Dokumentacja z projektu

UKŁAD DO PRZEPROWADZANIA TRANSFORMATY FFT

Projekt wykonały: Kinga Gaweł, Magda Kalawska

Przedmiot: Systemu dedykowane w układach programowalnych

Prowadzący: mgr Sebastian Koryciak

1. Cel projektu

Celem projektu było stworzenie układu, który odpowiednio przetworzy dane odczytane z detektora, następnie obliczy z próbek FFT, a końcowy wynik zostanie przesłany poprzez łącze Ethernetowe.

2. Schemat projektowy

Podczas realizacji projektu wyszczególniono następujące etapy:

I. Zamiana na moc odczytu z pojedynczego detektora

Próbki odczytane z 4 detektorów (A, B, C, R) są zapisywane na 12 bitach (wartości 0-4095). Pierwszą modyfikacją danych (zanim zostanie obliczone FFT) jest zamiana odczytanych wartości na moc. Zamianę opisują wzory:

A_d = (A-1700)/39 [dBm]

B_d = (B-1700)/39 [dBm]

C_d = (C-1700)/39 [dBm]

R_d = (R-1700)/39 [dBm]

II. Obliczenie wartości zespolonej sygnału na podstawie odczytu

Korzystając z poniższych wzorów otrzymujemy wartości RE i IM:

$$RE = -10^{(A_d - R_d)/10} + 0.5 * 10^{(B_d - R_d)/10} + 0.5 * 10^{(C_d - R_d)/10}$$

$$IM = -0.5 * \sqrt{3} * 10^{(B_d - R_d)/10} + 0.5 * \sqrt{3} * 10^{(C_d - R_d)/10}$$

III. Obliczenie FFT z wektora składającego się z par RE + IM

IV. Przesłanie wyników FFT po łączu Ethernetowym

3. Algorytm behawioralny w SV

W poniższym algorytmie na wejście podajemy próbki odczytane z detektorów natomiast na wyjściu jest wektor, z którego następnie zostanie obliczone FFT.

```
module signal_behavioral(
                                    input logic clk,
                                    input logic reset,
                                    input real s1 [0:1023],
                                    input real s2 [0:1023],
                                    input real s3 [0:1023],
                                    input real s4 [0:1023],
                                    output real signal [0:2*1024-1]
  );
  parameter N = 1024;
  parameter sqrt3 = 1.732;

  real x1 [0:N-1];
  real x2 [0:N-1];
  real x3 [0:N-1];
  real x4 [0:N-1];
  real re [0:N-1];
  real im [0:N-1];
  real dc_re;
  real dc_im;

  always_comb begin
    for(int n = 0; n < N-1; n = n+1) begin
      x1[n] = (s1[n]-1700.0)/39.0;
      x2[n] = (s2[n]-1700.0)/39.0;
      x3[n] = (s3[n]-1700.0)/39.0;
      x4[n] = (s4[n]-1700.0)/39.0;
    end
    for(int n = 0; n < N-1; n = n+1) begin
  re[n] = 10 ** ((x1[n]-x4[n])/10) + 0.5 * 10 ** ((x2[n]-x4[n])/10)
        + 0.5 * 10 ** ((x3[n]- x4[n])/10);
      im[n] = -0.5 * sqrt3 * 10 ** ((x2[n]-x4[n])/10) + 0.5 * sqrt3 * 10 ** ((x3[n]-x4[n])/10);
      dc_re = dc_re + re[n];
      dc_im = dc_im + im[n];
    end
    dc_re = dc_re / N;
    dc_im = dc_im / N;
    for(int n = 0; n < N-1; n = n+1) begin
      signal[2*n] = re[n] - dc_re;
      signal[2*n+1] = im[n] - dc_im;
    end
  end
  endmodule
```

4. Testbench

```
module signal_behavioral_TB();

  parameter N = 1024;
  logic clk, reset;
  real s1 [0:N-1];
  real s2 [0:N-1];
  real s3 [0:N-1];
  real s4 [0:N-1];
  real signal [0:2*N-1];

  signal_behavioral DUT (.clk, .s1, .s2, .s3, .s4, .signal);

  initial begin
    for(int n = 0; n<N-1; n= n+1) begin
      s1[n] = 40 + 2*n%40;
      s2[n] = 20 + 3*n%40;
      s3[n] = 60 + 2*n%40;
      s4[n] = 30 + 3*n%40;
    end
    clk = 1;
    reset = 1;
    #5;
    reset = 0;
  end

  always begin
    clk = ~clk;
    #5;
  end

endmodule
```

5. Koncepcja wersji syntezowalnej

Na podstawie algorytmu behawioralnego należało utworzyć maszynę stanów, która przetwarza odpowiednio próbki z detektorów. Problemem tutaj stanowiła funkcja wykładnicza 10^S , gdzie S jest liczbą rzeczywistą. Wartość odczytana z sensora mieści się w zakresie 0 - 4095. Zamiana tej wartości na moc odbywa się przy pomocy wzoru (x - 1700)/39. Z tego wynika, że z sensora otrzymujemy moc w zakresie od -43.59 dBm do 61.41 dBm.

Zakładając

$$S_1 = (A_d - R_d)/10 = (A-R)/390$$

$$S_2 = (B_d - R_d)/10 = (B-R)/390$$

$$S_3 = (C_d - R_d)/10 = (C-R)/390$$

możemy wywnioskować, że S1, S2, S3 będą w zakresie: -10.5 do 10.5 z dokładnością 1/390 (0.0026), co przekłada się na 2*4096 różnych wartości. W tej sytuacji zdecydowano wykorzystać LUT do zapisania wartości, będących wynikiem funkcji wykładniczej 10^S. Adres w pamięci będą wyznaczały różnice $W_1 = A-R$, $W_2 = B-R$, $W_3 = C-R$.

Ostatecznie wartości RE i IM można obliczyć w następujący sposób:

$$RE = ((ROM[W_3] + ROM[W_2]) >> 1) - ROM[W_1]$$

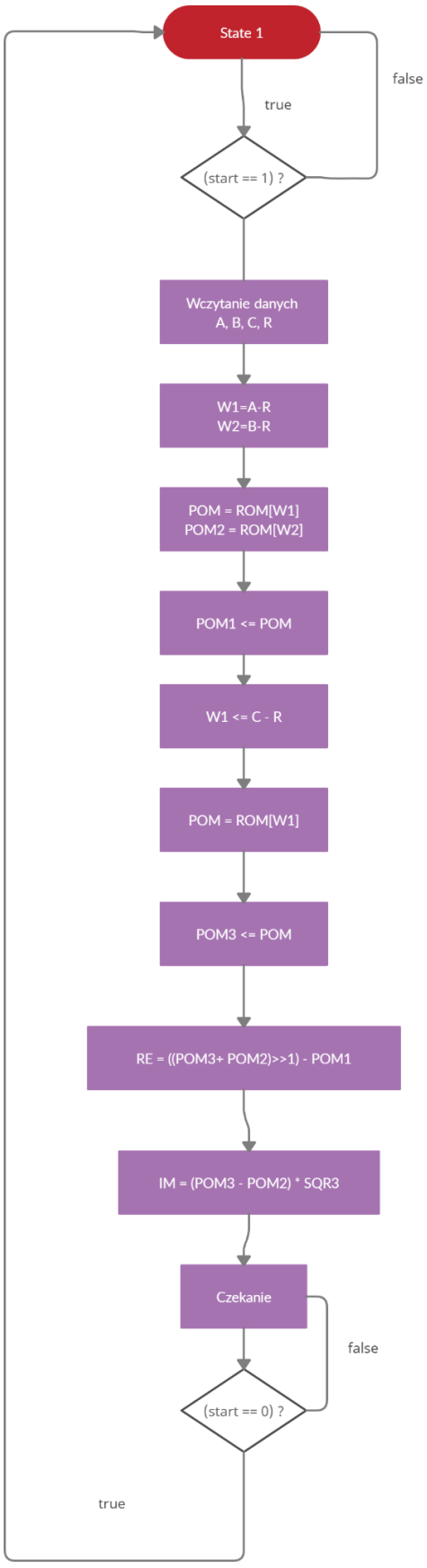
$$IM = (ROM[W_3] - ROM[W_2]) * SQR3$$

gdzie:

SQR3 = 0.5 * √3

ROM – pamięć, gdzie będą przechowywane wartości 10^S

Na rysunku 1 znajduje się schemat blokowy, przedstawiający przebieg programu.



Rysunek 1. Schemat blokowy

Obliczone wartości RE i IM zostaną na końcu podane do bloku FFT dostępnego w bibliotece DSP.

6. Wersja syntezowalna

Poniżej znajdują się dwa moduły – jeden moduł inicjalizuje pamięć dwuportową ROM, natomiast drugi jest to maszyna stanów, na której wyjściu otrzymujemy wartości RE i IM.

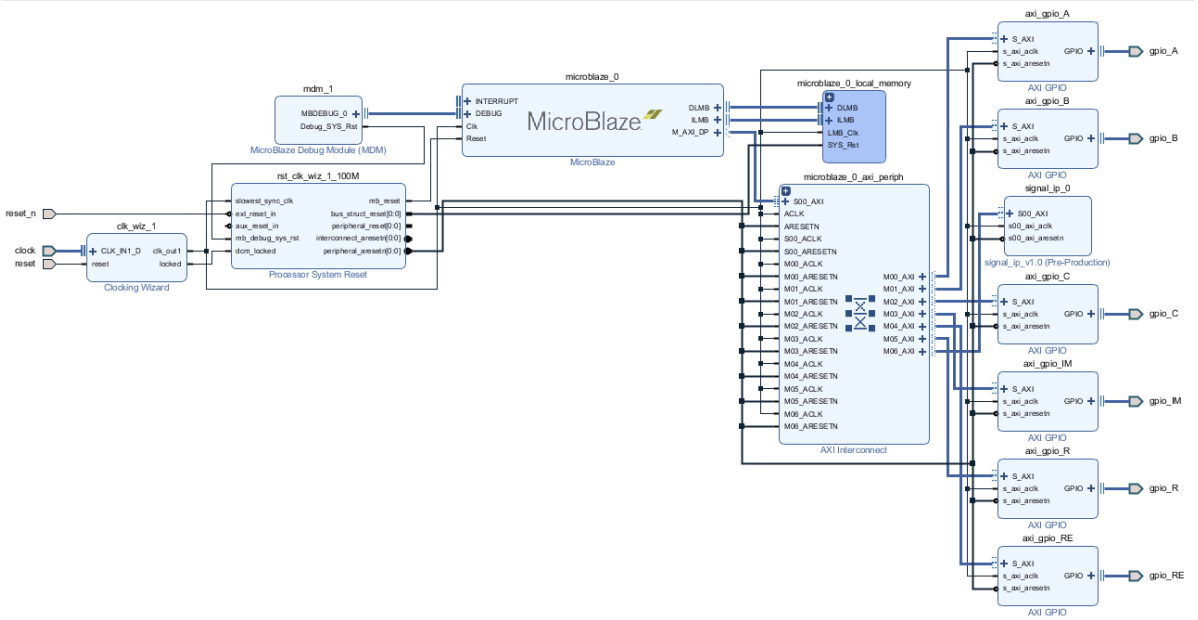
```
module dual_port_rom (
input clock, //clock
        input [11:0] addr_in_0,  //address for port 0
        input [11:0] addr_in_1,  //address for port 1
        output reg [49:0] data_out_0,    //output data from port 0.
        output reg [49:0] data_out_1 //output data from port 1.
   );

//memory declaration.
reg [49:0] rom[8077:0];
parameter MEM_INIT_FILE = "data_hex.txt";

// Initialize RAM from file
initial begin
 if (MEM_INIT_FILE != "") begin
   $readmemh(MEM_INIT_FILE, rom);
 end
end
always @(posedge clock) begin
   // Place data from RAM
      data_out_0 <= rom[addr_in_0];
      data_out_1 <= rom[addr_in_1];
end
endmodule


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

module signal_rtl(
            input  clock,
            input  reset,
            input start, //start processing
            input [11:0] A,
            input [11:0] B,
            input [11:0] C,
            input [11:0] R,

            output reg ready_out, //result is ready
            output reg [49:0] RE,
            output reg [49:0] IM
   );
   real sqrt3 = 0.866;
   reg [11:0] W1;
   reg [11:0] W2;
  // reg [11:0] W3;
   wire [49:0] POM;
```

```
    reg [49:0] POM1;
    wire [49:0] POM2;
    reg [49:0] POM3;
    reg [49:0] ROM;
    reg port_en_0;
    reg port_en_1;

dual_port_rom rom (
        clock, //clock
        W1,  //address for port 0
        W2,  //address for port 1
        POM,    //output data from port 0.
        POM2 //output data from port 1. );
    reg [3:0] state;
    parameter S1 = 4'h01, S2 = 4'h02, S3 = 4'h03, S4 = 4'h04, S5 = 4'h05,
     S6 = 4'h06, S7 = 4'h07;
    always @ (posedge clock)
    begin
        if(reset==1'b1)
        begin
          ready_out <= 1'b0;
          state <= S1;
        end
        else
        begin
           case(state)
              S1: begin
                 if(start == 1'b1) state <= S2; else state <= S1;
              end
              S2: begin
                 W1 <= A - R;
                 W2 <= B - R;
                //  W3 <= C - R;
                 ready_out <= 0;
                 state <= S3;
              end
              S3: begin
                 POM1 <=POM;
                 state <= S4;
              end
              S4:begin
                 W1 <= C - R; //we have dual-port ram so we use address W1 to read data POM3
                state <= S5;
              end
              S5:begin
                 POM3 <= POM;
                state <= S6;
              end
              S6:begin
                 RE = ((POM3 + POM2) >> 1) - POM1;
                 IM = (POM3 - POM2) * sqrt3;
                 ready_out = 1;
```

```
            state <= S7;
          end
          S7:begin
             if(start == 1'b0) state <= S1; else state <=S7;
          end
        endcase
      end
    end
endmodule
```

7. Microblaze

Na podstawie powstałego układu RTL został utworzony projekt z procesorem MicroBlaze, który umożliwił sprawdzenie logiki i działania z procesorem.



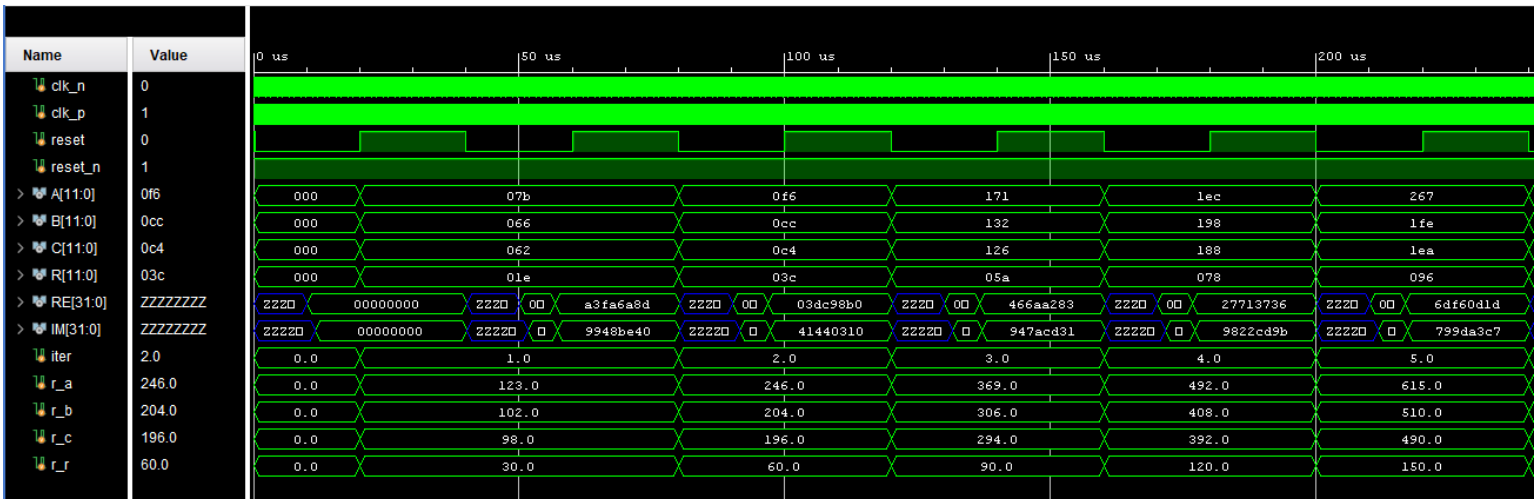
Rysunek 2. Schemat połączeń

Test bench:

```
`timescale 1ns / 1ps
module mb_design_tb;
reg clk_n, clk_p;
reg reset, reset_n;
wire [11:0] A, B, C, R;
wire [31:0] RE, IM;
real iter = 0;
real r_a;
real r_b;
real r_c;
real r_r;
```

```
 // Dip switches stimulus
assign A = r_a;
assign B = r_b;
assign C = r_c;
assign R = r_r;

 // Reset stimulus
 initial
     begin
       reset = 1'b1;
       reset_n = 1'b1;
      #10 reset = 1'b0;
       reset_n = 1'b1;

       end

 always
 begin
 #20000 reset = ~reset;
 end

 // Clocks stimulus
 initial
 begin
 clk_n = 1'b0; //set clk to 0
 clk_p = 1'b1;
 end
 always
 begin
 #5 clk_n = ~clk_n; //toggle clk every 5 time units
 clk_p = ~clk_p; //toggle clk every 5 time units
 end
    //increase iter
    //Signals stimuli
always @ (posedge reset)
   begin
   #20000
   if (iter < 5 ) iter = iter + 1; else iter = 1;
       r_a = 123 * iter;
       r_b = 102 * iter;
       r_c = 98 * iter;
       r_r = 30 * iter;

  end

 //Put sin and cos as real values
// always @*
 //begin
// r_sin = sin;
 //r_cos = cos;
 //r_sin = r_sin / 1024;
 //r_cos = r_cos / 1024;
```

```
//end
//Instantiate tested module
mb_design_wrapper mb_design_inst ( clk_n, clk_p, A, B, C, IM, RE, R, reset, reset_n);

endmodule
```



Rysunek 3. Symulacja z wykorzystaniem MicroBlaze