MULTIPLICATOR DE NUMERE COMPLEXE

Ștefan Gheorghe

UNIVERSITATEA “TRANSILVANIA” BRAȘOV

Cuprins

1 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.

2 Implementare cu un singur modul de multiplicare

2.1 Arhitectură

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

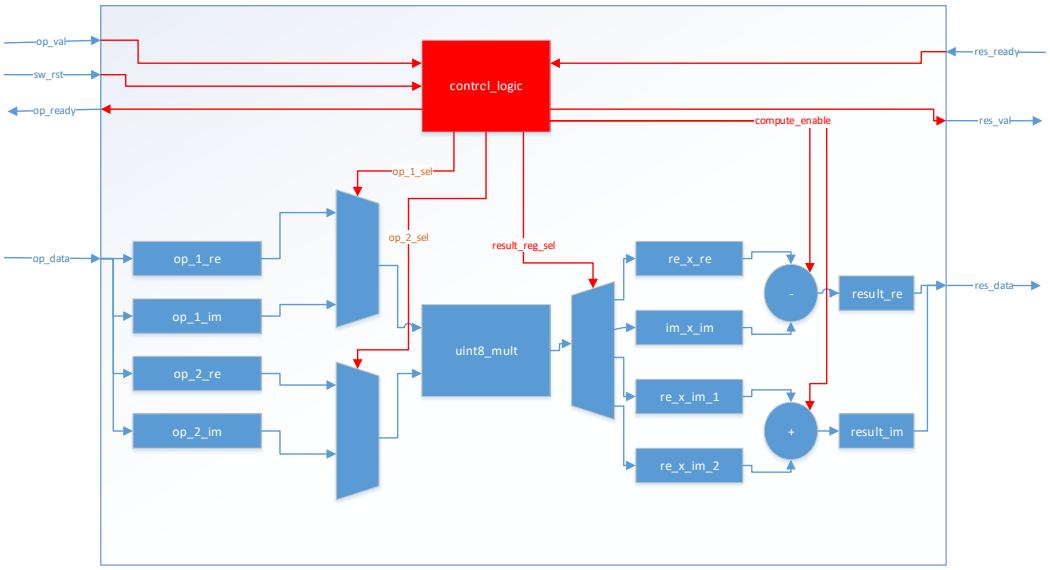

Figura 1 : Arhitectura modulului implementat

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_data	I	Datele de intrare ale modulului.
res_ready	I	Modulul Master este gata de a primi rezultatul.
res_val	O	Rezultatul multiplicarii este pregătit pentru a fi transmis.
res_data	O	Datele de ieșire ale modului.
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.

2.2 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.

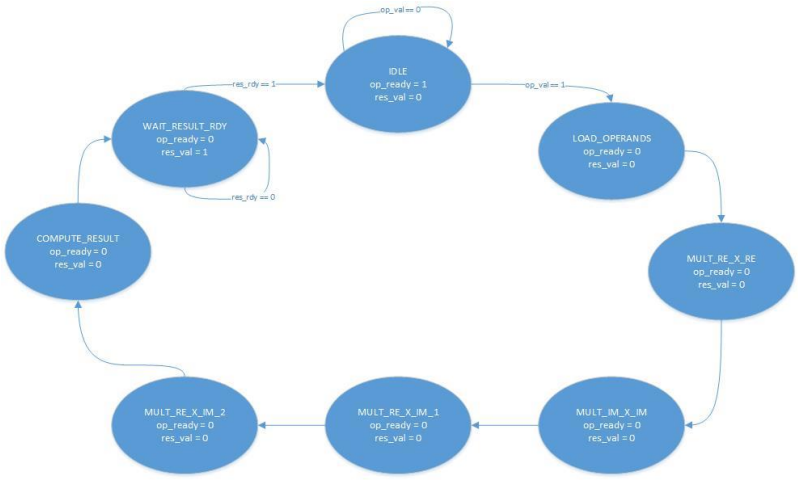

Figura 2: Graful de tranziție a stărilor

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.

2.3 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 activ
e 1
        output reg                      op_val          , // data valid signal
        output reg                      res_ready       , // the consumer is read
y 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", $tim
e);
                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 TRANSACTIO
NS 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_i
m);
    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_r
e);
    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", $tim
e);
            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 CORREC
TLY", $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

3 Implementare cu două module de multiplicare

3.1 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.

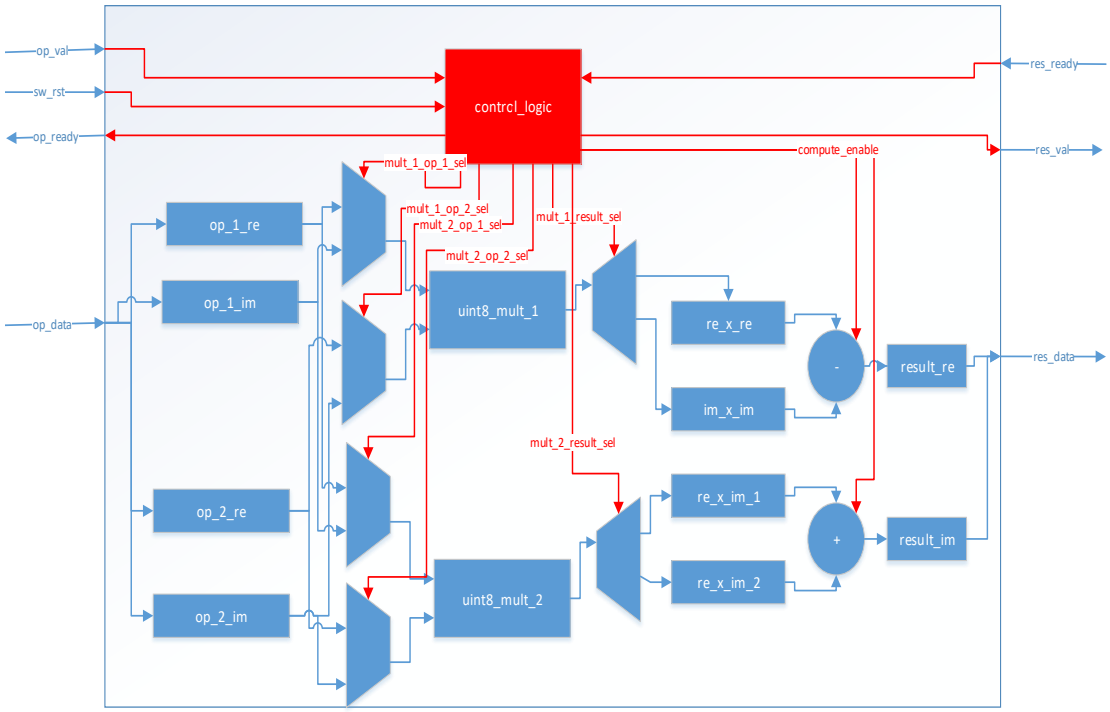

Figura 4 : Arhitectura modulului implementat

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.

3.2 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.

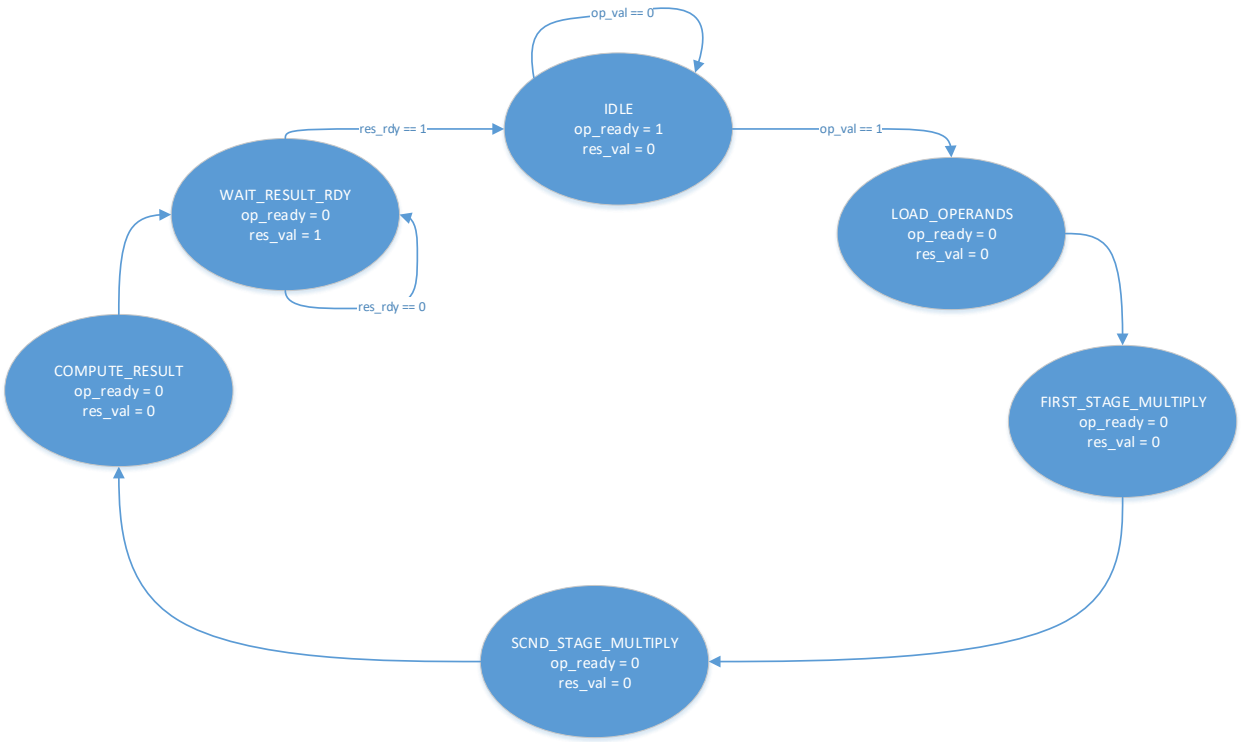

Figura 5: Graful de tranziție a stărilor

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.

3.3 Forme de undă obținute

4 Implementare cu patru module de multiplicare

4.1 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.

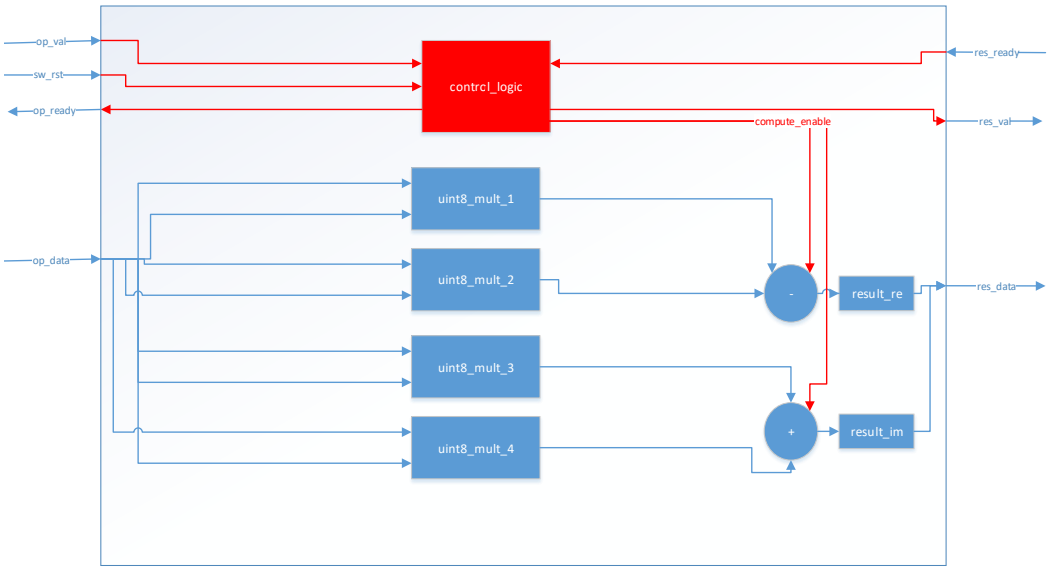

Figura 7 : Arhitectura modulului implementat

Tabel 5 : Prezentarea semnalelor interne.

Denumire	Tip	Explicatie
compute_enable	Intern	Semnal de enable pentru realizarea adunării și scăderii finale.

4.2 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.

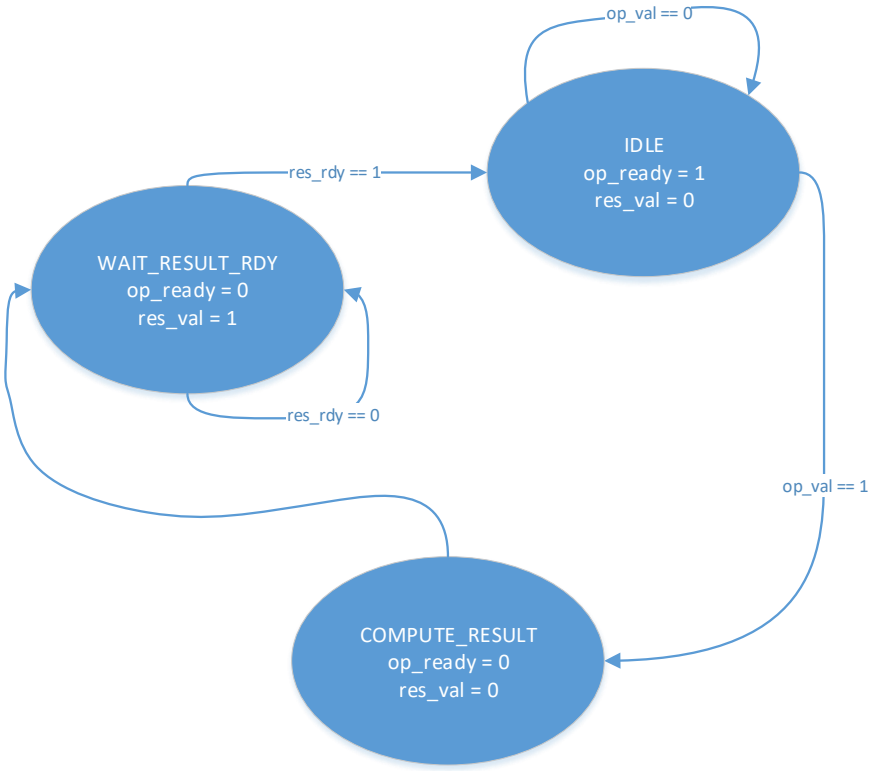

Figura 5: Graful de tranziție a stărilor

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.

4.3 Forme de undă obținute