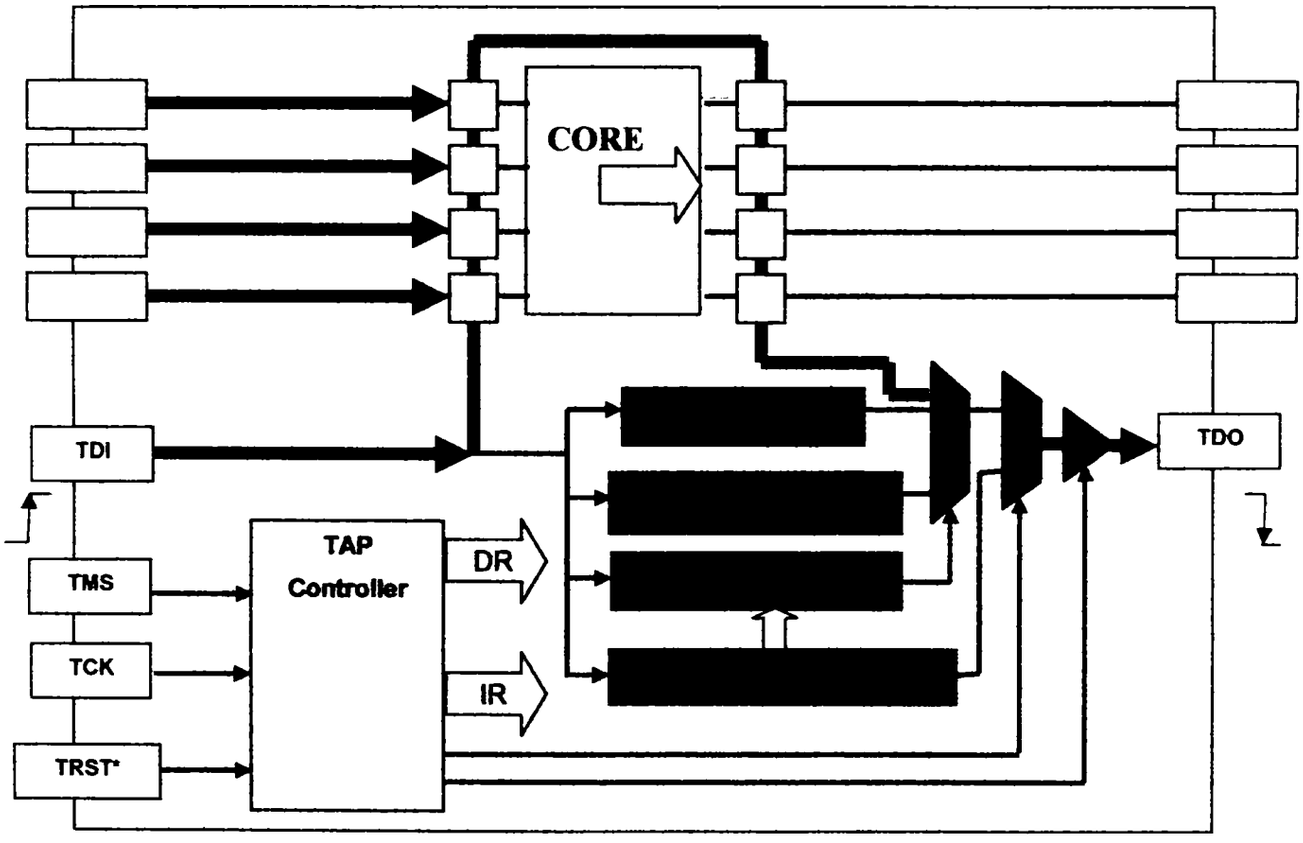


Figura 2.14. Instrucţiunea SAMPLE/PRELOAD

Instrucţiunea permite citirea nivelurilor logice de pe pinii circuitului scanat şi a ieşirilor logice interne ale circuitului, existând astfel posibilitatea de "spionare" a frontierei circuitului ţintă, prin intermediul magistralei JTAG. În versiunea revizuită a standardului IEE 1149.1. se prevede separarea instrucţiunii în SAMPLE şi PRELOAD dar va rămâne totuşi valabilă şi instrucţiunea SAMPLE/PRELOAD [131],[24].

2.8.3. Instrucţiunea BYPASS

Caracteristici:

- asigură o scurtare a căii de scanare (figura 2.15), deci reducerea timpului de testare;

- ieşirile şi intrările circuitului operează în mod normal;
- pe calea de scanare este selectat numai registrul BYPASS de un singur bit;
- instrucțiunea este obligatorie conform IEEE1149.1;
- instrucțiunile care nu sunt înțelese de circuit (codul care a fost introdus în registru de instrucțiune nu corespunde cu un cod valid pentru circuitul respectiv) sunt înlocuite cu instrucțiunea BYPASS.

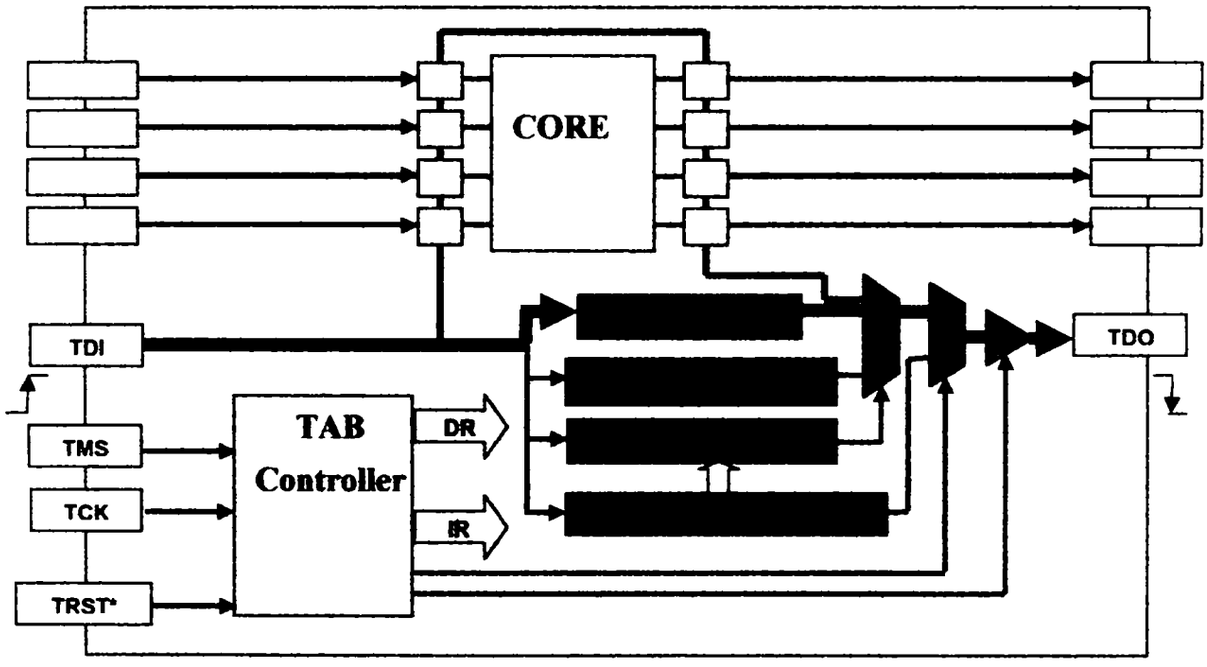


Figura 2.15. Instrucțiunea BYPASS

Această instrucțiune este foarte des utilizată, iar în capitolele 3 şi 4 se va exemplifica modul în care se utilizează şi implicațiile acesteia în reducerea timpului de testare.

2.8.4. Instrucțiunea INTEST (opțională)

Caracteristici:

- pinii de ieşire operează în modul de test, prezentând pe ieşiri conținutul celulelor de testare pe frontieră sau înaltă impedanță;
- intrările operează în modul de test, iar conținutul celulelor de scanare serială se transferă în UPDATE LATCH;
- permite testarea funcțională a logicii interne a circuitului, ieşirile nucleului logic fiind sunt memorate în celulele de scanare seriale înainte de deplasarea serială;
- citirea răspunsului şi introducerea noilor stimuli se face serial pe liniile TDO-TDI;
- după deplasare serială vectori de test sunt transferați în UPDATE-LATCH.

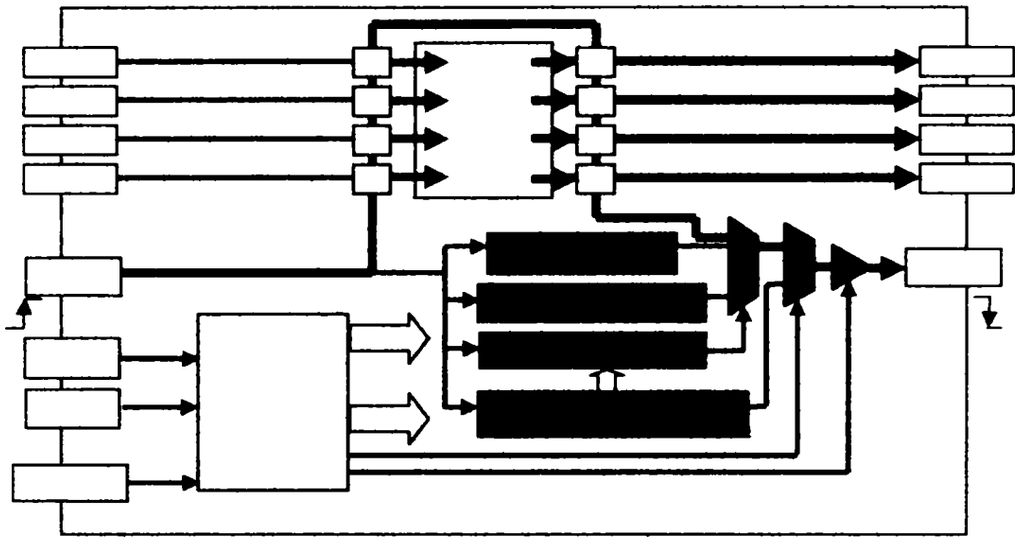


Figura 2.16. Instrucțiunea Intest

2.8.5. Instrucțiunea RUNBIST (opțională)

Instrucțiunea permite autotestarea circuitului, fiind opțională şi necesită existența în circuit a unei părți hardware suplimentare.

Caracteristici:

- pinii de ieşire operează în modul de test, prezentând pe ieşiri conținutul celulelor de testare pe frontieră sau înaltă impedanță;
- instrucțiunea rulează independent în nucleul circuitului în starea RUN-TEST/IDLE;
- după execuția testului, rezultatul este capturat într-un registru specific;
- rezultatul/semnătura se poate citi serial pe linia TDO.

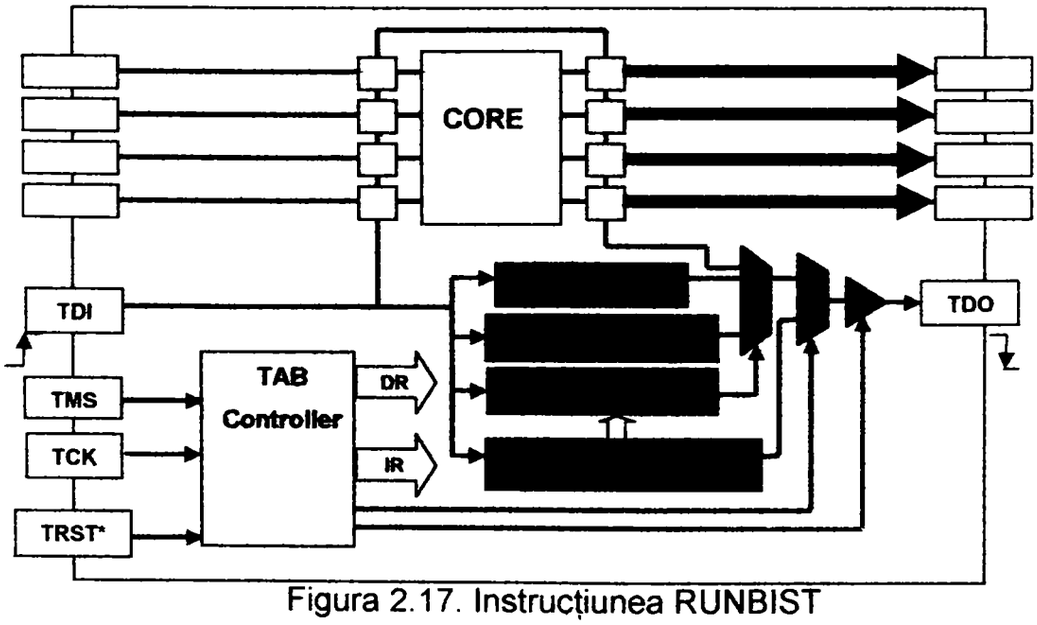


Figura 2.17. Instrucțiunea RUNBIST

2.8.6. Instrucţiunea Clamp (opţională)

Instrucţiunea asigură protecţia ieşirilor pe durata testării în circuit sau în timpul testării funcţionale. Ieşirile circuitului operează în modul de test şi prezintă valorile din registrele de scanare serială UPDATE LATCH, iar pe calea de scanare se introduce registrul BYPASS.

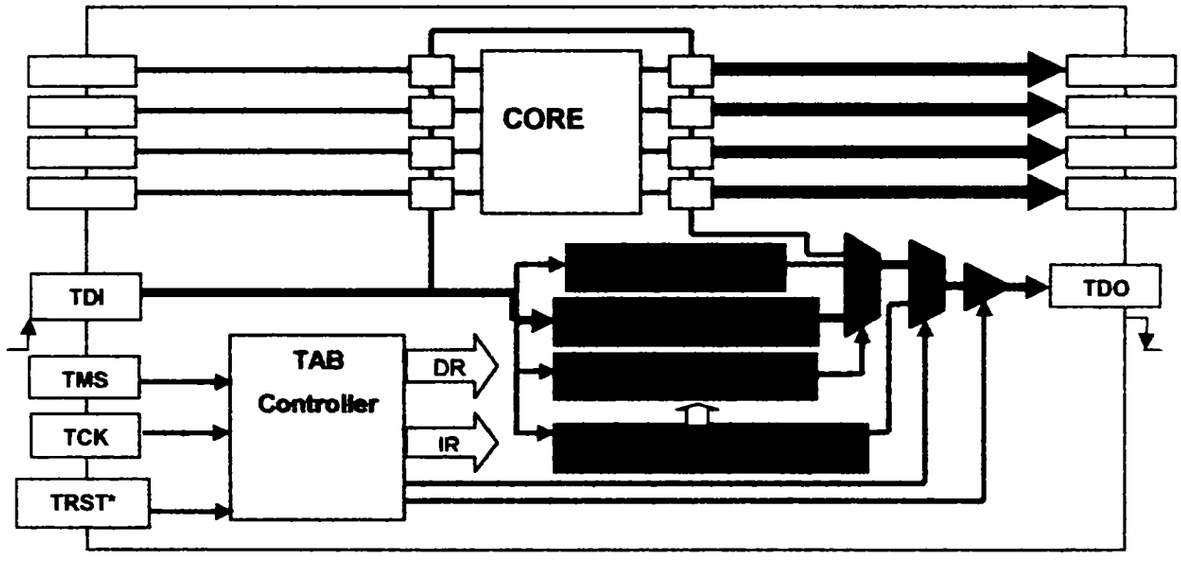


Figura 2.18. Instrucţiunea Clamp

2.8.7. Instrucţiunea HIGHZ (opţională)

Asigură invalidarea ieşirilor circuitului pe durata instrucţiunii de testare în circuit şi a testării funcţionale pe frontieră. Ieşirile vor fi comutate în înaltă impedanţă, inclusiv pinii, care în mod normal au numai două stări de funcţionare. Calea de scanare este prin intermediul registrului BYPASS, iar funcţionarea este identică cu cea care corespunde instrucţiunii Clamp, dar ieşirile circuitului sunt în înaltă impedanţă.

2.8.8. Instrucţiunea IDCODE

Instrucţiunea asigură identificarea circuitului. În timpul rulării acestei instrucţiuni, ieşirile şi intrările operează normal. Registrul de 32 de biţi de identificare este selectat pentru scanare şi captură, iar valorile citite serial indică producătorul, tipul şi versiunea circuitului. Această instrucţiune este obligatorie.

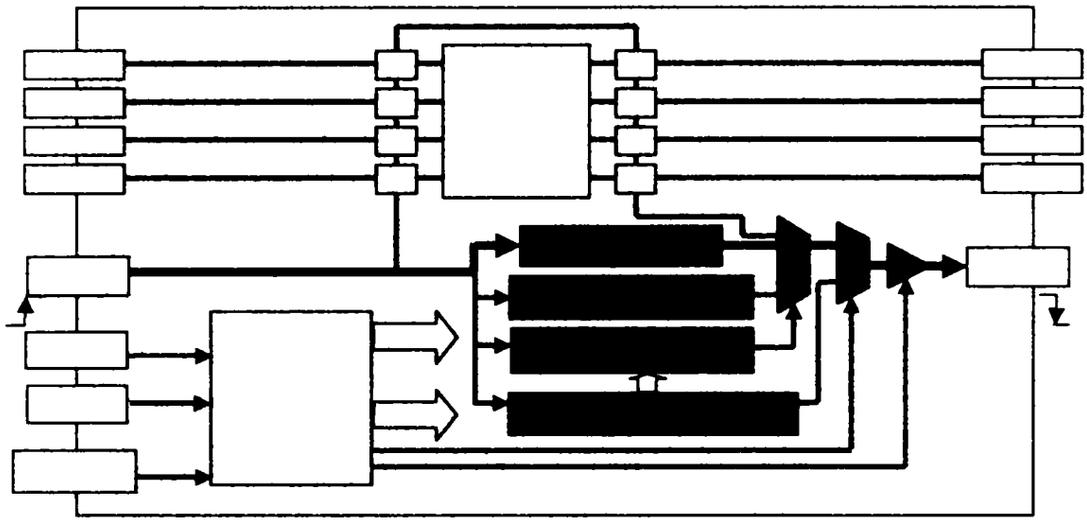


Figura 2.19. Instrucțiunea IDCODE

2.8.9. Instrucțiunea USERCODE (opțională)

Instrucțiunea este utilizată la identificarea circuitului în cazul PLD-urilor (programabile). În timpul rulării acestei instrucțiuni, ieşirile şi intrările operează normal. Registrul de 32 de biți de identificare este selectat pentru scanare şi captură, iar valorile citite serial indică producătorul, tipul şi versiunea. Funcționarea este identică cu cea a instrucțiunii IDCODE.

2.9. Celulele de scanare, de observare şi control

În figura 2.20 şi 2.21 sunt prezentate celule de testare pe frontieră.

Caracteristicile celulelor de testare pe frontieră sunt:

- asigură multiplexarea, captura sau selecția (selectează încărcarea paralelă sau serială);
- asigură multiplexarea "test" sau "date" (selectează calea de date sau datele de test);
- preîntâmpină prin intermediul latch-urile zgomotul pe pin pe durata scanării;
- celule de scanare serială există pentru fiecare pin (date şi semnale de control);
- există pe fiecare intrare sau ieşire de date.

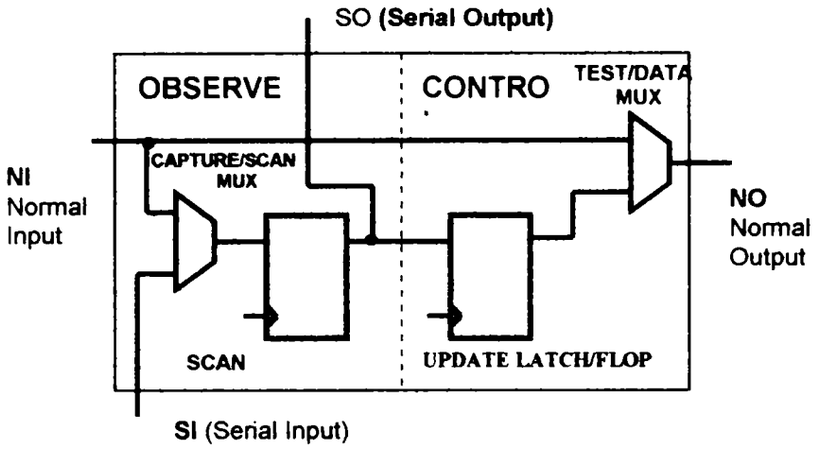


Figura 2.20. Celulă de observare şi control cu latch-uri

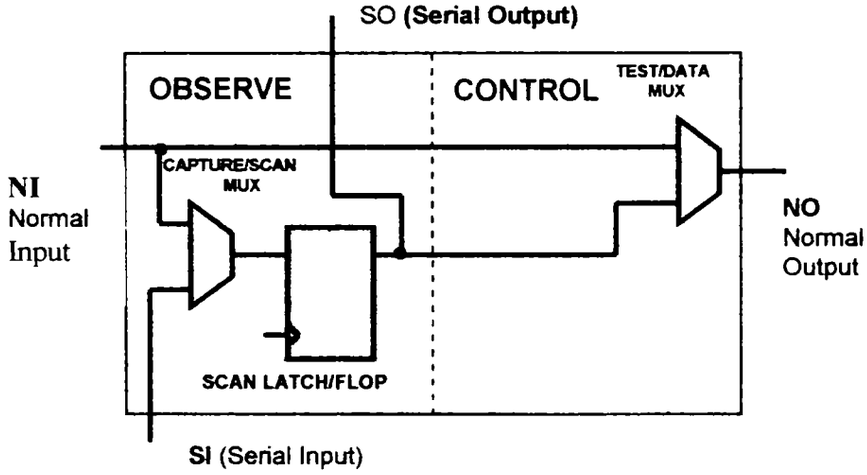


Figura 2.21. Celulă de observare şi control fără latch de control

În figura 2.21 se prezintă o celulă de scanare serială dar fără latch de ieşire. Inexistența latch-ului permite modificarea nivelului logic pe pinii de ieşire pe durata scanării, ceea ce în unele aplicații constituie un avantaj, dar în altele este un dezavantaj.

În figura 2.22 se prezintă o celulă cu posibilități de observare, care poate fi utilizată pentru toate intrările dacă instrucțiunea **INTEST** nu este implementată. Această celulă nu este utilizabilă pentru pinii de ieşire sau pentru pinii de intrare, care nu sunt de tact, dacă este folosită instrucțiunea INTEST.

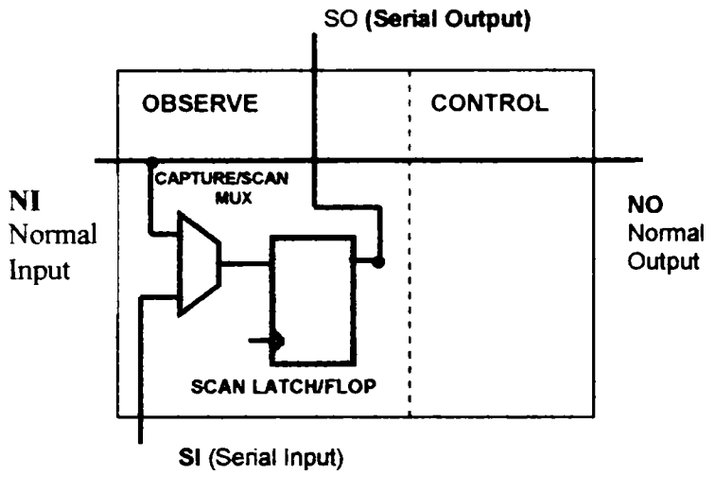


Figura 2.22. Celulă de observare

2.9.1. Celulele de scanare pe frontieră de pe intrări

În figura 2.23 se prezintă o intrare şi celula de testare pe frontieră asociată. Este necesar ca fiecare intrare a circuitului să prezinte cel puțin o celulă cu facilități de observare şi poate exista o celulă de scanare cu facilități de control.

Dacă instrucțiunea **INTEST** este suportată de circuit, atunci fiecare intrare, care nu este intrare de tact, trebuie să prezinte o celulă cu facilități de control (cu latch sau fără). Intrările de tact pot avea o celulă de scanare cu facilități de control.

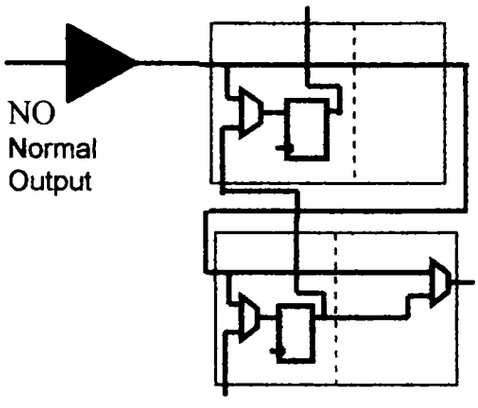


Figura 2.23. Celule de scanare pe intrări

2.9.2. Celulele de scanare de pe ieşirile cu 2 stări

În figura 2.24 se prezintă celula de testare de pe o ieşire cu 2 stări. Fiecare ieşire trebuie să fie prevăzută cu cel puțin o celulă de testare cu latch-uri pe ieşire şi cu facilități de observare şi control şi oricâte celule cu facilități de observare. Dacă instrucțiunea HighZ este suportată, toate ieşirile cu 2 stări trebuie implementate prin buffere cu 3 stări.

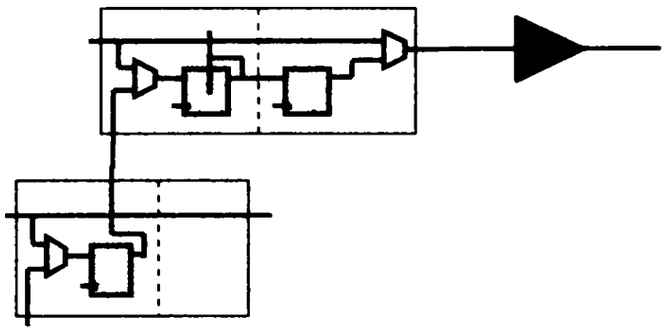


Figura 2.24. Celulele de scanare de pe ieşiri

2.9.3. Celulele de scanare de pe ieşirile cu 3 stări

În figura 2.25 se prezintă o celulă de scanare de pe o ieşire cu 3 stări. Fiecare ieşire cu 3 stări trebuie să prezinte cel puțin o celulă de scanare cu facilități de observare/control şi cu latch la ieşire şi poate avea un număr de celule de scanare numai cu facilități de observare.

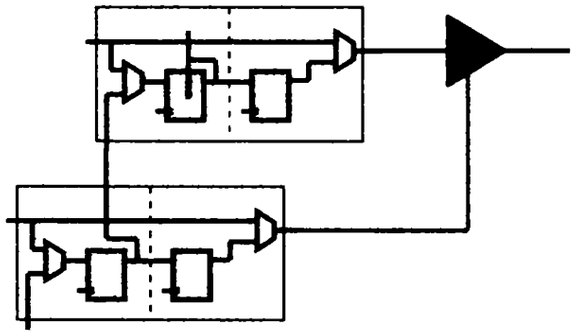


Figura 2.25. Celulele de scanare de pe ieşirile cu 3 stări

În figura 2.26 se prezintă celulele de scanare de pe ieşirile bidirecționale. Pinul fiind bidirecțional, înseamnă că există o celulă de scanare pentru intrări şi una pentru ieşiri, adică cel puțin o celulă de scanare cu latch şi cu posibilități de observare/control (pentru ieşiri) şi o celulă cu posibilități de observare (pentru intrări).

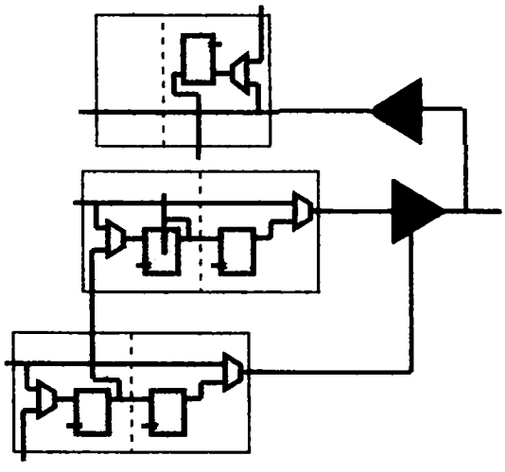


Figura 2.26. Celule de scanare de pe pinii bidirecționali

2.10. Standardul IEEE 1149.5 (MTM - Bus)

2.10.1. Generalităţi

Caracteristici principale:

- IEEE 1149.5 standardizează comunicaţia pentru test, diagnoză şi mentenanţă, permiţând folosirea unui Test Control Master şi până la 250 module slave;
- a derivat din programul VHSIC faza 2 (1987: IBM, Honeywell, TRW);
- devine standard complet în 1996 şi este destinat testării on-line şi off-line;
- prima aplicaţie în care acest standard s-a utilizat pe scară mare a fost la testarea modulelor electronice din avionul Boeing 777, efectuată de autorul acestei teze în 1998;
- spre deosebire de IEEE 1149.1 şi IEEE 1149.4 care sunt orientate la nivelul plachetei, IEEE 1149.5 este proiectat pentru a standardiza operaţiile de comunicare în procesul de testare a plăcilor electronice conectate la fundul de sertar (back plane).

2.10.2. Arhitectura 1149.5

În figura 2.27 se prezintă arhitectura magistralei 1149.5. Se poate observa că în cazul acestui standard avem un echipament "master" (de obicei echipamentul de test) şi o serie de module "slave" (care pot fi, de exemplu, plăcile electronice din cadrul echipamentului testat)[22], [131].

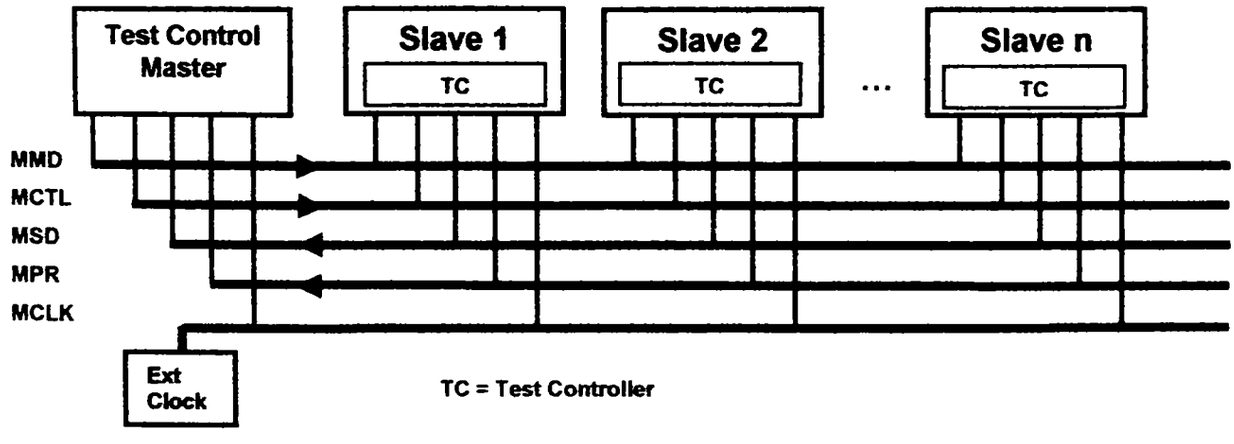


Figura 2.27. Arhitectura magistralei 1149.5

IEEE 1149.5 este definit pentru a fi utilizat în paralel sau într-o structură ierarhică împreună cu alte magistrale de test asigurând:

- **interoperabilitatea**. Acesta presupune că prin intermediul magistralei standard oferite se asigură o integrare şi interconectare facilă dintre modulele sistemului;

- **reducerea timpului alocat operațiilor de test**. Abordarea ierarhică permite efectuarea testelor în paralel în mai multe module testate şi prin aceasta reducerea timpului de test în ansamblu;
- **simplificarea softului de test**. Programele de test pot fi elaborate independent pentru fiecare modul în parte ;
- **descongestionarea traficului pe magistrală**. Dacă rutinele de test sunt implementate le nivelul modulelor, atunci traficul de date la nivelul magistralei este redus la transmiterea comenzilor de test şi la recepționarea rezultatelor
- **reducerea timpului de test** ;
- **interschimbare**. Dacă procedurile sau secvențele specifice de test sunt conținute la nivelul modulelor, atunci modulele identice ca funcționare dar proiectate diferit pot fi schimbate între ele fără a necesita o reprogramare specifică;
- **reducerea costului operațiilor de test**. Magistrala permite concentrarea controlului tuturor operații de test la nivelul unui singur modul de control. În felul acesta se reduc mult costurile de interfațare cu modulele testate.
- **simplificare**. Dacă interfața de test cu fiecare modul nu necesită informații specifice, atunci se poate utiliza o interfață comună pentru toate modulele, fără a fi necesară o programare specifică pentru fiecare modul.

2.10.3. Protocolul 1149.5

În figura 2.28 sunt prezentate nivelurile de comunicație corespunzătoare protocolului IEEE 1149.5. Se poate observa că protocolul este structurat pe 3 nivele, fiind un bazat pe pachete cu multiple moduri de adresare, iar interfața cu 1149.1 este definită.

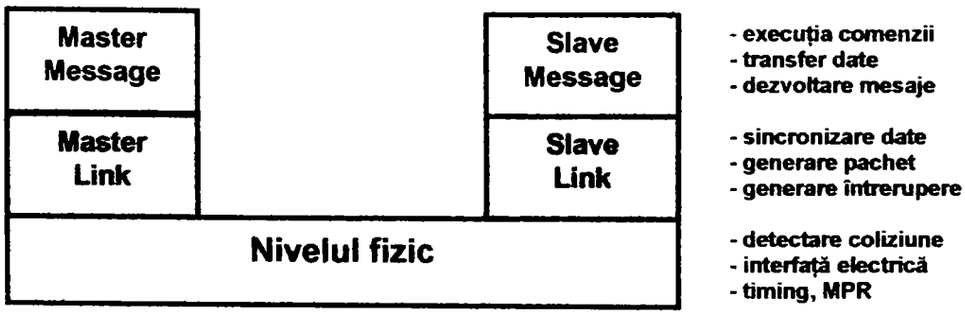


Figura 2.28. Protocolul 1149.5

2.10.4. Compararea 1149.1 şi 1149.5

În figura 2.29 se prezintă comparativ standardele 1149.1 şi 1149.5. Se poate observa că standardul 1149.5 este proiectat pentru testarea mai multor sisteme electronice interconectate prin magistrala 1149.5, iar comunicaţia între master şi slave se face pe bază de pachete, existând şi posibilitatea de corecţie a erorilor. Standardul 1149.1 este mai potrivit pentru testarea la nivel de circuit, placă sau sistem electronic, este mai uşor de folosit şi mai răspândit în practică, existând o serie de circuite standard de suport .

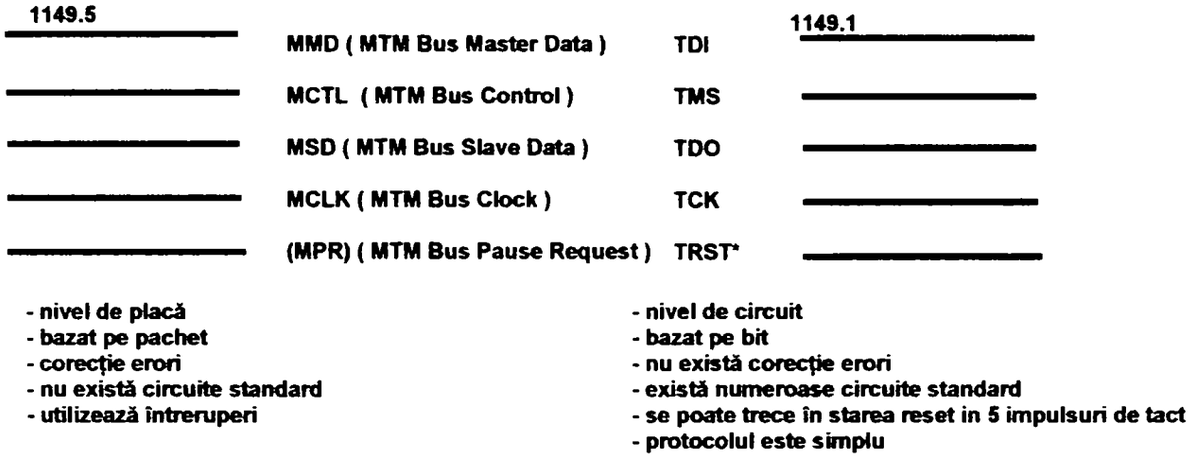


Figura 2.29. Comparaţie între 1149.1 şi 1149.5

Din păcate cu toate că IEEE 1149.5 poate ajuta foarte mult testarea sistemelor electronice complexe, răspândirea este relativ redusă şi nu mai este sprijinit de IEEE fiind de vânzare [88].

2.11. Standardul IEEE 1149.4

2.11. 1. Caracteristici principale

- este compatibil la nivel de controller cu IEEE 1149.1 şi a fost standardizat în 1999;
- adaugă posibilitatea de comutare a funcţiilor analogice (comutatoare sau buffere) la intrările /ieşirile analogice, pentru interfaţa cu magistrala de test;
- utilizează două magistrale analogice pentru toate circuitele de pe placă;
- prezintă suport opţional pentru testarea în circuit [21], [85], [131].

P1149.4 nu îşi propune:

- să rezolve toate problemele testării circuitelor integrate, echipamentelor şi sistemelor analogice şi numerice;

- să dicteze strategia de test a circuitelor integrate, a echipamentelor şi sistemelor analogice şi numerice.

P1149.4 îşi propune să:

- reducă dificultăţile testării circuitelor integrate, echipamentelor şi sistemelor analogice şi numerice ;
- să faciliteze proiectarea din considerente de testare.

2.11.2. Arhitectura părţii electronice suplimentare a circuitelor IEEE 1149.4

În fig. 2.30 se prezintă un circuit compatibil IEEE 1149.1 fără partea electronică suplimentară cerută de IEEE 1149.4. Pentru un astfel de circuit se obţin informaţii despre funcţionarea secţiunii analogice din interiorul chip-ului prin intermediul convertorului analog digital ADC şi se transmit semnale spre partea analogică prin intermediul convertorului digital analog DAC. Datele sunt vehiculate pe calea de test prin intermediul unor module Boundary Scan (DBM, Digital Boundary Module) având aceeaşi structură ca celule prezentate pentru standardul IEEE 1149.1.

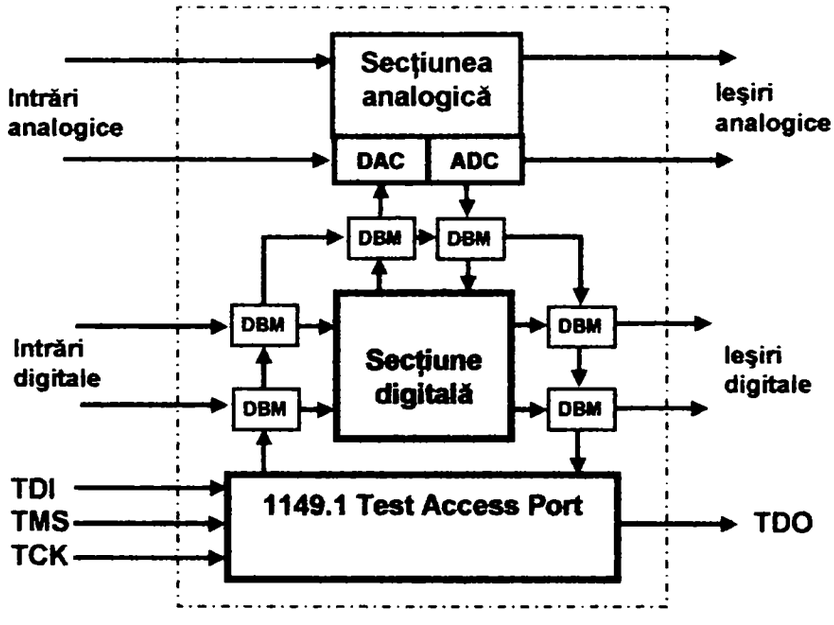


Fig. 2.30. Extensia standardului 1149.1 pentru testarea circuitelor analogice

În fig. 2.31 se prezintă standardul IEEE 1149.1 modificat prin introducerea unor celule speciale de interfaţare analogică (ABM, Analogue Boundary Module). Celulele ABM sunt conectate pe calea de scanare internă situaţie în care ele au un comportament digital permiţând transmiterea serială a datelor de test.

În acest context la funcţionarea în regim 1149.1 singurele diferenţe la sunt:

- în modul de lucru normal şi în modul test pinii digitali transferă numai date digitale;
- în modul normal de funcționare pinii analogici transferă date analogice iar în modul de test transferă date digitale.

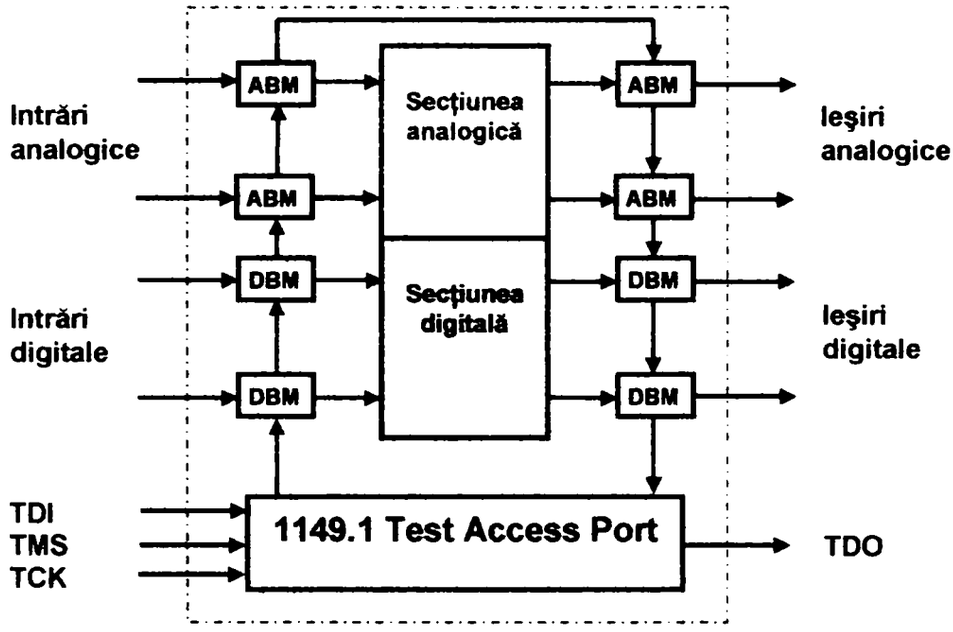


Fig. 2.31. Extensia IEEE 1149.1 pentru testarea circuitelor analogice (modul lucru 1149.1)

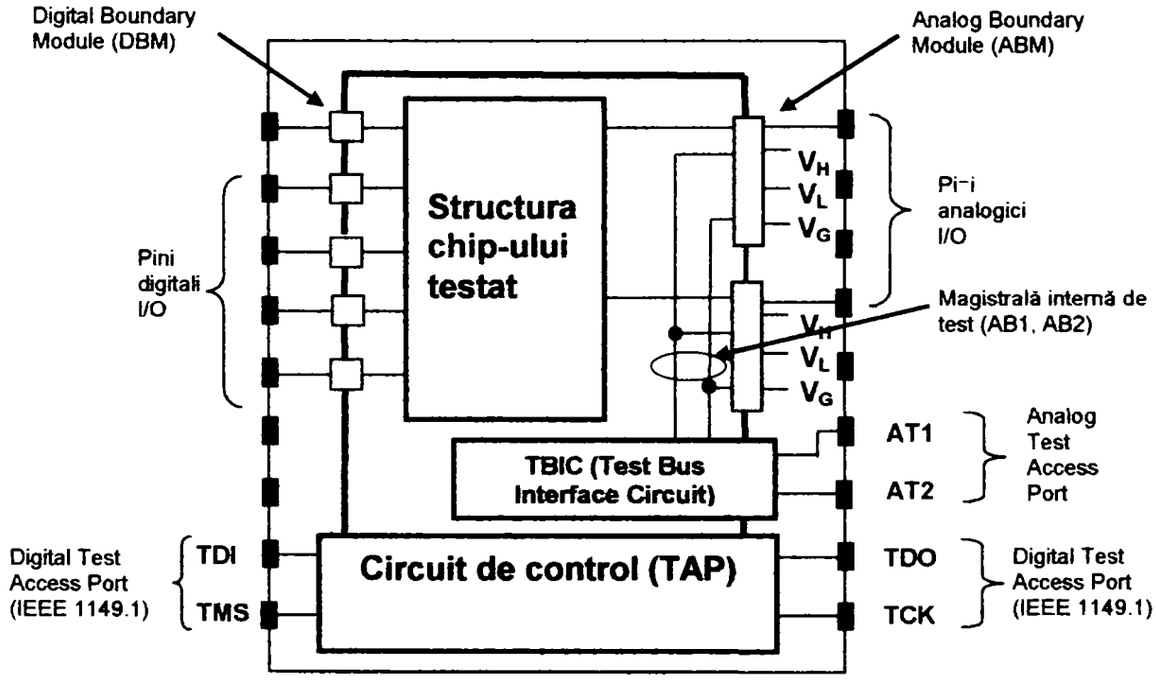


Fig. 2.33. Extensia IEEE 1149.1 pentru testarea circuitelor analogice în modul analogic

Din fig. 2.34 se poate observa că TBIC are în structură două blocuri funcționale:

- un bloc pentru autotestarea conexiunilor externe AT1 şi AT2 în regim digital;

- un bloc pentru conectarea magistralelor interne AB1 şi AB2 la magistrala analogică externă AT1, AT2.

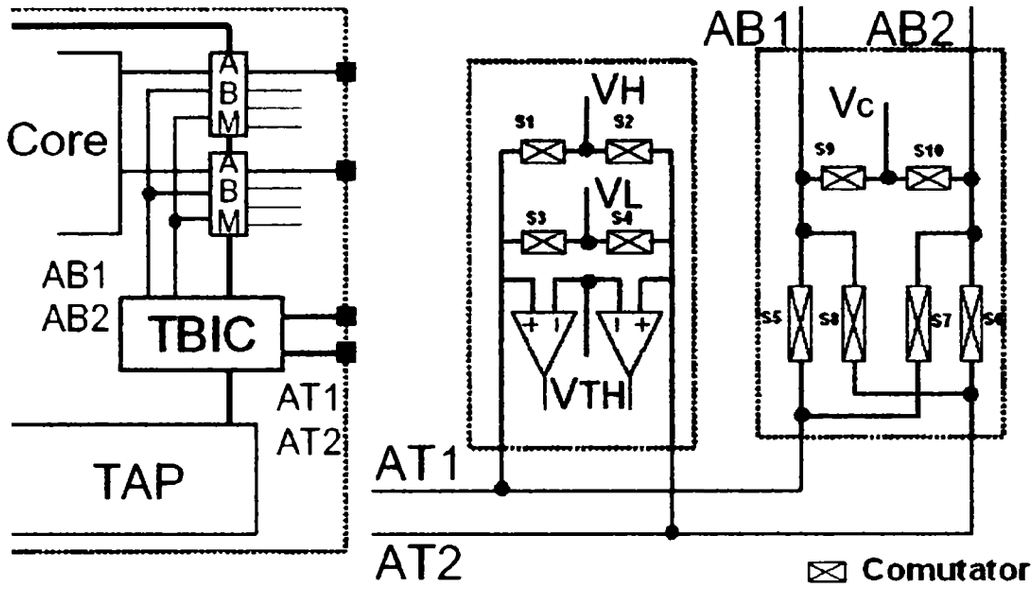


Fig. 2.34. Arhitectura TBIC

Analizând fig. 2.34 se pot observa următoarele:

- prin comutatoarele S1, S2, S3 sau S4 se pot comanda nivele logice VH sau VL pe una din cele două conexiuni externe AT1 sau AT2;
- prin intermediul comparatoarelor se pot digitiza (transforma în 0 logic sau în 1 logic în funcție de VTH) semnalele de pe cele două linii AT1 şi AT2;
- comutatoarele S5, S6, S7, S8 permit conectarea liniei ATx la liniile ABy (x=1,2; y =1,2).
- prin perechile de comutatoare S5 şi S7 sau S6 şi S8 se poate crea o buclă de reacție conectând AT1 la AB1 şi la AT2 sau AT2 la AB2 şi AT1 având rol de autocalibrare.

Analizând structura celulei ABM din fig. 2.35 se poate observa:

- modulele ABM sunt prevăzute şi cu trei tensiuni de referință, VH, VL, şi VG pentru masa analogică (ground);
- comutatorul CD permite deconectarea circuitului intern de la modulul ABM;
- prin intermediul comutatorului SB1 se poate injecta un curent în modulul ABM prin pinul AT1 şi linia AB1;
- prin intermediul comutatorului SB2 se poate citi o tensiune analogică din modulul ABM prin linia AB2 şi pinul AT2.
- comparatorul prezent în structura pinului are rolul de a digitiza (de a transforma în 0 logic sau în 1 logic) semnalul din modulul ABM.

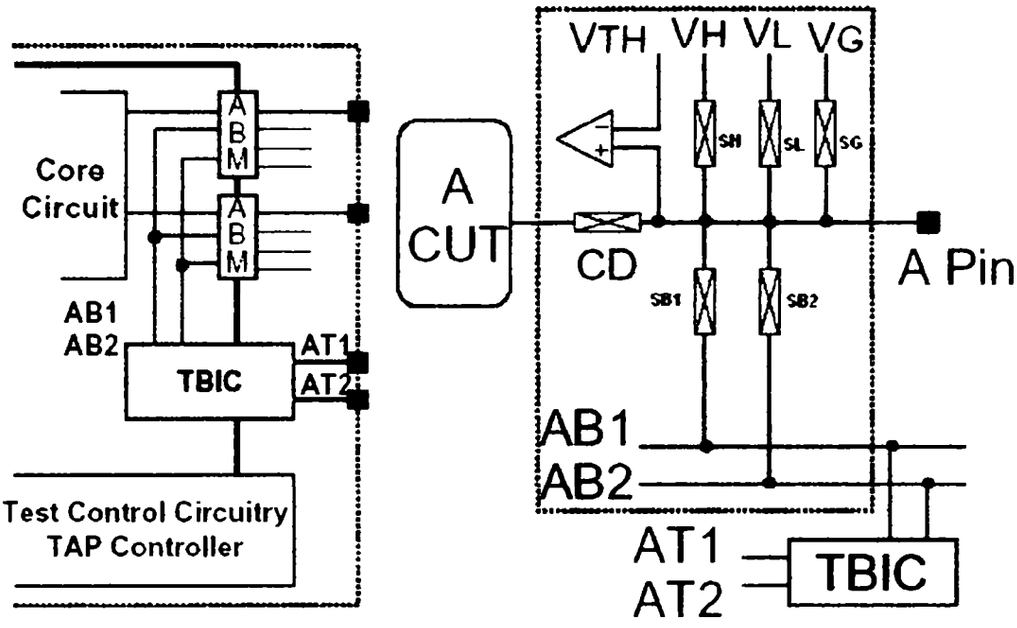


Fig. 2.35. Arhitectura celulei ABM

2.11.3. Implementarea standardului IEEE 1149.4 la nivelul unei plăci

În fig. 2.36 se prezintă modul de conectare al circuitelor IEEE 1149.1 şi IEEE1149.4. Se poate observa că magistrala JTAG se realizează ca la circuitele IEEE 1149.1 iar liniile AT1 şi AT2 se conectează în paralel la toate circuitele IEEE 1149.4.

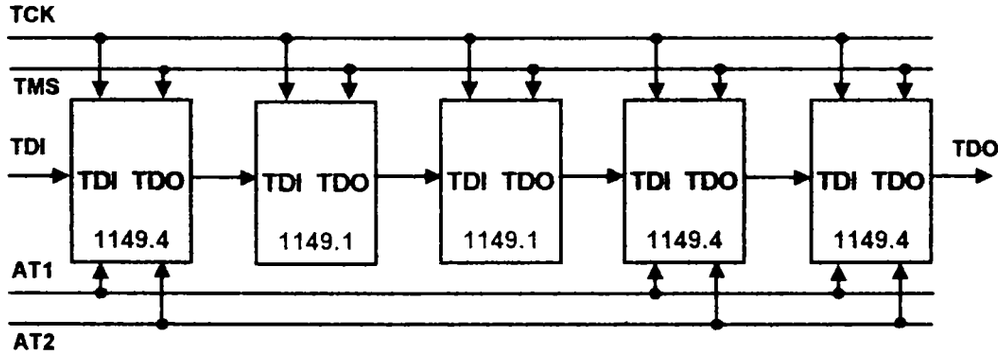


Fig. 2.36. Modul de conectare a circuitelor IEEE 1149.4 şi IEEE 1149.1

2.11.4. Principii de măsura

În figurile 2.37-2.39 se prezintă metodele utilizate de determinare a periferiei circuitelor IEEE 1149.1. Se poate observa că în generat principiul de măsură presupune injectarea unui curent prin intermediul TBIC şi ABM şi măsurarea tensiunii iar prin legea lui Ohm rezultă rezistența/impedanța echivalentă la borne.

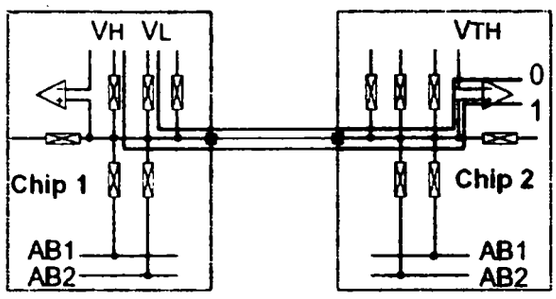


Fig. 2.37. Testarea scurtcircuit-ului sau a circuitului deschis

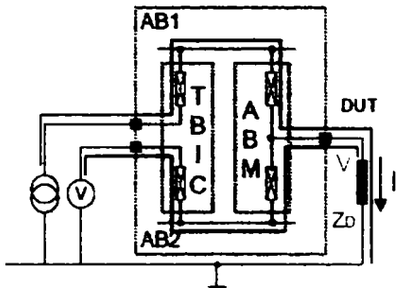


Fig. 2.38. Măsurarea unei impedanțe

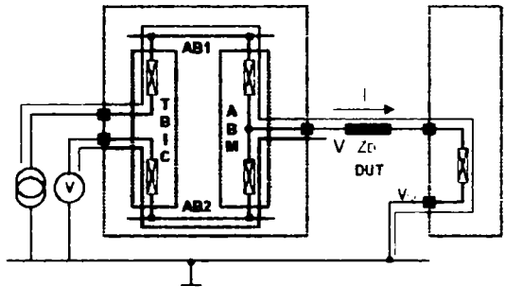


Fig. 2.39. Măsurarea unei impedanțe flotante

Se impune a se menționa că testarea cu circuite IEEE 1149.4 se poate aplica şi în regim dinamic dar trebuie să se țină cont că eroarea de măsură creşte odată cu creşterea frecvenței semnalului utilizat şi este afectată de capacități, rezistențe şi impedanțe.

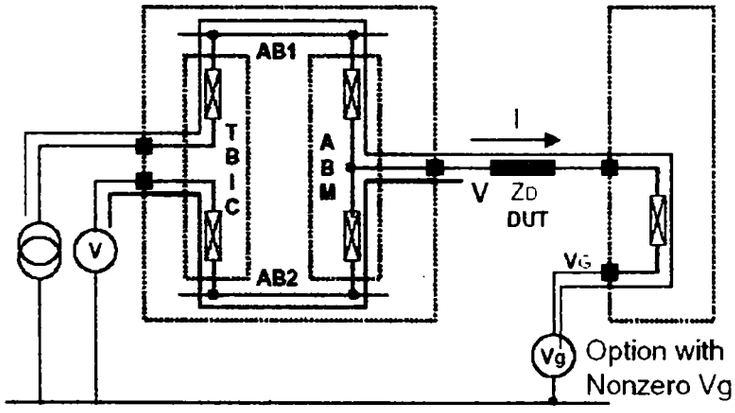


Fig. 2.40. Măsurarea unei impedanțe la configurarea cu ofset

2.12. Alte standarde de testare

Întrucât principiile standardului IEEE 1149.1 sunt utilizate într-un domeniu mai larg au apărut o serie de standarde suplimentare aşa cum se prezintă în continuare iar în fig. 2.41 se prezintă un sumar al acestora:

A. IEEE P1532– based In System Configuration of Programable Devices

Îşi propune să găsească o cale de armonizare a principiilor standardului IEEE 1149.1 cu modul specific de funcționare al dispozitivelor programabile.

B. Standardul IEEE P1500 Embeded Core Test Interface

Acest standard este în elaborare în scopul de a asigura testabilitatea pentru sisteme on-chip.

C. Standardul IEEE 1450 Standard Tester Interface Language (STIL) for Digital Test Vector Data

Acest proiect îşi propune să definească un limbaj care:

- să faciliteze transferul unor volum mare de informaţie de la mediile CAE (Computer Aided Engineering) spre mediile reprezentate de echipamentele de test automat, ATE (Automated Test Equipment);
- să specifice modelul, formatul şi informaţiile de timp care definesc vectorii de test pentru un circuit testat, DUT (Device Under Test);
- să faciliteze schimbul unui volum cât mai mare de informaţii între diferitele medii implicate în procesul de testare cum ar fi: mediile de generare automată a vectorilor de test, mediile BIST, etc. şi optimizarea acestor informaţii în scopul transferului lor spre ATE.

D. Standardul IEEE 1149.6

Acest proiect va defini o extensie a standardului IEEE 1149.1-2001 complementar cu IEEE 1149.4 pentru a permite testarea reţelelor cuplate AC şi a celor diferenţiale la viteze înalte. Acest proiect va specifica şi o extensie a limbajului BSDL pentru a suporta noua structură de intrare/ieşire.

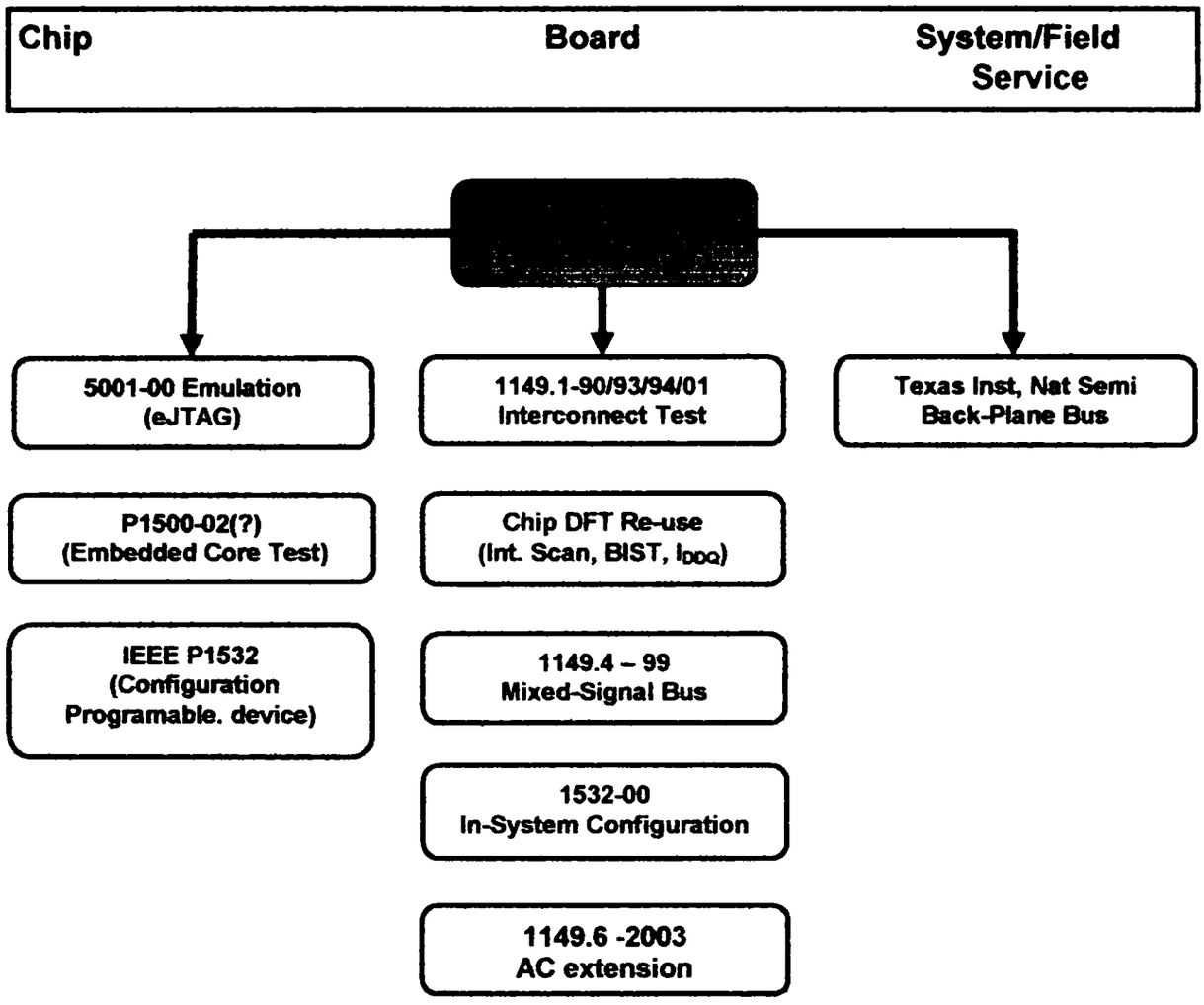


Fig. 2.41. Evoluția standardului JTAG

2.12. Concluzii

Scopul principal al acestui capitol este realizarea unei analize succinte a principiilor şi metodelor aferente testării JTAG, în strânsă legătură cu obiectivele pe care autorul le-a propus în cercetarea posibilităților de optimizare a testării echipamentelor electronice complexe.

În secțiunea 2.1. (***Generalități***) autorul prezintă pe scurt ce înseamnă JTAG, cum a apărut, ce elemente suplimentare prezintă circuitele cu facilități JTAG şi care este principiul testării pe frontieră. Din cadrul paragrafului se poate observa că circuitele cu facilități JTAG prezintă o parte hardware suplimentară relativ simplă şi ieftină, care permite testarea oricărui circuit oricât de complex (inclusiv a unui procesor Pentium) prin intermediul unei magistrale cu 4-5 fire. Abordarea hardware a testării JTAG va fi

continuată şi în capitolul următor, autorul propunând şi soluţii hardware originale de implementare a magistralei JTAG.

În secţiunea 2.2. (***Problematica testării clasice şi JTAG***) autorul a considerat necesar să prezinte pe scurt de ce este necesară şi superioară metoda de testare JTAG faţă de metodele de testare clasice. În această secţiune se subliniază că odată cu creşterea complexităţii sistemelor electronice, metodele clasice de testare utilizate în producţie, bazate pe echipamente de testare cu ace, nu mai pot ţine pasul cu miniaturizarea, iar metodele de testare bazate pe JTAG se vor impune.

Secţiunea 2.3. (***Bazele testării pe frontieră***) evidenţiază că la testarea JTAG sondele clasice s-au mutat în circuitele cu facilităţi JTAG, iar printr-o proiectare adecvată a echipamentului electronic sau a sistemului testat se poate asigura o testare superioară comparativ cu testarea clasică la un preţ mult mai redus.

În secţiunea 2.4. (***Celula de testare pe frontieră***) autorul prezintă componenţa "spionului" elementar al circuitelor JTAG, iar în secţiunea 2.5. (***Structura bloc a părţii IEEE1149.1***) se prezintă schematic hardware-ul suplimentar al circuitelor JTAG. Autorul consideră necesar să sublinieze că partea electronică suplimentară pentru testabilitate introdusă în circuitele integrate cu facilităţi JTAG este relativ simplă, iar datorită densităţii de integrare actuale a circuitelor integrate costul suplimentar este de regulă acceptabil.

În secţiunile 2.6. şi 2.7. (***TAP controller şi leşirile TAP***) se prezintă circuitul de comandă şi control al circuitelor JTAG şi automatul de stare al circuitelor JTAG. Este necesar a se sublinia că toate circuitele JTAG operează după un automat de stare relativ simplu, dar eficient.

Secţiunea 2.8. (***Instrucţiuni 1149.1***) prezintă cele mai utilizate instrucţiuni JTAG, insistându-se şi asupra operaţiilor pe care le efectuează instrucţiunile. Se poate remarca existenţa unui set de instrucţiuni standard JTAG, care asigură operaţiile elementare pentru testarea JTAG, dar de obicei producătorii de circuite cu facilităţi JTAG introduc instrucţiuni suplimentare.

În secţiunea 2.9 (***Celule de scanare, de observare şi control***) autorul prezintă diverse structuri de celule de observare şi control. Se poate observa că tipul celulei de scanare depinde de tipul pinului (bidirecţional, numai intrare sau numai ieşire), iar la proiectarea plăcii electronice trebuie să se ţină cont de acest aspect.

Secţiunea 2.10 (***Standardul IEEE 1149.5)*** a subliniat că IEEE 1149.5 este destinat în special testării la nivel de placă şi va fi utilizat tot mai frecvent, uneori în combinaţie cu

IEEE 1149.1, dar prezintă actualmente ca dezavantaj major răspândirea redusă a circuitelor de suport.

Secțiunea 2.11 (***Standardul IEEE 1149.4***) prezintă pe scurt ce înseamnă IEEE 1149.4 şi ce oferă în plus față de IEEE 1149.1. Din această secțiune rezultă că IEEE 1149.4 nu rezolvă complet problema testării părții analogice, dar poate uşura mult testarea sistemelor electronice complexe.

În secțiunea 2.12 (***Alte standarde de testare***) se prezintă unele standarde suplimentare care completează IEEE 1149.1 şi care extind domeniul de aplicativitate al scanării seriale.

Autorul a considerat necesar acest capitol introductiv pentru a permite cititorului înțelegerea următoarelor capitole şi pentru a se insista asupra unor aspecte teoretice, care vor fi pe larg analizate în capitolele următoare. În capitolul următor se va insista mai mult pe realizarea practică a testării utilizând facilitățile JTAG, pe limbajele folosite la testarea pe frontieră şi se vor prezenta câteva exemple de testare implementate de autor.

Pentru realizarea acestui capitol, autorul a făcut o sinteză a literaturii de specialitate din domeniul testării JTAG, a verificat practic conceptele teoretice importante ale testării JTAG, iar pentru a asigura o înțelegere mai bună a materialului prezentat, autorul a creat toate figurile prezentate în acest capitol.

CAPITOLUL 3

3. CONTRIBUȚII LA TESTAREA ECHIPAMENTELOR ELECTRONICE COMPLEXE UTILIZÂND JTAG

În cadrul capitolului precedent s-a realizat o prezentare sistematică, gradată şi completă a problematicii testării JTAG în strânsă legătură cu obiectivele pe care le-a propus autorul în cercetarea posibilităților de optimizare a testării echipamentelor electronice complexe.

În continuare, în acest capitol autorul va prezenta ce înseamnă testarea echipamentelor numerice complexe utilizând JTAG, după care se va aprofunda testarea JTAG pe o unitate complexă de aviație a controlului încărcăturii la avioanele Boeing ([65], [75], [78], [115], [133], [209], [213], [214]).

3.1. Metode de testare cu JTAG

Pe baza analizei bibliografice autorul a constatat existența a două categorii de echipamente electronice la care este posibilă testarea utilizând JTAG:

- plăci complet testabile cu JTAG;
- plăci parțial testabile cu JTAG.

În continuare, în această secțiune se vor prezenta metodele de test ale plăcilor complet şi parțial testabile JTAG şi se vor prezenta exemple concrete implementate de autorul tezei, care vor permite o mai bună înțelegere a următoarei secțiuni.

3.1.1. Plăci complet testabile cu JTAG

Cel mai fericit caz din punct de vedere al testării este cel al unei plăci sau al unui sistem electronic, care a fost proiectat având în vedere cerințele de testabilitate JTAG. În acest caz, aproape toate nodurile plăcii pot fi controlate cu ajutorul celulelor de testare pe frontieră, fiind necesar numai accesul la conectorul JTAG al plăcii testate. Aceste plăci

permit o testare completă bazată pe JTAG şi chiar generarea automată a vectorilor de test prin softuri adecvate, pe baza modelelor în BSDL ale circuitelor de pe placa de test. În secțiunea 3.2 se va prezenta metoda realizată şi folosită de autorul tezei la testarea unei unități de aviație cu ajutorul testării JTAG, iar pentru detalii suplimentare se pot studia referințele bibliografice ([40], [41], [43], [45], [58], [63], [75], [83], [115], [207]).

3.1.2. Plăci parțial testabile cu JTAG

De obicei, în această categorie de echipamente intră plăci, care prezintă o parte numerică şi o parte analogică (interfețe cu diverşi senzori, circuite de comandă şi control şi de putere). În acest caz, o parte importantă din placă se poate testa utilizând facilitățile oferite de JTAG, dar restul trebuie testat prin metode clasice (testoare cu sonde). Testarea mixtă (JTAG şi clasică) implică cheltuieli mai mici decât testarea clasică şi reprezintă o alternativă convenabilă pentru plăcile care nu au fost proiectate pentru o testabilitate completă cu JTAG. Autorul estimează că numărul plăcilor parțial testabile prin JTAG se va reduce, deoarece, din dorința de a reduce costurile testării, producătorii de echipamente electronice vor reproiecta plăcile electronice pentru a permite testarea completă numai prin JTAG. Din studiul bibliografic se remarcă deja tendința de utilizare a IEEE 1149.4 pentru testarea părții analogice şi abordarea ierarhică a testării IEEE 1149.1 pentru testarea sistemelor electronice complexe. Pentru informații suplimentare se poate studia bibliografia ([25], [45], [52], [89], [131]).

3.1.3. Exemplu de testare JTAG

În figura 3.1. se prezintă schema simplificată a unei plăci electronice testabile cu JTAG. Se poate observa că se utilizează buffere şi latch-uri conectate în serie, iar prin intermediul acestora se asigură citirea şi setarea magistralelor din sistem. Bistabilele şi latch-urile utilizate permit nu doar citirea intrărilor cu ajutorul magistralei JTAG, dar prezintă şi facilități de autotestare (circuitul selectat rulează testul intern şi va indica răspunsul pe JTAG). În concluzie, prin intermediul JTAG, în exemplul din figura 3.1. este posibilă testarea intrărilor/ieşirilor numerice, verificarea conexiunilor electrice ale procesorului, memoriei, ale bufferelor şi latch-urilor şi autotestarea circuitelor JTAG de pe placa electronică.

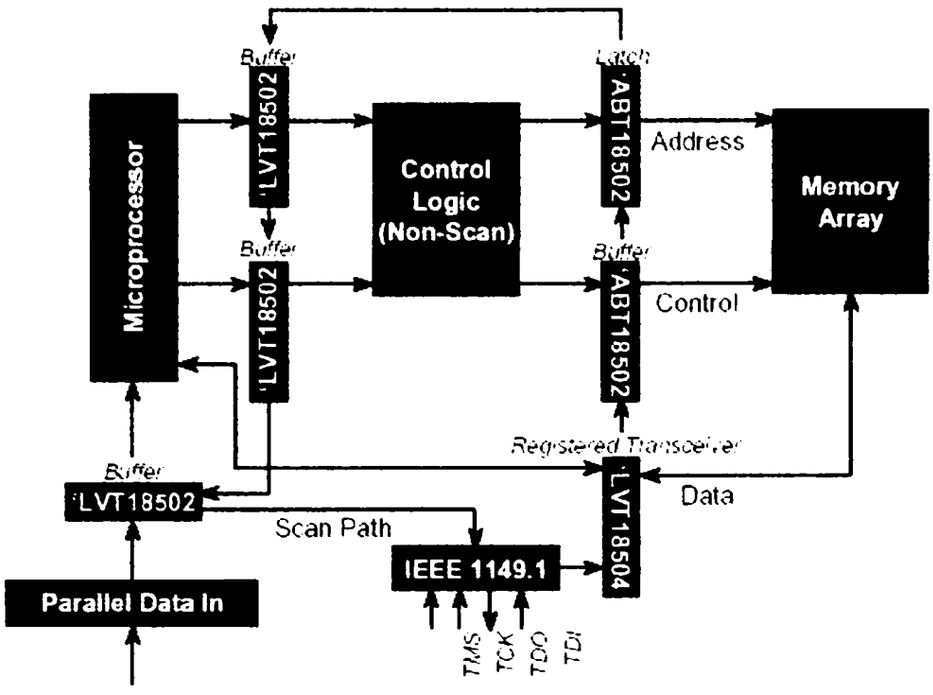
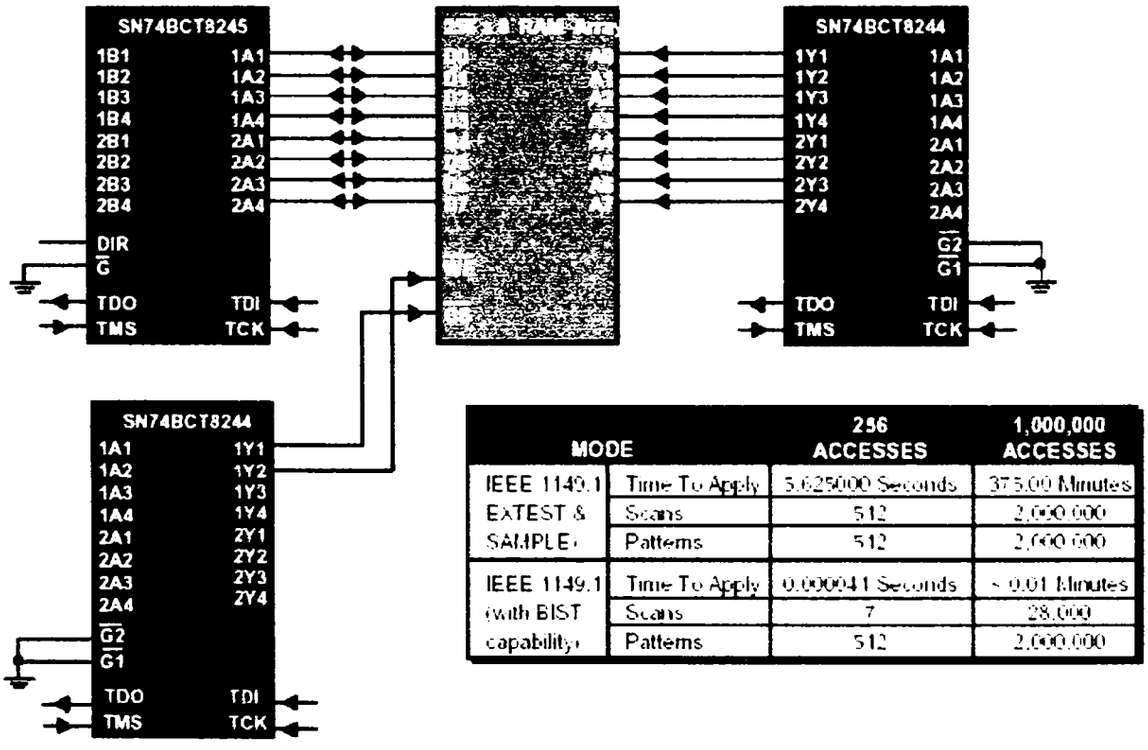


Figura 3.1. Exemplu de testare JTAG

În figura 3.2. se prezintă o posibilitate de testare a unei memorii utilizând JTAG.



MODE		256 ACCESSES	1,000,000 ACCESSES
IEEE 1149.1 EXTEST & SAMPLE	Time To Apply	5.625000 Seconds	375.00 Minutes
	Scans	512	2,000,000
	Patterns	512	2,000,000
IEEE 1149.1 (with BIST capability)	Time To Apply	0.000041 Seconds	< 0.01 Minutes
	Scans	7	28,000
	Patterns	512	2,000,000

Figura 3.2. Exemplu de testare a unei memorii RAM

Din figura 3.2. se poate observa că:

- testarea memoriei se poate face utilizând scanarea serială a bistabilelor conectate pe pinii memoriei;

- vectorii de test se pot genera automat pe baza modelelor circuitelor utilizate;
- se poate utiliza şi autotestarea (BIST) circuitelor de pe placă pentru reducerea timpului de testare.

În explicațiile din figura 3.2. sunt prezentate rezultatele celor două moduri de testare posibile:

- testarea prin scanare serială (utilizând instrucțiunile EXTEST şi SAMPLE). Această metodă este foarte flexibilă, dar necesită un timp de testare adeseori inacceptabil de lung;
- testarea utilizând instrucțiunea de autotestare (BIST). Această metodă este foarte rapidă, dar nu toate circuitele au implementată instrucțiunea BIST.

Un avantaj important al testării prin scanare serială este acela că se poate extinde foarte uşor la nivel de sistem. De exemplu, în figura 3.3. se prezintă un aparat care conține mai multe plăci cu facilități de testare JTAG.

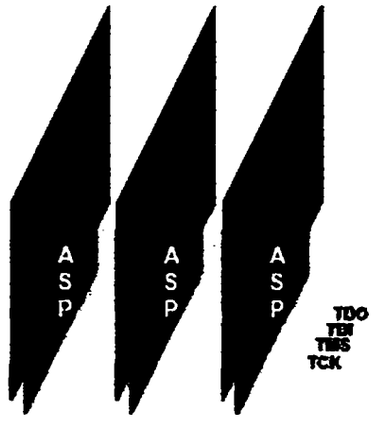


Figura 3.3. Testarea la nivel de sistem

Metoda din figura 3.3 va asigura:

- posibilitatea de testare a fiecărei plăci electronice în parte;
- testarea la nivel de sistem electronic;
- testarea proiectului la nivel de sistem;
- self-testul sistemului;
- generarea automată a vectorilor de test prin utilizarea modelelor în BSDL ale circuitelor integrate şi ale plăcilor electronice testate.

3.2. Contribuții la testarea cu JTAG. Testarea unei unități de aviație de control al încărcăturii cu JTAG.

În această secțiune se va prezenta metoda de testare JTAG realizată de autorul tezei şi folosită la testarea unei unități de aviație Boeing 777 de control al încărcăturii (CSC). Pentru înțelegerea metodei de test dezvoltate este necesară prezentarea în prealabil a unității testate, a stației de test utilizate, a metodei de comunicare pe JTAG şi a softurilor utilizate la testarea JTAG.

3.2.1. Prezentarea generală a unității testate

Unitatea de control a încărcăturii (CSC) conține următoarele module:

- *placa de alimentare şi unitatea centrală*. Placa de alimentare converteşte tensiunea de alimentare de 28 VDC în 5 VDC şi +-15 DC, alimentând şi celelalte plăci. Unitatea centrală este realizată cu două circuite FPGA Xilinx 4010 cu facilități de testare JTAG, asigură numărul de intrări/ieşiri necesare şi permite testarea prin JTAG a celorlalte plăci.
- *placa de comutare de curent alternativ*. Aceasta comandă 8 motoare trifazate de 115 VAC, 8 motoare liniare şi două bobine de frânare. În total, se utilizează 56 de relee, care permit controlul şi izolarea galvanică a liniilor de putere de curent alternativ.
- *placa de curent continuu*. Placa prezintă 85 de ieşiri discrete, care controlează dispozitivele de comandă din camera bagajelor (electromagneți, relee, LED-uri şi motoare de frânare). Toate ieşirile de forță sunt comutate de tranzistoare MOS de putere.

Pentru realizarea standului de testare a acestei unități de o complexitate relativ ridicată şi a cărei siguranță în funcționare este foarte importantă, autorul a folosit o stație specială de testare (SMART-2) - produsă de firma RADA din Israel - care va fi prezentată pe scurt în continuare ([65], [177], [178], [217]).

3.2.2. Structura stației SMART-2

Din schema bloc a stației SMART-2 prezentată în figura 3.4 se poate observa că stația conține următoarele aparate conectate pe GPIB:

- sursă de alimentare trifazată programabilă;

- sursă programabilă de alimentare de curent continuu;
- o matrice de comutatoare programabilă;
- multimetru numeric;
- osciloscop;
- frecvențmetru;
- calculator compatibil PC;
- generator de funcții programabil;
- sertar VXI;
- placă de scanare pe frontieră (Tektronix, VX4491);
- plăci de comunicație pe ARINC429 şi ARINC629.

Calculatorul compatibil PC comandă aparatele menționate prin intermediul unei magistrale GPIB, iar sistemele de operare posibile sunt: DOS, UNIX şi WINDOWS NT/XP.

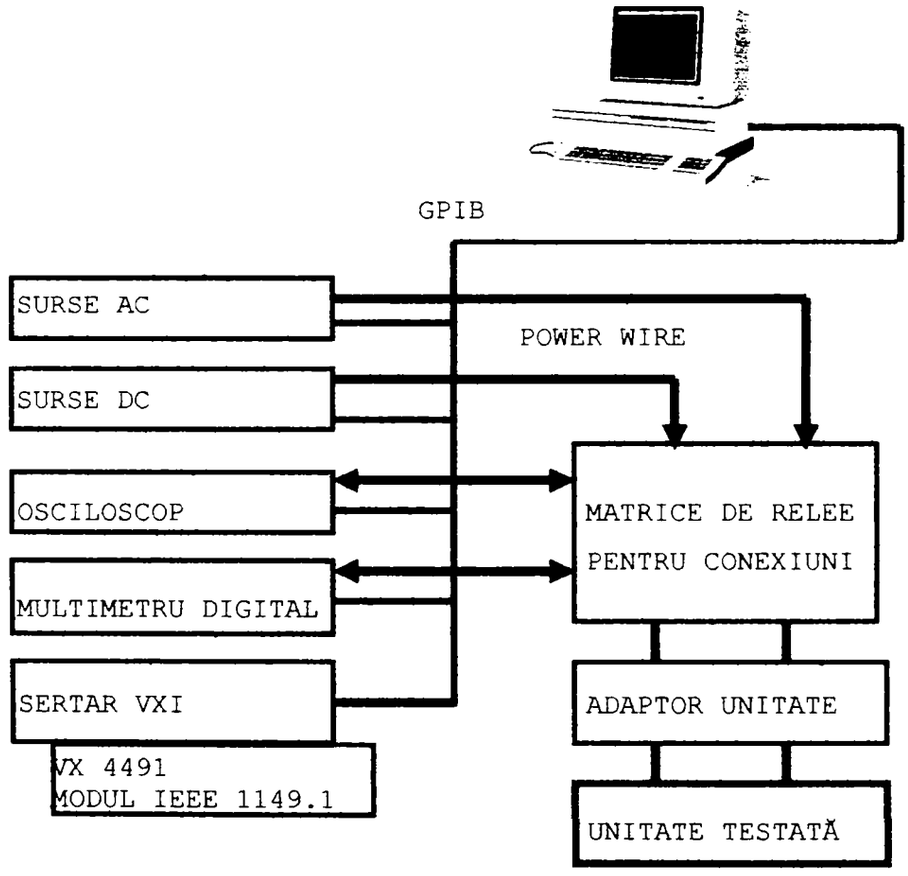


Figura 3.4. Prezentarea stației SMART-2

Din figura 3.4. se poate observa că toate echipamentele stației SMART-2 sunt programabile prin magistrala GPIB. Se mai poate observa că UNITATEA TESTATĂ este conectată la UNITATEA DE ADAPTARE, iar aceasta din urmă este conectată la MATRICEA DE CONEXIUNI programabile. Matricea de conexiuni permite realizarea a

zeci de conexiuni paralele între echipamentele stației şi unitatea de adaptare, fiind esențială în asigurarea flexibilității stației SMART-2.

Unitatea de adaptare este necesară din următoarele considerente:

- rezolvă problema incompatibilității conectorilor unităților testate;
- permite adăugarea unui hardware suplimentar pentru a simula condițiile reale de funcționare a unității testate (sarcini rezistive, sarcini inductive, hardware de interfață pentru interfețele de comunicație, amplificatoare etc);
- asigură flexibilitatea stației de testare. Astfel, fiecare unitate testată va avea o unitate de adaptare corespondentă, dar restul stației de test rămâne neschimbat.

Stația SMART-2 (figura 3.5.) s-a dovedit de-a lungul timpului foarte eficientă pentru testarea modulelor utilizate în aviația civilă, prezentând şi posibilități de extensie remarcabile. În cazul unității CSC adaptorul de unitate şi matricea de conexiuni asigură alimentarea adecvată a unității, conectarea la tensiuni corespunzătoare a unor pini de selecție ai unității CSC, conectarea instrumentelor de test şi a plăcii JTAG.

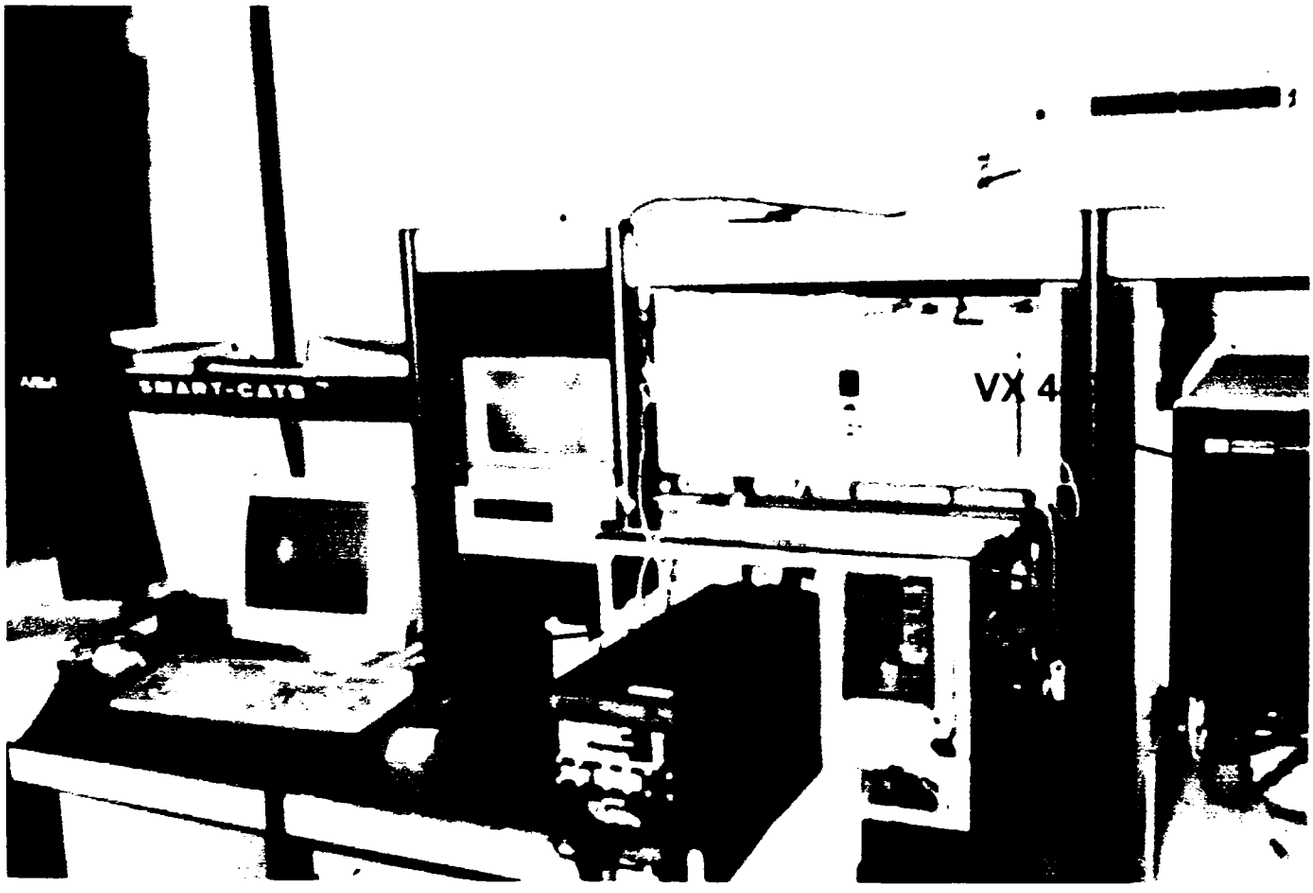


Figura 3.5. Stația SMART-2 şi unitatea testată (CSC)

În figura 3.5. se poate observa că unitatea de control a încărcăturii (CSC) este conectată la unitatea de adaptare (TUA), iar TUA este conectată la SWITCH MATRIX. La o analiză mai atentă a staţiei SMART-2 se observă că toate echipamentele şi instrumentele de măsură din structura staţiei sunt configurabile, programarea acestora realizându-se prin intermediul magistralei GPIB de către softul disponibil pe calculatorul PC. Suplimentar staţia dispune şi de un sertar VXI în care este introdusă o placă de interfaţă JTAG.

3.2.3. Conectarea unităţii de control a încărcăturii pe JTAG.

În această secţiune se analizează două metode de comunicaţie pe magistrala JTAG între unitatea de control a încărcăturii şi calculatorul PC al staţiei. S-a considerat necesară introducerea acestui aspect în teză, deoarece informaţiile din bibliografie despre acest subiect sunt foarte sumare, iar autorul tezei a trebuit să aprofundeze practic acest domeniu.

3.2.3.1. Simularea magistralei JTAG pe portul paralel al calculatoarelor PC

Deoarece la începutul punerii în funcţiune a unităţii de control a încărcăturii (CSC) au existat probleme de comunicaţie cu placa JTAG (Tektronix - VX 4491), autorul a simulat magistrala JTAG pe portul paralel al calculatorului PC (figura 3.6). Deoarece nivelele logice pe magistrala IEEE 1149.1 sunt compatibile TTL, iar magistrala IEEE 1149.1 este una sincronă, rezultă că din punct de vedere hardware prin intermediul portului paralel se poate comanda orice magistrală JTAG.

Pentru a asigura protocolul de comunicaţie conform specificaţiilor JTAG, autorul a dezvoltat un program în C++, codul sursă fiind prezentat în Anexa B. Programul conţine: o funcţie care permite resetarea soft a circuitului conectat pe magistrala JTAG (RESET), o funcţie care permite trimiterea instrucţiunii în registrul de instrucţiune (INSTRUCTION) şi o funcţie (LOAD) care este utilizată pentru încărcarea în registrul de date al circuitului conectat pe bus a fişierului de intrare specificat pe linia de comandă. Programul scrie în fişierul cu numele specificat pe linia de comandă datele citite de pe magistrala 1149.1.

Metoda de simulare a magistralei JTAG pe portul paralel, prezentată anterior, este accesibilă şi ieftină, putând fi utilizată pentru comunicaţia cu orice echipament JTAG, dar

prezintă performanțe de viteză cu una-două ordine de mărime mai modeste decât în cazul utilizării unei plăci JTAG profesionale.

Se impune a se sublinia că programul şi schema electronică prezentate în acest paragraf sunt contribuții originale ale autorului, care au fost verificate şi utilizate practic.

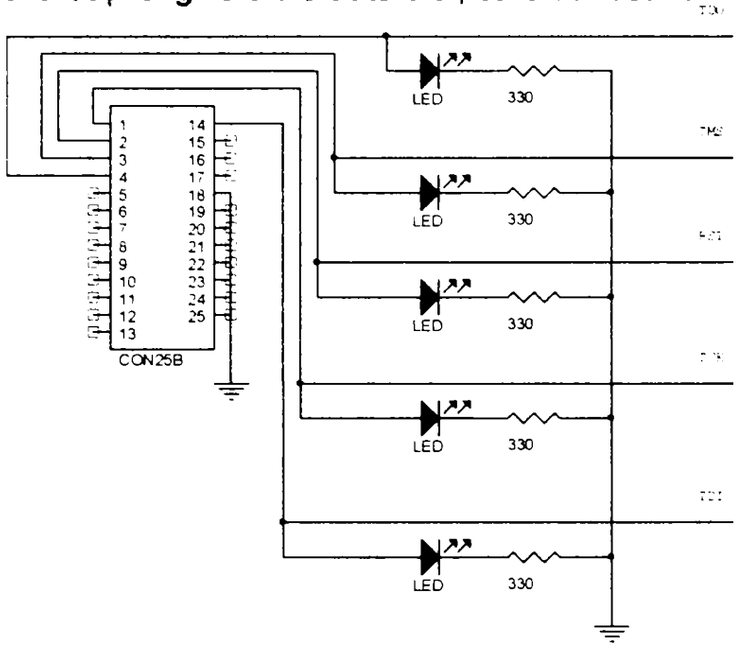


Figura 3.6. Obținerea magistralei IEEE 1149.1 de pe portul paralel al unui calculator PC

3.2.3.2. Conectarea JTAG cu Tektronix VX4491

În secțiunea precedentă s-a prezentat o metodă de comunicație pe magistrala JTAG a unității de control a încărcăturii de pe portul paralel al unui calculator compatibil PC. Metoda prezentată anterior satisface cerințele IEEE 1149.1, dar este relativ lentă datorită vitezei reduse de comunicație pe portul paralel. Pentru creşterea vitezei de lucru şi implementarea unei soluții "standard" în domeniul testării JTAG s-a folosit un modul JTAG produs de Tektronix (VXI4491). Acesta prezintă următoarele caracteristici:

- se introduce într-un sertar VXI;
- protocolul de comunicație este Word Serial Fast Data Channel, REV 1.0;
- prezintă nivel de întrerupere programabil 1-7;
- adresa de comunicație GPIB se stabileşte printr-un microîntrerupător;
- prezintă intrări de sincronizare;
- prezintă un port paralel de ieşire de 8 biți TTL-ECL;
- frecvența semnalului de tact TCK este programabilă (dacă este generată intern), frecvența maximă fiind de 25 MHz;
- semnalul de tact utilizat TCK poate fi intern sau extern;

- softul implementat permite o serie de operaţii directe (compararea vectorului aşteptat cu cel citit, memorarea mai multor vectori de test şi retrimiterea lor printr-o simplă selecţie);
- modulul este livrat cu un soft (Serial Toolbox) care permite generarea, trimiterea şi compararea vectorilor de test;
- o caracteristică negativă este aceea că placa face o iniţializare a magistralei IEEE 1149.1 la fiecare trimitere a unui vector de test, ceea ce uneori poate crea complicaţii suplimentare.

Deoarece pe calculatorul staţiei SMART-2 se utiliza o versiune de UNIX (UNIXWARE), autorul a dezvoltat driver-ul în mediul dezvoltare PAWS (produs de firma TYX), inclusiv pentru placa VX4491, iar o parte din codul sursă este prezentat în Anexa C.

În urma utilizării practice de către autorul tezei a simulării magistralei JTAG pe portul paralel al unui calculator PC (paragraful 3.2.3.1) şi a unei unităţi dedicate JTAG VX4491 au rezultat următoarele:

- este posibilă simularea magistralei JTAG pe portul paralel al unui calculator PC independent de complexitatea conectării circuitelor JTAG din lanţ şi de numărul acestora;
- dependent de performanţele calculatorului şi de sistemul de operare utilizat la simularea pe portul paralel al magistralei JTAG, frecvenţa maximă a semnalul de tact generabil este limitată în jur de 10 KHz. În cazul testării unităţii de control a încărcăturii Boeing 777 această limitare este acceptabilă întrucât transmiterea celui mai complex vector de test de 976 stari biţi s-a realizat în mai puţin de 0.5 secunde, adică acceptabil în acest proiect, dar inacceptabil de exemplu la testarea în serie mare cum ar fi cea din producţie;
- întrucât pe staţia de test SMART-2 interfaţa IEEE 1149.1 VX4491 este conectată la calculatorul de test indirect prin intermediul sertarului VXI şi prin magistrala GPIB, transferarea vectorilor de test este relativ lentă în comparaţie cu transferul prin interfaţa paralelă. În aceste condiţii, la proiectarea unei staţii de test este recomandabilă analiza prealabilă atentă a necesităţilor de transfer de date între echipamente şi alegerea soluţiei adecvate (conectarea directă prin GPIB, conectarea în reţea sau prin USB) [241], [245]).

- echipamentele dedicate de interfațare cu magistrala JTAG (de exemplu VX4491) permit utilizarea unui semnal de tact cu frecvența maximă de zeci de MHz, sunt relativ complexe, conțin memorii tampon de date dimensiuni relativ importante pentru vectorii de test şi acceptă o serie de comenzi complexe de la calculatorul cu care sunt conectate. Astfel în cazul utilizării unor astfel de echipamente procedura de testare se poate optimiza, adică la începutul testelor se pot încărca vectorii de test, vectorii de răspuns corecți şi programul care urmează să se ruleze în memoria tampon a echipamentului, după care intern în placa de interfață se rulează testele necesare, iar la final aceasta furnizează calculatorului de test rezultatul testului;
- frecvent echipamentele dedicate de interfațare cu magistrala JTAG dispun de mai multe canale fiind astfel posibilă rularea testelor în paralel pe mai multe magistrale JTAG.

3.2.4. Limbaje utilizate pentru IEEE 1149.1

Standardul IEEE 1149.1 a fost aprobat în anul 1990 ca urmare a cerinței unora dintre marii producători mondiali de echipamente electronice de a avea o metodă standard de testare, dar ulterior a apărut şi necesitatea unui limbaj pentru descrierea modului în care se execută scanarea serială şi testarea pe frontieră. În consecință, a fost înființat un subcomitet, care s-a ocupat de dezvoltarea unui limbaj de testare, rezultatul fiind BSDL (Boundary-Scan Description Language), adoptat ca şi parte a standardului IEEE 1149.1b-1994. Treptat, tot mai mulți producători de circuite integrate furnizează şi codul sursă (în BSDL, VHDL sau SVF) al circuitelor produse, ceea ce determină o scădere semnificativă a timpului de dezvoltare a testului şi deci a costurilor testării.

Teradyne [219], [250] estima că realizarea vectorilor de test pentru un microprocesor poate dura şase săptămâni:

- 1 săptămână - studierea circuitului;
- 4 săptămâni - dezvoltarea vectorilor de test;
- 2 săptămâni - verificarea vectorilor de test.

Costul estimat este în jur de 14.000 $.

Dacă microprocesorul suportă IEEE 1149.1, iar BSDL de test este furnizat de producător, timpul de dezvoltare a vectorilor de test este mai scurt de două ore, iar costurile aferente sunt mai mici de 100 $.

Datorită importanței limbajului de descriere a testării JTAG, în continuare, în această secțiune sunt prezentate pe scurt cele mai răspândite limbaje suportate de echipamentele de testare IEEE 1149.1 (BSDL, HSDL şi SVF) şi câteva exemple create de autorul tezei. Este necesar a se menționa că această scurtă prezentare a limbajelor utilizate la testarea JTAG este necesară pentru înțelegerea următoarelor secțiuni şi capitole din cadrul tezei.

3.2.4.1 Limbajul BSDL (Boundary-Scan Description Language)

3.2.4.1.1. Introducere

Caracteristici:

- BSDL este un limbaj standard de descriere a părții JTAG implementate în circuit;
- BSDL a fost adoptat ca o componentă a IEEE 1149.1 în 1140.1b-1994;
- BSDL este un subset al standardului VHDL (VHSIC Hardware Description Language);
- BSDL este utilizat ca format standard de intrare la diverse instrumente de testare, cum ar fi ATPG şi ATE şi pentru generarea automată a vectorilor de test;
- BSDL este formatul de ieşire a multor softuri de proiectare pentru testarea automată.

3.2.4.1.2. Elementele BSDL

Limbajul BSDL operează cu următoarele elemente:

- descrierea entității;
- parametri generici;
- descrierea portului logic;
- declarația "use";
- declarația de conformitate a componentelor;
- maparea pinilor;
- identificarea portului de scanare;
- descrierea registrului de instrucțiune;
- descrierea registrilor opționali;
- descrierea registrului de acces;
- descrierea registrului de scanare serială.

3.2.4.1.3. Descrierea entității

Declarația "**entity**" denumeşte entitatea respectivă, iar numele coincide de regulă cu numele circuitului (de exemplu SN74BCT8245A).

Sintaxă:

```
entity XZY is
{linii care descriu entitatea XZY/circuitul}
end XZY
```

3.2.4.1.4. Parametri generici

Parametri generici sunt parametri care pot proveni din exteriorul entității sau care pot fi inițializați cu valori implicite (de exemplu "DW").

Sintaxă:

```
generic (PINUL_FIZIC: string :="DW");
```

3.2.4.1.5. Descrierea portului logic

Aceasta permite definirea de nume logice pentru pinii de intrare/ieşire şi specificarea naturii acestora (intrări, ieşiri, bidirecționali sau de conectare (pinii de alimentare)).

Sintaxă:

```
port (OE:in bit;
Y:out bit_vector(1 to 3);
A:in bit_vector(1 to 3);
GND, VCC, NC:linkage bit;
TDO:out bit;
TMS, TDI, TCK:in bit);
```

3.2.4.1.6. Instrucțiunea USE

Aceasta permite specificarea definițiilor externe şi a bibliotecilor care se utilizează.

Sintaxă:

```
use STD_1149_1_1994.all;
```

3.2.4.1.7. Declarația de conformitate a componentei

Aceasta indică în ce categorie de conformitate IEEE 1149.1 în care se încadrează circuitul.

Sintaxă:

```
attribute COMPONENT_CONFORMANCE of XZY: entity is "STD_1149_1_1993"
```

3.2.4.1.8. Maparea pinilor

Prin maparea pinilor se asigură asignarea semnalelor la pinii fizici ai circuitului. Sintaxă:

```
attribute PIN_MAP of XYZ : entity is
PHYSICAL_PIN_MAP;
constant DW:PIN_MAP_STRING:=
"OE:1, Y:(2,3,4), A:(5,6,7), GND:8, VCC:9, " &
"TDO:10, TDI:11, TMS:12, TCK:13, NC:14";
```

3.2.4.1.9. Identificarea portului de scanare

Prin intermediul acestui element se definesc caracteristicile TAP.

Sintaxă:

```
attribute TAP_SCAN_IN of TDI : signal is TRUE;
attribute TAP_SCAN_OUT of TDO : signal is TRUE;
attribute TAP_SCAN_MODE of TMS : signal is TRUE;
attribute TAP_SCAN_CLOCK of TCK : signal is
(50.0e6, BOTH);
```

3.2.4.1.10. Definirea registrului de instrucțiune

Permite specificarea caracteristicilor registrelor specifice IEEE 1149.1 utilizate.

Sintaxă:

```
attribute INSTRUCTION_LENGTH of XYZ : entity is 2;
attribute INSTRUCTION_OPCODE of XYZ : entity is
"BYPASS (11),"&
"EXTEST (00),"&
"SAMPLE (10),"&
"IDCODE (01)"
attribute INSTRUCTION_CAPTURE of XYZ : entity is
"01";
```

3.2.4.1.11. Descrierea registrelor suplimentare

Permite specificarea valorilor opționale din registrul **IDCODE** şi **USERCODE**.

Sintaxă:

```
attribute IDCODE_REGISTER of XYZ : entity is
"01010100000011111100000000101111";
```

3.2.4.1.12. Descrierea registrelor de acces

Această instrucțiune defineşte pentru fiecare instrucțiune care registre sunt plasate între **TDI** şi **TDO**.

Sintaxă:

```
attribute REGISTER_ACCESS of XYZ : entity is
"BOUNDARY (EXTEST, SAMPLE),"&
"BYPASS (BYPASS)";
```

3.2.4.1.13. Descrierea registrelor de frontieră

Această instrucțiune permite specificarea pentru fiecare celulă de scanare pe frontieră a tipului şi a controlului asociat.

Sintaxă:

```
attribute BOUNDARY_LENGTH of XYZ : entity is 7;
attribute BOUNDARY_REGISTER of XYZ : entity is
"0 (BC_1, Y(1), output3, X, 6, 0, Z),"&
"1 (BC_1, Y(2), output3, X, 6, 0, Z),"&
"2 (BC_1, Y(3), output3, X, 6, 0, Z),"&
"3 (BC_1, A(1), input , X),"&
"4 (BC_1, A(2), input , X),"&
"5 (BC_1, A(3), input , X),"&
"6 (BC_1, OE , input , X),"&
"6 (BC_1, * , control, 0)";
```

3.2.4.1.14. Verificarea acurateței BSDL

După realizarea modelului BSDL şi a verificării sintactice şi semantice este necesară o validare suplimentară, în siliciu fiind posibilă apariția următoarelor erori:

- pini greşiți;
- celule greşite;
- ordinea greşită a registrelor de scanare;
- lungime greşită a registrelor de scanare;
- cod greşit al registrului de instrucțiune;
- locație greşită de control;
- valoare greşită de invalidare a celulei de control;
- celule de control greşite pentru I/O;
- valoare greşită a registrului de identificare;
- valoare greşită în registrul de captură.

3.2.4.1.15. Scurt exemplu de program scris în BSDL

În continuare se prezintă un program pe care autorul tezei l-a scris în BSDL pentru testarea pe frontieră a circuitului 74BCT8373. Acest circuit este de fapt un circuit clasic 74LS373, la care a fost adăugată partea JTAG.

```
entity sn74bct8373 is
generic (PHYSICAL_PIN_MAP : string := "UNDEFINED");

port (LE:in bit;
    Q:out bit_vector(1 to 8);
    D:in bit_vector(1 to 8);
    GND, VCC:linkage bit;
    OE_NEG:in bit;
    TDO:out bit;
    TMS, TDI, TCK:in bit;
    NC:linkage bit_vector(1 to 4));

use STD_1149_1_1990.all; -- Get Standard attributes and definitions

attribute PIN_MAP of sn74bct8373 : entity is PHYSICAL_PIN_MAP;

constant JT : PIN_MAP_STRING := "LE:1, Q:(2,3,4,5,7,8, " &
              "9,10), D:(23,22,21,20,19,17,16,15), " &
              "GND:6, VCC:18, OE_NEG:24, TDO:11, " &
              "TMS:12, TCK:13, TDI:14";

constant DW : PIN_MAP_STRING := "LE:1, Q:(2,3,4,5,7,8, " &
              "9,10), D:(23,22,21,20,19,17,16,15), " &
              "GND:6, VCC:18, OE_NEG:24, TDO:11, " &
              "TMS:12, TCK:13, TDI:14";

constant NT : PIN_MAP_STRING := "LE:1, Q:(2,3,4,5,7,8, " &
              "9,10), D:(23,22,21,20,19,17,16,15), " &
              "GND:6, VCC:18, OE_NEG:24, TDO:11, " &
              "TMS:12, TCK:13, TDI:14";

constant FK : PIN_MAP_STRING := "LE:9, Q:(10,11,12," &
              "13,16,17,18,19), D:(6,5,4,3,2,27,26,25), " &
              "GND:14, VCC:28, OE_NEG:7, TDO:20, " &
              "TMS:21, TCK:23, TDI:24, NC:(1,8,15,22)";

attribute TAP_SCAN_IN    of TDI : signal is true;
attribute TAP_SCAN_MODE  of TMS : signal is true;
attribute TAP_SCAN_OUT   of TDO : signal is true;
attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);
attribute INSTRUCTION_LENGTH of sn74bct8373 : entity is 8;

attribute INSTRUCTION_OPCODE of sn74bct8373 : entity is
      "BYPASS (11111111, 10001000, 00000101, 10000100, 00000001), " &
      "EXTEST (00000000, 10000000)," &
      "SAMPLE (00000010, 10000010)," &
      "INTEST (00000011, 10000011)," &
      "HIGHZ  (00000110, 10000110)," &   -- Bypass with outputs high-z
      "CLAMP  (00000111, 10000111)," &   -- Bypass with bs values
      "RUNT   (00001001, 10001001)," &     -- Boundary run test
      "READBN (00001010, 10001010)," &   -- Boundary read normal mode
      "READBT (00001011, 10001011)," &   -- Boundary read test mode
      "CELLTST(00001100, 10001100)," &   -- Boundary selftest normal
```

```
        "TOPHIP (00001101, 10001101)," &   -- Boundary toggle out test
        "SCANCN (00001110, 10001110)," &   -- BCR scan normal
        "SCANCT (00001111, 10001111)"  ;   -- BCR scan test
    attribute INSTRUCTION_CAPTURE of sn74bct8373 : entity is "10000001";
    attribute INSTRUCTION_DISABLE of sn74bct8373 : entity is "HIGHZ";
    attribute INSTRUCTION_GUARD   of sn74bct8373 : entity is "CLAMP";

    attribute REGISTER_ACCESS of sn74bct8373 : entity is
        "BOUNDARY (EXTEST, SAMPLE, INTEST, READBN, READBT, CELLTST)," &
        "BYPASS   (BYPASS, HIGHZ, CLAMP, RUNT, TOPHIP)," &
        "BCR[2]   (SCANCN, SCANCT)" ;

    attribute BOUNDARY_CELLS    of sn74bct8373 : entity is "BC_1";
    attribute BOUNDARY_LENGTH   of sn74bct8373 : entity is 18;

    attribute BOUNDARY_REGISTER of sn74bct8373 : entity is
    " 0 (BC_1, Q(8)   ,output3, X, 16, 1, Z), " &
    " 1 (BC_1, Q(7)   ,output3, X, 16, 1, Z), " &
    " 2 (BC_1, Q(6)   ,output3, X, 16, 1, Z), " &
    " 3 (BC_1, Q(5)   ,output3, X, 16, 1, Z), " &
    " 4 (BC_1, Q(4)   ,output3, X, 16, 1, Z), " &
    " 5 (BC_1, Q(3)   ,output3, X, 16, 1, Z), " &
    " 6 (BC_1, Q(2)   ,output3, X, 16, 1, Z), " &
    " 7 (BC_1, Q(1)   ,output3, X, 16, 1, Z), " &
    " 8 (BC_1, D(8)   ,input   , X)," &
    " 9 (BC_1, D(7)   ,input   , X)," &
    "10 (BC_1, D(6)   ,input   , X)," &
    "11 (BC_1, D(5)   ,input   , X)," &
    "12 (BC_1, D(4)   ,input   , X)," &
    "13 (BC_1, D(3)   ,input   , X)," &
    "14 (BC_1, D(2)   ,input   , X)," &
    "15 (BC_1, D(1)   ,input   , X)," &
    "16 (BC_1, OE_NEG,input    , X)," & -- merged Input/Control cell
    "16 (BC_1, *      ,control, 1)," & -- merged Input/Control cell
    "17 (BC_1, LE     ,input   , X) " ;

end sn74bct8373;
```

Din modelul BSDL prezentat anterior se poate observa că acesta conține la început definirea pinilor circuitului ("port"), urmată de maparea acestora şi maparea instrucțiunilor JTAG ("attribute"), iar în final se definesc regiştri utilizați. Modelul astfel rezultat este relativ simplu, descrie complet funcționarea circuitului "sn74bct8373" din punct de vedere al testării IEEE 1149.1 şi poate fi reutilizat pentru alte circuite din această familie.

În ultimul timp tot mai mulți producători de circuite integrate numerice fac publice (uneori şi pe Internet) modelele în BSDL şi VHDL ale circuitelor produse, ceea ce reduce semnificativ timpul de dezvoltare a mediului de testare şi contribuie la creşterea calității testării, precum şi la reducerea costurilor asociate.

3.2.4.2. Limbajul HSDL (Hierarchical Scan Description Language)

3.2.4.2.1. Introducere

Caracteristici:

- HSDL descrie modul în care circuitele IEEE 1149.1 sunt conectate la nivel de placă sau sistem;
- HSDL descrie atributele suplimentare IEEE 1149.1. pe care BSDL nu le suportă;
- HSDL a fost dezvoltat de Texas Instruments pentru circuitele/domeniul ASSET, dar actualmente este utilizat de compania ASSET InterTech, care a cumpărat ASSET în anul 1995.

3.2.4.2.2. Elementele HSDL

HDSL adaugă noi elemente la BSDL pentru descrierea membrilor, a căilor de scanare, a modulelor şi pentru a simplifica utilizarea interactivă. Elementele HDSL sunt:

- descrierea entității;
- parametri generici;
- descrierea portului logic;
- instrucțiunea "USE";
- [descrierea modulelor - opțional];
- [descrierea portului - opțional];
- maparea pinilor;
- identificarea portului de scanare;
- [descrierea membrilor - opțional];
- [descrierea compoziției bus-ului - opțional];
- descrierea căii;
- [conexiunea membrilor - opțional];
- [descrierea constrângerilor - opțional];
- [avertismente de proiectare - opțional].

3.2.4.2.3. Descrierea membrilor (opțională)

Membrii reprezintă circuite sau alte module care sunt componente ale altor module. De obicei, membrii sunt componente, dar unele plăci pot conține plăci mamă care necesită descrierea modulelor membre.

Sintaxă:

attribute MEMBERS of BOARD : entity is

```
"U1 (XYZ1, DW),"&
"U2 (XYZ2, DW)";
```

3.2.4.2.4. Descrierea componentelor de pe magistrală (opțională)

Magistralele în HSDL pot fi compuse din magistralele modulelor, ale membrilor şi ale circuitelor.

Sintaxă:

```
attribute BUS_COMPOSITION of BOARD : entity is
"bus1[4] (U1.Boundary[3,0]),"&
"bus2[4] (U2.Boundary[3,0])";
```

3.2.4.2.5. Descrierea căilor (opțională)

Aceasta descrie conexiunile TAP pe placă.
Sintaxă:

```
constant boardpath1 : STATIC_PATH :=
"U1, U2";
end BOARD;
```

3.2.4.3. Prezentarea formatului SVF (Serial Vector Format)

Formatul SVF a fost dezvoltat de Texas Instruments şi Teradyne în 1991 ([146], [147]) şi prezintă un format ASCII standard care specifică stimulii, răspunsul şi masca de date conform specificațiilor standardului IEEE 1149.1

SVF a apărut din necesitatea de a avea un standard independent de producător, transportabil pe mai multe platforme şi medii de simulare şi testare, de la verificarea proiectului până la diagnoză. Execuția testării pe frontieră este controlată prin secvența semnalelor TAP trimise la pinii circuitelor.

Specific pentru SVF este că magistrala IEEE 1149.1 nu se descrie pentru fiecare ciclu de tact, limbajul SVF permițând descrierea doar a tranzițiilor între diverse stări stabile. De exemplu, procesul de scanare în registrul de instrucțiuni este descris prin datele implicate şi prin stările stabile în care se intră, după ce s-a făcut scanarea. Stările de **Capture**, **Pauze** sau **Update** sunt mai degrabă implicite decât reprezentate explicit. SVF suportă conceptul de deplasament de scanare, care permite scanarea uşoară a componentelor din cadrul căii de scanare.

În următoarele subcapitole sunt prezentate comenzile specifice ale limbajului SVF.

3.2.4.3.1. Comanda ENDDR, ENDIR

Sintaxă:

ENDDR stare-stabilă

ENDIR stare-stabilă

Această comandă stabileşte starea finală la operaţia de scanare.

Exemplu:

ENDIR IDLE;

ENDDR DRPAUSE;

3.2.4.3.2. Comanda HDR, HIR

Sintaxă:

HDR lungime[TDI][TDO][MASK][SMASK]

HIR lungime[TDI][TDO][MASK][SMASK]

Această comandă specifică headerul care este introdus înaintea datelor de scanare la fiecare operaţie de scanare. Ea este utilă la scanarea în cadrul unui mediu, unde calea de scanare este compusă din mai multe circuite, care corespund standardului IEEE 1149.1.

- Lungime - un număr întreg pe 32 de biţi, care specifică numărul biţilor ce se vor scana;
- TDI - (opţional). Se specifică datele, care se vor introduce în circuitul ţintă, exprimat în hexa. Dacă acest parametru nu este prezent, se consideră valorile definite anterior.
- TDO - (opţional). Se specifică valorile cu care să fie comparate datele citite. Dacă acest parametru nu este prezent, nu se face operaţia de comparaţie.
- MASK - Defineşte masca utilizată la compararea datelor citite cu cele aşteptate. Dacă un bit dintr-o poziţie oarecare este pe "0", înseamnă că el nu contează în cadrul procesului de scanare, deci nu este luat în considerare.
- SMASK - Defineşte care dintre datele TDI nu prezintă importanţă. Dacă un bit oarecare este pe "1", înseamnă că poziţia respectivă din TDI contează.

3.2.4.3.3. Comanda PIO

Sintaxă:

PIO (vector_string)

Specifică un vector de test aplicat paralel.

Vector-string: reprezintă un set de caractere, care specifică direcţia şi starea pentru un pin specific corespunzător unui vector de test.

3.2.4.3.4. Comanda PIOMAP

Sintaxă:

PIOMAP ([coloana1 nume_logic].. [coloana1 nume_logic]);

Comanda asigură legătura dintre o coloană specifică şi numele logic asociat cu acea coloană. Această comandă este opțională.

Exemplu:

```
PIOMAP (    1 STROBE
            2 ALE
            3 DISABLE
            4 ENABLE
            5 CLEAR
            6 SET);
PIO (HLUDX2)
```

3.2.4.3.5. Comanda RUNTEST

Sintaxă:

RUNTEST nr_tact sel_tck [ENDSTATE end_state]

Comanda determină execuția în starea **Run Test-Idle** pentru un număr de impulsuri de tact specificate, după care se trece în starea specificată:

- nr_tact - numărul de perioade de tact în care busul rămâne în starea Run
- Test_Idle;
- sel_tck - specifică semnalul de tact utilizat TCK (Test Clock) sau SCK (System Clock);
- ENDSTATE end_state - este opțional şi specifică starea în care se forțează magistrala 1149.1, după ce s-a executat numărul specificat de impulsuri de tact.

```
RUNTEST 1000 TCK ENDSTATE DRPAUSE;
RUNTEST 20 SCK;
```

3.2.4.3.6. Comanda SDI, SIR (Scan Data Register, Scan Instruction Register)

Sintaxă:

SDR lungime[TDI] [TDO] [MASK] [SMASK] [PIO];

SIR lungime [TDI] [TDO] [MASK] [SMASK] [PIO];

- Lungime - un număr întreg pe 32 de biți, care specifică numărul de biți ce vor fi scanați;

- TDI - specifică datele ce vor fi introduse în circuitul ţintă şi sunt exprimate în hexa. Dacă acest parametru nu este prezent, valoarea introdusă în circuitul ţintă este ultima introdusă cu o comandă SDI-SIR.
- TDO - specifică valorile cu care să fie comparate datele citite. Dacă acest parametru nu este prezent, nu se face operaţia de comparaţie.
- MASK - defineşte masca utilizată la compararea datelor citite cu cele aşteptate. Dacă un bit dintr-o poziţie oarecare este pe "0", înseamnă că el nu contează în cadrul procesului de scanare, deci nu este luat în considerare.
- SMASK - defineşte care dintre datele TDI nu prezintă importanţă. Dacă un bit oarecare este pe "1", înseamnă că poziţia respectivă din TDI contează, iar dacă este "0", înseamnă că nu contează.

PIO este opţional şi specifică vectorul paralel.

3.2.4.3.7. Comanda STATE

Sintaxă:

STATE cale_1 cale_2 .. cale_n stare_stabilă

Parametri:

- cale_1 .. n sunt parametri opţionali şi descriu explicit starea prin care trece portul de acces pentru a ajunge la starea finală precizată în diagrama de stare. Stările stabile sunt: RESET, IDLE, DRSELECT, DRCAPTURE, DRSHIFT, DRPAUSE, DREXIT1, DREXIT2, DRUPDATE, IRSELECT, IRCAPTURE, IRSHIFT, IRPAUSE, IREXIT1, IREXIT2 şi IRUPDATE.
- Stare_stabilă, indică starea stabilă în care se forţează busul 1149.1. Stările stabile sunt IRPAUSE, DRPAUSE, RESET şi IDLE.

3.2.4.3.8. Comanda TDR, TIR (Traller Data Register, Traller Instruction Register)

Sintaxă:

TDR lungime [TDI] [TDO] [MASK][SMASK];

TRI lungime [TDI] [TDO] [MASK][SMASK];

- Lungime - un număr întreg pe 32 de biţi, care specifică numărul biţilor care se vor scana;
- TDI - specifică datele care se introduc în circuitul ţintă, exprimat în hexa. Dacă acest parametru nu este prezent, se consideră valorile definite anterior.

- TDO - specifică valorile cu care să fie comparate datele citite. Dacă acest parametru nu este prezent nu se face operația de comparare.
- MASK - defineşte masca utilizată la compararea datelor citite cu cele aşteptate. Dacă un bit dintr-o poziție oarecare este pe "0", înseamnă că el nu contează în cadrul procesului de scanare, deci nu este luat în considerare.
- SMASK - defineşte care dintre datele TDI nu prezintă importanță. Dacă un bit oarecare este pe "1", înseamnă că poziția respectivă din TDI contează.

3.2.4.3.9. Comanda TRST (Test RESET)

Sintaxă:

TRST mod

Această comandă descrie modul de utilizare a semnalului operațional Test Reset.

mod - arată când linia TRST este activă, inactivă, în starea de înaltă impedanță sau dacă există. Stările valide sunt:

- mod = ON (active 0 logic)
- mod = OFF (inactiv 1 logic)
- mod = Z (înaltă impedanță)
- mod = ABSENT

3.2.4.3.10. Exemplu de program scris în formatul SVF

În continuare, se prezintă un scurt program scris de autorul tezei în formatul SVF pentru interfața JTAG VX4491, care programează două circuite conectate în lanț pentru testarea pe frontieră.

```
! Begin Test Program
! Disable Test Reset line
TRST OFF;
! Initialize UUT
STATE RESET;
! End IR scans in DRPAUSE
ENDIR DRPAUSE;
! End DR scans in DRPAUSE
ENDDR DRPAUSE;
! 24 bit IR header
HIR 24 TDI (FFFFFF);
! 3 bit DR header
HDR 3 TDI (7);
! 16 bit IR trailer
```

```
TIR 16 TDI (FFFF);
! 2 bit DR trailer
TDR 2 TDI (3);
! 8 bit IR scan, load BIST opcode
SIR 8 TDI (41) TDO (81) MASK (FF);
! 16 bit DR scan, load BIST seed
SDR 16 TDI (ABCD);
! RUNBIST for 95 TCK Clocks
RUNTEST 95 TCK ENDSTATE IRPAUSE;
! 16 bit DR scan, check BIST status
SDR 16 TDI (0000) TDO(1234) MASK(FFFF);
! Enter Test-Logic-Reset
STATE RESET;
! End Test Program
```

Se poate observa că limbajul SVF este descriptiv şi relativ uşor, iar printr-o simplă parcurgere se pot înțelege operațiile care se execută. Acest limbaj se pretează foarte bine la echipamentele de testare automate bazate pe JTAG deoarece specifică vectorii de test de intrare, vectorii de test rezultați, măştile ce se pot aplica, testele ce urmează să se execute şi stările care se vor parcurge.

3.2.5. Structura unui program de test pe stația SMART-2

În continuare, pentru descrierea metodei de test implementată de autor pe stația SMART-2, se va prezenta pe scurt o secvență de program în ATLAS, dezvoltată sub mediul integrat PAWS ([178], [217]). În cadrul acestui subcapitol, nu se intenționează prezentarea limbajului ATLAS ([153], [217]), ci doar să se facă o prezentare a structurii unui program de test şi a metodologiei de testare utilizate.

Din cadrul programului prezentat în anexa D se pot observa următoarele:

- în prima parte a programului se declară resursele utilizate (surse de alimentare, instrumente de măsură, rezistențe, scurt circuite, semnale comune şi cele de masă etc.). Această declarație este importantă întrucât pe baza acesteia compilatorul alocă resursele disponibile, conform bazei de date a acestora.

  Exemplu:

```
000010 REQUIRE, 'AC PSU RES', SOURCE, AC SIGNAL,
 CONTROL,
```

```
    VOLTAGE RANGE 0V TO 250 V CONTINUOUS ERRLMT +- 3 V,
    THRU-RES 10 KOHM ERRLMT +- 5 PC,
CNX HI LO $
```

- sintaxa ATLAS permite ca variabilele să fie globale (adică utilizabile în cadrul întregului program) sau să fie declarate şi în cadrul procedurilor (adică variabilele sunt locale, deci disponibile doar în cadrul procedurii în care au fost declarate).

  Exemplu:

```
000500 DECLARE, VARIABLE, 'VOLT1', 'CURRENT-VAL' IS DECIMAL;
 'EXPECTED', 'STIMUL','OUTPUT-FILE'
  IS FILE OF STRING(80) OF CHAR $
```

- declararea şi specificarea "**DIGITAL CONFIGURATION**" este obligatorie în cazul în care se utilizează semnale numerice mai rar întâlnite;

  Exemplu:

```
000510 DEFINE, 'BSCAN', DIGITAL CONFIGURATION $
   20   DEFINE, 'BSCAN-DRIVER', DIGITAL SOURCE,
         DC SIGNAL USING 'BSCAN-STIM',
         VOLTAGE RANGE 0 V TO 5 V,
         CURRENT RANGE 5 MA TO 10 MA,
         LEVEL-LOGIC-ONE VOLTAGE 2.0 V,
         LEVEL-LOGIC-ZERO VOLTAGE 0.4 V,
         CNX HI 'TDI'
               LO EARTH $
   30   DEFINE, 'BSCAN-RECV', DIGITAL SENSOR, (VALUE),
         VOLTAGE RANGE 0 V TO 5 V,
         CURRENT RANGE 5 MA TO 10 MA,
         LEVEL-LOGIC-ZERO UL 0.8 V LL 0.4 V,
         LEVEL-LOGIC-ONE UL 5 V LL 2.0 V,
         CNX HI J1-DD1
               LO EARTH $
   000540 END, 'BSCAN' $
```

- de obicei, programul principal conţine doar apeluri de proceduri, iar o mare parte din cod este conţinut în proceduri (vezi exemplul precedent);

- terminarea programului se face cu instrucțiunea "**TERMINATE**", care nu numai că specifică asamblorului că s-a terminat programul, dar şi inițializează toate resursele stației (este foarte important ca stația să nu rămână într-o stare nedeterminată).

3.2.6. Teste executate pe unitatea CSC cu ajutorul testării pe frontieră

În exemplul anterior de program scris în ATLAS (anexa D), în programul principal se apelau procedurile:

```
100040 PERFORM, 'CITIRE-FISIERE' ('STIM.STM','EXPECT.STM','MASK.STM',
                    'STIM1', 'MASK1', 'EXPECTED-DATA') $
100050 PERFORM, 'BSCAN-TEST' ('STIM1', 'MASK1', 'EXPECT-DATA') $
```

Procedura "**CITIRE-FISIERE**" citeşte fişierele "**STIM.STM**", "**EXPECTED.STM**" şi "**MASK.STM**", prelucrează aceste fişiere şi returnează ariile de date "**STIM1**", "**MASK1**" şi "**EXPECTED-DATA**". În spatele acestei proceduri se ascund multe operații, care depind de sistemul de operare pe care se lucrează, dar care din punct de vedere al înțelegerii metodei de testare pe frontieră nu sunt importante.

Încărcarea datelor pentru testarea pe frontieră şi citirea răspunsului se face prin apelul procedurii:

```
100050 PERFORM, 'BSCAN-TEST' ('STIM1', 'MASK1', 'EXPECT-DATA') $
```

Procedura "**BSCAN-TEST**" conține liniile:

```
 10 STIMULATE, 'STIM-DATA', ON 'BSCAN-DRIVER' $
 20 PROVE, (VALUE INTO 'DATA')
    REF 'EXPECT-DATA',
    MASK-ONE 'MASK-DATA',
    SAVE-COMP 'WORD',
    ON 'BSCAN-RECV' $
```

Comanda "**STIMULATE**" trimite datele "**STIM-DATA**" pe busul IEEE 1149.1. Aceasta se realizează prin programarea modulului VX4491 prin intermediul busului GPIB al calculatorului. Comanda "**PROVE**" determină citirea datelor de pe placa VX4491 şi deci de pe frontiera circuitului IEEE 1149.1 țintă, prin intermediul driverelor plăcii VX4491. În cadrul

acestui paragraf nu se intenționează prezentarea în detaliu a plăcii VX4491 [147], dar trebuie menționat că are o complexitate relativ mare, că acceptă 78 de comenzi GPIB, iar formatul datelor acceptate de intrare, precum şi a celor furnizate, este relativ complicat.

Pe scurt, pentru realizarea testării pe frontieră este necesară scrierea testului în formatul SVF, după care se fac o serie de prelucrări şi conversii, rezultatul final fiind un fişier, care se va transmite plăcii VX4491. După încărcarea acestui fişier în memoria plăcii VX4491, se poate trece la execuție cu comanda "**RUN**" transmisă GPIB. Este de remarcat că modulul VX4491 generează toate semnalele corespunzătoare pe busul JTAG fără intervenția PC-ului. Citirea datelor în memoria internă a plăcii VX4491 se execută şi la încărcarea fişierului, iar citirea efectivă de către calculator a bufferului se face cu comanda GPIB "**LOAD**".

În continuare, se prezintă pe scurt câteva teste, care se execută asupra unității de control a încărcăturii cu ajutorul testării pe frontieră. Se poate spune că în principal se execută trei tipuri de teste:

- verificarea conexiunilor între cele două circuite FPGA Xilinx 4010;
- verificarea pinilor de ieşire pe partea de curent continuu;
- verificarea pinilor de ieşire pe partea de curent alternativ.

Structura de conectare a celor două circuite FPGA din cadrul unității este prezentată în figura 3.7. Între cele două FPGA-uri sunt o serie de conexiuni (exceptând magistrala IEEE 1149.1), ca atare o problemă importantă este verificarea acestora.

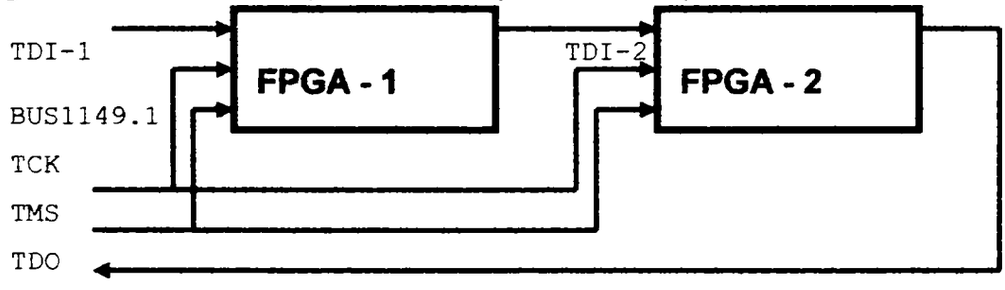


Figura 3.7. Interconectarea circuitelor FPGA la CSC

3.2.6.1. Verificarea conexiunii între cele două circuite FPGA XILINX 4010

Metoda de testare a interconexiunilor între cele două FPGA presupune programarea pinilor unui circuit în regim de ieşire, încărcarea unui vector de test şi citirea nivelelor logice pe celălalt circuit. În vectorul citit se maschează biții, care nu interesează, cu ajutorul fişierului de mască şi se compară apoi cu rezultatul aşteptat. Dacă există

diferențe înseamnă că există probleme de interconectare şi chiar mai mult, se poate afla pe ce pin au apărut erori.

Structura fişierului SVF utilizată la această categorie de teste este prezentată în continuare:

```
!! i1_01.INV Fisierul care se incarcă este i1_01
TRST ABSENT; ! Nu se utilizează semnalul de RESET
ENDIR IDLE;  ! Starea finală IDLE după scanarea IR
ENDDR IDLE;  ! Starea finală IDLE după scanarea DR
SIR 6 TDI (00); ! Registrul de instructiuni este de 6 biti iar codul transmis
                ! este 00 adică codul pentru instructiunea EXEC
SDR 976
TDI(80071C01C01F8E001FFFC7FFFFC7FC7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFF1FFE3FFF8000E001C01C0007000000038FC70070BFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFF); ! Se incarcă 976 biți de date
SDR 976
TDI(80071C01C01F8E001FFFC7FFFFC7FC7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFF1FFE3FFF8000E001C01C0007000000038FC70070BFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFF); ! Se face o incarcare pentru a se citi datele anterioare
```

Se poate observa că în fişierul de test "i1_01.SVF", la început se precizează că nu există semnal de RESET extern (se va executa automat RESET soft) şi că starea finală după instrucțiunile "IR" si "DR" este IDLE.

Instrucțiunea "SIR 6 TDI (00)" încarcă registrul de instrucțiuni al celor două circuite FPGA (întrucât sunt conectate în serie) cu codul "000", adică instrucțiunea EXEC.

Instrucțiunea: `SDR 976 TDI(80071C01C01F8E001FFFC7FFFFC7FC7FFFFFFFFFFFF`

```
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FFE3FFF8000E001C01C0007000000038FC70070BF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF);
```

încarcă în registrele de date ale ambelor circuite FPGA datele precizate direct. Este important de remarcat că fiecare circuit FPGA conține un registru de date de 488 biți, dar datorită conectării seriale, cu o singură instrucțiune se pot încărca ambele circuite.

Instrucțiunea din registrul SDR(EXEC) se execută de două ori, deoarece la prima execuție se încarcă datele în FPGA, iar a doua permite citirea datelor corespunzătoare valorilor programate anterior. Ca urmare a acestei instrucțiuni, se face şi citirea datelor de către placa VX4491, după care sunt citite de calculator pe magistrala GPIB şi comparate cu vectorii de referință.

3.2.6.2. Verificarea ieşirilor DC unității de control a încărcăturii

Testele ieşirilor de curent continuu presupun programarea pinilor de ieşire pe "0" logic sau lăsarea lor la 28 V şi verificarea răspunsului cu un multimetru.
Structura fişierului SVF pentru aceste teste este prezentată în continuare:

```
! dc01.INV
TRST ABSENT;
ENDIR IDLE;
ENDDR IDLE;
SIR 6 TDI (00);
SDR 976
TDI(FFFFFFFFFFFC75FFE800BFE170381F8E00E3F03FFFFE07E381FFFFF1FFFFFFF007FFFFFFC7F
F8FFFFFFFFFFFFFFFFFFFFFE3FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FFFFFFFFFFFFF
FFFFFFFF);
```

Din codul prezentat anterior se observă că vectorii de test se încarcă doar o singură dată, întrucât nu interesează datele citite. Verificarea tensiunii la ieşire se face cu un multimetru numeric comandat prin magistrala GPIB de către calculator.

3.2.6.3. Verificarea ieşirilor AC ale unității de control a încărcăturii

Testele de curent alternativ presupun programarea unei tensiuni alternative pe anumiți pini de ieşire în domeniul 113V..118V, sau lăsarea lor pe "0". Verificarea nivelului de tensiune se face cu un multimetru numeric comandat prin GPIB.

Structura fişierului SVF pentru această categorie de teste este prezentată în continuare:

```
! i101.INV
TRST ABSENT;
ENDIR IDLE;
ENDDR IDLE;
SIR 6 TDI (00);
```

```
SDR 976
TDI(FFFFFFFFFFFC75FFE800BFE170381F8E00E3F03FFFFE07E381FFFFF1FFFFFFF007FFFFFFC7F
F8FFFFFFFFFFFFFFFFFFFFFE3FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FFFFFFFFFFFF
FFFFFFFF);
```

Se poate observa că şi la această categorie de teste, se încarcă vectorul de test şi se verifică cu multimetrul numeric, comandat prin GPIB, dacă tensiunea pe pinul de ieşire corespunzător este corectă.

Ca urmare a implementării practice a metodelor de testare moderne elaborate de autorul tezei au rezultat următoarele:

- în cazul unei plăci sau al unui sistem electronic proiectat din considerente de testare (de exemplu, unitatea CSC), prin testarea JTAG este posibilă verificarea în proporție de mai bine de 90% a interconexiunilor interne ale unității (verificarea cablajului şi a lipiturilor), verificarea tuturor intrărilor şi ieşirilor numerice, iar cu ajutorul instrumentelor programabile de pe stația de test SMART-2 se poate face verificarea părții analogice;
- utilizarea unei stații de test performante (de exemplu SMART) şi a unui mediu de dezvoltare adecvat (de exemplu PAWS) permite reducerea semnificativă a timpului de dezvoltare a programului de test şi asigură flexibilitatea şi resursele hardware şi software necesare;
- deşi unitatea de control a încărcăturii Boeing 777 este relativ complicată, testarea acesteia se face relativ simplu, iar rezultatele testelor pot indica precis localizarea defecțiunii;
- cu adaptări minore la stația SMART există posibilitatea testării în paralel a mai multor unități CSC;
- prin optimizarea resurselor hardware şi software ale stației de test este posibilă utilizarea testării JTAG şi la verificarea producției de circuite integrate, echipamente şi sisteme electronice. De exemplu este posibilă reducerea semnificativă a timpului de testare prin încărcarea prealabilă a vectorilor de test şi a celor de răspuns aşteptat în memoria FLASH a interfeței JTAG, prin creşterea frecvenței de tact a magistralei JTAG şi prin proiectarea mai atentă din considerente de testare a circuitului integrat, a plăcii electronice sau a sistemului testat;

programele suplimentare de conversie a formatului vectorilor de test implementate de autor au permis automatizat generarea vectorilor de test şi au redus semnificativ timpul de dezvoltare a mediului de testare.

3.3. Testarea echipamentelor electronice auto

În cadrul acestui paragraf se analizează pe scurt stadiul actual al echipamentelor electronice ale automobilelor şi necesitățile la testarea acestora, scopul principal fiind determinarea metodei celei mai potrivite de testare [161], [162], [165], [167], [172], [241], [247], [249], [250], [252].

3.3.1. Arhitectura electrică a automobilului modern

În fig. 3.8 se prezintă arhitectură electrică a unui automobil putându-se observa că diversele echipamentele electronice sunt interconectate prin intermediul unor magistrale CAN (Controller Area Network) şi LIN (Local Interconnect Network), existând astfel un schimb intens de date dintre ECU-uri (Electronic Control Unit) din componența rețelei.

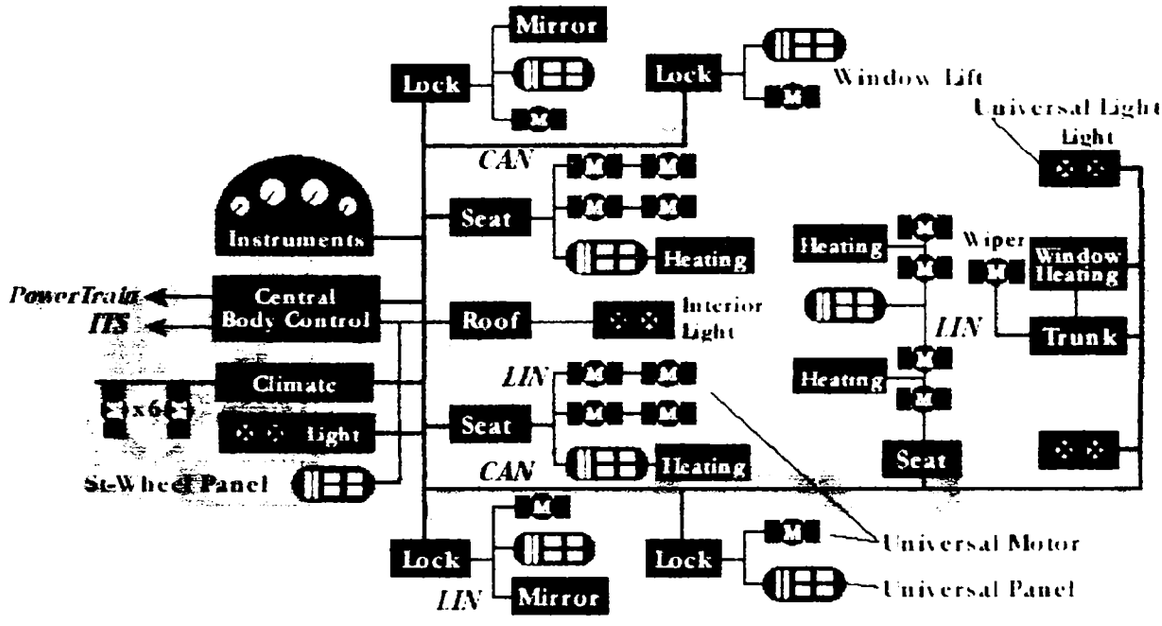


Fig.3.8. Conectarea echipamentelor electronice din automobilele actuale

De asemenea este important de remarcat că ECU-urile care schimbă relativ multe date între ele sunt conectate prin intermediul bus-ului CAN iar ECU-urile care schimbă mai puține date (de obicei cele de execuție) sunt conectate prin intermediul unei magistrale mai lente ca de exemplu LIN. Frecvent, în automobile moderne se întâlnesc două magistrale CAN, una de mare viteză şi una de viteză medie, existând în componența rețelei şi unele ECU-uri care prezintă şi rolul de punte între rețele. În ultimul timp se

constată tendința de utilizare şi a unor magistrale mai rapide pentru conectarea diverselor echipamente multimedia şi de navigație din automobil.

Se impune a se menționa că există tendința clară de creştere a numărului de ECU-uri, devenind obişnuit ca un automobil modern din categoria BMW 7 să prezinte peste 80 de ECU-uri, câteva magistrale-uri şi câteva punții între rețele

Caracteristici generale ale ECU-urilor din cadrul rețelei:

- conțin cel puțin un microcontroller;
- se alimentează de regulă la 12V sau 42V;
- prezintă uneori un număr de intrări analogice care se conectează la diverse comutatoare sau senzori;
- prezintă ieşiri analogice pentru comanda motoarelor electrice, valve, LED-uri sau becuri;
- de regulă citesc informațiile relevante pe magistralele pe care sunt conectate şi le folosesc în execuție;
- transmit informațiile relevante pentru alte ECU-uri pe diverse magistrale (CAN, LIN);
- prezintă posibilități de auto-diagnoză prin intermediul magistralei la care sunt conectate;
- tot mai frecvent există posibilitatea de a actualiza programele din componența ECU-urilor prin intermediul magistralei CAN;
- conțin memorii EEPROM care păstrează o serie de informații de configurare, coduri de eroare şi de identificare.

3.3.2. Cerințe principale la testarea echipamentelor electronice auto

Întrucât necesitățile de testare diferă în funcție de scopul urmărit analiza se va face pe următoarele faze:

- la dezvoltarea ECU-ului;
- la testarea de sistem din punct de vedere al conformității conform specificațiilor;
- la producția ECU-ului;
- la atelierul de service auto.

3.3.2.1. Testarea la dezvoltarea ECU-ului

Testarea la dezvoltarea urmăreşte verificarea ECU-ului proiectat din punct de vedere HW şi software presupunând asigurarea în mediul de dezvoltare al ECU-ului a

echipamentelor necesare pentru simularea celorlalte ECU-uri din cadrul rețelei şi pentru simularea intrărilor/ieşiri aşa cum se prezintă în fig. 3.9.

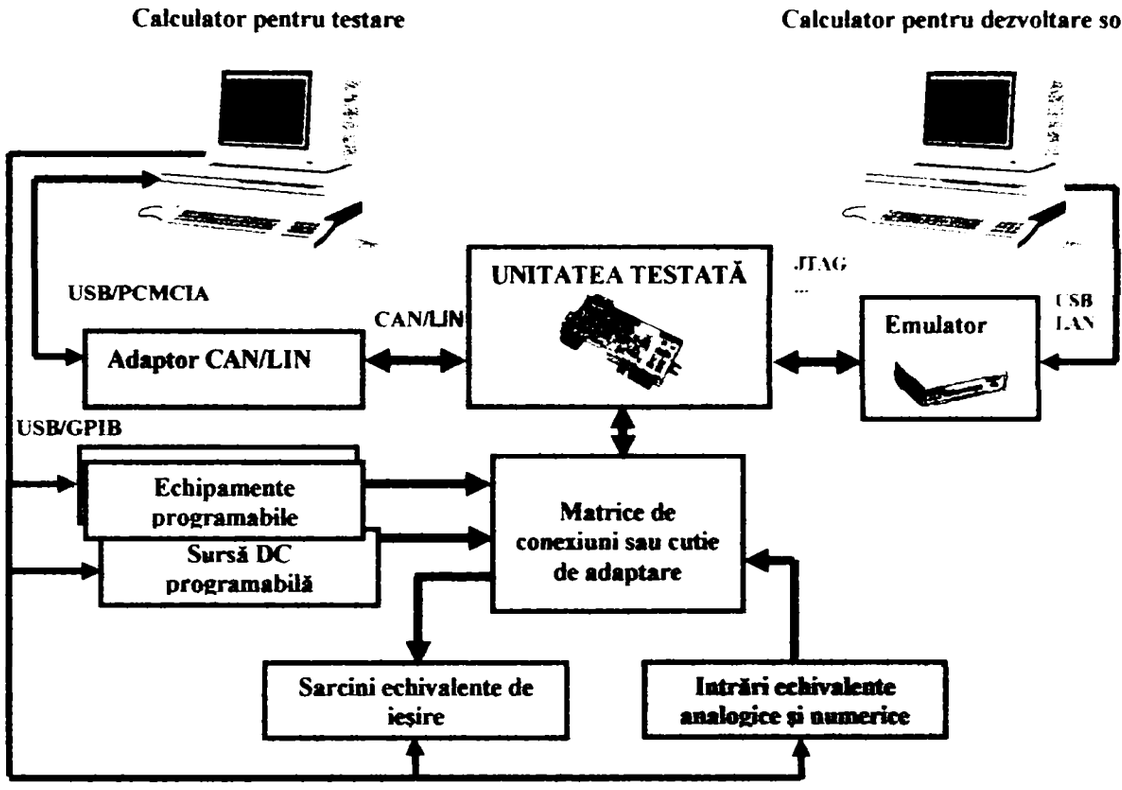


Fig. 3.9. Mediu de testare la dezvoltarea programelor

Din fig. 3.9 se poate observa că la calculatorul utilizat la dezvoltarea soft-ului este conectat un emulator, iar calculatorul de testare prin intermediul adaptorului CAN/LIN şi a celorlalte echipamente programabile simulează mediul de funcționare al unității testate. În aceste condiții testarea la dezvoltare presupune următorii paşi:

- încărcarea şi rularea în unitatea testată a soft-ului adecvat;
- aplicarea prin intermediul calculatorului de test, a programelor de test şi a echipamentelor programabile a secvenței de test;
- compararea funcționării unității testate cu specificațiile.

Este necesar a se menționa că tipul programelor de testare şi tipul echipamentelor utilizate (fig. 3.9) depind în principal de tipul unității existând uneori mari deosebiri.

3.3.2.2. Testarea de conformitate

Testele de conformitate se execută la nivel de sistem, ele urmăresc validarea funcționării ECU-ului conform specificațiilor şi implică conectarea acestuia într-un mediu de test cât mai apropiat de cel real precum şi verificarea răspunsului la diverşi stimuli de test. În aceste condiții mediul de testare este asemănător cu cel din fig. 3.9, cu deosebire că nu se mai utilizează emulatorul întrucât soft-ul necesar este încărcat prealabil în ECU.

Cât priveşte testele executate, în acest caz se insistă asupra comportării la nivel de sistem fiind necesar ca mediul ce testare să fie cât mai apropiat de mediul real de funcţionare. Întrucât majoritatea ECU-urile utilizate în automobile pot intra în modul de lucru de diagnoză sunt posibile următoarele operaţii principale prin intermediului unei magistrale de comunicaţie:

- citirea şi modificarea memoriei RAM;
- citirea şi modificarea memoriei EEPROM;
- citirea şi modificarea intrărilor şi ieşirilor;
- reprogramarea memoriei program (dacă ECU-ul conţine memorie FLASH).

Deseori modul de diagnoză se utilizează la testele de sistem sau de conformitate şi uşurează mult testarea.

3.3.2.3. Testarea la producţie

Testarea la producţie a ECU-urilor presupune de obicei două categorii de teste:

- testarea HW a ECU-ului pe linia de producţie;
- testarea ECU-ului produs la nivel de sistem la sfârşitul producţiei.

Testarea HW a ECU-ului se realizează uzual prin intermediul unui program de test relativ mic (1-2Kbyte) care se încarcă temporar în memoria RAM sau FLASH a ECU-ului testat. Scopul principal al acestor teste este de a verifica cablajul, conexiunile electrice, memoria RAM, EEPROM, intrările şi ieşirile. De regulă gradul de acoperire al acestor teste este dependent de cât de bine este proiectată unitatea testată din punct de vedere HW, fiind posibilă uneori şi utilizarea testării JTAG.

Utilizarea limitată a testării JTAG în domeniul auto este explicabilă mai ales din motive istorice şi de cost, adică producătorii de echipamente electronice auto preferă să nu utilizeze circuite cu facilităţi JTAG pentru a reduce la maxim costul plăcilor electronice produse. Totuşi, autorul tezei estimează că creşterea complexităţii plăcilor electronice şi cerinţele tot mai riguroase de calitate vor face ca testarea JTAG să fie utilizată mai frecvent la producţie mai cu seamă că uneori se utilizează deja microcontrolere programabile prin JTAG.

Testarea ECU-ului produs la sfârşitul liniei de producţie este asemănătoare cu testarea la nivel de sistem, cu deosebirea că de obicei se face în paralel pentru mai multe ECU-uri în diverse condiţii critice, iar numărul de teste este mai redus pentru a reduce timpul de testare pe bandă de producţie.

3.3.2.4. Testarea la service

Testarea la service-ul auto a ECU-urilor prezintă de obicei următoarele cerințe:

- trebuie să fie cât mai uşor fără a fi necesară demontarea unității testate;
- să permită accesarea unor informații din memoria EEPROM ca de exemplu codurile cu erorile care au apărut în timpul funcționării;
- să permită citirea/reprogramarea memoriei EEPROM;
- este recomandabil să permită re-programarea memoriei FLASH;
- este necesar ca ECU-ul testat să poată furniza şi alte informații referitor la senzorii şi unele elemente mecanice din sistem pentru a permite diagnosticarea subsistemului în care este montat ECU-ul.

Ca urmare a acestor cerințe, testarea la service-ul auto se realizează cu ajutorul unui echipament special de diagnoză, de regulă pe magistrala CAN pe baza facilităților de diagnoză oferite de ECU-urile din maşină.

3.3.3. Concluzii la testarea echipamentelor electronice auto

Din cadrul acestui paragraf rezultată că testarea ECU-urilor utilizate în automobile se realizează în principal cu ajutorul facilităților de diagnoză pe care acestea le au implementate prin intermediul unor magistrale standard de comunicație (CAN). Este necesar să se menționeze că există şi situații în care microcontroller-ele utilizate prezintă facilități JTAG ceea ce reduce costurile testări HW pe linia de producție şi costurile de dezvoltare întrucât emulatoarele bazate pe JTAG sunt de 4-5 ori mai ieftine decât emulatoarele active.

3.4. Analiza comparativă a testării JTAG şi a testării cu echipamente configurabile

Scopul acestei secțiuni este de a analiza comparativ testarea unui sistem electronic complex cu ajutorul JTAG şi cu ajutorul echipamentelor de testare configurabile.

Pentru o analiză obiectivă a celor două metode de testare, aceasta se va efectua în următoarele condiții:

- se va realiza testarea unității de control a încărcăturii (CSC), prezentate în secțiunea anterioară, prin ambele metode de testare;
- din punct de vedere hardware, în ambele cazuri se utilizează stația de test SMART-2, prezentată în secțiunea precedentă. În cazul testării JTAG,

comunicația cu unitatea testată se face utilizând magistrala JTAG şi modulul VX4491, iar în cazul testării cu echipamente de testare programabile, comunicația cu unitatea testată se face printr-o interfață serială RS232.

- din punct de vedere software se utilizează acelaşi mediu de lucru (PAWS şi UNIXWARE) în cazul ambelor metode de testare;
- întrucât cele două abordări ale testării necesită conexiuni electrice diferite se vor utiliza unități diferite de adaptare;
- se va realiza testarea funcțională în aceleaşi condiții în ambele cazuri.

Rezultatele testării unității CSC prin JTAG (cazul A) şi cu ajutorul echipamentelor de testare configurabile (cazul B) sunt prezentate în tabelul 3.1.

Din tabel se poate observa că rezultatul numeric final este favorabil metodei JTAG, dar trebuie avute în vedere şi următoarele concluzii:

- testarea JTAG este net superioară dacă unitatea testată este proiectată din considerente de testare;
- la testarea JTAG timpul de test este mai redus, iar testabilitatea este mai ridicată;
- datorită hardware-ului JTAG suplimentar introdus, echipamentele cu facilități de testare JTAG sunt mai scumpe.

Pe baza rezultatelor practice ale metodelor elaborate şi implementate de autorul tezei a rezultat că în cazul echipamentelor şi a sistemelor electronice complexe proiectarea din considerente de testabilitate şi testarea utilizând JTAG este obligatorie în vederea asigurării unei foarte bune calități a produselor, iar în cazul produselor de serie la care prețul de producție este foarte important se recomandă celelalte metode de testare.
Acest rezultat este în concordanță cu analiza asupra echipamentelor electronice auto efectuată în paragraful 3.3. şi care a dovedit că testarea JTAG este potrivită pentru echipamente şi sisteme electronice complexe proiectate din considerente de testare, dar nu este prea rentabilă la echipamentele de complexitate medie şi redusă.

Nr. crt.	Criteriu de analiză	Metoda A (JTAG)	Metoda B (Conf)	Observaţii
1	Testarea cablajului unităţii testate	9	1	1. Metoda A permite testarea conexiunilor electrice din interiorul unităţii testate, dar gradul de testare depinde de proiectare. Deoarece unitatea CSC a fost proiectată din considerente de testare, metoda A a obţinut aproape punctajul maxim. 2. Întrucât metoda B nu a permis testarea cablajului s-a acordat punctaj minim.
2.	Testarea unităţii centrale (testarea FPGA-urilor şi a microcontrollerului)	8	5	1. Metoda A permite o testare completă a FPGA-urilor (se rulează selftestul) şi o testare funcţională a microcontrollerului prin intermediul FPGA-urilor şi a magistralei JTAG. 2. Metoda B permite o testare acceptabilă a microcontrollerului prin executarea unei subrutine de test şi comunicarea răspunsului pe interfaţa serială. De asemenea, este posibilă o testare parţială a circuitelor FPGA, dar de regulă insuficientă.
3.	Auto-identificarea echipamentului	10	10	În cazul metodei A se execută instrucţiunea IDCODE, iar în cazul metodei B se solicită această informaţie serial.
4.	Teste pentru verificarea tensiunilor pe pinii de curent continuu	10	9	1. Metoda A permite schimbarea tensiunilor de pe ieşirile de curent continuu prin intermediul magistralei JTAG. 2. Metoda B permite schimbarea tensiunilor de pe ieşirile de curent continuu prin intermediul comenzilor trimise pe interfaţa serială, dar unitatea testată trebuie să permită această facilitate.
5.	Teste pentru verificarea tensiunilor pe pinii de curent alternativ	10	9	1. La metoda A, schimbarea tensiunilor de pe ieşirile de curent alternativ se realizează convenabil prin intermediul magistralei JTAG. 2. La metoda B, schimbarea tensiunilor de pe ieşirile de curent alternativ se realizează prin intermediul comenzilor trimise pe interfaţa serială, dar unitatea testată trebuie să permită această facilitate.
6.	Complexitatea unităţii de adaptare (notă maximă pentru complexitate minimă)	8	7	În ambele cazuri este necesară realizarea unor adaptări hardware, dar metoda B necesită adaptări suplimentare. La metoda B este necesară controlarea de la calculator a mai multor linii de intrare şi ieşire, ceea ce la metoda A se face prin instrucţiuni JTAG.
7.	Timpul de testare (notă maximă pentru timp minim)	9	8	Metoda B este de regulă mai lentă datorită vitezei de comunicaţie mai redusă pe interfaţa serială.
8.	Gradul de testabilitate a unităţii CSC	9	7	Dacă unitatea testată este bine proiectată din considerente de testare JTAG, se obţine o testabilitate apropiată de 95%.
9.	Costul staţiei de test (cost minim pentru punctaj maxim)	5	9	1. Metoda A necesită un modul de interfaţă VX4491 şi software corespunzător, ceea ce va creşte costul staţiei. 2. La metoda B este necesară asigurarea unei interfeţe seriale suplimentare, dar aceasta nu constituie de regulă o problemă.
10.	Costurile suplimentare de producţie pentru asigurarea testabilităţii echipamentului	4	9	1. Metoda A necesită proiectarea echipamentului respectiv din considerente de testare, ceea ce creşte costul de dezvoltare, iar circuitele JTAG introduse cresc costul de producţie al echipamentului. 2. La metoda B este necesară implementarea de funcţionalităţi de test în softul echipamentului respectiv, dar de regulă costul asociat este acceptabil .
	TOTAL	**8.2**	**7.4**	

Observaţie: Notarea performanţelor s-a făcut de la 0 la 10 (10 pentru maxim).

Tabelul 3.1. Compararea metodelor de test

3.5. Concluzii

În secțiunea 3.1. (***Metode de testare JTAG***) s-au prezentat posibilitățile de testare cu JTAG şi câteva exemple practice sugestive întâlnite în literatura de specialitate. Se impune a menționa că domeniul de utilizare al echipamentelor care folosesc magistrale JTAG este în continuă extindere, existând nu numai echipamente de testare bazate pe JTAG, ci şi relativ multe programatoare şi emulatoare de microcontrollere şi microprocesoare.

În secțiunea 3.2. (***Contribuții la testarea cu JTAG. Testarea completă a unei unități de aviație de control a încărcăturii cu JTAG***) s-a prezentat stația de test utilizată (SMART-2), s-a făcut o sinteză a limbajelor utilizate pentru testarea IEEE 1149.1, s-a descris o parte din testele implementate pe unitatea de control a încărcăturii şi s-au prezentat secvențele de cod utilizate.

În cadrul acestui paragraf autorul tezei are următoarele contribuții importante:

- conceperea, dezvoltarea şi implementarea metodei de simulare a magistralei JTAG pe portul paralel al calculatoarelor compatibile PC;
- conceperea, dezvoltarea şi implementarea driverului pentru placa VX4491 în mediul de dezvoltare PAWS;
- analiza comparativă pe baza rezultatelor practice între simularea magistralei JTAG pe portul paralel şi utilizarea unor echipamente dedicate de interfață JTAG;
- analiza practică a performanțelor echipamentelor de interfață JTAG şi recomandarea unor soluții practice de îmbunătățire a stației de testare;
- conceperea, implementarea şi testarea practică a unor programe de testare JTAG în limbajele BSDL, HSDL şi SVF, precum şi prezentarea unor recomandări rezultate din practică;
- conceperea, dezvoltarea şi implementarea metodei de testare a unității de control a încărcăturii Boeing 777 prin testarea JTAG, precum şi elaborarea unor recomandări pe baza rezultatelor practice;
- analiza comparativă pe stația SMART-2 a testării JTAG şi a testării cu echipamente configurabile.

Concluziile principale rezultate din secțiunea 3.2 sunt:

- este posibilă simularea magistralei JTAG pe portul paralel al unui calculator PC independent de complexitatea conectării circuitelor JTAG din lanț şi de numărul acestora, dar frecvența maximă a semnalului de tact este dependendentă în principal de performanțele calculatorului şi de sistemul de operare utilizat şi în general este limitată la circa 10 KHz. Astfel, în cazul testării unității de control a încărcăturii transferarea unui vector de test de 976 biți dura aproape 0.5 s, adică acceptabil în acest proiect, dar inacceptabil la testarea pentru producție;
- întrucât în cazul stației SMART-2, interfața IEEE 1149.1 VX4491 este conectată la calculatorul de test indirect prin intermediul sertarului VXI şi prin magistrala GPIB, transferarea vectorilor de test este relativ lentă în comparație cu transferul prin interfața paralelă. În aceste condiții, la proiectarea unei stații de test este recomandabilă analiza prealabilă atentă a necesităților de transfer de date între diferitele echipamente şi alegerea soluției adecvate (conectarea directă prin GPIB, conectarea în rețea sau prin USB) [241], [247]);
- echipamentele dedicate de interfațare cu magistrala JTAG (de exemplu VX4491) permit utilizarea unui semnal de tact cu frecvența maximă de zeci de MHz, sunt relativ complexe, conțin memorii tampon de date de dimensiuni relativ importante pentru vectorii de test şi acceptă o serie de comenzi complexe de la calculatorul cu care sunt conectate. Astfel în cazul utilizării interfețelor JTAG dedicate procedura de testare se poate optimiza, adică la începutul testelor se pot încărca vectorii de test, vectorii de răspuns corecți şi programul de test în memoria tampon a echipamentului, după care intern în placa de interfață independent de calculatorul master se rulează testele necesare, iar la final aceasta furnizează calculatorului de test rezultatul testului;
- frecvent echipamentele dedicate de interfațare cu magistrala JTAG dispun de mai multe canale fiind astfel posibilă rularea testelor în paralel pe mai multe magistrale JTAG;
- există serii de circuite JTAG de suport (buffere, monitoare de bus, distribuitoare de bus JTAG etc.) produse de firme de prestigiu, iar în ultimul timp tot mai mulți producători de circuite integrate numerice fac publice (uneori şi pe Internet) modelele în BSDL şi VHDL ale circuitelor produse, ceea ce reduce semnificativ

timpul de dezvoltare a mediului de testare JTAG şi contribuie la creşterea calităţii testării, precum şi la reducerea costurilor asociate;

- tot mai multe circuite integrate (FPGA, CPLD, PLD, microprocesoare şi microcontrollere) prezintă facilităţi de testare JTAG;
- există deja o standardizare în domeniul limbajelor utilizate la testarea JTAG (BSDL, HSDL şi SVF);
- în cazul unei plăci sau al unui sistem electronic proiectat din considerente de testare (de exemplu unitatea CSC), prin testarea JTAG este posibilă verificarea în proporţie de mai bine de 90% a interconexiunilor interne ale unităţii (verificarea cablajului şi a lipiturilor), verificarea tuturor intrărilor şi ieşirilor numerice, iar prin intermediul resurselor de pe staţia de test SMART-2 verificarea părţii analogice. Generalizând rezultă că testarea JTAG se pretează foarte bine la testarea circuitelor integrate, a plăcilor şi sistemelor electronice care au fost proiectate din considerente de testabilitate, situaţie în care se poate ajunge la o testabilitate apropiată de 100%;
- utilizarea unei staţii de test performante şi a unui mediu de dezvoltare adecvat permite reducerea semnificativă a timpului de dezvoltare al programelor de test şi asigură flexibilitatea şi resursele hardware şi software necesare;
- pentru testarea completă a unităţii de control a încărcăturii s-au utilizat 266 de fişiere de stimuli şi 148 de fişiere de răspuns (vezi anexa E) şi sunt necesare circa 3 ore (durata este relativ mare datorită numărului foarte mare de teste şi datorită softului complex utilizat: UnixWare + PAWS).
- deşi unitatea de control a încărcăturii Boeing 777 este relativ complicată, testarea acesteia se face relativ simplu, iar rezultatele testelor pot indica precis localizarea defecţiunii;
- prin optimizarea resurselor hardware şi software ale staţiei de test este posibilă utilizarea testării JTAG şi la producţia de circuite integrate, echipamente şi sisteme electronice. Astfel este posibilă reducerea semnificativă a timpului de testare prin încărcarea prealabilă a vectorilor de test şi a celor de răspuns aşteptat în memoria FLASH a interfeţei JTAG, prin creşterea frecvenţei de tact a magistralei JTAG şi prin proiectarea mai atentă din considerente de testare a circuitului integrat, a plăcii sau a sistemului electronic testat;

- programele suplimentare de conversie a formatului vectorilor de test implementate de autor au permis automatizat generării vectorilor de test şi au redus semnificativ timpul de dezvoltare a mediului de testare.
- testarea JTAG se poate aplica foarte bine şi în cazul în care echipamentul testat este compus din mai multe plăci electronice;
- testarea JTAG se pretează şi la testarea echipamentelor electronice complexe, care necesită o foarte mare siguranţă în exploatare (de exemplu, unitatea de control a încărcăturii);
- metoda de testare JTAG este deja folosită şi există o serie de firme de prestigiu, care oferă echipamente de testare JTAG;

În secţiunea 3.3. (***Testarea echipamentelor electronice auto***) se analizează pe scurt necesităţile de testare a echipamentelor electronice auto şi posibilele soluţii. Din analiza efectuată şi din experienţa practică, autorul susţine că testarea cu echipamente configurabile completată de facilităţile de diagnoză din unităţile testate reprezintă o metodă potrivită în majoritatea cazurilor dar nici testarea JTAG nu trebuie neglijată. De asemenea a rezultat că programele de testare devin tot mai complexe şi mai importante fiind recomandabilă utilizarea unor soluţii consacrate în domeniu [241], [247], [250], [252].

În secţiunea 3.4. (***Analiza comparativă a testării JTAG şi a testării cu echipamente configurabile***), autorul a comparat testarea unităţii de control a încărcăturii prin echipamente configurabile şi prin JTAG. În urma analizei efectuate, autorul a ajuns la concluzia că testarea JTAG este foarte potrivită pentru testarea echipamentelor şi sistemelor numerice complexe a căror testabilitate şi siguranţă în funcţionare este foarte importantă, dar poate fi o metodă prea scumpă pentru echipamentele electronice de serie la care asigurarea unui preţ redus este o prioritate. De asemenea, a rezultat că testarea cu echipamente configurabile poate fi o soluţie potrivită pentru testarea în laborator sau la dezvoltarea unui produs, dar gradul de testabilitate depinde în foarte mare măsură de fiecare proiect în parte.

Pe scurt, în acest capitol autorul a prezentat o utilizare concretă a testării JTAG, implementată pentru British Aeroplane, metoda de testare dovedindu-şi avantajele faţă de implementările clasice, fiind utilizată la testarea unităţii de aviaţie Boeing 777 de control a încărcăturii. De asemenea, este necesar a se sublinia că atât metoda de test prezentată în acest capitol, cât şi implementarea ei sunt originale şi au fost prezentate şi în paginile IFAC 2000 (IFAC Workshop Ostrava, Czeck Republic, 8-9 February 2000) [75].

CAPITOLUL 4

4. CONTRIBUȚII LA CONECTAREA CIRCUITELOR JTAG

În cadrul capitolului 3 ("CONTRIBUȚII LA TESTAREA ECHIPAMENTELOR ELECTRONICE COMPLEXE UTILIZÂND JTAG"), autorul a analizat posibilitatea aplicării testării JTAG, realizând şi implementări practice ale acesteia. Din analiza şi din implementările efectuate de autor, prezentate în capitolul precedent, a rezultat că testarea JTAG este o soluție foarte potrivită pentru testarea unor echipamente şi sisteme electronice complexe la care siguranța în funcționare şi testabilitatea sunt foarte importante. Autorul a mai constatat că odată cu creşterea densității de integrare şi a miniaturizării componentelor şi echipamentelor electronice este necesară şi o extindere a testării cu ajutorul JTAG.

În cadrul acestui capitol autorul îşi propune:

- analiza testării JTAG la nivel de sistem electronic;
- aprofundarea diferitele metode de conectare pe magistrala JTAG;
- analiza performanțele diverselor configurații de conectare JTAG;
- analiza testării ierarhice.

Autorul tezei va efectua această analiză pe baza unor exemple concrete şi va face o comparație a diverselor configurații de conectare, prezentând şi recomandări pentru fiecare metodă de conectare analizată.

Se impune a se menționa că în literatura de specialitate ([65], [83], [94], [126], [131], [207] [212], [213], [214], [242], [245], [251]) există doar informații sumare asupra domeniului prezentat în acest capitol şi că toate analizele efectuate în acest capitol reprezintă contribuții ale autorului tezei.

4.1. Posibilități standard de conectare a circuitelor JTAG

În cadrul acestei secțiuni se analizează câteva metode de conectare JTAG şi se compară avantajele şi dezavantajele diverselor configurări. Scopul principal al secțiunii

este ca pe baza rezultatelor analizei să se recomande metodele de conectare JTAG adecvate pentru diverse echipamente sau sisteme electronice.

4.1.1. Conectarea în cascadă a circuitelor JTAG

La proiectarea unei plăci electronice, care conține mai multe circuite JTAG, este foarte important să se aleagă modul cel mai potrivit de conectare a acestora. Una dintre metodele de conectare posibile este conectarea serie (vezi figura 4.1.), situație în care circuitele JTAG sunt conectate din punct de vedere al căii de date în serie.

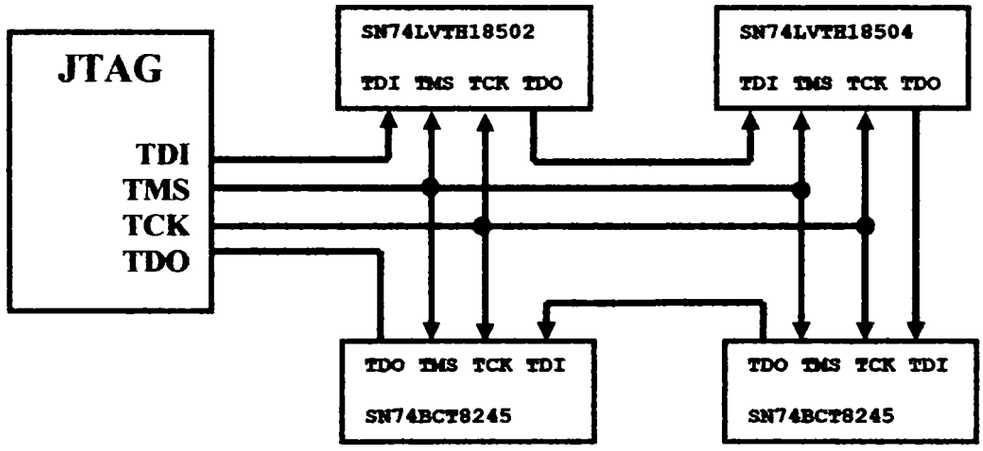


Figura 4.1. Topologia serie

Din figura 4.1. se poate observa că semnalul TMS şi TCK se aplică simultan la toate circuitele, iar din punct de vedere al căii de date TDI-TDO circuitele sunt conectate în serie.

În tabelul 4.1. se prezintă o scurtă descriere a circuitelor utilizate în exemplul 4.1.

TIP	NR.DE BIȚI	IEŞIRI	NR. DE CELULE	DESCRIERE
SN74LVTH18502	18	3s	47	Universal 18-bit transceivers
SN74LVTH18504	20	3s	47	Universal 20-bit transceivers
SN74BCT8245	8	3s	17	Universal 8-bit transceivers

Tabelul 4.1. Scurtă descriere a circuitelor din exemplul 4.1.

Întrucât timpul de testare este un parametru foarte important, acesta se va analiza în continuare pentru exemplul din figura 4.1., după care se va face extrapolarea la cazul general.

În tabelul 4.2. se indică numărul de impulsuri ale semnalului de tact TCK pentru a aduce circuitul JTAG într-o anumită stare. Cu "n_{IP}" se notează lungimea registrului de IP, iar prin "n_{DR}" se notează lungimea registrului de date.

Stare TAP	**TCK**	**Observații**
Test-Logic-Reset	1	TMS trece pe 0 înainte de frontul descrescător al TCK.
Run-Test/Idle	2	
Select-DR-Scan	3	
Select-IT-Scan	4	
Capture-IR	5	
Shift-IR	6	TDO devine activ, iar TDI trebuie să fie valid pe frontul descrescător al semnalului de tact. Primul bit este introdus în TAP pe frontul crescător al semnalului de tact.
Shift-IR	n_{IP}+5	Biții sunt introduşi în registrul IR pe fiecare front crescător al semnalului de tact.
Exit1-IR	n_{IP}+6	TDO devine inactiv pe frontul descrescător al semnalului de tact.
Update-IR	n_{IP}+7	IR este încărcat cu noua instrucțiune pe frontul descrescător al TCK.
Select-DR-Scan	n_{IP}+8	
Capture-DR	n_{IP}+9	
Shift-DR	n_{IP}+10	TDO devine activ, iar TDI trebuie să fie valid pe frontul descrescător al semnalului de tact. Primul bit este introdus în TAP pe frontul crescător al semnalului de tact.
Shift-DR	11+n_{IP}+ n_{ID}	
Exit1-DR	12+n_{IP}+ n_{ID}	TDO devine inactivă pe frontul descrescător al semnalului de tact.
Update-DR	13+n_{IP}+ n_{ID}	Datele trimise sunt actualizate în DR pe frontul descrescător al semnalului de tact.
Select-DR-Scan	14+n_{IP}+ n_{ID}	
Select-IR-Scan	15+n_{IP}+ n_{ID}	
Test-Logic-Reset	16+n_{IP}+ n_{ID}	Testare completă.

Tabelul 4.2. Timpul de execuție pentru diferite stări JTAG

În continuare, se pune problema evaluării timpului de testare pe baza exemplului din figura 4.1, iar pentru a efectua o analiza concretă se vor verifica conexiunile electrice între circuitul SN74LVTH18504 şi celelalte circuite de pe placă (SN74LVTH18504 şi SN74BCT8245). În acest condiţii, toate circuitele de pe placă trebuie încărcate cu vectorii de test, adică se va rula de două ori instrucţiunea EXTEST pentru toate circuitele de pe placă. La prima încărcare/rulare, ieşirile circuitelor se încarcă prin JTAG cu vectorii de test, iar la a doua instrucţiune EXTEST, se citesc intrările.

Conform tabelului 4.2., numărul de impulsuri de tact (N_{TCK}) pentru a încărca instrucţiunea EXTEST şi un vector de test în fiecare circuit JTAG din exemplul 4.1. este dat de relaţia 4.1., unde N este numărul de circuite JTAG conectate în cascadă, N_{IP} este lungimea registrului de instrucţiune, iar N_{DP} este lungimea registrului de date pentru circuitul respectiv din buclă.

$$N_{TCK} = 16 + \sum_{1}^{N} N_{IP} + \sum_{1}^{N} N_{DP} \qquad (4.1.)$$

În concluzie, pentru a executa instrucţiunea EXTEST şi pentru a încărca un vector de test în două circuite din calea de scanare din exemplul 4.1., sunt necesare N_{TCK} impulsuri de test, conform relaţiei 4.2.:

$$N_{TCK} = 16 + 4*8 + 2*17 + 47 + 47 = 172 \text{ (impulsuri de tact)} \qquad (4.2.)$$

Dacă se utilizează un semnal de tact cu frecvenţa de 10 MHz rezultă că sunt necesare 16.8 μs pentru a transmite sau citi un vector de test de 130 biţi, ceea ce este acceptabil în majoritatea aplicaţiilor.

Conform specificaţiilor JTAG, pentru a încărca o instrucţiune într-un circuit din figura 4.1., trebuie să se parcurgă întreaga buclă; deci, conform relaţiei 4.1. rezultă că trebuie să aplicăm numărul de impulsuri din relaţia 4.3.

$$N_{TCK} = 16 + \sum_{1}^{N} N_{IP} + \sum_{1}^{N} N_{DP} = 16 + \sum_{1}^{N} N_{IP} = 16 + 4*8 = 48 \qquad (4.3.)$$

Dacă circuitele nu ar fi fost conectate în cascadă, pentru încărcarea unei instrucţiuni JTAG erau necesare 16 + 8 = 24 impulsuri de tact, faţă de 48 impulsuri de tact, conform relaţiei 4.3. Această analiză evidenţiază un dezavantaj al conectării în cascadă şi anume o creştere a timpului de încărcare a vectorilor de test.

Pentru încărcarea sau citirea datelor în sau din registrul de date al unui circuit JTAG din figura 4.1. nu este cazul să se parcurgă întreaga buclă de scanare, fiind posibilă

punerea în starea de BYPASS a celorlalte circuite de pe calea de scanare, astfel, timpul de scanare reducându-se simțitor. De exemplu, dacă se dorește încărcarea registrului de date al circuitului **SN74LVTH18504** în conformitate cu relația 4.1. este necesar să aplicăm numărul de impulsuri de tact din relația 4.4.

$$N_{TCK} = 16 + \sum_{1}^{N} N_{ID} = 16 + 3 * N_{BYPASS} + 47 = 66 \qquad (4.4)$$

În relația 4.4. s-a ținut cont de faptul că pentru parcurgerea unui circuit aflat în starea BYPASS este necesar să se aplice un singur puls de tact, iar registrul de date al circuitului SN74LVT18504 are 47 biți.

În tabelul 4.3. sunt sintetizate rezultatele prezentate anterior pentru exemplul din figura 4.1.

Condiții de test	**Număr impulsuri de tact**
Încărcare instrucțiuni şi date în toate cele 4 circuite	172
Încărcare instrucțiuni într-un circuit JTAG în cazul conectării în cascadă (serie)	48
Încărcare instrucțiuni într-un circuit JTAG dacă circuitul nu este conectat în cascadă	24
Încărcare registru de date pentru SN74LVTH18504	63
Încărcare registru de date pentru SN74BCT245	33
Încărcare instrucțiuni şi date pentru circuitul SN74LVTH18504	48+66=114

Tabelul 4.3. Numărul de impulsuri de tact pentru diverse operații

Din tabelul 4.3. rezultă că metoda de conectare în cascadă este eficientă dacă strategia de test impune transferul vectorilor de test pentru toate circuitele din lanț, dar durează cu aproximativ 50% mai mult pentru încărcarea instrucțiunilor şi numai cu 5% mai mult pentru încărcarea registrului de date (deoarece celelalte circuite din lanț pot fi aduse în starea BYPASS).

Un alt dezavantaj al conectării în cascadă este lipsa siguranței în funcționare, adică dacă se defectează un circuit din calea de scanare, testarea tuturor circuitelor din calea respectivă este compromisă.

4.1.2. Conectarea în stea a circuitelor JTAG

Din figura 4.2. se observă că la conectarea în stea semnalul de tact se aplică la toate circuitele, dar fiecare circuit are o linie separată TMS. În cazul conectării în stea, încărcarea unui vector de test într-un circuit JTAG se face direct fără a parcurge celelalte circuite şi ca atare pentru aceeaşi cantitate de date transmisă timpul de testare este mai redus decât în cazul conectării în cascadă.

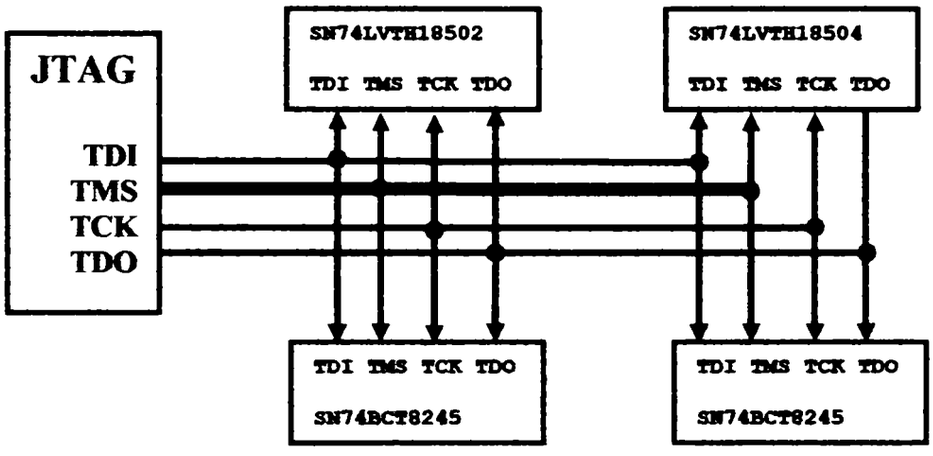


Figura 4.2. Topologia stea

În continuare, se pune problema determinării timpului de testare la verificarea conexiunilor electrice dintre circuitul SN74LVTH18504 şi celelalte circuite de pe placă (SN74LVTH18504 şi SN74BCT8245). În această situaţie, circuitul SN74LVTH18504 se încarcă cu vectorul de test, adică se va rula de două ori instrucţiunea EXTEST, iar ieşirile celorlalte circuite JTAG de pe placă trebuie aduse în stările dorite, adică vectorii de test trebuie încărcaţi şi în aceste circuite. La prima execuţie a instrucţiunii EXTEST se încarcă vectorul de test în circuitul SN74LVTH18504 şi se pun ieşirile la nivelul logic specificat în vectorul de test, iar la a doua instrucţiune EXTEST se citesc intrările. Timpul de execuţie a instrucţiunii EXTEST este dat de relaţia 4.5. şi rezultă din tabelul 4.1.

$$N_{TCK} = 16 + N_{IP} + N_{DP} = 16 + 8 + 48 = 80 \text{ (impulsuri de tact)} \quad (4.5.)$$

Dacă se compară timpul de încărcare a unui vector de test într-un circuit în cazul unei conectări în stea (relaţia 4.5.) cu timpul de încărcare a unui vector de test într-un circuit în cazul unei conectări în serie (4.2.), rezultă un important câştig de viteză (172/80 = 215%). În realitate, reducerea timpului de testare în cazul conectării stea, faţă de conectarea serie, nu este chiar atât de importantă şi depinde foarte mult de:

- numărul de circuite JTAG de pe placă;
- lungimile registrelor de date de pe placă;
- algoritmul de testare folosit;
- particularitățile constructive ale plăcii.

În concluzie, conexiunea stea prezintă următoarele avantaje față de conexiunea serie:

- timp mai redus de încărcare şi de citire a vectorilor de test;
- siguranță mai mare în funcționare. Defectarea unui circuit din calea de scanare va afecta calea respectivă, dar celelalte căi de scanare vor funcționa normal, fapt ce permite continuarea testării plăcii sau a sistemului electronic.

Dezavantajele conectării stea față de conectarea serie sunt:

- este necesar un controller JTAG, care să genereze semnalele TMS pentru fiecare circuit JTAG în parte;
- existența unor linii suplimentare TMS pentru fiecare circuit JTAG duce la creşterea complexității cablajului echipamentului respectiv.

Din cele prezentate în acest paragraf rezultă că: conectarea serie a circuitelor JTAG este recomandată în cazul echipamentelor mai puțin complexe (număr mai redus de circuite JTAG) a căror testabilitate în orice moment nu este foarte critică, iar conexiunea stea este recomandată în cazul aplicațiilor complexe sau critice din punct de vedere al testării.

4.2. Conectarea ierarhică a circuitelor JTAG

În cadrul paragrafelor precedente s-au prezentat topologiile de conectare serială şi în stea, ambele având avantaje şi dezavantaje. În cadrul acestui paragraf se va prezenta o metodă de conectare, care prezintă avantajele metodelor de conectare serie şi stea şi este foarte potrivită pentru testarea JTAG a unor sisteme complexe şi critice.

4.2.1. Protocolul ascuns

Conform specificațiilor 1149.1, busul utilizat prezintă minim 4 semnale:

- TMS - selectarea modului de test;
- TCK - semnalul de tact;
- TDO - ieşirea de date;
- TDI - intrarea de date.

Conform specificațiilor IEEE1141.1, linia TMS este utilizată pentru: a controla starea aplicaților JTAG; a scana datele de intrare; a se intra în starea IDLE sau a se intra în starea RESET. Dacă prin intermediul semnalul TMS se pune magistrala JTAG în starea IDLE (RT/IDLE, PAUSE-IR sau PAUSE-DR) sau în starea RESET (TLRST), toate aplicațiile sunt invalidate de a răspunde la datele transmise pe liniile TDI sau TDO, deci se poate implementa un protocol de transmitere a datelor pe liniile TDI şi TDO între TBC (Test Bus Controller) şi ASP (Addressable Shadow Port) sau HASP (Hierarchically Addresable Shadow Port).

Protocolul ascuns se compune din:

- protocolul de selecție;
- protocolul de confirmare.

Prin intermediul protocolului ascuns se transmite protocolul de selecție, care face legătura între TBC şi ASP, direct sau prin intermediul unuia sau a mai multor HASP. După transmiterea protocolului de selecție se recepționează protocolul de confirmare de la ASP-ul selectat, direct sau prin intermediul unuia sau a mai multor HASP-uri, la TBC. Protocolul de confirmare este utilizat pentru a indica realizarea conexiunii. Transmisia datelor se realizează pe liniile TDI-TDO utilizând metoda de semnalizare cu biții pereche. Semnalul TMS nu este implicat în protocolul ascuns, el fiind utilizat pentru a ține aplicațiile JTAG în starea PAUSE sau RESET pe durata transmiterii datelor protocolului ascuns. Metoda de semnalizare a perechilor de biți permite transmiterea datelor şi comenzilor pe un singur fir. Semnalele de control se utilizează pentru a porni şi opri protocolul de selecție şi de răspuns şi pentru a delimita mesajul de date. La protocolul de selecție şi confirmare se utilizează următoarele perechi de biți:

- Semnalul Idle (I) - semnal de control identificabil prin doi biți succesivi la nivelul logic „1" de la transmițător la receptor;
- Semnalul de selecție (S) - semnal de control identificabil prin doi biți succesivi la nivelul logic „0" de la transmițător la receptor;
- Date de nivel logic 1 (D) - semnal compus dintr-un bit de „0" urmat de un bit de „1" de la transmițător la receptor;
- Date de nivel logic 0 (D) - semnal compus dintr-un bit de „1" urmat de un bit de „0" de la transmițător la receptor.

Semnalele perechi de biți prezentate sunt transmise pe linia TDO a TBC-urilor, ASP-urilor sau HASP-urilor pe fronturile descrescătoare ale semnalului de tact TCK, iar intrările se citesc pe fronturile crescătoare ale semnalului de tact.

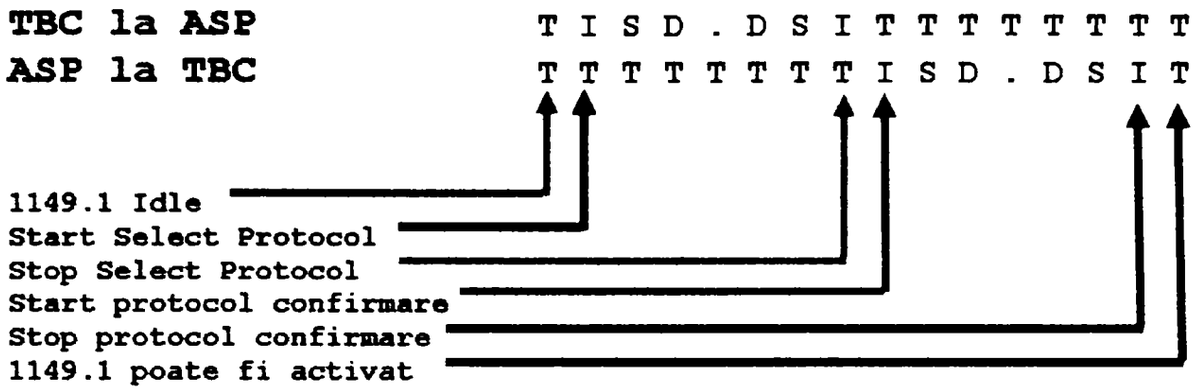


Figura 4.3. Protocolul de selecție şi confirmare

Din figura 4.3. se poate observa că înainte şi după secvența de protocol, ieşirile TDO ale TBC sau ASP sunt invalidate, iar busul este tras pe "1" logic prin rezistențe de Pull-up. Secvența "**I S D ..D S I T**" transmisă de la TBC la ASP este protocolul de selecție, iar secvența "**I S D..D S I**" transmisă de la ASP la TBC este protocolul de răspuns. După protocolul de selecție şi validare, ASP selectat este conectat la TBC, fiind gata pentru testarea JTAG. Semnalul "**I**" de la începutul fiecărei secvențe este folosit ca delimitator față de nivelul precedent "**T**". Aceasta permite evitarea intrării neintenționate în protocolul de selecție sau răspuns, când magistrala JTAG intră în starea IDLE sau RESET. Starea "**I**" de la sfârşitul protocolului de selecție indică terminarea protocolului. În cadrul secvenței "**S D ..D S**" din cadrul protocolului de selecție se transmite adresa ASP, accesată ca o secvență de biți de „1" şi „0".

4.2.2. Exemplu de accesare ierarhică

În următoarele exemple se prezintă modul în care TBC poate accesa ierarhic ASP-uri sau HASP-urile, prezentarea pornind de la structuri mai simple spre cele mai complexe. De asemenea, se va face o analiză comparativă a timpului de încărcare a vectorilor de test şi se vor analiza problemele de sincronizare, care pot apărea la sistemele pe mai multe nivele între liniile magistralei JTAG.

4.2.2.1. Sistem pe un singur nivel

În figura 4.4. se prezintă un sistem de testare pe un singur nivel, TBC-ul fiind conectat prin intermediul busului JTAG (cu 4 fire) la un număr de ASP-uri, iar fiecare ASP

este conectat la placa testată tot prin intermediul unui bus JTAG cu 4 fire. Vom nota ASP n:m, unde n indică adresa ASP, iar m indică nivelul pe care se găseşte ASP-ul.

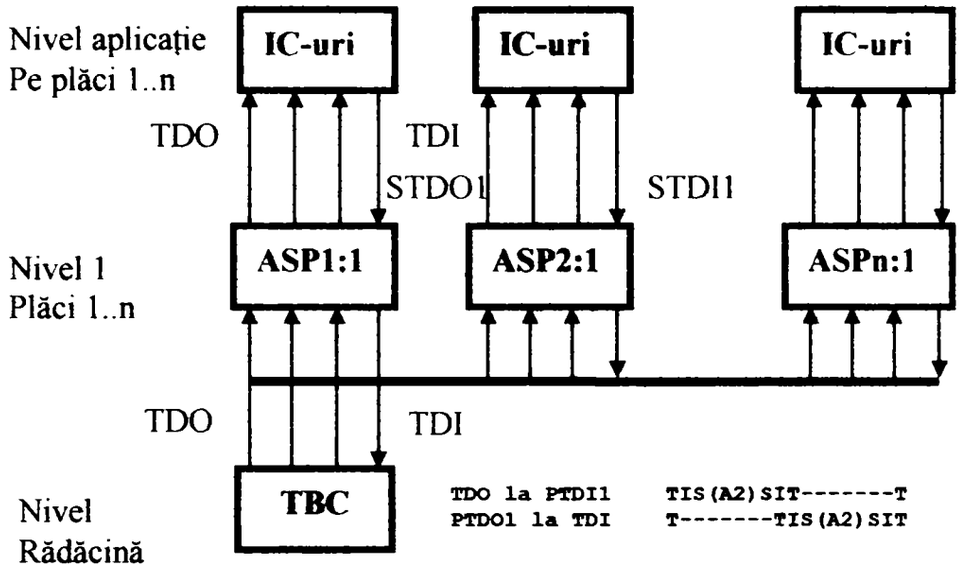


Figura 4.4. Sistem de test pe un singur nivel

Înainte de a accesa placa de aplicaţie trebuie făcută legătura între TBC şi placa ţintă. Pentru a face legătura între aplicaţia de pe ASP2:1 şi TBC, se va transmite protocolul de selecţie pe un singur nivel cu adresa A2, iar ASP2:1 va răspunde cu protocolul de validare. TBC va urmări conexiunea prin verificarea adresei de răspuns, iar dacă răspunsul corespunde, legătura este făcută şi se poate comunica prin protocolul 1149.1.

4.2.2.2. Sistem pe două nivele

În figura 4.5. este prezentat un sistem cu două nivele, TBC fiind conectat la un nivel HASP, iar HASP-urile sunt conectate la ASP-uri. În figura 4.5. se prezintă conexiunile doar pentru HASP1:1, dar şi celelalte HASP-uri sunt conectate la ASP-uri.

Înainte ca o aplicaţie să poate fi accesată de TBC, trebuie să se realizeze o conexiune ierarhică între TBC şi aplicaţie. Pentru a face conexiunea între aplicaţia de pe ASP2:2 şi TBC se va transmite un mesaj de "select protocol" de la TDO la PTDI1 la toate HASP-urile de pe nivelul 1. Protocolul de selecţie ierarhic pe două nivele diferă de protocolul de selecţie pe un singur nivel, prin adresa suplimentară transmisă. Astfel, prima

adresă din mesaj, A1, selectează HASP1:1, iar ce-a de a doua selectează ASP2:2. HASP1:1 va recepționa SA1S şi va urmări ce semnal va urma. Dacă apare un semnal "I" după mesajul de adresă, înseamnă că există un protocol doar pe un singur nivel şi trebuie trimis protocolul de răspuns. Dacă după mesajul de adresă urmează "S", HASP1:1 recunoaşte că protocolul de selecție este ierarhic şi va urma o adresă. Ca răspuns la începutul celei de-a doua adrese (SA2S) de la TBC, HASP1:1 validează ieşirea STDO1 şi trimite un semnal "I", după care retransmite adresa A2 la ASP-ul de pe nivelul 2. După ce TBC a terminat transmisia pe nivelele ierarhice ale protocolului de selecție, se setează "T" pe linia TDO şi se monitorizează linia TDI pentru protocolul de confirmare. De asemenea şi HASP1:1, după ce a terminat transmisia protocolului de selecție pune linia TDO pe "T" şi monitorizează linia TDI. După ce ASP2:2 a recepționat protocolul de confirmare de la HASP1:1, porneşte protocolul de răspuns la HASP1:1.

După semnalul "I", ASP2:2 trimite secvența (SA2S) de la ieşirea PTDO2 la intrarea STDI1 a HASP1:1. Ca răspuns la primul semnal "S", HASP1:1 validează ieşirea PTDO1 şi porneşte retransmiterea protocolului de validare de la ASP2:2 la TBC. După ce ASP2:2 a trimis mesajul de adrese la STDI1, ASP2:2 va trimite "I", care va determina ca HASP1:1 să retransmită pe linia PTDO1 mesajul de adresă recepționat de la ASP2:2.

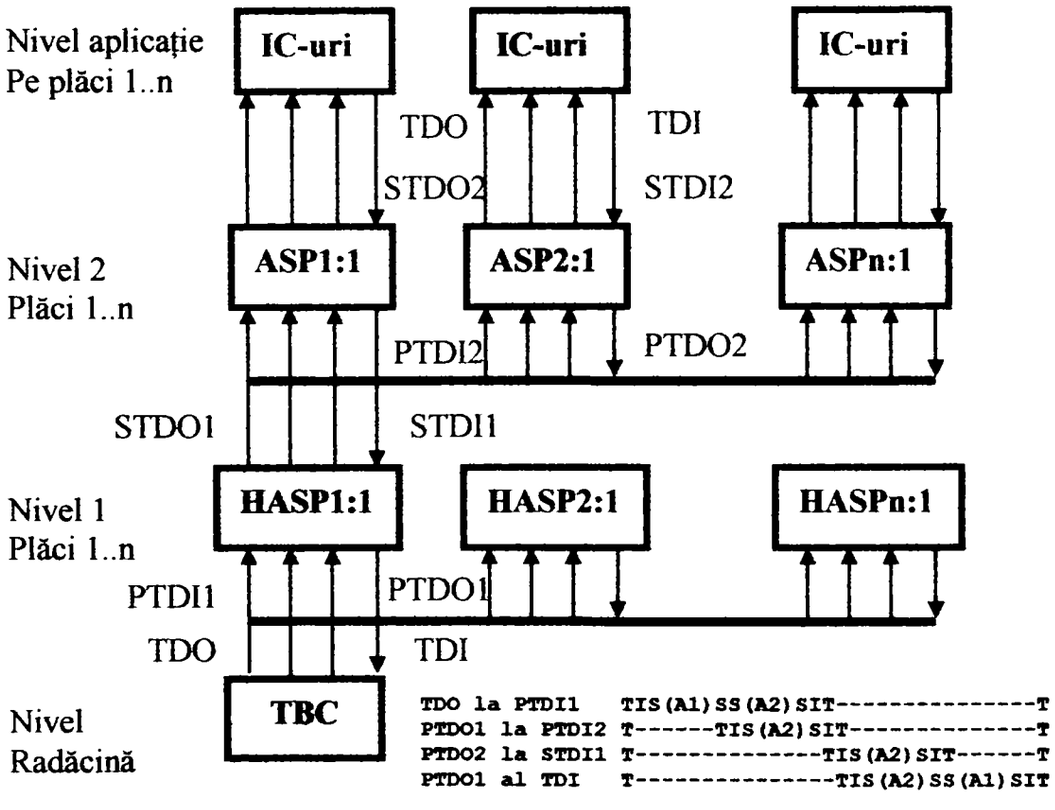


Figura 4.5. Sistem de test pe două nivele

4.2.2.3. Sisteme pe mai multe nivele

În paragrafele precedente s-a prezentat abordarea ierarhică pe un nivel şi pe două niveluri, dar metoda se poate extinde pe oricâte niveluri este necesar [44], [203], [240], [241], [245]. O problemă importantă care apare la sistemele pe mai multe niveluri, este sincronizarea transferului de date. Astfel, cu cât avem mai multe nivele ierarhice, cu atât timpul de propagare creşte, ceea ce limitează viteza de transmitere a datelor. Pentru a evita acest dezavantaj, se practică o sincronizare a liniilor HASP, o posibilă metodă prezentându-se în figura 4.6.

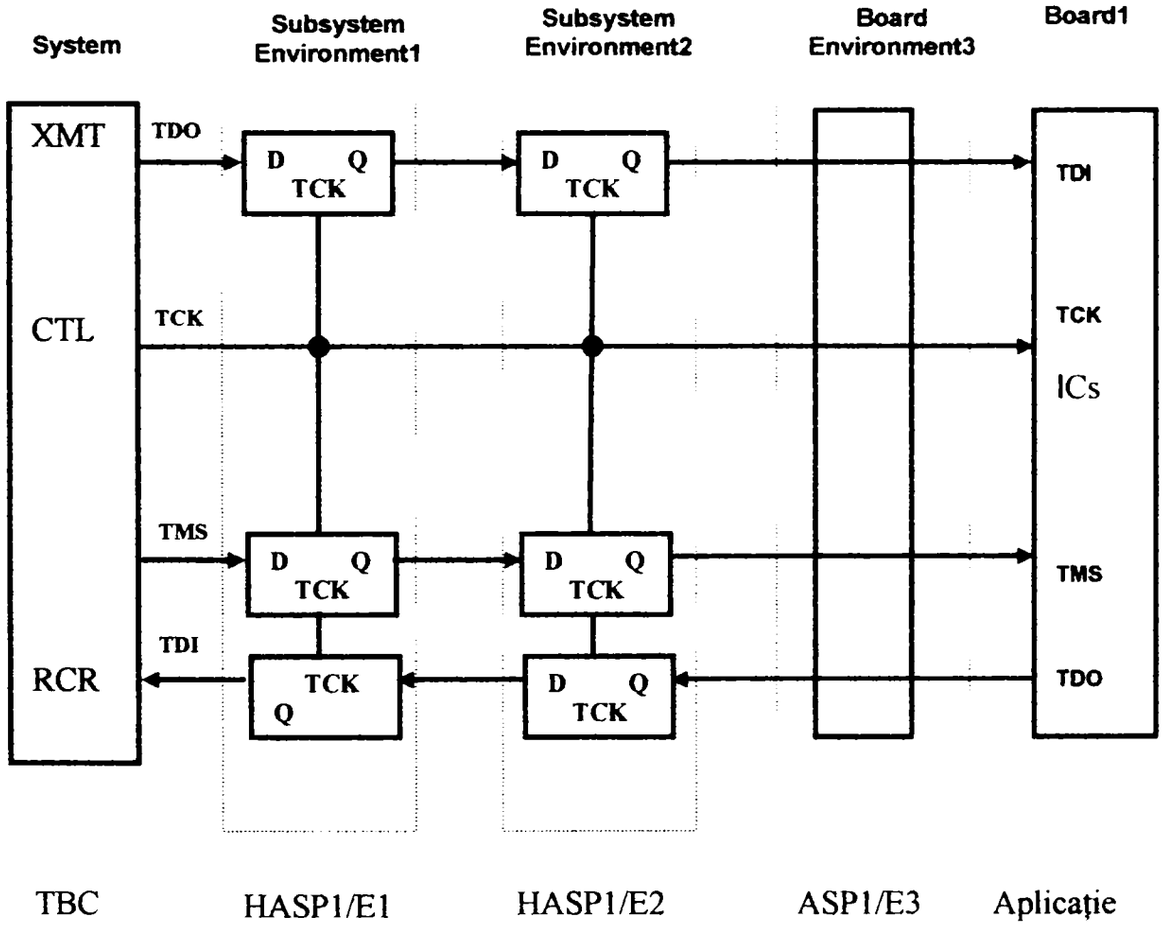


Figura 4.6. Sincronizarea liniilor HASP

4.2.2.4. Timpul de testare la testarea ierarhică

Pentru evaluarea timpul de testare se consideră cazul din fig. 4.4., în care IC-ul este de tipul SN74BCT8245, iar circuitele ASP sunt de tipul SN74ACT8997. Astfel la transmiterea protocolului de selecţie şi de confirmare se transmite următoarea secvenţă:

```
TDO la PTDI1     TIS(A2)SIT-------T
PTDO1 la TDI     T-------TIS(A2)SIT
```

Figura 4.7. Secvenţa de selecţie şi de confirmare

Având în vedere că circuitul SN74ACT8997 prezintă adresa de selecție pe 4 biți, din relația 4.7 rezultă numărul necesar de impulsuri de tact.

$$N_{TCK} = 2 * \sum_{1}^{N} \mathrm{TIS(A2)SIT} + 2 * \sum_{1}^{N} \mathrm{TIS(A2)SIT} = 2 * 18 = 36 \qquad (4.7)$$

Comparând rezultatele precedente cu cele din tabelul 4.3. rezultă că în configurația considerată, pentru încărcarea unui vector de test sunt necesare cel puțin 36 de impulsuri de tact pentru selectarea căii de comunicație şi confirmarea răspunsului, adică încărcarea ierarhică a unei instrucțiuni este cu (36+24-48)/(36+24) = 20% mai puțin eficientă decât configurația serie din figura 4.1.

Totuşi, configurarea ierarhică prezintă următoarele avantaje suplimentare:

- siguranța în funcționare este foarte bună, deoarece o defecțiune la un ASP sau o magistrală compromite cel mult ramura pe care este situat circuitul sau magistrala respectivă şi nu întreaga rețea;
- metoda se pretează foarte bine la testarea la nivel de sistem, iar fiecare ramură poate fi o placă separată sau un subansamblu separat;
- încărcarea suplimentară a busului JTAG, datorită protocolului de selecție şi confirmare a circuitelor ASP la un sistem complex, este mai redusă decât la conectarea serie;
- circuitele ASP sunt disponibile, relativ simple şi deci ieftine;
- nu este necesară o conexiune separată pentru fiecare circuit ca în cazul conexiunii stea;
- secvența protocolului de selectare/răspuns este dependentă de numărul de nivele ierarhice şi de numărul de biți alocați pentru adresa circuitului țintă, fiind independentă de complexitatea circuitelor JTAG de pe ramuri. În aceste condiții, eficiența transmiterii informației este mai mare în cazul sistemelor complexe care conțin circuite JTAG cu lungimi mari ale registrelor de date.

Dezavantajele conectării ierarhice:

- datorită existenței circuitelor ASP şi HASP de pe ramuri, costul acestei configurații este mai ridicat decât la conectarea serie şi se apropie de costurile conexiunii stea;
- protocolul de selecție şi confirmare pentru accesarea ierarhică a circuitelor țintă determină o încărcare suplimentară a busului JTAG.

În concluzie, conectarea ierarhică este potrivită pentru testarea sistemelor electronice complexe şi foarte complexe care conțin mai multe circuite compatibile JTAG

şi care necesită o bună sau foarte bună siguranță în exploatare, dar nu este recomandată în cazul sistemelor simple.

4.3. Concluzii

În cadrul acestui capitol autorul a insistat pe posibilitățile de conectare a circuitelor JTAG, analizând în detaliu avantajele şi dezavantajele fiecărei metode, scopul principal fiind compararea acestora şi propunerea unor recomandări de utilizare a diverselor configurații în funcție de necesitățile de testare în cadrul echipamentului sau sistemului respectiv. Se impune a se menționa că în literatura de specialitate există informații superficiale cu privire la posibilitățile de conectare a circuitelor JTAG, iar unele aspecte importante sunt neglijate.

În aceste condiții principalele contribuții personale ale autorului sunt:

- analiza în detaliu a conectării JTAG serie pentru un caz particular de 4 circuite JTAG şi în special aprofundarea timpului de încărcare/citire a vectorilor de test;
- analiza în detaliu a conectării JTAG stea pentru acelaşi caz particular de 4 circuite JTAG şi în special aprofundarea timpului de încărcare/citire a vectorilor de test;
- generalizarea cazurilor particulare analizate în detaliu şi compararea performanțelor conectării serie şi stea;
- aprofundarea conectării ierarhice a circuitelor JTAG, a avantajelor şi dezavantajelor acestora comparativ cu conectarea serie şi stea.

Ca urmare a analizei efectuată de autor în secțiunea 4.1. (***Posibilități standard de conectare a circuitelor JTAG***) rezultă că conectarea circuitelor JTAG în serie este recomandată în cazul unor sisteme cu maxim 4 circuite JTAG, timpul consumat la încărcarea vectorilor de test şi a instrucțiunilor prin intermediul lanțului serial fiind în acest caz acceptabil. Pe lângă simplitatea de conectare şi folosire, conectarea serie este mai ieftină decât celelalte metode de conectare şi este suportată de toate programele de generare automată a vectorilor de test. Din aprofundarea efectuată în secțiunea 4.1.2 (***Conectarea în stea a circuitelor JTAG***) a rezultat că dacă avem un sistem mai complex cu un număr mai mare de circuite JTAG, situate chiar pe plăci electronice diferite, este de preferat să folosim conectarea stea. Astfel, conectarea stea va oferi o siguranță sporită a testării întrucât defectarea unui circuit JTAG nu va compromite testarea JTAG a întregului echipament, permițând şi o încărcare mai rapidă a vectorilor de test în circuitele JTAG

componente. Deoarece există conexiuni electrice suplimentare pentru liniile TMS şi un controler de bus JTAG suplimentar, metoda de conectare stea implică un cost mai ridicat decât metoda serie.

În paragraful 4.2. (***Conectarea ierarhică***) autorul analizează o metodă de conectare ierarhică, aprofundează protocolul utilizat, analizează timpul de încărcare şi descărcare a vectorilor de test, iar în final prezintă avantajele şi dezavantajele acestei metode. Întrucât sistemele electronice complexe conectate ierarhic prezintă mai multe magistrale JTAG, defectarea unui circuit de pe o ramură va afecta doar ramura respectivă. Deoarece la conectarea ierarhică nu se utilizează semnale TMS pentru fiecare ramură în parte, preţul de implementare a acestei conectări este acceptabil, fiind mai mare decât la conectarea serie, dar de obicei mai mic decât la conectarea stea. Ca urmare a investigaţiilor făcute, autorul a constatat că circuitele ASP din nodurile JTAG sunt disponibile la mai mulţi producători, iar în rest se pot folosi aceleaşi circuite ca şi la conectarea serie sau stea.

Astfel din analiza efectuată de autor în cadrul paragrafului 4.2. a rezultat că conectarea ierarhică se pretează la sisteme complexe distribuite pe mai multe plăci electronice, care necesită o siguranţă mare în exploatare şi performanţe de viteză relativ bune.

Este necesar a se menţiona că soluţiile tehnice folosite la conectarea circuitelor JTAG sunt mature şi uşor de folosit, dar este foarte importantă alegerea metodei adecvate de conectare a circuitelor JTAG şi trebuie analizate cu atenţie elementele de noutate, care apar în special la softurile utilizate pentru generarea automată a vectorilor de test ([40], [41], [44], [78], [176], [181], [203], [214]).

Analiza metodelor de conectare JTAG şi analiza matematică a conectărilor JTAG, adică analiza timpului de încărcare şi descărcare a vectorilor de test este contribuţia personală a autorului, acest subiect nefiind analizat în detaliu în literatura de specialitate.

În finalul acestui capitol se impune a se menţiona că din analiza bibliografică făcută în capitolul 1 (STADIUL ACTUAL AL TESTĂRII ECHIPAMENTELOR ELEC-TRONICE COMPLEXE) a rezultat că metoda JTAG este o metodă în expansiune, deosebit de potrivită la testarea sistemelor electronice complexe. În capitolul 2 (ANALIZA STRUCTURALĂ ŞI POSIBILITĂŢILE OFERITE DE STANDARDUL IEEE 1149.1 PENTRU OPTIMIZAREA TESTĂRII ECHIPAMENTELOR NUMERICE COMPLEXE) s-au prezentat posibilităţile şi simplitatea testării JTAG. Din capitolul 3 (CONTRIBUŢII LA TESTAREA ECHIPAMENTELOR ELECTRONICE COMPLEXE UTILIZÂND JTAG) a reieşit în principal

că testarea funcţională a unei unităţi complexe şi critice este realizabilă utilizând JTAG. Iar din capitolul 4 (CONTRIBUŢII LA CONECTAREA CIRCUITELOR JTAG) s-a putut observa că topologia optimă de conectare a circuitelor JTAG depinde de aplicaţia concretă, iar conectarea circuitelor JTAG este relativ simplă şi cu mari posibilităţi de extindere. Un subiect important încă neanalizat în cadrul tezei este modul de generare a vectorilor de test, iar în capitolul următor autorul va analiza posibilităţile de automatizare a generării vectorilor de test prin folosirea modelelor în VHDL la nivel de circuit integrat, placă sau sistem electronic complex.

CAPITOLUL 5

5. CONTRIBUȚII LA MODELAREA ÎN VHDL A TESTĂRII JTAG

Ca urmare a analizelor efectuate de autor în capitolele precedente a rezultat că utilizarea JTAG la testarea echipamentelor şi a sistemelor electronice complexe prezintă o serie de avantaje față de celelalte metode de testare, dar pentru reducerea costurilor şi a timpului de dezvoltare a testelor este necesară automatizarea generării vectorilor de test. În aceste condiții, scopul acestui capitol este de a analiza posibilitățile de automatizare a generării vectorilor de test pentru un echipament sau sistem electronic complex şi de a concepe şi dezvolta o metodă de automatizare a generării vectorilor de test destinată testării JTAG.

5.1. Metode moderne de proiectare a echipamentelor electronice

Pe baza analizei bibliografice şi a activității practice efectuate în acest domeniu ([2], [3], [4], [15], [48], [56], [209], [234], [237], [238]) autorul tezei a constatat că odată cu utilizarea tot mai frecventă a circuitelor programabile (CPLD, FPGA şi PLD), a microcontrollerelor, a microprocesoarelor, precum şi a circuitelor ASIC, producătorii de circuite integrate au tendința să proiecteze circuitele respective pe baza modelelor în VHDL a sub-ansamblurilor componente. În aceste condiții, limbajul VHDL a devenit larg utilizat de producătorii de circuite integrate, de echipamente şi de sisteme electronice complexe, iar mulți dintre aceştia pun la dispoziția utilizatorilor şi modelele în VHDL a circuitelor produse uşurând astfel şi modelarea sistemului ([3], [36], [149], [234], [236], [237], [238], [247], [251], [254]).

Pe scurt, se poate spune că VHDL-ul a devenit un limbaj al proiectanților de circuite numerice, limbaj cu care se poate realiza modelarea şi simularea circuitelor numerice, a echipamentelor sau a sistemelor electronice, dar există totuşi dezavantajul că nu este rezolvată problema testării sistemului sau a circuitului produs. Pentru uşurarea testării circuitelor integrate şi a echipamentelor electronice, în ultimul timp producătorii preferă să adauge hardware şi soft suplimentar de testare şi diagnoză, iar tot mai frecvent sunt adăugate facilități de testare JTAG.

În consecință modelarea în VHDL se utilitează tot mai frecvent la proiectare părții electronice (circuit integrat, placă sau sistem electronic) iar testarea se face utilizând JTAG sau echipamente de testare configurabile. Modelarea circuitului integrat, a echipamentului sau sistemului electronic permite simularea şi verificarea proiectului, dar poate permite şi generarea automată a vectorilor de test.

În cadrul acestui capitol nu s-a intenționat o prezentare în detaliu a limbajului VHDL, pentru aceasta se poate studia documentația de specialitate ([2], [3], [4], [15], [149], [237], [210]), dar s-a considerat necesar să se exemplifice modelarea şi simularea în VHDL la nivel de sistem.

Este necesar a se sublinia că există programe produse de firme de prestigiu în domeniu testării care permit generarea automată a vectorilor de test pe baza modelelor în BSDL a circuitelor componente şi a schemei electrice de conectare. Dezavantajul major actual al acestor programe este că nu pot genera vectorii de test în situația unor sisteme cu microprocesor sau microcontroller întrucât funcționarea sistemului este dependentă de programul care se rulează în microcontroller sau microprocesor.

5.2. Modelarea magistralei JTAG

În anexa F1, se prezenta ca exemplu de analiză modelul în VHDL a magistralei JTAG, model care a fost implementat pentru un registru de date de 18 biți şi registrul de instrucțiune de 8 biți. Semnalele generate sunt compatibile 1149.1, iar programul a fost scris de autorul tezei în ActiveVHDL. La scrierea programului s-a avut în vedere că fiecare circuit integrat sau echipament care funcționează conform specificațiilor 1149.1 trebuie să funcționeze conform diagramei din figura 2.10 (vezi capitolul 2).

Din programul principal al modelului se observă că se inițializează semnalele de pe magistrală, se trimite instrucțiunea 11 HEXA în registrul de instrucțiuni, după care se încarcă 011101010000101001 în registrul de date, implementarea fiind relativ uşoară, iar simularea în ActivVHDL este intuitivă şi permite testarea uşoară a modelului.

Ca urmare a aprofundării şi implementării practice din această secțiune au rezultat următoarele concluzii principale:

- modelul în VHDL al magistralei JTAG este relativ simplu şi poate fi dezvoltat generic, adică se poate reutiliza uşor şi pentru alte lungimi ale registrelor de date şi instrucţiuni;
- este posibilă utilizarea modelului magistralei JTAG pentru sinteza hardware a părţii JTAG a circuitelor integrate;
- modulul prezentat poate fi considerat ca un modul separat care se integrează uşor în modelul VHDL al unui circuit integrat sau echipament electronic
- este posibilă extinderea relativ uşoară a modelului şi pentru circuite integrate complexe cu instrucţiuni private JTAG precum şi pentru circuitele cu facilităţi IEEE 1149.4.

5.3. Modelarea în VHDL a sistemului numeric pentru procesorul TMS320C203

În continuare se prezintă modelul în VHDL al unui sistem numeric (figura 5.1), care conţine un procesor TMS320C203, o memorie program de 32 x 16 biţi (de uz didactic), o memorie de date de 32K x 16 biţi (de uz didactic), un port de ieşire de 8 biţi şi un port de intrare de 8 biţi.

Se impune a se preciza că modelul procesorului a fost implementat de autor în colaborare ca urmare a unui contract cu firma ELBIT din Israel, că modelele circuitelor SN74BCT8374 şi SN74ACT8994 au fost preluate de pe Internet şi că celelalte componente ale sistemului au fost modelate şi implementate de autor.

Din figura 5.1. se poate observa că pentru a permite testarea sistemului prin JTAG, placa a fost echipată cu două bistabile octale de tipul SN74BCT8374, unul pentru intrări, unul pentru ieşirile numerice şi un monitor de bus SN74ACT8994, conectat pe busul de date.

Pentru a exemplifica modul în care se face modelarea în VHDL a unei plăci electronice şi arhitectura programui, se prezintă în continuare o parte a codului modelului în VHDL.

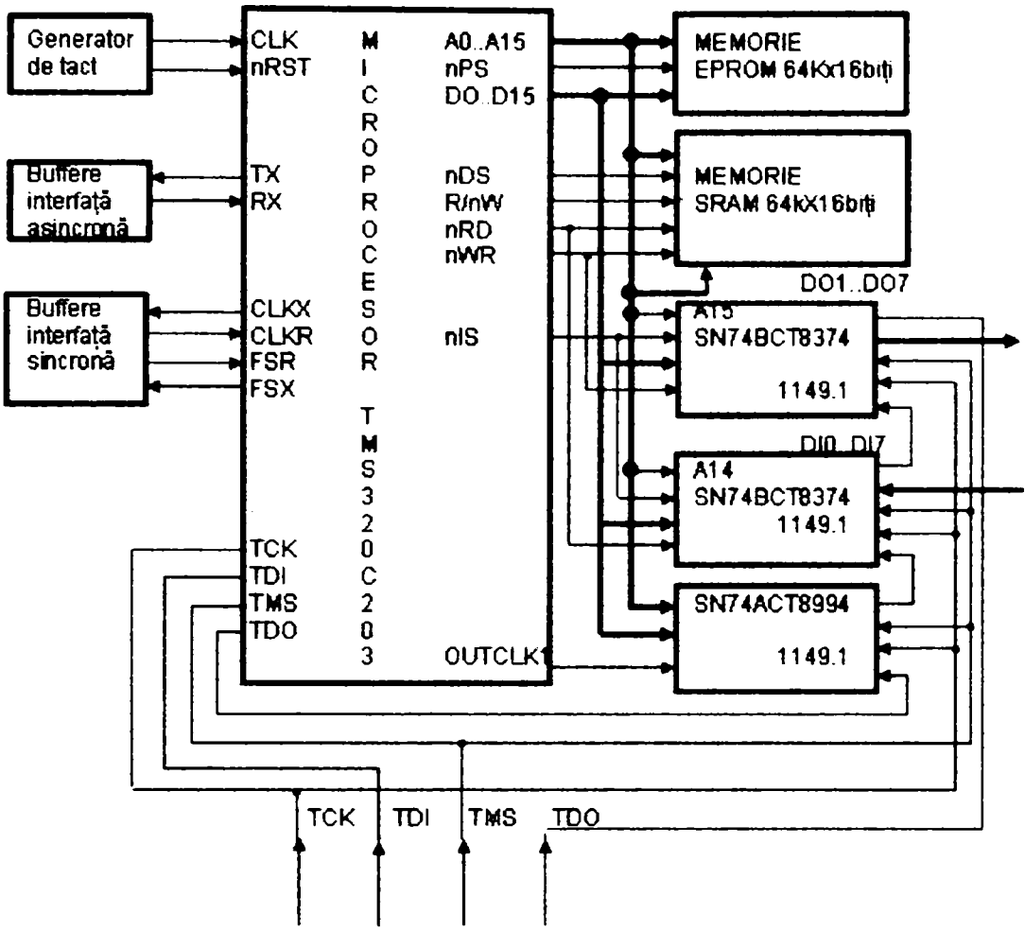


Figura 5.1. Exemplu de sistem modelat în VHDL

Fracment din codul sursă de modelare în VHDL a unui system cu procesor TMS320C203

```
--  File: SYSTEM_MODEL.vhd
--  created by Design Wizard: 05/27/98 21:44:37
--  @Demian Petru
library IEEE,work;
use IEEE.Std_Logic_1164.all;
use work.all;

entity SYSTEM_MODEL is
end SYSTEM_MODEL;

architecture SYSTEM_MODEL of SYSTEM_MODEL is
signal A          : STD_LOGIC_VECTOR (15 downto 0);
signal D          : STD_LOGIC_VECTOR (15 downto 0);
```

```
signal DI, DOUT    : STD_LOGIC_VECTOR (7 downto 0);
signal TCK, TMS, TMS320_TDI,TMS320_TDO, CLK, nRST, TX, RX, CLKX, CLKR,
FSR, FSX, nPS, R_nW, nRD, nWR, nIS, OUTCLK1, TX_UART, RX_UART, nCS, nOE,
nCS_RAM, nCS_OUT, OUT_TDI, OUT_TDO, nCS_IN, IN_TDI, IN_TDO, TDI,
       MONITOR_TDI, MONITOR_TDO     : STD_LOGIC;

-- COMPONENTA GENERATOR DE TACT
component Generator_tact
    port (
        CLK: out STD_LOGIC;
        nRST: out STD_LOGIC
    );
end component;

-- COMPONENTA BUFFER_UART (PORT SERIAL ASINCRON)
component Buffer_UART
    port (
        TX_UART: in STD_LOGIC;
        RX_UART: out STD_LOGIC
    );
end component;

-- COMPONENTA BUFFER_SINCRO (PORT SERIAL SINCRON)
component Buffer_Sincro
port (
        CLKX: in STD_LOGIC;
        CLKR: out STD_LOGIC;
        FSR: out STD_LOGIC;
        FSX: in STD_LOGIC
    );
end component;

-- COMPONENTA MEMORIE EPROM (MEMORIA DE PROGRAM)
component Memorie_EPROM
    port (
        D: inout STD_LOGIC_VECTOR (15 downto 0);
        A: inout STD_LOGIC_VECTOR (15 downto 0);
        nCS: in STD_LOGIC;
        nOE: in STD_LOGIC
    );
end component;
```

```
-- COMPONENTA MEMORIE SRAM (MEMORIA DE DATE)
component Memorie_SRAM
    port (
        D: inout STD_LOGIC_VECTOR (15 downto 0);
        A: in STD_LOGIC_VECTOR (15 downto 0);
        nRD: in STD_LOGIC;
        nWR: in STD_LOGIC;
        nCS_RAM: in STD_LOGIC
    );
end component;

-- COMPONENTA BUFFER_OUT (PORT DE IESIRE PE 8 BITI)
component Buffer_out
port (
        D: inout STD_LOGIC_VECTOR (15 downto 0);
        nWR: in STD_LOGIC;
        nCS_OUT: in STD_LOGIC;
        TCK: in STD_LOGIC;
        TMS: in STD_LOGIC;
        OUT_TDI: in STD_LOGIC;
        OUT_TDO: out STD_LOGIC;
        DOUT: out STD_LOGIC_VECTOR (7 downto 0)
    );
end component;

-- COMPONENTA BUFFER_IN (PORT DE INTRARE PE 8 BITI)
component Buffer_in
    port (
        D: inout STD_LOGIC_VECTOR (15 downto 0);
        nRD: in STD_LOGIC;
        nCS_IN: in STD_LOGIC;
        TMS: in STD_LOGIC;
        TCK: in STD_LOGIC;
        IN_TDI: in STD_LOGIC;
        IN_TDO: out STD_LOGIC;
        DI: in STD_LOGIC_VECTOR (7 downto 0)
    );
end component;

-- COMPONENTA MONITOR (MONITORUL DE BUS SN74ACT8994)
```

```
component MONITOR
    port (
        TMS: in STD_LOGIC;
        TDI: in STD_LOGIC;
        TCK: in STD_LOGIC;
        MONITOR_TDI: in STD_LOGIC;
        MONITOR_TDO: out STD_LOGIC;
        D: inout STD_LOGIC_VECTOR (15 downto 0)
    );
end component;

-- COMPONENTA TMS320 (PROCESORUL TMS320C203)
component TMS320
    port (
        TCK: in STD_LOGIC;
        TMS: in STD_LOGIC;
        TMS320_TDI: in STD_LOGIC;
        TMS320_TDO: out STD_LOGIC;
        CLK: in STD_LOGIC;
        nRST: in STD_LOGIC;
        TX: out STD_LOGIC;
        RX: in STD_LOGIC;
        CLKX: out STD_LOGIC;
        CLKR: in STD_LOGIC;
        FSR: in STD_LOGIC;
        FSX: out STD_LOGIC;
        A: inout STD_LOGIC_VECTOR (15 downto 0);
        D: inout STD_LOGIC_VECTOR (15 downto 0);
        nPS: out STD_LOGIC;
        R_nW: out STD_LOGIC;
        nRD: out STD_LOGIC;
        nWR: out STD_LOGIC;
        nIS: out STD_LOGIC;
        OUTCLK1: out STD_LOGIC
    );
end component;

for SYSTEM: TMS320 use entity work.TMS320(STRUCTURE);
for SYSTEM: Generator_tact use entity
            work.Generator_tact(Generator_tact);
for SYSTEM: Buffer_Sincro use entity work.Buffer_Sincro(Buffer_Sincro);
```

```
for SYSTEM: Buffer_UART use entity work.Buffer_UART(Buffer_UART);
for SYSTEM: Memorie_Eprom use entity work.Memorie_Eprom(Memorie_Eprom);
for SYSTEM: Memorie_SRAM use entity work.Memorie_SRAM(Memorie_SRAM);
for SYSTEM: Buffer_out use entity work.Buffer_out(Buffer_out);
for SYSTEM: Buffer_in use entity work.Buffer_in(Buffer_in);
for SYSTEM: Monitor use entity work.Monitor(Monitor);

begin

 SYSTEM1: Generator_tact  port map (CLK, nRST );
 SYSTEM2: Buffer_UART  port map (TX_UART, RX_UART );
 SYSTEM3: Buffer_Sincro port map (CLKX, CLKR, FSR, FSX);
 SYSTEM4: Memorie_EPROM port map (D, A, nCS, nOE);
 SYSTEM5: Memorie_SRAM  port map (D, A, nRD, nWR, nCS_RAM );
 SYSTEM6: Buffer_out port map (D, nWR, nCS_OUT, TCK, TMS, OUT_TDI,
          OUT_TDO, DOUT);
 SYSTEM7: Buffer_in  port map (D, nRD, nCS_IN, TMS, TCK, IN_TDI, IN_TDO,
          DI );
 SYSTEM8: MONITOR  port map (TMS, TDI, TCK, MONITOR_TDI, MONITOR_TDO,
          D);
 SYSTEM9: TMS320   port map (TCK, TMS, TMS320_TDI, TMS320_TDO, CLK,
          nRST, TX, RX, CLKX, CLKR, FSR, FSX, A, D, nPS, R_nW, nRD,
          nWR, nIS, OUTCLK1);
 -- se introduc procesele specifice utilizatorului modelului
end SYSTEM_MODEL;
```

Din codul sursă al modelului prezentat anterior se poate observa că acesta este construit modular, adică pentru fiecare componentă este definită interfața de tipul:

```
component TMS320
    port (
        TCK: in STD_LOGIC;
        TMS: in STD_LOGIC;
        .....
    );
end component;
```

iar modelul componentei este implementat într-un modul separat specificat printr-o declarare de tipul:

```
for SYSTEM: TMS320 use entity work.TMS320(STRUCTURE);
```

În figura 5.2. se prezintă rezultatul modelării în VHDL a procesorului TMS320C203 (simulare în ActiveVHDL), iar în anexa F2 sunt prezentate modelele în VHDL pentru procesor şi pentru testbench.

Analizând modelul în VHDL al procesorului TMS320C203 prezentat în anexa F2 se poate observa că acesta este implementat modular, fiind compus din modelele în VHDL ale modulelor componente (generatorul de tact, blocul de reset şi întrerupere, blocul de acces al memoriilor, portul serial sincron şi portul serial asincron). Autorul a ajuns treptat la această abordare datorită complexităţii mari a procesorului, precum şi din dorinţa de a simula cât mai exact funcţionarea reală a acestuia. Astfel, din punct de vedere hardware un procesor este compus din diferite module care sunt conectate la un generator de tact, care prezintă un număr de intrări, respectiv ieşiri şi care sunt conectate între ele şi cu mediul exterior. Prin abordarea modulară a procesorului s-a reuşit modelarea modulelor componente şi chiar să se asigure funcţionarea în paralel a acestora (de exemplu, transmisia unui caracter pe interfaţa serială asincronă se poate face în paralel cu o transmisie pe interfaţa serială sincronă) exact ca la procesorul real. În etapa finală a proiectului, autorul tezei a participat la integrarea modulelor sistemului şi la testarea funcţionării în ansamblu. Modelele prezentate au format obiectul unui contract de colaborare cu firma ELBIT din Israel şi au fost utilizate în principal pentru testarea funcţionării unui sistem numeric real considerând şi timpii de propagare ai componentelor reale. Prin rularea modelului prezentat anterior, luând în considerare diferite scenarii se pot constata foarte uşor problemele reale care ar putea apărea, de exemplu datorită timpului de răspuns al procesorului sau al memoriilor în diverse configuraţii ale tensiunii de alimentare, ale frecvenţei de lucru al procesorului sau a versiunii componentelor electronice utilizate.

Suplimentar autorul tezei a integrat şi modelul în VHDL pentru interfaţa JTAG rezultând astfel posibilitatea utilizării simulării la nivel de sistem la generarea automată a vectorilor de test pentru testarea JTAG. Astfel, pentru generarea automată a vectorilor de test în testbench-ul modelului s-au adăugat testele care urmează să fie rulate, iar rezultatele acestora s-au salvat în fişiere de date. Autorul a mai remarcat că dacă modelul în VHDL al circuitului integrat sau al echipamentului electronic este bine realizat, este posibilă introducerea în testbench-ul sistemului a instrucţiunilor JTAG, similar testării reale şi salvarea în fişiere de date a rezultatelor adică a vectorilor de test.

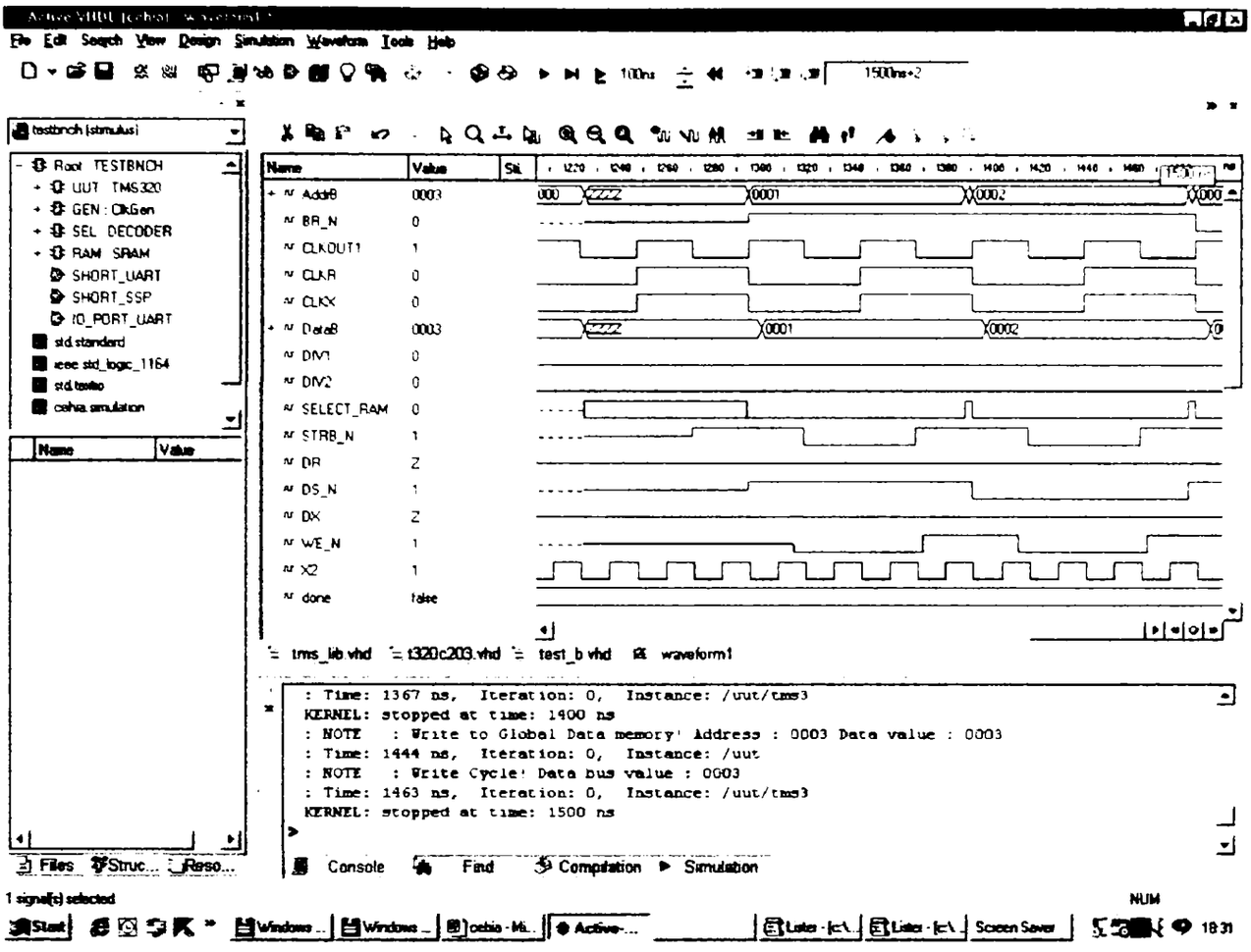


Figura 5.2. Modelarea în VHDL a procesorului TMS320C203

Ca urmare a parcurgerii bibliografiei şi a modelării practice în VHDL realizate de autor au rezultat următoarele concluzii importante:

- modelarea în VHDL la nivel de placă sau sistem electronic este realizabilă şi de cele mai multe ori este uşurată de disponibilitatea modelelor în VHDL ale componentelor electronice utilizate;
- modelul în VHDL la nivel de placă sau sistem electronic permite simularea sistemului înlocuind prototipul, reducându-se astfel timpul de dezvoltare şi costurile de dezvoltare;
- sunt disponibile medii de dezvoltare în VHDL care permit proiectarea circuitelor integrate, a echipamentelor şi sistemelor electronice în VHDL;
- modelul în VHDL la nivel de sistem poate fi utilizat cu modificări relativ reduse la generarea vectorilor de test pentru testarea JTAG. Dacă modelul în VHDL al sistemului este bine realizat şi conţine modelarea circuitelor JTAG din sistem nu este necesară intervenţia în model, generarea automată a vectorilor de test

realizându-se relativ simplu prin adăugarea testelor suplimentare în testbench-ul modelului;

- modelul în VHDL la nivel de sistem se poate utiliza şi pentru sinteză, ceea ce uşurează mult proiectarea circuitelor ASIC cu facilităţi de testare incluse.

5.4. Concluzii

Ca urmare a aprofundării testării JTAG din capitolele precedente, a rezultat că testarea circuitelor integrate, a echipamentelor şi a sistemelor electronice complexe utilizând JTAG prezintă o serie de avantaje faţă de celelalte metode de testare, dar se impune şi găsirea unor metode de generare automată a vectorilor de test. În aceste condiţii, scopul acestui capitol este de a analiza posibilităţile de automatizare a generării vectorilor de test pentru un echipament sau sistem electronic complex şi de a concepe şi implementa o nouă metodă de generare a vectorilor de test pentru testarea JTAG, autorul tezei aducând următoarele contribuţii importante:

- analiza pe baza literaturii de specialitate şi a implementărilor practice realizate personal a tendinţelor în proiectarea circuitelor integrate, a echipamentelor şi a sistemelor numerice;
- proiectarea şi realizarea modelului în VHDL a magistralei JTAG;
- proiectarea şi realizarea în colaborare a modelului în VHDL a procesorului TMS320C203, a memoriilor EEPROM, FLASH şi RAM şi a circuitului SN74ACT8994;
- modelarea şi simularea în VHDL a unui sistem complet cu procesorul TMS320C203 şi simularea funcţionării acestuia în diverse condiţii (diverse frecvenţe de tact ale procesorului, diferite tipuri de memorii, diverse variante de procesor şi diferite tensiuni de alimentare);
- implementarea generării automate a vectorilor de test pentru testarea la nivel de sistem utilizând JTAG pentru un sistem electronic modelat în VHDL care conţine un procesor TMS320C203 (figura 5.1);

Astfel, în secţiunea 5.1. (***Metode moderne de proiectare a echipamentelor electronice***), pe baza analizei efectuate de autor, se subliniază că limbajul VHDL se utilizează frecvent la proiectarea circuitelor electronice, iar modelele în VHDL ale componentelor devin mai accesibile. Autorul a mai subliniat că în condiţiile în care sunt

disponibile modelele în VHDL ale circuitelor componente ale unui sistem electronic se poate face relativ uşor simularea sistemului, reducându-se semnificativ costurile de dezvoltare a produsului.

Secţiunea 5.2. (***Modelarea magistralei JTAG***) prezintă modelul în VHDL a magistralei JTAG (model realizat de autor) şi rezultatele simulării. Din această secţiune rezultă că modelarea în VHDL a părţii JTAG a unui circuit se poate face relativ simplu şi permite modelarea cu acurateţe a comportării reale. Suplimentar, modelul poate fi utilizat şi pentru sinteză hardware (proiectarea circuitului respectiv), este modular şi poate fi uşor integrat în alte sisteme.

În secţiunea 5.3. (***Modelarea în VHDL a sistemului numeric pentru procesorul TMS320C203***) autorul a prezentat un model în VHDL a unui sistem cu procesorul TMS320C203 şi a analizat problematica modelării în VHDL a unui procesor. Ca urmare a aprofundării de către autor a acestui domeniu a rezultat că prin modelare VHDL la nivel de sistem se poate face o eficientă simulare a sistemului, se pot genera automat vectorii de test şi se poate face chiar şi sinteza circuitului (proiectarea ASIC-ului). Pentru informaţii suplimentare în anexa F2 este prezentată şi o parte a codului sursă al modelului în VHDL al procesorului TMS320C203, model dezvoltat în colaborare de către autor.

Având în vedere informaţiile prezentate în acest capitol se estimează că modelarea, simularea în VHDL şi testarea JTAG vor constitui instrumente obligatorii pentru realizarea unor circuite integrate, echipamente şi sisteme electronice complexe şi fiabile la un preţ rezonabil. Se impune a se menţiona că acest capitol este original, iar rezultatele au fost prezentate la IFAC 2000 (IFAC Workshop Ostrava, Czech Republic, 8-9 February 2000) şi au fost aplicate practic la modelarea în VHDL la nivel de sistem de către firma Elbit.

CAPITOLUL 6

6. CONCLUZII

Această teză, rezultat al activității de perfecționare prin doctorat, corelată cu preocupările profesionale ale autorului, reflectă contribuțiile originale şi realizările concrete ale acestuia în domeniul testării în ultimii ani.

Activitatea laborioasă de cercetare şi concepția inovativă a autorului în domeniul testării s-au manifestat prin conceperea de soluții de testare a echipamentelor electronice de aviație şi de automobile, prin implementarea de metode de modelare şi simulare în VHDL la nivel de sistem, prin publicarea unor lucrări de specialitate, respectiv participări la conferințe şi sesiuni de comunicări.

În continuare se prezintă principalele contribuții ale autorului, analizându-se pe capitole realizările acestuia şi concluziile rezultate din activitatea de cercetare pentru teza de doctorat.

CAPITOLUL 1. STADIUL ACTUAL AL TESTĂRII ECHIPAMENTELOR ELECTRONICE COMPLEXE

Acest capitol introductiv îşi propune să analizeze principalele grupe de echipamente de testare existente pe piață, precum şi caracteristicilor ale acestora.

Pe baza studiului bibliografic şi a experienței în domeniu, autorul a realizat următoarea clasificare a echipamentelor de testare:

- echipamentele de testare dedicate producției;
- echipamentele de testare configurabile;
- echipamentele de testare bazate pe JTAG.

Din studiul bibliografic efectuat de autor au rezultat următoarele concluzii principale:

- echipamentele de testare dedicate producției sunt de regulă special adaptate pentru un anumit produs şi sunt foarte scumpe, iar în cazul unor produse mai complexe au apărut limite tehnologice. Cele mai multe echipamente de testare dedicate producției, se bazează pe testarea clasică cu ace, ca urmare nu mai pot ține pasul cu tendințele de miniaturizare;

- echipamentele de testare configurabile sunt foarte diverse, existând deja o serie de firme producătoare importante, fiind posibil ca utilizarea acestora să se extindă în defavoarea echipamentelor de testare dedicate producției;
- echipamentele de testare bazate pe JTAG, deşi au apărut doar după 1990, au toate şansele să fie tot mai frecvent utilizate, inclusiv în producție. Se manifestă deja tendința de a considera echipamentele bazate pe JTAG ca o soluție standard şi de a proiecta echipamentele electronice ținând cont de considerentele de testare JTAG;
- testarea utilizând JTAG are şanse reale de dezvoltare, existând multe breşe insuficient aprofundate în care se pot obține rezultate importante atât teoretice, cât şi practice.

În urma analizei efectuate în acest capitol introductiv au rezultat următoarele direcții care necesită aprofundare şi pe care autorul le-a urmărit în cadrul capitolelor următoare ale tezei:

- analiza posibilităților de testare cu JTAG, a domeniului de utilizare, a limitărilor existente şi a performanțelor testării;
- sinteza limbajelor utilizate la testarea IEEE 1149.1 şi exemplificarea practică a acestora în cazul unor proiecte reale;
- analiza performanțelor la testarea JTAG şi propunerea unor metode şi soluții superioare celor utilizate în prezent;
- analiza posibilităților de testare cu ajutorul echipamentelor de testare configurabile şi cu ajutorul testării JTAG şi compararea acestora;
- analiza posibilității de utilizare a testării JTAG la nivel de circuit integrat, echipament electronic sau sistem electronic complex;
- abordarea ierarhică a testării JTAG şi analiza timpilor de testare;
- analiza proiectării circuitelor integrate şi a sistemelor numerice avându-se în vedere considerentele de testare;
- generarea automată a vectorilor de test pentru testarea cu JTAG pe baza modelării în VHDL a circuitului integrat, a echipamentului electronic sau a sistemului electronic.

CAPITOLUL 2. ANALIZA STRUCTURALĂ ŞI POSIBILITĂŢILE OFERITE DE STANDARDUL IEEE 1149.1 PENTRU OPTIMIZAREA TESTĂRII ECHIPAMENTELOR NUMERICE COMPLEXE

În cadrul acestui capitol autorul a aprofundat şi analizat standardul IEEE 1149.1, IEEE 1149.1b, IEEE 1149.4 şi IEEE 1149.5 cu scopul de a realiza o analiză succintă a principiilor şi metodelor aferente testării JTAG în strânsă legătură cu obiectivele propuse în cercetarea posibilităţilor de optimizare a testării echipamentelor electronice complexe.

Principalele contribuţii personale ale autorului în cadrul acestui capitol sunt:

- sinteza materialului bibliografic referitor la testarea JTAG şi prezentarea acestuia sub o formă cât mai simplă pentru cititor, motiv pentru care autorul a folosit frecvent figuri explicative şi a introdus exemple sugestive;
- analiza problematicii testării clasice şi a testării JTAG, precum şi furnizarea de exemple concrete;
- sinteza informaţiilor referitoare la standardele IEEE 1149.4 şi IEEE 1149.5 şi compararea acestora cu IEEE 1149.1;
- identificarea unor aspecte din domeniul testării utilizând JTAG insuficient aprofundate şi care merită să fie analizate în detaliu în cadrul tezei.

În urma aprofundării şi a analizelor efectuate de autor au rezultat următoarele concluzii principale:

- la nivelul circuitului există o magistrală sincronă cu 4-5 fire compatibil TTL foarte uşor de utilizat şi ieftin;
- comunicaţia se face sincron existând o linie de ieşire date şi una de intrare date;
- fiecare circuit cu facilităţi JTAG conţine o parte hardware suplimentară, care asigură toate operaţiile specifice JTAG;
- fiecare pin de intrare sau ieşire a circuitelor cu facilităţi JTAG poate fi citit sau setat prin intermediul celulelor de scanare;
- în cazul testării JTAG sondele clasice cu ace s-au mutat de pe cablaj în interiorul circuitelor;
- toate circuitele JTAG, indiferent de complexitate, prezintă anumite stări bine determinate în funcţie de semnalele ce se aplică pe magistrala JTAG (figura 2.10.);
- există 4 instrucţiuni JTAG obligatorii, care asigură operaţiile JTAG elementare, dar producătorul de circuite integrate adaugă uneori instrucţiuni suplimentare;
- IEEE 1149.5 standardizează comunicaţia pentru test, diagnoză şi întreţinere, asigurând un Test Control Master şi până la 250 module „slave". IEEE 1149.5 este

potrivit pentru testarea la nivel de placă electronică, comunicația între Master şi Slave făcându-se pe bază de pachete cu corecții de erori, dar dezavantajul major al acestui standard este lipsa circuitelor suport;

- standardul IEEE 1149.4 îşi propune să reducă dificultățile testării analogice şi să faciliteze proiectarea din considerente de testare;
- în ultimi ani au apărut completări ale standardului IEEE 1149.1 care extind domeniul de aplicație al testării pe frontieră.

CAPITOLUL 3. CONTRIBUȚII LA TESTAREA ECHIPAMENTELOR ELECTRONICE COMPLEXE UTILIZÂND JTAG

Principalele contribuțiile personale în acest capitol sunt:

- conceperea, dezvoltarea şi implementarea metodei de simulare a magistralei JTAG pe portul paralel al calculatoarelor compatibile PC;
- conceperea, dezvoltarea şi implementarea driverului pentru placa VX4491 în mediul de dezvoltare PAWS;
- analiza comparativă pe baza rezultatelor practice între simularea magistralei JTAG pe portul paralel şi utilizarea unor echipamente dedicate de interfață JTAG;
- analiza practică a performanțelor echipamentelor de interfață JTAG şi recomandarea unor soluții practice de îmbunătățire a performanțelor stației de testare;
- conceperea, implementarea şi testarea practică a unor programe de testare JTAG în limbajele BSDL, HSDL şi SVF, precum şi prezentarea unor recomandări rezultate din practică;
- conceperea, dezvoltarea şi implementarea metodei de testare a unității de control a încărcăturii Boeing 777 prin testarea JTAG; elaborarea unor recomandări pe baza rezultatelor practice şi prezentarea acestora în paginile IFAC 2000 (IFAC Workshop Ostrava, Czeck Republic, 8-9 February 2000) [75];
- sinteza necesităților de testare a echipamentelor electronice auto şi prezentarea pe baza experienței practice a soluțiilor posibile;
- analiza comparativă pe stația SMART-2 a testării JTAG şi a testării cu echipamente configurabile.

Concluziile principale rezultate din acest capitol sunt:

- este posibilă simularea magistralei JTAG pe portul paralel al unui calculator PC independent de complexitatea conectării circuitelor JTAG din lanț şi de numărul

acestora, dar frecvența maximă a semnalului de tact este dependentă în principal de performanțele calculatorului şi de sistemul de operare utilizat;

- magistrala JTAG este simplă, ieftină (4 sau 5 semnale) şi uşor de utilizat. Soluția cea mai accesibilă de realizare a unei magistrale JTAG este utilizarea portului paralel al unui calculator PC;
- la proiectarea unei stații de test este recomandabilă analiza prealabilă atentă a necesităților de transfer de date între diferitele echipamente şi alegerea soluției adecvate (conectarea directă prin GPIB, conectarea în rețea sau prin USB);
- întrucât echipamentele dedicate de interfațare cu magistrala JTAG (de exemplu VX4491) sunt relativ complexe şi pot funcționa relativ autonom, procedura de testare se poate optimiza, adică la începutul testelor se pot încărca vectorii de test, vectorii de răspuns corecți şi programul de test în memoria tampon a echipamentului, după care intern în placa de interfață independent de calculatorul master se rulează testele necesare, iar la final rezultatele sunt furnizate de către placa de interfață calculatorului master;
- frecvent echipamentele dedicate de interfațare cu magistrala JTAG dispun de mai multe canale, fiind astfel posibilă rularea testelor în paralel pe mai multe magistrale JTAG;
- există serii de circuite JTAG de suport (buffere, monitoare de bus, distribuitoare de bus JTAG etc.) produse de firme de prestigiu, iar în ultimul timp tot mai mulți producători de circuite integrate numerice fac publice (uneori şi pe Internet) modelele în BSDL şi VHDL ale circuitelor produse, ceea ce reduce semnificativ timpul de dezvoltare a mediului de testare JTAG, contribuie la creşterea calității testării şi la reducerea costurilor asociate;
- tot mai multe circuite integrate (FPGA, CPLD, PLD, microprocesoare şi microcontrollere) prezintă facilități de testare JTAG;
- există deja o standardizare în domeniul limbajelor utilizate la testarea JTAG (BSDL, HSDL şi SVF);
- testarea JTAG se pretează foarte bine la testarea circuitelor integrate, a plăcilor şi sistemelor electronice care au fost proiectate din considerente de testabilitate, situație în care se poate ajunge la o testabilitate apropiată de 100%;

- utilizarea unei stații de test performante şi a unui mediu de dezvoltare adecvat permite reducerea semnificativă a timpului de dezvoltare a programelor de test şi asigură flexibilitatea şi resursele hardware şi software necesare;
- deşi unitatea de control a încărcăturii Boeing 777 este relativ complicată, testarea acesteia se face relativ simplu iar rezultatele testelor pot indica precis localizarea defecțiunii;
- prin optimizarea resurselor hardware şi software ale stației de test este posibilă utilizarea testării JTAG şi la producția de circuite integrate, echipamente şi sisteme electronice. Astfel este posibilă reducerea semnificativă a timpului de testare prin încărcarea prealabilă a vectorilor de test şi a celor de răspuns aşteptat în memoria FLASH a interfeței JTAG, prin creşterea frecvenței de tact a magistralei JTAG şi prin proiectarea mai atentă din considerente de testare a circuitului integrat, a plăcii sau a sistemului electronic testat;
- programele suplimentare de conversie a formatului vectorilor de test implementate de autor au permis automatizarea generării vectorilor de test şi au redus semnificativ timpul de dezvoltare a mediului de testare fiind recomandabilă utilizarea lor în practică;
- metoda de testare JTAG se poate aplica foarte bine şi în cazul în care echipamentul testat este compus din mai multe plăci electronice;
- testarea JTAG se pretează şi la testarea echipamentelor electronice complexe, care necesită o foarte mare siguranță în exploatare (de exemplu, unitatea de control a încărcăturii);
- metoda de testare JTAG este deja folosită şi există o serie de firme de prestigiu care oferă echipamente de testare JTAG;
- metoda de testare JTAG se poate utiliza şi la echipamentele electronice auto de complexitate mare şi este uşurata de faptul că o serie de microcontrolere utilizate prezintă facilități JTAG.
- costul unui echipament de testare bazat pe JTAG este cu cel puțin un ordin de mărime mai redus decât acela al unui echipament de testare clasic (cu sonde);
- pentru aplicarea eficientă a testării cu metoda JTAG este necesar să se țină cont de criteriile de testabilitate încă din fazele inițiale de proiectare a echipamentului respectiv.

CAPITOLUL 4. CONTRIBUȚII LA CONECTAREA CIRCUITELOR JTAG

Principalele contribuții personale în acest capitol sunt:

- analiza în detaliu a conectării JTAG serie pentru un caz particular de 4 circuite JTAG şi în special aprofundarea timpului de încărcare/citire a vectorilor de test;
- analiza în detaliu a conectării JTAG stea pentru acelaşi caz particular de 4 circuite JTAG şi în special aprofundarea timpului de încărcare/citire a vectorilor de test;
- generalizarea cazurilor particulare analizate în detaliu şi compararea performanțelor conectării serie şi stea;
- aprofundarea conectării ierarhice a circuitelor JTAG, a avantajelor şi dezavantajelor acestei conexiuni comparativ cu conectarea serie şi stea.

Ca urmare a analizei bibliografice şi a implementărilor practice realizate de autor au rezultat următoarele concluzii:

- conectarea circuitelor JTAG este relativ simplă, metodele existente (serie, stea sau conectare ierarhică) fiind suportate de majoritatea programelor existente pe piață;
- metoda de conectare serie a circuitelor JTAG este preferată la sisteme de mică complexitate, fiind cea mai ieftină şi mai uşor de realizat;
- metoda de conectare în stea a circuitelor JTAG implică costuri hardware mai mari decât metoda serie, dar are şi o siguranță în funcționare mai ridicată decât metoda serie (distrugerea unui circuit nu determină întreruperea lanțului);
- la sistemele complexe (cu mai mult de 4 circuite JTAG) lungimea traseului de scanare poate deveni importantă (sute de stări), ceea ce influențează negativ timpul de testare;
- metoda de conectare a circuitelor JTAG în serie este convenabilă dacă strategia de test implică transferul vectorilor de test tuturor circuitele din lanț, dar este mai lentă cu aproximativ 50% la încărcarea instrucțiunilor (la un lanț de 4 instrucțiuni) şi doar cu 5% la încărcarea registrului de date (deoarece celelalte circuite din lanț pot fi aduse în starea BYPASS);
- se pot obține reduceri semnificative ale timpului de încărcare a vectorilor de test prin optimizarea modului de testare;
- conectarea ierarhică combină avantajele conectării serie şi stea fiind potrivită pentru sistemele electronice de complexitate medie şi mare;

- încărcarea suplimentară a magistralei JTAG ca urmare a protocolului de selecție şi confirmare în cazul conectării ierarhice este de regulă acceptabilă (<10%);
- siguranța în funcționare a conectării ierarhice este relativ mare, întrucât defectarea unui circuit de pe o ramură afectează doar ramura respectivă;
- la sistemele complexe cu mai multe niveluri este necesar să se realizeze sincronizarea liniilor de date (figura 4.6.);
- în scopul reducerii timpului de încărcare a vectorilor de test există tendința de a creşte frecvența de tact a magistralei JTAG (actualmente este de ordinul zecilor de MHz).
- soluțiile tehnice folosite de IEEE 1149.1 din punct de vedere al conectării sunt mature şi relativ uşor de folosit, iar elementele de noutate apar în special în programele tot mai complexe utilizate pentru generarea automată a vectorilor de test;
- la sistemele complexe generarea automată a vectorilor de test pentru testarea JTAG devine o necesitate.

CAPITOLUL 5. CONTRIBUȚII LA MODELAREA ÎN VHDL A TESTĂRII JTAG

Principalele contribuții personale ale autorului în acest capitol sunt:

- analiza pe baza literaturii de specialitate şi a implementărilor practice realizate de autorul tezei a tendințelor în proiectarea circuitelor integrate, a echipamentelor şi a sistemelor numerice;
- proiectarea şi realizarea modelului în VHDL a magistralei JTAG;
- proiectarea şi realizarea în colaborare a modelului în VHDL a procesorului TMS320C203, a memoriilor EEPROM, FLASH şi RAM şi a circuitului SN74ACT8994;
- modelarea şi simularea în VHDL a unui sistem complet cu procesorul TMS320C203 şi simularea funcționării acestuia în diverse condiții (diverse frecvențe de tact ale procesorului, diferite tipuri de memorii, diverse variante de procesor şi diferite tensiuni de alimentare);
- implementarea generării automate a vectorilor de test pentru testarea la nivel de sistem utilizând JTAG pentru un sistem electronic modelat în VHDL care conține un procesor TMS320C203 (figura 5.1);
- rezultatele modelării în VHDL obținute de autorul tezei au fost prezentate la IFAC 2000 (IFAC Workshop Ostrava, Czech Republic, 8-9 February 2000). De

asemenea, modelele în VHDL prezentate au fost implementate şi dezvoltate de autorul tezei pentru firma ELBIT din Israel.

Ca urmare a analizei bibliografice şi a implementărilor practice realizate de autor au rezultat următoarele concluzii:

- proiectarea în VHDL a circuitelor numerice este des utilizată, iar în ultimul timp se constată o creştere a importanţei modelării, simulării şi sintezei în VHDL, atât la nivel de circuit, cât şi la nivel de sistem;
- limbajul BSDL este un subset al limbajului VHDL, deci la realizarea modelului în BSDL al unei componente, pentru testarea JTAG, se poate utiliza modelul în VHDL al componentei respective;
- modelarea şi simularea în JTAG a busului de testare pe frontieră este uşor de realizat;
- modelarea, simularea şi sinteza modelului unui circuit electronic JTAG în VHDL este relativ uşor de realizat;
- modelarea şi simularea în VHDL a unui circuit complex (microcontroller) este posibilă şi poate fi utilizată la modelarea şi simularea unui sistem electronic sau chiar la sinteza circuitelor respective;
- prin modelarea şi simularea în VHDL la nivel de sistem se poate analiza în detaliu, încă din fazele de dezvoltare a proiectului, comportarea circuitului integrat, a echipamentului sau sistemului electronic, la un preţ mult mai redus;
- prin modelarea şi simularea în VHDL la nivel de circuit integrat, echipament sau sistem electronic se pot genera şi vectorii de test pentru testarea automată, inclusiv pentru testarea JTAG.

În ansamblu, în activitatea de pregătire prin doctorat autorul s-a concentrat pe optimizarea testării circuitelor integrate numerice, a echipamentelor şi sistemelor electronice complexe, scopul principal fiind găsirea unor metode mai performante de testare a acestora. Sumarul activităţii de doctorat este prezentat în figura 6.1.

Concluzia principală a tezei este că testarea JTAG se poate aplica cu succes în testarea la nivel de circuit integrat, echipament şi sistem electronic, existând posibilităţi de automatizare a acesteia fiind de aşteptat ca aceasta să fie tot mai des folosită.

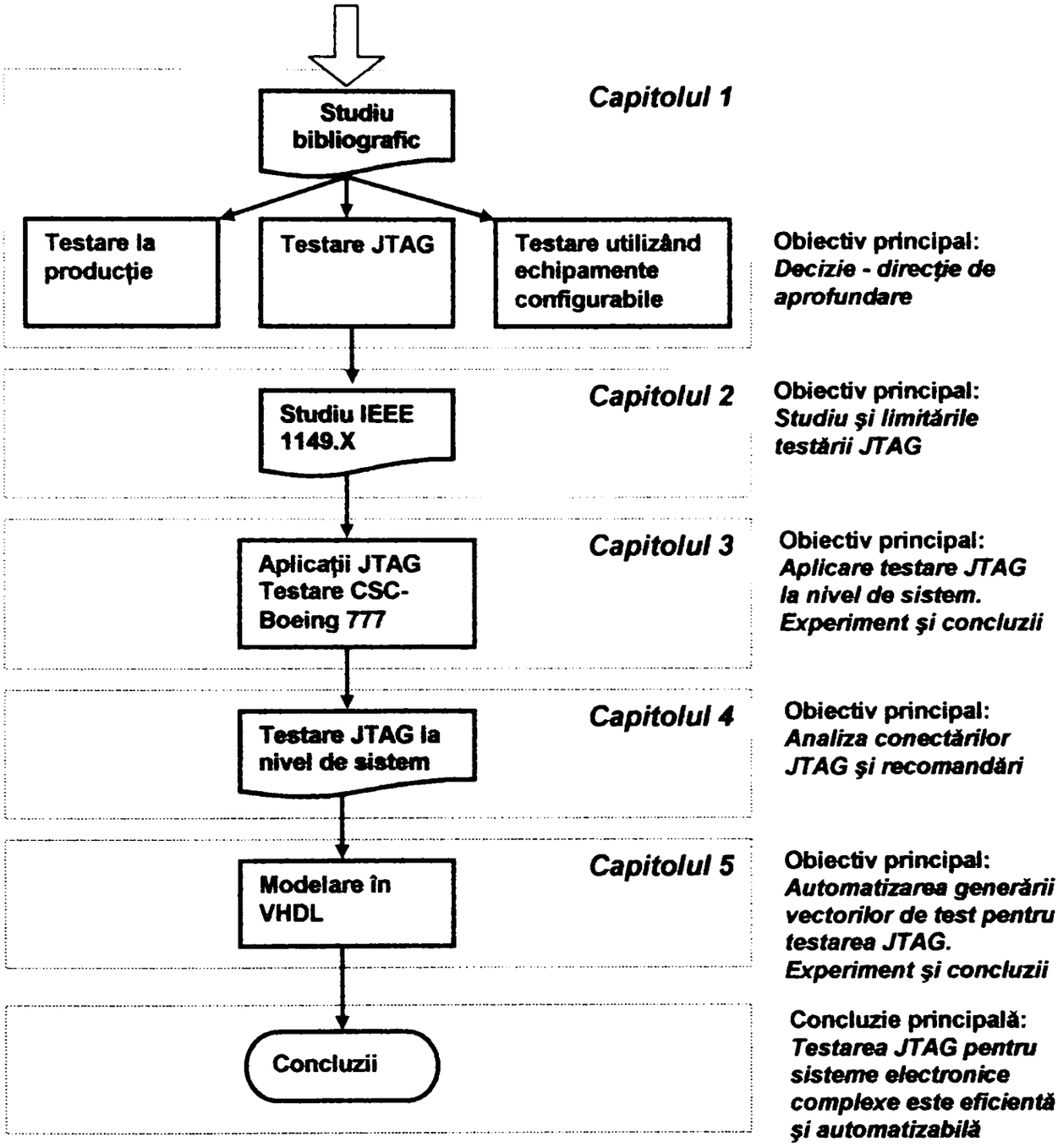


Figura 6.1. Sumar al activităţii de doctorat

Anexa A. Lista de termeni şi acronime

ASIC - *Application-Specific Integrated Circuit*, circuit integrat orientat pe aplicaţie
ASP - *Addressable Scan Port*, port adresabil de scanare
ATE - *Automatic Test Equipment*, echipament de testare automată
ATPG - *Automatic Test Pattern Generation*, generare automată a modelului
BIST - *Built-In Self-Test*, testare internă inclusă
B/S - *Boundary-Scan*, scanare pe frontieră
BSC - *Boundary-Scan Cell*, celulă de scanare pe frontieră
BSDL - *Boundary-Scan Description Language*, limbaj de descriere a scanării pe frontieră
BSR - *Boundary-Scan Register*, registru de scanare pe frontieră
BST - *Boundary-Scan Test*, testare prin scanare pe frontieră
CAE - *Computer-Aided Engineering*, inginerie asistată de calculator
CAN - *Controller Network Array*
CSC - *Cargo system controller*, unitate de control a încărcăturii
DFT - *Design-for-Test*, proiectare pentru testare
DR - *Data Register*, registru de date
DSP - *Digital Signal Processing/Processor*, procesor de semnal
EDA - *Electronic Design Automation*, proiectarea electronică automată
eTBC - *Embedded Test Bus Controller*, controllerul magistralei de test
ECU - *Electronic Control Unit*, unitate electronică de control
FPGA - *Field-Programmable Gate Array*, arie de porţi programabile
HASP - *Hierarchically Addressable Shadow Port*, port ierarhic adresabil de scanare
HSDL - *Hierarchical Scan Description Language*, limbaj de descriere a scanării ierarhice
ICE - *In-Circuit Emulation*, emulare în circuit
ICT - *In-Circuit Test*, testare în circuit
IEEE - *Institute of Electrical & Electronics Engineers*
IR - *Instruction Register*, registru de instrucţiune
ISP - *In-System Programming*, programare în circuit
JTAG - *Joint Test Action Group*, grup de firme producătoare de circuite integrate care au propus standardul IEEE 1149.1
PCB - *Printed Circuit Board*, cablaj imprimat
PLD - *Programmable Logic Device*, structură logică programabilă
PRPG - *Pseudo-Random Pattern Generation*, generare pseudo aleatoare de modele
PSA - *Parallel Signature Analysis*, analiză paralelă a semnăturii
SVF - *Serial Vector Format*, format de descriere a vectorilor de test
TAP - *Test Access Port*, portul de testare
TBC - *Test Bus Controller*, magistrala controllerului de testare
TCK - *Test Clock*, semnalul de tact de testare
TDI - *Test Data Input*, linie de date de intrare
TDO - *Test Data Output*, linie de date de ieşire
TMS - *Test Mode Select*, linie de selecţie
TRST - *Test Reset*, linie de reset
TUA - *Test Unit Addapter*, unitate de adaptare
UUT - *Unit Under Test*, unitate care va fi testată

Anexa B. Simularea bus-ului IEEE 1149.1 pe portul paralel

```
// @ Demian Petru, 1.10.1997 //
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

/* TDO = linia DO   */
/* TMS = linia D1   */
/* RST = linia D2
/* TCK = linia STROBE */
/* TDI = linia BUSY   */

unsigned int port=0x378, Dout=0x01;
FILE *pf,*sf;

//FUNCTIA GENEREAZA SEMNALUL DE TACT "TCK" CU FRECVENTA DE 500HZ
int tact()
 {
 int datai;
 outportb(port+2,0x00); /* front crescator */
 delay(1);
 outportb(port+2,0x01);
 delay(1);
 datai=(inportb(port+1)& 0x80)>>7; /* front descrescator */
 if (datai==1) datai =0;
   else datai =1;
 return datai;
 }

//FUNCTIA PERMITE RESETAREA CIRCUITELOR DE PE BUSUL DE TESTARE
//PE FRONTIERA PRIN PUNEREA LINIEI "TMS" PE "1" LOGIC PE DURATA
//A 5 IMPULSURI DE TACT
/* instructiunea RESET   */
void reset()
 {
 int data,i;
 data =inportb(port);
 outportb(port,data|0x02); /* se pune linia TMS pe 1 */
 for(i=0;i<5;i++)
   {
   tact();
   }
 }

//FUNCTIA TRIMITE IN REGISTRUL DE INSTRUCTIUNE AL CIRCUITULUI DE PE BUSUL
//IEEE 1149.1 CODUL INSTRUCTIUNII SPECIFICAT. ACESTA SE FACE PRIN COMANDA
//ADECVATA A LINIILOR "TCK", "TMS" SI "TDI" CONFORM ORDINOGRAMEI DIN FIG.2.3
 /* se trimite instructiunea "comanda" dupa care se trece in starea */
 /* RUN-TEST/IDLE                                                   */
void instruction(unsigned char comanda)
 {
 int data,i;
 outportb(port,0x0); /*TMS =1 */
 tact();             /*idle   */
 outportb(port,0x2);
 tact();             /* select DR-SCAN */
 outportb(port,0x2);
 tact();             /* select IR-SCAN */
```

```
 outportb(port,0x0);
 tact();              /* capture IR      */
 outportb(port,0x0);
 tact();              /* shift-IR        */
 for(i=0;i<5;i++)
   {
   if (((comanda >>i) &0x01)==1)
    outportb(port,0x01);
   else outportb(port,0x0);
   tact();
   }
/* TMS =1 pentru parasire     */
 if (((comanda >>5) &0x01)==1)
    outportb(port,0x3);
   else outportb(port,0x2);
   tact();           /* exit-ir          */
  outportb(port,0x2);
  tact();            /* update-ir        */
  outportb(port,0x0);
  tact();            /* run-test-idle    */
 }

//FUNCTIA INCARCA NUMARUL DE OCTETI SPECIFICAT "number" DIN ARIA
//"array" IN CIRCUITUL CONECTAT PE BUSUL IEEE 1149.1
void load(int number,int array[255])
{
 int data,i,datai,j, larray[255];
 outportb(port,0x0); /* TMS =1   */
 tact();                    /* idle     */
 outportb(port,0x2);
 tact();              /* select DR-SCAN */
 outportb(port,0x0);
 tact();              /* capture DR */
 outportb(port,0x0);
 tact();              /* shift-DR   */
 for(j=0;j<number;j++)
 {
 /* execute for every byte   */
  data =0;
   for(i=0;i<8;i++)
     {
     if (((j != number-1) | (i<7))==1)
      {
       if (((array[number-j-1] >>i) &0x01)==1)
      outportb(port,0x01);
      else outportb(port,0x0);
}
      else
      {
       if (((array[number-j-1] >>7) &0x01)==1)
      outportb(port,0x3);
       else outportb(port,0x2);
}
     datai=tact();
     data =(data | (datai <<i));
     larray[number-j-1] = data; /* intoarce octetul de date citit */
     }/* end for number    */
 } /* end for j   */
  outportb(port,0x2);
  tact();            /* update-ir   */
  outportb(port,0x0);
  tact();            /* run-test-idle */
```

```
  for(i=0;i<255;i++)
   array[i] = larray[i];
}


//PROGRAMUL PRINCIPAL PENTRU TRIMITEREA INSTRUCTIUNI (PRECIZATA PE LINIA
//DE COMANDA). ACEASTA FUNCTIE SE REDEFINESTE "MAIN" DACA SE DORESTE
//REALIZAREA UNUI EXECUTABIL CARE SA TRIMITA O INSTRUCTIUNE IN REGISTRUL DE
//INSTRUCTIUNE.
/* main for instrucion     */
void i_main(int argc,char *arg[])
{
int mcomanda;
char comanda[8];
if (argc==2)
    {
    mcomanda =0x3F;
    }
   else
   {
    strcpy(comanda,arg[1]);
    mcomanda=atoi(comanda);
   }
mcomanda = 0x09;
printf("IR =%x HEX\n",mcomanda);
instruction(mcomanda);
}

//FUNCTIE DE CONVERSIE
unsigned char conv(char car)
{
unsigned char rez;
/* car = car &0xdf;  */
 switch (car)
  {
   case '0': rez=0;
          break;
   case '1': rez=1;
          break;
   case '2': rez=2;
          break;
   case '3': rez=3;
          break;
   case '4': rez=4;
          break;
   case '5': rez=5;
          break;
   case '6': rez=6;
          break;
   case '7': rez=7;
          break;
   case '8': rez=8;
          break;
   case '9': rez=9;
          break;
   case 'A': rez=10;
          break;
   case 'B': rez=11;
          break;
   case 'C': rez=12;
          break;
   case 'D': rez=13;
```

```
          break;
   case 'E': rez=14;
          break;
   case 'F': rez=15;
          break;
   default:rez=0;
  }
return rez;
}

//FUNCTIA MAIN PENTRU "RESET". ACEASTA FUNCTIE SE REDENUMESTE "MAIN" DACA SE
//DORESTE REALIZAREA UNUI PROGRAM CARE FACE NUMAI "RESET" DE BUSUL B1149.1
void reset_main(void)
{
/* outportb(port,0x00);     */
outportb(port,0x01);  /* TDO */
/* outportb(port,0x02);  /* /* TMS */
/* outportb(port+2,0x01);
/* outportb(port+2,0x0);  /* /* TCK */
reset();
}



//FUNCTIA "MAIN" PENTRU "LOAD". FUNCTIA VA CITI FISIERUL SPECIFICAT PE LINIA
//DE COMANDA, IL TRIMITE PE BUSUL 1149.1 SI SALVEAZA INTR-UN FISIER RASPUNSUL
//CITIT PE LINIA "TDO"
void main(int argc,char *arg[])
{
int mcomanda,marray[255],i,j,nr_caracter;
char numefisieri[12],numefisiero[12],r;
unsigned char data1, data2;
if (argc< 2)
 {
  for(i=0;i<244;i++)
   {
    marray[i]=0x0;
   }
  marray[0] =0x1f;
  strcpy(numefisieri,"in");
  strcpy(numefisiero,"out");
 }
  else
  {
   strcpy(numefisieri,arg[1]);
   strcpy(numefisiero,arg[2]);
  }
 pf=fopen(numefisieri,"r");
 sf=fopen(numefisiero,"w");
 if (pf==0)
   {
   fprintf(stderr, "\n Warning - Unable to open file %s \n", numefisieri);
   }
 if (sf==0)
   {
   fprintf(stderr, "\n Warning - Unable to open file %s \n", numefisiero);
   }
  i=0;
  do
   {
   r=getc(pf);
   data2=conv(r)<<4;
```

```
   r=getc(pf);
   data1=conv(r);
   if (r!=EOF)
      {
        marray[i]=data1 |data2;     /* sa format un octet  */
      }
    i++;
   }
  while((r!=EOF)&&(i<5000)); /* nr maxim de caractere 500000 */
  fclose(pf);
  nr_caracter =i-1;
  load(nr_caracter,marray);
  printf("Nr. of byte = %d \n",nr_caracter);
  for(i=0;i<nr_caracter;i++)
    {
      data1 = marray[i] &0x0F; /* LOW  */
      data2 = (marray[i] &0xF0)>>4; /* HIGH */
    fprintf(sf, "%x",data2);
    fprintf(sf, "%x",data1);
    }
 fclose(sf);
}
```

Anexa C. Codul sursă pentru driverul VX 4491

```
*******************************************************************************
** TIMTEH S.R.L.                                                             **
** DIGITAL B1149 DINAMIC DESCRIPTION                                         **
** Version: 13.03.1998                                                       **
** Author: Demian Petru                                                      **
*******************************************************************************
*=============================================================================*
*                                                                             *
*                  DeviceDB DYNAMIC DESCRIPTION                               *
*                                                                             *
*=============================================================================*
*****   Built-In Function Definitions - Short names   *****

def, fnc, Cat   ==      1;      ** ConcatenateStrings
def, fnc, Cmp   ==      2;      ** Compare Strings
def, fnc, Cpy   ==      3;      ** CopyStrings
def, fnc, Ilc   ==      4;      ** InsertLeadingCharacters
def, fnc, Itc   ==      5;      ** InsertTrailingCharacters
def, fnc, Itoc  ==      6;      ** IntegerToNumericString
def, fnc, Tup   ==      7;      ** LowerCaseToUpperCase
def, fnc, Slw   ==      8;      ** RemoveLeadingWhiteSpace
def, fnc, Stw   ==      9;      ** RemoveTrailingWhiteSpace
def, fnc, Rev   ==      10;     ** ReverseString
def, fnc, Sfw   ==      11;     ** SetCurrentFieldWidth
def, fnc, Tlo   ==      12;     ** UpperCaseToLowerCase
def, fnc, Bic   ==      13;     ** BusInterfaceClear
def, fnc, Bto   ==      14;     ** SetBusTimeOut
def, fnc, Wcmd  ==      15;     ** WriteCommandString
def, fnc, Wdat  ==      16;     ** WriteDataString
def, fnc, Rtn   ==      17;     ** GetCurrentFunctionReturn
def, fnc, Print ==      18;     ** Print
def, fnc, Abort ==      19;     ** Abort
def, fnc, Stop  ==      20;     ** Stop
def, fnc, Apnd  ==      21;     ** AppendString
def, fnc, Split ==      22;     ** ParseCharacters
def, fnc, Gnum  ==      23;     ** ParseDecimalString
def, fnc, Gint  ==      24;     ** ParseNumericString
def, fnc, GCla  ==      25;     ** GetBusControllerListenAddress
def, fnc, GCta  ==      26;     ** GetBusControllerTalkAddress
def, fnc, GDla  ==      27;     ** GetBusDeviceListenAddress
def, fnc, GDid  ==      28;     ** GetBusDeviceName
def, fnc, GDta  ==      29;     ** GetBusDevciceTalkAddress
def, fnc, Sdev  ==      30;     ** SetBusDeviceNumber
def, fnc, Seoi  ==      31;     ** SetEoiLineFlag
def, fnc, Talk  ==      32;     ** TalkToBusDevice
def, fnc, Ctof  ==      33;     ** DecimalStringToFloat
def, fnc, Ftoc  ==      34;     ** FloatToDecimalString
def, fnc, Ctoi  ==      35;     ** NumericStringToInteger
def, fnc, Sdp   ==      36;     ** SetDecimalPointFlag
def, fnc, Sexp  ==      37;     ** SetExponentIndicator
def, fnc, SexpS ==      38;     ** SetExponentSignFlag
def, fnc, SexpW ==      39;     ** SetExponentWidth
def, fnc, SfW   ==      40;     ** SetFractionWidth
def, fnc, SiW   ==      41;     ** SetIntegerWidth
def, fnc, SmanS ==      42;     ** SetNumberSignFlag
def, fnc, Listn ==      43;     ** ListenToBusDevice
def, fnc, Read  ==      44;     ** ReadDataString
def, fnc, Dsp   ==      45;     ** Display
def, fnc, GpK   ==      46;     ** GetPathCount
def, fnc, GpD   ==      47;     ** GetPathData
def, fnc, GiK   ==      48;     ** IndirectPathCount
def, fnc, GiD   ==      49;     ** IndirectPathToDirect
def, fnc, RBin  ==      50;     ** ReadBinary
def, fnc, WBin  ==      51;     ** WriteBinary
def, fnc, XtoD  ==      52;     ** HexToDigital
```

```
def, fnc, DtoX  ==      53;     ** DigitalToHex
def, fnc, FmtDtoS   ==  54;     ** FormatDataToString           (NEW)
def, fnc, FmtIBPut  ==  55;     ** FormattedBufferedBusOutput   (NEW)
def, fnc, CclrF ==      56;     ** ClearBusClearFlag
def, fnc, CkSum ==      57;     ** ClearGetCheckSum
*****
def, fnc, GetClockParameter      == 62;  ** Get Clock Parameter
def, fnc, RemoveClock            == 61;  ** Reset Clock
*******************************************************************************


*****   Built-In Function Definitions - Long names   *****

def, fnc, ConcatenateStrings            ==  1;
def, fnc, CompareStrings                ==  2;
def, fnc, CopyString                    ==  3;
def, fnc, InsertLeadingCharacters       ==  4;
def, fnc, InsertTrailingCharacters      ==  5;
def, fnc, IntegerToNumericString        ==  6;
def, fnc, LowerCaseToUpperCase          ==  7;
def, fnc, RemoveLeadingWhiteSpace       ==  8;
def, fnc, RemoveTrailingWhiteSpace      ==  9;
def, fnc, ReverseString                 == 10;
def, fnc, SetFieldWidth                 == 11;
def, fnc, UpperCaseToLowerCase          == 12;
def, fnc, InterfaceClear                == 13;
def, fnc, SetBusDeviceTimeout           == 14;
def, fnc, WriteCommandString            == 15;
def, fnc, WriteDataString               == 16;
def, fnc, GetFunctionReturn             == 17;
def, fnc, AppendString                  == 21;
def, fnc, ParseCharacters               == 22;
def, fnc, ParseDecimalString            == 23;
def, fnc, ParseNumericString            == 24;
def, fnc, GetBusControllerListenAddress == 25;
def, fnc, GetBusControllerTalkAddress   == 26;
def, fnc, GetBusDeviceListenAddress     == 27;
def, fnc, GetBusDeviceName              == 28;
def, fnc, GetBusDeviceTalkAddress       == 29;
def, fnc, SetBusDeviceNumber            == 30;
def, fnc, SetEoiLineFlag                == 31;
def, fnc, TalkToBusDevice               == 32;
def, fnc, DecimalStringToFloat          == 33;
def, fnc, FloatToDecimalString          == 34;
def, fnc, NumericStringToInteger        == 35;
def, fnc, SetDecimalPointFlag           == 36;
def, fnc, SetExponentIndicator          == 37;
def, fnc, SetExponentSignFlag           == 38;
def, fnc, SetExponentWidth              == 39;
def, fnc, SetFractionWidth              == 40;
def, fnc, SetIntegerWidth               == 41;
def, fnc, SetNumberSignFlag             == 42;
def, fnc, ListenToBusDevice             == 43;
def, fnc, ReadDataString                == 44;
def, fnc, Display                       == 45;
def, fnc, GetPathCount                  == 46;
def, fnc, GetPathData                   == 47;
def, fnc, IndirectPathCount             == 48;
def, fnc, IndirectPathToDirect          == 49;
def, fnc, ReadBinary                    == 50;
def, fnc, WriteBinary                   == 51;
def, fnc, HexStringToDigital            == 52;
def, fnc, DigitalToHexString            == 53;
def, fnc, EncodeString                  == 54;
def, fnc, EncodeBuf                     == 54;
def, fnc, FTalk                         == 55;
def, fnc, SetClearFlag                  == 56;
def, fnc, SetGetChecksum                == 57;
def, fnc, CheckSRQLine                  == 58;
```

```
def, fnc, GetToken                          == 59;
def, fnc, WriteToGPIO                       == 91;
def, fnc, ReadFromGPIO                      == 92;
def, fnc, TalkVXI                           == 93;
def, fnc, ListenVXI                         == 94;
def, fnc, ReadSTB                           == 95;
def, fnc, OpenSerial                        == 101;
def, fnc, CloseSerial                       == 102;
def, fnc, WriteAsciiSerial                  == 103;
def, fnc, ReadAsciiSerial                   == 104;
def, fnc, SetTimeoutSerial                  == 105;
*
*
def, fnc, StartClock                        == 61;        ** Start Clock
def, fnc, GetClock                          == 62;        ** Get Clock Parameter
def, fnc, StopClock                         == 61;        ** Reset Clock
*
def, fnc, WritePXB                  == 110;               ** Added for RADA 608
def, fnc, InitPXB                   == 111;               ** Added for RADA 608
*
def, fnc, ReadPort                  == 112;               ** direct IO control
def, fnc, WritePort                 == 113;               ** direct IO control
def, fnc, InitIO                    == 114;               ** initialize direct IO
*
* WriteRS485(int|dig iobase, int|dig[] buf, int count)
* ReadRS485(int|dig iobase, int|dig[] buf, int count, dec maxtime)
*      returns # of read bytes
* WriteReadRS485(int|dig iobase, int|dig[] buf, int tx_count, int rx_count, dec maxtime)
*      returns # of read bytes
*
def, fnc, OpenRS485                 == 115;               ** open RS485
def, fnc, WriteRS485                        == 116;               ** write RS485
def, fnc, ReadRS485                 == 117;               ** read RS485
def, fnc, WriteReadRS485            == 119;               ** write & read RS485
*
def, fnc, InitEthNet                        == 121;               ** initialize Ethernet
def, fnc, OpenEthNet                        == 122;               ** open Ethernet
def, fnc, WriteAsciiEthNet          == 123;               ** write Ethernet
def, fnc, ReadAsciiEthNet           == 124;               ** read Ethernet
def, fnc, CloseEthNet                       == 125;               ** close Ethernet
def, fnc, GetMsgEthNet                      == 126;               ** get message Ethernet
def, fnc, SetTimeoutEthNet          == 127;               ** set timeout Ethernet
*
*******************************************************************************
dcl {
     int debug;
     int simulation;
     int dual;
    }

*******************************************************************************
*                                  DUMMY MACRO                                *
*******************************************************************************

def, mac, DummyMacro( );
{
;
}



*******************************************************************************
*                MANUAL INTERVENTION DYNAMIC DESCRIPTION                      *
*******************************************************************************

dcl
{
    int MI_MaxTime;
}
```

```
def, mac, MI_InxEvent( T);
dec T;  * MAX-TIME
{
      MI_MaxTime = T + 0.125;
      'manual-intervention' = FALSE;
}

def, mac, MI_FthEvent( @R);
bln R;  * EVENT-INDICATOR
{
      while(( 'manual-intervention' == FALSE) && ( MI_MaxTime > 0))
      {
         MI_MaxTime = MI_MaxTime - 0.25;
         SETDELAY(0.25);
      }
      R = 'manual-intervention;
}

*****************************************************************************
** TIMTEH S.R.L.                                                           **
** DIGITAL B1149 DINAMIC DESCRIPTION                                       **
** Version: 13.03.1998                                                     **
** Author: Demian Petru                                                    **
*****************************************************************************
dcl
{
int B1149_stim;
   }

* Display GPIB-bus error message
def, mac, B1149_BusDeviceError(B1149_error-msg);
txt B1149_error-msg 64, B1149_device 5;
{
GetBusDeviceName(B1149_device);
Display("\n");
Display(B1149_device,"--",B1149_error-msg,"\n");
}


*  Return respunses for B1149  (text)
def, mac, B1149_ListenBus(@B1149_buffer);
txt B1149_buffer 256;
int B1149_temp, B1149_itemp;
{
*      SetBusTimeOut(5);
       ListenToBusDevice(B1149_buffer);
       GetFunctionReturn(B1149_itemp);
       if(B1149_itemp < 1)
              B1149_BusDeviceError("BUS LISTEN ERROR");
       else
            {
            ;
*         B1149_char-read = B1149_itemp;
            }
}


def, mac, B1149_TalkBus(B1149_buffer);
txt B1149_buffer 256;
txt B1149_device 10;
int B1149_itemp;
{
       GetBusDeviceName(B1149_device);
       AppendString("\r\n",B1149_buffer);
*      Display("msg=",B1149_buffer);
        SetEoiLineFlag();
       TalkToBusDevice(B1149_buffer);
       GetFunctionReturn(B1149_itemp);
```

```
      if(B1149_itemp<0){
      B1149_BusDeviceError("BUS TALK ERROR ON DIG_ARINC429");
      }
}

* Transmit binary command to B1149
def, mac, B1149B_TalkBus(B1149_buffer);
txt B1149_buffer 256;
txt B1149_device 10;
int B1149_itemp;
{
        WriteCommandString( B1149_buffer);
        GetFunctionReturn( B1149_itemp);
        if( B1149_itemp < 0)
           B1149_BusDeviceError( "GPIB Command error");
        CopyString( "", B1149_buffer);
}



* Initialize adress of port and frequency of clock signal for PC16550D
def, mac, B1149_INITIALIZE();
{
;
}

* Initialize B1149
def, mac, B1149_RESET();
{
SETDEVICE(<B1149>);
B1149_TalkBus("*rst");
B1149_TalkBus("BSN:RESETTAP");
Display("RESET \n");
*B1149_TalkBus("OPC?");
}

* SetupB1149 for transmit
def, mac, SetupB1149-T(B1149_fnc, B1149_boud_rate,B1149_max_time);
int B1149_fnc,B1149_canal, B1149_boud_temp_H, B1149_boud_temp_L, B1149_parity;
dec B1149_max_time;
dig B1149_temp 16, B1149_temp_8 8;
dec B1149_boud_rate, B1149_boud_temp;

{
;
*D485T_max_time = D485_max_time; *Maxim time for transmision
B1149_RESET();
}

* SetupB1149 for recept
def, mac, SetupB1149-R(B1149_fnc, B1149_boud_rate,B1149_max_time);
int B1149_fnc,B1149_canal,B1149_parity;
dig B1149_temp 16, B1149_temp_8 8;
dec B1149_boud_rate, B1149_boud_temp, B1149_boud_temp_H, B1149_boud_temp_L;
dec B1149_max_time;
txt B1149_adr 3, B1149_buffer 256;
{
*  B1149_RESET();
;
}


* Transmit digital "B1149_data"
def, mac, InitB1149-T(B1149_fnc, @ B1149_data, B1149_stim);
int B1149_fnc , B1149_canal, B1149_stim, B1149_I;
dig B1149_data(1000):8,B1149_d(1000):8, B1149_data_R 8, B1149_temp 16;
txt B1149_adr 3,B1149_buffer 255;
{
   SETDEVICE(<B1149>);
```

```
* B1149_stim = B1149_stim ;
* Display("Data =",B1149_buffer,"\n");
* Display("Stim =",B1149_stim,"\n");
**B1149_TalkBus("*rst");

*B1149_TalkBus("BSN:LOGMODE LOGFAILTDO");
B1149_TalkBus("BSN:LOGMODE LOGALLTDO");
B1149_TalkBus("HEADER OFF");
B1149_TalkBus("CLOCK:FREQ 100000");
B1149_TalkBus("CLOCK:FREQ?");
B1149_ListenBus(B1149_buffer);
B1149_TalkBus("ID?");
B1149_ListenBus(B1149_buffer);
* Display("Identification string =",B1149_buffer,"\n");
**B1149_TalkBus("*rst");
**B1149_TalkBus("BSN:RESETTAP OFF");
B1149_TalkBus("NEW");
B1149_TalkBus("TESTNAME \"TEST_OK\"");
B1149B_TalkBus("_?");     * UNT, UNL
GetBusControllerTalkAddress( B1149_adr);      ** GetBusControllerTalkAddress
CopyString(  B1149_adr, B1149_buffer);
GetBusDeviceListenAddress( B1149_adr);    ** GetBusDeviceListenAddress
ConcatenateStrings(B1149_buffer,B1149_adr,B1149_buffer);
B1149B_TalkBus(B1149_buffer);
for(B1149_I = 1; B1149_I <= B1149_stim ;B1149_I= B1149_I +1)
  {
*  WriteBinary(B1149_data(B1149_I), 1, 1);
 B1149_d(B1149_I) = B1149_data(B1149_I);
* WriteBinary(B1149_data, 220, 1);
*  Display("D( ",B1149_I,")=",B1149_data(B1149_I),"\n");
  }
B1149_d(B1149_stim+1) = X'0D' ;
B1149_d(B1149_stim+2) = X'0A' ;
B1149_stim = B1149_stim +2;
WriteBinary(B1149_d, B1149_stim, 1);
B1149B_TalkBus("_?");     * UNT, UNL
*GetBusControllerTalkAddress( B1149_adr);      ** GetBusControllerTalkAddress
*CopyString(  B1149_adr, B1149_buffer);
*GetBusDeviceListenAddress( B1149_adr);    ** GetBusDeviceListenAddress
*ConcatenateStrings(B1149_buffer,B1149_adr,B1149_buffer);
*B1149B_TalkBus(B1149_buffer);
*WriteBinary(B1149_d, 292, 1);
B1149B_TalkBus("_?");     * UNT, UNL

B1149_TalkBus("TESTNAME?");
B1149_ListenBus(B1149_buffer);
Display("LOAD =",B1149_buffer,"\n");

B1149_TalkBus("START");
SETDELAY(1);
B1149_TalkBus("STOP");
*B1149_ListenBus(B1149_buffer);
*Display("Load = ",B1149_buffer,"\n");
}


def, mac, InitB1149-R(B1149_fnc, B1149_data, B1149_stim);
int B1149_fnc, B1149_i;
dig B1149_data 8, B1149_stim 8;
{
* Display("B1149_stimR_init",B1149_stim,"\n");
*  Display("Data = ", B1149_data , "\n");
   SETDEVICE(<B1149>);
}



* receive data
def, mac, FetchB1149-R(B1149_fnc, @B1149_data,@B1149_stim);
```

```
int B1149_fnc, B1149_canal, B1149_i,B1149_numberbytes;
int B1149_stim, B1149_I ; * number of word
int B1149_ofset;
int B1149_data_temp;
dig B1149_datai(244):9, B1149_data_R 8, B1149_binary(1024):8, B1149_data(122):8;
txt B1149_buffer 128, B1149_tbuffer 128,B1149_ttbuffer 128, B1149_adr 3;
dig B1149_d1:9,B1149_d2:9,B1149_d3:9,B1149_d4:9;
int B1149_temp;
{
   SETDEVICE(<B1149>);
* B1149_TalkBus("BSN:LOGALLTDO");
 B1149_TalkBus("BSN:LOGMODE?");
 B1149_ListenBus(B1149_buffer);
 B1149_TalkBus("RESULTSIZE?");
 B1149_ListenBus(B1149_ttbuffer);
 ParseNumericString(B1149_ttbuffer,B1149_buffer,B1149_tbuffer);
 NumericStringToInteger(B1149_buffer,B1149_numberbytes); * convert to integer
* B1149_numberbytes = B1149_numberbytes +3;
* Display("Number of bytes=",B1149_numberbytes,"\n");
* B1149_numberbytes = 323;
 if (B1149_numberbytes != 0 )
  {
   B1149_TalkBus("RESULTS?");
   SetEoiLineFlag();
*   B1149_ListenBus(B1149_buffer);   ****************
   B1149B_TalkBus("_?");    * UNT, UNL
   B1149B_TalkBus(">Ie");
   GetBusControllerListenAddress( B1149_adr);     ** GetBusControllerListenAddress
   CopyString(  B1149_adr, B1149_buffer);
   GetBusDeviceTalkAddress( B1149_adr);   ** GetBusDeviceTalkAddress
   ConcatenateStrings(B1149_buffer,B1149_adr,B1149_buffer);
*   B1149_TalkBus(B1149_buffer);
   ReadBinary(B1149_binary , B1149_numberbytes, 1);
*   B1149_ListenBus(B1149_buffer);
  B1149_ofset = 267;
  for(B1149_I=1; B1149_I <123; B1149_I= B1149_I+ 1)
     {
      B1149_temp = B1149_I + B1149_ofset;
      B1149_datai(B1149_I) = B1149_binary(B1149_temp) ;
      Display("B1149_datai(",B1149_I,")=",B1149_datai(B1149_I),"\n");
     }
  B1149B_TalkBus("_?");
  B1149_ListenBus(B1149_buffer);
  B1149_TalkBus("BSN:RESETTAP");
  for(B1149_I=1; B1149_I <122; B1149_I = B1149_I +2 )
    {
     B1149_d1 = (B1149_datai(B1149_I+1) & X'0F') << 4;
     B1149_d2 = (B1149_datai(B1149_I+1) & X'F0') >> 4;
     B1149_d3 = (B1149_datai(B1149_I) & X'0F') << 4;
     B1149_d4 = (B1149_datai(B1149_I) & X'F0') >> 4;
 *   Display("B1149_di1(",B1149_I,")=",B1149_d1,"\n");
 *   Display("B1149_di2(",B1149_I,")=",B1149_d2,"\n");
 *   Display("B1149_di3(",B1149_I,")=",B1149_d3,"\n");
 *   Display("B1149_di4(",B1149_I,")=",B1149_d4,"\n");
     B1149_data(B1149_I) = B1149_d1 | B1149_d2;
     B1149_data(B1149_I + 1) = B1149_d3 | B1149_d4;
 *    Display("B1149_data(",B1149_I,")=",B1149_data(B1149_I),"\n");
 *    Display("B1149_data(",B1149_I,")=",B1149_data(B1149_I+1),"\n");
    }
   }
  B1149_stim =122;
}



*** CSC DYNAMIC DESCRIPTION ***************************************************
*******************************************************************************
** Reserved macro **
def, mac, RESETALL();
```

```
{
*** CSC RESET ALL ***************************************************************
}


def, mac, INITIALIZE();
{
*** CSC INITIALIZE***************************************************************
}


def, mac, TERMINATE();
{
*** CSC TERMINATE ***************************************************************
RESETALL();
}



*=============================================================================*
*                                                                             *
*                           STATIC DESCRIPTION                                *
*                                                                             *
*=============================================================================*


*******************************************************************************
*                  MANUAL INTERVENTION STATIC DESCRIPTION                     *
*******************************************************************************
begin DEV MI;

   begin;
      cnx event-out NUL-EV;

      init MI_InxEvent( max-time: 0);
      fetch MI_FthEvent( event-indicator result);

      event monitor (manual-intervention) event;

        control
        {
           event-out;
           event-indicator;
           max-time max 3000 sec;
        }
    end;
end;  ** DEV MI


*** SWITCH STATIC DESCRIPTION ************************************************
*******************************************************************************
** TIMTEH S.R.L.                                                             **
** DIGITAL B1149 STATIC  DESCRIPTION                                         **
** Version: 13.03.1998                                                       **
** Author: Demian Petru                                                      **
*******************************************************************************
begin dev B1149;
begin FNC=1 ;
  source logic signal;
  reset B1149_RESET();
  control {
  voltage  range -15 v to 15 v;
     }
   cnx hi TDO-1149 , lo GND-1149;
 end;

begin FNC=2 ;
  sensor (value) logic signal;
  reset B1149_RESET();
```

```
  control {
  voltage  range -15 v to 15 v;
  value range 0 to 2000;
     }
   cnx hi TDI-1149, lo GND-1149;
 end;

begin FNC=3;
 function digital configuration;
 setup B1149_INITIALIZE();
 reset B1149_RESET();
 control
  {
  voltage range -15 v to 15 v continuous;
  value range 0 to 2000;
  }
   cnx hi TDO-1149, lo GND-1149;
 end;

begin FNC=4;
 function digital configuration;
 setup B1149_INITIALIZE();
 reset B1149_RESET();
 control
  {
  voltage range -15 v to 15 v continuous;
  value range 0 to 1600;
  }
   cnx hi TDI-1149, lo GND-1149;
 end;

* Static part for transmision
begin FNC=5 using B1149;
  function logic signal;
  reset B1149_RESET();
  setup SetupB1149-T($FNC, bit-rate: 10000, max-time:1);
  init InitB1149-T($FNC, (dig) stim:b'0',stim count:1);
  control
   {
    stim range 0 to 65536;
    level-logic-one voltage range 0.0 v to 15 v errlmt +-0.01v;
    level-logic-zero voltage range -15 v to 0.0 v errlmt +-0.01v;
    level-logic-quies voltage range -15 v to 15 v;
    serial-msb-first;
    word-length range 1 bits to 244 bits;
    pulse-class rz;
    voltage range -15 v to 15 v continuous ;
    bit-rate max 10000000 bits/sec;
    max-time range 0.001 sec to 10 sec;
    }
   cnx hi TDO-1149, lo GND-1149;
 end;


* Static part for reception
begin FNC=6 using B1149;
  function -value logic signal;
  setup SetupB1149-R($FNC, bit-rate: 10000,max-time:1);
*  reset B1149_RESET();
*  init InitB1149-R($FNC, stim:b'0',stim count:1);
  fetch FetchB1149-R($FNC, (dig) value result:b'0', value result count:0);
  control
   {
    stim range 0 to 65536;
    ref range 0 to 65536;
    level-logic-one voltage range 0.0 v to 10.65 v errlmt +-0.01v;
    level-logic-zero voltage range -10.65 v to 0.0 v errlmt +-0.01v;
    level-logic-quies voltage range -10.65 v to 10.65 v;
    serial-msb-first;
```

```
    word-length 244 bits;
    mask-one range 1 to 1024 ;
    fault-count range 1 to 1024 ;
    pulse-class rz;
    voltage range -15 v to 15 v continuous ;
    bit-rate max 10000000 bits/sec;
    value range 0 to 2000;
    max-time range 0.001 sec to 10 sec;
    }
   cnx hi TDI-1149, lo GND-1149;
 end;
end;
*******************************************************************************
***** END B1149 ***************************************************************
*******************************************************************************
end;
```

Anexa D. Exemplu de program scris in Atlas

```
000001 BEGIN, ATLAS PROGRAM 'EXEMPLU'                                    $
C                                                                        $
C SCOPUL ACESTUI PROGRAM ESTE DE A REALIZA O TESTARE FUNCIONALA LA        $
C O UNITITATE IPOTETICA                                                  $
C @Demian Petru, Feb. 1999                                               $
C ************************************************************************ $
C *                    REQUIRE STATEMENTS                              * $
C ************************************************************************ $
C                                                                        $
C SE SPECIFICA CA SE VA UTILIZA O SURSA DE TENSIUNE ALTERNATIVA           $
  000010 REQUIRE, 'AC PSU RES', SOURCE, AC SIGNAL,
          CONTROL,
             VOLTAGE RANGE 0V TO 250 V CONTINUOUS ERRLMT +- 3 V,
             THRU-RES 10 KOHM ERRLMT +- 5 PC,
          CNX HI LO $
C                                                                        $
C SE SPECIFICA CA SE VA UTILIZA O SURSA DE TENSIUNE CONTINUA              $
  000020 REQUIRE, '28VDC PSU', SOURCE, DC SIGNAL,
          CONTROL,
            VOLTAGE RANGE 0 V TO 50 V BY 1 V ERRLMT +- 0.1 V,
          CAPABILITY,
            CURRENT MAX 1 A,
          CNX HI LO $
C                                                                        $
C SE SPECIFICA O CONEXIUNE (SCURT) INTRE DOI PINI                        $
 000100 REQUIRE, 'CONN', LOAD, SHORT, CNX HI LO $
C                                                                        $
C SE SPECIFICA UN PIN LEGAT LA MASA                                      $
 000110 REQUIRE, 'GROUND', LOAD, EARTH, CNX $
C                                                                        $
C SE SPECIFICA CA SE VOR UTILIZA REZISTENTE                              $
 000120 REQUIRE, 'RES LOAD', 'RES LOAD-1',
           'RES LOAD-2', 'RES LOAD-3',
          CONTROL,
            RES RANGE 1 OHM TO 300 OHM BY 1 OHM ERRLMT +- 5 PC,
          CNX HI LO $
C                                                                        $
C SE SPECIFICA O REZISTENTA PROGRAMABILA                                 $
```

```
 000130 REQUIRE, 'CURRENT-LOAD', LOAD, IMPEDANCE,
          CONTROL,
            RES RANGE 0 OHM TO 5.6 OHM CONTINUOUS,
            CURRENT-LMT RANGE 0 A TO 5 A CONTINUOUS,
          CNX HI LO $
C                                                                        $
C SE SPECIFICA MULTIMETRUL PENTRU MASURAREA REZISTENTELOR                 $
 000140 REQUIRE, 'OHMMETER', SENSOR (RES), IMPEDANCE,
          CAPABILITY,
            RES RANGE 0.0 OHM TO 11000000 OHM BY .1 OHM ERRLMT +-1 PC,
          LIMIT,
            PWR-LMT MAX 0.25 W,
          CNX HI LO $
C                                                                        $
C SE SPECIFICA MULTIMETRUL PENTRU MASURAREA TENSIUNILOR ALTERNATIVE      $
  000150 REQUIRE, 'AC VOLTMETER', SENSOR (VOLTAGE), AC SIGNAL,
          CAPABILITY,
            VOLTAGE RANGE 0.0 V TO 250 V BY .1 V ERRLMT +-1 PC,
          CNX HI LO $
C                                                                        $
C SE SPECIFICA MULTIMETRUL UTILIZAT PENTRU MASURAREA TENSIUNI DE VIRF$
 000160 REQUIRE, 'AC VOLTMETER-PP', SENSOR (VOLTAGE-PP),
          AC SIGNAL,
          CAPABILITY,
            VOLTAGE-PP RANGE 0.0 V TO 500 V ERRLMT +-.1 V,
          CNX HI LO $
C                                                                        $
C SE SPECIFICA MULTIMETRUL PENTRU MASURAREA TENSIUNILOR ALTERNATIVE      $
 000170 REQUIRE, 'DC-VOLTMETER', SENSOR (VOLTAGE), DC SIGNAL,
          CAPABILITY,
            VOLTAGE RANGE -20 V TO 40.0 V ERRLMT +- 1 V,
          CNX HI LO $
C                                                                        $
C SE SPECIFICA MULTIMETRUL UTILIZAT LA MASURAREA TENSIUNILOR CONTINUE$
 000180 REQUIRE, 'DMM', SENSOR (VOLTAGE), DC SIGNAL,
          CAPABILITY,
            VOLTAGE RANGE -120 V TO 120 V ERRLMT +- 0.1 V,
          CNX HI LO $
C                                                                        $
C SE SPECIFICA MULTIMETRU CA AMPERMETRU                                  $
 000190 REQUIRE, 'DC-AMMETER', SENSOR (CURRENT), DC SIGNAL,
```

```
          CAPABILITY,
            CURRENT RANGE 0.0 A TO 1.0 A ERRLMT +- 1 PC,
          CNX VIA HI LO $
C                                                                          $
C ************************************************************************ $
C *                     DECLARE STATEMENTS                               * $
C ************************************************************************ $
C                      ***** VARIABLE *****                                $
 000500 DECLARE, VARIABLE, 'VOLT1', 'CURRENT-VAL' IS DECIMAL;
          'EXPECTED', 'STIMUL','OUTPUT-FILE'
           IS FILE OF STRING(80) OF CHAR $
C                                                                          $
C ************************************************************************ $
C *                   DEFINE DIGITAL CONFIGURATION                       * $
C ************************************************************************ $
C$
C ***  DEFINE CONFIGURATION STRUCTURE FOR BOUNDARY SCAN          ***  $
 000510 DEFINE, 'BSCAN', DIGITAL CONFIGURATION $
     20   DEFINE, 'BSCAN-DRIVER', DIGITAL SOURCE,
            DC SIGNAL USING 'BSCAN-STIM',
            VOLTAGE RANGE 0 V TO 5 V,
            CURRENT RANGE 5 MA TO 10 MA,
            LEVEL-LOGIC-ONE VOLTAGE 2.0 V,
            LEVEL-LOGIC-ZERO VOLTAGE 0.4 V,
            CNX HI 'TDI'
                LO EARTH $
     30   DEFINE, 'BSCAN-RECV', DIGITAL SENSOR, (VALUE),
            VOLTAGE RANGE 0 V TO 5 V,
            CURRENT RANGE 5 MA TO 10 MA,
            LEVEL-LOGIC-ZERO UL 0.8 V LL 0.4 V,
            LEVEL-LOGIC-ONE UL 5 V LL 2.0 V,
            CNX HI J1-DD1
                LO EARTH $
 000540 END, 'BSCAN' $
C$
C ************************************************************************ $
C *              DEFINE PROCEDURES                                       * $
C IN CONTINUARE DE OBICEI SE INTRODUC PROCEDURILE CARE SUNT APELATE DIN$
C CADRUL PROGRAMULUI PRINCIPAL                                             $

C ************************************************************************ $
```

```
C * PROCEDURA 'VERIFICARE-TUA'                                          * $
C *-------------------------------------------------------------------* $
C SCOP: DE OBICEI SE UTILIZEAZA O PROCEDURA PENTRU A VERIFICA CA       * $
C UNITATEA DE ADAPTARE (TUA) ESTE CEA CORESPUNZATOARE                  * $
C * RESOURCES USED: 'OHMMETER'                                         * $
C * PROCEDURES USED: NONE                                             * $
C ********************************************************************* $
050000 DEFINE, 'VERIFICARE-TUA', PROCEDURE
                ('HI-PIN', 'LO-PIN' IS 'ALLPINS',
                 'RES-VAL' IS DECIMAL) $
C                                                                       $
     10 VERIFY, (RES), IMPEDANCE USING 'OHMMETER',
          GT 'RES-VAL' OHM,
          RES MAX 10000000 OHM,
          CNX HI 'HI-PIN'
              LO 'LO-PIN' $
C$
 050999 END, 'VERIFARE-TUA' $
C$
C ********************************************************************* $
C * PROCEDURA 'VERIFICARE-TENSIUNE'                                    * $
C * RESOURCES USED: '28VDC PSU'                                        * $
C *                 'DC VOLTMETER'                                     * $
C ********************************************************************* $
 051000 DEFINE, 'VERIFICARE-TENSIUNE', PROCEDURE
                ('VOLTAGE' IS DECIMAL) $
C$
     10 CHANGE, DC SIGNAL USING '28VDC PSU',
          VOLTAGE 'VOLTAGE' RANGE 0V TO 5 V ERRLMT +- 0.1 V,
          CURRENT MAX 1 A,
          CNX HI J1-F21
              LO EARTH $
     20 WAIT FOR, TIME 300 MSEC $
     30 VERIFY, (VOLTAGE ERRLMT +- 0.25 V),
          DC SIGNAL USING 'DC-VOLTMETER',
          UL 5.25 V LL 4.75 V,
          VOLTAGE RANGE 0 V TO 5 V,
          CNX HI J1-BK11
              LO EARTH $
052999 END, 'VERIFICARE-TENSIUNE' $
C$
```

```
C ************************************************************************ $
C * PROCEDURE 'VERIFICARE-CURENT'                                        * $
C * RESOURCES USED: '28VDC PSU'                                          * $
C *                 'dc-ammeter'                                         * $
C ************************************************************************ $
053000 DEFINE, 'VERIFICARE-CURENT', PROCEDURE
                ('VOLTAGE', 'UL AMP', 'LL AMP' IS DECIMAL) $
C$
     10 CHANGE, DC SIGNAL USING '28VDC PSU',
          VOLTAGE 'VOLTAGE' RANGE 0V TO 5 V ERRLMT +- 0.1 V,
          CURRENT MAX 1 A,
          CNX HI J1-F21
              LO EARTH $
     20 WAIT FOR, TIME 1.0 SEC $
     30 VERIFY, (CURRENT), DC SIGNAL USING 'DC-AMMETER',
          UL 'UL AMP' A LL 'LL AMP' A,
          CURRENT MAX 1 A,
        CNX VIA J1-F21 $
 053999 END, 'VERICARE-CURENT' $
C$
C ************************************************************************ $
C * PROCEDURE 'BSCAN-TEST'                                               * $
C * SCOP: ACEASTA PROCEDURA INCARCA VECTORII DE TEST IN FPGA SI           * $
C * CITESTE RASPUNSUL                                                    * $
C * VARIABLES USED:                                                      * $
C *   -VAR-            -TYPE-             -FUNCTION-                     * $
C *  'STIM-DATA'        STRING OF BITS  BIT PATTERN                      * $
C *  'MASK-DATA'        STRING OF BITS  BIT PATTERN                      * $
C *  'EXPECT-DATA'      STRING OF BITS  BIT PATTERN                      * $
C *  'HI-PIN'           CONN            UUT PIN CONNECTION               * $
C *  'LO-PIN'           CONN            UUT PIN CONNECTION               * $
C *                                                                      * $
C * PROCEDURES USED: NONE.                                               * $
C ************************************************************************ $
057000 DEFINE, 'BSCAN-TEST', PROCEDURE
                    ('STIM-DATA' IS STRING(975) OF BIT,
                     'MASK-DATA' IS STRING(975) OF BIT,
                     'EXPECT-DATA' IS STRING(975) OF BIT) $
C$
     10 STIMULATE, 'STIM-DATA', ON 'BSCAN-DRIVER' $
     20 PROVE, (VALUE INTO 'DATA')
```

```
          REF 'EXPECT-DATA',
          MASK-ONE 'MASK-DATA',
          SAVE-COMP 'WORD',
          ON 'BSCAN-RECV' $
     30 IF ,NOGO, THEN
        OUTPUT, TEXT, FROM C'RASPUNS DIFERIT DE CEL ASTEPTAT' $

 057999 END, 'BSCAN-TEST' $

C ************************************************************************ $
C * PROCEDURE 'CITIRE-FISIER'                                            * $
C * SCOP: ACEASTA PROCEDURA CITESTE FISIERELE DE STIMULI, DE RESPUNS     * $
C * SI FISIERUL MASCA, EXECUTA CONVERSII SI REINTOARCE ARILE DE DATE     * $
C * CORESPUNZATOARE STIMULILOR, DATELOR CORECTE SI MASTI.                 * $
C * INTRUCIT OPERATIILE IMPLICATE DE ACEASTA PROCEDURA SUNT RELATIV      * $
C * COMPLICATE DATORITA LIMITARILOR SINTAXEI ATLAS PROCEDURA NU VA       * $
C * FI PREZENTATA COMPLET.                                                * $
C * VARIABLES USED:                                                      * $
C *    -VAR-            -TYPE-              -FUNCTION-                   * $
C *   'FILE.STM'         STRING OF CHAR                                   * $
C *   'EXPECTED.STM'     STRING OF CHAR                                   * $
C *   'MASK.STM'         STRING OF CHAR                                   * $
C *   'STIM-DATA'        STRING OF BITS  BIT PATTERN                      * $
C *   'MASK-DATA'        STRING OF BITS  BIT PATTERN                      * $
C *   'EXPECT-DATA'      STRING OF BITS  BIT PATTERN                      * $
C ************************************************************************ $
058000 DEFINE, 'BSCAN-TEST', PROCEDURE
                (
                 'STIM-NAME' IS STRING(256) OF CHAR,
                 'EXPECTED-NAME' IS STRING(256) OF CHAR,
                 'MASK-NAME' IS STRING(256) OF CHAR,

                 'STIM-DATA' IS STRING(976) OF BIT,
                 'MASK-DATA' IS STRING(976) OF BIT,
                 'EXPECT-DATA' IS STRING(976) OF BIT) $
C$
C    SE CITESC FISIERELE DE STIMULI, DE RESPUNS ASTEPTAT SI MASCA.    $
C    SE EXECUTA CONVERSII DE FORMAT PENTRU CA DATELE REZULTATE SA     $
C    FIE ARII DE SIRURI DE 976 DE BITI.                                $
 087999 END, 'BSCAN-TEST' $
```

```
C *                          END OF PREAMBLE                                * $
C ************************************************************************** $
C                                                                             $
 100000 COMMENCE, MAIN PROCEDURE $
C                                                                             $
C ************************************************************************** $
C *                           MAIN PROCEDURE                                * $
C ************************************************************************** $
 100010 PERFORM, 'VERIFICARE-TUA' ('J1-1',   'J1-2',  120) $
 100020 PERFORM, 'VERIFICARE-TENSIUNE' ('J1-1',   'J1-2',  120) $
 100030 PERFORM, 'VERIFICARE-CURENT' (110, 5, 0.3) $
 100040 PERFORM, 'CITIRE-FISIERE' ('STIM.STM','EXPECT.STM','MASK.STM',
                              'STIM1', 'MASK1', 'EXPECTED-DATA') $
 100050 PERFORM, 'BSCAN-TEST' ('STIM1', 'MASK1', 'EXPECT-DATA') $
 999990 FINISH $
 999999 TERMINATE, ATLAS PROGRAM 'EXEMPLU'                                    $
```

Anexa E. Vector de test pentru unitatea de control a încărcăturii (CSC)

1. Exemplu de vector de test pentru testarea ieşirilor

```
! i1_01.INV, @Demian Petru, 24.08.1997
TRST ABSENT;
ENDIR IDLE;
ENDDR IDLE;
SIR 6 TDI (00);
SDR                                                                    976
TDI(80071C01C01F8E001FFFC7FFFFC7FC7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFF1FFE3FFF8000E001C01C0007000000038FC70070BFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFF);
SDR                                                                    976
TDI(80071C01C01F8E001FFFC7FFFFC7FC7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFF1FFE3FFF8000E001C01C0007000000038FC70070BFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFF);
```

2. Exemplu de vector de test pentru testarea ieşirilor de curent alternativ

```
! ac01.INV
TRST ABSENT;
ENDIR IDLE;
ENDDR IDLE;
SIR 6 TDI (00);
SDR                                                                    976
TDI(FFFFFFFFFFFC75FFE800BFE170381F8E00E3F03FFFFE07E381FFFFF1FFFFFFF007FFFFFFC7F
F8FFFFFFFFFFFFFFFFFFFFE3FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFF);
```

3. Crearea vectorilor de test

IN THE CARGO SISTEM CONTROLLER DOCUMENTATION THE STIMULUS AND EXPECTED VECTOR IS DESCRIBE IN FILES "w02db000.exp" AND "w02db000.sti".

FOR GENERATE THE "SBS" FILE IT IS NECESSARY TO SLIT THE SOURCE FILE (one file for each vector) AND USE SOME UTILITY FILES.

THE "w02db000.exp" WAS SPLIT IN FILES WITH NEAR THE SAME NAME LIKE THE VECTOR NAME (because the vector name was to long) WITH THE EXTENSION .EXP AND THE FILE "w02db000.sti" WAS SLIT IN FILES WITH EXTENSION .STI.

FOR MAKE *.SBS PLEASE USE FOLLOING COMMAND LINE:

m1.bat i1_01 ENTER (for vector i1_01.sti)

m2.bat ac01 ENTER (for vector ac01.sti)

Content of m1.bat and m2.bat

```
******** m1.bat ******************************************************
INV %1.STI %1.INV
SVF1 %1.INV %1.SVF
DEL %1.INV
SVF2SBSF %1.SVF %1.OUT -fs -b
ADDHEAD %1.OUT IN.SBS
DEL %1.MAP
DEL %1.OUT
CONV
MOVE OUT %1.HEX_

***** m2.bat ************************************************************
INV %1.STI %1.INV
SVF2 %1.INV %1.SVF
DEL %1.INV
SVF2SBSF %1.SVF %1.OUT -fs -b
ADDHEAD %1.OUT IN.SBS
DEL %1.MAP
DEL %1.OUT
CONV
MOVE OUT %1.HEX_
```

Used utility programs:

- INV.EXE is use to inverse the order of data bit in the input file %1.sti and writes the output file %1.inv. This utility program was written in C++.
- SVF1.EXE make *.svf file using the input file %1.inv. This program was written in C++.
- SVF2.EXE make *.svf file using the input file %1.inv. This file was written in C++.
- SVF2SBSF.EXE make *.sbs file (Tecktronix software)
- ADDHEAD.EXE add header to *.sbs file (Tecktronix software)
- CONV.EXE convert binary *.sbs file in the *.hex file for export to UNIX

In UNIX OS it is necessary to convert %1.HEX file in STRING 80 OF CHART (for read in PAWS) and is use the following command line:

```
ASC2BIN %1.HEX  "vector name"       ENTER
```

Anexa F. Modelarea în VHDL

Anexa F.1. Modelul în VHDL a magistralei IEEE 1149.1

```
--
--  File: c:\my designs\b1149\SRC\TBC.VHD
--  created by Design Wizard: 07/31/98 15:38:16
--  Autor: Petru Demian
--  Version 2.1
library IEEE;
use IEEE.std_logic_1164.all;
use std.textio.all;

entity TBC is
    port (
        TDI: inout STD_LOGIC;
        TDO: out STD_LOGIC;
        TMS: out STD_LOGIC;
        TCK: inout STD_LOGIC;
        RST: inout STD_LOGIC
    );
end TBC;

architecture TBC of TBC is
begin
 CLOCK: process
    variable clktmp: std_logic := '0';
  begin
        TCK <= clktmp;  -- Attach your clock here
        clktmp := not clktmp;
        wait for 50 NS;
  end process CLOCK;

 PROCEDURE_MAIN: process
 variable data :Std_Logic_Vector(17 downto 0);

-- PROCEDURA INITIALIZEAZA IESIRILE PE NIVELE LOGICE IMPLICITE
  procedure INIT is
   begin
    TMS <= '1';
    TDO <= '1';
    TDI <= 'H';
```

```
    RST <= 'H';
   end INIT;

-- RESET SOFT
  procedure RESET is
   begin
    report "Soft RESET";
    for I in 0 to 5 loop
      TMS <= '1';
      wait until FALLING_EDGE(TCK);
    end loop;
   end RESET;
-- END RESET SOFT


-- PROCEDURA INCARCA IN REGISTRUL DE INSTRUCTUINE CONTINUTUL PRECIZAT DE
"IP_REGISTER"
  procedure INSTRUCTION(
    IP_REGISTER: in STD_LOGIC_VECTOR(7 downto 0)) is
   begin
     report "Send instruction";
-- se trece din starea TEST-LOGIC-RESET in starea RUN-TEST/IDLE
    TMS <= '0'; --
    wait until RISING_EDGE(TCK);

-- se trece in starea SELECTE-DR-SCAN
    TMS <= '1';
    wait until RISING_EDGE(TCK);

-- se trece in starea SELECTE-IR-SCAN
    TMS <= '1';
    wait until RISING_EDGE(TCK);

-- se trece in starea CAPTURE-IR
    TMS <= '0';
    wait until RISING_EDGE(TCK);

-- se trece in starea SHIFT-IR si se transmite pe linia TDO instructiunea
"IP_REGISTER"
   TMS <= '0';
   for I in 0 to 7 loop
```

```
    TDO <= IP_REGISTER(7- I);
    wait until RISING_EDGE(TCK);
   end loop;

-- se trece in starea EXIT1-IR
   TMS <= '1';
   wait until RISING_EDGE(TCK);

-- se trece in starea UPDATE-IR
   TMS <= '0';
   wait until RISING_EDGE(TCK);

-- se trece in starea RUN-TEST/IDLE
   TMS <= '0';
   wait until RISING_EDGE(TCK);

   end INSTRUCTION;


-- INSTRUCTIUNEA INCARCA IN REGISTRUL DE DATE CONTINUTUL PRECIZAT DE
"DATA_REGISTER"
 procedure LOAD_DATA (
   DATA_REGISTER: in STD_LOGIC_VECTOR(17 downto 0)) is
   variable DATA_INPUT: Std_Logic_Vector(17 downto 0);
 begin
    report "Send DATA and read respons";
-- se trece din starea TEST-LOGIC-RESET in starea RUN-TEST/IDLE
    TMS <= '0'; --
    wait until RISING_EDGE(TCK);

-- se trece in starea SELECTE-DR-SCAN
    TMS <= '1';
    wait until RISING_EDGE(TCK);

-- se trece in starea CAPTURE-DR
    TMS <= '0';
    wait until RISING_EDGE(TCK);

-- se trece in starea SHIFT-DR si se transmite pe linia TDO data "DATA_REGISTER"
    TMS <= '0';
    for I in 0 to 17 loop
```

```
      TDO <= DATA_REGISTER(17- I);
      wait until RISING_EDGE(TCK);
      wait until FALLING_EDGE(TCK);
      DATA(17-I) := TDI;
     end loop;

-- se trece in starea EXIT1-IR
     TMS <= '1';
     wait until RISING_EDGE(TCK);

-- se trece in starea UPDATE-IR
     TMS <= '0';
     wait until RISING_EDGE(TCK);

-- se trece in starea RUN-TEST/IDLE
     TMS <= '0';
     wait until RISING_EDGE(TCK);
 end LOAD_DATA;
-----------------------------------------------------------------
-----------------------------------------------------------------
-- Programul principal                                          -
-----------------------------------------------------------------
 begin
  INIT;                -- initializeaza semnalele de pe bus
  RESET;               -- reset soft
  INSTRUCTION(X"11"); -- trimite in registrul IR codul 11 HEXA
  LOAD_DATA("0111010100001010001");   -- trimite in registrul DR codul 5A HEXA
  RESET;
  wait;
 end process PROCEDURE_MAIN;
 end TBC;
```

Anexa F.2. Modelul în VHDL pentru TMS320c203

```
--=============================================================================--
--
-- Design units    : TMS320C203(BoardLevel) (Entity and architecture)
--
-- File name       : T320C203.vhd
--
-- Purpose         : The TMS320C203 is a digital signal processor produced by
--                   Texas Instruments.
--
-- Note            : All timing parameter names are compliant with Vital Level 0.
--                   Selection of Minimal, Nominal or Maximal timing is performed
--                   by the TMS320C203_Lib.Simulation.SimCond type generic SimCondition.
--                   Selection of processor type is performed by the
--                   TMS320C203_Lib.Simulation.ProcessorType type generic Processor.
--
--                   This model is modelled after the design (the design was not
--                   synthesised from this model). The model is compliant to the
--                   data sheet for the TMS320C203 dated June 1995
--                   ( code : SPRS025 ) and the TMS320C203 User's Manual
--                   (code : SPRU127b)
--
--
-- Limitations     : Test pins not included in model. Also, TOUT,BOOT,BIO,XF,PLL5V
--
-- Errors          : None known
--
-- Library         : TMS_Lib
--
-- Dependencies    : IEEE.Std_Logic_1164
--
--
-- Simulator       : ModelSim PE/PLUS version 4.7b on a Windows95 PC
--                   ActiveVHDL version 3.2 on a Windows95 PC
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Revision list
-- Version    Author        Date           Changes
--
-- 0.1        Demian Petru, Ovidiu Lupas 06 March 98      New model
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- Entity for CLKOUT1 generator
-------------------------------------------------------------------------------
library ieee,work;
use ieee.Std_Logic_1164.all;
use work.Simulation.all;
use work.TMS320C203_Tim.all;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
entity Clk is
    generic(
      tperiod_CI      : Time := Tables_tperiod_CI(TMS320C203_50_5V)(Nominal);      -- tc(CI)
      tpw_CI_posedge  : Time := Tables_tpw_CI_posedge(TMS320C203_50_5V)(Nominal); -- tw(CIH)
      tpw_CI_negedge  : Time := Tables_tpw_CI_negedge(TMS320C203_50_5V)(Nominal); -- tw(CIL)
      tperiod_CO      : Time := Tables_tperiod_CO(TMS320C203_50_5V)(Nominal);          -- tc(CO)
      tpw_CO_posedge  : Time := Tables_tpw_CO_posedge(TMS320C203_50_5V)(Nominal); -- tw(COH)
      tpw_CO_negedge  : Time := Tables_tpw_CO_negedge(TMS320C203_50_5V)(Nominal); -- tw(COL)
      tdelay_CIH_CO   : Time := Tables_tdelay_CIH_CO(TMS320C203_50_5V)(Nominal)); -- td(CIH-CO)
    port (
        -- Oscillator, PLL and Timer signals (7)
        CLKOUT1         : inout Std_Logic;     -- Master clock output
        X2              : in    Std_Logic;     -- Oscillator input
        DIV1            : in    Std_Logic;     -- Clock mode 1
        DIV2            : in    Std_Logic;     -- Clock mode 2
        CLKOUT1_ENABLE  : in    Std_Logic);    -- Enables the master clock generation
end Clk;
-------------------------------------------------------------------------
-- Architecture for CLKOUT1 generator
-------------------------------------------------------------------------
architecture BEHAVIOR_Clk of Clk is
-------------------------------------------------------------------------
-- Signals
-------------------------------------------------------------------------
```

```
      signal X2_Delay_X2_1,X2_Delay_X2_0 : Std_Logic;
      signal temp_CLKOUT1 : Std_Logic;
begin--========================== Architecture ==============================--
  ------------------------------------------------------------------------------
  -- Function of CLKOUT1 bit of register CLK not implemented.
  -- Implementation of system clock for configuration of DIV pins 0 and 1
  ------------------------------------------------------------------------------
  SysClk : process (X2,CLKOUT1_ENABLE)
      ---------------------------------------------------------------------
      -- enables, when true, to change the CLKOUT1 level
      ---------------------------------------------------------------------
      variable clktmp  : Std_Logic := '1';
      variable FirstTime: Boolean := true;
      variable tdel    : Time;
  begin
      if FirstTime then       -- Initialize the Clkout1 signal and DIV variable
         FirstTime := false;
         CLKOUT1 <= '1';
      end if;
      if not X2'STABLE then
         if ((DIV1 = '0') and (DIV2 = '0')) then -- divide by 2
            if X2= '1' then
               CLKOUT1 <= transport not CLKOUT1 after tdelay_CIH_CO;
             end if;
         end if;
         if ((DIV1 = '1') and (DIV2 = '0')) then -- multiply by one
            CLKOUT1 <= transport not CLKOUT1 after tdelay_CIH_CO;
         end if;
         if ((DIV1 = '0') and (DIV2 = '1')) then -- multiply by two
            tdel := tpw_CI_posedge;-- + tdelay_CIH_CO;
            CLKOUT1 <= transport not CLKOUT1;-- after tdelay_CIH_CO;
            CLKOUT1 <= transport CLKOUT1 after tdel;
         end if;
         if ((DIV1 = '1') and (DIV2 = '1')) then -- multiply by four
            CLKOUT1 <= transport not CLKOUT1;                    -- after tdelay_CIH_CO;
            CLKOUT1 <= transport CLKOUT1 after tpw_CI_posedge/2;  -- + tdelay_CIH_CO);
            CLKOUT1 <= transport not CLKOUT1 after tpw_CI_posedge;-- + tdelay_CIH_CO);
            CLKOUT1 <= transport CLKOUT1 after 3 * tpw_CI_posedge/2;-- + tdelay_CIH_CO);
         end if;
      end if; -- end if X2'STABLE
  end process SysClk;
end BEHAVIOR_Clk;--============== End of architecture ==============--
------------------------------------------------------------------------
-- Entity for reset and interrupt events
------------------------------------------------------------------------
library ieee,work;
use ieee.Std_Logic_1164.all;
use work.Simulation.all;
use work.TMS320C203_Tim.all;
use work.TMS320C203_Def.all;
------------------------------------------------------------------------
------------------------------------------------------------------------
entity RstInt is
 generic(
    tperiod_CI        : Time := Tables_tperiod_CI(TMS320C203_50_5V)(Nominal);      -- tc(CI)
    -- RS_N,INTx_N,NMI_N,BIO_N,TOUT and XF timing parameters
    tsetup_RS_CI      : Time := Tables_tsetup_RS_CI(TMS320C203_50_5V)(Nominal);    -- tsu(RS)CI
    tsetup_RS_CO      : Time := Tables_tsetup_RS_CO(TMS320C203_50_5V)(Nominal);    -- tsu(RS)CO
    tpw_RS_negedge    : Time := Tables_tpw_RS_negedge(TMS320C203_50_5V)(Nominal); -- tw(RSL)
    tdelay_RSN_fetch  : Time := Tables_tdelay_RSN_fetch(TMS320C203_50_5V)(Nominal);  -- td(EX)
    tpw_INTx_negedge  : Time := Tables_tpw_INTx_negedge(TMS320C203_50_5V)(Nominal);  -- tw(IN)
    tdelay_INTx_fetch : Time := Tables_tdelay_INTx_fetch(TMS320C203_50_5V)(Nominal); -- td(IN)
    -- External DMA timing parameters
    tdelay_HLDN_HLDAN : Time := Tables_tdelay_HLDN_HLDAN(TMS320C203_50_5V)(Nominal); -- td(H-HA)
    tdelay_HLD_HLDA   : Time := Tables_tdelay_HLD_HLDA(TMS320C203_50_5V)(Nominal); -- td(HH-HA)
    thighimped_HLDAN  : Time := Tables_thighimped_HLDAN(TMS320C203_50_5V)(Nominal); -- tz(M-HA)
    tenable_HLDA      : Time := Tables_tenable_HLDA(TMS320C203_50_5V)(Nominal));   -- ten(HA-M)
 port (
      -- Data and Address Buses (32)
     DataB    : inout  Std_Logic_Vector(15 downto 0);  -- Data bus
     AddrB    : out  Std_Logic_Vector(15 downto 0);  -- Address bus
     -- Memory control signals (8)
     PS_N     : out    Std_Logic;      -- Program select
     DS_N     : out    Std_Logic;      -- Data select
     RW_N     : out    Std_Logic;      -- Read/write
     IS_N     : out    Std_Logic;      -- I/O space select
     READY    : in     Std_Logic;      -- Data ready input
```

```
    RD_N      : out    Std_Logic;       -- Read select
    WE_N      : out    Std_Logic;       -- Write enable
    STRB_N    : out    Std_Logic;       -- Strobe
    -- Oscillator and PLL signals (2)
    CLKOUT1   : in     Std_Logic;       -- Master clock output
    X2        : in     Std_Logic;       -- clock input
    -- Multi-processing signals (2)
    BR_N      : out    Std_Logic;       -- Bus request
    HOLDA_N   : out    Std_Logic;       -- Hold acknowledge
    -- Initialization, Interrupts and Reset signals (5)
    RS_N      : in     Std_Logic;       -- Hardware reset
    NMI_N     : in     Std_Logic;       -- Nonmaskable interrupt
    INT1_N    : in     Std_Logic;       -- Interrupt 1 /HOLD_N request
    INT2_N    : in     Std_Logic;       -- Interrupt 2
    INT3_N    : in     Std_Logic;       -- Interrupt 3
    -- Internal signals
    RINT           : in     Std_Logic;       -- Synchronous serial port/receive
    XINT           : in     Std_Logic;       -- Synchronous serial port/transmit
    TXRXINT        : in     Std_Logic;       -- Asynchronous serial port
    Stop           : out    Std_Logic;       -- Internal signal
    CLKOUT1_ENABLE : out    Std_Logic;       -- Internal signal
    Soft_Reset     : out    Std_Logic );
end RstInt;--================== End of entity ======================--
-------------------------------------------------------------------------
-- Architecture for Reset and Interrupts controller
-------------------------------------------------------------------------
architecture BEHAVIOR_RstInt of RstInt is
     signal Local : Std_Logic;
-------------------------------------------------------------------------
-- Global variables
-------------------------------------------------------------------------
       ---------------------------------------------------------------------
       shared variable MODE,INTM : Bit := '1';      -- HOLD mode!!!
begin--========================== Architecture ==============================--
  ---------------------------------------------------------------------------------------
  -- Implements all the interrupt events.
  -- For maskable interrupts, the functions of mask bits are not implemented
  -- If an interrupt occures, the interrupt vector will be fetched after 9 cycles
  ---------------------------------------------------------------------------------------
ResetInt : process  (NMI_N,INT1_N,INT2_N,INT3_N,RINT,XINT,TXRXINT,RS_N,X2,Local)
       variable tdel : Time;
       variable Reset,Int : Boolean := false;
       variable Counter : Integer;
  ---------------------------------------------------------------------------
  ---------------------------------------------------------------------------
begin
  ---------------------------------------------------------------------------------------
  -- Implementation of all functions driven by RS_N
  ---------------------------------------------------------------------------------------
    if (RS_N'EVENT and RS_N = '0') then
       Reset := true;
       Counter := 0;
       Soft_Reset <='1';
    elsif (X2'EVENT and X2 = '0' and Reset = true and RS_N = '0') then
       Counter := Counter + 1;
    elsif (RS_N'Event and RS_N = '1') then
       Reset := false;
       if Counter > 6 then
          STOP <= '1';  -- the MAIN process is disabled
          CLKOUT1_ENABLE <= '1'; -- the SYSCLK process is disabled
          CLKOUT1_ENABLE <= '0' after 50 ns;  -- the SYSCLK process is enabled
          MODE := '0';   -- Mode bit is set to 0
          AddrB <= "0000000000000000" after tdelay_RSN_fetch;
          Local <= '1' after tdelay_RSN_fetch;
       elsif Counter < 6 then
            report "Not a valid reset operation: RS_N was not asserted for at least 6 clock
cycles."
            severity Warning;
       end if;
    elsif (not INT1_N'STABLE and Reset = false) then
       if (INT1_N = '0' and MODE = '1') then        -- interrupt1 mode
           Int  := true;
           Counter := 0;
       elsif (INT1_N = '0' and MODE = '0') then   -- entering the Hold mode
             tdel := tdelay_HLDN_HLDAN - thighimped_HLDAN;
             STOP <= '1';
             AddrB <= "ZZZZZZZZZZZZZZZZ" after tdel;
```

```
            DataB <= "ZZZZZZZZZZZZZZZZ" after tdel;
            PS_N <= 'Z' after tdel;
            DS_N <= 'Z' after tdel;
            IS_N <= 'Z' after tdel;
            BR_N <= 'Z' after tdel;
            STRB_N <= 'Z' after tdel;
            RD_N <= 'Z' after tdel;
            RW_N <= 'Z' after tdel;
            WE_N <= 'Z' after tdel;
            HOLDA_N <= '0' after tdelay_HLDN_HLDAN;
      elsif INT1_N = '1' then              -- exiting the Hold mode
            tdel := tdelay_HLD_HLDA + tenable_HLDA;
            HOLDA_N <= '1' after tdelay_HLD_HLDA;
            AddrB <= "0101010101010101" after tdel;
            DataB <= "1010101010101010" after tdel;
            PS_N <= '1' after tdel;
            DS_N <= '1' after tdel;
            IS_N <= '1' after tdel;
            BR_N <= '1' after tdel;
            STRB_N <= '1' after tdel;
            RD_N <= '1' after tdel;
            RW_N <= '1' after tdel;
            WE_N <= '1' after tdel;
            STOP <= '0' after tdel;
            Local <= '1' after tdel;
      end if;
    elsif (CLKOUT1'EVENT and CLKOUT1 = '0' and Int = true and INT1_N = '0') then
          Counter := Counter + 1;
    elsif (INT1_N'Event and INT1_N = '1' and Int = true) then
          Int := false;
          if Counter > 1 then
                AddrB <= "0000000000000010" after tdelay_INTx_fetch;
                Local <= '1' after tdelay_INTx_fetch;
          elsif Counter < 1 then
              report "Not a valid interrupt operation: INT1_N was not asserted for at least 1
CLKOUT1 cycle."
              severity Warning;
          end if;
    elsif (not INT2_N'STABLE and INT2_N = '0' and Reset = false) then
          Int := true;
          Counter := 0;
    elsif (CLKOUT1'EVENT and CLKOUT1 = '0' and Int = true and INT2_N = '0') then
          Counter := Counter + 1;
    elsif (INT2_N'Event and INT2_N = '1' and Int = true) then
          Int := false;
          if Counter > 1 then
                AddrB <= "0000000000000100" after tdelay_INTx_fetch;
                Local <= '1' after tdelay_INTx_fetch;
          elsif Counter < 1 then
              report "Not a valid interrupt operation: INT2_N was not asserted for at least 1
CLKOUT1 cycle."
              severity Warning;
          end if;
    elsif (not INT3_N'STABLE and INT3_N = '0' and Reset = false) then
          Int := true;
          Counter := 0;
    elsif (CLKOUT1'EVENT and CLKOUT1 = '0' and Int = true and INT3_N = '0') then
          Counter := Counter + 1;
    elsif (INT3_N'Event and INT3_N = '1' and Int = true) then
          Int := false;
          if Counter > 1 then
                AddrB <= "0000000000000100" after tdelay_INTx_fetch;
                Local <= '1'after tdelay_INTx_fetch;
          elsif Counter < 1 then
              report "Not a valid interrupt operation: INT3_N was not asserted for at least 1
CLKOUT1 cycle."
              severity Warning;
          end if;
    elsif (not RINT'STABLE and RINT = '1') then
          AddrB <= To_StdLogicVector(7) after tdelay_INTx_fetch;
          Local <= '1' after tdelay_INTx_fetch;
    elsif (not XINT'STABLE and XINT = '1') then
          AddrB <= To_StdLogicVector(8) after tdelay_INTx_fetch;
          Local <= '1' after tdelay_INTx_fetch;
    elsif (not TXRXINT'STABLE and TXRXINT = '1') then
          AddrB <= To_StdLogicVector(9) after tdelay_INTx_fetch;
          Local <= '1' after tdelay_INTx_fetch;
```

```
      elsif (not NMI_N'STABLE and NMI_N = '0' and Reset = false) then
            Int := true;
            Counter := 0;
      elsif (CLKOUT1'EVENT and CLKOUT1 = '0' and Int = true and NMI_N = '0') then
            Counter := Counter + 1;
      elsif (NMI_N'Event and NMI_N = '1' and Int = true) then
            Int := false;
            if Counter > 1 then
                  AddrB <= To_StdLogicVector(24) after tdelay_INTx_fetch;
                  Local <= '1' after tdelay_INTx_fetch;
            elsif Counter < 1 then
                report "Not a valid interrupt operation: NMI_N was not asserted for at least 1
CLKOUT1 cycle."
                severity Warning;
            end if;
      elsif Local = '1' then
            Local <= '0';
            AddrB <= "ZZZZZZZZZZZZZZZZ" after tperiod_CI;
            DataB <= "ZZZZZZZZZZZZZZZZ" after tperiod_CI;
            PS_N <= 'Z' after tperiod_CI;
            DS_N <= 'Z' after tperiod_CI;
            IS_N <= 'Z' after tperiod_CI;
            BR_N <= 'Z' after tperiod_CI;
            STRB_N <= 'Z' after tperiod_CI;
            RD_N <= 'Z' after tperiod_CI;
            RW_N <= 'Z' after tperiod_CI;
            WE_N <= 'Z' after tperiod_CI;
            HOLDA_N <= 'Z' after tperiod_CI;
            Soft_Reset <= '0' after 2 * tperiod_CI;
       end if;
end process ResetInt;
end BEHAVIOR_RstInt;--================ End of architecture ================--
------------------------------------------------------------------------
-- Entity for memory access events
------------------------------------------------------------------------
library ieee,work;
use ieee.Std_Logic_1164.all;
use work.Simulation.all;
use work.TMS320C203_Tim.all;
use work.TMS320C203_Def.all;          -- For custom functions
------------------------------------------------------------------------
------------------------------------------------------------------------
entity MemAcc is
  generic(
    tsetup_A_RDN    : Time := Tables_tsetup_A_RDN(TMS320C203_50_5V)(Nominal);   -- tsu(A)RD
    thold_A_RD      : Time := Tables_thold_A_RD(TMS320C203_50_5V)(Nominal);     -- th(A)RD
    tdelay_CON_A    : Time := Tables_tdelay_CON_A(TMS320C203_50_5V)(Nominal);   -- td(COL-A)
    thold_A_CON     : Time := Tables_thold_A_CON(TMS320C203_50_5V)(Nominal);    -- th(A)CO
    tdelay_CO_RD    : Time := Tables_tdelay_CO_RD(TMS320C203_50_5V)(Nominal);   -- td(CO-RD)
    tdelay_CO_S     : Time := Tables_tdelay_CO_S(TMS320C203_50_5V)(Nominal);    -- td(COL-S)
    tpw_RD_negedge  : Time := Tables_tpw_RD_negedge(TMS320C203_50_5V)(Nominal);-- tw(RDL)
    tpw_RD_posedge  : Time := Tables_tpw_RD_posedge(TMS320C203_50_5V)(Nominal);-- tw(RDH)
    tdelay_RD_WEN   : Time := Tables_tdelay_RD_WEN(TMS320C203_50_5V)(Nominal);  -- td(RDW)
    taccess_A       : Time := Tables_taccess_A(TMS320C203_50_5V)(Nominal);      -- ta(A)
    tsetup_D_RD     : Time := Tables_tsetup_D_RD(TMS320C203_50_5V)(Nominal);    -- tsu(D)RD
    thold_D_RD      : Time := Tables_thold_D_RD(TMS320C203_50_5V)(Nominal);     -- th(D)RD
    thold_D_A       : Time := Tables_thold_D_A(TMS320C203_50_5V)(Nominal);      -- th(D)A
    tsetup_D_CON_R  : Time := Tables_tsetup_D_CON_R(TMS320C203_50_5V)(Nominal);-- tsu(DCOL)R
    thold_D_CON_R   : Time := Tables_thold_D_CON_R(TMS320C203_50_5V)(Nominal);  -- th(DCOL)R
    taccess_RD      : Time := Tables_taccess_RD(TMS320C203_50_5V)(Nominal);     -- ta(RD)
      -- Write timing parameters
    tsetup_A_WN     : Time := Tables_tsetup_A_WN(TMS320C203_50_5V)(Nominal);    -- tsu(A)W
    thold_A_W       : Time := Tables_thold_A_W(TMS320C203_50_5V)(Nominal);      -- th(A)W
    tsetup_A_CON    : Time := Tables_tsetup_A_CON(TMS320C203_50_5V)(Nominal);   -- tsu(A)CO
    thold_A_CON_W   : Time := Tables_thold_A_CON_W(TMS320C203_50_5V)(Nominal);  -- th(A)COw
--  tpw_NSN         : Time :=Tables_tpw_NSN(TMS320C203_50_5V)(Nominal);         -- tw(NSN)
    tpw_W_negedge   : Time := Tables_tpw_W_negedge(TMS320C203_50_5V)(Nominal);  -- tw(WL)
--  tpw_W_posedge   : Time := Tables_tpw_W_posedge(TMS320C203_50_5V)(Nominal);  -- tw(WH)
    tdelay_CO_WE    : Time := Tables_tdelay_CO_WE(TMS320C203_50_5V)(Nominal);   -- td(CO-W)
--- tdelay_WE_RDN   : Time := Tables_tdelay_WE_RDN(TMS320C203_50_5V)(Nominal);  -- td(WRD)
    tsetup_D_WE     : Time := Tables_tsetup_D_WE(TMS320C203_50_5V)(Nominal);    -- tsu(D)W
    thold_D_WE      : Time := Tables_thold_D_WE(TMS320C203_50_5V)(Nominal);     -- th(D)W
    tsetup_D_CON_W  : Time := Tables_tsetup_D_CON_W(TMS320C203_50_5V)(Nominal);-- tsu(DCOL)W
    thold_D_CON_W   : Time := Tables_thold_D_CON_W(TMS320C203_50_5V)(Nominal));-- th(DCOL)W
--  ten_D_W         : Time := Tables_ten_D_W(TMS320C203_50_5V)(Nominal));       -- ten(D)W
  port (
      -- Data and Address Buses (32)
```

```
        DataB           : inout Std_Logic_Vector(15 downto 0);        -- Data bus
        AddrB           : out           Std_Logic_Vector(15 downto 0);        -- Address bus
        -- Memory control signals (8)
        PS_N            : out           Std_Logic;                            -- Program select
        DS_N            : out           Std_Logic;                            -- Data select
        RW_N            : out           Std_Logic;                            -- Read/write
        IS_N            : out           Std_Logic;                            -- I/O space select
        HOLDA_N         : out       Std_Logic;              -- Global Memory
        READY           : in            Std_Logic;                            -- Data ready input
        RD_N            : out           Std_Logic;                            -- Read select
        WE_N            : out           Std_Logic;                            -- Write enable
        STRB_N  : out           Std_Logic;                          -- Strobe
      -- Reset signals (1)
      RS_N          : in      Std_Logic;       -- Hardware reset
        -- Multi-processing signals (5)
        BR_N            : out           Std_Logic;                            -- Bus request
        -- Oscillator signal (1)
        CLKOUT1 : in            Std_Logic;
        VAddrB  : in    Std_Logic_Vector(15 downto 0);
        VDataB  : in    Std_Logic_Vector(15 downto 0);
        VRW_N           : in    Std_Logic;
        VIS_N           : in    Std_Logic;
        VDS_N           : in    Std_Logic;
        VPS_N           : in    Std_Logic;
        VBR_N           : in    Std_Logic;
        Busy_cycle  :out Std_Logic;
        Wait_nr_cycle    : in Std_Logic_Vector(15 downto 0));
end MemAcc; --================== End of entity ======================--
-------------------------------------------------------------------------
-- Architecture for memory access controller
-------------------------------------------------------------------------
architecture BEHAVIOR_MemAcc of MemAcc is
     signal  Read_DataB : Std_Logic_Vector(15 downto 0);     -- Data bus
begin--============================ Architecture ================================--
  ---------------------------------------------------------------------------------
  -- Implements the read and write cycles to all memory spaces
  ---------------------------------------------------------------------------------
  Read_Write : process (CLKOUT1,VIS_N,VDS_N,VPS_N,VBR_N,RS_N)
     variable read_on_off : Boolean := false;
     variable read_clkout1: integer;
     variable read_wait: integer:=1;
     variable write_on_off : Boolean := false;
     variable write_clkout1: integer;
     variable write_wait: integer:=2;
     variable write_end: Boolean:= true;
     variable Good1,Good2,Good3 : Boolean := true;
     variable str4,str_4 : string(1 to 4);
     variable VVIS_N,VVDS_N,VVPS_N,VVBR_N,VVRW_N: Std_Logic;
     variable VVAddrB,VVDataB: Std_Logic_Vector(15 downto 0);
     variable cycle_new :Boolean :=false;
     variable read_on_off_temp,write_on_off_temp: Boolean :=true;
  ---------------------------------------------------------------------------------
  ---------------------------------------------------------------------------------
  begin
    if RS_N'EVENT and RS_N = '0' then -- initialize all the external signals used
      AddrB <= "ZZZZZZZZZZZZZZZZ";
      DataB <= "ZZZZZZZZZZZZZZZZ";
      PS_N <= 'Z';
      DS_N <= 'Z';
      IS_N <= 'Z';
      BR_N <= 'Z';
      STRB_N <= 'Z';
      RD_N <= 'Z';
      RW_N <= 'Z';
      WE_N <= 'Z';
    end if;
    -- end read cycle
    if read_on_off_temp =false and CLKOUT1'EVENT and CLKOUT1='0' then
       Read_DataB <= DataB; -- read the Data Bus
       RD_N <= transport '1';
 --    Vec2Hex (VVDataB,str_4,Good2);
 --    Vec2Hex (Read_DataB,str4,Good2);
 --    report "Read Cycle! Data bus value : "&str4 &". Expected Data : "&str_4 &""
 --    severity Note;
       AddrB <=transport "ZZZZZZZZZZZZZZZZ" after thold_A_CON;
       IS_N <= transport '1' after thold_A_CON;
       DS_N <= transport '1' after thold_A_CON;
```

```
      PS_N <= transport '1' after thold_A_CON;
      BR_N <= transport '1' after thold_A_CON;
      if cycle_new = false then
         read_on_off :=false;
      end if;
      read_on_off_temp:=true;
    end if;
 -- end write cycle
    if write_on_off_temp =false and CLKOUT1'EVENT and CLKOUT1='0' then
       write_on_off_temp := true;
       RD_N <= transport '1' ;
       AddrB <=transport "ZZZZZZZZZZZZZZZZ" after tdelay_CO_WE + thold_A_W;
       DataB <=transport "ZZZZZZZZZZZZZZZZ" after tdelay_CO_WE + thold_D_WE;
       IS_N <=transport '1' after tdelay_CO_WE + thold_A_W;
       DS_N <=transport '1' after tdelay_CO_WE + thold_A_W;
       PS_N <=transport '1' after tdelay_CO_WE + thold_A_W;
       BR_N <=transport '1' after tdelay_CO_WE + thold_A_W;
       WE_N <=transport '1' after tdelay_CO_WE;
       if cycle_new = false then
          write_on_off :=false;
       end if;
    end if;
-- store signal VIS,VDS,VPS and VBR --
    if (not VIS_N'STABLE and VIS_N ='0') or (not VDS_N'STABLE and VDS_N ='0') or
    (not VPS_N'STABLE and VPS_N ='0') or (not VBR_N'STABLE and VBR_N ='0')  then
        cycle_new := true;
        read_wait :=1 + To_Logic_Integer(Wait_nr_cycle);
        write_wait := 2 + To_Logic_Integer(Wait_nr_cycle);
        Busy_cycle <='1';
        VVIS_N := VIS_N;
        VVDS_N := VDS_N;
        VVPS_N := VPS_N;
        VVBR_N := VBR_N;
        VVRW_N := VRW_N;
        VVAddrB :=VAddrB;
        VVDataB :=VDataB;
        VVRW_N :=VRW_N;
        if VRW_N ='1' then
           read_clkout1 :=0;
           read_on_off :=true;
        else
           write_clkout1 :=0;
           write_on_off :=true;
        end if;
    end if;
    if read_on_off =true and CLKOUT1'EVENT then
       if read_clkout1 =0 and CLKOUT1 ='0' then
          IS_N <= transport VVIS_N after tdelay_CON_A;
          DS_N <= transport VVDS_N after tdelay_CON_A;
          PS_N <= transport VVPS_N after tdelay_CON_A;
          BR_N <= transport VVBR_N after tdelay_CON_A;
          RD_N <= transport '1';
          RD_N <= transport '0' after tsetup_A_RDN + tdelay_CON_A;
          AddrB <=transport VVAddrB after tdelay_CON_A;
          DataB <=transport "ZZZZZZZZZZZZZZZZ" after tdelay_CON_A + taccess_A; --!!
          WE_N <= transport '1';
          STRB_N <=transport '0' after tdelay_CO_S;
          RW_N <=transport '1';
       end if; -- end if read_clkout1
       if CLKOUT1='0' and READY ='1' then
          read_clkout1 :=read_clkout1 +1; -- count number of rising edge of CLKOUT1
       end if;
       if read_clkout1 =read_wait and CLKOUT1='1' then
          Busy_cycle <='L';
          read_on_off_temp:=false;
       end if;
    end if; -- end read_on_off
    if write_on_off =true and CLKOUT1'EVENT then
       if write_clkout1 =0 and CLKOUT1 ='0' then
          IS_N <=transport VVIS_N after thold_A_CON_W;
          DS_N <=transport VVDS_N after thold_A_CON_W;
          PS_N <=transport VVPS_N after thold_A_CON_W;
          BR_N <=transport VVBR_N after thold_A_CON_W;
          RD_N <=transport '1';
          AddrB <=transport VVAddrB after thold_A_CON_W;
          STRB_N <=transport '1';
          RW_N <=transport '0' after thold_A_CON_W;
```

```
--          STRB_N <= transport '0' after tdelay_CO_S;
            WE_N <=transport '0' after thold_A_CON_W + tsetup_A_WN;
            DataB <=transport VVDataB after tdelay_CO_WE + thold_D_WE;
            Vec2Hex (VVDataB,str4,Good2);
            report "Write Cycle! Data bus value : "&str4 &""
            severity Note;
         end if; -- end if read_clkout1
         if CLKOUT1='0' and READY='1' then
            write_clkout1 :=write_clkout1 +1; -- count number of rising edge of CLKOUT1
         end if;
         if write_clkout1 =2 then
            STRB_N <= '0';
         end if;
         if write_clkout1 =write_wait and CLKOUT1='1' then
            Busy_cycle <='L';
            write_on_off_temp:=false;
         end if;
         if Write_clkout1 >Write_wait and CLKOUT1='0' then  -- end read cycle
         end if;
      end if; --write_on_off
  end process Read_Write;
end BEHAVIOR_MemAcc;--================ End of architecture ================--
-------------------------------------------------------------------------
-- Entity for asynchronous serial port                                 --
-------------------------------------------------------------------------
library ieee,work;
use STD.textio.all;
use ieee.std_logic_1164.all;
use work.TMS320C203_Def.all;          -- For custom functions
use work.TMS320C203_Tim.all;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
entity UART is
--    generic(CLK_PERIOD:TIME );
     port (
    TX       : out Std_logic;               -- Asynchronous transmit pin
    RX       : in  Std_logic;               -- Asynchronous receive pin
    CLKOUT1  : in  Std_logic;               -- Master clock output
    ADTR     : inout Std_Logic_Vector(15 downto 0); -- Asynchronous data transmit/receive register
    ASPCR    : in Std_Logic_Vector(15 downto 0); -- Asynchronous serial port control register
    IOSR     : in Std_Logic_Vector(15 downto 0); -- Asynchronous I/O status register
    BRD      : in  Std_Logic_Vector(15 downto 0);  -- Asynchronous boud-rate devisor register
    TXRXINT  : out Std_Logic;               -- Asynchronous hardware interrupt
    LOAD     : in Std_Logic;                -- Start UART transmision
    READ     : in Std_Logic;                -- Read UART receive data
    IO0      : inout Std_Logic;             -- Software controlled I/O pin 0
    IO1      : inout Std_Logic;             -- Software controlled I/O pin 1
    IO2      : inout Std_Logic;             -- Software controlled I/O pin 2
    IO3      : inout Std_Logic;             -- Software controlled I/O pin 3
    Soft_Reset : in Std_Logic);             -- Internal signal
end UART;
-------------------------------------------------------------------------
-- Architecture for asynchronous serial port                           -
-------------------------------------------------------------------------
library ieee,work;
use STD.textio.all;
use ieee.std_logic_1164.all;
use work.TMS320C203_Def.all;
use work.TMS320C203_Tim.all;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
architecture BEHAVIOR_UART of UART is
  signal RX_CLK,TX_CLK,START_BIT: BIT:='0';
  signal AXSR: STD_Logic_Vector(15 downto 0);
  signal ARSR: STD_Logic_Vector(15 downto 0);
  signal BRD_TX_DIV,BRD_RX_DIV,LOAD_DATA: STD_LOGIC;
  signal BRD_RX_START: BOOLEAN := FALSE; -- start receive clock
  signal BRD_TX_STOP: BOOLEAN := FALSE; --
  signal IOSR_READ: STD_LOGIC_VECTOR(15 downto 0):="0000000000000000";
  signal TXRXINT_TX: STD_LOGIC :='0';
  signal TXRXINT_RX: STD_LOGIC :='0';
  begin
-------------------------------------------------------------------------
-- Process UART_TX_DIV divides with BRD *16 the signal OUTCLOCK1
-- This process is used for generate transmision boud rate
-- Divisor START when BRD_TX_START =TRUE and STOP when BRD_TX_STOP=FALSE
-- Input signal: LOAD,CLKOUT1,BRD_TX_STOP,Soft_Reset
```

```
-- Output signal for another process: BRD_TX_DIV (clock for transmision)
-- Observation: Soft_Reset=1 stop transmision (used for RESET)
-------------------------------------------------------------------------
  UART_TX_DIV: process(LOAD,CLKOUT1,BRD_TX_STOP)
  variable BRD_prescaler: INTEGER :=8;  -- temporary variable
  variable BRD_counter: INTEGER :=0;    --
  variable clktmp: std_LOGIC ;     --
  variable BRD_TX_START: BOOLEAN := FALSE; -- start transmision
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  begin
    if Soft_Reset ='1' then
       BRD_TX_START :=FALSE; -- stop TRANSMISION
    end if;
    if not BRD_TX_STOP'STABLE and BRD_TX_STOP =TRUE then
       BRD_TX_START :=FALSE; -- stop clock for transmision
    end if;
    if BRD_TX_START = TRUE and not CLKOUT1'STABLE and CLKOUT1 ='0' then
       BRD_counter := BRD_counter -1;
       if BRD_counter = 0 then
          BRD_counter := BRD_prescaler;  --
          clktmp := not clktmp;
          BRD_TX_DIV <= clktmp;
       end if;
    end if;
    if not LOAD'STABLE and LOAD ='1' then
       BRD_prescaler := 8* To_Logic_Integer(BRD); -- signal LOAD start divisor
       BRD_counter := BRD_prescaler;  -- load prescaled
       clktmp :='0' ;
       BRD_TX_START :=TRUE;
       BRD_TX_DIV <= clktmp;
    end if;
  end process UART_TX_DIV;
-------------------------------------------------------------------------
-- Process UART_RX_DIV  divides by BRD *16  the signal OUTCLOCK1
-- This process is used for generate reception boud rate
-- Divisor START when BRD_RX_START =TRUE and stop when BRD_RX_START =FALSE;
-- Input signal: BRD_RX_START,CLKOUT1
-- Output signal for another process: BRD_RX_DIV (clock for transmision)
-------------------------------------------------------------------------
  UART_RX_DIV: process(BRD_RX_START,CLKOUT1)
  variable BRD_prescaler: INTEGER :=8;
  variable BRD_counter: INTEGER :=0;
  variable clktmp: std_LOGIC ;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  begin
    if not BRD_RX_START'STABLE and BRD_RX_START =TRUE then
       BRD_prescaler := 8* To_Logic_Integer(BRD);-- load prescaler for
       BRD_counter := BRD_prescaler;              -- boud rates
       clktmp :='0' ;
       BRD_RX_DIV <= clktmp;
    end if;
    if BRD_RX_START =TRUE and not CLKOUT1'STABLE and CLKOUT1 ='0' then
       BRD_counter := BRD_counter -1;
       if BRD_counter = 0 then
          BRD_counter := BRD_prescaler;
          clktmp := not clktmp;
          BRD_RX_DIV <= clktmp;
       end if;
    end if;
  end process UART_RX_DIV;

-------------------------------------------------------------------------
-- Process UART_TX is used for transmision --
-- Input signal: BRD_TX_DIV,LOAD,BRD_TX_STOP,AXSR
-- Output signal:TX,TXRXINT_TX,BRD_TX_STOP
-- Observation: Soft_Reset=1 stop transmision (used for RESET)
-------------------------------------------------------------------------
  UART_TX: process(BRD_TX_DIV,LOAD)
  variable TX_COUNTER: INTEGER :=-1;
  variable TX_STOP: BOOLEAN :=FALSE;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  begin
    if Soft_Reset ='1' then
       BRD_TX_STOP <=TRUE; -- stop TRANSMISION
```

```
      TX <='1'; -- if RESET is active
    end if;
    if not LOAD'STABLE and LOAD ='1' then  -- condition for START transmision
       AXSR(15 downto 8) <="00000000";
       AXSR(7 downto 0) <= ADTR(7 downto 0);
       TX_COUNTER :=-1;
       TXRXINT_TX <= '0';
       TX_STOP := FALSE;
       BRD_TX_STOP <=FALSE;
    end if;
    if BRD_TX_DIV'EVENT and BRD_TX_DIV ='0' and TX_STOP =FALSE then
       if TX_COUNTER =-1 then
          TX <='0';  -- start bit
       end if;
       if TX_COUNTER < 8 and TX_COUNTER > -1 then
          TX <= AXSR(TX_COUNTER); -- shift data to TX pin
       end if;
       if TX_COUNTER =8 then
          TX_STOP :=TRUE;
          BRD_TX_STOP <=TRUE;
          TX <='1';  -- stop bit
          TXRXINT_TX <='1';
       end if;
       TX_COUNTER :=TX_COUNTER +1;
    end if;
  end process UART_TX;

--------------------------------------------------------------------------
-- Receiving of the serial input line is initiated by the first
-- negative transition of RX (starting bit). After a half a clock
-- period of signal BRD_RX_DIV sample RX again.
-- If is still a logic 0 , then sampling process continues. Note
-- that the sampling is in the middle of the input clock period,
-- which is the most reliable place to sample.
-- Input signal: RX,BRD_RX_DIV,Soft_Reset
-- Output signal:TXRXINT_RX, ARSR= contain receive data
--------------------------------------------------------------------------
  UART_INPUT: process(RX,BRD_RX_DIV,Soft_Reset)
    variable RX_FLAG:BOOLEAN :=TRUE;
    variable RX_COUNTER: INTEGER :=-1;
    variable str4 : string (1 to 4);
    variable Good : Boolean;
--------------------------------------------------------------------------
--------------------------------------------------------------------------
    begin
    if Soft_Reset ='1' then
       ARSR(15 downto 0) <="0000000000000000";
       RX_FLAG :=TRUE;
    end if;
-- Search START bit
      if RX_FLAG then
         if not RX'STABLE and RX='0' then
            BRD_RX_START <=TRUE;
            RX_COUNTER := -1;
            ARSR(15 downto 8) <="00000000";
            TXRXINT_RX <='0';
         end if;
      end if;
-- if START bit is valid begin reception
      if BRD_RX_START = TRUE and not BRD_RX_DIV'STABLE and BRD_RX_DIV ='1' then
         if RX_COUNTER = -1 then  -- jump start bit
            RX_FLAG :=FALSE;
         end if;
         if RX_COUNTER < 8 and RX_COUNTER > -1 then
            ARSR(RX_COUNTER)<=RX;  -- read RX in ARSR register
         end if;
         if RX_COUNTER =8 then
            if not RX ='1' then
               report "STOP bit not found for UART" ;
            end if;
            BRD_RX_START <=FALSE;  -- stop bit
            RX_FLAG :=TRUE;
            -- Read the Rx register
            Vec2Hex (ARSR,str4,Good);
            report "UART Receive Data! : "&str4 &""
            severity Note;
            TXRXINT_RX <='1';
```

```
        end if;
        RX_COUNTER :=RX_COUNTER +1;
        TXRXINT_RX <='0';
      end if;
 end process UART_INPUT;

-------------------------------------------------------------------------
-- Process UART_READ is used to read the receive data in ADTR register --
-- Input signal: READ,ARSR
-- Output signal:ADTR = contain receive data
-------------------------------------------------------------------------
  UART_READ:process (READ)
  begin
      if not READ'STABLE and READ ='1' and BRD_RX_START =FALSE then
         ADTR <=ARSR;
    --   TXRXINT_RX <='0';
    --   READ <='0';
       else
         ADTR(7 downto 0) <="ZZZZZZZZ";
       end if;
  end process UART_READ;

-------------------------------------------------------------------------
-- Process is used for make TXRXINT signal
-- Input signal:TXRXINT_TX,TXRXINT_RX
-- Output signal:TXRXINT
-------------------------------------------------------------------------
 UART_TXRXINT:process (TXRXINT_TX,TXRXINT_RX)
    begin
     TXRXINT <= TXRXINT_TX or TXRXINT_RX;
    end process UART_TXRXINT;

-------------------------------------------------------------------------
-- Procedure UART_ASPCR is used to program the general propose I/O pins
-- General propose I/O pins are set as input or output by writing
-- in the Asynchronous Serial Port Control Register (ASPCR)
--  ASPCR  bit0 =1     => IO0 is configured as an output, else input
--         bit1 =1     => IO1 is configured as an output, else input
--         bit2 =1     => IO2 is configured as an output, else input
--         bit3 =1     => IO3 is configured as an output, else input
-- The logic level of general propose I/O pins programmed as output is
-- determined by bit3,bit2,bit1,bit0 in the I/O status register.
-- Input signal:ASPCR,IOSR
-- Output signal:IO0,IO1,IO2,IO3
-------------------------------------------------------------------------
  UART_ASPCR: process(ASPCR,Soft_Reset)
    begin
      if ASPCR(3) ='1' then
         IO3 <=IOSR(3);
      else
         IO3 <='Z';
      end if;
      if ASPCR(2) ='1' then
         IO2 <=IOSR(2);
      else
         IO2 <='Z';
      end if;
      if ASPCR(1) ='1' then
         IO1 <=IOSR(1);
      else
         IO1 <='Z';
      end if;
      if ASPCR(0) ='1' then
         IO0 <=IOSR(0);
      else
         IO0 <='Z';
      end if;
    end process UART_ASPCR;
-------------------------------------------------------------------------
-- Process UART_IOSR read I/O Status Register if input logic
-- level of general propose I/O pins was changed.
-- Input signal:IO0,IO1,IO2,IO3,ASPCR
-- Output signal:IOSR_READ =input data and status of IO0,IO1,IO2,IO3
-------------------------------------------------------------------------
  UART_IOSR: process(IO0,IO1,IO2,IO3)
  begin
      if ASPCR(3) ='0' then
```

```
        if not IO3'STABLE then
           IOSR_READ(7) <='1';
           IOSR_READ(3) <=IO3;
        end if;
      end if;
      if ASPCR(2) ='0' then
        if not IO2'STABLE then
           IOSR_READ(6) <='1';
           IOSR_READ(2) <=IO2;
        end if;
      end if;
      if ASPCR(1) ='0' then
        if not IO1'STABLE then
           IOSR_READ(5) <='1';
           IOSR_READ(1) <=IO1;
        end if;
      end if;
      if ASPCR(0) ='0' then
        if not IO0'STABLE then
           IOSR_READ(4) <='1';
           IOSR_READ(0) <=IO0;
        end if;
      end if;
  end process UART_IOSR;
end BEHAVIOR_UART;
-------------------------------------------------------------------------
-- Entity for the Synchronous Serial Port
-------------------------------------------------------------------------
library ieee,work;
use ieee.Std_Logic_1164.all;
use work.Simulation.all;
use work.TMS320C203_Def.all;          -- For custom functions
use work.TMS320C203_Tim.all;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
entity SSP is
  generic (
    tperiod_CI       : Time := Tables_tperiod_CI(TMS320C203_50_5V)(Nominal);       -- tc(CI)
    tperiod_SCK      : Time := Tables_tperiod_SCK(TMS320C203_50_5V)(Nominal);      -- tc(SCK)
    tdelay_CLKX_DX   : Time := Tables_tdelay_CLKX_DX(TMS320C203_50_5V)(Nominal);-- td(DX)
    tdisable_CLKX_DX : Time := Tables_tdisable_CLKX_DX(TMS320C203_50_5V)(Nominal);--tdis(DX)
    tdelay_FSX_CLKX  : Time := Tables_tdelay_FSX_CLKX(TMS320C203_50_5V)(Nominal);--td(FS)
    thold_FSX_CLKX   : Time := Tables_thold_FSX_CLKX(TMS320C203_50_5V)(Nominal));--h(FS)H
  port (
    DX          : out   std_LOGIC;
    FSX         : inout std_LOGIC;
    XINT        : out   std_LOGIC;
    XData_In    : in    std_logic_vector(15 downto 0);
    WR2SSP_EN   : in    std_logic;
    RD_SSP_EN   : in    std_logic;
    SSPCR       : in    std_logic_vector(15 downto 0);
    DR          : in    std_LOGIC;
    FSR         : in    std_LOGIC;
    RINT        : out   std_LOGIC;
    RData_Out   : out   std_logic_vector(15 downto 0);
    CLKOUT1     : in    Std_LOGIC;
    CLKR        : in    Std_LOGIC;
    CLKX        : inout Std_LOGIC;
    Soft_Reset  : in    Std_Logic);
end SSP;
-------------------------------------------------------------------------
-- Architecture for Synchronous Serial Port
-------------------------------------------------------------------------
architecture BEHAVIOR of SSP is
-------------------------------------------------------------------------
-- Global variables
-------------------------------------------------------------------------
shared variable wr_pointer,Rwr_pointer  : integer := 1;
shared variable rd_pointer,Rrd_pointer  : integer := 0;
shared variable No_Of_Words : Std_Logic_Vector(2 downto 0) := "000";
shared variable XSR,RSR  : Std_Logic_Vector(15 downto 0) := "0000000000000000";
shared variable FR : std_logic_vector(1 downto 0);
-------------------------------------------------------------------------
-- Internal signals
-------------------------------------------------------------------------
signal TCOMP,StartXCont,LSB,OVF,DXOUT_EN  : std_logic := '0';
signal MCM,TXM,RRST : std_logic := '1';
```

```
signal FREE,SOFT,XRST,FSM,DLB : std_logic;
signal FT : std_logic_vector(1 downto 0);
signal R_EN,EndRword : std_logic := '0';
-------------------------------------------------------------------------
-------------------------------------------------------------------------
begin
 ConfigX : process(CLKOUT1)
-------------------------------------------------------------------------
-- The process implements the Synchronous Serial Port Control Register
-- (SSPCR).
-- Input signal : CLKOUT1
-- Output signals :FREE,SOFT,FT,RRST,TXM,MCM,FSM,DLB
-------------------------------------------------------------------------
    begin
      if (not CLKOUT1'STABLE and CLKOUT1 = '0') then
         FREE <= SSPCR(15) after 2 ns;
         SOFT <= SSPCR(14) after 2 ns;
         FT(1 downto 0) <= SSPCR(11 downto 10) after 2 ns;
         TXM  <= SSPCR(3) after 2 ns;
         MCM  <= SSPCR(2) after 2 ns;
         FSM  <= SSPCR(1) after 2 ns;
         DLB  <= SSPCR(0) after 2 ns;
      end if;
 end process ConfigX;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 ResetX : process(Soft_Reset,CLKOUT1)
-------------------------------------------------------------------------
-- The process implements the reset of the port accordingly to the
-- processor reset
-- Input signals : Soft_Reset, CLKOUT1
-- Output signals : XRST -- The bit 5 of the SSPCR is used to reset the
--                          Transmitter portion of the port
--                  RRST -- The bit 4 of the SSPCR is used to reset the
--                          Receiver portion of the port
-------------------------------------------------------------------------
  begin
    if Soft_Reset = '1' then
       XRST <= '0';
       RRST <= '0';
    else
     if (not CLKOUT1'STABLE and CLKOUT1 = '0') then
       XRST <= SSPCR(5) after 2 ns;
       RRST <= SSPCR(4) after 2 ns;
     end if;
    end if;
 end process ResetX;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 CLOCKX: process(CLKOUT1)
-------------------------------------------------------------------------
-- The process generates the internal transmit clock(CLKX), used in the
-- internal clock transmit mode
-- Input signals : CLKOUT1
--                 MCM -- The bit 2 of the SSPCR is used to select the source
--                     -- for the transmit clock (internal or external)
-- Output signal : CLKX -- The Internal generated transmit clock
-------------------------------------------------------------------------
    variable CHNGX : boolean := True;
    variable clkval: Std_ULogic := '0';
 begin
   if MCM = '1' then
      if (not CLKOUT1'STABLE and CLKOUT1 = '1') then
         clkval := not clkval;
         CLKX <= clkval;
      end if;
   else
      null;
   end if;
 end process CLOCKX;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 Xrdwrfifo     : process(WR2SSP_EN,CLKX,FSX,LSB)
-------------------------------------------------------------------------
-- The process implements the Transmit 4 Words FIFO Memory of the SSP
-- Input signals :  WR2SSP_EN  -- The signal is set active to '1' when the
--                              processor writes one word to the port.
```

```
--                  CLKX        -- The transmit clock
--                  FSX         -- The Frame Sync Signal for transmission
--                  LSB         -- The Signal is active to '1' everytime the
--                                 Least Significant Bite is sent out
-- Output signals : TCOMP       -- The bit 13 of the SSPCR. The bit is cleared
--                                 to '0' when all data in the transmit FIFO
--                                 buffer has been transmitted and is set to '1'
--                                 when new data is written to the transmit FIFO
--                                 buffer
--                  StartXCont -- The signal is set to '1' for one clockX period
--                                 whenever to the transmit FIFO buffer is written
--                                 one word and the FIFO buffer was empty before.
--                  XINT        -- The transmit interrupt to the processor. It
--                                 signals the processor that the FIFO buffer can
--                                 accept words.
-------------------------------------------------------------------------
    variable FIFO_Data : SSP_Data;
 begin
    if Soft_Reset = '1' then
       TCOMP <= '0';
       wr_pointer := 1;
    end if;
    if ((not CLKX'STABLE and CLKX = '0') and WR2SSP_EN = '1') then
       case wr_pointer is
            when 0 =>   TCOMP <= '0'; -- FIFO is empty
                        No_Of_Words := "000";
            when 1 =>   FIFO_Data(1) := XData_In;
                        if TCOMP = '0' then
                           XSR(15 downto 0) := FIFO_Data(1);
                        end if;
                        wr_pointer := wr_pointer + 1;
                        rd_pointer := 1;
                        TCOMP <= '1';
                        No_Of_Words := "001";
                        if (TCOMP = '0') then
                            StartXCont <= '1';
                        end if;
            when 2 =>   FIFO_Data(2) := XData_In;
                        wr_pointer := wr_pointer + 1;
                        No_Of_Words := "010";
            when 3 =>   FIFO_Data(3) := XData_In;
                        wr_pointer := wr_pointer + 1;
                        No_Of_Words := "011";
            when 4 =>   FIFO_Data(4) := XData_In;
                        wr_pointer := wr_pointer + 1;
                        No_Of_Words := "100";
            when others => null;
       end case;
    end if;
    if ((not CLKX'STABLE and CLKX = '0') and StartXCont = '1') then
         StartXCont <= '0';
    end if;
    if (((not CLKX'STABLE and CLKX = '0') and rd_pointer > 0) and (FSX = '1' or (LSB = '1' and FSM =
'0'))) then
          TCOMP <= '1';
          case wr_pointer is
               when 5 =>  XSR(15 downto 0) := FIFO_Data(1);
                          FIFO_Data(1) := FIFO_Data(2);
                          FIFO_Data(2) := FIFO_Data(3);
                          FIFO_Data(3) := FIFO_Data(4);
                          wr_pointer   := wr_pointer - 1;
               when 4 =>  XSR(15 downto 0) := FIFO_Data(1);
                          FIFO_Data(1) := FIFO_Data(2);
                          FIFO_Data(2) := FIFO_Data(3);
                          wr_pointer   := wr_pointer - 1;
               when 3 =>  XSR(15 downto 0) := FIFO_Data(1);
                          FIFO_Data(1) := FIFO_Data(2);
                          wr_pointer   := wr_pointer - 1;
               when 2 =>  XSR(15 downto 0) := FIFO_Data(1);
                          wr_pointer:= wr_pointer - 1;
                          rd_pointer:= 0;
               when others => null;
          end case;
          case FT(1 downto 0) is
               when "00" =>  if wr_pointer < 5 then
                                XINT <= '1';
                                XINT <= '0' after tperiod_SCK;
```

```
                        end if;
              when "01" =>  if wr_pointer < 4 then
                              XINT <= '1';
                              XINT <= '0' after tperiod_SCK;
                            end if;
              when "10" =>  if wr_pointer > 3 then
                              XINT <= '1';
                              XINT <= '0' after tperiod_SCK;
                            end if;
              when "11" =>  if wr_pointer < 2 then
                              XINT <= '1';
                              XINT <= '0' after tperiod_SCK;
                            end if;
              when others => XINT <= '0';
          end case;
    end if;
    if (not LSB'STABLE and LSB = '1' and rd_pointer = 0) then
        TCOMP <= '0' after tdelay_CLKX_DX;
    end if;
 end process Xrdwrfifo;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 DX_OUT_Enable : process(CLKX,FSX)
-------------------------------------------------------------------------
-- The process generates the DXOUT_EN signal used to enable the output
-- of the data in the XSR (Transmit Shift Register) on the DX pin
-- Input signals : FSX      -- The frame sync pulse for transmission
--                 CLKX     -- The transmit clock
-- Output signal : DXOUT_EN -- Used by the OUT_Data process to enable
--                             the transmission
-------------------------------------------------------------------------
    variable str4 : string(1 to 4);
    variable Good : Boolean;
 begin
    if FSM = '1' then
       if ((not CLKX'STABLE and CLKX = '0') and FSX = '1' and TCOMP = '1') then
          DXOUT_EN <= '1';
          -- Read the SS register
          Vec2Hex (XSR,str4,Good);
          report "SSP Transmit Data! : "&str4 &""
          severity Note;
       end if;
    else
       if FSM ='0' then
        if ((not CLKX'STABLE and CLKX = '0') and (FSX = '1' or LSB ='1') and TCOMP = '1') then
            DXOUT_EN <= '1';
            -- Read the SS register
            Vec2Hex (XSR,str4,Good);
            report "SSP Transmit Data! : "&str4 &""
            severity Note;
        end if;
       end if;
    end if;
    if ((not CLKX'STABLE and CLKX = '0') and TCOMP = '0') then
        DXOUT_EN <= '0';
    end if;
 end process DX_OUT_Enable;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 FSX_GEN: process(CLKX,StartXCont,LSB)
-------------------------------------------------------------------------
-- The process generates the transmission frame sync signal for the
-- transmission in concordance with the TXM (bit 3) and FSM (bit 1) bits
-- of the SSPCR
-- Input signals  : CLKX       -- The transmit clock
--                  StartXCont -- The rising edge of this signal is used to initiate
--                                the output data on the DX pin
--                  LSB        -- The rising edge of this signal is used to initiate
--                                a new word transmission as long as the transmit
--                                FIFO buffer is not empty at the moment
-- Output signals : FSX        -- The transmit internal generated frame sync signal
-------------------------------------------------------------------------
    begin
       if TXM = '1' then -- Internal Frame sync is selected
         case FSM is
              when '0' => -- Continuous Mode Transmission is selected
                          if ((not CLKX'STABLE and CLKX = '1') and StartXCont = '1') then
```

```
                              FSX <= '1' after tdelay_FSX_CLKX;
                         end if;
               when '1' => -- Burst Mode Transmission is selected
                         if ((((not CLKX'STABLE and CLKX = '1') and StartXCont = '1') or
                         (not LSB'STABLE and LSB = '1')) and wr_pointer > 1) then
                              FSX <= '1' after tdelay_FSX_CLKX;
                         end if;
               when others => FSX <= '0';
          end case;
          if ((not CLKX'STABLE and CLKX = '1') and FSX = '1') then
               FSX <= '0' after thold_FSX_CLKX;
          end if;
        else  -- External Frame sync is selected
            null;
        end if;
 end process FSX_GEN;
-- The following process send the bits out of the TMS via pin DX
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 Out_Data : process(DXOUT_EN,CLKX)
-------------------------------------------------------------------------
-- The process implements the output stage of the SSP
-- Input signals    : DXOUT_EN  -- As long as the signal is active '1' the
--                                 data from the transmit shift register (XSR)
--                                 is shifted out to the DX pin. Otherwise the
--                                 DX pin is driven to high impedance
--                    CLKX       -- The transmit clock
-- Output signals   : LSB        -- The signal is active to '1' during the driving
--                                 on the DX pin of the least significant bit of
--                                 the word from XSR. The signal is used in the
--                                 FSX_Gen process to generate the frame sync pulse
--                                 in the Burst Mode Transmission with Internal Frame
--                                 Sync(FSM = 1, TXM = 1)
--                    DX         -- The transmission output pin
-------------------------------------------------------------------------
     variable i : integer := 15;
 begin
     if XRST = '1' then
       if (DXOUT_EN = '1' and (not CLKX'STABLE and CLKX = '1')) then
           LSB <= '0';
           if i /= -1 then
              DX <= XSR(i) after tdelay_CLKX_DX;
              if i = 0 then
                 LSB <= '1';
              end if;
              i:= i-1;
              if (DXOUT_EN = '1' and i = -1) then
                  i:=15;
              end if;
           end if;
        else
           if ((DXOUT_EN = '0' and LSB = '1') and (not CLKX'STABLE and CLKX = '1')) then
               LSB <= '0';
               DX <= 'Z' after tdisable_CLKX_DX;
           end if;
        end if;
     else
       DX <= 'Z';
     end if;
 end process Out_Data;
-- ========================================================================--
-- The following processes implements the receiver section of the synchronous serial port    --
-- ========================================================================-
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 R_EN_GEN : process(CLKR,FSR,EndRword)
 begin
   if FSM = '1' then -- Burst Mode is selected
     if ((not CLKR'STABLE and CLKR = '0') and FSR = '1') then
           R_EN <= '1';
     end if;
     if ((not CLKR'STABLE and CLKR = '0') and (RRST ='0' or OVF = '1' or EndRword = '1')) then
         R_EN <= '0';
     end if;
   else
    if FSM = '0' then -- Continuous Mode is selected
     if ((not CLKR'STABLE and CLKR = '0') and (RRST = '1' and FSR = '1')) then
```

```
        R_EN <= '1';
     end if;
     if ((not CLKR'STABLE and CLKR = '0') and RRST = '0') then
        R_EN <= '0';
     end if;
    end if;
   end if;
 end process R_EN_GEN;
--------------------------------------------------------------------------
--------------------------------------------------------------------------
 In_Data : process(CLKR,R_EN)
    variable i : integer := 15;
    variable str4 : string(1 to 4);
    variable Good : Boolean;
 begin
    if ((not CLKR'STABLE and CLKR = '0') and R_EN = '1') then
       EndRword <= '0' after tdisable_CLKX_DX;
       if EndRword = '0' then
          RSR(i) := DR;
          if i = 0 then
             EndRword <= '1';
             -- Read the SS register
             --Vec2Hex (RSR,str4,Good);
             --report "SSP Receive Data! : "&str4 &""
             --severity Note;
          end if;
          i:= i-1;
          if (R_EN = '1' and i = -1) then
              i:=15;
          end if;
       end if;
    end if;
 end process In_Data;
--------------------------------------------------------------------------
--------------------------------------------------------------------------
 Rrdwrfifo: process(RD_SSP_EN,CLKR,FSR,EndRword)
       variable FIFO_Data: SSP_Data;
 begin
   if ((not CLKR'STABLE and CLKR = '0') and EndRword = '1') then
        case Rwr_pointer is
             when 1 => FIFO_Data(1) := RSR;
                       Rwr_pointer := Rwr_pointer + 1;
                       Rrd_pointer := 1;
             when 2 => FIFO_Data(2) := RSR;
                       Rwr_pointer := Rwr_pointer + 1;
             when 3 => FIFO_Data(3) := RSR;
                       Rwr_pointer := Rwr_pointer + 1;
             when 4 => FIFO_Data(4) := RSR;
                       Rwr_pointer := Rwr_pointer + 1;
                       OVF <= '1';
             when others => null;
        end case;
   end if;
   if ((not CLKR'STABLE and CLKR = '1') and Rrd_pointer > 0 and RD_SSP_EN = '1') then
        case wr_pointer is
             when 5 => RData_Out(15 downto 0) <= FIFO_Data(1);
                       FIFO_Data(1) := FIFO_Data(2);
                       FIFO_Data(2) := FIFO_Data(3);
                       FIFO_Data(3) := FIFO_Data(4);
                       Rwr_pointer  := Rwr_pointer - 1;
                       OVF <= '0';
             when 4 => RData_Out(15 downto 0) <= FIFO_Data(1);
                       FIFO_Data(1) := FIFO_Data(2);
                       FIFO_Data(2) := FIFO_Data(3);
                       Rwr_pointer  := Rwr_pointer - 1;
             when 3 => RData_Out(15 downto 0) <= FIFO_Data(1);
                       FIFO_Data(1) := FIFO_Data(2);
                       Rwr_pointer  := Rwr_pointer - 1;
             when 2 => RData_Out(15 downto 0) <= FIFO_Data(1);
                       Rwr_pointer := Rwr_pointer - 1;
                       Rrd_pointer := 0;
             when others => null;
        end case;
        case FR(1 downto 0) is
             when "00" => if Rwr_pointer > 1 then
                             RINT <= '1';
                             RINT <= '0' after tperiod_SCK;
```

```
                    end if;
            when "01" => if Rwr_pointer > 2 then
                            RINT <= '1';
                            RINT <= '0' after tperiod_SCK;
                         end if;
            when "10" => if Rwr_pointer > 3 then
                            RINT <= '1';
                            RINT <= '0' after tperiod_SCK;
                         end if;
            when "11" => if wr_pointer > 4 then
                            RINT <= '1';
                            RINT <= '0' after tperiod_SCK;
                         end if;
            when others => RINT <= '0';
        end case;
   end if;
 end process Rrdwrfifo;
end BEHAVIOR;
-------------------------------------------------------------------------
-- Entity for PROCESSOR                                                 -
-------------------------------------------------------------------------
library std;
use std.textio.all;
library IEEE,work;
use IEEE.Std_Logic_1164.all;
use work.TMS320C203_Def.all;          -- For custom functions
use work.TMS320C203_Tim.all;
use work.Simulation.all;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
entity TMS320 is
   generic (
        SimCondition    : SimCondType := Nominal;
    Processor   : ProcessorType := TMS320C203_50_5V);
    port(
        DataB           : inout Std_LOGIC_Vector(15 downto 0);        -- Data bus
        AddrB           : inout       Std_LOGIC_Vector(15 downto 0);        -- Address bus
        -- Memory control signals (8)
        PS_N            : out         Std_Logic;                            -- Program select
        DS_N            : out         Std_Logic;                            -- Data select
        RW_N            : out         Std_Logic;                            -- Read/write
        IS_N            : out         Std_Logic;                            -- I/O space select
        READY           : in          Std_Logic;                            -- Data ready input
        RD_N            : out         Std_Logic;                            -- Read select
        WE_N            : out         Std_Logic;                            -- Write enable
        STRB_N  : out           Std_Logic;                          -- Strobe
        -- Multi-processing signals (6)
        BR_N            : out         Std_Logic;                            -- Bus request
        HOLDA_N : out           Std_Logic;                          -- Hold acknowledge
        IO0             : inout Std_Logic;                          -- Software controlled I/O
        IO1             : inout Std_Logic;                          -- Software controlled I/O
        IO2             : inout Std_Logic;                          -- Software controlled I/O
        IO3             : inout Std_Logic;                          -- Software controlled I/O
        -- Initialization, Interrupts and Reset signals (5)
        RS_N            : in          Std_Logic;                            -- Hardware reset
        NMI_N           : in          Std_Logic;                            -- Nonmaskable interrupt
        INT1_N  : in            Std_Logic;                          -- Interrupt 1 /HOLD_N request
        INT2_N  : in            Std_Logic;                          -- Interrupt 2
        INT3_N  : in            Std_Logic;                          -- Interrupt 3
        -- Oscillator, PLL and Timer signals (4)
        CLKOUT1 : inout Std_LOGIC;                          -- Master clock output
        X2              : in          Std_Logic;                            -- Oscillator input
        DIV1            : in          Std_Logic;                            -- Clock mode 1
        DIV2            : in          Std_Logic;                            -- Clock mode 2
        -- Serial port and UART signals (8)
        CLKX            : inout Std_Logic;                          -- Transmit clock
        CLKR            : in          Std_Logic;                            -- Receive clock
        FSR             : in          Std_Logic;                            --   Frame   synchro   /
receive
        FSX             : inout Std_Logic;                          -- Frame synchro / transmit
        DR              : in          Std_Logic;                            -- Serial data receive
        DX              : inout Std_Logic;                          -- Serial port transmit
        TX              : out         Std_Logic;                            --  Asynchronous  transmit
pin
        RX              : in          Std_Logic);                           --  Asynchronous  receive
pin
end TMS320;--================== End of entity ==================--
```

```
-------------------------------------------------------------------------
-- Architecture for TMS320C203                                          -
-------------------------------------------------------------------------
library std;
use std.textio.all;
library IEEE,work;
use IEEE.Std_Logic_1164.all;
use work.TMS320C203_Def.all;          -- For custom functions
use work.TMS320C203_Tim.all;
use work.Simulation.all;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
architecture STRUCTURE of TMS320 is
-----------------------------------------------
-- Clk and ResetInterrupts signals            --
-----------------------------------------------
signal CLKOUT1_ENABLE: Std_Logic; -- used for stop clock after RESET
signal STOP : STD_LOGIC;          -- Read UART receive data
signal Soft_Reset: Std_Logic;
-----------------------------------------------
-- Asynchronous Serial Port signals           --
-----------------------------------------------
signal ADTR: STD_LOGIC_VECTOR(15 downto 0);
signal ASPCR: STD_LOGIC_VECTOR(15 downto 0); -- Asynchronous serial port control register
signal IOSR: STD_LOGIC_VECTOR(15 downto 0);  -- Asynchronous I/O status register
signal BRD:  STD_LOGIC_VECTOR(15 downto 0);  -- Asynchronous boud-rate devisor register
signal TXRXINT: std_logic;                   -- Asynchronous hardware interrupt
signal LOAD: STD_LOGIC :='0';                -- Start UART transmision
signal READ: STD_LOGIC :='0';                -- Read UART receive data
signal WAIT1: Std_Logic;
-----------------------------------------------
-- Synchronous Serial Port signals            --
-----------------------------------------------
signal XINT,RINT,WR2SSP_EN,RD_SSP_EN,loopback_SSP  : std_logic :='0';
signal XData_In,RData_Out: Std_Logic_vector(15 downto 0);
signal SSPCR: Std_Logic_vector(15 downto 0) := "1000111100001110";
-----------------------------------------------
signal VDataB: Std_Logic_Vector(15 downto 0);
signal VAddrB:        Std_Logic_Vector(15 downto 0);-- Address bus
signal VPS_N:  Std_Logic;                           -- Program select
signal VDS_N:  Std_Logic;                           -- Data select
signal VRW_N:   Std_Logic;                          -- Read/write
signal VIS_N:   Std_Logic;                          -- I/O space select
signal VREADY:  Std_Logic;                          -- Data ready input
signal VRD_N:   Std_Logic;                          -- Read select
signal VWE_N:   Std_Logic;                          -- Write enable
signal VSTRB_N:       Std_Logic;                          -- Strobe
--signal VHOLDA_N:   Std_Logic;
signal VBR_N:   Std_Logic;
signal Busy_cycle: Std_Logic; -- Busy_cycle='1': when instriction not finish
signal Wait_nr_cycle:  Std_Logic_Vector(15 downto 0);
-------------------------------------------------------------------------
-- Component Clk                                                        -
-------------------------------------------------------------------------
component Clk
    generic(
        tperiod_CI        : Time := Tables_tperiod_CI(TMS320C203_50_5V)(Nominal);     -- tc(CI)
        tpw_CI_posedge : Time := Tables_tpw_CI_posedge(TMS320C203_50_5V)(Nominal); -- tw(CIH)
        tpw_CI_negedge : Time := Tables_tpw_CI_negedge(TMS320C203_50_5V)(Nominal); -- tw(CIL)
        tperiod_CO        : Time := Tables_tperiod_CO(TMS320C203_50_5V)(Nominal);     -- tc(CO)
        tpw_CO_posedge : Time := Tables_tpw_CO_posedge(TMS320C203_50_5V)(Nominal); -- tw(COH)
        tpw_CO_negedge : Time := Tables_tpw_CO_negedge(TMS320C203_50_5V)(Nominal); -- tw(COL)
        tdelay_CIH_CO  : Time := Tables_tdelay_CIH_CO(TMS320C203_50_5V)(Nominal)); -- td(CIH-CO)
    port (
        -- Oscillator and PLL signals (4)
        CLKOUT1 : inout Std_Logic;                          -- Master clock output
        X2              : in          Std_Logic;                    -- Oscillator input
        DIV1            : in          Std_Logic;                    -- Clock mode 1
        DIV2            : in          Std_Logic;                    -- Clock mode 2
        -- Internal signal
        CLKOUT1_ENABLE : in    Std_Logic);                  -- Internal signal
end component;
-------------------------------------------------------------------------
-- Component RstInt                                                     -
-------------------------------------------------------------------------
component RstInt
        generic (
```

```
        tperiod_CI              : Time := Tables_tperiod_CI(TMS320C203_50_5V)(Nominal);      -- tc(CI)
        -- RS_N,INTx_N,NMI_N,BIO_N,TOUT and XF timing parameters
        tsetup_RS_CI    : Time := Tables_tsetup_RS_CI(TMS320C203_50_5V)(Nominal);    -- tsu(RS)CI
        tsetup_RS_CO    : Time := Tables_tsetup_RS_CO(TMS320C203_50_5V)(Nominal);    -- tsu(RS)CO
        tpw_RS_negedge : Time := Tables_tpw_RS_negedge(TMS320C203_50_5V)(Nominal); -- tw(RSL)
        tdelay_RSN_fetch        : Time := Tables_tdelay_RSN_fetch(TMS320C203_50_5V)(Nominal);-- td(EX)
        tpw_INTx_negedge        :   Time   :=   Tables_tpw_INTx_negedge(TMS320C203_50_5V)(Nominal);   --
tw(IN)
        tdelay_INTx_fetch          Time   :=   Tables_tdelay_INTx_fetch(TMS320C203_50_5V)(Nominal);   --
td(IN)
        -- External DMA timing parameters
        tdelay_HLDN_HLDAN       :   Time   :=   Tables_tdelay_HLDN_HLDAN(TMS320C203_50_5V)(Nominal);   --
td(H-HA)
        tdelay_HLD_HLDA         : Time := Tables_tdelay_HLD_HLDA(TMS320C203_50_5V)(Nominal); -- td(HH-
HA)
        thighimped_HLDAN        : Time := Tables_thighimped_HLDAN(TMS320C203_50_5V)(Nominal); -- tz(H-
HA)
        tenable_HLDA    : Time := Tables_tenable_HLDA(TMS320C203_50_5V)(Nominal));  -- ten(HA-H)
    port (
        -- Data and Address Buses (32)
        DataB           : inout Std_Logic_Vector(15 downto 0);        -- Data bus
        AddrB           : out           Std_Logic_Vector(15 downto 0);        -- Address bus
        -- Memory control signals (8)
        PS_N            : out           Std_Logic;                            -- Program select
        DS_N            : out           Std_Logic;                            -- Data select
        RW_N            : out           Std_Logic;                            -- Read/write
        IS_N            : out           Std_Logic;                            -- I/O space select
        READY           : in            Std_Logic;                            -- Data ready input
        RD_N            : out           Std_Logic;                            -- Read select
        WE_N            : out           Std_Logic;                            -- Write enable
        STRB_N  : out           Std_Logic;                          -- Strobe
        -- Oscillator and PLL signals (4)
        CLKOUT1 : in            Std_Logic;                          -- Master clock output
        X2              : in            Std_Logic;                            -- Oscillator input
        -- Multi-processing signals (2)
        BR_N            : out           Std_Logic;                            -- Bus request
        HOLDA_N : out           Std_Logic;                          -- Hold acknowledge
        -- Initialization, Interrupts and Reset signals (5)
        RS_N            : in            Std_Logic;                            -- Hardware reset
        NMI_N           : in            Std_Logic;                            -- Nonmaskable interrupt
        INT1_N  : in            Std_Logic;                          -- Interrupt 1 /HOLD_N request
        INT2_N  : in            Std_Logic;                          -- Interrupt 2
        INT3_N  : in            Std_Logic;                          -- Interrupt 3
        -- Internal signals
        RINT            : in            Std_Logic;                            --   Synchronous   serial
port/receive
        XINT            : in            Std_Logic;                            --   Synchronous   serial
port/transmit
        TXRXINT : in            Std_Logic;                          -- Asynchronous serial port
        Stop            : out   Std_Logic;                                  -- Internal signal
        CLKOUT1_ENABLE : out    Std_Logic;                          -- Internal signal
        Soft_Reset : out Std_Logic );
end component;
-------------------------------------------------------------------------
-- Component MemAcc                                                    -
-------------------------------------------------------------------------
component MemAcc
    generic(
    tsetup_A_RDN        : Time := Tables_tsetup_A_RDN(TMS320C203_50_5V)(Nominal);-- tsu(A)RD
        thold_A_RD              : Time := Tables_thold_A_RD(TMS320C203_50_5V)(Nominal);             --
th(A)RD
        tdelay_CON_A    : Time := Tables_tdelay_CON_A(TMS320C203_50_5V)(Nominal);           -- td(COL-
A)
        thold_A_CON             : Time := Tables_thold_A_CON(TMS320C203_50_5V)(Nominal);            --
th(A)CO
        tdelay_CO_RD    : Time := Tables_tdelay_CO_RD(TMS320C203_50_5V)(Nominal);           --   td(CO-
RD)
        tdelay_CO_S             : Time := Tables_tdelay_CO_S(TMS320C203_50_5V)(Nominal);            --
td(COL-S)
        tpw_RD_negedge : Time := Tables_tpw_RD_negedge(TMS320C203_50_5V)(Nominal); -- tw(RDL)
        tpw_RD_posedge : Time := Tables_tpw_RD_posedge(TMS320C203_50_5V)(Nominal); -- tw(RDH)
        tdelay_RD_WEN   : Time := Tables_tdelay_RD_WEN(TMS320C203_50_5V)(Nominal);          -- td(RDW)
        taccess_A               : Time := Tables_taccess_A(TMS320C203_50_5V)(Nominal);              --
ta(A)
        tsetup_D_RD             : Time := Tables_tsetup_D_RD(TMS320C203_50_5V)(Nominal);            --
tsu(D)RD
```

```
        thold_D_RD              : Time := Tables_thold_D_RD(TMS320C203_50_5V)(Nominal);              --
th(D)RD
        thold_D_A               : Time := Tables_thold_D_A(TMS320C203_50_5V)(Nominal);               --
th(D)A
        tsetup_D_CON_R : Time := Tables_tsetup_D_CON_R(TMS320C203_50_5V)(Nominal); -- tsu(DCOL)R
        thold_D_CON_R  : Time := Tables_thold_D_CON_R(TMS320C203_50_5V)(Nominal);          --
th(DCOL)R
        taccess_RD              : Time := Tables_taccess_RD(TMS320C203_50_5V)(Nominal);              --
ta(RD)
-- Write timing parameters
        tsetup_A_WN             : Time := Tables_tsetup_A_WN(TMS320C203_50_5V)(Nominal);             --
tsu(A)W
        thold_A_W               : Time := Tables_thold_A_W(TMS320C203_50_5V)(Nominal);               --
th(A)W
        tsetup_A_CON    : Time := Tables_tsetup_A_CON(TMS320C203_50_5V)(Nominal);           --
tsu(A)CO
        thold_A_CON_W   : Time := Tables_thold_A_CON_W(TMS320C203_50_5V)(Nominal);          --
th(A)COw
--      tpw_NSN             : Time := Tables_tpw_NSN(TMS320C203_50_5V)(Nominal);                    --
tw(NSN)
        tpw_W_negedge   : Time := Tables_tpw_W_negedge(TMS320C203_50_5V)(Nominal);          -- tw(WL)
--      tpw_W_posedge   : Time := Tables_tpw_W_posedge(TMS320C203_50_5V)(Nominal);          -- tw(WH)
        tdelay_CO_WE    : Time := Tables_tdelay_CO_WE(TMS320C203_50_5V)(Nominal);           --  td(CO-
W)
--      tdelay_WE_RDN   : Time := Tables_tdelay_WE_RDN(TMS320C203_50_5V)(Nominal);          -- td(WRD)
        tsetup_D_WE             : Time := Tables_tsetup_D_WE(TMS320C203_50_5V)(Nominal);             --
tsu(D)W
        thold_D_WE              : Time := Tables_thold_D_WE(TMS320C203_50_5V)(Nominal);              --
th(D)W
        tsetup_D_CON_W : Time := Tables_tsetup_D_CON_W(TMS320C203_50_5V)(Nominal); -- tsu(DCOL)W
        thold_D_CON_W  : Time := Tables_thold_D_CON_W(TMS320C203_50_5V)(Nominal));          --
th(DCOL)W
    port (
        -- Data and Address Buses (32)
        DataB           : inout Std_Logic_Vector(15 downto 0);        -- Data bus
        AddrB           : out           Std_Logic_Vector(15 downto 0);        -- Address bus
        -- Memory control signals (8)
        PS_N            : out           Std_Logic;                            -- Program select
        DS_N            : out           Std_Logic;                            -- Data select
        RW_N            : out           Std_Logic;                            -- Read/write
        IS_N            : out           Std_Logic;                            -- I/O space select
        HOLDA_N         : out       Std_Logic;              -- Global Memory
        READY           : in            Std_Logic;                            -- Data ready input
        RD_N            : out           Std_Logic;                            -- Read select
        WE_N            : out           Std_Logic;                            -- Write enable
        STRB_N  : out           Std_Logic;                          -- Strobe
        -- Initialization, Interrupts and Reset signals (1)
        RS_N            : in            Std_Logic;                            -- Hardware reset
        -- Multi-processing signals (5)
        BR_N            : out           Std_Logic;                            -- Bus request
        -- Oscillator signal (1)
        CLKOUT1 : in            Std_Logic;
        VAddrB          : in    Std_Logic_Vector(15 downto 0);
        VDataB          : in    Std_Logic_Vector(15 downto 0);
        VRW_N                   : in    Std_Logic;
        VIS_N                   : in    Std_Logic;
        VDS_N                   : in    Std_Logic;
        VPS_N                   : in    Std_Logic;
        VBR_N                   : in    Std_Logic;
        Busy_cycle  :out Std_Logic;
        Wait_nr_cycle           : in Std_Logic_Vector(15 downto 0));
end component;
-------------------------------------------------------------------------
-- Component UART                                                       -
-------------------------------------------------------------------------
component UART
   port (
     TX : out std_logic;                          -- Asynchronous transmit pin
     RX : in std_logic;                           -- Asynchronous receive pin
     CLKOUT1: in std_logic;                      -- Master clock output
     ADTR      : inout STD_LOGIC_VECTOR(15 downto 0); -- Asynchronous data transmit and receive
register
     ASPCR     : in STD_LOGIC_VECTOR(15 downto 0); -- Asynchronous serial port control register
     IOSR      : in STD_LOGIC_VECTOR(15 downto 0); -- Asynchronous I/O status register
     BRD       : in STD_LOGIC_VECTOR(15 downto 0);  -- Asynchronous boud-rate devisor register
     TXRXINT: out std_logic;                  -- Asynchronous hardware interrupt
     LOAD      : in STD_LOGIC;                     -- Start UART transmision
```

```
    READ       : in STD_LOGIC;                        -- Read UART receive data
    IO0        : inout STD_LOGIC;                        -- Software controlled I/O pin 0
    IO1        : inout STD_LOGIC;                        -- Software controlled I/O pin 1
    IO2        : inout STD_LOGIC;                        -- Software controlled I/O pin 2
    IO3        : inout STD_LOGIC;                       -- Software controlled I/O pin 3
    Soft_Reset : in Std_Logic);
end component;
-------------------------------------------------------------------------
-- Component SSP                                                       -
-------------------------------------------------------------------------
component SSP
       generic (
       tperiod_CI             : Time := Tables_tperiod_CI(TMS320C203_50_5V)(Nominal);      -- tc(CI)
       tperiod_SCK            : Time := Tables_tperiod_SCK(TMS320C203_50_5V)(Nominal);     -- tc(SCK)
       tdelay_CLKX_DX : Time := Tables_tdelay_CLKX_DX(TMS320C203_50_5V)(Nominal); -- td(DX)
       tdisable_CLKX_DX: Time := Tables_tdisable_CLKX_DX(TMS320C203_50_5V)(Nominal);      --
tdis(DX)
       tdelay_FSX_CLKX        : Time := Tables_tdelay_FSX_CLKX(TMS320C203_50_5V)(Nominal); --td(FS)
       thold_FSX_CLKX : Time := Tables_thold_FSX_CLKX(TMS320C203_50_5V)(Nominal)); --th(FS)H
    port (
        DX : out std_LOGIC;
        FSX: inout std_LOGIC;
        XINT: out std_LOGIC;
        XData_In: in std_logic_vector(15 downto 0);
        WR2SSP_EN: in std_logic;
        RD_SSP_EN: in std_logic;
        SSPCR: in std_logic_vector(15 downto 0);
        DR: in std_LOGIC;
        FSR: in std_LOGIC;
        RINT: out std_LOGIC;
        RData_Out: out std_logic_vector(15 downto 0);
        CLKOUT1: in Std_LOGIC;
        CLKR : in Std_LOGIC;
        CLKX : inout Std_LOGIC;
          Soft_Reset : in Std_Logic);
end component SSP;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
begin
   TMS1: Clk
    generic map (
        Tables_tperiod_CI(Processor)(SimCondition),      -- tc(CI)
        Tables_tpw_CI_posedge(Processor)(SimCondition), -- tw(CIH)
        Tables_tpw_CI_negedge(Processor)(SimCondition), -- tw(CIL)
        Tables_tperiod_CO(Processor)(SimCondition),      -- tc(CO)
        Tables_tpw_CO_posedge(Processor)(SimCondition), -- tw(COH)
        Tables_tpw_CO_negedge(Processor)(SimCondition), -- tw(COL)
        Tables_tdelay_CIH_CO(Processor)(SimCondition)) -- td(CIH-CO)
     port map (CLKOUT1,X2,DIV1,DIV2,CLKOUT1_ENABLE);
   TMS2: RstInt
    generic map (
        Tables_tperiod_CI(Processor)(SimCondition),  -- tc(CI)
        Tables_tsetup_RS_CI(Processor)(SimCondition),        -- tsu(RS)CI
        Tables_tsetup_RS_CO(Processor)(SimCondition),        -- tsu(RS)CO
        Tables_tpw_RS_negedge(Processor)(SimCondition),      -- tw(RSL)
        Tables_tdelay_RSN_fetch(Processor)(SimCondition),-- td(EX)
        Tables_tpw_INTx_negedge(Processor)(SimCondition), -- tw(IN)
        Tables_tdelay_INTx_fetch(Processor)(SimCondition), -- td(IN)
        Tables_tdelay_HLDN_HLDAN(Processor)(SimCondition), -- td(H-HA)
        Tables_tdelay_HLD_HLDA(Processor)(SimCondition), -- td(HH-HA)
        Tables_thighimped_HLDAN(Processor)(SimCondition), -- tz(M-HA)
        Tables_tenable_HLDA(Processor)(SimCondition))        -- ten(HA-M)
    port map(DataB,AddrB,PS_N,DS_N,RW_N,
             IS_N,READY,RD_N,WE_N,STRB_N,
             CLKOUT1,X2,BR_N,HOLDA_N,RS_N,
             NMI_N,INT1_N,INT2_N,INT3_N,
             RINT,XINT,TXRXINT,Stop,CLKOUT1_ENABLE,Soft_Reset);
   TMS3: MemAcc
    generic map (
    Tables_tsetup_A_RDN(Processor)(SimCondition),-- tsu(A)RD
        Tables_thold_A_RD(Processor)(SimCondition),          -- th(A)RD
        Tables_tdelay_CON_A(Processor)(SimCondition),                -- td(COL-A)
        Tables_thold_A_CON(Processor)(SimCondition),         -- th(A)CO
        Tables_tdelay_CO_RD(Processor)(SimCondition),                -- td(CO-RD)
        Tables_tdelay_CO_S(Processor)(SimCondition),         -- td(COL-S)
        Tables_tpw_RD_negedge(Processor)(SimCondition),      -- tw(RDL)
        Tables_tpw_RD_posedge(Processor)(SimCondition),      -- tw(RDH)
```

```
        Tables_tdelay_RD_WEN(Processor)(SimCondition),              -- td(RDW)
        Tables_taccess_A(Processor)(SimCondition),          -- ta(A)
        Tables_tsetup_D_RD(Processor)(SimCondition),        -- tsu(D)RD
        Tables_thold_D_RD(Processor)(SimCondition),         -- th(D)RD
        Tables_thold_D_A(Processor)(SimCondition),          -- th(D)A
        Tables_tsetup_D_CON_R(Processor)(SimCondition),     -- tsu(DCOL)R
        Tables_thold_D_CON_R(Processor)(SimCondition),              -- th(DCOL)R
        Tables_taccess_RD(Processor)(SimCondition),         -- ta(RD)
-- Write timing parameters
        Tables_tsetup_A_WN(Processor)(SimCondition),        -- tsu(A)W
        Tables_thold_A_W(Processor)(SimCondition),          -- th(A)W
        Tables_tsetup_A_CON(Processor)(SimCondition),               -- tsu(A)CO
        Tables_thold_A_CON_W(Processor)(SimCondition),              -- th(A)COw
        Tables_tpw_W_negedge(Processor)(SimCondition),              -- tw(WL)
        Tables_tdelay_CO_WE(Processor)(SimCondition),               -- td(CO-W)
        Tables_tsetup_D_WE(Processor)(SimCondition),        -- tsu(D)W
        Tables_thold_D_WE(Processor)(SimCondition),         -- th(D)W
        Tables_tsetup_D_CON_W(Processor)(SimCondition),     -- tsu(DCOL)W
        Tables_thold_D_CON_W(Processor)(SimCondition))              -- th(DCOL)W
    port map (DataB,AddrB,PS_N,DS_N,RW_N,IS_N,HOLDA_N,READY,RD_N,
                          WE_N,STRB_N,RS_N,BR_N,CLKOUT1,VAddrB,VDataB,VRW_N,
                          VIS_N,VDS_N,VPS_N,VBR_N,Busy_cycle,Wait_nr_cycle);
  TMS4: UART
     port map (TX,RX,CLKOUT1,ADTR,ASPCR,IOSR,BRD,TXRXINT,
                  LOAD,READ,IO0,IO1,IO2,IO3,Soft_Reset);
  TMS5: SSP
        generic map(
        Tables_tperiod_CI(Processor)(Nominal),                      -- tc(CI)
        Tables_tperiod_SCK(Processor)(Nominal),             -- tc(SCK)
        Tables_tdelay_CLKX_DX(Processor)(Nominal),    -- td(DX)
        Tables_tdisable_CLKX_DX(Processor)(Nominal), -- tdis(DX)
        Tables_tdelay_FSX_CLKX(Processor)(Nominal),   --td(FS)
        Tables_thold_FSX_CLKX(Processor)(Nominal))    --th(FS)H
        port map (DX,FSX,XINT,XData_In,WR2SSP_EN,RD_SSP_EN,
         SSPCR,DR,FSR,RINT,RData_Out,CLKOUT1,CLKR,CLKX,Soft_Reset);

 TMS320_MAIN : process
--------------------------------------------------------------------------
--------------------------------------------------------------------------
  procedure PWRITE(
     set_data             : in Std_Logic_Vector(15 downto 0);
     set_address          : in Std_Logic_Vector(15 downto 0);
     set_wait             : in Std_Logic_Vector(15 downto 0)) is
     variable str4,str_4  : string(1 to 4);
     variable Good2       : Boolean;
  begin
      Vec2Hex (set_data,str4,Good2);
      Vec2Hex (set_address,str_4,Good2);
      report "Write to Program memory! Address : "&str_4 &" Data value : "&str4 &""
      severity Note;
      VDataB <= set_data;
      VAddrB <= set_address;
      Wait_nr_cycle <= set_wait;
      VRW_N <= '0';
      VIS_N <= '1';
      VDS_N <= '1';
      VBR_N <= '1';
      VPS_N <= '0';
      wait until BUSY_cycle ='L'and BUSY_cycle'EVENT;
      VPS_N <= '1';
      wait for 5 ns;
  end PWRITE;
--------------------------------------------------------------------------
--------------------------------------------------------------------------
  procedure PREAD(
     set_data             : in Std_Logic_Vector(15 downto 0);
     set_address          : in Std_Logic_Vector(15 downto 0);
     set_wait             : in Std_Logic_Vector(15 downto 0))  is
     variable str4,str_4  : string(1 to 4);
     variable Good2       : Boolean;
  begin
      Vec2Hex (set_data,str4,Good2);
      Vec2Hex (set_address,str_4,Good2);
      report "Read from Program memory! Address : "&str_4 &" Expected data : "&str4 &""
      severity Note;
      VDataB <= set_data;
      VAddrB <= set_address;
```

```
      Wait_nr_cycle <= set_wait;
      VRW_N <= '1';
      VIS_N <= '1';
      VDS_N <= '1';
      VBR_N <= '1';
      VPS_N <= '0';
      wait until BUSY_cycle ='L' and BUSY_cycle'EVENT;
      VPS_N <= '1';
      wait for 5 ns;
  end PREAD;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  procedure DWRITE(
     set_data             : in Std_Logic_Vector(15 downto 0);
     set_address          : in Std_Logic_Vector(15 downto 0);
     set_wait             : in Std_Logic_Vector(15 downto 0))  is
     variable str4,str_4  : string(1 to 4);
     variable Good2       : Boolean;
  begin
      Vec2Hex (set_data,str4,Good2);
      Vec2Hex (set_address,str_4,Good2);
      report "Write to Local Data memory! Address : "&str_4 &" Data value : "&str4 &""
      severity Note;
      VDataB <= set_data;
      VAddrB <= set_address;
      Wait_nr_cycle <= set_wait;
      VRW_N <= '0';
      VIS_N <= '1';
      VDS_N <= '0';
      VBR_N <= '1';
      VPS_N <= '1';
      wait until BUSY_cycle ='L'and BUSY_cycle'EVENT;
      VDS_N <= '1';
      wait for 5 ns;
  end DWRITE;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  procedure DREAD(
     set_data             : in Std_Logic_Vector(15 downto 0);
     set_address          : in Std_Logic_Vector(15 downto 0);
     set_wait             : in Std_Logic_Vector(15 downto 0)) is
     variable str4,str_4  : string(1 to 4);
     variable Good2       : Boolean;
  begin
     Vec2Hex (set_data,str4,Good2);
     Vec2Hex (set_address,str_4,Good2);
     report "Read from Local Data memory! Address : "&str_4 &" Expected value : "&str4 &""
     severity Note;
     VDataB <= set_data;
     VAddrB <= set_address;
     Wait_nr_cycle <= set_wait;
     VRW_N <= '1';
     VIS_N <= '1';
     VDS_N <= '0';
     VBR_N <= '1';
     VPS_N <= '1';
     wait until BUSY_cycle ='L' and BUSY_cycle'EVENT;
     VDS_N <= '1';
  end DREAD;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  procedure GWRITE (
     set_data             : in Std_Logic_Vector(15 downto 0);
     set_address          : in Std_Logic_Vector(15 downto 0);
     set_wait             : in Std_Logic_Vector(15 downto 0)) is
     variable str4,str_4  : string(1 to 4);
     variable Good2       : Boolean;
  begin
      Vec2Hex (set_data,str4,Good2);
      Vec2Hex (set_address,str_4,Good2);
      report "Write to Global Data memory! Address : "&str_4 &" Data value : "&str4 &""
      severity Note;
      VDataB <= set_data;
      VAddrB <= set_address;
      Wait_nr_cycle <= set_wait;
      VRW_N <= '0';
      VIS_N <= '1';
```

```
      VDS_N <= '1';
      VBR_N <= '0';
      VPS_N <= '1';
      wait until BUSY_cycle ='L'and BUSY_cycle'EVENT;
      VBR_N <= '1';
      wait for 5 ns;
  end GWRITE;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  procedure GREAD (
     set_data              : in Std_Logic_Vector(15 downto 0);
     set_address           : in Std_Logic_Vector(15 downto 0);
     set_wait              : in Std_Logic_Vector(15 downto 0)) is
     variable str4,str_4   : string(1 to 4);
     variable Good2        : Boolean;
  begin
     Vec2Hex (set_data,str4,Good2);
     Vec2Hex (set_address,str_4,Good2);
     report "Read from Global Data memory! Address : "&str_4 &" Expected value : "&str4 &""
     severity Note;
     VDataB <= set_data;
     VAddrB <= set_address;
     Wait_nr_cycle <= set_wait;
     VRW_N <= '1';
     VIS_N <= '1';
     VDS_N <= '1';
     VBR_N <= '0';
     VPS_N <= '1';
     wait until BUSY_cycle ='L' and BUSY_cycle'EVENT;
     VBR_N <= '1';
  end GREAD;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  procedure IWRITE(
     set_data              : in Std_Logic_Vector(15 downto 0);
     set_address           : in Std_Logic_Vector(15 downto 0);
     set_wait              : in Std_Logic_Vector(15 downto 0)) is
     variable str4,str_4   : string(1 to 4);
     variable Good2        : Boolean;
  begin
     Vec2Hex (set_data,str4,Good2);
     Vec2Hex (set_address,str_4,Good2);
     report "Write to I/O port! Address : "&str_4 &" Data value : "&str4 &""
     severity Note;
     VDataB <= set_data;
     VAddrB <= set_address;
     Wait_nr_cycle <= set_wait;
     VRW_N <= '0';
     VIS_N <= '0';
     VDS_N <= '1';
     VBR_N <= '1';
     VPS_N <= '1';
     wait until BUSY_cycle ='L'and BUSY_cycle'EVENT;
     VIS_N <= '1';
     wait for 5 ns;
  end IWRITE;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  procedure IREAD(
     set_data :in Std_Logic_Vector(15 downto 0);
     set_address :in Std_Logic_Vector(15 downto 0);
     set_wait :in Std_Logic_Vector(15 downto 0)) is
     variable str4,str_4 : string(1 to 4);
     variable Good2        : Boolean;
  begin
     Vec2Hex (set_data,str4,Good2);
     Vec2Hex (set_address,str_4,Good2);
     report "Read from I/O port! Address : "&str_4 &" Expected data : "&str4 &""
     severity Note;
     VDataB <= set_data;
     VAddrB <= set_address;
     Wait_nr_cycle <= set_wait;
     VRW_N <= '1';
     VIS_N <= '0';
     VDS_N <= '1';
     VBR_N <= '1';
     VPS_N <= '1';
```

```
    wait until BUSY_cycle ='L' and BUSY_cycle'EVENT;
    VIS_N <= '1';
  end IREAD;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-- This procedure is used for asynchronous transmision                -
-- Change of signal LOAD will start process in arhitectures UARD for
-- transmision data ADTR(7 downto 0) on TX line
  procedure WRUART(
    set_data :in Std_Logic_Vector(15 downto 0);
    set_baud_rate :in Std_Logic_Vector(15 downto 0)) is
    begin
      LOAD <='0';
      wait for 100 NS;
      ADTR(15 downto 0) <=set_data;
      BRD <= set_baud_rate;
      LOAD <='1';
--      wait for 150 * Tables_tperiod_CI(Processor)(SimCondition);
    end WRUART;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-- This procedure is used for asynchronous reception data             -
-- Change of signal READ will start process in arhitectures UARD for
-- read data in register ADTR(7 downto 0)
  procedure RDUART(
    set_data :in Std_Logic_Vector(15 downto 0);
    set_baud_rate :in Std_Logic_Vector(15 downto 0)) is
    begin
    READ <='1';
    BRD <= set_baud_rate;
    wait for 100 NS;
    READ <='0';
    end RDUART;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-- This procedures is used for program general propose I/O pins
-- General propose I/O pins are set as input or output by writing
-- in the Asynchronous Serial Port Control Register (ASPCR)
--  ASPCR  bit0 =1     => IO0 is configured as an output, else input
--         bit1 =1     => IO1 is configured as an output, else input
--         bit2 =1     => IO2 is configured as an output, else input
--         bit3 =1     => IO3 is configured as an output, else input
-- The logic level of general propose I/O pins programed as output is
-- determined by bit3,bit2,bit1,bit0 in the I/O status register (IOSR).
  procedure WRIOP(
    set_data :in Std_Logic_Vector(15 downto 0);
    set_mod :in Std_Logic_Vector(15 downto 0)) is
    begin
     ASPCR(15 downto 4) <="000000000000";
     ASPCR(3 downto 0) <= set_mod(3 downto 0);
     IOSR(15 downto 4) <= "000000000000";
    IOSR(3 downto 0) <= set_data(3 downto 0);
    end WRIOP;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-- This procedures is used for read general propose I/O pins
-- Read of general purpose I/O pins is done in background and is store
-- in register IOSR_READ
  procedure RDIOP(
    set_data :in Std_Logic_Vector(15 downto 0);
    set_address :in Std_Logic_Vector(15 downto 0)) is
    begin
    end RDIOP;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
 procedure WRSSP(
    set_data  :in Std_Logic_Vector(15 downto 0);
    set_mod :in Std_Logic_Vector(15 downto 0)) is
    begin
        XData_In <= set_data;
        case set_mod(2 downto 0) is
                when "001" => SSPCR <= "1000001100111110"; -- Burst Mode with Internal Frame Sync
                when "010" => SSPCR <= "1000111100111100"; -- Continuous Mode with Internal Frame
Sync
                when "011" => SSPCR <= "1000001100110110"; -- Burst Mode with External Frame Sync
                when "100" => SSPCR <= "1000111100110100"; -- Continuous Mode with External Frame
Sync
```

```
                when others => null;
        end case;

        WR2SSP_EN <= '1';
                wait for 2.1*Tables_tperiod_CI(Processor)(SimCondition);
                WR2SSP_EN <= '0';
--      wait for 64*Tables_tperiod_CI(Processor)(SimCondition);

    end WRSSP;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  procedure RDSSP(
--    set_data :in Std_Logic_Vector(15 downto 0);
    set_mod :in Std_Logic_Vector(15 downto 0)) is
    begin
        SSPCR <= set_mod;
        loopback_SSP <='1';
        wait for 1 ms;
        loopback_SSP <='0';
    end RDSSP;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  procedure WAIT_TIME(
    set_low :in Std_Logic_Vector(15 downto 0);
    set_high :in Std_Logic_Vector(15 downto 0)) is
    variable WAIT_COUNTER1: INTEGER :=0;
    variable WAIT_COUNTER2: INTEGER :=0;
    begin
     WAIT_COUNTER2 := 65536* To_Logic_Integer(set_high)+ To_Logic_Integer(set_low);
--    WAIT_COUNTER2 :=2;
    wait1 <='1';
     for i in 1 to WAIT_COUNTER2 loop
      wait until RISING_EDGE(CLKOUT1);
     end loop;
    wait1 <='0';
    end WAIT_TIME;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  procedure Main_case is
    variable    Commands        :  CommandArray(1 to 100);
        variable        NumCommands             :       Natural;
        variable        Good                    :       Boolean;
        variable        PC                      :       Integer;
        constant        HeaderMsg               :       String          := "Main process:";
        constant        MsgSeverity             :       Severity_Level := Failure;
        variable    ErrFlag                     :       Boolean;
  begin
    -- Read_Parse_file
    FileParser ("TMS320.CMD",Commands,NumCommands,ErrFlag);
        if ErrFlag then
                report HeaderMsg&"Errors found in TMS320C203.cmd file. Execute halts!!!"
                severity MsgSeverity;
                wait;
        else
                PC := 0;
  --      wait until BUSY='0';
                while PC < NumCommands Loop
             if Soft_Reset ='1' then
                PC:= 0;
                ASPCR(15 downto 0) <="0000000000000000";
                wait until Soft_Reset = '0';
             end if;
                PC := PC + 1;
          --    if PC > 0 then
                        case Commands(pc).command is
                  when "PWRIT" => PWRITE(Commands(pc).data,Commands(PC).addr,Commands(pc).sdata);
                                when                                    "PREAD"                                 =>
PREAD(Commands(pc).data,Commands(PC).addr,Commands(pc).sdata);
                                when                                    "DWRIT"                                 =>
DWRITE(Commands(pc).data,Commands(PC).addr,Commands(pc).sdata);
                                when                                    "DREAD"                                 =>
DREAD(Commands(pc).data,Commands(PC).addr,Commands(pc).sdata);
                  when "GWRIT" => GWRITE(Commands(pc).data,Commands(PC).addr,Commands(pc).sdata);
                                when                                    "GREAD"                                 =>
GREAD(Commands(pc).data,Commands(PC).addr,Commands(pc).sdata);
                                when                                    "IWRIT"                                 =>
IWRITE(Commands(pc).data,Commands(PC).addr,Commands(pc).sdata);
```

```
                              when                                    "IREAD"                                 =>
IREAD(Commands(pc).data,Commands(PC).addr,Commands(pc).sdata);
                 when "WRSSP" => WRSSP(Commands(pc).data,Commands(PC).addr);
                              when "RDSSP" => RDSSP(Commands(PC).addr);
                              when "WRUAR" => WRUART(Commands(pc).data,Commands(PC).addr);
                              when "RDUAR" => RDUART(Commands(pc).data,Commands(PC).addr);
                              when "WRIOP" => WRIOP(Commands(pc).data,Commands(PC).addr);
                 when "RDIOP" => RDIOP(Commands(pc).data,Commands(PC).addr);
                 when "WAITC" => WAIT_TIME(Commands(pc).data,Commands(PC).addr);
                              when others      => null;
                       end case;
        --      end if;
         end loop;
        end if;
--      report "End of TMS320C203 commands";
    wait until Soft_reset ='1';
--      wait;
        end Main_Case;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
begin

 wait for 10 ns;
  Main_case;
end process;
end STRUCTURE;
```

Anexa F.3. Test Bench în VHDL pentru TMS320c203

```
--===========================================================================--
-- Design units     : TestBench
--
-- File name        : TestBench.vhd
--
-- Purpose          : This FILE defines all the entities and architectures
--                  used to test the TMS320C203 design.
--
-- Note             : Components are held as simple as possible.
--
-- Limitations      : None known
--
-- Errors           : None known
--
-- Library          : TMS320C203_Lib
--
-- Dependencies     : IEEE.Std_Logic_1164
--
-- Author           : Ovidiu Lupas
--                  Petru Demian
--                  TimTeh Electronics s.r.l.
--                  Calea Martirilor nr. 1
--                  1900 Timisoara
--                  Romania
--                  http://www.timteh.ro
--
-- Simulator        : ModelSim PE/PLUS version 4.7b on a Windows95 PC
--                  ActiveVHDL version 3.2 on a Windows95 PC
-------------------------------------------------------------------------------
-- Structure of the TestBench is :
--
-- Clock Generator                TMS320C203                        Memory
--
--|               |          |                        |          |               |
--|            CLK|__________\|X2          A0..A15   |__________\| A0..A15       |
--|               |          /|                       |          /|               |
--|           INT1|__________\|INT1_N      D0..D15   |/_________\| D0..D15       |
--|               |          /|                       |\         /|               |
--|           INT2|__________\|INT2_N                 |          |               |
--|               |          /|                       |          |               |
--|           INT3|__________\|INT3_N                 |          |               |
--|               |          /|                  WE_N|__________\|WE_N           |
--|            NMI|__________\|NMI_N                  |          /|               |
--|               |          /|                  RD_N|__________\|RD_N           |
--|          RESET|__________\|RST_N                  |          /|               |
--|               |          /|                  PS_N|____|\     |               |
--|           DIV1|__________\|DIV1              IS_N|____| \    |               |
--|               |          /|                  DS_N|____|  \_|CS_N           |
--|           DIV2|__________\|DIV2              BR_N|____|  /  |               |
--|               |          /|                       |    | /   |               |
--|          READY|__________\|READY                  |    |/    |               |
--|               |          /|                       |          |               |
--|_______________|           |                       |          |_______________|
--              ____________|Tx                    Dx|_____________________
--             |            |                        |                     |
--             |            |                     FSx|__________________  |
--             |            |                        |                  | |
--             |___________\|Rx                  CLKx|_______________   | |
--                         /|                        |               |  | |
--                          |                     Dr |/______________|__|_|
--                          |                        |\              |  |
--                          |                     FSr|/______________|__|
--                          |                        |\              |
--                          |                    CLKr|/______________|
--                          |                        |\
--                          |                        |
--                          |                        |
--                          |________________________|
-------------------------------------------------------------------------------
-- Revision list
-- Version    Author         Date              Changes
--
-- 0.1          OL          6 March 98        New model
-------------------------------------------------------------------------------
```

```
--
-------------------------------------------------------------------------------
-- Clock generator
-------------------------------------------------------------------------------
library IEEE,work;
use IEEE.Std_Logic_1164.all;
use work.simulation.all;
use work.TMS320C203_Tim.all;
--
entity ClkGen is
   generic (
      tpw_CI_posedge : Time := Tables_tpw_CI_posedge(TMS320C203_50_5V)(Nominal));  -- tc(CI)
   port (
      READY     : out     Std_Logic;     -- Ready pin
      RS_N      : out     Std_Logic;     -- Reset Pin
      NMI_N     : out     Std_Logic;     -- NMI Pin
      DIV1      : out     Std_Logic;     -- Clock mode signal 1
      DIV2      : out     Std_Logic;     -- Clock mode signal 2
      INT1_N    : out     Std_Logic;       -- INT1 Pin
      INT2_N    : out     Std_Logic;       -- INT2 Pin
      INT3_N    : out     Std_Logic;       -- Reset Pin
      X2        : out     Std_Logic);    -- Oscillator clock
end ClkGen;--==================== End of entity ==============================--
--================================ Architecture ==============================--
architecture Behavioural of ClkGen is
begin
      DIV1 <= '0';
      DIV2 <= '0';
      READY <= '1';
-------------------------------------------------------------------------------
-- Provide the external clock signal to the processor
-------------------------------------------------------------------------------
ClkDriver : process
    variable clktmp: std_logic := '0';
    begin
        X2 <= clktmp;  -- Attach your clock here
        clktmp := not clktmp;
        wait for tpw_CI_posedge;
end process ClkDriver;
-------------------------------------------------------------------------------
-- Provide the reset signal to the processor
-------------------------------------------------------------------------------
Rst : process
begin
--    wait for 100 ns;
      RS_N <= '0';
      wait for 300 ns;
      RS_N <= '1';
      wait for 10000 ns;
end process Rst;
-------------------------------------------------------------------------------
-- Provide the interrupt signals to the processor
-------------------------------------------------------------------------------
Int : process
begin
--    wait for 100 ns;
--    Int1_N <= '0';
--    wait for 50 ns;
--    Int1_N <= '1';
      wait for 5000 ns;
end process Int;
end architecture Behavioural;--==================== End of Architecture ====================--
-------------------------------------------------------------------------------
-- DECODER FOR SELECT MEMORY
-------------------------------------------------------------------------------
library ieee,work,std;
use ieee.std_logic_1164.all;
use work.Simulation.all;
use work.TMS320C203_Def.all;          -- For custom functions
use work.TMS320C203_Tim.all;
-------------------------------------------------------------------------------
entity DECODER is
port (
    PS_N    : in Std_Logic;          -- Program select
    DS_N    : in Std_Logic;          -- Data select
    IS_N    : in Std_Logic;          -- I/O space select
    BR_N    : in Std_Logic;          -- Data ready input
```

```
    SELECT_RAM :out Std_Logic);     -- Select memory
end DECODER;--===================== End of entity ==============================--
--================================= Architecture ==============================--
architecture Behavioural_DECODER of DECODER is
begin
   process (PS_N,DS_N,IS_N,BR_N)
   begin
          SELECT_RAM <= PS_N and DS_N and IS_N and BR_N;
   end process;
end architecture Behavioural_DECODER;--===== End of Architecture ==================--
--------------------------------------------------------------------------------
-- Memory
--------------------------------------------------------------------------------
library ieee,work,std;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.Simulation.all;
use work.TMS320C203_Def.all;          -- For custom functions
use work.TMS320C203_Tim.all;
--
entity SRAM is
   generic (
      output_enable_to_valid_data : Time :=8 ns;
       output_disable_to_output_inactive : Time := 13 ns);
   port (
       SELECT_RAM  : in     Std_Logic;
       WE_N        : in     Std_Logic;  -- Ready pin
       RD_N        : in     Std_Logic;  -- Ready pin
       DataB       : inout  Std_Logic_Vector(15 downto 0);  -- Data bus
       AddrB       : in     Std_Logic_Vector(15 downto 0)); -- Address bus
end SRAM;--===================== End of entity ==============================--
--================================= Architecture ==============================--
architecture Behavioural_RAM of SRAM is
   type MEMORY is array(0 to 31) of Std_Logic_Vector(15 downto 0);
   signal DDATA:  Std_Logic_Vector(15 downto 0);
   signal DAddrB: Std_Logic_Vector(15 downto 0);
   signal WSEL: Std_Logic;
begin
   process (SELECT_RAM,RD_N,WE_N)
       variable MEM: Memory;
   begin
     if SELECT_RAM ='0' then
        if not RD_N'STABLE then
           if RD_N ='0' then
              DataB <= Mem(To_Logic_Integer(AddrB)) after output_enable_to_valid_data;
           else
              DataB <= "ZZZZZZZZZZZZZZZZ" after output_disable_to_output_inactive;
           end if;
        elsif WE_N ='1' and not WE_N'STABLE then
           MEM(To_Logic_Integer(AddrB)) := DataB;
        end if;
     else
        DataB <="ZZZZZZZZZZZZZZZZ" after output_disable_to_output_inactive;
     end if;
   end process;
end architecture Behavioural_RAM;--================= End of Architecture ================--
--------------------------------------------------------------------------------
-- TestBench
--------------------------------------------------------------------------------
library ieee,work,std;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.Simulation.all;
use work.TMS320C203_Def.all;          -- For custom functions
use work.TMS320C203_Tim.all;
use work.ClkGen;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
entity TESTBNCH is
end TESTBNCH;
--================================= Architecture ==============================--
architecture stimulus of TESTBNCH is
     signal DataB      : Std_Logic_Vector(15 downto 0);  -- Data bus
     signal AddrB      : Std_Logic_Vector(15 downto 0);  -- Address bus
     signal PS_N       : Std_Logic;                      -- Program select
     signal DS_N       : Std_Logic;                      -- Data select
     signal RW_N       : Std_Logic;                      -- Read/write
```

```
    signal IS_N        : Std_Logic;                    -- I/O space select
    signal READY       : Std_Logic;                    -- Data ready input
    signal RD_N        : Std_Logic;                    -- Read select
    signal WE_N        : Std_Logic;                    -- Write enable
    signal STRB_N      : Std_Logic;                    -- Strobe
    signal BR_N        : Std_Logic;                    -- Bus request
    signal HOLDA_N     : Std_Logic;                    -- Hold acknowledge
    signal IO0         : Std_Logic;                    -- Software controlled I/O
    signal IO1         : Std_Logic;                    -- Software controlled I/O
    signal IO2         : Std_Logic;                    -- Software controlled I/O
    signal IO3         : Std_Logic;                    -- Software controlled I/O
    signal RS_N        : Std_Logic;                    -- Hardware reset
    signal NMI_N       : Std_Logic;                    -- Nonmaskable interrupt
    signal INT1_N      : Std_Logic;                    -- Interrupt 1 /HOLD_N request
    signal INT2_N      : Std_Logic;                    -- Interrupt 2
    signal INT3_N      : Std_Logic;                    -- Interrupt 3
    signal CLKOUT1     : Std_Logic;                    -- Master clock output
    signal X2          : Std_Logic;                    -- Oscillator input
    signal DIV1        : Std_Logic;                    -- Clock mode 1
    signal DIV2        : Std_Logic;                    -- Clock mode 2
    signal CLKX        : Std_Logic;                    -- Transmit clock
    signal CLKR        : Std_Logic;                    -- Receive clock
    signal FSR         : Std_Logic;                    -- Frame synchro / receive
    signal FSX         : Std_Logic;                    -- Frame synchro / transmit
    signal DR          : Std_Logic;                    -- Serial data receive
    signal DX          : Std_Logic;                    -- Serial port transmit
    signal TX          : Std_Logic;                    -- Asynchronous transmit pin
    signal RX          : Std_Logic;                    -- Asynchronous receive pin
    signal SELECT_RAM  : Std_Logic;
    signal done        : boolean := false;
-------------------------------------------------------------------------------
-- Component declarations
-------------------------------------------------------------------------------
component TMS320
  generic (
    SimCondition : SimCondType   := Nominal;
    Processor    : ProcessorType := TMS320C203_50_5V);
  port(
    DataB    : inout   Std_Logic_Vector(15 downto 0); -- Data bus
     AddrB   : inout   Std_Logic_Vector(15 downto 0); -- Address bus
    -- Memory control signals (8)
    PS_N     : out     Std_Logic;          -- Program select
    DS_N     : out     Std_Logic;          -- Data select
    RW_N     : out     Std_Logic;          -- Read/write
    IS_N     : out     Std_Logic;          -- I/O space select
    READY    : in      Std_Logic;          -- Data ready input
    RD_N     : out     Std_Logic;          -- Read select
    WE_N     : out     Std_Logic;          -- Write enable
    STRB_N   : out     Std_Logic;          -- Strobe
      -- Multi-processing signals (5)
    BR_N     : out     Std_Logic;          -- Bus request
    HOLDA_N  : out     Std_Logic;          -- Hold acknowledge
    IO0      : inout   Std_Logic;          -- Software controlled I/O
    IO1      : inout   Std_Logic;          -- Software controlled I/O
    IO2      : inout   Std_Logic;          -- Software controlled I/O
    IO3      : inout   Std_Logic;          -- Software controlled I/O
      -- Initialization, Interrupts and Reset signals (7)
    RS_N     : in      Std_Logic;          -- Hardware reset
    NMI_N    : in      Std_Logic;          -- Nonmaskable interrupt
    INT1_N   : in      Std_Logic;          -- Interrupt 1 /HOLD_N request
    INT2_N   : in      Std_Logic;          -- Interrupt 2
    INT3_N   : in      Std_Logic;          -- Interrupt 3
      -- Oscillator, PLL and Timer signals (7)
    CLKOUT1  : inout   Std_Logic;          -- Master clock output
    X2       : in      Std_Logic;          -- Oscillator input
    DIV1     : in      Std_Logic;          -- Clock mode 1
    DIV2     : in      Std_Logic;          -- Clock mode 2
    -- Serial port and UART signals (8)
    CLKX     : inout   Std_Logic;          -- Transmit clock
    CLKR     : inout   Std_Logic;          -- Receive clock
    FSR      : inout   Std_Logic;          -- Frame synchro / receive
    FSX      : inout   Std_Logic;          -- Frame synchro / transmit
    DR       : inout   Std_Logic;          -- Serial data receive
    DX       : inout   Std_Logic;          -- Serial port transmit
    TX       : out     Std_Logic;          -- Asynchronous transmit pin
    RX       : in      Std_Logic);         -- Asynchronous receive pin
end component ;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
component ClkGen
   port (
      READY    : out    Std_Logic;        -- Ready pin
      RS_N     : out    Std_Logic;         -- Reset Pin
      NMI_N    : out    Std_Logic;        -- NMI Pin
      DIV1     : out    Std_Logic;        -- Clock mode signal 1
      DIV2     : out    Std_Logic;        -- Clock mode signal 2
      INT1_N   : out    Std_Logic;        -- INT1 Pin
      INT2_N   : out    Std_Logic;        -- INT2 Pin
      INT3_N   : out    Std_Logic;        -- Reset Pin
      X2       : out    Std_Logic);       -- Oscillator clock
end component;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
component Decoder
    port (
      PS_N        : in Std_Logic;         -- Program select
      DS_N        : in Std_Logic;         -- Data select
      IS_N        : in Std_Logic;         -- I/O space select
      BR_N        : in Std_Logic;         -- Data ready input
      SELECT_RAM :out Std_Logic);         -- Select memory
end component;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
component SRAM
    port (
       SELECT_RAM  : in     Std_Logic;
       WE_N        : in     Std_Logic;                          -- Ready pin
       RD_N        : in     Std_Logic;                          -- Ready pin
       DataB       : inout  Std_Logic_Vector(15 downto 0);  -- Data bus
       AddrB       : in     Std_Logic_Vector(15 downto 0)); -- Address bus
end component;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
for UUT: TMS320 use entity work.TMS320(STRUCTURE);
for GEN: ClkGen use entity work.ClkGen(Behavioural);
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
begin
-------------------------------------------------------------------------------
-- Instantiation of components
-------------------------------------------------------------------------------
  UUT: TMS320 port map (
       DataB,AddrB,PS_N,DS_N,RW_N,IS_N,READY,RD_N,WE_N,
       STRB_N,BR_N,HOLDA_N,IO0,IO1,IO2,IO3,RS_N,
       NMI_N,INT1_N,INT2_N,INT3_N,CLKOUT1,X2,
       DIV1,DIV2,CLKX,CLKR,FSR,FSX,DR,DX,TX,RX);
  GEN: ClkGen port map (
       READY,RS_N,NMI_N,DIV1,DIV2,INT1_N,INT2_N,INT3_N,X2);
  SEL: DECODER port map (
       PS_N,DS_N,IS_N,BR_N,SELECT_RAM) ;
  RAM: SRAM port map (
       SELECT_RAM,WE_N,RD_N,DataB,AddrB);
-------------------------------------------------------------------------------
-- Implements the loopback for asynchronous serial port
-------------------------------------------------------------------------------
   SHORT_UART: process(TX)
   begin
        RX <= TX;
   end process SHORT_UART;
-------------------------------------------------------------------------------
-- Implements the loopback for synchronous serial port
-------------------------------------------------------------------------------
   SHORT_SSP: process(DX,FSX,CLKX)
   begin
        DR <= DX;
        FSR <= FSX;
        CLKR <= CLKX;
   end process SHORT_SSP;
-------------------------------------------------------------------------------
-- Applies signal on IO software controlled pins
-------------------------------------------------------------------------------
   IO_PORT_UART: process
   begin
     IO3 <='0';
```

```
    IO2 <='0';
--    IO1 <='1';
    IO0 <='1';
    wait for 400 NS;
    IO3 <='1';
    wait for 40 NS;
  end process IO_PORT_UART;
end stimulus;
```

Anexa F.4. Manual de utilizare pentru modelarea TMS320c203

1) Procesoare suportate:

TMS320C203 50ns@5V
TMS320C203 35ns@5V
TMS320C203 25ns@5V
TMS320LC203 50ns@3.5V

În biblioteca TMS320C203_Lib sunt definite tabelele „ProcessorType" şi „SimCondType", iar utilizatorul poate selecta tipul de procesor şi cazul de simulare şi pe de nivelul superior (de exemplu din TestBench.vhd) cu sintaxa

```
generic (
    SimCondition : SimCondType := Nominal;
    Processor : ProcessorType := TMS320C203_50_5V);
```

Procesoarele posibile sunt:

```
TMS320C203_50_5V for TMS320C203 50ns@5V
TMS320C203_35_5V for TMS320C203 35ns@5V
TMS320C203_25_5V for TMS320C203 25ns@5V
TMS320C203_50_3_3V for TMS320LC203 50ns@3.5V
```

Cazurile de simulare sunt: Minimal, Nominal şi Maximal.
În implementarea curentă tactul utilizat provine de pe pinul X2 divizat cu 2.

2) Comenzile implementate în fişierul de comandă sunt:

- scriere în spaţiul memoriei de program
  PWRIT Address Data Wait_state_number
  exemplu: PWRIT 0000 0001 0002
  Comanda scrie la adresa 0000 (hexa), data 0001 (hexa) şi înseriază 0002 (hexa) stări wait.
- citire din spaţiul memoriei de program
  PREAD Address Data Wait_state_number

- scriere în memoria de date locală
  DWRIT Address Data Wait_state_number
- citire din memoria de date locală
  DREAD Address Data Wait_state_number

- scriere în memoria globală de date
  GWRIT Addr Data Wait_state_number
- citire din memoria globală de date
  GREAD Addr Data Wait_state_number

- scriere în spaţiul I/O
  IWRIT Addr Data Wait_state_number
- citire din spaţiul I/O
  IREAD Addr Data Wait_state_number

- aşteaptă HIGH*65536+LOW pulsuri CLKOUT1
  WAITC HIGH LOW
  Exemplu: WAITC 0000 0010
  Comanda aşteaptă 16 pulsuri de tact CLKOUT1

- scriere la portul serial asincron
  WRUAR BRD DATA

Exemplu: WRUAR 0001 2211
Comanda trimite octetul 11 (hexa) cu frecvenţa programat în registrul BRD (CLKOUT1 /16 *1)
- citire de la portul serial asincron
RDUAR BRD DATA

- scriere la portul serial sincron
WRSSP MODE WORD
- MODE=1 pentru Burst Mode With Internal Frame Sync
- MODE=2 pentru Continuous Mode With Internal Frame Sync
- MODE=3 pentru Burst Mode With External Frame Sync
- MODE=4 pentru Continuous Mode With External Frame Sync
Exemplu: WRSSP 0001 1111
Comanda trimite cuvântul 1111 (hexa) în modul Burst cu Internal Frame Sync
- citire de la portul serial sincron
RDSSP MODE WORD

- scriere la porturile I/O
WRIOP MODE DATA
Programează pinii I/O ca intrări sau ieşiri:
MODE = XXXX XXXX XXXX IO3 IO2 IO1 IO0
Dacă bitul IO "n" este pe 1 pinul "n" este programat ca ieşire.

DATA = date pentru pinii de ieşire
Exemplu: WRIOP 0002 XXX2
Comanda programează bitul 2 ca ieşire cu "1" logic iar IO3,IO2 and IO0 ca intrare.
- citire de pe pinii I/O
RDIOP MODE WORD

3) Efectul scrierii în registrul Wait State Register a fost implementat şi simulat. De asemenea au fost simulate şi combinaţii de două comenzi.

4) Operaţiile pe IO0-IO3 sub controlul registrului ASPCR au fost implementate.

5) Întreruperile au fost simulate şi verificate pentru valabilitatea timing-ului. De asemenea a fost simulat semnalul pe pinul RS_N pin al procesorului.

6) Toate combinaţiile de două comenzi au fost simulate iar modelul procesorului TMS320C203 este structurat în 5 componente cu arhitectură separată:
master clock generator
reset and interrupts controller
memory access controller
synchronous serial port
asynchronous serial port.
Fişiere componente:
- TMS_LIB.VHD = conţine bibliotecile
- T320C203.VHD = conţine modelul procesorului
- Test_b.vhd = test bench pentru procesor
- TMS320.cmd = fişier de comandă pentru procesor

Anexa G. Bibliografie

[1] Miron Abramovici, Melvin A. Breuer, Arthur D. Friedman, Digital System Testing and Testable Design.

[2] Accolade PeakVHDL software.

[3] Acculogic Inc. Boundary Scan Product. http://www.acculogic.com.

[4] Actel, IEEE Standard 1149.1 (JTAG) in the 3200DX Family, Application Note, 2003.

[5] Agilent Technologies, AC EXTEST Preliminary Specification, 2001.

[6] Aldec Inc. Active-VHDL. http://www.aldec.com.

[7] Altera. IEEE 1149.1 (JTAG) Boundary-Scan Testing in Altera Devices. February 1998. Application Note 39. http://www.altera.com.

[8] John Andrews, P1149.1A Extensions to IEEE-STD-1149.1-1990, Application Note /890.

[9] J. Andrews, Roadmap for Extending IEEE 1149.1 for Hierarchical Control of Locally Stored. Standardized Command Set, Test Programs, Proceedings of IEEE International Test Conference, 1994, pp. 300-306.

[10] F. W. Angelotti, Modeling for Structured System Interconnect Test, IEEE International Test Conference, 1994, pp. 127-133.

[11] K. Arabi, B. Kaminska, Oscillation-Test Strategy for Analog and Mixed-Signal Integrated Circuits, VLSI Test Symposium, 1996, pp. 476-482.

[12] K. Arabi, B. Kaminska, Testing Analog and Mixed-Signal Integrated Circuits Using Oscillation-Test Method, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, Issue 7, July 1997, pp. 745-753.

[13] John Arena, Open Strategy Tools Synthesize Better Defect Detection between Multiple Inspection & Test Systems, Teradyne Assembly and Test Division.

[14] John J. Arena, Rehosting Legacy Test Program Sets from Military ATE, Assembly Test Division Teradyne Inc., Boston, MA 02118.

[15] J. R. Armstrong, Chip-Level Modeling with VHDL, Englewood Cliffs, NJ: Prentice-Hall, 1989, 148 p. ISBN 0131331906. TK7874.A75.

[16] James R. Armstrong, Chip-Level Modeling with VHDL, Virginia Polytechnic Institute and State University.

[17] P. J. Ashenden, The Designer's Guide to VHDL, Morgan Kaufmann, 688 p. ISBN 1-55860-270-4. TK7888.3.A863.

[18] Peter J. Ashenden, The VHDL Cookbook, Dept. Computer Science, University of Adelaide, South Australia.

[19] AT&T. AT&T Boundary-Scan Master Manual. AT&T, 1993.

[20] J. Baston, In-Circuit Testing, Van Nostrand Reinhold, New York, 1985.

[21] R. G. Bennetts, DFT Technology Backgrounders, IEEE 1149.4-99 Mixed-Signal Test Bus Standard, 2001.

[22] R. G. Bennetts, Guidelines for Board Design For Test (DFT) Based on Boundary Scan, Part1&Part2, 2003.

[23] R. G. Bennetts, Guidelines for Chip Design For Test (DFT) Based on Boundary Scan, 2003.

[24] R. G. Bennetts, Boundary Scan Tutorial, Tutorial, 2002.

[25] Harry Blender, Boundary-Scan Test - A Practical Approach, Kluwer Academic Publishers, 1993.

[26] Gary J. Boutin, Flying Probe: An Integral Test Method, Circuits Assembly Magazine, August 2002.

[27] Brian Hatt Test from Systems Operation Raytheon Systems Company and Charles P. Pothier from Assembly Test Division Teradyne Inc., Raytheon Beta Site Evaluation of Teradyne's PC-Based VXI Functional Test Solution.

[28] Brian R. Wilkins, An Introduction to IEEE-1149.4 Analog and Mixed-Signal Boundary-Scan Test, Department of Electronics and Computer Science, University of Southampton, Southampton, England, Chip Center, 2001.

[29] Margaret Cadogan, Teresa Lopes, Specifying an IVI Class for Digital Test Instrumentation, Teradyne Inc.

[30] Cisco Systems, Cisco Input Test Buffer Design Examples for AC Boundary-Scan, 2002.

[31] Cisco Systems, AC Boundary-scan Specification for IEEE, 2001.

[32] C.-H. Chiang, S. K. Gupta, BIST TPGs for Faults in Board Level Interconnect via Boundary Scan, In IEEE VLSI Test Symposium, 1997, pp. 376-382, T. S. Corporation, Tundra Universe User Manual.

[33] Chen-Huan Chiang, Sandeep K. Gupta, BIST TPG for Faults in System Backplanes. Electrical Engineering - Systems University of Southern California, Los Angeles, CA 90089-2562.

[34] Chipcenter. An Introduction to IEEE-1149.4 Analog and Mixed-Signal Boundary-Scan Test. http://www.chipcenter.com.

[35] Corelis. JTAG Controller Communicates With Windows PC Across Parallel Port. http://www.corelis.com.

[36] Corelis.NetICE™, LAN-based JTAG/ROM Emulator. http://www.corelis.com.

[37] Corelis. PCI-1149.1/Turbo™ High-Performance Boundary-Scan Test and In-System Programming Controller. http://www.corelis.com.

[38] Corelis. PIO-1149.1/E. Parallel Port Boundary-Scan Controller. ScanPlus Flash™ In-Circuit JTAG Flash Programming Tool. http://www.corelis.com.

[39] Corelis. SCANIO™-280; Digital I/O Module. http://www.corelis.com.

[40] Corelis. ScanPlus Debugger. Boundary-Scan Test Coverage Analysis Tool. http://www.corelis.com.

[41] Corelis. ScanPlus DFT Analyzer. Boundary-Scan Test Coverage Analysis Tool. http://www.corelis.com.

[42] Corelis. ScanPlus Flash™. In-Circuit JTAG Flash Programming Tool. http://www.corelis.com.

[43] Corelis. ScanPlus Merge™. System-Level Test Productivity Tool. http://www.corelis.com.

[44] Corelis. ScanPlus Overview. Boundary Test System. http://www.corelis.com.

[45] Corelis. ScanPlus Runner EDO. Boundary-Scan Test Coverage Analysis Tool. http://www.corelis.com.

[46] Cyril Drancourt, Introduction au JTAG, 2003

[47] Cypress. Using IEEE 1149.1 Boundary Scan (JTAG) with Cypress, Ultra37000™ CPLDs, 1998. http://www.cypress.com.

[48] Jim Davies, The Test of Time - Special Report - Case Study - ATE Specification, New Electronics, 12 December 2000.

[49] P. Demian, A. Gontean, 50 MHz PC Coupled Logic Analyser, Buletinul Ştiinţific şi Tehnic al Univ. Tehnice Timişoara, Tom 39(53), 1994, pag. 171-174.

[50] P. Demian, A. Gontean, Including PLDs in Logic Analyser Design Speeds Up Scheme and Cuts Costs, Sesiunea de comunicări cu participare internaţională OPTIM '94, Universitatea "Transilvania" Braşov, 12-14 mai 1994, pp. 183-186.

[51] P. Demian, A. Gontean, Implementing a PC Data Acquisition System with the MAX 165 Circuit, Proceedings of the Symposium on Electronics and Telecomunications Timişoara, 29-30 sept. 1994, Volume III, pag. 217-220.

[52] P. Demian, A. Gontean, Low Cost Logic Analyser, Sesiunea de comunicări cu participare internațională OPTIM '94, Universitatea "Transilvania" Braşov, 12-14 mai 1994, pp. 219-222.

[53] P. Demian, A. Gontean, Loadable Binary Counters using XC 3100, Proccedings of the Symposion on Electronics and Telecommunications, Timişoara, 26-27 sept 1996, vol. III, pag. 64-67.

[54] P. Demian, A. Gontean, Microcontroller Based IC Tester, Sesiunea de comunicări cu participare internațională SCS'93, Universitatea Tehnică Iaşi, 4-5 noiembrie 1993, pp. 398-400.

[55] P. Demian, A. Gontean, Software Enviroment for IC Testing, Sesiunea de comunicări SCS'93, Universitatea Tehnică Iaşi, 4-5 noiembrie 1993, pp. 401-404.

[56] P. Demian, A. Gontean, TMS320C203 VHDL System Modelling , Programmable Devices and Systems, A Proceedings volume from the IFAC Workshop, 8-9 February 2000, Ostrava, Czech Republic, IFAC – International Federation of Automatic Control, Pergamon, Elsevier Science Ltd., Oxford, ISBN 0-08-043620-X, pag. 81-84.

[57] P. Demian, A. Gontean, Using a Microcontroller for Logic Testing, Buletinul Ştiințific şi Tehnic al Universității Politehnice Timişoara, Tom 38 (52), 1993, pp. 171-176.

[58] P. Demian, A. Gontean, C. Alexandrescu, Principiul testării pe frontieră aplicat sistemelor numerice, Proceedings of the Scientific Communications Meeting of "Aurel Vlaicu" University, Arad, 16-17 mai 1996, vol. 8, pp. 253-256.

[59] Mike Dewey, Implementing Production Test Systems with the PXI Architecture, GenRad Inc.

[60] DLI Company. Boundary Scan Test Equipment. http://www.dli.de/products/embedded_debug_tools/boundary_scan_test.html.

[61] Magnus Eckersand, Using At-Speed BIST to Test LVDS Serializer/Deserializer Function, National Semiconductor/Sweden. http://www.national.com/scan.

[62] ERICSSON, IEEE 1149.4 Mixed-Signal Test Bus Standard, 2002.

[63] John Evans, Integration of DFT into RASSP, Lockheed Martin Advanced Technology Laboratories.

[64] Billy Fenton, Using Microprocessor and DSP Debug Interfaces for Manufacturing Functional Test and Diagnosis, eTronix Conference (www.etronixexpo.com) February 25 - March 1, 2001 at Anaheim, CA.

[65] Fluke and Philips. The ABC of Boundary-Scan Test.

[66] GenRad Inc. GPX621 GenRad's High-Performance, Affordable Test System Mainframe. http://www.genrad.com.

[67] GenRad Inc. GR Navigate™. Integrated Software Development Environment. http://www.genrad.com.

[68] GenRad Inc. GR Pilot, Fixtureless Flying Probe Test System. http://www.genrad.com.

[69] GenRad Inc. GR Versa™Cell Phone Test Solution. Instrument Independent Solution for Production Test at the End-of-line. http://www.genrad.com.

[70] GenRad Inc. Today's Technology Changing with the Future: GR TestStation. http://www.genrad.com.

[71] GenRad Inc. TS 12X - Production Test System. http://www.genrad.com.

[72] GenRad Inc. VERSA Functional Test Applications in Automotive Systems. http:// www.genrad-europe.com.

[73] GenRad Inc. VIPER Electrical. GenRad's Comprehensive Inspection System Solution Delivers Higher Performance at a Lower Cost.

[74] Victor Golovtsov, Lev Uvarov, Track Finder Crate: Configuration and Testing, July 2002.

[75] A. Gontean, P. Demian, Using Boundary Scan Test for High Tech Industrial Equipment, Programmable Devices and Systems, A Proceedings volume from the IFAC Workshop, 8-9 February 2000, Ostrava, Czech Republic, IFAC – International Federation of Automatic Control, Pergamon, Elsevier Science Ltd., Oxford, ISBN 0-08-043620-X, pag. 269-273.

[76] O. F. Haberl, T. Kropf, Self Testable Boards with Standard IEEE 1149.5, Module Test and Maintenance (MTM) Bus Interface, In Proceedings European, Design and Test Conference, 1994, pp. 220-225.

[77] Peter Hansen, Boundary Scan Will Also Improve the Design Process, Penton Publishing, 1997.

[78] P. Hansen, Hierarchical Design: Taking Boundary Scan to the Next Level, Proceedings of IEEE AUTOTESTCON, 1996.

[79] P. Hansen, Taking Advantage of Boundary Scan in Loaded-Board Testing. The Test Access Port and Boundary-Scan Architecture (eds. C. M. Maunder and R. E. Tulloss), IEEE, 1990, p. 81.

[80] P. Hansen, The Web Leads a Revolution in ATE Programming Environments, Proceedings of IEEE AUTOTESTCON, 1997.

[81] Peter Hansen, The World Wide Web Leads a Revolution in ATE Programming Environments, Teradyne Inc.

[82] Harry Jim, Using Boundary-Scan Test to Find Structural Faults at the Board Level Design & Test Expo, Dallas, 1993.

[83] Hewlett-Packard. Design for Testability Using Boundary-Scan 1149. Application Note 1210-7.

[84] J. Andrew Hutchinson, Advantages of Digital Convergence for Functional Test. Teradyne Inc., 179 Lincoln St., Boston, MA. 02111.

[85] IEEE Std. 1149.4 Mixed Signal Working Group. The Test Technology Technical Council IEEE Computer Society. www.ee.ncu.edu.tw/~ccsu,

[86] IEEE. A Standard Module Test and Maintenance Bus. IEEE Standard Protocol 1149.5.

[87] IEEE. High Speed Serial Bus. IEEE P1394.

[88] IEEE. Products and Projects Status Report, 2004,

[89] EEE. P1149.5. Standard Module Test and Maintenance Bus Protocol.

[90] IEEE P1394. High Speed Serial Bus.

[91] IEEE Standard 1149.5. ISBN 1-55937-558-2.

[92] IEEE Standard Module Test and Maintenance (MTM) Bus Protocol. IEEE 1995.

[93] IEEE Standard Test Access Port and Boundary-Scan Architecture. IEEE Std 1149.1-1990 (Includes IEEE Std 1149.1a-1993). IEEE Standards Board, 345 East, 47th Street, New York, NY, October 1993.

[94] IEEE Supplement to IEEE Std 1149.1-1990. IEEE Standard Test Access Port and Boundary-Scan Architecture. IEEE Std 1149.1b-1994. IEEE Standards Board, 345 East 47th Street, New York, NY, March 1995.

[95] C. Jeffrey, A. Lechner & A. Richardson, Online Monitoring for Automotive Sub-systems Using 1149.4, Centre for Microsystems Engineering, Lancaster University,

[96] Simon Jones, Open Standards-Based Software Tools Optimize the PCB Manufacturing Enterprise.

[97] JTAG Technologies. Assistance and Maintenance. http://www.jtag.com.

[98] JTAG Technologies. Boundary-Scan Training, Application, Assistance and Maintenance. http://www.jtag.com.

[99] JTAG Technologies. BSDL 512 Verifyer. http://www.jtag.com.

[100] JTAG Technologies. Engineering, Manufacturing and Field Service. Test Application Development. http://www.jtag.com.

[101] JTAG Technologies. Extended I/O Scan System (XIOS 512). http:// www.jtag.com.

[102] JTAG Technologies. Hierarchical Boundary-Scan - A Scan Chip-Set Solution. November 16, 2001. http://www.jtag.com.

[103] JTAG Technologies. In-System Flash Programming Using Boundary-Scan. In System Flash Programming Tools. http://www.jtag.com.

[104] JTAG Technologies. In-System PLD Programming Using Boundary-Scan. http://www.jtag.com.

[105] JTAG Technologies. JT 2122/168 and JT 2122/F168 DIMM 128, Digital I/O Modules for Parallel Access. http://www.jtag.com.

[106] JTAG Technologies. JT 2135 TAP Extender™. Overcomes Cable Length Restrictions. http:// www.jtag.com.

[108] JTAG Technologies. JT 3710/PXI DataBlaster™. CompactPCI/PXI Boundary-Scan Controller for Testing Flash Memory and PLD on-Board Programming. http://www.jtag.com.

[109] JTAG Technologies. JTAG Support Services Boundary-Scan Training. http://www.jtag.com.

[110] JTAG Technologies. JTAG Visualizer. Graphical Boundary-Scan Representation and Exploration. http://www.jtag.com.

[111] JTAG Technologies. Liberez la puisance du Boundary-Scan. Solutions pour le test et la programation au moyen de la norme IEEE 1149.1 Boundary-Scan. JTAG Brochure-France. http://www.jtag.com.

[112] JTAG Technologies. Manufacturing Test and on-Board Programming Production Station Software Packages. http://www.jtag.com.

[113] JTAG Technologies. Massively-Parallel Programming and Board Testing via Boundary-Scan. January 22, 2002. http: //www.jtag.com.

[114] JTAG Technologies. NI Test Platform Integrates with Boundary-Scan Technology to Lower Cost of Test, National Instruments. February 19, 2002. http://www.jtag.com.

[115] JTAG Technologies. Some Suggested Design Provisions for Boundary-Scan Test & ISP Implementations. JTAG_design_hints. http://www.jtag.com.

[116] JTAG Technologies. Unlock the Power of Boundary-Scan. http://www.jtag.com.

[117] A.G. Kapsch, Customer Case Study, Austria, February 2001.

[118] K. T. Komegay, K. Roy, Integrated Test Solutions and Test Economics for MCMs, Proceedings of IEEE International Test Conference, 1995, pp. 193-201.

[119] D. Landis, Applications of IEEE 1149.5 Module Test and Maintenance Bus, IEEE International Test Conference, 1992.

[120] J. H. Lau, Handbook of Fine Pick Sourface Mount Technology, Van Nostrand Reinhold, New York, 1994.

[121] J.-C. Lien, M. A. Breuer, Maximal Diagnosis for Wiring Networks, In Proceedings IEEE International Test Conference, 1991, pp. 96-105.

[122] J. M. Martins Ferreira, Ricardo J. Costa, Gustavo R. Alves, Martyn Cooper, The Pearl Digital Electronic Lab: Full Access to the Workbench via the Web.

[123] Colin M. Maunder, Integrating Internal Scan Paths, Los Alamitos, 1990.

[124] Pat McHugh, IEEE Test Technology Standards. Presentation Slides Peter Flaming Application of IEEE Std 1149.1: An Overview, Los Alamitos, 1990.

[125] Brian Miller, Reliance on Worst-Case Timing. Electronic Engineering Times, March 1992.

[126] National Semiconductor. SCAN Data Book. 2002.

[127] National Semiconductor. SCAN92LV090, 9 Channel Bus LVDS Transceiver with 1149.1 Access. http://www.national.com/scan.

[128] National Semiconductor. SCAN921023/1224, 20 MHz-66 MHz 10-Bit SerDes with IEEE 1149.1 and At-Speed BIST Test Modes. http://www.national.com/scan.

[129] National Semiconductor. SCANSTA111, Multidrop Addressable IEEE 1149.1 (JTAG) Multiplexer. http://www.national.com/scan.

[130] A. Osseiran, Analog & Mixed-Signal Boundary Scan: a Guide to the 1149.4 Test Standard, Kluwer, 1999. http://ww.wkap.nl.

[131] Dan Pitica, Testarea echipamentelor electronice, Curs anul V Electronică Aplicată

[132] Kenneth P. Parker, The Boundary-Scan Handbook, Kluwer Academic Publishers, Boston, 1998.

[133] K. Parker, Boundary-Scan Handbook: Analog & Digital, Kluwer, 1998, cap. 7.

[134] Perry, Fundamentals of Mixed-Signal Test, 1999. http:www.soft-test.com.

[135] C. Poirier, IEEE 1149.5 to 1149.1 Data and Protocol Conversion, Proceedings IEEE International Test Conference, 1991, pp. 96-105.

[136] C. Poirier, IEEE P1149.5 to 1149.1 Data and Protocol, Proceedings of IEEE International Test Conference, 1993, pp. 527-535.

[137] R. P. Prasad, Surface Mount Technology - Principles and Practice, Van Nostrand Reinhold, New York, 1994.

[138] PXI Specification, Revision 2.0.

[139] Jason Rayfield, A Testing Time for Contract Manufacture, Teradyne. Test, May 2001.

[140] M. Renovell, F. Azais, Y. Bertrand, Optimized Implementations of the Multi-Configuration DFT Technique for Analog Circuits, Design, Automation and Test in Europe, 1998, pp. 815-821.

[141] T. Riedel, A. Ambler, M. Wahl, Life Cycle Cost Analysis, AUTOTEST-CON, Salt Lake City, 1998.

[142] T. Riedel, M. Wahl, A. Ambler, Cost Analysis Linking Design, Test & Maintenance, European Test Workshop, Constance, 1999.

[143] T. Riedel, M. Wahl, A. Ambler, Design for Testability and Maintenance for Long Lasting Systems, WDTA, Dubrovnik,1999.

[144] Dave Rolince, Applying Virtual Test Principles to Digital Test Program Development, Teradyne Inc.

[145] David Rolince, Extend the Frontiers of Boundary-Scan Test, Test & Measurement World, February 2001.

[146] David Rolince, Simplifying TPS Development and Execution Using a PC, Web-Based Environment. Teradyne Inc. http://www.teradyne.com.

[147] Stan Runyon, ATE Environment Opens, Electronic Engineering EETIMES, June 1998.

[148] M. Sidiropulos, V. Stopjakova, H. Manhaeve, Implementation of a BIC Monitor in a New Analog BIST Structure, IEEE International Workshop on IDDQ Testing, 1996, pp. 59-63.

[149] Kevin Skahill, VHDL for Programmable Logic, Cypress Semiconductor.

[150] Michael J. Smith, Step by Step, Step 8: Test/Inspection, Surface Mount Technology, September 2002 edition.

[151] M. Soma, V. Kolarik, A Design-for-Test Technique for Switched-Capacitors Filters, IEEE European Design & Test Conference, Paris, March 1994, pp. 42-47.

[152] J. Sosnowski, M. Hankus, Using IEEE 1149.1 Interface in Systems with FPGAS, IFAC - Programmable Devices and Systems (PDS 2000), 2000.

[153] *** Standard Atlas Subset for Modular Testing, Arinc Specification 626-1., Maryland, USA.

[154] Bob Stasonis, Functional Test in a High Density PCB Environment, GenRad Inc.

[155] Bob Stasonis, Use PXI to its Fullest, http://www.genrad.com.

[156] Bob Stasonis, Kelly Darach, Open Standards Show Promise for the Test & Measurement and Automation Industries, http://www.pxisa.org/.

[157] Phillip Stern, High-Performance Component Software Changes the Rules for Configuring ATE, Assembly Test Division Teradyne Inc., 179 Lincoln Street, Boston, MA 02111.

[158] L. Storey, C. Lapihuska, E. Atwood, L. Su, A Test Methodology to Support an ASEM MCM Foundry, Proceedings of IEEE International Test Conference, 1994, pp. 426-435.

[159] C. Su, S.-J. Jou, Y.-T. Ting, Decentralized BIST for 1149.1 and 1149.5 Based Interconnects, In Proceedings European Design and Test Conference, 1996.

[160] Bernard Sutton, The Functional Test Reviva, GenRad Europe Inc.

[161] Siemens VDO. BSW training, End of Line testing, 2001

[162] Siemens VDO. CANoe training, 2001

[163] Siemens VDO. KWP2000 specification, 2002

[164] Siemens VDO. MPECU Software Requirement Specification, 2002

[165] Siemens VDO-BSK. Aida basics courses, 2003

[166] Siemens VDO. Portable Object Oriented Language, Tutorial 1-3, 2003

[167] Siemens VDO. Automatic Testing Process, Prezentarea implementării realizată în grupul I IP General Motor, 2003.

[168] Siemens VDO. Automatic Testing Process Using SVT. Prezentarea propunerii pentru testarea automata a afişajului Opel MY 4.5, 2003.

[170] Siemens VDO, AISeq-HW-Measurement. Studiu de fezabilitate asupra soluțiilor HW de interfațare cu mediul de testare AISeq, 2004.

[171] Siemens VDO. Automatic testing meeting minutes, Repoartele întâlnirilor grupul de automatic testing din departamentul I IP, 2004.

[172] Siemens VDO. Automatic testing tool requirement. Analiza necesităților de testare a instrumentelor de bord, 2004.

[173] Siemens VDO. Audit software. Mediu de verificare a instrumentelor de bord după producție, 2004.

[174] SYNATRON, Mixed-Signal Boundary-Scan Evaluation System, 2004.

[175] Tecwings. Customer Case Study. Austria, December 2001.

[176] Tektronix. Integrated Debug of Embedded Systems with the TLA700 Series Logic Analyzers. http://www.tektronix.com.

[177] Tektronix. Serial Vector Format Specification.

[178] Tektronix. VX4491 Serial Test Module.

[179] Teradyne. Circuit Board, Tests and Inspection. http://www.teradyne.com/prods/cbt/products/prod_lib.html.

[180] Teradyne Inc. Digital Test Failure Data Language.VICTORY 2.30 Reference Manual, 1996.

[181] Teradyne. D2B Software. Test & Inspection. http://www.teradyne.com/prods/cbt/products/soft-d2b/prod_soft-d2b.html

[182] Teradyne. Electronics Manufacture & Test - Case Study. April 2002. http:// www.teradyne.com.

[183] Teradyne. Ericsson Creates Innovative Test Solutions with the Help of Teradyne. TeraView, July 1999, Teradyne's Quarterly Newsletter. http://www.teradyne.com.

[184] Teradyne. Fully Integrated Test and Handling System - Spectrum In-Line. http://www.teradyne.com, http:// www.pematech-rohwedder.com.

[185] Teradyne. GENEVA Functional Test Solution Teradyne's VXI-Based Test and Measurement System. http:// www.teradyne.com/cbt.

[186] Teradyne. GR Pilot LX™ Flying Prober - Fixtureless Production Test Systems. http:// www.teradyne.com.

[187] Teradyne. GR Test Manager - Full Featured Test Development and Execution Environment for the GENEVA and GR Versa Functional Test Platforms. http:// www.teradyne.com/cbt.

[188] Teradyne. Implementing an Integrated Test Strategy - Teradyne Global Case Study. http:// www.teradyne.com.

[189] Teradyne. LASAR V6.60 Simulation System - The Defense Industry Standard for Digital TPS Development. http:// www.teradyne.com.

[190] Teradyne. Lightning ZTM Flash Programming Package High-Speed Flash Programming Option for Z1800-Series TM Manufacturing Process Test Systems.

[191] Teradyne. Multi-Channel. Multi-Protocol Support for Automotive ECU Manufacturers. http:// www.teradyne.com/cbti.

[192] Teradyne. MultiScan. Teradyne's Vectorless Test System.

[193] Teradyne. M9-Series of VXI Digital Test Instruments Versatile C-Size VXI Instruments for High-Performance Digital Testing. http:// www.teradyne.com.

[194] Teradyne. Pilot Adds to Service Flexibility, Teradyne Global Case Study. http://www.teradyne.com.

[195] Teradyne. PRISM-Z™ Analog Measurement System for Z1800-Series Test Systems.

[196] Teradyne. Spectrum TM 8800-Series. Open Architecture for Test Strategy Flexibility. http:// www.teradyne.com.

[197] Teradyne. Spectrum™ 8911/8912 Functional Test Platform - Low-Cost Solutions for Functional, Applications - Specific Test Requirements. http:// www.teradyne.com.

[198] Teradyne. Summit 1100 & 1100HR - High-Performance, Semi-Automatic Rework Systems. http:// www.teradyne.com/cbti.

[199] Teradyne. Teradyne GRPX1040 Compact PCI C-Band Optical Amplifier. http://www.teradyne.com/prods/cbt.

[200] Teradyne. Test & Inspection. Technology Summary.

[201] Teradyne. TestStation SE™ in-Circuit Test System. http:// www.teradyne.com/cbti.

[202] Teradyne. TestStudio™ Web-Based ATE Operating Environment. http:// www.teradyne.com.

[203] Teradyne. VICTORY Boundary-Scan Test Software, 2002.

[204] Teradyne. Z1803. Integrating Low Cost with High Fault Coverage for Advanced Technology Boards.

[205] Teradyne. Z1888 Manufacturing Process Test System.

[206] Texas Instruments. A Look at Boundary Scan from a Designer's Perspective. Baltimore, Maryland, October 1994.

[207] Texas Instruments. ASSET Technical Overview Data Sheet.

[208] Texas Instruments. Digital Bus Monitor, 2002.

[209] Texas Instruments. GPL Scope Products, 1997.

[210] Texas Instruments. IEEE 1149.1 JTAG Testability, 1997.

[211] Texas Instruments. JTAG/IEEE 1149.1 Design Considerations, 1990.

[212] Texas Instruments. JTAG Tutorial - Seminar Part 1.

[213] Texas Instruments. JTAG Tutorial - Seminar Part 2.

[214] Texas Instruments. System Testability Using Standard Logic, 1996.

[215] Texas Instruments. TMS230C2XX, Advance Information, 1995.

[216] Texas Instruments Inc. and Teradyne. Serial Vector Format Specification, Revision C. September 1994.

[217] *** TYX Training Course 6005-D ARINC-626. ATLAS Programming Guide Book.

[218] D. Vazquez, J. L. Huertas, A. Rueda, Reducing the Impact of DFT on the Performance of Analog Integrated Circuits: Improved SW-OP AMP Design, VLSI Test Symposium, 1996, pp. 42-47.

[219] **** VHDL for Programmable Logic. http://cseng.awl.com

[220] *** VHDL PaceMaker Software.

[221] VXIbus & PXI Newsletter, vol. 12, Issue 1, November 2000.

[222] VXIbus Standard, System Specification 1.4.

[223] P.T. Wagner, Interconnect Testing with Boundary Scan, In Proceedings IEEE International Test Conference, 1987, pp. 52-57.

[224] Michael G. Wahl*, Christoph Maaß*, Tony Ambler, Model Based Cost Optimization for Engineering and Management.

[225] D.E. Wedge, Simulation, Fault Dictionaries and Boundary Scan: Testing and Diagnosing Logic.

[226] G. Wedge, Using Boundary Scan with a Fault Dictionary to Test and Diagnose Clusters of Non-Scan Logic, Proceedings of IEEE AUTOTESTCON, 1996.

[227] Gene Wedge, Tom Conner, A Roadmap for Boundary-Scan Test Re. Victory Software Development, Teradyne Inc. International Test Conference, October 1996.

[228] S. Wei, P. K. Nag, R. D. Blanton, A. Gattiker, W. Maly, To DFT or not to DFT? IEEE ITC 1997, Washington, 1997, pp. 557-566.

[229] L. Whetsel, A Proposed Method of Accessing 1149.1 in a Backplane Environment, In Proceedings IEEE International Test Conference, 1992, pp. 206-216.

[230] L. Whetsel, Hierarchically Accessing 1149.1 Applications in a System Environment, Proceedings of IEEE International Test Conference, 1993, pp. 517-526.

[231] Lee Whetsel, Hierarchically Accessing 1149.1 Applications in a System Environment, Texas Instruments, Baltimore, Maryland, October 1993.

[232] Brian R. Wilkins, An Introduction to IEEE-1149.4 Analog and Mixed-Signal Boundary-Scan Test, Department of Electronics and Computer Science University of Southampton, Southampton, England.

[233] Randy Zeger, Pat Annese, Margaret Cadogan, Integrating COTS VXI Hardware and Software for the Marine Corps Third Echelon Test System, ManTech Systems Engineering Corp & Teradyne Inc.

Adrese pe Internet

[234] Actel, http://www.actel.com.

[235] Acculogic, http://www.acculogic.com

[236] Advanced Micro Devices, http://www.amd.com.

[237] Aldec Inc., http://www.aldec.com.

[238] Altera, http://www.altera.com.

[239] Asset Inter Tech, http://www.asse-intertech.com.

[240] InterIlitech, http://www.intellitech.com

[241] Corelis, http://www.corelis.com.

[242] Cypress, http://www.cypress.com.

[243] Data I/O, http://www.dataio.com.

[244] GenRad Inc., http://www.genrad.com.

[245] JTAG Technologies, http://jtag.com

[246] Lattice Semiconductor, http://www.lattice.com

[247] National Instruments, http://www.ni.com.

[248] SGS-Thomson Microelectronics, http://www.st.com

[249] Tektronix, http://www.tecktronix.com.

[250] Teradyne, http://www.teradyne.com.

[251] Texas Instruments, http://www.ti.com.

[252] Vector, http://www.vector-informatik.com.

[253] Viewlogic Systems, http://www.viewlogic.com.

[254] Xilinx, http://www.xilinx.com.