Ștefan Gheorghe

Universitatea “Transilvania” brașov

Multiplicator de numere complexe

Cuprins

[1 Prezentare generală 2](#_Toc34822555)

[2 Implementare cu un singur modul de multiplicare 2](#_Toc34822556)

[2.1 Arhitectură 2](#_Toc34822557)

[2.2 Logica de control 3](#_Toc34822558)

[2.3 Scenarii de test și forme de undă 4](#_Toc34822559)

[3 Implementare cu două module de multiplicare 10](#_Toc34822560)

[3.1 Arhitectură 10](#_Toc34822561)

[3.2 Logica de control 11](#_Toc34822562)

[3.3 Forme de undă obținute 12](#_Toc34822563)

[4 Implementare cu patru module de multiplicare 12](#_Toc34822564)

[4.1 Arhitectură 12](#_Toc34822565)

[4.2 Logica de control 13](#_Toc34822566)

[4.3 Forme de undă obținute 14](#_Toc34822567)

# Prezentare generală

Circuitul implementat realizează înmulțirea a două numere complexe reprezentate sub forma algebrică.

Părțile reale și imaginare ale operanzilor sunt numere întregi reprezentate pe 8 biți, în complement față de 2.

În acest document sunt prezentate trei variante de implementare, fiecare cu un grad diferit de paralelism.

# Implementare cu un singur modul de multiplicare

## Arhitectură

În Figura 1 este prezentată arhitectura modulului. Tabelul 1 prezintă interfețele modulului, semnalele interne și semnificația acestora.



Figura 1 : Arhitectura modulului implementat

Tabel 1 : Prezentarea interfețelor și semnalelor interne.

|  |  |  |
| --- | --- | --- |
| Denumire | Tip | Explicatie |
| op\_val | I | Semnalizează faptul că operanzii sunt gata de preluat de către modul. |
| sw\_rst | I | Reset sincron al modulului. |
| op\_ready | O | Semnalizează faptul că modulul este pregătit pentru a primi noi operanzi. |
| op\_1\_re | I | Partea reală a primului operand. |
| op\_1\_im | I | Partea imaginară a primului operand. |
| op\_2\_re | I | Partea reală a celui de-al doilea operand. |
| op\_2\_im | I | Partea imaginară a celui de-al doilea operand. |
| res\_ready | I | Modulul Master este gata de a primi rezultatul. |
| res\_val | O | Rezultatul multiplicarii este pregătit pentru a fi transmis. |
| result\_real | O | Partea reală a rezultatului obținut. |
| result\_im | O | Partea imaginară a rezultatului obținut. |
| op\_1\_sel | Intern | Semnalul de selecție pentru primul operand al modulului de multiplicare. |
| op\_2\_sel | Intern | Semnalul de selecție pentru al doilea operand al modulului de multiplicare. |
| res\_reg\_sel | Intern | Semnalul de selecție a registrului în care va fi stocat rezultatul multiplicării curente. |
| compute\_enable | Intern | Semnal de enable pentru realizarea adunării și scăderii finale. |

## Logica de control

În Figura 2 este prezentat graful de tranziții al modulului implementat. Tabelul 2 conține o scurtă explicație a fiecărei stări în parte.

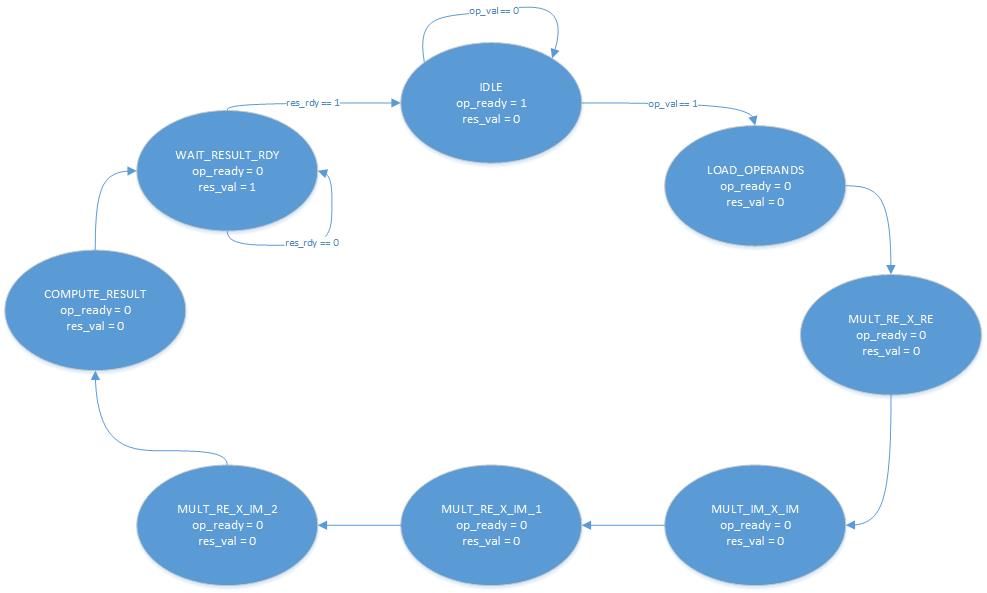


Figura 2: Graful de tranziție a stărilor

Tabel 2 : Explicarea stărilor modulului.

|  |  |
| --- | --- |
| Denumire | Explicație |
| IDLE | Modulul este în așteptare de noi operanzi, op\_ready este 1. |
| LOAD\_OPERANDS | Operanzii sunt încărcați în registrele interne. |
| MULT\_RE\_X\_RE | Se înmulțesc părțile reale ale fiecărui operand și se stochează în registrul corespunzător. |
| MULT\_IM\_X\_IM | Se înmulțesc părțile imaginare ale fiecărui operand și se stochează în registrul corespunzător. |
| MULT\_RE\_X\_IM\_1 | Se înmulțește partea reală a primului operand cu partea imaginară a celui de-al doilea operand și se stochează în registrul corespunzător. |
| MULT\_RE\_X\_IM\_2 | Se înmulțește partea imaginară a primului operand cu partea reală a celui de-al doilea operand și se stochează în registrul corespunzător. |
| COMPUTE\_RESULT | Se calculează adunarea și scăderea finale. |
| WAIT\_RESULT\_RDY | Se așteaptă semnalul de res\_ready, res\_val este activ. |

## Scenarii de test și forme de undă

Pentru testarea modulului au fost implementate mai multe scenarii de test. Acestea sunt prezentate mai jos, în codul pentru modulul complex\_nr\_mult\_tb. Acest modul, împreună cu monitor\_complex\_multiplier, vor fi folosite pentru simularea și verificarea funcționalității tuturor modulelor implementate în cadrul acestui proiect.

Fiecare scenariu de test are un task asociat. Din modulul test\_environment se poate alege ce scenariu va fi rulat. Proiectantul are posibilitatea de a selecta operanzii, de a trimite valori aleatoare, de a trimite valori extreme sau de a efectua mai multe calcule consecutiv.

    // Module:  complex\_nr\_mult\_tb

    // Author:  Gheorghe Stefan

    // Date:    06.03.2020

    module complex\_nr\_mult\_tb#(

        parameter DATA\_WIDTH    = 8,

        parameter TEST\_SCENARIO = 0

    )(

        input   clk         , // clock signal

        input   rstn        , // asynchronous reset active 0

        input   op\_ready    , // module is ready to receive new operands

        input   res\_val     , // result valid signal

        output reg                       sw\_rst              , // software reset active 1

        output reg                       op\_val              , // data valid signal

        output reg                       res\_ready           , // the consumer is ready to receive the result

        output reg [DATA\_WIDTH-1 : 0]    op\_1\_re             , // input for the real part of the first operand

        output reg [DATA\_WIDTH-1 : 0]    op\_1\_im             , // input for the imaginary part of the first operand

        output reg [DATA\_WIDTH-1 : 0]    op\_2\_re             , // input for the real part of the second operand

        output reg [DATA\_WIDTH-1 : 0]    op\_2\_im               // input for the imaginary part of the second operand

    );

        // Internal signals and registers

        reg [DATA\_WIDTH-1 : 0] op\_1\_re\_reg;

        reg [DATA\_WIDTH-1 : 0] op\_1\_im\_reg;

        reg [DATA\_WIDTH-1 : 0] op\_2\_re\_reg;

        reg [DATA\_WIDTH-1 : 0] op\_2\_im\_reg;

        // Task for driving operands on bus

        task write\_operands;

            input [DATA\_WIDTH-1 : 0] op\_1\_re\_value;

            input [DATA\_WIDTH-1 : 0] op\_1\_im\_value;

            input [DATA\_WIDTH-1 : 0] op\_2\_re\_value;

            input [DATA\_WIDTH-1 : 0] op\_2\_im\_value;

            begin

                op\_1\_re <= op\_1\_re\_value;

                op\_1\_im <= op\_1\_im\_value;

                op\_2\_re <= op\_2\_re\_value;

                op\_2\_im <= op\_2\_im\_value;

                $display("%M %t - OPERANDS VALUES ON THE BUS", $time);

            end

        endtask

        task module\_wait;

            input [9:0]  wait\_cycles;   // how many cycles to wait

            integer i;

            begin

                for (i=0; i<wait\_cycles; i=i+1) begin

                @(posedge clk);

                end

                $display("%M %t - WAIT  -> %d clock cycles", $time, wait\_cycles);

            end

        endtask

        task write\_valid;

            begin

                op\_val <= 'b1;

                $display("%M %t - OPERAND VALID SIGNAL ASSERTED", $time);

                @(posedge clk);

                @(posedge clk);

                op\_val <= 'b0;

                $display("%M %t - OPERAND VALID SIGNAL DEASSERTED", $time);

            end

        endtask

        task write\_result\_ready;

            begin

                res\_ready <= 'b1;

                $display("%M %t - RESULT READY SIGNAL ASSERTED", $time);

                @(posedge clk);

                res\_ready <= 'b0;

                $display("%M %t - RESULT READY SIGNAL DEASSERTED", $time);

            end

        endtask

        task test\_scenario\_selected\_values;

            begin

                $display("%M %t - STARTED TEST SCENARIO WITH SELECTED VALUES", $time);

                write\_operands(2,3,4,2);

                module\_wait(2);

                write\_valid;

                module\_wait(20);

                write\_result\_ready;

                $stop;

            end

        endtask

        task test\_scenario\_random\_values;

            begin

                op\_1\_re\_reg = $random;

                op\_1\_im\_reg = $random;

                op\_2\_re\_reg = $random;

                op\_2\_im\_reg = $random;

                $display("%M %t - STARTED TEST SCENARIO WITH RANDOM VALUES", $time);

                write\_operands(op\_1\_re\_reg,op\_1\_im\_reg,op\_2\_re\_reg,op\_2\_im\_reg);

                module\_wait(2);

                write\_valid;

                module\_wait(20);

                write\_result\_ready;

                $stop;

            end

        endtask

        task test\_scenario\_corner\_case;

            begin

                op\_1\_re\_reg = {DATA\_WIDTH{1'b1}};

                op\_1\_im\_reg = {DATA\_WIDTH{1'b1}};

                op\_2\_re\_reg = {DATA\_WIDTH{1'b1}};

                op\_2\_im\_reg = {DATA\_WIDTH{1'b1}};

                $display("%M %t - STARTED TEST SCENARIO WITH CORNER CASE VALUES", $time);

                write\_operands(op\_1\_re\_reg,op\_1\_im\_reg,op\_2\_re\_reg,op\_2\_im\_reg);

                module\_wait(2);

                write\_valid;

                module\_wait(20);

                write\_result\_ready;

                $stop;

            end

        endtask

        task test\_scenario\_multiple\_transactions;

            input [9:0] transaction\_number;

            integer i;

            begin

                $display("%M %t - STARTED FIRST TEST SCENARIO WITH MULTIPLE TRANSACTIONS VALUES", $time);

                for (i=0; i<transaction\_number; i=i+1)

                begin

                    op\_1\_re\_reg = $random;

                    op\_1\_im\_reg = $random;

                    op\_2\_re\_reg = $random;

                    op\_2\_im\_reg = $random;

                    write\_operands(op\_1\_re\_reg,op\_1\_im\_reg,op\_2\_re\_reg,op\_2\_im\_reg);

                    module\_wait(2);

                    write\_valid;

                    module\_wait(20);

                    write\_result\_ready;

                end

                $stop;

            end

        endtask

        initial

        begin

            wait(~rstn);

            case (TEST\_SCENARIO)

                0:  test\_scenario\_selected\_values;

                1:  test\_scenario\_random\_values;

                2:  test\_scenario\_corner\_case;

                3:  test\_scenario\_multiple\_transactions(3);

                default:    test\_scenario\_selected\_values;

            endcase

        end

    endmodule // complex\_nr\_mult\_tb

Modulul monitor\_complex\_multiplier are rolul de a verifica automat dacă operațiile efectuate sunt corecte.

// Module:  monitor\_complex\_multiplier

// Author:  Gheorghe Stefan

// Date:    06.03.2020

module monitor\_complex\_multiplier#(

    parameter DATA\_WIDTH = 8

)(

    input                       clk             ,

    input                       rstn            ,

    input                       sw\_rst          ,

    input                       op\_val          ,

    input                       res\_ready       ,

    input [DATA\_WIDTH-1 : 0]    op\_1\_re         ,

    input [DATA\_WIDTH-1 : 0]    op\_1\_im         ,

    input [DATA\_WIDTH-1 : 0]    op\_2\_re         ,

    input [DATA\_WIDTH-1 : 0]    op\_2\_im         ,

    input                       op\_ready        ,

    input                       res\_val         ,

    input [DATA\_WIDTH\*2-1 : 0]  result\_re       ,

    input [DATA\_WIDTH\*2-1 : 0]  result\_im

);

    //Internal registers for checking the functionality

    reg [DATA\_WIDTH\*2-1 : 0] predicted\_result\_re;

    reg [DATA\_WIDTH\*2-1 : 0] predicted\_result\_im;

    always @(posedge clk or negedge rstn)

    begin

        if(~rstn) predicted\_result\_re <= 'b0;

        else if(sw\_rst) predicted\_result\_re <= 'b0;

        else if(op\_val) predicted\_result\_re <= (op\_1\_re \* op\_2\_re) - (op\_1\_im \* op\_2\_im);

    end

    always @(posedge clk or negedge rstn)

    begin

        if(~rstn) predicted\_result\_im <= 'b0;

        else if(sw\_rst) predicted\_result\_im <= 'b0;

        else if(op\_val) predicted\_result\_im <= (op\_1\_re \* op\_2\_im) + (op\_1\_im \* op\_2\_re);

    end

    always @(posedge clk)

    begin

        if (res\_ready && res\_val) begin

            if(result\_re == predicted\_result\_re)

                $display("%M %t - REAL PART OF THE RESULT IS COMPUTED CORRECTLY", $time);

            else

                $display("%M %t - REAL PART OF THE RESULT WAS NOT COMPUTED CORRECTLY", $time);

        end

    end

    always @(posedge clk)

    begin

        if (res\_ready  && res\_val) begin

            if(result\_im == predicted\_result\_im)

                $display("%M %t - IMAGINARY PART OF THE RESULT IS COMPUTED CORRECTLY", $time);

            else

                $display("%M %t - IMAGINARY PART OF THE RESULT WAS NOT COMPUTED CORRECTLY", $time);

        end

    end

endmodule // monitor\_complex\_multiplier

Formele de undă obținute în urma simulării primului scenariu de test cu valorile preluate din specificațiile proiectului sunt prezentate în Figura 3.

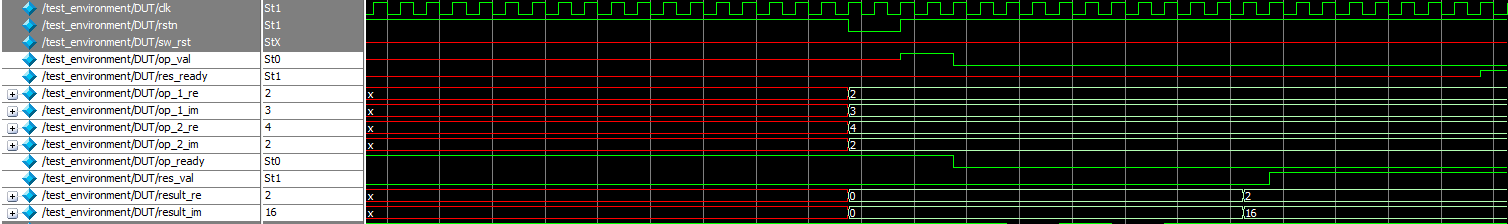


Figura 3 : Forme de undă obținute

# Implementare cu două module de multiplicare

## Arhitectură

În Figura 4 este prezentată arhitectura modulului. Tabelul 3 prezintă semnalele interne ale modulului și semnificația acestora. Interfața acestuia este aceeași cu implementarea cu un singur modul de multiplicare, interfață prezentată în Tabelul 2.



Figura 4 : Arhitectura modulului implementat

Tabel 3 : Prezentarea semnalelor interne.

|  |  |  |
| --- | --- | --- |
| Denumire | Tip | Explicatie |
| mult\_1\_op\_1\_sel | Intern | Semnal de selecție pentru primul operand al multiplicatorului 1. |
| mult\_1\_op\_2\_sel | Intern | Semnal de selecție pentru al doilea operand al multiplicatorului 1. |
| mult\_2\_op\_1\_sel | Intern | Semnal de selecție pentru primul operand al multiplicatorului 2. |
| mult\_2\_op\_2\_sel | Intern | Semnal de selecție pentru al doilea operand al multiplicatorului 2. |
| mult\_1\_result \_sel | Intern | Semnal de selecție pentru registrul de stocare al rezultatului de la ieșirea multiplicatorului 1. |
| mult\_2\_result \_sel | Intern | Semnal de selecție pentru registrul de stocare al rezultatului de la ieșirea multiplicatorului 2. |
| compute\_enable | Intern | Semnal de enable pentru realizarea adunării și scăderii finale. |

## Logica de control

În Figura 5 este prezentat graful de tranziții al modulului implementat. Tabelul 4 conține o scurtă explicație a fiecărei stări în parte.



Figura 5: Graful de tranziție a stărilor

Tabel 4 : Explicarea stărilor modulului.

|  |  |
| --- | --- |
| Denumire | Explicație |
| IDLE | Modulul este în așteptare de noi operanzi, op\_ready este 1. |
| LOAD\_OPERANDS | Operanzii sunt încărcați în registrele interne. |
| FIRST\_STAGE\_MULTIPLY | Se înmulțesc părțile reale ale fiecărui operand și părțile imaginare între ele. |
| SCND\_STAGE\_MULTIPLY | Se calculează valorile pentru adunare. |
| COMPUTE\_RESULT | Se calculează adunarea și scăderea finale. |
| WAIT\_RESULT\_RDY | Se așteaptă semnalul de res\_ready, res\_val este activ. |

## Forme de undă obținute

# Implementare cu patru module de multiplicare

## Arhitectură

În Figura 7 este prezentată arhitectura modulului. Tabelul 5 prezintă semnalele interne ale modulului și semnificația acestora. Interfața acestuia este aceeași cu implementarea cu un singur modul de multiplicare, interfață prezentată în Tabelul 2.



Figura 7 : Arhitectura modulului implementat

Tabel 5 : Prezentarea semnalelor interne.

|  |  |  |
| --- | --- | --- |
| Denumire | Tip | Explicatie |
| compute\_enable | Intern | Semnal de enable pentru realizarea adunării și scăderii finale. |

## Logica de control

În Figura 8 este prezentat graful de tranziții al modulului implementat. Tabelul 5 conține o scurtă explicație a fiecărei stări în parte.



Figura 5: Graful de tranziție a stărilor

Tabel 6 : Explicarea stărilor modulului.

|  |  |
| --- | --- |
| Denumire | Explicație |
| IDLE | Modulul este în așteptare de noi operanzi, op\_ready este 1. |
| COMPUTE\_RESULT | Se calculează adunarea și scăderea. |
| WAIT\_RESULT\_RDY | Se așteaptă semnalul de res\_ready, res\_val este activ. |

## Forme de undă obținute