

**CATEDRA: PROIECT LA STRUCTURA SISTEMELOR DE CALCUL**

**TITLU PROIECT: CIRCUITE DE INMULTIRE ZECIMALA**

**NUME STUDENT: BORDEIANU EMANUEL**

**NUMĂR GRUPĂ: 30235**

**NUME ÎNDRUMĂTOR DE PROIECT: ALEXANDRU DAN BUTIRI**

**DATA: 25.12.2020**

**Cuprins**

Introducere .................................................................. 3

Fundamentare teoretică ………………………….….. 5

Proiectare şi implementare ………………………….. 7

Rezultate experimentale …………………………….. 10

Concluzii ……………………………………………. 17

Rezumat ....................................................................... 18

Bibliografie .................................................................. 19

Anexa A ....................................................................... 20

Anexa B ....................................................................... 30

Anexa C ....................................................................... 51

**Introducere**

Proiectul este implementat în limbajul VHDL, iar, ca resursă hardware, am folosit plăcuța de dezvolare Nexys4DDR pe care vom testa rezultatele înmulțirii. Totuși, proiectul poate fi testat fără probleme și pe alte placuțe de dezvoltare care au cel putin 16 switch-uri si 8 afișoare de câte 7 segmente. Un exemplu de astfel de plăcuță este Basys3 din familia Artix7.

Un multiplicator binar este un circuit electronic utilizat în electronica digitală, cum ar fi un computer, pentru a multiplica două numere binare. Este construit folosind sumatoare binare. O varietate de tehnici de calcul aritmetic poate fi utilizată pentru a implementa un multiplicator digital. Majoritatea tehnicilor implică calculul unui set de produse parțiale și apoi însumarea produselor parțiale împreună. Microprocesoarele timpurii nu aveau nici o instrucțiune de multiplicare. Pe măsură ce mai mulți tranzistoare pe cip au devenit disponibile datorită integrării la scară mai mare, a devenit posibilă punerea unui număr suficient de sumatoare pe un singur cip pentru a însuma toate produsele parțiale simultan, mai degrabă decât reutilizarea unui singur sumator pentru a gestiona fiecare produs parțial unul câte unul.

Problema pe care acest proiect o rezolvă este aceea a circuitelor de înmulțire a numerelor zecimale. O reprezentare a numărului întreg fara semn va fi utilizată pentru toate operațiunile întregi din operațiunile noastre. În funcție de shift-ări și adunări, structurile de multiplicare întregi sunt împărțite in urmatoarele categorii:

1. Ripple Carry Array multiplier (RCA);

2. Right Shift Array multiplier (RSA);

3. Carry Save Array multiplier (CSA);

4. Braun Array multiplier (BA);

5. Wallace Tree architecture (WT).

Primele patru se bazează pe combinația de deplasari și adunări de produse parțiale. Diferențele structurale majore sunt propagarea biților de transport și dispunerea celulelor, care pot fi fie o deplasare la stânga, fie o structură de deplasare la dreapta. Arborele Wallace are o structură similară, dar în loc să adauge produsele parțiale rând cu rând, folosește o structură arborescentă echilibrată pentru a obține ceva paralelism și a reduce lungimea drumului critic. Arhitectura BA (Braun Array multiplier) este o modificare ceva mai optimizată a structurii CSA (Carry Save Array multiplier). Aceste diferențe structurale vor fi importante pentru transformarea ulterioară în multiplicatori reconfigurabili.

Wallace tree este o implementare hardware eficientă a unui circuit digital care înmulțește două numere întregi. Acest circuit de înmulțire este foarte eficient, din punct de vedere al timpului în care se execută operațiile până la obținerea produsului final. Un dezavantaj al acestui circuit este acela că are o structură destul de complexă. De exemplu, acesta nu poate face trecerea rapidă de la un circuit de 8 biți la unul pe 16 biți, din pricina structurii arborescente pe care acesta o are. Algoritmul de înmulțire Wallace Tree are trei pași:

înmulțeste fiecare bit al unuia dintre argumente, cu fiecare bit al celuilalt;

reduce numărul de produse parțiale la două cu straturi de adunatoare complete și jumătate;

grupeaza firele în două numere și adăugați-le cu un sumator convențional.

În comparație cu adăugarea naivă a produselor parțiale cu sumatoare obișnuite, beneficiul arborelui Wallace este viteza mai mare. Are straturi de reducere O (log n), dar fiecare strat are doar O (1) întârziere de propagare. O adăugare naivă de produse parțiale ar necesita timp O (log n)^2. Dintr-o perspectivă teoretică a complexității, algoritmul Wallace tree plasează multiplicarea în clasa NC. Dezavantajul Wallace tree, în comparație cu adăugarea naivă de produse parțiale, este numărul de porți mult mai mare. Aceste calcule iau în considerare doar întârzierile de poartă și nu se ocupă de întârzierile prin cablu, care pot fi, de asemenea, foarte substanțiale. Wallace tree poate fi, de asemenea, reprezentat de un copac de 3/2 sau 4/2 addere. Uneori este combinat cu codificarea Booth.

Înmulțitorul care folosește Carry Propagate Adder efectuează operațiile de înmulțire similar cu metoda de înmulțire predată în școli. Ideea este de a se crea 16 vectori de 32 de biti fiecare. Vectorii vor lua valori la fel ca vectorii rezultați în urma înmulțirii fiecărui bit al deînmulțitului cu cel al înmulțitorului. La scrierea de cod, ideea este de a deplasa spre stanga deînmulțitul dacă bitul de la un anumit index este 1. Dacă bitul de la un oarecare index este 0, atunci tot acel vector va fi nul (echivalent înmulțirii cu zero). După aceea, se adună primul vector cu al doilea. Rezultatul adunării se adună cu al treilea vector. Rezultatul adunării se adună cu al patrulea vector șamd. În final, suma rezultată va reprezenta produsul celor 2 numere. Deși proiectul prevede înmulțirea numerelor cu maximum 4 cifre, ceea ce ar însemna că avem nevoie de cel puțin 14 biți, din motive de simetrie, am considerat că este mai bine să implementez proiectul pe 16 biti.

În următoarele secțiuni ale raportului vom prezenta fundamentarea teoretică - în care se specifică cum sunt modelele, metode și tehnologii care pot fi utilizate; proiectare și implementare - care reprezintă partea principală a raportului proiectului conținând etapele parcurse pentru realizarea obiectivelor proiectului; rezultate experimentale – in care se demonstrează că rezultatele obținute sunt valide; concluzii – sumarul raportului; bibliografie.

**Fundamentare teoretică**

Proiectul constă în alcătuirea unor circuite de înmulțire zecimală a două numere. Exemplu de astfel de circuite/metode de înmulțire sunt Wallace Tree, înmulțirea matriceală, înmulțirea într-o bază superioară, Tehnica de înmulțire Booth, înmulțirea prin deplasare și adunare, înmulțirea care folosește Carry Propagate Adder etc. Fiecare dintre aceste metode de înmulțire are diverse avantaje și dezavantaje. Cu toate acestea, metodele de înmulțire pe care le-am implementat sunt Wallace Tree și înmulțire folosind Carry Propagate Adder. Fiecare dintre aceste circuite de înmulțire este realizat pe 16 biți.

Wallace tree are trei pași:

Ideea de baza este aceea de a înmulți fiecare bit al deînmulțitului cu fiecare bit al celuilalt operand, adică înmulțitorul. Reducem numărul de produse parțiale la două cu straturi de adunatoare complete și jumătate. Grupăm “firele”, adică rezultatele parțiale, în două numere și le adăugăm cu un sumator convențional.

Pentru a înțelege mai limpede pașii descriși mai sus vom adăuga câteva informații suplimentare. Primul pas, așa cum s-a spus mai sus, este multiplicarea fiecărui bit dintr-un număr cu fiecare bit al celuilalt, ceea ce se realizează ca o poartă ȘI, rezultând n ^ 2 biți. În al doilea pas, biții rezultați sunt reduși la două numere; acest lucru se realizează după cum urmează: atât timp cât există trei sau mai multe “fire” cu aceeași pondere, folosim următoarea convenție: Selectăm trei fire cu aceleași ponderi și le introducem într-un sumator complet. Rezultatul va fi un fir de ieșire cu aceeași greutate și un fir de ieșire cu o pondere mai mare pentru fiecare trei fire de intrare.

Dacă au mai rămas două fire de aceeași pondere, le introducem într-un semi-sumator.

Dacă rămâne doar un fir, îl conectăm la următorul strat.

Exemplu de înmulțire Wallace Tree:

1 1 1 0 0 1

1 1 0 1 0 1 (\*)

1 1 1 0 0 1 (PP0) – Numărul produsului partial

0 0 0 0 0 0 (PP1)

1 1 1 0 0 1 (PP2)

0 0 0 0 0 0 (PP3)

1 1 1 0 0 1 (PP4)

1 1 1 0 0 1 (PP5)

1 0 1 1 1 1 0 0 1 1 0 1 (Produsul final)

Înmulțirea care folosește Carry Propagate Adder are 2 pași importanți și anume: 1) generarea a 16 vectori de câte 32 de biți fiecare obținuți după o anumită regulă. Primul vector se obține concatenând un vector de 16 biți, cu valoarea 0 pe fiecare bit, cu deînmulțitul dacă bitul cel mai puțin semnificativ al înmulțitorului este 1, în caz contrar, primul vector va avea valoarea 0 pe toți biții. Al doilea vector se obține concatenând un vector de 15 biți, cu valoarea 0 pe fiecare bit, cu deînmulțitul dacă al doilea cel mai puțin semnificativ bit al înmulțitorului este 1, în caz contrar, al doilea vector va avea valoarea 0 pe toți biții. Ideea (poate chiar „formula”) de generare al fiecărui vector este: vectorul\_cu\_numărul\_i+1 = concateneaza un vector de 16 – i zerouri cu deînmulțitul dacă bitul de la indexul i al înmulțitorului este 1, în caz contrar vectorul\_cu\_numărul\_i+1 va fi egal cu un vector nul de 32 de biți (i aparține intervalului [0, 15]).

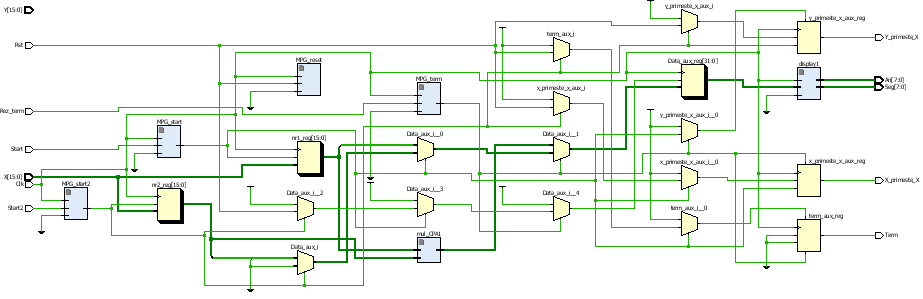
2) adunarea primilor doi vectori generați la pasul 1, apoi adunarea acestei prime sume la cel de-al treilea vector. Vom continua să adunăm suma cu numărul i la vectorul cu numărul i+2 (i aparține intervalului [2, 14]).

**Proiectare și implementare**

Proiectul este scris în limbajul de descriere hardware VHDL, folosindu-se mediul de dezvoltare Vivado 2016.4. Astfel, rezultă faptul că acest proiect este implementat în software, dar el folosește și resurse hardware, deoarece poate funcționa pe o plăcuță FPGA, cum ar fi Nexys4 sau Basys3.

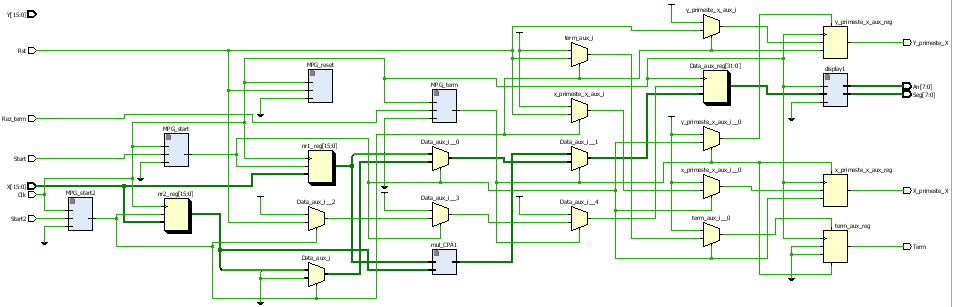
Algoritmii pe care am ales să-i implementez, în acest proiect, sunt Wallace Tree și înmulțitorul care folosește Carry Propagate Adder. Algoritmul de înmulțire al lui Booth și cel de înmulțire matriceală au fost implementați la laborator, din această cauză nu aveam permisiunea să-i implementez la proiect. Așa cum am exemplificat mai sus, mai erau și alți algoritmi de înmulțire pe care aș fi putut să-i implementez, dar am ales să îi implementez pe aceștia doi din următoarele motive: Wallace Tree este un algoritm foarte rapid în ceea ce privește timpul de calculare al unei înmulțiri, mai ales datorită structurii arborescente pe care o are; înmulțitorul care folosește Carry Propagate Adder are un algoritm de înmulțire ușor de înțeles, deoarece este foarte asemănător cu metoda clasică de înmulțire pe foaie.

În continuare se prezintă modulele folosite pentru înmulțitorul care folosește Carry Propagate Adder. Modulele folosite pentru acest circuit de înmulțire sunt în număr de 7. Unul dintre aceste module este full\_adder care are 3 intrări și 2 ieșiri – fiecare dintre acestea fiind pe un bit. Acesta are rolul de a face operațiile specifice unui sumator complet. Al doilea module este carry\_prop\_adder care are 2 intrări și o ieșire – fiecare pe câte 32 de biți. Rolul acestui modul este de a aduna, prin cascadare, doi operanzi de câte 32 de biți fiecare. Un alt modul folosit este multiplier\_CPA\_16 care reprezintă înmulțitorul propriu-zis. Aici se generează 16 vectori de câte 32 de biți înmulțind pe rând fiecare bit al deînmulțitului cu înmulțitorul. După generarea acestor vectori, se adună primii doi vectori folosind modulul precedent. Această primă sumă rezultată se adună la cel de-al treilea vector. Cea de-a doua sumă se adună cu al patrulea vector șamd. Suma cu numărul 15 reprezintă produsul final al celor două numere. Al patrulea modul este cel al bancului de test în care am verificat corectitudinea operațiilor efectuate de către înmulțitor. Al cincilea modul implementat este MPG pe care îl folosim pentru a se efectua anumite operații la apăsarea pe un anumit buton. De exemplu, dacă apăsăm butonul de sus se va încărca primul număr, dacă apăsăm pe butonul din stânga se va încărca cel de-al doilea număr, dacă apăsăm pe butonul din mijloc se va afișa produsul celor două numere introduse. Butonul de jos are ca scop resetarea circuitului. Al șaselea modul implemetat este displ7seg pe care îl folosim la afișarea informațiilor pe cele 8 afișoare ale plăcuței de dezoltare. Al șaptelea modul implementat este Inmultire\_mainn\_CPA\_16 pe care il folosim la conectarea tuturor celorlalte module astfel încât să putem introduce date si să le afișăm rezultatele pe plăcuță.



Schema de mai sus este schema generală (care cuprinde componentele și semnalele necesare pentru a implementa proiectul de o plăcuță de dezvoltare) a circuitului de înmulțire care folosește Carry Propagate Adder.

În continuare se prezintă modulele folosite pentru înmulțitorul Wallace Tree. Modulele folosite pentru acest circuit de înmulțire sunt în număr de 6. Unul dintre aceste module este full\_adder care are 3 intrări și 2 ieșiri – fiecare dintre acestea fiind pe un bit. Acesta are rolul de a face operațiile specifice unui sumator complet. Al doilea modul folosit este wallace16 care reprezintă înmulțitorul propriu-zis. Aici se generează o matrice care va conține produsele parțiale. După generarea acestei matrice, se formează 16 stagii. În primul stagiu plasăm produsele parțiale de la rândurile 0 și 1 în sumatoare complete, iar la intrările de carry in vom pune valoarea 0 la toate cele 16 sumatoare complete. În stagiul 2, sumele rezultate vor avea rol de intrări, alături de produsele parțiale de la linia 2, în sumatoarele complete, iar ieșirile de transport din primul stagiu vor avea rol de intrări ca și carry in în sumatoarele respective. Procesul efectuat la stagiul 2 este valabil și pentru celelate 14 stagii rămase. Al treilea modul este cel al bancului de test în care am verificat corectitudinea operațiilor efectuate de către înmulțitor. Al patrulea modul implementat este MPG pe care îl folosim pentru a se efectua anumite operații la apăsarea pe un anumit buton. De exemplu, dacă apăsăm butonul de sus se va încărca primul număr, dacă apăsăm pe butonul din stânga se va încărca cel de-al doilea număr, dacă apăsăm pe butonul din mijloc se va afișa produsul celor două numere introduse. Butonul de jos are ca scop resetarea circuitului. Al cincilea modul implemetat este displ7seg pe care îl folosim la afișarea informațiilor pe cele 8 afișoare ale plăcuței de dezoltare. Al șaselea modul implementat este Inmultire\_wallace\_16\_mainn pe care il folosim la conectarea tuturor celorlalte module astfel încât să putem introduce date si să le afișăm rezultatele pe plăcuță.

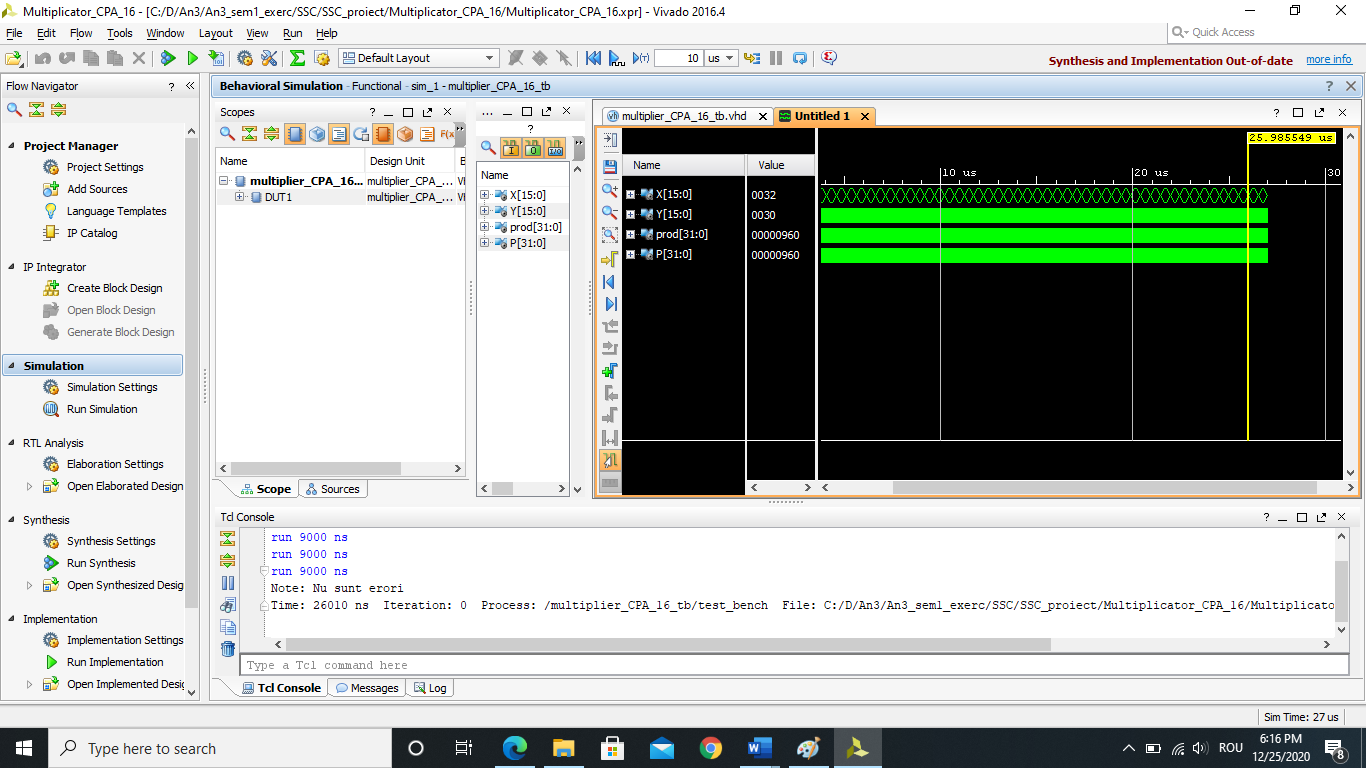


Schema de mai sus este schema generală (care cuprinde componentele și semnalele necesare pentru a implementa proiectul de o plăcuță de dezvoltare) a circuitului de înmulțire care folosește algoritmul Wallace Tree.

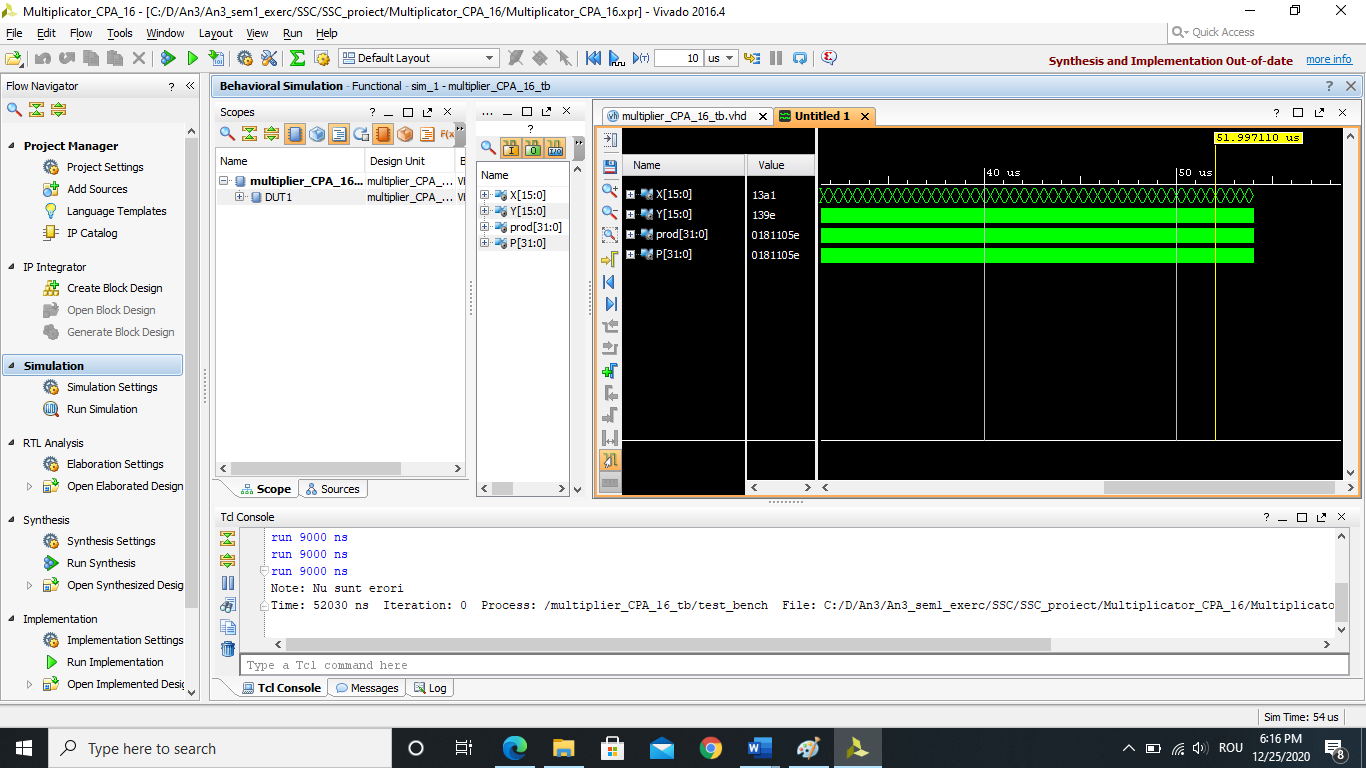
**Rezultate experimentale**

Sistemul de operare folosit pentru realizarea acestui proiect este Windows 10. Mediul software folosit este Vivado 2016.4, platformele hardware pe care proiectul poate să fie testat sunt plăcuțele Basys3, Nexys4, dar și altele. Limbajul în care a fost scris proiectul este VHDL.

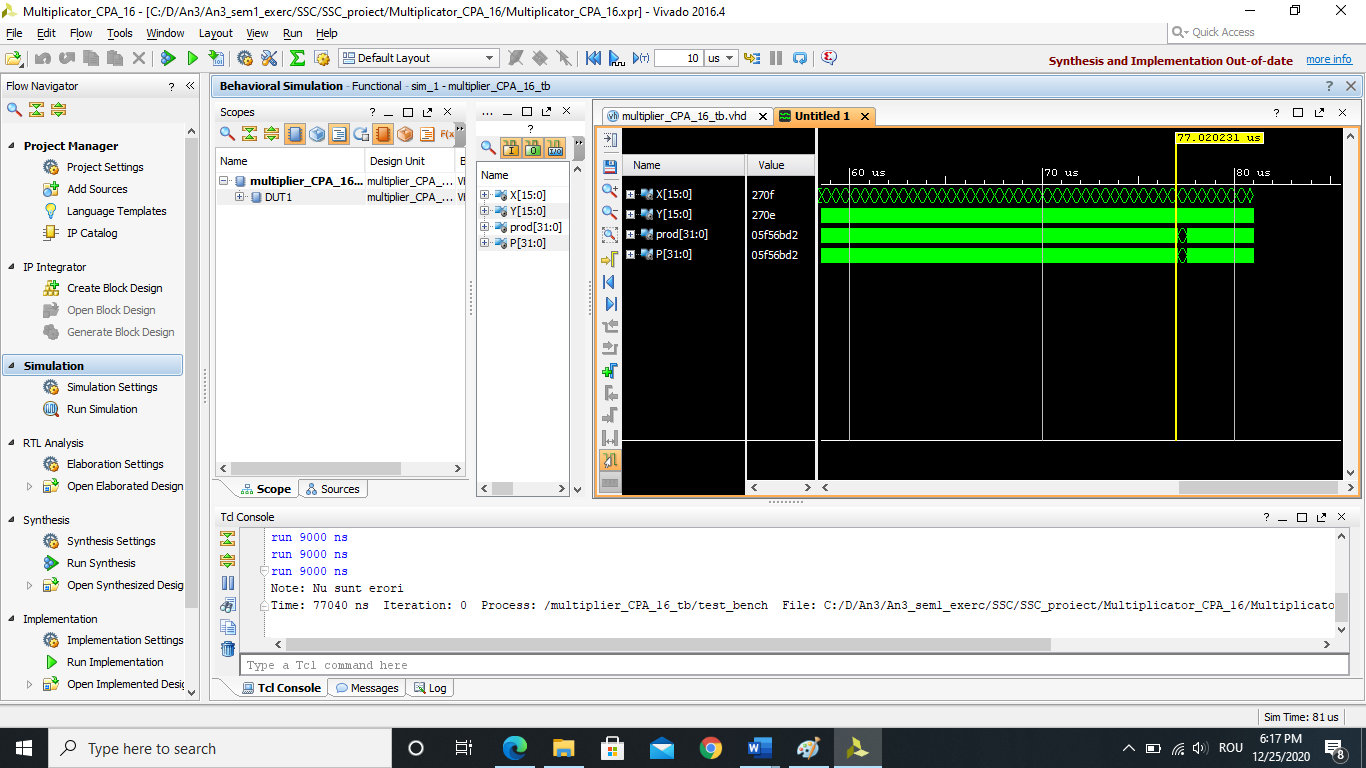
Mai jos sunt rezultatele simulării bancului de test pentru circuitul care este implementat prin înmulțirea care folosește Carry Propagate Adder.



Rezultate banc de test pentru CPA cu valori de la 0 la 50.

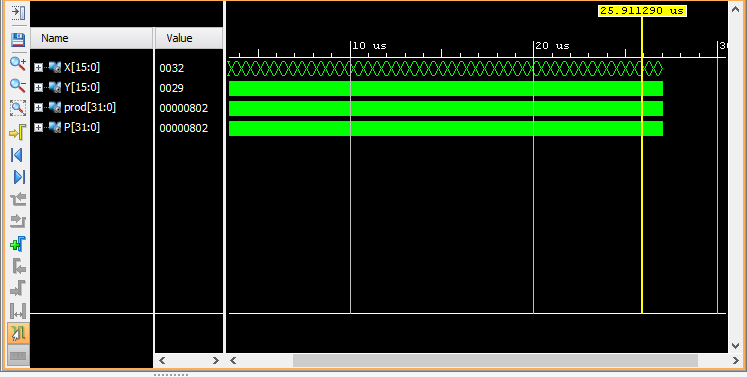


Rezultate banc de test pentru CPA cu valori între 4975 și 5025.

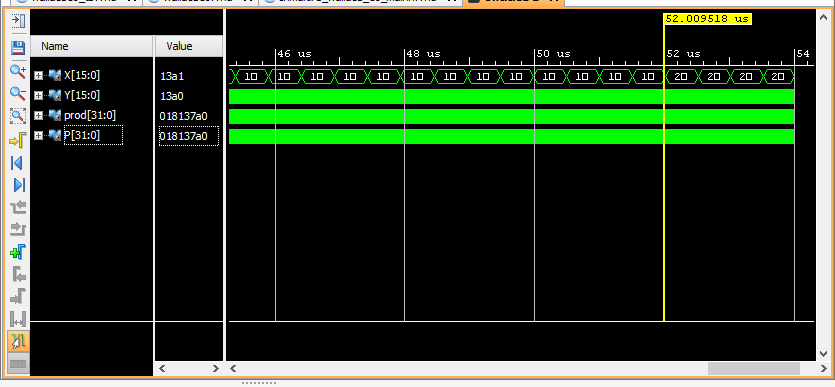


Rezultate banc de test pentru CPA cu valori între 9950 și 9999.

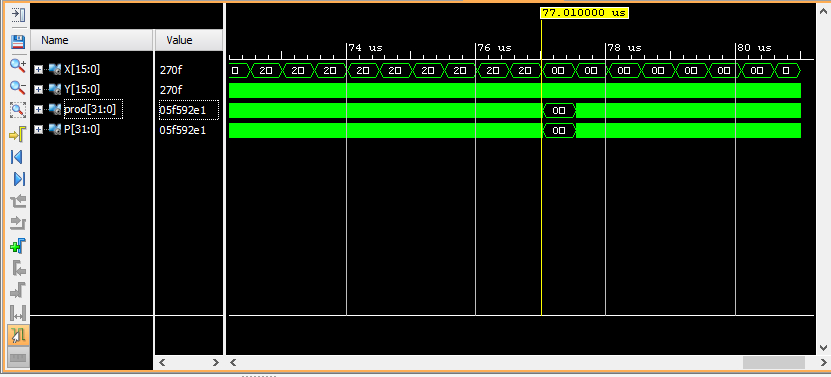
Mai jos sunt rezultatele simulării bancului de test pentru circuitul care este implementat prin înmulțirea care folosește Wallace Tree.



Rezultate banc de test pentru wallace tree cu valori între 0 și 50.



Rezultate banc de test pentru wallace tree cu valori între 4975 și 5025.



Rezultate banc de test pentru wallace tree cu valori între 9950 și 9999.

Mai jos sunt poze ale implementării pe plăcuța de dezvoltare Basys3.

A picture containing text, electronics, circuit

Description automatically generated

În imaginea de mai sus am introdus numărul zecimal 50, apoi am apăsat butonul de sus. Ca dovadă a încărcării numărului se aprinde ledul din dreapta.

A picture containing text, electronics, circuit

Description automatically generated

În imaginea de mai sus am introdus tot numărul zecimal 50, după care am apăsat butonul din stânga. Ca dovadă a încărcării numărului se aprinde al doilea led din dreapta.

A picture containing text, electronics, circuit

Description automatically generated

În imaginea de mai sus este afișat rezultatul înmulțirii lui 50 cu 50. Pentru afișarea rezultatului am apăsat butonul din dreapta. După apăsarea acestui buton, se aprinde al treilea led din dreapta. Rezultatul obținut este corect, deoarece 50 \* 50 = 2500 (în baza 10), iar 32 \* 32 = 9C4 (în baza 16).

A picture containing text, electronics, circuit

Description automatically generated

În imaginea de mai sus am introdus numărul zecimal 5000, apoi am apăsat butonul de sus. Ca dovadă a încărcării numărului se aprinde ledul din dreapta.

A picture containing text, electronics

Description automatically generated

În imaginea de mai sus am introdus tot numărul zecimal 5000, după care am apăsat butonul din stânga. Ca dovadă a încărcării numărului se aprinde al doilea led din dreapta.

A picture containing text, electronics, circuit

Description automatically generated

În imaginea de mai sus este afișat rezultatul înmulțirii lui 5000 cu 5000. Pentru afișarea rezultatului am apăsat butonul din dreapta. După apăsarea acestui buton, se aprinde al treilea led din dreapta. Rezultatul obținut este corect, deoarece 5000 \* 5000 = 25000000 (în baza 10), iar 1388 \* 1388 = 17D7840 (în baza 16).

A picture containing text, electronics

Description automatically generated

În imaginea de mai sus am introdus numărul zecimal 9999, apoi am apăsat butonul de sus. Ca dovadă a încărcării numărului se aprinde ledul din dreapta.

A picture containing text, electronics, circuit

Description automatically generated

În imaginea de mai sus am introdus tot numărul zecimal 9999, după care am apăsat butonul din stânga. Ca dovadă a încărcării numărului se aprinde al doilea led din dreapta.

A picture containing text, electronics, circuit

Description automatically generated

În imaginea de mai sus este afișat rezultatul înmulțirii lui 9999 cu 9999. Pentru afișarea rezultatului am apăsat butonul din dreapta. După apăsarea acestui buton, se aprinde al treilea led din dreapta. Rezultatul obținut este corect, deoarece 9999 \* 9999 = 99980001 (în baza 10), iar 270F \* 270F = 5F592E1 (în baza 16).

Ca o comparație între algoritmul lui Booth, care a fost implementat la laborator și algoritmul Wallace Tree, care a fost implementat în acest proiect, putem remarca faptul că algoritmul lui Booth reduce numărul de produse parțiale. Acest lucru îl face utilizat pe scară largă pentru cazurile în care avem operanzi cu mai mult de 16 biți. Principalul dezavantaj al multiplicatorului Booth este complexitatea circuitului pentru a genera un bit de produs parțial. Algoritmul Wallace Tree este foarte rapid, deoarece numărul nivelurilor logice necesare pentru efectuarea însumării poate fi redus datorită structurii arborescente pe care acest algoritm o are. Dezavantajul acestui algoritm este că acela că este complex din punct de vedere al implementării, deoarece are stagii care nu pot fi scrise respectând o anumită formulă, este, oarecum, iregulat.

**Concluzii**

Problema pe care acest proiect a rezolvat-o este aceea a implementării unor circuite de înmulțire a unor numere zecimale cu maximum 4 cifre. Obiectivele au fost îndeplinite așa cum se poate observa la capitolul “Rezultate experimentale”. Rezultatele obținute sunt corecte.

În urma realizării acestui proiect mi-am îmbunătățit cunoștințele despre limbajul VHDL, dar și modul în care se implementează anumite circuite de înmulțire. De asemenea, am înțeles mai bine cum trebuie să procedez la implementarea proiectului pe o plăcuță de dezvoltare. Menționez că au fost necesare mai multe generări ale bitstream-ului, astfel că a fost necesară analizarea mai aprofundată a codului scris pentru a corecta erorile. Algoritmul Wallace Tree nu a generat rezultate corecte, deși am petrecut mai multe zile pentru găsirea greșelilor. În final, am implementat codul ajutându-mă de o schemă văzută pe internet al unui înmulțitor wallace tree pe 8 biți.

Un avantaj al acestui proiect este că utilizează algoritmi care execută operația de inmulțire foarte repede. Un dezavantaj al acestui proiect este acela că nu efectuează inmulțirea numerelor negative.

Acest proiect poate avea aplicații chiar pentru elevii de la gimnaziu care învață conversia într-o altă bază și pot verifica dacă calculele de pe foaie corespund cu cele de pe plăcută/aplicație software care încorporează acest proiect.

O dezvoltare viitoare a proiectului ar fi aceea de a putea alege, folosind un buton, care algoritm să execute operația de înmulire.

**Rezumat**

Proiectul are ca scop implementarea câtorva circuite de înmulțire zecimală.

Numerele, care sunt înmulțite, au câte 4 cifre fiecare. Pentru înmulțirea acestora, vom folosi diferite metode (circuite) de înmulțire. De asemenea, proiectul necesită implementarea pe o placă de dezvoltare FPGA, iar pentru testarea corectitudinii proiectului am creat un fișier al bancului de test in care am testat corectitudinea inmulțirii numerelor de la 0 la 50; de la 4975 la 5025 și de la 9950 la 9999. Niște exemple de astfel de circuite sunt: algoritmul de înmulțire al lui Booth; Wallace Tree; înmulțirea matriceală; înmulțirea prin deplasare si adunare, înmulțirea într-o bază superioară etc.

Circuitele de inmulțire folosite în proiect sunt: Wallace Tree și înmulțirea folosind Carry Propagate Adder. Funcționalitatea acestui proiect va fi testata pe placa Nexys4DDR. De la switch-uri se vor seta multiplicatorul și multiplicandul, adica valorile care urmeaza a fi inmultite. Proiectul este realizat în VHDL.

Rezultatele obtinute sunt corecte, deoarece am testat corectitudinea atât în fișierul bancului de test cât și pe placa de dezvoltare. Numărul 9999 are nevoie de 14 biți pentru a fi reprezentat ca număr binar. Aceasta înseamnă că rezultatul poate avea maximum 28 de biți. Totuși, cu acordul domnului profesor Alexandru Dan Butiri, am extins proiectul pentru a accepta numere de 16 biți la intrări, putând avea ca rezultat un număr de maximum 32 de biți.

În concluzie, prin intermediul acestui proiect, am invatat cum funcționează 4 algoritmi de înmulțire (doi dintre aceștia au fost făcuți la partea de laborator, iar ceilalți doi au fost implementați de mine). De asemenea, am învățat să lucrez mai bine și cu plăcuța de dezvoltare, datorită faptului că nu s-au afișat corect numerele de la primele încercări.

**Bibliografie**

<https://en.wikipedia.org/wiki/Wallace_tree>

<https://en.wikipedia.org/wiki/Binary_multiplier#:~:text=A%20binary%20multiplier%20is%20an,is%20built%20using%20binary%20adders>

<http://www.csit-sun.pub.ro/courses/Indrumar_CN_New.pdf>

**Anexa A**

Cod Înmulțitor folosind Carry Propagate Adder

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity full\_adder is

port ( a : in std\_logic;

b: in std\_logic ;

cin : in std\_logic ;

sum : out std\_logic ;

cout : out std\_logic

);

end full\_adder;

architecture Behavioral of full\_adder is

begin

sum <= (a xor b xor cin);

cout <= (a and b) or (cin and b) or (a and cin);

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity carry\_prop\_adder is

port ( x: in std\_logic\_vector(31 downto 0);

y :in std\_logic\_vector(31 downto 0);

sum: out std\_logic\_vector(31 downto 0)

);

end carry\_prop\_adder;

architecture Behavioral of carry\_prop\_adder is

component full\_adder is

port ( a : in std\_logic;

b: in std\_logic ;

cin : in std\_logic ;

sum : out std\_logic ;

cout : out std\_logic

);

end component;

signal cout\_cin : std\_logic\_vector (31 downto 1);

begin

sum(0) <= x(0);

FA1 : full\_adder port map( a=>x(1),b=>y(1), cin=>'0', sum=>sum(1),cout=>cout\_cin(1) );

FA2\_7 : for i in 2 to 31 generate

FA : full\_adder port map(a=>x(i),b=>y(i),cin=>cout\_cin(i-1),

sum=>sum(i),cout=>cout\_cin(i));

end generate;

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity multiplier\_CPA\_16 is

port ( a : in std\_logic\_vector(15 downto 0);

b : in std\_logic\_vector(15 downto 0);

prod: out std\_logic\_vector(31 downto 0)

);

end multiplier\_CPA\_16;

architecture Behavioral of multiplier\_CPA\_16 is

component carry\_prop\_adder is

port ( x : in std\_logic\_vector(31 downto 0);

y : in std\_logic\_vector(31 downto 0);

sum: out std\_logic\_vector(31 downto 0)

);

end component;

type sr is array (0 to 15) of std\_logic\_vector(31 downto 0);

signal r : sr;

signal dummy\_prod : sr;

begin

r(0) <= "0000000000000000" & a when b(0) = '1' else "00000000000000000000000000000000";

r(1) <= "000000000000000" & a & "0" when b(1) = '1' else "00000000000000000000000000000000";

r(2) <= "00000000000000" & a & "00" when b(2) = '1' else "00000000000000000000000000000000";

r(3) <= "0000000000000" & a & "000" when b(3) = '1' else "00000000000000000000000000000000";

r(4) <= "000000000000" & a & "0000" when b(4) = '1' else "00000000000000000000000000000000";

r(5) <= "00000000000" & a & "00000" when b(5) = '1' else "00000000000000000000000000000000";

r(6) <= "0000000000" & a & "000000" when b(6) = '1' else "00000000000000000000000000000000";

r(7) <= "000000000" & a & "0000000" when b(7) = '1' else "00000000000000000000000000000000";

r(8) <= "00000000" & a & "00000000" when b(8) = '1' else "00000000000000000000000000000000";

r(9) <= "0000000" & a & "000000000" when b(9) = '1' else "00000000000000000000000000000000";

r(10) <= "000000" & a & "0000000000" when b(10) = '1' else "00000000000000000000000000000000";

r(11) <= "00000" & a & "00000000000" when b(11) = '1' else "00000000000000000000000000000000";

r(12) <= "0000" & a & "000000000000" when b(12) = '1' else "00000000000000000000000000000000";

r(13) <= "000" & a & "0000000000000" when b(13) = '1' else "00000000000000000000000000000000";

r(14) <= "00" & a & "00000000000000" when b(14) = '1' else "00000000000000000000000000000000";

r(15) <= "0" & a & "000000000000000" when b(15) = '1' else "00000000000000000000000000000000";

INITIALIZ : carry\_prop\_adder port map(x=>r(0),y=>r(1),sum=>dummy\_prod(1) );

MULTIPLY : for i in 2 to 15 generate

MTP: carry\_prop\_adder port map(x=>dummy\_prod(i-1), y=>r(i), sum=>dummy\_prod(i) );

end generate;

prod <= dummy\_prod(15);

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity multiplier\_CPA\_16\_tb is

-- Port ( );

end multiplier\_CPA\_16\_tb;

architecture Behavioral of multiplier\_CPA\_16\_tb is

component multiplier\_CPA\_16 is

port ( a : in std\_logic\_vector(15 downto 0);

b : in std\_logic\_vector(15 downto 0);

prod: out std\_logic\_vector(31 downto 0)

);

end component;

signal X : STD\_LOGIC\_VECTOR(15 downto 0) := x"0000";

signal Y : STD\_LOGIC\_VECTOR(15 downto 0):= x"0000";

signal prod : STD\_LOGIC\_VECTOR(31 downto 0) := x"00000000";

signal P : STD\_LOGIC\_VECTOR(31 downto 0) := x"00000000";

begin

DUT1: multiplier\_CPA\_16 port map( A => X, B => Y, prod => P );

test\_bench : process

variable rez\_rau : integer := 0;

begin

for I in 0 to 50 loop

for J in 0 to 50 loop

X <= CONV\_STD\_LOGIC\_VECTOR(I, 16);

Y <= CONV\_STD\_LOGIC\_VECTOR(J, 16);

prod <= CONV\_STD\_LOGIC\_VECTOR(I\*J, 32);

wait for 10 ns;

if (P /= prod) then

rez\_rau := rez\_rau + 1;

report "Rezultat asteptat (" & STD\_LOGIC'image (P(31)) &

STD\_LOGIC'image (P(30)) &

STD\_LOGIC'image (P(29)) & STD\_LOGIC'image (P(28)) &

STD\_LOGIC'image (P(27)) & STD\_LOGIC'image (P(26)) &

STD\_LOGIC'image (P(25)) & STD\_LOGIC'image (P(24)) &

STD\_LOGIC'image (P(23)) & STD\_LOGIC'image (P(22)) &

STD\_LOGIC'image (P(21)) & STD\_LOGIC'image (P(20)) &

STD\_LOGIC'image (P(19)) & STD\_LOGIC'image (P(18)) &

STD\_LOGIC'image (P(17)) & STD\_LOGIC'image (P(16)) &

STD\_LOGIC'image (P(15)) & STD\_LOGIC'image (P(14)) &

STD\_LOGIC'image (P(13)) & STD\_LOGIC'image (P(12)) &

STD\_LOGIC'image (P(11)) & STD\_LOGIC'image (P(10)) &

STD\_LOGIC'image (P(9)) & STD\_LOGIC'image (P(8)) &

STD\_LOGIC'image (P(7)) & STD\_LOGIC'image (P(6)) &

STD\_LOGIC'image (P(5)) & STD\_LOGIC'image (P(4)) &

STD\_LOGIC'image (P(3)) & STD\_LOGIC'image (P(2)) &

STD\_LOGIC'image (P(1)) & STD\_LOGIC'image (P(0))

& ") /= Valoare obtinuta (" & STD\_LOGIC'image (prod(31)) &

STD\_LOGIC'image (prod(30)) &

STD\_LOGIC'image (prod(29)) & STD\_LOGIC'image (prod(28)) &

STD\_LOGIC'image (prod(27)) & STD\_LOGIC'image (prod(26)) &

STD\_LOGIC'image (prod(25)) & STD\_LOGIC'image (prod(24)) &

STD\_LOGIC'image (prod(23)) & STD\_LOGIC'image (prod(22)) &

STD\_LOGIC'image (prod(21)) & STD\_LOGIC'image (prod(20)) &

STD\_LOGIC'image (prod(19)) & STD\_LOGIC'image (prod(18)) &

STD\_LOGIC'image (prod(17)) & STD\_LOGIC'image (prod(16)) &

STD\_LOGIC'image (prod(15)) & STD\_LOGIC'image (prod(14)) &

STD\_LOGIC'image (prod(13)) & STD\_LOGIC'image (prod(12)) &

STD\_LOGIC'image (prod(11)) & STD\_LOGIC'image (prod(10)) &

STD\_LOGIC'image (prod(9)) & STD\_LOGIC'image (prod(8)) &

STD\_LOGIC'image (prod(7)) & STD\_LOGIC'image (prod(6)) &

STD\_LOGIC'image (prod(5)) & STD\_LOGIC'image (prod(4)) &

STD\_LOGIC'image (prod(3)) & STD\_LOGIC'image (prod(2)) &

STD\_LOGIC'image (prod(1)) & STD\_LOGIC'image (prod(0)) &

") la t = " & TIME'image (now)

severity ERROR;

end if;

end loop;

end loop;

if rez\_rau > 0 then

report "Numar erori = " & INTEGER'image(rez\_rau);

else

report "Nu sunt erori";

end if;

wait for 10 ns;

for I in 4975 to 5025 loop

for J in 4975 to 5025 loop

X <= CONV\_STD\_LOGIC\_VECTOR(I, 16);

Y <= CONV\_STD\_LOGIC\_VECTOR(J, 16);

prod <= CONV\_STD\_LOGIC\_VECTOR(I\*J, 32);

wait for 10 ns;

if (P /= prod) then

rez\_rau := rez\_rau + 1;

report "Rezultat asteptat (" & STD\_LOGIC'image (P(31)) &

STD\_LOGIC'image (P(30)) &

STD\_LOGIC'image (P(29)) & STD\_LOGIC'image (P(28)) &

STD\_LOGIC'image (P(27)) & STD\_LOGIC'image (P(26)) &

STD\_LOGIC'image (P(25)) & STD\_LOGIC'image (P(24)) &

STD\_LOGIC'image (P(23)) & STD\_LOGIC'image (P(22)) &

STD\_LOGIC'image (P(21)) & STD\_LOGIC'image (P(20)) &

STD\_LOGIC'image (P(19)) & STD\_LOGIC'image (P(18)) &

STD\_LOGIC'image (P(17)) & STD\_LOGIC'image (P(16)) &

STD\_LOGIC'image (P(15)) & STD\_LOGIC'image (P(14)) &

STD\_LOGIC'image (P(13)) & STD\_LOGIC'image (P(12)) &

STD\_LOGIC'image (P(11)) & STD\_LOGIC'image (P(10)) &

STD\_LOGIC'image (P(9)) & STD\_LOGIC'image (P(8)) &

STD\_LOGIC'image (P(7)) & STD\_LOGIC'image (P(6)) &

STD\_LOGIC'image (P(5)) & STD\_LOGIC'image (P(4)) &

STD\_LOGIC'image (P(3)) & STD\_LOGIC'image (P(2)) &

STD\_LOGIC'image (P(1)) & STD\_LOGIC'image (P(0))

& ") /= Valoare obtinuta (" & STD\_LOGIC'image (prod(31)) &

STD\_LOGIC'image (prod(30)) &

STD\_LOGIC'image (prod(29)) & STD\_LOGIC'image (prod(28)) &

STD\_LOGIC'image (prod(27)) & STD\_LOGIC'image (prod(26)) &

STD\_LOGIC'image (prod(25)) & STD\_LOGIC'image (prod(24)) &

STD\_LOGIC'image (prod(23)) & STD\_LOGIC'image (prod(22)) &

STD\_LOGIC'image (prod(21)) & STD\_LOGIC'image (prod(20)) &

STD\_LOGIC'image (prod(19)) & STD\_LOGIC'image (prod(18)) &

STD\_LOGIC'image (prod(17)) & STD\_LOGIC'image (prod(16)) &

STD\_LOGIC'image (prod(15)) & STD\_LOGIC'image (prod(14)) &

STD\_LOGIC'image (prod(13)) & STD\_LOGIC'image (prod(12)) &

STD\_LOGIC'image (prod(11)) & STD\_LOGIC'image (prod(10)) &

STD\_LOGIC'image (prod(9)) & STD\_LOGIC'image (prod(8)) &

STD\_LOGIC'image (prod(7)) & STD\_LOGIC'image (prod(6)) &

STD\_LOGIC'image (prod(5)) & STD\_LOGIC'image (prod(4)) &

STD\_LOGIC'image (prod(3)) & STD\_LOGIC'image (prod(2)) &

STD\_LOGIC'image (prod(1)) & STD\_LOGIC'image (prod(0)) &

") la t = " & TIME'image (now)

severity ERROR;

end if;

end loop;

end loop;

if rez\_rau > 0 then

report "Numar erori = " & INTEGER'image(rez\_rau);

else

report "Nu sunt erori";

end if;

wait for 10 ns;

for I in 9950 to 9999 loop

for J in 9950 to 9999 loop

X <= CONV\_STD\_LOGIC\_VECTOR(I, 16);

Y <= CONV\_STD\_LOGIC\_VECTOR(J, 16);

prod <= CONV\_STD\_LOGIC\_VECTOR(I\*J, 32);

wait for 10 ns;

if (P /= prod) then

rez\_rau := rez\_rau + 1;

report "Rezultat asteptat (" & STD\_LOGIC'image (P(31)) &

STD\_LOGIC'image (P(30)) &

STD\_LOGIC'image (P(29)) & STD\_LOGIC'image (P(28)) &

STD\_LOGIC'image (P(27)) & STD\_LOGIC'image (P(26)) &

STD\_LOGIC'image (P(25)) & STD\_LOGIC'image (P(24)) &

STD\_LOGIC'image (P(23)) & STD\_LOGIC'image (P(22)) &

STD\_LOGIC'image (P(21)) & STD\_LOGIC'image (P(20)) &

STD\_LOGIC'image (P(19)) & STD\_LOGIC'image (P(18)) &

STD\_LOGIC'image (P(17)) & STD\_LOGIC'image (P(16)) &

STD\_LOGIC'image (P(15)) & STD\_LOGIC'image (P(14)) &

STD\_LOGIC'image (P(13)) & STD\_LOGIC'image (P(12)) &

STD\_LOGIC'image (P(11)) & STD\_LOGIC'image (P(10)) &

STD\_LOGIC'image (P(9)) & STD\_LOGIC'image (P(8)) &

STD\_LOGIC'image (P(7)) & STD\_LOGIC'image (P(6)) &

STD\_LOGIC'image (P(5)) & STD\_LOGIC'image (P(4)) &

STD\_LOGIC'image (P(3)) & STD\_LOGIC'image (P(2)) &

STD\_LOGIC'image (P(1)) & STD\_LOGIC'image (P(0))

& ") /= Valoare obtinuta (" & STD\_LOGIC'image (prod(31)) &

STD\_LOGIC'image (prod(30)) &

STD\_LOGIC'image (prod(29)) & STD\_LOGIC'image (prod(28)) &

STD\_LOGIC'image (prod(27)) & STD\_LOGIC'image (prod(26)) &

STD\_LOGIC'image (prod(25)) & STD\_LOGIC'image (prod(24)) &

STD\_LOGIC'image (prod(23)) & STD\_LOGIC'image (prod(22)) &

STD\_LOGIC'image (prod(21)) & STD\_LOGIC'image (prod(20)) &

STD\_LOGIC'image (prod(19)) & STD\_LOGIC'image (prod(18)) &

STD\_LOGIC'image (prod(17)) & STD\_LOGIC'image (prod(16)) &

STD\_LOGIC'image (prod(15)) & STD\_LOGIC'image (prod(14)) &

STD\_LOGIC'image (prod(13)) & STD\_LOGIC'image (prod(12)) &

STD\_LOGIC'image (prod(11)) & STD\_LOGIC'image (prod(10)) &

STD\_LOGIC'image (prod(9)) & STD\_LOGIC'image (prod(8)) &

STD\_LOGIC'image (prod(7)) & STD\_LOGIC'image (prod(6)) &

STD\_LOGIC'image (prod(5)) & STD\_LOGIC'image (prod(4)) &

STD\_LOGIC'image (prod(3)) & STD\_LOGIC'image (prod(2)) &

STD\_LOGIC'image (prod(1)) & STD\_LOGIC'image (prod(0)) &

") la t = " & TIME'image (now)

severity ERROR;

end if;

end loop;

end loop;

if rez\_rau > 0 then

report "Numar erori = " & INTEGER'image(rez\_rau);

else

report "Nu sunt erori";

end if;

wait for 10 ns;

end process test\_bench;

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

entity MPG is

Port ( clk : in std\_logic;

rst,input:in std\_logic;

q\_out:out std\_logic

);

end MPG;

architecture Behavioral of MPG is

signal Q1, Q2, Q3 : std\_logic;

begin

process(clk)

begin

if (clk'event and clk = '1') then

if (rst = '1') then

Q1 <= '0';

Q2 <= '0';

Q3 <= '0';

else

Q1 <= input;

Q2 <= Q1;

Q3 <= Q2;

end if;

end if;

end process;

Q\_OUT <= Q1 and Q2 and (not Q3);

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

use IEEE.STD\_LOGIC\_ARITH.all;

entity displ7seg is

Port ( Clk : in STD\_LOGIC;

Rst : in STD\_LOGIC;

Data : in STD\_LOGIC\_VECTOR (31 downto 0); -- datele pentru 8 cifre (cifra 1 din stanga: biti 31..28)

An : out STD\_LOGIC\_VECTOR (7 downto 0); -- selectia anodului activ

Seg : out STD\_LOGIC\_VECTOR (7 downto 0)); -- selectia catozilor (segmentelor) cifrei active

end displ7seg;

architecture Behavioral of displ7seg is

constant CNT\_100HZ : integer := 2\*\*20; -- divizor pentru rata de reimprospatare de ~100 Hz (cu un ceas de 100 MHz)

signal Num : integer range 0 to CNT\_100HZ - 1 := 0;

signal NumV : STD\_LOGIC\_VECTOR (19 downto 0) := (others => '0');

signal LedSel : STD\_LOGIC\_VECTOR (2 downto 0) := (others => '0');

signal Hex : STD\_LOGIC\_VECTOR (3 downto 0) := (others => '0');

begin

-- Proces pentru divizarea ceasului

divclk: process (Clk)

begin

if (Clk'event and Clk = '1') then

if (Rst = '1') then

Num <= 0;

elsif (Num = CNT\_100HZ - 1) then

Num <= 0;

else

Num <= Num + 1;

end if;

end if;

end process;

NumV <= CONV\_STD\_LOGIC\_VECTOR (Num, 20);

LedSel <= NumV (19 downto 17);

-- Selectia anodului activ

An <= "11111110" when LedSel = "000" else

"11111101" when LedSel = "001" else

"11111011" when LedSel = "010" else

"11110111" when LedSel = "011" else

"11101111" when LedSel = "100" else

"11011111" when LedSel = "101" else

"10111111" when LedSel = "110" else

"01111111" when LedSel = "111" else

"11111111";

-- Selectia cifrei active

Hex <= Data (3 downto 0) when LedSel = "000" else

Data (7 downto 4) when LedSel = "001" else

Data (11 downto 8) when LedSel = "010" else

Data (15 downto 12) when LedSel = "011" else

Data (19 downto 16) when LedSel = "100" else

Data (23 downto 20) when LedSel = "101" else

Data (27 downto 24) when LedSel = "110" else

Data (31 downto 28) when LedSel = "111" else

X"0";

-- Activarea/dezactivarea segmentelor cifrei active

Seg <= "11111001" when Hex = "0001" else -- 1

"10100100" when Hex = "0010" else -- 2

"10110000" when Hex = "0011" else -- 3

"10011001" when Hex = "0100" else -- 4

"10010010" when Hex = "0101" else -- 5

"10000010" when Hex = "0110" else -- 6

"11111000" when Hex = "0111" else -- 7

"10000000" when Hex = "1000" else -- 8

"10010000" when Hex = "1001" else -- 9

"10001000" when Hex = "1010" else -- A

"10000011" when Hex = "1011" else -- b

"11000110" when Hex = "1100" else -- C

"10100001" when Hex = "1101" else -- d

"10000110" when Hex = "1110" else -- E

"10001110" when Hex = "1111" else -- F

"11000000"; -- 0

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity Inmultire\_mainn\_CPA\_16 is

Port ( X : in STD\_LOGIC\_VECTOR(15 downto 0);

Y : in STD\_LOGIC\_VECTOR(15 downto 0);

Rst : in STD\_LOGIC;

Start : in STD\_LOGIC;

Start2 : in STD\_LOGIC;

Rez\_term : in std\_logic;

Clk : in STD\_LOGIC;

X\_primeste\_X : out STD\_LOGIC;

Y\_primeste\_X : out STD\_LOGIC;

An: out STD\_LOGIC\_VECTOR(7 downto 0);

Seg: out STD\_LOGIC\_VECTOR(7 downto 0);

Term : out STD\_LOGIC

);

end Inmultire\_mainn\_CPA\_16;

architecture Behavioral of Inmultire\_mainn\_CPA\_16 is

component multiplier\_CPA\_16 is

port ( a : in std\_logic\_vector(15 downto 0);

b : in std\_logic\_vector(15 downto 0);

prod: out std\_logic\_vector(31 downto 0)

);

end component;

component displ7seg is

Port ( Clk : in STD\_LOGIC;

Rst : in STD\_LOGIC;

Data : in STD\_LOGIC\_VECTOR (31 downto 0); -- datele pentru 8 cifre (cifra 1 din stanga: biti 31..28)

An : out STD\_LOGIC\_VECTOR (7 downto 0); -- selectia anodului activ

Seg : out STD\_LOGIC\_VECTOR (7 downto 0)); -- selectia catozilor (segmentelor) cifrei active

end component;

component MPG is

Port ( clk : in std\_logic;

rst,input:in std\_logic;

q\_out:out std\_logic

);

end component;

signal nr1, nr2 : STD\_LOGIC\_VECTOR(15 downto 0) := x"0000";

signal Data\_aux : STD\_LOGIC\_VECTOR(31 downto 0);

signal P : STD\_LOGIC\_VECTOR(31 downto 0);

signal en\_start, en\_start2, en\_rez : std\_logic := '0';

signal x\_primeste\_x\_aux, y\_primeste\_x\_aux, term\_aux : std\_logic := '0';

signal rst\_aux : std\_logic := '0';

begin

mul\_CPA1 : multiplier\_CPA\_16 port map(A => nr1 , B => nr2, prod => P);

MPG\_start : MPG port map(clk=>Clk ,rst=>'0',input=>Start, q\_out=> en\_start);

MPG\_start2 : MPG port map(clk=>Clk ,rst=>'0',input=>Start2, q\_out=> en\_start2);

MPG\_term : MPG port map(clk=>Clk ,rst=>'0',input=>Rez\_term, q\_out=> en\_rez);

MPG\_reset : MPG port map(clk=>Clk ,rst=>'0',input=>Rst, q\_out=> rst\_aux);

p1 : process (clk, en\_start, en\_start2, en\_rez)

begin

if rising\_edge(clk) then

if Rst = '1' then

Y\_primeste\_X\_aux <= '0';

X\_primeste\_X\_aux <= '0';

term\_aux <= '0';

Data\_aux <= x"00000000";

end if;

if en\_start2 = '1' then

nr2 <= X;

Y\_primeste\_X\_aux <= '1';

X\_primeste\_X\_aux <= '0';

term\_aux <= '0';

Data\_aux <= x"0000" & nr2;

end if;

if en\_start = '1' then

nr1 <= X;

X\_primeste\_X\_aux <= '1';

Y\_primeste\_X\_aux <= '0';

term\_aux <= '0';

Data\_aux <= x"0000" & nr1;

end if;

if en\_rez = '1' then

term\_aux <= '1';

X\_primeste\_X\_aux <= '0';

Y\_primeste\_X\_aux <= '0';

Data\_aux <= P;

end if;

end if;

Y\_primeste\_X <= Y\_primeste\_X\_aux;

X\_primeste\_X <= X\_primeste\_X\_aux;

term <= term\_aux;

end process;

display1 : displ7seg port map(Clk => Clk, Rst => '0', Data => Data\_aux,

An => An, Seg => Seg);

end Behavioral;

**Anexa B**

Algoritmul de înmulțire Wallace Tree

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity full\_adder is

port ( a : in std\_logic;

b: in std\_logic ;

c : in std\_logic ;

sum : out std\_logic ;

cout : out std\_logic

);

end full\_adder;

architecture Behavioral of full\_adder is

begin

sum <= (a xor b xor c);

cout <= (a and b) or (c and b) or (a and c);

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity wallace16 is

Port ( A : in STD\_LOGIC\_VECTOR (15 downto 0);

B : in STD\_LOGIC\_VECTOR (15 downto 0);

prod : out STD\_LOGIC\_VECTOR (31 downto 0)

);

end wallace16;

architecture Behavioral of wallace16 is

component full\_adder is

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

sum : out STD\_LOGIC;

cout : out STD\_LOGIC);

end component;

type tip\_pp is array (0 to 15) of std\_logic\_vector (15 downto 0);

signal pp : tip\_pp;

-------- st 1

signal s\_101, s\_102, s\_103, s\_104, s\_105, s\_106, s\_107, s\_108 : STD\_LOGIC := '0';

signal s\_109, s\_110, s\_111, s\_112, s\_113, s\_114, s\_115 : STD\_LOGIC := '0';

signal cout\_101, cout\_102, cout\_103, cout\_104, cout\_105, cout\_106, cout\_107, cout\_108 : STD\_LOGIC := '0';

signal cout\_109, cout\_110, cout\_111, cout\_112, cout\_113, cout\_114, cout\_115 : STD\_LOGIC := '0';

-------- st 2

signal s\_201, s\_202, s\_203, s\_204, s\_205, s\_206, s\_207, s\_208 : STD\_LOGIC := '0';

signal s\_209, s\_210, s\_211, s\_212, s\_213, s\_214, s\_215 : STD\_LOGIC := '0';

signal cout\_201, cout\_202, cout\_203, cout\_204, cout\_205, cout\_206, cout\_207, cout\_208 : STD\_LOGIC := '0';

signal cout\_209, cout\_210, cout\_211, cout\_212, cout\_213, cout\_214, cout\_215 : STD\_LOGIC := '0';

-------- st 3

signal s\_301, s\_302, s\_303, s\_304, s\_305, s\_306, s\_307, s\_308 : STD\_LOGIC := '0';

signal s\_309, s\_310, s\_311, s\_312, s\_313, s\_314, s\_315 : STD\_LOGIC := '0';

signal cout\_301, cout\_302, cout\_303, cout\_304, cout\_305, cout\_306, cout\_307, cout\_308 : STD\_LOGIC := '0';

signal cout\_309, cout\_310, cout\_311, cout\_312, cout\_313, cout\_314, cout\_315 : STD\_LOGIC := '0';

-------- st 4

signal s\_401, s\_402, s\_403, s\_404, s\_405, s\_406, s\_407, s\_408 : STD\_LOGIC := '0';

signal s\_409, s\_410, s\_411, s\_412, s\_413, s\_414, s\_415 : STD\_LOGIC := '0';

signal cout\_401, cout\_402, cout\_403, cout\_404, cout\_405, cout\_406, cout\_407, cout\_408 : STD\_LOGIC := '0';

signal cout\_409, cout\_410, cout\_411, cout\_412, cout\_413, cout\_414, cout\_415 : STD\_LOGIC := '0';

-------- st 5

signal s\_501, s\_502, s\_503, s\_504, s\_505, s\_506, s\_507, s\_508 : STD\_LOGIC := '0';

signal s\_509, s\_510, s\_511, s\_512, s\_513, s\_514, s\_515 : STD\_LOGIC := '0';

signal cout\_501, cout\_502, cout\_503, cout\_504, cout\_505, cout\_506, cout\_507, cout\_508 : STD\_LOGIC := '0';

signal cout\_509, cout\_510, cout\_511, cout\_512, cout\_513, cout\_514, cout\_515 : STD\_LOGIC := '0';

-------- st 6

signal s\_601, s\_602, s\_603, s\_604, s\_605, s\_606, s\_607, s\_608 : STD\_LOGIC := '0';

signal s\_609, s\_610, s\_611, s\_612, s\_613, s\_614, s\_615 : STD\_LOGIC := '0';

signal cout\_601, cout\_602, cout\_603, cout\_604, cout\_605, cout\_606, cout\_607, cout\_608 : STD\_LOGIC := '0';

signal cout\_609, cout\_610, cout\_611, cout\_612, cout\_613, cout\_614, cout\_615 : STD\_LOGIC := '0';

-------- st 7

signal s\_701, s\_702, s\_703, s\_704, s\_705, s\_706, s\_707, s\_708 : STD\_LOGIC := '0';

signal s\_709, s\_710, s\_711, s\_712, s\_713, s\_714, s\_715 : STD\_LOGIC := '0';

signal cout\_701, cout\_702, cout\_703, cout\_704, cout\_705, cout\_706, cout\_707, cout\_708 : STD\_LOGIC := '0';

signal cout\_709, cout\_710, cout\_711, cout\_712, cout\_713, cout\_714, cout\_715 : STD\_LOGIC := '0';

-------- st 8

signal s\_801, s\_802, s\_803, s\_804, s\_805, s\_806, s\_807, s\_808 : STD\_LOGIC := '0';

signal s\_809, s\_810, s\_811, s\_812, s\_813, s\_814, s\_815 : STD\_LOGIC := '0';

signal cout\_801, cout\_802, cout\_803, cout\_804, cout\_805, cout\_806, cout\_807, cout\_808 : STD\_LOGIC := '0';

signal cout\_809, cout\_810, cout\_811, cout\_812, cout\_813, cout\_814, cout\_815 : STD\_LOGIC := '0';

-------- st 9

signal s\_901, s\_902, s\_903, s\_904, s\_905, s\_906, s\_907, s\_908 : STD\_LOGIC := '0';

signal s\_909, s\_910, s\_911, s\_912, s\_913, s\_914, s\_915 : STD\_LOGIC := '0';

signal cout\_901, cout\_902, cout\_903, cout\_904, cout\_905, cout\_906, cout\_907, cout\_908 : STD\_LOGIC := '0';

signal cout\_909, cout\_910, cout\_911, cout\_912, cout\_913, cout\_914, cout\_915 : STD\_LOGIC := '0';

-------- st 10

signal s\_1001, s\_1002, s\_1003, s\_1004, s\_1005, s\_1006, s\_1007, s\_1008 : STD\_LOGIC := '0';

signal s\_1009, s\_1010, s\_1011, s\_1012, s\_1013, s\_1014, s\_1015 : STD\_LOGIC := '0';

signal cout\_1001, cout\_1002, cout\_1003, cout\_1004, cout\_1005, cout\_1006, cout\_1007, cout\_1008 : STD\_LOGIC := '0';

signal cout\_1009, cout\_1010, cout\_1011, cout\_1012, cout\_1013, cout\_1014, cout\_1015 : STD\_LOGIC := '0';

-------- st 11

signal s\_1101, s\_1102, s\_1103, s\_1104, s\_1105, s\_1106, s\_1107, s\_1108 : STD\_LOGIC := '0';

signal s\_1109, s\_1110, s\_1111, s\_1112, s\_1113, s\_1114, s\_1115 : STD\_LOGIC := '0';

signal cout\_1101, cout\_1102, cout\_1103, cout\_1104, cout\_1105, cout\_1106, cout\_1107, cout\_1108 : STD\_LOGIC := '0';

signal cout\_1109, cout\_1110, cout\_1111, cout\_1112, cout\_1113, cout\_1114, cout\_1115 : STD\_LOGIC := '0';

-------- st 12

signal s\_1201, s\_1202, s\_1203, s\_1204, s\_1205, s\_1206, s\_1207, s\_1208 : STD\_LOGIC := '0';

signal s\_1209, s\_1210, s\_1211, s\_1212, s\_1213, s\_1214, s\_1215 : STD\_LOGIC := '0';

signal cout\_1201, cout\_1202, cout\_1203, cout\_1204, cout\_1205, cout\_1206, cout\_1207, cout\_1208 : STD\_LOGIC := '0';

signal cout\_1209, cout\_1210, cout\_1211, cout\_1212, cout\_1213, cout\_1214, cout\_1215 : STD\_LOGIC := '0';

-------- st 13

signal s\_1301, s\_1302, s\_1303, s\_1304, s\_1305, s\_1306, s\_1307, s\_1308 : STD\_LOGIC := '0';

signal s\_1309, s\_1310, s\_1311, s\_1312, s\_1313, s\_1314, s\_1315 : STD\_LOGIC := '0';

signal cout\_1301, cout\_1302, cout\_1303, cout\_1304, cout\_1305, cout\_1306, cout\_1307, cout\_1308 : STD\_LOGIC := '0';

signal cout\_1309, cout\_1310, cout\_1311, cout\_1312, cout\_1313, cout\_1314, cout\_1315 : STD\_LOGIC := '0';

-------- st 14

signal s\_1401, s\_1402, s\_1403, s\_1404, s\_1405, s\_1406, s\_1407, s\_1408 : STD\_LOGIC := '0';

signal s\_1409, s\_1410, s\_1411, s\_1412, s\_1413, s\_1414, s\_1415 : STD\_LOGIC := '0';

signal cout\_1401, cout\_1402, cout\_1403, cout\_1404, cout\_1405, cout\_1406, cout\_1407, cout\_1408 : STD\_LOGIC := '0';

signal cout\_1409, cout\_1410, cout\_1411, cout\_1412, cout\_1413, cout\_1414, cout\_1415 : STD\_LOGIC := '0';

-------- st 15

signal s\_1501, s\_1502, s\_1503, s\_1504, s\_1505, s\_1506, s\_1507, s\_1508 : STD\_LOGIC := '0';

signal s\_1509, s\_1510, s\_1511, s\_1512, s\_1513, s\_1514, s\_1515 : STD\_LOGIC := '0';

signal cout\_1501, cout\_1502, cout\_1503, cout\_1504, cout\_1505, cout\_1506, cout\_1507, cout\_1508 : STD\_LOGIC := '0';

signal cout\_1509, cout\_1510, cout\_1511, cout\_1512, cout\_1513, cout\_1514, cout\_1515 : STD\_LOGIC := '0';

-------- st 16

signal cout\_1601, cout\_1602, cout\_1603, cout\_1604, cout\_1605, cout\_1606, cout\_1607, cout\_1608 : STD\_LOGIC := '0';

signal cout\_1609, cout\_1610, cout\_1611, cout\_1612, cout\_1613, cout\_1614 : STD\_LOGIC := '0';

signal p0, p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14, p15 : STD\_LOGIC := '0';

signal p16, p17, p18, p19, p20, p21, p22, p23, p24, p25, p26, p27, p28, p29, p30, p31, p32 : STD\_LOGIC := '0';

begin

gen\_pp1: for i in 0 to 15 generate

gen\_pp2: for j in 0 to 15 generate

pp(i)(j) <= A(j) and B(i);

end generate;

end generate;

--------------- stage 1

F1\_01: full\_adder port map (a=>pp(1)(0), b=>pp(0)(1), c=>'0', sum=>s\_101, cout=>cout\_101 );

F1\_02: full\_adder port map (a=>pp(1)(1), b=>pp(0)(2), c=>'0', sum=>s\_102, cout=>cout\_102 );

F1\_03: full\_adder port map (a=>pp(1)(2), b=>pp(0)(3), c=>'0', sum=>s\_103, cout=>cout\_103 );

F1\_04: full\_adder port map (a=>pp(1)(3), b=>pp(0)(4), c=>'0', sum=>s\_104, cout=>cout\_104 );

F1\_05: full\_adder port map (a=>pp(1)(4), b=>pp(0)(5), c=>'0', sum=>s\_105, cout=>cout\_105 );

F1\_06: full\_adder port map (a=>pp(1)(5), b=>pp(0)(6), c=>'0', sum=>s\_106, cout=>cout\_106 );

F1\_07: full\_adder port map (a=>pp(1)(6), b=>pp(0)(7), c=>'0', sum=>s\_107, cout=>cout\_107 );

F1\_08: full\_adder port map (a=>pp(1)(7), b=>pp(0)(8), c=>'0', sum=>s\_108, cout=>cout\_108 );

F1\_09: full\_adder port map (a=>pp(1)(8), b=>pp(0)(9), c=>'0', sum=>s\_109, cout=>cout\_109 );

F1\_10: full\_adder port map (a=>pp(1)(9), b=>pp(0)(10), c=>'0', sum=>s\_110, cout=>cout\_110 );

F1\_11: full\_adder port map (a=>pp(1)(10), b=>pp(0)(11), c=>'0', sum=>s\_111, cout=>cout\_111 );

F1\_12: full\_adder port map (a=>pp(1)(11), b=>pp(0)(12), c=>'0', sum=>s\_112, cout=>cout\_112 );

F1\_13: full\_adder port map (a=>pp(1)(12), b=>pp(0)(13), c=>'0', sum=>s\_113, cout=>cout\_113 );

F1\_14: full\_adder port map (a=>pp(1)(13), b=>pp(0)(14), c=>'0', sum=>s\_114, cout=>cout\_114 );

F1\_15: full\_adder port map (a=>pp(1)(14), b=>pp(0)(15), c=>'0', sum=>s\_115, cout=>cout\_115 );

-----------------------------------------------------------------

------------------------ stage 2

F2\_01: full\_adder port map (a=>pp(2)(0), b=>s\_102, c=>cout\_101, sum=>s\_201, cout=>cout\_201 );

F2\_02: full\_adder port map (a=>pp(2)(1), b=>s\_103, c=>cout\_102, sum=>s\_202, cout=>cout\_202 );

F2\_03: full\_adder port map (a=>pp(2)(2), b=>s\_104, c=>cout\_103, sum=>s\_203, cout=>cout\_203 );

F2\_04: full\_adder port map (a=>pp(2)(3), b=>s\_105, c=>cout\_104, sum=>s\_204, cout=>cout\_204 );

F2\_05: full\_adder port map (a=>pp(2)(4), b=>s\_106, c=>cout\_105, sum=>s\_205, cout=>cout\_205 );

F2\_06: full\_adder port map (a=>pp(2)(5), b=>s\_107, c=>cout\_106, sum=>s\_206, cout=>cout\_206 );

F2\_07: full\_adder port map (a=>pp(2)(6), b=>s\_108, c=>cout\_107, sum=>s\_207, cout=>cout\_207 );

F2\_08: full\_adder port map (a=>pp(2)(7), b=>s\_109, c=>cout\_108, sum=>s\_208, cout=>cout\_208 );

F2\_09: full\_adder port map (a=>pp(2)(8), b=>s\_110, c=>cout\_109, sum=>s\_209, cout=>cout\_209 );

F2\_10: full\_adder port map (a=>pp(2)(9), b=>s\_111, c=>cout\_110, sum=>s\_210, cout=>cout\_210 );

F2\_11: full\_adder port map (a=>pp(2)(10), b=>s\_112, c=>cout\_111, sum=>s\_211, cout=>cout\_211 );

F2\_12: full\_adder port map (a=>pp(2)(11), b=>s\_113, c=>cout\_112, sum=>s\_212, cout=>cout\_212 );

F2\_13: full\_adder port map (a=>pp(2)(12), b=>s\_114, c=>cout\_113, sum=>s\_213, cout=>cout\_213 );

F2\_14: full\_adder port map (a=>pp(2)(13), b=>s\_115, c=>cout\_114, sum=>s\_214, cout=>cout\_214 );

F2\_15: full\_adder port map (a=>pp(2)(14), b=>pp(1)(15), c=>cout\_115, sum=>s\_215, cout=>cout\_215 );

---------------------------------------------------------

--------------------------- stage 3

--pp(2)(15);

F3\_01: full\_adder port map (a=>pp(3)(0), b=>s\_202, c=>cout\_201, sum=>s\_301, cout=>cout\_301 );

F3\_02: full\_adder port map (a=>pp(3)(1), b=>s\_203, c=>cout\_202, sum=>s\_302, cout=>cout\_302 );

F3\_03: full\_adder port map (a=>pp(3)(2), b=>s\_204, c=>cout\_203, sum=>s\_303, cout=>cout\_303 );

F3\_04: full\_adder port map (a=>pp(3)(3), b=>s\_205, c=>cout\_204, sum=>s\_304, cout=>cout\_304 );

F3\_05: full\_adder port map (a=>pp(3)(4), b=>s\_206, c=>cout\_205, sum=>s\_305, cout=>cout\_305 );

F3\_06: full\_adder port map (a=>pp(3)(5), b=>s\_207, c=>cout\_206, sum=>s\_306, cout=>cout\_306 );

F3\_07: full\_adder port map (a=>pp(3)(6), b=>s\_208, c=>cout\_207, sum=>s\_307, cout=>cout\_307 );

F3\_08: full\_adder port map (a=>pp(3)(7), b=>s\_209, c=>cout\_208, sum=>s\_308, cout=>cout\_308 );

F3\_09: full\_adder port map (a=>pp(3)(8), b=>s\_210, c=>cout\_209, sum=>s\_309, cout=>cout\_309 );

F3\_10: full\_adder port map (a=>pp(3)(9), b=>s\_211, c=>cout\_210, sum=>s\_310, cout=>cout\_310 );

F3\_11: full\_adder port map (a=>pp(3)(10), b=>s\_212, c=>cout\_211, sum=>s\_311, cout=>cout\_311 );

F3\_12: full\_adder port map (a=>pp(3)(11), b=>s\_213, c=>cout\_212, sum=>s\_312, cout=>cout\_312 );

F3\_13: full\_adder port map (a=>pp(3)(12), b=>s\_214, c=>cout\_213, sum=>s\_313, cout=>cout\_313 );

F3\_14: full\_adder port map (a=>pp(3)(13), b=>s\_215, c=>cout\_214, sum=>s\_314, cout=>cout\_314 );

F3\_15: full\_adder port map (a=>pp(3)(14), b=>pp(2)(15), c=>cout\_215, sum=>s\_315, cout=>cout\_315 );

---------------------------------------------------------

--------------------------- stage 4

--pp(3)(15);

F4\_01: full\_adder port map (a=>pp(4)(0), b=>s\_302, c=>cout\_301, sum=>s\_401, cout=>cout\_401 );

F4\_02: full\_adder port map (a=>pp(4)(1), b=>s\_303, c=>cout\_302, sum=>s\_402, cout=>cout\_402 );

F4\_03: full\_adder port map (a=>pp(4)(2), b=>s\_304, c=>cout\_303, sum=>s\_403, cout=>cout\_403 );

F4\_04: full\_adder port map (a=>pp(4)(3), b=>s\_305, c=>cout\_304, sum=>s\_404, cout=>cout\_404 );

F4\_05: full\_adder port map (a=>pp(4)(4), b=>s\_306, c=>cout\_305, sum=>s\_405, cout=>cout\_405 );

F4\_06: full\_adder port map (a=>pp(4)(5), b=>s\_307, c=>cout\_306, sum=>s\_406, cout=>cout\_406 );

F4\_07: full\_adder port map (a=>pp(4)(6), b=>s\_308, c=>cout\_307, sum=>s\_407, cout=>cout\_407 );

F4\_08: full\_adder port map (a=>pp(4)(7), b=>s\_309, c=>cout\_308, sum=>s\_408, cout=>cout\_408 );

F4\_09: full\_adder port map (a=>pp(4)(8), b=>s\_310, c=>cout\_309, sum=>s\_409, cout=>cout\_409 );

F4\_10: full\_adder port map (a=>pp(4)(9), b=>s\_311, c=>cout\_310, sum=>s\_410, cout=>cout\_410 );

F4\_11: full\_adder port map (a=>pp(4)(10), b=>s\_312, c=>cout\_311, sum=>s\_411, cout=>cout\_411 );

F4\_12: full\_adder port map (a=>pp(4)(11), b=>s\_313, c=>cout\_312, sum=>s\_412, cout=>cout\_412 );

F4\_13: full\_adder port map (a=>pp(4)(12), b=>s\_314, c=>cout\_313, sum=>s\_413, cout=>cout\_413 );

F4\_14: full\_adder port map (a=>pp(4)(13), b=>s\_315, c=>cout\_314, sum=>s\_414, cout=>cout\_414 );

F4\_15: full\_adder port map (a=>pp(4)(14), b=>pp(3)(15), c=>cout\_315, sum=>s\_415, cout=>cout\_415 );

---------------------------------------------------------

--------------------------- stage 5

--pp(4)(15);

F5\_01: full\_adder port map (a=>pp(5)(0), b=>s\_402, c=>cout\_401, sum=>s\_501, cout=>cout\_501 );

F5\_02: full\_adder port map (a=>pp(5)(1), b=>s\_403, c=>cout\_402, sum=>s\_502, cout=>cout\_502 );

F5\_03: full\_adder port map (a=>pp(5)(2), b=>s\_404, c=>cout\_403, sum=>s\_503, cout=>cout\_503 );

F5\_04: full\_adder port map (a=>pp(5)(3), b=>s\_405, c=>cout\_404, sum=>s\_504, cout=>cout\_504 );

F5\_05: full\_adder port map (a=>pp(5)(4), b=>s\_406, c=>cout\_405, sum=>s\_505, cout=>cout\_505 );

F5\_06: full\_adder port map (a=>pp(5)(5), b=>s\_407, c=>cout\_406, sum=>s\_506, cout=>cout\_506 );

F5\_07: full\_adder port map (a=>pp(5)(6), b=>s\_408, c=>cout\_407, sum=>s\_507, cout=>cout\_507 );

F5\_08: full\_adder port map (a=>pp(5)(7), b=>s\_409, c=>cout\_408, sum=>s\_508, cout=>cout\_508 );

F5\_09: full\_adder port map (a=>pp(5)(8), b=>s\_410, c=>cout\_409, sum=>s\_509, cout=>cout\_509 );

F5\_10: full\_adder port map (a=>pp(5)(9), b=>s\_411, c=>cout\_410, sum=>s\_510, cout=>cout\_510 );

F5\_11: full\_adder port map (a=>pp(5)(10), b=>s\_412, c=>cout\_411, sum=>s\_511, cout=>cout\_511 );

F5\_12: full\_adder port map (a=>pp(5)(11), b=>s\_413, c=>cout\_412, sum=>s\_512, cout=>cout\_512 );

F5\_13: full\_adder port map (a=>pp(5)(12), b=>s\_414, c=>cout\_413, sum=>s\_513, cout=>cout\_513 );

F5\_14: full\_adder port map (a=>pp(5)(13), b=>s\_415, c=>cout\_414, sum=>s\_514, cout=>cout\_514 );

F5\_15: full\_adder port map (a=>pp(5)(14), b=>pp(4)(15), c=>cout\_415, sum=>s\_515, cout=>cout\_515 );

---------------------------------------------------------

--------------------------- stage 6

--pp(5)(15);

F6\_01: full\_adder port map (a=>pp(6)(0), b=>s\_502, c=>cout\_501, sum=>s\_601, cout=>cout\_601 );

F6\_02: full\_adder port map (a=>pp(6)(1), b=>s\_503, c=>cout\_502, sum=>s\_602, cout=>cout\_602 );

F6\_03: full\_adder port map (a=>pp(6)(2), b=>s\_504, c=>cout\_503, sum=>s\_603, cout=>cout\_603 );

F6\_04: full\_adder port map (a=>pp(6)(3), b=>s\_505, c=>cout\_504, sum=>s\_604, cout=>cout\_604 );

F6\_05: full\_adder port map (a=>pp(6)(4), b=>s\_506, c=>cout\_505, sum=>s\_605, cout=>cout\_605 );

F6\_06: full\_adder port map (a=>pp(6)(5), b=>s\_507, c=>cout\_506, sum=>s\_606, cout=>cout\_606 );

F6\_07: full\_adder port map (a=>pp(6)(6), b=>s\_508, c=>cout\_507, sum=>s\_607, cout=>cout\_607 );

F6\_08: full\_adder port map (a=>pp(6)(7), b=>s\_509, c=>cout\_508, sum=>s\_608, cout=>cout\_608 );

F6\_09: full\_adder port map (a=>pp(6)(8), b=>s\_510, c=>cout\_509, sum=>s\_609, cout=>cout\_609 );

F6\_10: full\_adder port map (a=>pp(6)(9), b=>s\_511, c=>cout\_510, sum=>s\_610, cout=>cout\_610 );

F6\_11: full\_adder port map (a=>pp(6)(10), b=>s\_512, c=>cout\_511, sum=>s\_611, cout=>cout\_611 );

F6\_12: full\_adder port map (a=>pp(6)(11), b=>s\_513, c=>cout\_512, sum=>s\_612, cout=>cout\_612 );

F6\_13: full\_adder port map (a=>pp(6)(12), b=>s\_514, c=>cout\_513, sum=>s\_613, cout=>cout\_613 );

F6\_14: full\_adder port map (a=>pp(6)(13), b=>s\_515, c=>cout\_514, sum=>s\_614, cout=>cout\_614 );

F6\_15: full\_adder port map (a=>pp(6)(14), b=>pp(5)(15), c=>cout\_515, sum=>s\_615, cout=>cout\_615 );

---------------------------------------------------------

--------------------------- stage 7

--pp(6)(15);

F7\_01: full\_adder port map (a=>pp(7)(0), b=>s\_602, c=>cout\_601, sum=>s\_701, cout=>cout\_701 );

F7\_02: full\_adder port map (a=>pp(7)(1), b=>s\_603, c=>cout\_602, sum=>s\_702, cout=>cout\_702 );

F7\_03: full\_adder port map (a=>pp(7)(2), b=>s\_604, c=>cout\_603, sum=>s\_703, cout=>cout\_703 );

F7\_04: full\_adder port map (a=>pp(7)(3), b=>s\_605, c=>cout\_604, sum=>s\_704, cout=>cout\_704 );

F7\_05: full\_adder port map (a=>pp(7)(4), b=>s\_606, c=>cout\_605, sum=>s\_705, cout=>cout\_705 );

F7\_06: full\_adder port map (a=>pp(7)(5), b=>s\_607, c=>cout\_606, sum=>s\_706, cout=>cout\_706 );

F7\_07: full\_adder port map (a=>pp(7)(6), b=>s\_608, c=>cout\_607, sum=>s\_707, cout=>cout\_707 );

F7\_08: full\_adder port map (a=>pp(7)(7), b=>s\_609, c=>cout\_608, sum=>s\_708, cout=>cout\_708 );

F7\_09: full\_adder port map (a=>pp(7)(8), b=>s\_610, c=>cout\_609, sum=>s\_709, cout=>cout\_709 );

F7\_10: full\_adder port map (a=>pp(7)(9), b=>s\_611, c=>cout\_610, sum=>s\_710, cout=>cout\_710 );

F7\_11: full\_adder port map (a=>pp(7)(10), b=>s\_612, c=>cout\_611, sum=>s\_711, cout=>cout\_711 );

F7\_12: full\_adder port map (a=>pp(7)(11), b=>s\_613, c=>cout\_612, sum=>s\_712, cout=>cout\_712 );

F7\_13: full\_adder port map (a=>pp(7)(12), b=>s\_614, c=>cout\_613, sum=>s\_713, cout=>cout\_713 );

F7\_14: full\_adder port map (a=>pp(7)(13), b=>s\_615, c=>cout\_614, sum=>s\_714, cout=>cout\_714 );

F7\_15: full\_adder port map (a=>pp(7)(14), b=>pp(6)(15), c=>cout\_615, sum=>s\_715, cout=>cout\_715 );

---------------------------------------------------------

--------------------------- stage 8

--pp(7)(15);

F8\_01: full\_adder port map (a=>pp(8)(0), b=>s\_702, c=>cout\_701, sum=>s\_801, cout=>cout\_801 );

F8\_02: full\_adder port map (a=>pp(8)(1), b=>s\_703, c=>cout\_702, sum=>s\_802, cout=>cout\_802 );

F8\_03: full\_adder port map (a=>pp(8)(2), b=>s\_704, c=>cout\_703, sum=>s\_803, cout=>cout\_803 );

F8\_04: full\_adder port map (a=>pp(8)(3), b=>s\_705, c=>cout\_704, sum=>s\_804, cout=>cout\_804 );

F8\_05: full\_adder port map (a=>pp(8)(4), b=>s\_706, c=>cout\_705, sum=>s\_805, cout=>cout\_805 );

F8\_06: full\_adder port map (a=>pp(8)(5), b=>s\_707, c=>cout\_706, sum=>s\_806, cout=>cout\_806 );

F8\_07: full\_adder port map (a=>pp(8)(6), b=>s\_708, c=>cout\_707, sum=>s\_807, cout=>cout\_807 );

F8\_08: full\_adder port map (a=>pp(8)(7), b=>s\_709, c=>cout\_708, sum=>s\_808, cout=>cout\_808 );

F8\_09: full\_adder port map (a=>pp(8)(8), b=>s\_710, c=>cout\_709, sum=>s\_809, cout=>cout\_809 );

F8\_10: full\_adder port map (a=>pp(8)(9), b=>s\_711, c=>cout\_710, sum=>s\_810, cout=>cout\_810 );

F8\_11: full\_adder port map (a=>pp(8)(10), b=>s\_712, c=>cout\_711, sum=>s\_811, cout=>cout\_811 );

F8\_12: full\_adder port map (a=>pp(8)(11), b=>s\_713, c=>cout\_712, sum=>s\_812, cout=>cout\_812 );

F8\_13: full\_adder port map (a=>pp(8)(12), b=>s\_714, c=>cout\_713, sum=>s\_813, cout=>cout\_813 );

F8\_14: full\_adder port map (a=>pp(8)(13), b=>s\_715, c=>cout\_714, sum=>s\_814, cout=>cout\_814 );

F8\_15: full\_adder port map (a=>pp(8)(14), b=>pp(7)(15), c=>cout\_715, sum=>s\_815, cout=>cout\_815 );

---------------------------------------------------------

--------------------------- stage 9

--pp(8)(15);

F9\_01: full\_adder port map (a=>pp(9)(0), b=>s\_802, c=>cout\_801, sum=>s\_901, cout=>cout\_901 );

F9\_02: full\_adder port map (a=>pp(9)(1), b=>s\_803, c=>cout\_802, sum=>s\_902, cout=>cout\_902 );

F9\_03: full\_adder port map (a=>pp(9)(2), b=>s\_804, c=>cout\_803, sum=>s\_903, cout=>cout\_903 );

F9\_04: full\_adder port map (a=>pp(9)(3), b=>s\_805, c=>cout\_804, sum=>s\_904, cout=>cout\_904 );

F9\_05: full\_adder port map (a=>pp(9)(4), b=>s\_806, c=>cout\_805, sum=>s\_905, cout=>cout\_905 );

F9\_06: full\_adder port map (a=>pp(9)(5), b=>s\_807, c=>cout\_806, sum=>s\_906, cout=>cout\_906 );

F9\_07: full\_adder port map (a=>pp(9)(6), b=>s\_808, c=>cout\_807, sum=>s\_907, cout=>cout\_907 );

F9\_08: full\_adder port map (a=>pp(9)(7), b=>s\_809, c=>cout\_808, sum=>s\_908, cout=>cout\_908 );

F9\_09: full\_adder port map (a=>pp(9)(8), b=>s\_810, c=>cout\_809, sum=>s\_909, cout=>cout\_909 );

F9\_10: full\_adder port map (a=>pp(9)(9), b=>s\_811, c=>cout\_810, sum=>s\_910, cout=>cout\_910 );

F9\_11: full\_adder port map (a=>pp(9)(10), b=>s\_812, c=>cout\_811, sum=>s\_911, cout=>cout\_911 );

F9\_12: full\_adder port map (a=>pp(9)(11), b=>s\_813, c=>cout\_812, sum=>s\_912, cout=>cout\_912 );

F9\_13: full\_adder port map (a=>pp(9)(12), b=>s\_814, c=>cout\_813, sum=>s\_913, cout=>cout\_913 );

F9\_14: full\_adder port map (a=>pp(9)(13), b=>s\_815, c=>cout\_814, sum=>s\_914, cout=>cout\_914 );

F9\_15: full\_adder port map (a=>pp(9)(14), b=>pp(8)(15), c=>cout\_815, sum=>s\_915, cout=>cout\_915 );

---------------------------------------------------------

--------------------------- stage 10

--pp(9)(15);

F10\_01: full\_adder port map (a=>pp(10)(0), b=>s\_902, c=>cout\_901, sum=>s\_1001, cout=>cout\_1001 );

F10\_02: full\_adder port map (a=>pp(10)(1), b=>s\_903, c=>cout\_902, sum=>s\_1002, cout=>cout\_1002 );

F10\_03: full\_adder port map (a=>pp(10)(2), b=>s\_904, c=>cout\_903, sum=>s\_1003, cout=>cout\_1003 );

F10\_04: full\_adder port map (a=>pp(10)(3), b=>s\_905, c=>cout\_904, sum=>s\_1004, cout=>cout\_1004 );

F10\_05: full\_adder port map (a=>pp(10)(4), b=>s\_906, c=>cout\_905, sum=>s\_1005, cout=>cout\_1005 );

F10\_06: full\_adder port map (a=>pp(10)(5), b=>s\_907, c=>cout\_906, sum=>s\_1006, cout=>cout\_1006 );

F10\_07: full\_adder port map (a=>pp(10)(6), b=>s\_908, c=>cout\_907, sum=>s\_1007, cout=>cout\_1007 );

F10\_08: full\_adder port map (a=>pp(10)(7), b=>s\_909, c=>cout\_908, sum=>s\_1008, cout=>cout\_1008 );

F10\_09: full\_adder port map (a=>pp(10)(8), b=>s\_910, c=>cout\_909, sum=>s\_1009, cout=>cout\_1009 );

F10\_10: full\_adder port map (a=>pp(10)(9), b=>s\_911, c=>cout\_910, sum=>s\_1010, cout=>cout\_1010 );

F10\_11: full\_adder port map (a=>pp(10)(10), b=>s\_912, c=>cout\_911, sum=>s\_1011, cout=>cout\_1011 );

F10\_12: full\_adder port map (a=>pp(10)(11), b=>s\_913, c=>cout\_912, sum=>s\_1012, cout=>cout\_1012 );

F10\_13: full\_adder port map (a=>pp(10)(12), b=>s\_914, c=>cout\_913, sum=>s\_1013, cout=>cout\_1013 );

F10\_14: full\_adder port map (a=>pp(10)(13), b=>s\_915, c=>cout\_914, sum=>s\_1014, cout=>cout\_1014 );

F10\_15: full\_adder port map (a=>pp(10)(14), b=>pp(9)(15), c=>cout\_915, sum=>s\_1015, cout=>cout\_1015 );

---------------------------------------------------------

--------------------------- stage 11

--pp(10)(15);

F11\_01: full\_adder port map (a=>pp(11)(0), b=>s\_1002, c=>cout\_1001, sum=>s\_1101, cout=>cout\_1101 );

F11\_02: full\_adder port map (a=>pp(11)(1), b=>s\_1003, c=>cout\_1002, sum=>s\_1102, cout=>cout\_1102 );

F11\_03: full\_adder port map (a=>pp(11)(2), b=>s\_1004, c=>cout\_1003, sum=>s\_1103, cout=>cout\_1103 );

F11\_04: full\_adder port map (a=>pp(11)(3), b=>s\_1005, c=>cout\_1004, sum=>s\_1104, cout=>cout\_1104 );

F11\_05: full\_adder port map (a=>pp(11)(4), b=>s\_1006, c=>cout\_1005, sum=>s\_1105, cout=>cout\_1105 );

F11\_06: full\_adder port map (a=>pp(11)(5), b=>s\_1007, c=>cout\_1006, sum=>s\_1106, cout=>cout\_1106 );

F11\_07: full\_adder port map (a=>pp(11)(6), b=>s\_1008, c=>cout\_1007, sum=>s\_1107, cout=>cout\_1107 );

F11\_08: full\_adder port map (a=>pp(11)(7), b=>s\_1009, c=>cout\_1008, sum=>s\_1108, cout=>cout\_1108 );

F11\_09: full\_adder port map (a=>pp(11)(8), b=>s\_1010, c=>cout\_1009, sum=>s\_1109, cout=>cout\_1109 );

F11\_10: full\_adder port map (a=>pp(11)(9), b=>s\_1011, c=>cout\_1010, sum=>s\_1110, cout=>cout\_1110 );

F11\_11: full\_adder port map (a=>pp(11)(10), b=>s\_1012, c=>cout\_1011, sum=>s\_1111, cout=>cout\_1111 );

F11\_12: full\_adder port map (a=>pp(11)(11), b=>s\_1013, c=>cout\_1012, sum=>s\_1112, cout=>cout\_1112 );

F11\_13: full\_adder port map (a=>pp(11)(12), b=>s\_1014, c=>cout\_1013, sum=>s\_1113, cout=>cout\_1113 );

F11\_14: full\_adder port map (a=>pp(11)(13), b=>s\_1015, c=>cout\_1014, sum=>s\_1114, cout=>cout\_1114 );

F11\_15: full\_adder port map (a=>pp(11)(14), b=>pp(10)(15), c=>cout\_1015, sum=>s\_1115, cout=>cout\_1115 );

---------------------------------------------------------

--------------------------- stage 12

--pp(11)(15);

F12\_01: full\_adder port map (a=>pp(12)(0), b=>s\_1102, c=>cout\_1101, sum=>s\_1201, cout=>cout\_1201 );

F12\_02: full\_adder port map (a=>pp(12)(1), b=>s\_1103, c=>cout\_1102, sum=>s\_1202, cout=>cout\_1202 );

F12\_03: full\_adder port map (a=>pp(12)(2), b=>s\_1104, c=>cout\_1103, sum=>s\_1203, cout=>cout\_1203 );

F12\_04: full\_adder port map (a=>pp(12)(3), b=>s\_1105, c=>cout\_1104, sum=>s\_1204, cout=>cout\_1204 );

F12\_05: full\_adder port map (a=>pp(12)(4), b=>s\_1106, c=>cout\_1105, sum=>s\_1205, cout=>cout\_1205 );

F12\_06: full\_adder port map (a=>pp(12)(5), b=>s\_1107, c=>cout\_1106, sum=>s\_1206, cout=>cout\_1206 );

F12\_07: full\_adder port map (a=>pp(12)(6), b=>s\_1108, c=>cout\_1107, sum=>s\_1207, cout=>cout\_1207 );

F12\_08: full\_adder port map (a=>pp(12)(7), b=>s\_1109, c=>cout\_1108, sum=>s\_1208, cout=>cout\_1208 );

F12\_09: full\_adder port map (a=>pp(12)(8), b=>s\_1110, c=>cout\_1109, sum=>s\_1209, cout=>cout\_1209 );

F12\_10: full\_adder port map (a=>pp(12)(9), b=>s\_1111, c=>cout\_1110, sum=>s\_1210, cout=>cout\_1210 );

F12\_11: full\_adder port map (a=>pp(12)(10), b=>s\_1112, c=>cout\_1111, sum=>s\_1211, cout=>cout\_1211 );

F12\_12: full\_adder port map (a=>pp(12)(11), b=>s\_1113, c=>cout\_1112, sum=>s\_1212, cout=>cout\_1212 );

F12\_13: full\_adder port map (a=>pp(12)(12), b=>s\_1114, c=>cout\_1113, sum=>s\_1213, cout=>cout\_1213 );

F12\_14: full\_adder port map (a=>pp(12)(13), b=>s\_1115, c=>cout\_1114, sum=>s\_1214, cout=>cout\_1214 );

F12\_15: full\_adder port map (a=>pp(12)(14), b=>pp(11)(15), c=>cout\_1115, sum=>s\_1215, cout=>cout\_1215 );

---------------------------------------------------------

--------------------------- stage 13

--pp(12)(15);

F13\_01: full\_adder port map (a=>pp(13)(0), b=>s\_1202, c=>cout\_1201, sum=>s\_1301, cout=>cout\_1301 );

F13\_02: full\_adder port map (a=>pp(13)(1), b=>s\_1203, c=>cout\_1202, sum=>s\_1302, cout=>cout\_1302 );

F13\_03: full\_adder port map (a=>pp(13)(2), b=>s\_1204, c=>cout\_1203, sum=>s\_1303, cout=>cout\_1303 );

F13\_04: full\_adder port map (a=>pp(13)(3), b=>s\_1205, c=>cout\_1204, sum=>s\_1304, cout=>cout\_1304 );

F13\_05: full\_adder port map (a=>pp(13)(4), b=>s\_1206, c=>cout\_1205, sum=>s\_1305, cout=>cout\_1305 );

F13\_06: full\_adder port map (a=>pp(13)(5), b=>s\_1207, c=>cout\_1206, sum=>s\_1306, cout=>cout\_1306 );

F13\_07: full\_adder port map (a=>pp(13)(6), b=>s\_1208, c=>cout\_1207, sum=>s\_1307, cout=>cout\_1307 );

F13\_08: full\_adder port map (a=>pp(13)(7), b=>s\_1209, c=>cout\_1208, sum=>s\_1308, cout=>cout\_1308 );

F13\_09: full\_adder port map (a=>pp(13)(8), b=>s\_1210, c=>cout\_1209, sum=>s\_1309, cout=>cout\_1309 );

F13\_10: full\_adder port map (a=>pp(13)(9), b=>s\_1211, c=>cout\_1210, sum=>s\_1310, cout=>cout\_1310 );

F13\_11: full\_adder port map (a=>pp(13)(10), b=>s\_1212, c=>cout\_1211, sum=>s\_1311, cout=>cout\_1311 );

F13\_12: full\_adder port map (a=>pp(13)(11), b=>s\_1213, c=>cout\_1212, sum=>s\_1312, cout=>cout\_1312 );

F13\_13: full\_adder port map (a=>pp(13)(12), b=>s\_1214, c=>cout\_1213, sum=>s\_1313, cout=>cout\_1313 );

F13\_14: full\_adder port map (a=>pp(13)(13), b=>s\_1215, c=>cout\_1214, sum=>s\_1314, cout=>cout\_1314 );

F13\_15: full\_adder port map (a=>pp(13)(14), b=>pp(12)(15), c=>cout\_1215, sum=>s\_1315, cout=>cout\_1315 );

---------------------------------------------------------

--------------------------- stage 14

--pp(13)(15);

F14\_01: full\_adder port map (a=>pp(14)(0), b=>s\_1302, c=>cout\_1301, sum=>s\_1401, cout=>cout\_1401 );

F14\_02: full\_adder port map (a=>pp(14)(1), b=>s\_1303, c=>cout\_1302, sum=>s\_1402, cout=>cout\_1402 );

F14\_03: full\_adder port map (a=>pp(14)(2), b=>s\_1304, c=>cout\_1303, sum=>s\_1403, cout=>cout\_1403 );

F14\_04: full\_adder port map (a=>pp(14)(3), b=>s\_1305, c=>cout\_1304, sum=>s\_1404, cout=>cout\_1404 );

F14\_05: full\_adder port map (a=>pp(14)(4), b=>s\_1306, c=>cout\_1305, sum=>s\_1405, cout=>cout\_1405 );

F14\_06: full\_adder port map (a=>pp(14)(5), b=>s\_1307, c=>cout\_1306, sum=>s\_1406, cout=>cout\_1406 );

F14\_07: full\_adder port map (a=>pp(14)(6), b=>s\_1308, c=>cout\_1307, sum=>s\_1407, cout=>cout\_1407 );

F14\_08: full\_adder port map (a=>pp(14)(7), b=>s\_1309, c=>cout\_1308, sum=>s\_1408, cout=>cout\_1408 );

F14\_09: full\_adder port map (a=>pp(14)(8), b=>s\_1310, c=>cout\_1309, sum=>s\_1409, cout=>cout\_1409 );

F14\_10: full\_adder port map (a=>pp(14)(9), b=>s\_1311, c=>cout\_1310, sum=>s\_1410, cout=>cout\_1410 );

F14\_11: full\_adder port map (a=>pp(14)(10), b=>s\_1312, c=>cout\_1311, sum=>s\_1411, cout=>cout\_1411 );

F14\_12: full\_adder port map (a=>pp(14)(11), b=>s\_1313, c=>cout\_1312, sum=>s\_1412, cout=>cout\_1412 );

F14\_13: full\_adder port map (a=>pp(14)(12), b=>s\_1314, c=>cout\_1313, sum=>s\_1413, cout=>cout\_1413 );

F14\_14: full\_adder port map (a=>pp(14)(13), b=>s\_1315, c=>cout\_1314, sum=>s\_1414, cout=>cout\_1414 );

F14\_15: full\_adder port map (a=>pp(14)(14), b=>pp(13)(15), c=>cout\_1315, sum=>s\_1415, cout=>cout\_1415 );

---------------------------------------------------------

--------------------------- stage 15

--pp(14)(15);

F15\_01: full\_adder port map (a=>pp(15)(0), b=>s\_1402, c=>cout\_1401, sum=>s\_1501, cout=>cout\_1501 );

F15\_02: full\_adder port map (a=>pp(15)(1), b=>s\_1403, c=>cout\_1402, sum=>s\_1502, cout=>cout\_1502 );

F15\_03: full\_adder port map (a=>pp(15)(2), b=>s\_1404, c=>cout\_1403, sum=>s\_1503, cout=>cout\_1503 );

F15\_04: full\_adder port map (a=>pp(15)(3), b=>s\_1405, c=>cout\_1404, sum=>s\_1504, cout=>cout\_1504 );

F15\_05: full\_adder port map (a=>pp(15)(4), b=>s\_1406, c=>cout\_1405, sum=>s\_1505, cout=>cout\_1505 );

F15\_06: full\_adder port map (a=>pp(15)(5), b=>s\_1407, c=>cout\_1406, sum=>s\_1506, cout=>cout\_1506 );

F15\_07: full\_adder port map (a=>pp(15)(6), b=>s\_1408, c=>cout\_1407, sum=>s\_1507, cout=>cout\_1507 );

F15\_08: full\_adder port map (a=>pp(15)(7), b=>s\_1409, c=>cout\_1408, sum=>s\_1508, cout=>cout\_1508 );

F15\_09: full\_adder port map (a=>pp(15)(8), b=>s\_1410, c=>cout\_1409, sum=>s\_1509, cout=>cout\_1509 );

F15\_10: full\_adder port map (a=>pp(15)(9), b=>s\_1411, c=>cout\_1410, sum=>s\_1510, cout=>cout\_1510 );

F15\_11: full\_adder port map (a=>pp(15)(10), b=>s\_1412, c=>cout\_1411, sum=>s\_1511, cout=>cout\_1511 );

F15\_12: full\_adder port map (a=>pp(15)(11), b=>s\_1413, c=>cout\_1412, sum=>s\_1512, cout=>cout\_1512 );

F15\_13: full\_adder port map (a=>pp(15)(12), b=>s\_1414, c=>cout\_1413, sum=>s\_1513, cout=>cout\_1513 );

F15\_14: full\_adder port map (a=>pp(15)(13), b=>s\_1415, c=>cout\_1414, sum=>s\_1514, cout=>cout\_1514 );

F15\_15: full\_adder port map (a=>pp(15)(14), b=>pp(14)(15), c=>cout\_1415, sum=>s\_1515, cout=>cout\_1515 );

---------------------------------------------------------

--------------------------- stage 16

--pp(15)(15);

F16\_01: full\_adder port map (a=>s\_1502, b=>cout\_1501, c=>'0', sum=>p16, cout=>cout\_1601 );

F16\_02: full\_adder port map (a=>s\_1503, b=>cout\_1502, c=>cout\_1601, sum=>p17, cout=>cout\_1602 );

F16\_03: full\_adder port map (a=>s\_1504, b=>cout\_1503, c=>cout\_1602, sum=>p18, cout=>cout\_1603 );

F16\_04: full\_adder port map (a=>s\_1505, b=>cout\_1504, c=>cout\_1603, sum=>p19, cout=>cout\_1604 );

F16\_05: full\_adder port map (a=>s\_1506, b=>cout\_1505, c=>cout\_1604, sum=>p20, cout=>cout\_1605 );

F16\_06: full\_adder port map (a=>s\_1507, b=>cout\_1506, c=>cout\_1605, sum=>p21, cout=>cout\_1606 );

F16\_07: full\_adder port map (a=>s\_1508, b=>cout\_1507, c=>cout\_1606, sum=>p22, cout=>cout\_1607 );

F16\_08: full\_adder port map (a=>s\_1509, b=>cout\_1508, c=>cout\_1607, sum=>p23, cout=>cout\_1608 );

F16\_09: full\_adder port map (a=>s\_1510, b=>cout\_1509, c=>cout\_1608, sum=>p24, cout=>cout\_1609 );

F16\_10: full\_adder port map (a=>s\_1511, b=>cout\_1510, c=>cout\_1609, sum=>p25, cout=>cout\_1610 );

F16\_11: full\_adder port map (a=>s\_1512, b=>cout\_1511, c=>cout\_1610, sum=>p26, cout=>cout\_1611 );

F16\_12: full\_adder port map (a=>s\_1513, b=>cout\_1512, c=>cout\_1611, sum=>p27, cout=>cout\_1612 );

F16\_13: full\_adder port map (a=>s\_1514, b=>cout\_1513, c=>cout\_1612, sum=>p28, cout=>cout\_1613 );

F16\_14: full\_adder port map (a=>s\_1515, b=>cout\_1514, c=>cout\_1613, sum=>p29, cout=>cout\_1614 );

F16\_15: full\_adder port map (a=>pp(15)(15), b=>cout\_1515, c=>cout\_1614, sum=>p30, cout=>p31 );

p0 <= pp(0)(0);

p1 <= s\_101 ;

p2 <= s\_201 ;

p3 <= s\_301 ;

p4 <= s\_401 ;

p5 <= s\_501 ;

p6 <= s\_601 ;

p7 <= s\_701 ;

p8 <= s\_801 ;

p9 <= s\_901 ;

p10 <= s\_1001;

p11 <= s\_1101;

p12 <= s\_1201;

p13 <= s\_1301;

p14 <= s\_1401;

p15 <= s\_1501;

prod(0) <= pp(0)(0);

prod(1) <= s\_101 ;

prod(2) <= s\_201 ;

prod(3) <= s\_301 ;

prod(4) <= s\_401 ;

prod(5) <= s\_501 ;

prod(6) <= s\_601 ;

prod(7) <= s\_701 ;

prod(8) <= s\_801 ;

prod(9) <= s\_901 ;

prod(10) <= s\_1001;

prod(11) <= s\_1101;

prod(12) <= s\_1201;

prod(13) <= s\_1301;

prod(14) <= s\_1401;

prod(15) <= s\_1501;

prod(16) <= p16;

prod(17) <= p17;

prod(18) <= p18;

prod(19) <= p19;

prod(20) <= p20;

prod(21) <= p21;

prod(22) <= p22;

prod(23) <= p23;

prod(24) <= p24;

prod(25) <= p25;

prod(26) <= p26;

prod(27) <= p27;

prod(28) <= p28;

prod(29) <= p29;

prod(30) <= p30;

prod(31) <= p31;

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity wallace16\_tb is

-- Port ( );

end wallace16\_tb;

architecture Behavioral of wallace16\_tb is

component wallace16 is

Port ( A : in STD\_LOGIC\_VECTOR (15 downto 0);

B : in STD\_LOGIC\_VECTOR (15 downto 0);

prod : out STD\_LOGIC\_VECTOR (31 downto 0)

);

end component;

signal X: STD\_LOGIC\_VECTOR(15 downto 0) := x"0000";

signal Y: STD\_LOGIC\_VECTOR(15 downto 0):= x"0000";

signal prod : STD\_LOGIC\_VECTOR(31 downto 0) := x"00000000";

signal P: STD\_LOGIC\_VECTOR(31 downto 0) := x"00000000";

begin

DUT1 : wallace16 port map( A => X, B => Y, prod => P);

test: process

variable rez\_rau : integer := 0;

begin

for I in 0 to 50 loop

for J in 0 to 50 loop

X <= CONV\_STD\_LOGIC\_VECTOR(I, 16);

Y <= CONV\_STD\_LOGIC\_VECTOR(J, 16);

prod <= CONV\_STD\_LOGIC\_VECTOR(I\*J, 32);

wait for 10 ns;

if (P /= prod) then

rez\_rau := rez\_rau + 1;

report "Rezultat asteptat (" & STD\_LOGIC'image (P(31)) &

STD\_LOGIC'image (P(30)) &

STD\_LOGIC'image (P(29)) & STD\_LOGIC'image (P(28)) &

STD\_LOGIC'image (P(27)) & STD\_LOGIC'image (P(26)) &

STD\_LOGIC'image (P(25)) & STD\_LOGIC'image (P(24)) &

STD\_LOGIC'image (P(23)) & STD\_LOGIC'image (P(22)) &

STD\_LOGIC'image (P(21)) & STD\_LOGIC'image (P(20)) &

STD\_LOGIC'image (P(19)) & STD\_LOGIC'image (P(18)) &

STD\_LOGIC'image (P(17)) & STD\_LOGIC'image (P(16)) &

STD\_LOGIC'image (P(15)) & STD\_LOGIC'image (P(14)) &

STD\_LOGIC'image (P(13)) & STD\_LOGIC'image (P(12)) &

STD\_LOGIC'image (P(11)) & STD\_LOGIC'image (P(10)) &

STD\_LOGIC'image (P(9)) & STD\_LOGIC'image (P(8)) &

STD\_LOGIC'image (P(7)) & STD\_LOGIC'image (P(6)) &

STD\_LOGIC'image (P(5)) & STD\_LOGIC'image (P(4)) &

STD\_LOGIC'image (P(3)) & STD\_LOGIC'image (P(2)) &

STD\_LOGIC'image (P(1)) & STD\_LOGIC'image (P(0))

& ") /= Valoare obtinuta (" & STD\_LOGIC'image (prod(31)) &

STD\_LOGIC'image (prod(30)) &

STD\_LOGIC'image (prod(29)) & STD\_LOGIC'image (prod(28)) &

STD\_LOGIC'image (prod(27)) & STD\_LOGIC'image (prod(26)) &

STD\_LOGIC'image (prod(25)) & STD\_LOGIC'image (prod(24)) &

STD\_LOGIC'image (prod(23)) & STD\_LOGIC'image (prod(22)) &

STD\_LOGIC'image (prod(21)) & STD\_LOGIC'image (prod(20)) &

STD\_LOGIC'image (prod(19)) & STD\_LOGIC'image (prod(18)) &

STD\_LOGIC'image (prod(17)) & STD\_LOGIC'image (prod(16)) &

STD\_LOGIC'image (prod(15)) & STD\_LOGIC'image (prod(14)) &

STD\_LOGIC'image (prod(13)) & STD\_LOGIC'image (prod(12)) &

STD\_LOGIC'image (prod(11)) & STD\_LOGIC'image (prod(10)) &

STD\_LOGIC'image (prod(9)) & STD\_LOGIC'image (prod(8)) &

STD\_LOGIC'image (prod(7)) & STD\_LOGIC'image (prod(6)) &

STD\_LOGIC'image (prod(5)) & STD\_LOGIC'image (prod(4)) &

STD\_LOGIC'image (prod(3)) & STD\_LOGIC'image (prod(2)) &

STD\_LOGIC'image (prod(1)) & STD\_LOGIC'image (prod(0)) &

") la t = " & TIME'image (now)

severity ERROR;

end if;

end loop;

end loop;

for I in 4975 to 5025 loop

for J in 4975 to 5025 loop

X <= CONV\_STD\_LOGIC\_VECTOR(I, 16);

Y <= CONV\_STD\_LOGIC\_VECTOR(J, 16);

prod <= CONV\_STD\_LOGIC\_VECTOR(I\*J, 32);

wait for 10 ns;

if (P /= prod) then

rez\_rau := rez\_rau + 1;

report "Rezultat asteptat (" & STD\_LOGIC'image (P(31)) &

STD\_LOGIC'image (P(30)) &

STD\_LOGIC'image (P(29)) & STD\_LOGIC'image (P(28)) &

STD\_LOGIC'image (P(27)) & STD\_LOGIC'image (P(26)) &

STD\_LOGIC'image (P(25)) & STD\_LOGIC'image (P(24)) &

STD\_LOGIC'image (P(23)) & STD\_LOGIC'image (P(22)) &

STD\_LOGIC'image (P(21)) & STD\_LOGIC'image (P(20)) &

STD\_LOGIC'image (P(19)) & STD\_LOGIC'image (P(18)) &

STD\_LOGIC'image (P(17)) & STD\_LOGIC'image (P(16)) &

STD\_LOGIC'image (P(15)) & STD\_LOGIC'image (P(14)) &

STD\_LOGIC'image (P(13)) & STD\_LOGIC'image (P(12)) &

STD\_LOGIC'image (P(11)) & STD\_LOGIC'image (P(10)) &

STD\_LOGIC'image (P(9)) & STD\_LOGIC'image (P(8)) &

STD\_LOGIC'image (P(7)) & STD\_LOGIC'image (P(6)) &

STD\_LOGIC'image (P(5)) & STD\_LOGIC'image (P(4)) &

STD\_LOGIC'image (P(3)) & STD\_LOGIC'image (P(2)) &

STD\_LOGIC'image (P(1)) & STD\_LOGIC'image (P(0))

& ") /= Valoare obtinuta (" & STD\_LOGIC'image (prod(31)) &

STD\_LOGIC'image (prod(30)) &

STD\_LOGIC'image (prod(29)) & STD\_LOGIC'image (prod(28)) &

STD\_LOGIC'image (prod(27)) & STD\_LOGIC'image (prod(26)) &

STD\_LOGIC'image (prod(25)) & STD\_LOGIC'image (prod(24)) &

STD\_LOGIC'image (prod(23)) & STD\_LOGIC'image (prod(22)) &

STD\_LOGIC'image (prod(21)) & STD\_LOGIC'image (prod(20)) &

STD\_LOGIC'image (prod(19)) & STD\_LOGIC'image (prod(18)) &

STD\_LOGIC'image (prod(17)) & STD\_LOGIC'image (prod(16)) &

STD\_LOGIC'image (prod(15)) & STD\_LOGIC'image (prod(14)) &

STD\_LOGIC'image (prod(13)) & STD\_LOGIC'image (prod(12)) &

STD\_LOGIC'image (prod(11)) & STD\_LOGIC'image (prod(10)) &

STD\_LOGIC'image (prod(9)) & STD\_LOGIC'image (prod(8)) &

STD\_LOGIC'image (prod(7)) & STD\_LOGIC'image (prod(6)) &

STD\_LOGIC'image (prod(5)) & STD\_LOGIC'image (prod(4)) &

STD\_LOGIC'image (prod(3)) & STD\_LOGIC'image (prod(2)) &

STD\_LOGIC'image (prod(1)) & STD\_LOGIC'image (prod(0)) &

") la t = " & TIME'image (now)

severity ERROR;

end if;

end loop;

end loop;

for I in 9950 to 9999 loop

for J in 9950 to 9999 loop

X <= CONV\_STD\_LOGIC\_VECTOR(I, 16);

Y <= CONV\_STD\_LOGIC\_VECTOR(J, 16);

prod <= CONV\_STD\_LOGIC\_VECTOR(I\*J, 32);

wait for 10 ns;

if (P /= prod) then

rez\_rau := rez\_rau + 1;

report "Rezultat asteptat (" & STD\_LOGIC'image (P(31)) &

STD\_LOGIC'image (P(30)) &

STD\_LOGIC'image (P(29)) & STD\_LOGIC'image (P(28)) &

STD\_LOGIC'image (P(27)) & STD\_LOGIC'image (P(26)) &

STD\_LOGIC'image (P(25)) & STD\_LOGIC'image (P(24)) &

STD\_LOGIC'image (P(23)) & STD\_LOGIC'image (P(22)) &

STD\_LOGIC'image (P(21)) & STD\_LOGIC'image (P(20)) &

STD\_LOGIC'image (P(19)) & STD\_LOGIC'image (P(18)) &

STD\_LOGIC'image (P(17)) & STD\_LOGIC'image (P(16)) &

STD\_LOGIC'image (P(15)) & STD\_LOGIC'image (P(14)) &

STD\_LOGIC'image (P(13)) & STD\_LOGIC'image (P(12)) &

STD\_LOGIC'image (P(11)) & STD\_LOGIC'image (P(10)) &

STD\_LOGIC'image (P(9)) & STD\_LOGIC'image (P(8)) &

STD\_LOGIC'image (P(7)) & STD\_LOGIC'image (P(6)) &

STD\_LOGIC'image (P(5)) & STD\_LOGIC'image (P(4)) &

STD\_LOGIC'image (P(3)) & STD\_LOGIC'image (P(2)) &

STD\_LOGIC'image (P(1)) & STD\_LOGIC'image (P(0))

& ") /= Valoare obtinuta (" & STD\_LOGIC'image (prod(31)) &

STD\_LOGIC'image (prod(30)) &

STD\_LOGIC'image (prod(29)) & STD\_LOGIC'image (prod(28)) &

STD\_LOGIC'image (prod(27)) & STD\_LOGIC'image (prod(26)) &

STD\_LOGIC'image (prod(25)) & STD\_LOGIC'image (prod(24)) &

STD\_LOGIC'image (prod(23)) & STD\_LOGIC'image (prod(22)) &

STD\_LOGIC'image (prod(21)) & STD\_LOGIC'image (prod(20)) &

STD\_LOGIC'image (prod(19)) & STD\_LOGIC'image (prod(18)) &

STD\_LOGIC'image (prod(17)) & STD\_LOGIC'image (prod(16)) &

STD\_LOGIC'image (prod(15)) & STD\_LOGIC'image (prod(14)) &

STD\_LOGIC'image (prod(13)) & STD\_LOGIC'image (prod(12)) &

STD\_LOGIC'image (prod(11)) & STD\_LOGIC'image (prod(10)) &

STD\_LOGIC'image (prod(9)) & STD\_LOGIC'image (prod(8)) &

STD\_LOGIC'image (prod(7)) & STD\_LOGIC'image (prod(6)) &

STD\_LOGIC'image (prod(5)) & STD\_LOGIC'image (prod(4)) &

STD\_LOGIC'image (prod(3)) & STD\_LOGIC'image (prod(2)) &

STD\_LOGIC'image (prod(1)) & STD\_LOGIC'image (prod(0)) &

") la t = " & TIME'image (now)

severity ERROR;

end if;

end loop;

end loop;

if rez\_rau > 0 then

report "Numar erori = " & Integer'image(rez\_rau);

else

report "Nu sunt erori";

end if;

wait for 10 ns;

end process test;

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

entity MPG is

Port ( clk : in std\_logic;

rst,input:in std\_logic;

q\_out:out std\_logic

);

end MPG;

architecture Behavioral of MPG is

signal Q1, Q2, Q3 : std\_logic;

begin

process(clk)

begin

if (clk'event and clk = '1') then

if (rst = '1') then

Q1 <= '0';

Q2 <= '0';

Q3 <= '0';

else

Q1 <= input;

Q2 <= Q1;

Q3 <= Q2;

end if;

end if;

end process;

Q\_OUT <= Q1 and Q2 and (not Q3);

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

use IEEE.STD\_LOGIC\_ARITH.all;

entity displ7seg is

Port ( Clk : in STD\_LOGIC;

Rst : in STD\_LOGIC;

Data : in STD\_LOGIC\_VECTOR (31 downto 0); -- datele pentru 8 cifre (cifra 1 din stanga: biti 31..28)

An : out STD\_LOGIC\_VECTOR (7 downto 0); -- selectia anodului activ

Seg : out STD\_LOGIC\_VECTOR (7 downto 0)); -- selectia catozilor (segmentelor) cifrei active

end displ7seg;

architecture Behavioral of displ7seg is

constant CNT\_100HZ : integer := 2\*\*20; -- divizor pentru rata de reimprospatare de ~100 Hz (cu un ceas de 100 MHz)

signal Num : integer range 0 to CNT\_100HZ - 1 := 0;

signal NumV : STD\_LOGIC\_VECTOR (19 downto 0) := (others => '0');

signal LedSel : STD\_LOGIC\_VECTOR (2 downto 0) := (others => '0');

signal Hex : STD\_LOGIC\_VECTOR (3 downto 0) := (others => '0');

begin

-- Proces pentru divizarea ceasului

divclk: process (Clk)

begin

if (Clk'event and Clk = '1') then

if (Rst = '1') then

Num <= 0;

elsif (Num = CNT\_100HZ - 1) then

Num <= 0;

else

Num <= Num + 1;

end if;

end if;

end process;

NumV <= CONV\_STD\_LOGIC\_VECTOR (Num, 20);

LedSel <= NumV (19 downto 17);

-- Selectia anodului activ

An <= "11111110" when LedSel = "000" else

"11111101" when LedSel = "001" else

"11111011" when LedSel = "010" else

"11110111" when LedSel = "011" else

"11101111" when LedSel = "100" else

"11011111" when LedSel = "101" else

"10111111" when LedSel = "110" else

"01111111" when LedSel = "111" else

"11111111";

-- Selectia cifrei active

Hex <= Data (3 downto 0) when LedSel = "000" else

Data (7 downto 4) when LedSel = "001" else

Data (11 downto 8) when LedSel = "010" else

Data (15 downto 12) when LedSel = "011" else

Data (19 downto 16) when LedSel = "100" else

Data (23 downto 20) when LedSel = "101" else

Data (27 downto 24) when LedSel = "110" else

Data (31 downto 28) when LedSel = "111" else

X"0";

-- Activarea/dezactivarea segmentelor cifrei active

Seg <= "11111001" when Hex = "0001" else -- 1

"10100100" when Hex = "0010" else -- 2

"10110000" when Hex = "0011" else -- 3

"10011001" when Hex = "0100" else -- 4

"10010010" when Hex = "0101" else -- 5

"10000010" when Hex = "0110" else -- 6

"11111000" when Hex = "0111" else -- 7

"10000000" when Hex = "1000" else -- 8

"10010000" when Hex = "1001" else -- 9

"10001000" when Hex = "1010" else -- A

"10000011" when Hex = "1011" else -- b

"11000110" when Hex = "1100" else -- C

"10100001" when Hex = "1101" else -- d

"10000110" when Hex = "1110" else -- E

"10001110" when Hex = "1111" else -- F

"11000000"; -- 0

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity Inmultire\_wallace\_16\_mainn is

Port ( X : in STD\_LOGIC\_VECTOR(15 downto 0);

Y : in STD\_LOGIC\_VECTOR(15 downto 0);

Rst : in STD\_LOGIC;

Start : in STD\_LOGIC;

Start2 : in STD\_LOGIC;

Rez\_term : in std\_logic;

Clk : in STD\_LOGIC;

X\_primeste\_X : out STD\_LOGIC;

Y\_primeste\_X : out STD\_LOGIC;

An: out STD\_LOGIC\_VECTOR(7 downto 0);

Seg: out STD\_LOGIC\_VECTOR(7 downto 0);

Term : out STD\_LOGIC

);

end Inmultire\_wallace\_16\_mainn;

architecture Behavioral of Inmultire\_wallace\_16\_mainn is

component wallace16 is

Port ( A : in STD\_LOGIC\_VECTOR (15 downto 0);

B : in STD\_LOGIC\_VECTOR (15 downto 0);

prod : out STD\_LOGIC\_VECTOR (31 downto 0)

);

end component;

component displ7seg is

Port ( Clk : in STD\_LOGIC;

Rst : in STD\_LOGIC;

Data : in STD\_LOGIC\_VECTOR (31 downto 0); -- datele pentru 8 cifre (cifra 1 din stanga: biti 31..28)

An : out STD\_LOGIC\_VECTOR (7 downto 0); -- selectia anodului activ

Seg : out STD\_LOGIC\_VECTOR (7 downto 0)); -- selectia catozilor (segmentelor) cifrei active

end component;

component MPG is

Port ( clk : in std\_logic;

rst,input:in std\_logic;

q\_out:out std\_logic

);

end component;

signal nr1, nr2 : STD\_LOGIC\_VECTOR(15 downto 0) := x"0000";

signal Data\_aux : STD\_LOGIC\_VECTOR(31 downto 0);

signal P : STD\_LOGIC\_VECTOR(31 downto 0);

signal en\_start, en\_start2, en\_rez : std\_logic := '0';

signal x\_primeste\_x\_aux, y\_primeste\_x\_aux, term\_aux : std\_logic := '0';

signal rst\_aux : std\_logic := '0';

begin

mul\_CPA1 : wallace16 port map(A => nr1 , B => nr2, prod => P);

MPG\_start : MPG port map(clk=>Clk ,rst=>'0',input=>Start, q\_out=> en\_start);

MPG\_start2 : MPG port map(clk=>Clk ,rst=>'0',input=>Start2, q\_out=> en\_start2);

MPG\_term : MPG port map(clk=>Clk ,rst=>'0',input=>Rez\_term, q\_out=> en\_rez);

MPG\_reset : MPG port map(clk=>Clk ,rst=>'0',input=>Rst, q\_out=> rst\_aux);

p1 : process (clk, en\_start, en\_start2, en\_rez)

begin

if rising\_edge(clk) then

if Rst = '1' then

Y\_primeste\_X\_aux <= '0';

X\_primeste\_X\_aux <= '0';

term\_aux <= '0';

Data\_aux <= x"00000000";

end if;

if en\_start2 = '1' then

nr2 <= X;

Y\_primeste\_X\_aux <= '1';

X\_primeste\_X\_aux <= '0';

term\_aux <= '0';

Data\_aux <= x"0000" & nr2;

end if;

if en\_start = '1' then

nr1 <= X;

X\_primeste\_X\_aux <= '1';

Y\_primeste\_X\_aux <= '0';

term\_aux <= '0';

Data\_aux <= x"0000" & nr1;

end if;

if en\_rez = '1' then

term\_aux <= '1';

X\_primeste\_X\_aux <= '0';

Y\_primeste\_X\_aux <= '0';

Data\_aux <= P;

end if;

end if;

Y\_primeste\_X <= Y\_primeste\_X\_aux;

X\_primeste\_X <= X\_primeste\_X\_aux;

term <= term\_aux;

end process;

display1 : displ7seg port map(Clk => Clk, Rst => '0', Data => Data\_aux,

An => An, Seg => Seg);

end Behavioral;

**Anexa C**

Fișierul de constrângeri care este valabil pentru ambele proiecte

## This file is a general .xdc for the Nexys4 DDR Rev. C

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project

## Clock signal

set\_property -dict {PACKAGE\_PIN E3 IOSTANDARD LVCMOS33} [get\_ports Clk]

create\_clock -period 10.000 -name sys\_clk\_pin -waveform {0.000 5.000} -add [get\_ports Clk]

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { X[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { X[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { X[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { X[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { X[4] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { X[5] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { X[6] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { X[7] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { X[8] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { X[9] }]; #IO\_25\_34 Sch=sw[9]

set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { X[10] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { X[11] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { X[12] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { X[13] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { X[14] }]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { X[15] }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

## LEDs

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { X\_primeste\_X }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { Y\_primeste\_X }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { Term }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

#set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { LED[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

#set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { LED[4] }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

#set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { LED[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

#set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { LED[6] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

#set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { LED[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

#set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { LED[8] }]; #IO\_L16N\_T2\_A15\_D31\_14 Sch=led[8]

#set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { LED[9] }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

#set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { LED[10] }]; #IO\_L22P\_T3\_A05\_D21\_14 Sch=led[10]

#set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { LED[11] }]; #IO\_L15N\_T2\_DQS\_DOUT\_CSO\_B\_14 Sch=led[11]

#set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { LED[12] }]; #IO\_L16P\_T2\_CSI\_B\_14 Sch=led[12]

#set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { LED[13] }]; #IO\_L22N\_T3\_A04\_D20\_14 Sch=led[13]

#set\_property -dict { PACKAGE\_PIN V12 IOSTANDARD LVCMOS33 } [get\_ports { LED[14] }]; #IO\_L20N\_T3\_A07\_D23\_14 Sch=led[14]

#set\_property -dict { PACKAGE\_PIN V11 IOSTANDARD LVCMOS33 } [get\_ports { LED[15] }]; #IO\_L21N\_T3\_DQS\_A06\_D22\_14 Sch=led[15]

#set\_property -dict { PACKAGE\_PIN R12 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_B }]; #IO\_L5P\_T0\_D06\_14 Sch=led16\_b

#set\_property -dict { PACKAGE\_PIN M16 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_G }]; #IO\_L10P\_T1\_D14\_14 Sch=led16\_g

#set\_property -dict { PACKAGE\_PIN N15 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_R }]; #IO\_L11P\_T1\_SRCC\_14 Sch=led16\_r

#set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_B }]; #IO\_L15N\_T2\_DQS\_ADV\_B\_15 Sch=led17\_b

#set\_property -dict { PACKAGE\_PIN R11 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_G }]; #IO\_0\_14 Sch=led17\_g

#set\_property -dict { PACKAGE\_PIN N16 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_R }]; #IO\_L11N\_T1\_SRCC\_14 Sch=led17\_r

##7 segment display

set\_property -dict {PACKAGE\_PIN T10 IOSTANDARD LVCMOS33} [get\_ports {Seg[0]}]

set\_property -dict {PACKAGE\_PIN R10 IOSTANDARD LVCMOS33} [get\_ports {Seg[1]}]

set\_property -dict {PACKAGE\_PIN K16 IOSTANDARD LVCMOS33} [get\_ports {Seg[2]}]

set\_property -dict {PACKAGE\_PIN K13 IOSTANDARD LVCMOS33} [get\_ports {Seg[3]}]

set\_property -dict {PACKAGE\_PIN P15 IOSTANDARD LVCMOS33} [get\_ports {Seg[4]}]

set\_property -dict {PACKAGE\_PIN T11 IOSTANDARD LVCMOS33} [get\_ports {Seg[5]}]

set\_property -dict {PACKAGE\_PIN L18 IOSTANDARD LVCMOS33} [get\_ports {Seg[6]}]

set\_property -dict {PACKAGE\_PIN H15 IOSTANDARD LVCMOS33} [get\_ports {Seg[7]}]

set\_property -dict {PACKAGE\_PIN J17 IOSTANDARD LVCMOS33} [get\_ports {An[0]}]

set\_property -dict {PACKAGE\_PIN J18 IOSTANDARD LVCMOS33} [get\_ports {An[1]}]

set\_property -dict {PACKAGE\_PIN T9 IOSTANDARD LVCMOS33} [get\_ports {An[2]}]

set\_property -dict {PACKAGE\_PIN J14 IOSTANDARD LVCMOS33} [get\_ports {An[3]}]

set\_property -dict {PACKAGE\_PIN P14 IOSTANDARD LVCMOS33} [get\_ports {An[4]}]

set\_property -dict {PACKAGE\_PIN T14 IOSTANDARD LVCMOS33} [get\_ports {An[5]}]

set\_property -dict {PACKAGE\_PIN K2 IOSTANDARD LVCMOS33} [get\_ports {An[6]}]

set\_property -dict {PACKAGE\_PIN U13 IOSTANDARD LVCMOS33} [get\_ports {An[7]}]

##Buttons

#set\_property -dict {PACKAGE\_PIN C12 IOSTANDARD LVCMOS33} [get\_ports Rst]

#set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { BTNC }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

set\_property -dict {PACKAGE\_PIN M18 IOSTANDARD LVCMOS33} [get\_ports {Start}]

set\_property -dict {PACKAGE\_PIN P17 IOSTANDARD LVCMOS33} [get\_ports {Start2}]

set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { Rez\_term }]; #IO\_L10N\_T1\_D15\_14 Sch=btnr

set\_property -dict {PACKAGE\_PIN P18 IOSTANDARD LVCMOS33} [get\_ports {Rst}]

##Pmod Headers

##Pmod Header JA

#set\_property -dict { PACKAGE\_PIN C17 IOSTANDARD LVCMOS33 } [get\_ports { JA[1] }]; #IO\_L20N\_T3\_A19\_15 Sch=ja[1]

#set\_property -dict { PACKAGE\_PIN D18 IOSTANDARD LVCMOS33 } [get\_ports { JA[2] }]; #IO\_L21N\_T3\_DQS\_A18\_15 Sch=ja[2]

#set\_property -dict { PACKAGE\_PIN E18 IOSTANDARD LVCMOS33 } [get\_ports { JA[3] }]; #IO\_L21P\_T3\_DQS\_15 Sch=ja[3]

#set\_property -dict { PACKAGE\_PIN G17 IOSTANDARD LVCMOS33 } [get\_ports { JA[4] }]; #IO\_L18N\_T2\_A23\_15 Sch=ja[4]

#set\_property -dict { PACKAGE\_PIN D17 IOSTANDARD LVCMOS33 } [get\_ports { JA[7] }]; #IO\_L16N\_T2\_A27\_15 Sch=ja[7]

#set\_property -dict { PACKAGE\_PIN E17 IOSTANDARD LVCMOS33 } [get\_ports { JA[8] }]; #IO\_L16P\_T2\_A28\_15 Sch=ja[8]

#set\_property -dict { PACKAGE\_PIN F18 IOSTANDARD LVCMOS33 } [get\_ports { JA[9] }]; #IO\_L22N\_T3\_A16\_15 Sch=ja[9]

#set\_property -dict { PACKAGE\_PIN G18 IOSTANDARD LVCMOS33 } [get\_ports { JA[10] }]; #IO\_L22P\_T3\_A17\_15 Sch=ja[10]

##Pmod Header JB

#set\_property -dict { PACKAGE\_PIN D14 IOSTANDARD LVCMOS33 } [get\_ports { JB[1] }]; #IO\_L1P\_T0\_AD0P\_15 Sch=jb[1]

#set\_property -dict { PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 } [get\_ports { JB[2] }]; #IO\_L14N\_T2\_SRCC\_15 Sch=jb[2]

#set\_property -dict { PACKAGE\_PIN G16 IOSTANDARD LVCMOS33 } [get\_ports { JB[3] }]; #IO\_L13N\_T2\_MRCC\_15 Sch=jb[3]

#set\_property -dict { PACKAGE\_PIN H14 IOSTANDARD LVCMOS33 } [get\_ports { JB[4] }]; #IO\_L15P\_T2\_DQS\_15 Sch=jb[4]

#set\_property -dict { PACKAGE\_PIN E16 IOSTANDARD LVCMOS33 } [get\_ports { JB[7] }]; #IO\_L11N\_T1\_SRCC\_15 Sch=jb[7]

#set\_property -dict { PACKAGE\_PIN F13 IOSTANDARD LVCMOS33 } [get\_ports { JB[8] }]; #IO\_L5P\_T0\_AD9P\_15 Sch=jb[8]

#set\_property -dict { PACKAGE\_PIN G13 IOSTANDARD LVCMOS33 } [get\_ports { JB[9] }]; #IO\_0\_15 Sch=jb[9]

#set\_property -dict { PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 } [get\_ports { JB[10] }]; #IO\_L13P\_T2\_MRCC\_15 Sch=jb[10]

##Pmod Header JC

#set\_property -dict { PACKAGE\_PIN K1 IOSTANDARD LVCMOS33 } [get\_ports { JC[1] }]; #IO\_L23N\_T3\_35 Sch=jc[1]

#set\_property -dict { PACKAGE\_PIN F6 IOSTANDARD LVCMOS33 } [get\_ports { JC[2] }]; #IO\_L19N\_T3\_VREF\_35 Sch=jc[2]

#set\_property -dict { PACKAGE\_PIN J2 IOSTANDARD LVCMOS33 } [get\_ports { JC[3] }]; #IO\_L22N\_T3\_35 Sch=jc[3]

#set\_property -dict { PACKAGE\_PIN G6 IOSTANDARD LVCMOS33 } [get\_ports { JC[4] }]; #IO\_L19P\_T3\_35 Sch=jc[4]

#set\_property -dict { PACKAGE\_PIN E7 IOSTANDARD LVCMOS33 } [get\_ports { JC[7] }]; #IO\_L6P\_T0\_35 Sch=jc[7]

#set\_property -dict { PACKAGE\_PIN J3 IOSTANDARD LVCMOS33 } [get\_ports { JC[8] }]; #IO\_L22P\_T3\_35 Sch=jc[8]

#set\_property -dict { PACKAGE\_PIN J4 IOSTANDARD LVCMOS33 } [get\_ports { JC[9] }]; #IO\_L21P\_T3\_DQS\_35 Sch=jc[9]

#set\_property -dict { PACKAGE\_PIN E6 IOSTANDARD LVCMOS33 } [get\_ports { JC[10] }]; #IO\_L5P\_T0\_AD13P\_35 Sch=jc[10]

##Pmod Header JD

#set\_property -dict { PACKAGE\_PIN H4 IOSTANDARD LVCMOS33 } [get\_ports { JD[1] }]; #IO\_L21N\_T3\_DQS\_35 Sch=jd[1]

#set\_property -dict { PACKAGE\_PIN H1 IOSTANDARD LVCMOS33 } [get\_ports { JD[2] }]; #IO\_L17P\_T2\_35 Sch=jd[2]

#set\_property -dict { PACKAGE\_PIN G1 IOSTANDARD LVCMOS33 } [get\_ports { JD[3] }]; #IO\_L17N\_T2\_35 Sch=jd[3]

#set\_property -dict { PACKAGE\_PIN G3 IOSTANDARD LVCMOS33 } [get\_ports { JD[4] }]; #IO\_L20N\_T3\_35 Sch=jd[4]

#set\_property -dict { PACKAGE\_PIN H2 IOSTANDARD LVCMOS33 } [get\_ports { JD[7] }]; #IO\_L15P\_T2\_DQS\_35 Sch=jd[7]

#set\_property -dict { PACKAGE\_PIN G4 IOSTANDARD LVCMOS33 } [get\_ports { JD[8] }]; #IO\_L20P\_T3\_35 Sch=jd[8]

#set\_property -dict { PACKAGE\_PIN G2 IOSTANDARD LVCMOS33 } [get\_ports { JD[9] }]; #IO\_L15N\_T2\_DQS\_35 Sch=jd[9]

#set\_property -dict { PACKAGE\_PIN F3 IOSTANDARD LVCMOS33 } [get\_ports { JD[10] }]; #IO\_L13N\_T2\_MRCC\_35 Sch=jd[10]

##Pmod Header JXADC

#set\_property -dict { PACKAGE\_PIN A14 IOSTANDARD LVDS } [get\_ports { XA\_N[1] }]; #IO\_L9N\_T1\_DQS\_AD3N\_15 Sch=xa\_n[1]

#set\_property -dict { PACKAGE\_PIN A13 IOSTANDARD LVDS } [get\_ports { XA\_P[1] }]; #IO\_L9P\_T1\_DQS\_AD3P\_15 Sch=xa\_p[1]

#set\_property -dict { PACKAGE\_PIN A16 IOSTANDARD LVDS } [get\_ports { XA\_N[2] }]; #IO\_L8N\_T1\_AD10N\_15 Sch=xa\_n[2]

#set\_property -dict { PACKAGE\_PIN A15 IOSTANDARD LVDS } [get\_ports { XA\_P[2] }]; #IO\_L8P\_T1\_AD10P\_15 Sch=xa\_p[2]

#set\_property -dict { PACKAGE\_PIN B17 IOSTANDARD LVDS } [get\_ports { XA\_N[3] }]; #IO\_L7N\_T1\_AD2N\_15 Sch=xa\_n[3]

#set\_property -dict { PACKAGE\_PIN B16 IOSTANDARD LVDS } [get\_ports { XA\_P[3] }]; #IO\_L7P\_T1\_AD2P\_15 Sch=xa\_p[3]

#set\_property -dict { PACKAGE\_PIN A18 IOSTANDARD LVDS } [get\_ports { XA\_N[4] }]; #IO\_L10N\_T1\_AD11N\_15 Sch=xa\_n[4]

#set\_property -dict { PACKAGE\_PIN B18 IOSTANDARD LVDS } [get\_ports { XA\_P[4] }]; #IO\_L10P\_T1\_AD11P\_15 Sch=xa\_p[4]

##VGA Connector

#set\_property -dict { PACKAGE\_PIN A3 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_R[0] }]; #IO\_L8N\_T1\_AD14N\_35 Sch=vga\_r[0]

#set\_property -dict { PACKAGE\_PIN B4 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_R[1] }]; #IO\_L7N\_T1\_AD6N\_35 Sch=vga\_r[1]

#set\_property -dict { PACKAGE\_PIN C5 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_R[2] }]; #IO\_L1N\_T0\_AD4N\_35 Sch=vga\_r[2]

#set\_property -dict { PACKAGE\_PIN A4 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_R[3] }]; #IO\_L8P\_T1\_AD14P\_35 Sch=vga\_r[3]

#set\_property -dict { PACKAGE\_PIN C6 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_G[0] }]; #IO\_L1P\_T0\_AD4P\_35 Sch=vga\_g[0]

#set\_property -dict { PACKAGE\_PIN A5 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_G[1] }]; #IO\_L3N\_T0\_DQS\_AD5N\_35 Sch=vga\_g[1]

#set\_property -dict { PACKAGE\_PIN B6 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_G[2] }]; #IO\_L2N\_T0\_AD12N\_35 Sch=vga\_g[2]

#set\_property -dict { PACKAGE\_PIN A6 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_G[3] }]; #IO\_L3P\_T0\_DQS\_AD5P\_35 Sch=vga\_g[3]

#set\_property -dict { PACKAGE\_PIN B7 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_B[0] }]; #IO\_L2P\_T0\_AD12P\_35 Sch=vga\_b[0]

#set\_property -dict { PACKAGE\_PIN C7 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_B[1] }]; #IO\_L4N\_T0\_35 Sch=vga\_b[1]

#set\_property -dict { PACKAGE\_PIN D7 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_B[2] }]; #IO\_L6N\_T0\_VREF\_35 Sch=vga\_b[2]

#set\_property -dict { PACKAGE\_PIN D8 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_B[3] }]; #IO\_L4P\_T0\_35 Sch=vga\_b[3]

#set\_property -dict { PACKAGE\_PIN B11 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_HS }]; #IO\_L4P\_T0\_15 Sch=vga\_hs

#set\_property -dict { PACKAGE\_PIN B12 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_VS }]; #IO\_L3N\_T0\_DQS\_AD1N\_15 Sch=vga\_vs

##Micro SD Connector

#set\_property -dict { PACKAGE\_PIN E2 IOSTANDARD LVCMOS33 } [get\_ports { SD\_RESET }]; #IO\_L14P\_T2\_SRCC\_35 Sch=sd\_reset

#set\_property -dict { PACKAGE\_PIN A1 IOSTANDARD LVCMOS33 } [get\_ports { SD\_CD }]; #IO\_L9N\_T1\_DQS\_AD7N\_35 Sch=sd\_cd

#set\_property -dict { PACKAGE\_PIN B1 IOSTANDARD LVCMOS33 } [get\_ports { SD\_SCK }]; #IO\_L9P\_T1\_DQS\_AD7P\_35 Sch=sd\_sck

#set\_property -dict { PACKAGE\_PIN C1 IOSTANDARD LVCMOS33 } [get\_ports { SD\_CMD }]; #IO\_L16N\_T2\_35 Sch=sd\_cmd

#set\_property -dict { PACKAGE\_PIN C2 IOSTANDARD LVCMOS33 } [get\_ports { SD\_DAT[0] }]; #IO\_L16P\_T2\_35 Sch=sd\_dat[0]

#set\_property -dict { PACKAGE\_PIN E1 IOSTANDARD LVCMOS33 } [get\_ports { SD\_DAT[1] }]; #IO\_L18N\_T2\_35 Sch=sd\_dat[1]

#set\_property -dict { PACKAGE\_PIN F1 IOSTANDARD LVCMOS33 } [get\_ports { SD\_DAT[2] }]; #IO\_L18P\_T2\_35 Sch=sd\_dat[2]

#set\_property -dict { PACKAGE\_PIN D2 IOSTANDARD LVCMOS33 } [get\_ports { SD\_DAT[3] }]; #IO\_L14N\_T2\_SRCC\_35 Sch=sd\_dat[3]

##Accelerometer

#set\_property -dict { PACKAGE\_PIN E15 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_MISO }]; #IO\_L11P\_T1\_SRCC\_15 Sch=acl\_miso

#set\_property -dict { PACKAGE\_PIN F14 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_MOSI }]; #IO\_L5N\_T0\_AD9N\_15 Sch=acl\_mosi

#set\_property -dict { PACKAGE\_PIN F15 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_SCLK }]; #IO\_L14P\_T2\_SRCC\_15 Sch=acl\_sclk

#set\_property -dict { PACKAGE\_PIN D15 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_CSN }]; #IO\_L12P\_T1\_MRCC\_15 Sch=acl\_csn

#set\_property -dict { PACKAGE\_PIN B13 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_INT[1] }]; #IO\_L2P\_T0\_AD8P\_15 Sch=acl\_int[1]

#set\_property -dict { PACKAGE\_PIN C16 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_INT[2] }]; #IO\_L20P\_T3\_A20\_15 Sch=acl\_int[2]

##Temperature Sensor

#set\_property -dict { PACKAGE\_PIN C14 IOSTANDARD LVCMOS33 } [get\_ports { TMP\_SCL }]; #IO\_L1N\_T0\_AD0N\_15 Sch=tmp\_scl

#set\_property -dict { PACKAGE\_PIN C15 IOSTANDARD LVCMOS33 } [get\_ports { TMP\_SDA }]; #IO\_L12N\_T1\_MRCC\_15 Sch=tmp\_sda

#set\_property -dict { PACKAGE\_PIN D13 IOSTANDARD LVCMOS33 } [get\_ports { TMP\_INT }]; #IO\_L6N\_T0\_VREF\_15 Sch=tmp\_int

#set\_property -dict { PACKAGE\_PIN B14 IOSTANDARD LVCMOS33 } [get\_ports { TMP\_CT }]; #IO\_L2N\_T0\_AD8N\_15 Sch=tmp\_ct

##Omnidirectional Microphone

#set\_property -dict { PACKAGE\_PIN J5 IOSTANDARD LVCMOS33 } [get\_ports { M\_CLK }]; #IO\_25\_35 Sch=m\_clk

#set\_property -dict { PACKAGE\_PIN H5 IOSTANDARD LVCMOS33 } [get\_ports { M\_DATA }]; #IO\_L24N\_T3\_35 Sch=m\_data

#set\_property -dict { PACKAGE\_PIN F5 IOSTANDARD LVCMOS33 } [get\_ports { M\_LRSEL }]; #IO\_0\_35 Sch=m\_lrsel

##PWM Audio Amplifier

#set\_property -dict { PACKAGE\_PIN A11 IOSTANDARD LVCMOS33 } [get\_ports { AUD\_PWM }]; #IO\_L4N\_T0\_15 Sch=aud\_pwm

#set\_property -dict { PACKAGE\_PIN D12 IOSTANDARD LVCMOS33 } [get\_ports { AUD\_SD }]; #IO\_L6P\_T0\_15 Sch=aud\_sd

##USB-RS232 Interface

#set\_property -dict { PACKAGE\_PIN C4 IOSTANDARD LVCMOS33 } [get\_ports { UART\_TXD\_IN }]; #IO\_L7P\_T1\_AD6P\_35 Sch=uart\_txd\_in

#set\_property -dict { PACKAGE\_PIN D4 IOSTANDARD LVCMOS33 } [get\_ports { UART\_RXD\_OUT }]; #IO\_L11N\_T1\_SRCC\_35 Sch=uart\_rxd\_out

#set\_property -dict { PACKAGE\_PIN D3 IOSTANDARD LVCMOS33 } [get\_ports { UART\_CTS }]; #IO\_L12N\_T1\_MRCC\_35 Sch=uart\_cts

#set\_property -dict { PACKAGE\_PIN E5 IOSTANDARD LVCMOS33 } [get\_ports { UART\_RTS }]; #IO\_L5N\_T0\_AD13N\_35 Sch=uart\_rts

##USB HID (PS/2)

#set\_property -dict { PACKAGE\_PIN F4 IOSTANDARD LVCMOS33 } [get\_ports { PS2\_CLK }]; #IO\_L13P\_T2\_MRCC\_35 Sch=ps2\_clk

#set\_property -dict { PACKAGE\_PIN B2 IOSTANDARD LVCMOS33 } [get\_ports { PS2\_DATA }]; #IO\_L10N\_T1\_AD15N\_35 Sch=ps2\_data

##SMSC Ethernet PHY

#set\_property -dict { PACKAGE\_PIN C9 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_MDC }]; #IO\_L11P\_T1\_SRCC\_16 Sch=eth\_mdc

#set\_property -dict { PACKAGE\_PIN A9 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_MDIO }]; #IO\_L14N\_T2\_SRCC\_16 Sch=eth\_mdio

#set\_property -dict { PACKAGE\_PIN B3 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_RSTN }]; #IO\_L10P\_T1\_AD15P\_35 Sch=eth\_rstn

#set\_property -dict { PACKAGE\_PIN D9 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_CRSDV }]; #IO\_L6N\_T0\_VREF\_16 Sch=eth\_crsdv

#set\_property -dict { PACKAGE\_PIN C10 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_RXERR }]; #IO\_L13N\_T2\_MRCC\_16 Sch=eth\_rxerr

#set\_property -dict { PACKAGE\_PIN C11 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_RXD[0] }]; #IO\_L13P\_T2\_MRCC\_16 Sch=eth\_rxd[0]

#set\_property -dict { PACKAGE\_PIN D10 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_RXD[1] }]; #IO\_L19N\_T3\_VREF\_16 Sch=eth\_rxd[1]

#set\_property -dict { PACKAGE\_PIN B9 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_TXEN }]; #IO\_L11N\_T1\_SRCC\_16 Sch=eth\_txen

#set\_property -dict { PACKAGE\_PIN A10 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_TXD[0] }]; #IO\_L14P\_T2\_SRCC\_16 Sch=eth\_txd[0]

#set\_property -dict { PACKAGE\_PIN A8 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_TXD[1] }]; #IO\_L12N\_T1\_MRCC\_16 Sch=eth\_txd[1]

#set\_property -dict { PACKAGE\_PIN D5 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_REFCLK }]; #IO\_L11P\_T1\_SRCC\_35 Sch=eth\_refclk

#set\_property -dict { PACKAGE\_PIN B8 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_INTN }]; #IO\_L12P\_T1\_MRCC\_16 Sch=eth\_intn

##Quad SPI Flash

#set\_property -dict { PACKAGE\_PIN K17 IOSTANDARD LVCMOS33 } [get\_ports { QSPI\_DQ[0] }]; #IO\_L1P\_T0\_D00\_MOSI\_14 Sch=qspi\_dq[0]

#set\_property -dict { PACKAGE\_PIN K18 IOSTANDARD LVCMOS33 } [get\_ports { QSPI\_DQ[1] }]; #IO\_L1N\_T0\_D01\_DIN\_14 Sch=qspi\_dq[1]

#set\_property -dict { PACKAGE\_PIN L14 IOSTANDARD LVCMOS33 } [get\_ports { QSPI\_DQ[2] }]; #IO\_L2P\_T0\_D02\_14 Sch=qspi\_dq[2]

#set\_property -dict { PACKAGE\_PIN M14 IOSTANDARD LVCMOS33 } [get\_ports { QSPI\_DQ[3] }]; #IO\_L2N\_T0\_D03\_14 Sch=qspi\_dq[3]

#set\_property -dict { PACKAGE\_PIN L13 IOSTANDARD LVCMOS33 } [get\_ports { QSPI\_CSN }]; #IO\_L6P\_T0\_FCS\_B\_14 Sch=qspi\_csn