Perancangan Level 1 Lomba LSI | **Judge The O X of 3x3**

Presentasi Desain Level 1 Lomba LSI
EL4013 - Perancangan Sistem VLSI
2024/2025

Anggota kelompok

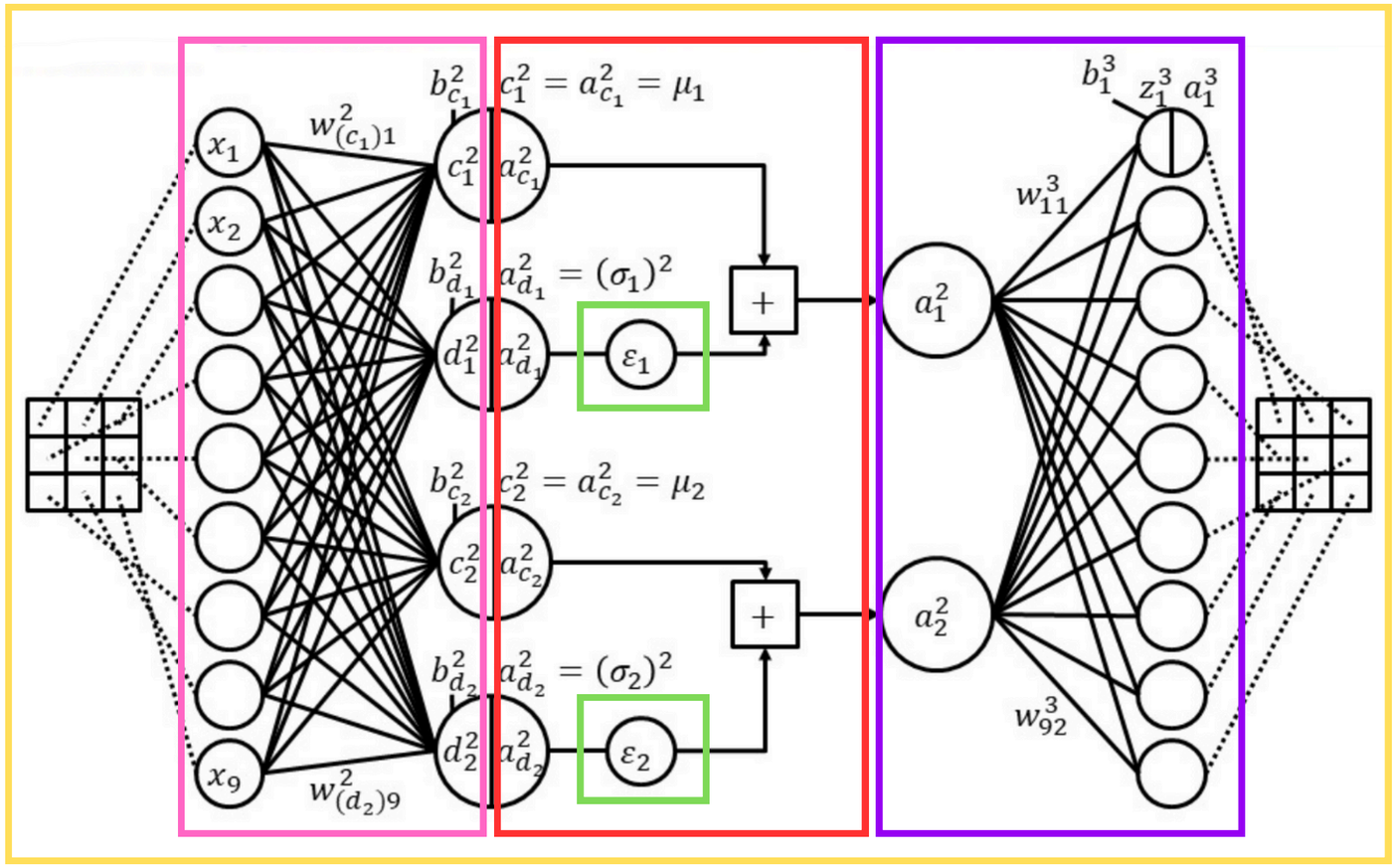
Vanny Alviolani I.
13221020

Rafael Aditya C.W.
13221066

Ahmad Hafidz Aliim
13221055

Maritza Humaira
13221026

Kevin Reagen
13221087



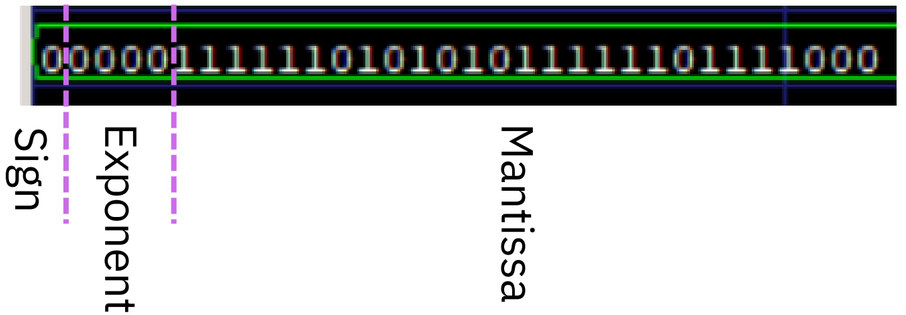
OUTLINE

- Encoder
- Sampling
- Random Generator
- Decoder
- Top Level

Tipe Data

Tipe data yang digunakan pada perancangan ini adalah **fixed point 32 bit signed**, dengan rincian **1 bit signed**, **4 bit exponent**, dan **27 bit mantissa**.

Alasan digunakan tipe data ini adalah diinginkan bentuk desimal/float dengan **akurasi tinggi**, sehingga diperbanyak bit mantissa. **Hanya** terdapat **4 bit exponent** karena sebagian besar nilai output encoder dan decoder pada training AE di MATLAB **dalam rentang -8 hingga 8**.



Encoder

Seluruh nilai **input, weight, bias,** dan output digabung ke dalam variabel bertipe **wire signed** dengan **ukuran n*32 bit**

```
module encoder_fixed_point #(
    parameter N_input = 9,          // Jumlah Input
    parameter M_output = 4,         // Jumlah Output
    parameter BITSIZE = 32          // Fixed Point 32-bit
)(
    input wire signed [N_input*BITSIZE-1:0] x,           // Input
    input wire signed [N_input*M_output*BITSIZE-1:0] w,  // Weight
    input wire signed [M_output*BITSIZE-1:0] b,          // Bias
    output wire signed [M_output*BITSIZE-1:0] out        // Output
);
```

Digunakan juga array untuk menyimpan hasil perkalian dan penjumlahan sementara

```
    // Internal wires for results
    wire signed [BITSIZE-1:0] mult_result [0:N_input-1][0:M_output-1];
    wire signed [BITSIZE-1:0] tree_sum_result [0:M_output-1];
    wire signed [BITSIZE-1:0] final_result [0:M_output-1];
```

Encoder

Perkalian antar **weight dan bias** secara paralel menggunakan block **generate**

```
genvar i, j;

// Perkalian Paralel
generate
    for (i = 0; i < N_input; i = i + 1) begin : gen_x
        for (j = 0; j < M_output; j = j + 1) begin : gen_w
            fixed_point_multiply mult_inst (
                .A(x[(i+1)*BITSIZE-1:i*BITSIZE]),
                .B(w[(j*N_input + i)*BITSIZE +: BITSIZE]),
                .C(mult_result[i][j])
            );
        end
    end
endgenerate
```

Loop generate **men-slice** nilai input dari variabel **x** dan nilai weight dari variabel **w** untuk diinput ke dalam modul fixed_point_multiply

Output dari multiplier disimpan ke array mult_result

Encoder

Penjumlahan hasil perkalian input dan weight dilakukan secara paralel menggunakan **adder tree 4 level**

```
// Adder Tree
generate
    for (j = 0; j < M_output; j = j + 1) begin : gen_adder_tree
        // Level 1: Tambahkan pasangan-pasangan
        wire signed [BITSIZE-1:0] level1_sum [0:4];
        for (i = 0; i < 4; i = i + 1) begin : level1
            fixed_point_add adder_level1 (
                .A(mult_result[2*i][j]),
                .B(mult_result[2*i+1][j]),
                .C(level1_sum[i])
            );
        end
        assign level1_sum[4] = mult_result[8][j]; // Sisa input

        // Level 2: Tambahkan hasil dari Level 1
        wire signed [BITSIZE-1:0] level2_sum [0:2];
        for (i = 0; i < 2; i = i + 1) begin : level2
            fixed_point_add adder_level2 (
                .A(level1_sum[2*i]),
                .B(level1_sum[2*i+1]),
                .C(level2_sum[i])
            );
        end
        assign level2_sum[2] = level1_sum[4]; // Sisa input
```

Level 1 :

- Input 0 + Input 1
- Input 2 + Input 3
- Input 4 + Input 5
- Input 6 + Input 7
- Input 8

Level 2 :

- (Input 0 + Input 1) + (Input 2 + Input 3)
- (Input 4 + Input 5) + (Input6 + Input7)
- Input 8

Encoder

Penjumlahan hasil perkalian input dan weight dilakukan secara paralel menggunakan **adder tree 4 level**

```
        // Level 3: Tambahkan hasil dari Level 2
        wire signed [BITSIZE-1:0] level3_sum;
        fixed_point_add adder_level3 (
            .A(level2_sum[0]),
            .B(level2_sum[1]),
            .C(level3_sum)
        );

        // Final result dari Level 3 dan sisa
        fixed_point_add adder_final (
            .A(level3_sum),
            .B(level2_sum[2]),
            .C(tree_sum_result[j])
        );
    end
endgenerate
```

Level 3 :

- (Input 0 + Input 1) + (Input 2 + Input 3) + (Input 4 + Input 5) + (Input6 + Input7)
- Input 8

Level 4 :

- (Input 0 + Input 1) + (Input 2 + Input 3) + (Input 4 + Input 5) + (Input6 + Input7) + Input 8

Encoder

Penambahan bias untuk setiap output menggunakan blok **generate**

```
// Penambahan Bias
generate
    for (j = 0; j < M_output; j = j + 1) begin : gen_bias
        fixed_point_add adder_bias (
            .A(tree_sum_result[j]),
            .B(b[(j+1)*BITSIZE-1:j*BITSIZE]),
            .C(final_result[j])
        );
        assign out[(j+1)*BITSIZE-1:j*BITSIZE] = final_result[j];
    end
endgenerate
```

Loop generate **men-slice** nilai bias dari variabel **b** untuk dijumlah dengan hasil kali input dengan weight

Encoder

Hasil uji coba dengan testbench

```
module encoder_fixed_point_tb;
    // Parameters
    parameter N_input = 9;
    parameter M_output = 4;
    parameter BITSIZE = 32; // 1-bit sign, 4-bit exponent, 27-bit mantissa

    // Testbench signals
    reg signed [N_input * BITSIZE - 1:0] x;          // Input data
    reg signed [N_input * M_output * BITSIZE - 1:0] w; // Weights
    reg signed [M_output * BITSIZE - 1:0] b;         // Bias
    wire signed [M_output * BITSIZE - 1:0] out;      // Output data

    // Instantiate the DUT
    encoder_fixed_point #(
        .N_input(N_input),
        .M_output(M_output),
        .BITSIZE(BITSIZE)
    ) dut (
        .x(x),
        .w(w),
        .b(b),
        .out(out)
    );

    initial begin
        $dumpfile("encoder_fixed_point_tb.vcd");
        $dumpvars(0, encoder_fixed_point_tb);

        // Initialize inputs
        x = {9{32'b0_0001_10000000000000000000000000}}; // Set all x inputs to 1.5
        w = {36{32'b0_0001_00000000000000000000000000}}; // Set all weights to 1.0
        b = {4{32'b0_0001_00000000000000000000000000}}; // Set all biases to 1.0

        // Wait for a short period to allow calculations to propagate
        #100;
        // Wait for simulation
        #100;

        $stop;
        end

        integer i;
        initial begin
            for (i = 0; i < M_output; i = i + 1) begin
                #1;
                $display("Output[%0d] in in Binary: %b", i, out[(i+1)*BITSIZE-1 -: BITSIZE]);
            end


        // End simulation
        $finish;

    end
endmodule
```

Encoder

Hasil uji coba dengan testbench

Seluruh nilai input = 1.5, weight = 1, bias = 1

```
// Initialize inputs
x = {9{32'b0_0001_10000000000000000000000000}}; // Set all x inputs to 1.5
w = {36{32'b0_0001_00000000000000000000000000}}; // Set all weights to 1.0
b = {4{32'b0_0001_00000000000000000000000000}}; // Set all biases to 1.0
```

Output = (9*1.5)+1 = 14.5

```
C:\Users\LENOVO\Documents\punya vanny\SEMESTER 7\VLSI\Encoder VAE Level 1>vvp encoder_fixed_point_tb
VCD info: dumpfile encoder_fixed_point_tb.vcd opened for output.
Output[0] in in Binary: 01110100000000000000000000000000
Output[1] in in Binary: 01110100000000000000000000000000
Output[2] in in Binary: 01110100000000000000000000000000
Output[3] in in Binary: 01110100000000000000000000000000
encoder_fixed_point_tb.v:55: $finish called at 4000 (1ps)
```

Pseudo Random Generator

True Random tidak mudah untuk diimplementasikan dalam FPGA, karena itu digunakan Pseudo Random Generator yang memanfaatkan seed dan mengubah menjadi hal yang mirip seperti sesuatu yang acak

Digunakan PRNG metode **XORSHIFT**, yaitu dengan melakukan **XOR** dan **Shifting** dari seed acak. hal ini dapat dilakukan beberapa kali untuk memastikan distribusi acak yang lebih rata. selain itu, metode ini juga mudah diimplementasikan dan di integrasikan dengan clock, dimana setiap clock maka akan diacak dari sebelumnya.

Pseudo Random Generator

seed dijadikan paramter, dan bisa langsung mengoutputkan value random yang telah diskalakan dengan tambahan clk dan reset

```
#(
    // Parameter for seed initialization
    parameter SEED = 32'hDEADBEEF,
    parameter a = 13, b = 17, c = 5
)
(
    input wire clk,           // Clock signal
    input wire rst,           // Reset signal
    output reg [31:0] random_out // Random number output
);
    parameter MIN_VALUE = 32'b00000000000000000000000000;
    parameter MAX_VALUE = 32'b00000100000000000000000000;
```

Pseudo Random Generator

rumus XORSHIFT yang digunakan

$$X \leftarrow X \oplus (X \gg a) X \leftarrow X \oplus (X \ll b) X \leftarrow X \oplus (X \gg c)$$

hasil implementasi pada verilog

```
always @(posedge clk or posedge rst) begin
    if (rst) begin
        state <= SEED; // Initialize state with seed on reset
    end else begin
        // XORSHIFT algorithm with a, b, c
        state <= state ^ (state >> a);
        state <= state ^ (state << b);
        state <= state ^ (state >> c);
    end
end
```

Pseudo Random Generator

rumus untuk menskalakan output diantara 2 range (hal ini digunakan karena epsilon di rumus VAE yang diberikan berada di range [0 1]

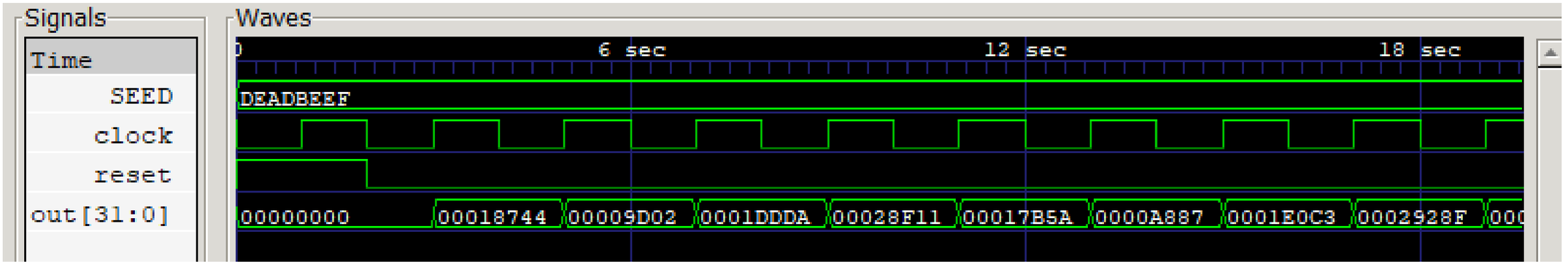
$$\mathrm{output} = (\mathrm{random\_value} \% \mathrm{range}) + \mathrm{min\_value}$$

hasil implementasi pada verilog

```
// Assign state to output
always @(posedge clk or posedge rst) begin
    if (rst) begin
        random_out <= 0; // Reset output
    end else begin
        random_out <= (state % (MAX_VALUE - MIN_VALUE + 1)) + MIN_VALUE;
    end
end
```

Pseudo Random Generator

hasil output GTKWave



output berubah setiap rising edge clock dengan seed yang sama. output berada pada rentang [0 1] atau dalam hexa 32 bit menjadi [00000000 00040000] berhasil di scaling

Pseudo Random Generator

untuk mempermudah pengimplementasian keseluruhan kode, maka keseluruhan block PRNG ini diubah menjadi kombinasional tanpa clock dan reset untuk proof of concept kali ini

hasil implementasi pada verilog

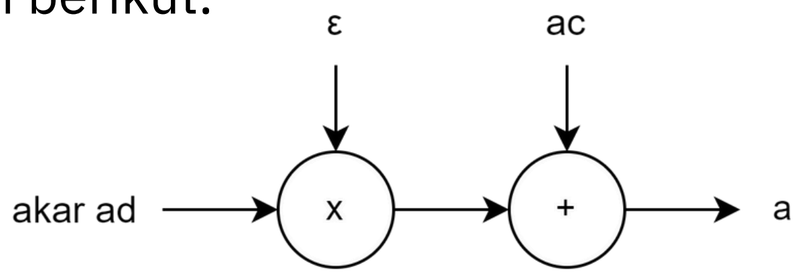
```
// XORSHIFT logic
wire [31:0] state_a, state_b, state_c;

// Internal seed initialization
localparam [31:0] seed = SEED;

// XORSHIFT logic applied to the seed
assign state_a = seed ^ (seed >> a);
assign state_b = state_a ^ (state_a << b);
assign state_c = state_b ^ (state_b >> c);

// Assign final value with modulo for range limitation
assign random_out = (state_c % (MAX_VALUE - MIN_VALUE + 1)) + MIN_VALUE;
```

Sampling

Sampling dilakukan dengan melakukan persamaan berikut.

$$a_i^2 = N(\mu_i, \sigma_i) = \mu_i + \sigma_i \varepsilon_i$$
$$= N\left(a_{c_i}^2, \sqrt{a_{d_i}^2}\right) = a_{c_i}^2 + \sqrt{a_{d_i}^2}\varepsilon_i$$

Maka input yang digunakan adalah nilai ac (mean), ad (variance), dan ε (random). Dengan DFD sebagai berikut.

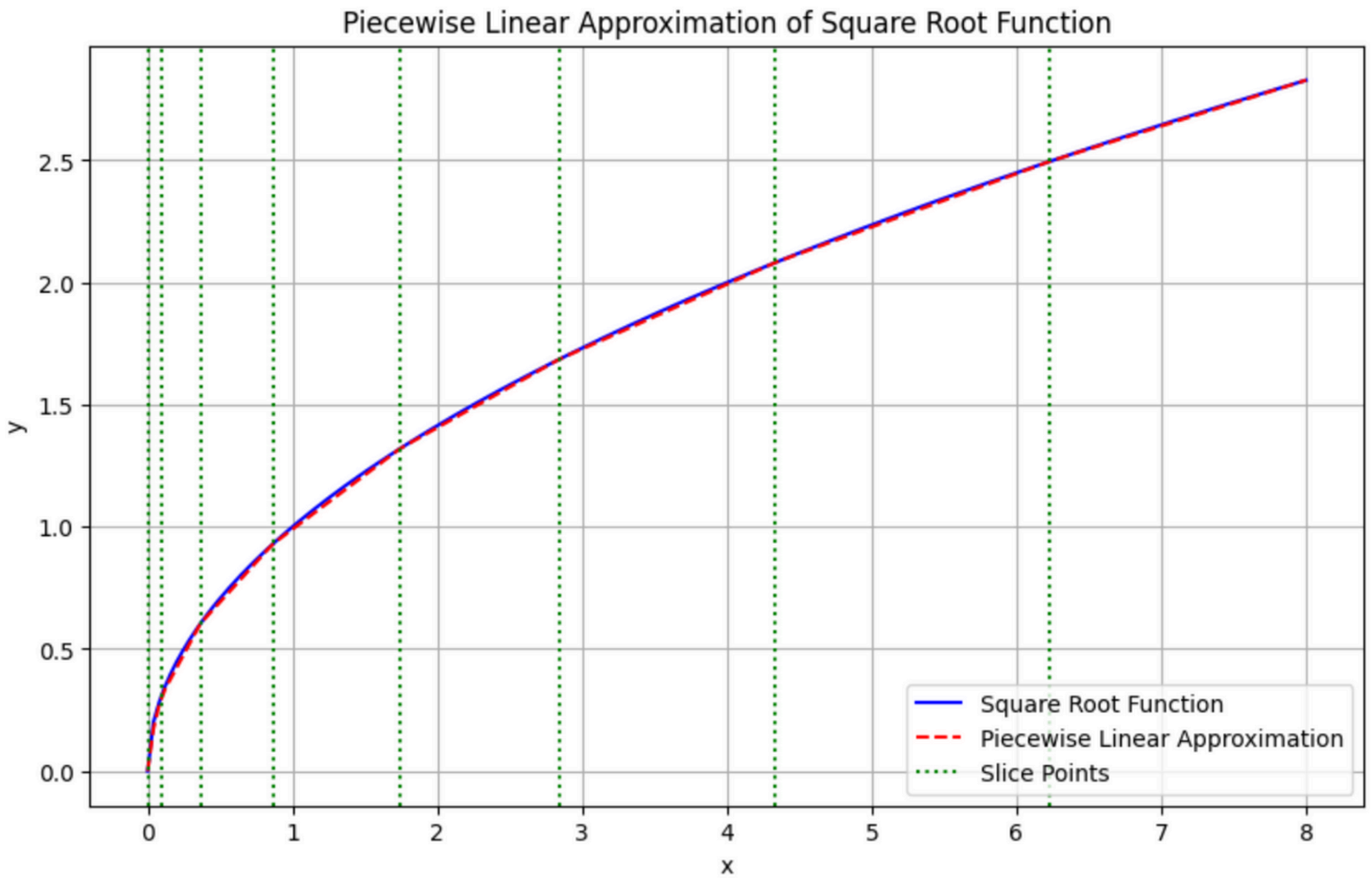


Dilakukan perhitungan akar dua terhadap ad, namun FPGA tidak dapat melakukan operasi ini. Oleh karena itu, digunakan **Piecewise Linear Approximation (PLA) 8 slice** terhadap persamaan akar pangkat dua.

PLA Square Root 8 Slice

Nilai Titik Slice Optimal

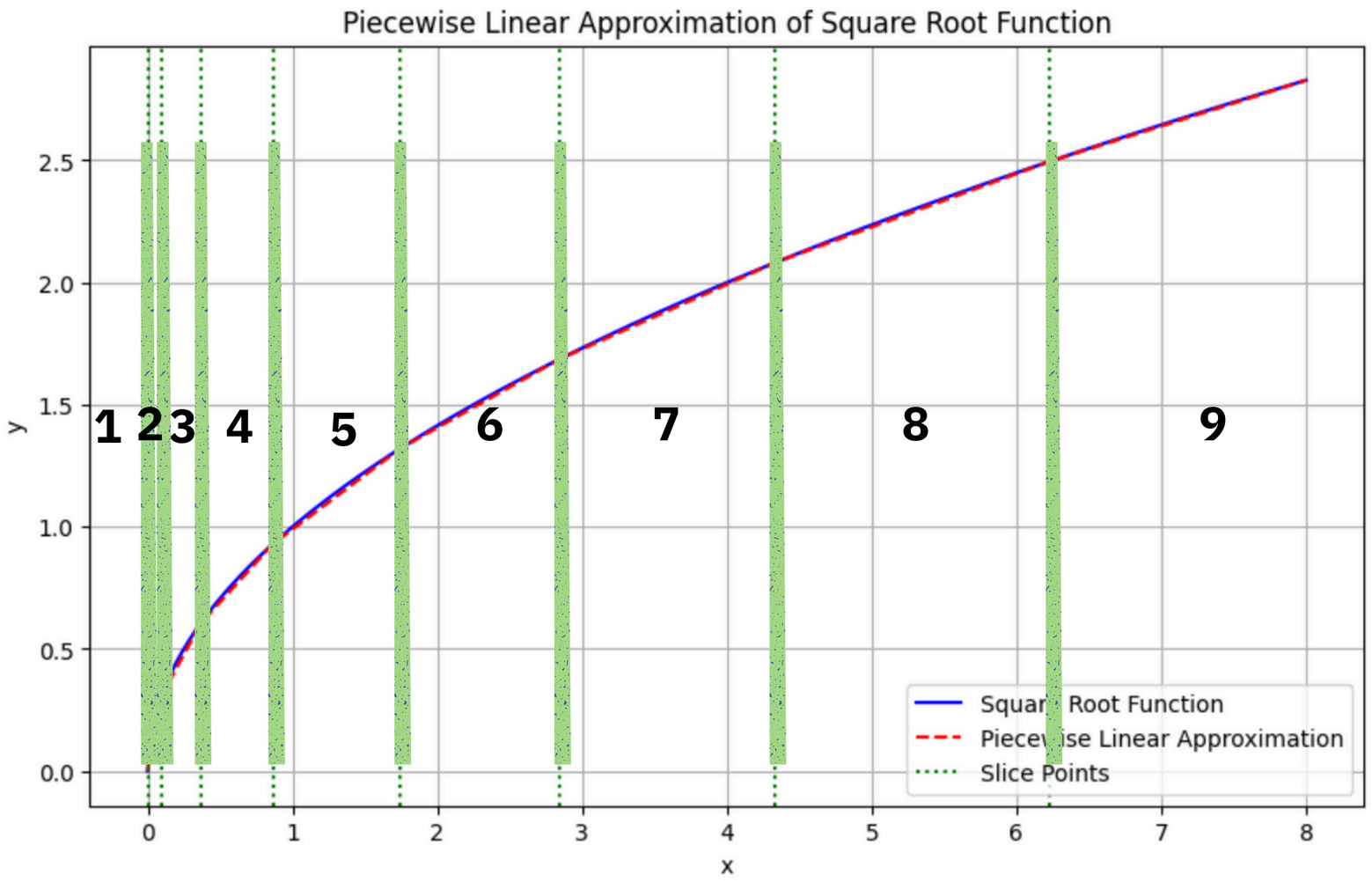
x1 = 0.00916727
x2 = 0.09394428
x3 = 0.36476698
x4 = 0.86324246
x5 = 1.74648981
x6 = 2.84044815
x7 = 4.33743198
x8 = 6.2260287



PLA Square Root 8 Slice

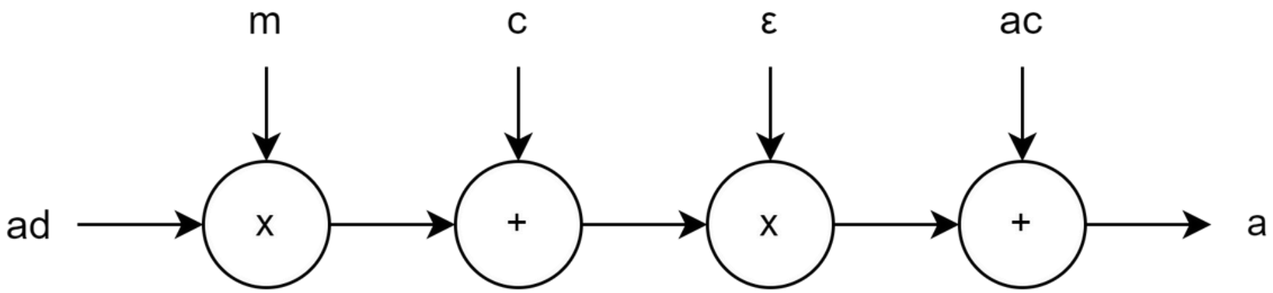
Persamaan Tiap Region

Region 1: y = 10.4443x
Region 2: y = 2.4860x + 0.0730
Region 3: y = 1.0983x + 0.2033
Region 4: y = 0.6523x + 0.3660
Region 5: y = 0.4443x + 0.5456
Region 6: y = 0.3326x + 0.7407
Region 7: y = 0.2654x + 0.9315
Region 8: y = 0.2184x + 1.1352
Region 9: y = 0.1878x + 1.3257



Sampling

Akibat **adanya PLA untuk Square Root**, maka **DFD** menjadi seperti berikut.



Dilakukan perhitungan **PLA terlebih dahulu**, sesuai persamaan garis **y = mx + c**, dengan input x adalah input ad (variance). **Setelah itu** barulah bisa dilakukan perhitungan **persamaan sampling**.

Sampling - Setup Variabel

Seluruh nilai **ac, ad,** dan **a** digabung ke dalam variabel bertipe **wire** dengan **ukuran n*32 bit**

```
input wire [N_input*BITSIZE-1:0] ac,          // Input Mean
input wire [N_input*BITSIZE-1:0] ad,          // Input Variance
output wire [M_output*BITSIZE-1:0] a,         // Output
```

Digunakan juga **array** untuk menyimpan **slope dan intercept** sesuai region, **hasil square root** dengan Piecewise, dan **hasil perkalian penjumlahan** pada persamaan sampling

```
// PENYIMPANAN INTERNAL
// Penyimpanan Bagian Penentuan Region
wire [31:0] m_out [0:N_input-1];        // Array untuk menyimpan m terpilih (tergantung region)
wire [31:0] c_out [0:N_input-1];        // Array untuk menyimpan c terpilih (tergantung region)

// Penyimpanan Hasil Square Root dengan Piecewise
wire [BITSIZE-1:0] mult_result_piecewise [0:N_input-1];         // Hasil perkalian m * x pada persamaan garis
wire [BITSIZE-1:0] add_result_piecewise [0:N_input-1];          // Hasil penjumlahan dengan c pada persamaan garis

// Penyimpanan Hasil Sampling
wire [BITSIZE-1:0] mult_result_sampling [0:N_input-1];      // Hasil perkalian ad dengan epsilon
wire [BITSIZE-1:0] add_result_sampling [0:M_output-1];                  // Penjumlahan ac dengan hasil perkalian
```

Sampling - Mux Menentukan Region

Untuk **menentukan region PLA** yang sesuai untuk input ad (variance), digunakan **multiplexer**.

```
// Compare piecewise
generate
    for (i = 0; i < N_input; i = i + 1) begin : gen_compare
        compare_8float custom_mux (
            .data  (ad[(i+1)*BITSIZE-1:i*BITSIZE]),
            .x1    (x1),
            .x2    (x2),
            .x3    (x3),
            .x4    (x4),
            .x5    (x5),
            .x6    (x6),
            .x7    (x7),
            .x8    (x8),
            .m1    (m1),
            .m2    (m2),
            .m3    (m3),
            .m4    (m4),
            .m5    (m5),
            .m6    (m6),
            .m7    (m7),
            .m8    (m8),
            .m9    (m9),
            .c1    (c1),
            .c2    (c2),
            .c3    (c3),
            .c4    (c4),
            .c5    (c5),
            .c6    (c6),
            .c7    (c7),
            .c8    (c8),
            .c9    (c9),
            .m     (m_out[i]),
            .c     (c_out[i])
        );
    end
endgenerate
```

Loop generate **men-slice** nilai input dari variabel **ad** untuk disesuaikan termasuk region yang mana. **Outputnya** adalah nilai **slope** (m) dan **intercept** (c) sesuai persamaan pada region tersebut.

Sampling - Perhitungan Persamaan Garis

Karena telah diketahui region dari input ad (variance), maka dilakukan **perhitungan persamaan garis y = mx + c** untuk **mengetahui akar pangkat dua** dari input ad tersebut.

```
// Perkalian m dengan ad (sebagai x)
generate
    for (i = 0; i < N_input; i = i + 1) begin : gen_mult_piecewise
        fixed_point_multiply M1 (
            .A(ad[(i+1)*BITSIZE-1:i*BITSIZE]), // Input ad
            .B(m_out[i]),                      // Slope m_out[i]
            .C(mult_result_piecewise[i])       // Hasil perkalian
        );
    end
endgenerate

// Penjumlahan dengan c
generate
    for (i = 0; i < N_input; i = i + 1) begin : gen_add_piecewise
        fixed_point_add A1 (
            .A(mult_result_piecewise[i]),      // Hasil perkalian m * x
            .B(c_out[i]),                      // Intercept m_out[i]
            .C(add_result_piecewise[i])        // Hasil penjumlahan
        );
    end
endgenerate
```

Loop generate perkalian **men-slice** nilai input dari variabel **ad** untuk melakukan **perkalian** dengan nilai **slope** (m). Hasil perkalian ini kemudian **dijumlah** dengan nilai **intercept** (c).

Sampling - Mengambil Nilai ε

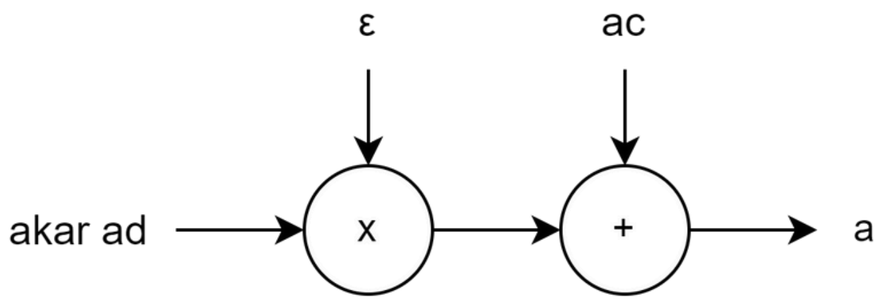
Nilai ad (variance) perlu dikali dengan nilai **random ε**. Nilai random ini **didapatkan dari** blok **Random Generator**.

```
// Ambil nilai epsilon
wire [31:0] e[0:N_input-1];

// Declare PRNG instance
generate
    for (i = 0; i < N_input; i = i + 1) begin : gen_epsilon
        // Multiply z[i] with epsilon (random number)
        PRNG prng_inst (
            .random_out(e[i])  // Connect PRNG output to epsilon
        );
    end
endgenerate
```

Loop generate **mengambil output random** yang kemudian **disimpan di array e**.

Sampling - Perhitungan Sampling

Karena telah diketahui region dari input ad (variance), maka dilakukan **perhitungan persamaan garis y = mx + c** untuk **mengetahui akar pangkat dua** dari input ad tersebut.



Operasi dilakukan sesuai DFG di atas. **Perkalian** dilakukan antara **hasil PLA** dengan nilai random **ε**. Kemudian hasilnya **dijumlahkan** dengan **slice input ac** (mean).

```
// Perkalian akar ad dengan epsilon
// Logic Perkalian Paralel
generate
    for (i = 0; i < N_input; i = i + 1) begin : gen_mult_sampling
        // Multiply z[i] with epsilon (random number)
        fixed_point_multiply mult_inst (
            .A(add_result_piecewise[i]),   // Input z (square root of ad)
            .B(e[i]),                    // Random epsilon from PRNG
            .C(mult_result_sampling[i])    // Hasil perkalian
        );
    end
endgenerate

// Penjumlahan dengan ac
generate
    for (i = 0; i < M_output; i = i + 1) begin : gen_add_sampling
        fixed_point_add A1 (
            .A(mult_result_sampling[i]),       // Hasil perkalian akar ad * epsilon
            .B(ac[(i+1)*BITSIZE-1:i*BITSIZE]), // Input ac
            .C(add_result_sampling[i])        // Hasil penjumlahan
        );
        assign a[(i+1)*BITSIZE-1:i*BITSIZE] = add_result_sampling[i];
    end
endgenerate
```

Sampling - Simulasi

Contoh, digunakan **input** berikut.

```
ac =
{
    32'b00011011011100001010001111010111, // ac2 = 3.43 (MSB)
    32'b00101101001100110011001100110011  // ac1 = 5.65 (LSB)
};

ad =
{
    32'b00000111000010100011110101110000, // ad2 = 0.88 (MSB)
    32'b00010001000010100011110101110000  // ad1 = 2.13 (LSB)
};
```

Maka didapatkan **output** berikut.

```
Output[0] in Hex: 2d3645e8, in Binary: 00101101001101100100010111101000
Output[1] in Hex: 1b72a04b, in Binary: 00011011011100101010000001001011
```

Dalam **desimal**
a[0] = 5.6515005230903625
a[1] = 3.4309697970747948

Dengan menggunakan **kalkulator**, didapatkan hasil berikut.
a[0] = 5.65151
a[1] = 3.43097

Decoder - Setup Variabel

Seluruh nilai **input, weight, bias,** dan output digabung ke dalam variabel bertipe **wire signed** dengan **ukuran N_Input × 32 bit**

```
module decoder_fixed_point #(
    parameter N_input = 2,          // Jumlah Input
    parameter M_output = 9,         // Jumlah Output
    parameter BITSIZE = 32          // Fixed Point 32-bit
)(
    input wire signed [N_input*BITSIZE-1:0] z,           // Input
    input wire signed [N_input*M_output*BITSIZE-1:0] w,  // Weight
    input wire signed [M_output*BITSIZE-1:0] b,          // Bias
    output wire signed [M_output*BITSIZE-1:0] out        // Output
);
```

Digunakan juga array untuk menyimpan hasil perkalian dan penjumlahan sementara

```
// Variabel Penyimpanan Operasi Penjumlahan dan Perkalian
wire signed [BITSIZE-1:0] mult_result [0:N_input-1][0:M_output-1];    // Hasil perkalian
wire signed [BITSIZE-1:0] final_result [0:M_output-1];                // Hasil akhir setelah ditambahkan bias
wire signed [BITSIZE-1:0] tree_sum_result [0:M_output-1];             // Hasil adder tree untuk setiap output
```

Decoder - Perkalian Weight dan Input

Perkalian antar **weight** dengan **input** dilakukan secara **paralel** menggunakan block **generate**

```
// Logic Perkalian Paralel
generate
    for (i = 0; i < N_input; i = i + 1) begin : gen_z
        for (j = 0; j < M_output; j = j + 1) begin : gen_w
            fixed_point_multiply mult_inst (
                .A(z[(i+1)*BITSIZE-1:i*BITSIZE]),
                .B(w[(j*N_input + i)*BITSIZE +: BITSIZE]),
                .C(mult_result[i][j]) // Hasil perkalian
            );
        end
    end
endgenerate
```

Loop generate **men-slice** nilai input dari variabel **x** dan nilai weight dari variabel **w** untuk diinput ke dalam modul fixed_point_multiply

Output dari multiplier disimpan ke array mult_result [i][j]

Decoder - Parallel Adder Tree

Penjumlahan hasil perkalian input dan weight dilakukan secara **paralel** menggunakan **adder tree 1 level** saja

```
// 1-Level Parallel Adder Tree (Penjumlahan Langsung dari 2 Multiplier)
generate
    for (j = 0; j < M_output; j = j + 1) begin : gen_adder_tree
        wire [BITSIZE-1:0] level1_sum;

        fixed_point_add level1_adder
        (
            .A(mult_result[0][j]),
            .B(mult_result[1][j]),
            .C(level1_sum)
        );

        assign tree_sum_result[j] = level1_sum;
    end
endgenerate
```

Jumlah Minimum Tree/Level = log(N_Input)

- Input [0][0] + Input [1][0]
- Input [0][1] + Input [1][1]
- Input [0][2] + Input [1][2]
- Input [0][3] + Input [1][3]
- Input [0][4] + Input [1][4]
- Input [0][5] + Input [1][5]
- Input [0][6] + Input [1][6]
- Input [0][7] + Input [1][7]

Decoder - Penjumlahan dengan Bias

Penambahan bias untuk setiap output menggunakan blok **generate**

```
// Penjumlahan Bias Paralel
generate
    for (j = 0; j < M_output; j = j + 1) begin : gen_bias
        fixed_point_add adder_bias (
            .A(tree_sum_result[j]),                    // Hasil Akhir Penjumlahan dari Modul Perkalian
            .B(b[(j+1)*BITSIZE-1:j*BITSIZE]),          // Bias
            .C(final_result[j])                        // Hasil Akhir Setelah Bias
        );
        assign out[(j+1)*BITSIZE-1:j*BITSIZE] = final_result[j];
    end
endgenerate
```

Loop generate **men-slice** nilai bias dari variabel **b** untuk dijumlah dengan hasil kali input dengan weight

Decoder - Simulasi Output

```
z =
{
    32'b1_0001_00000000000000000000000000, // z[1]
    32'b0_0001_00000000000000000000000000  // z[0]
};

// Weights: 32-bit fixed-point, flattened
w =
{
    32'b0_0001_00000000000000000000000000, // w[1][8]
    32'b0_0001_00000000000000000000000000, // w[0][8]
    32'b0_0011_10000000000000000000000000, // w[1][7]
    32'b0_0110_00000000000000000000000000, // w[0][7]
    32'b1_0111_00000000000000000000000000, // w[1][6]
    32'b0_0111_00000000000000000000000000, // w[0][6]
    32'b0_0111_10000000000000000000000000, // w[1][5]
    32'b1_0111_00000000000000000000000000, // w[0][5]
    32'b0_0111_00000000000000000000000000, // w[1][4]
    32'b1_0111_10000000000000000000000000, // w[0][4]
    32'b0_0111_10000000000000000000000000, // w[1][3]
    32'b1_0111_00000000000000000000000000, // w[0][3]
    32'b0_0111_00000000000000000000000000, // w[1][2]
    32'b0_0111_00000000000000000000000000, // w[0][2]
    32'b0_0011_00000000000000000000000000, // w[1][1]
    32'b0_0110_00000000000000000000000000, // w[0][1]
    32'b0_0001_00000000000000000000000000, // w[1][0]
    32'b0_0001_00000000000000000000000000  // w[0][0]
};

// Biases: 32-bit fixed-point, flattened
b =
{
    32'b0_0001_00000000000000000000000000, // b[8]
    32'b0_0111_10000000000000000000000000, // b[7]
    32'b1_0111_00000000000000000000000000, // b[6]
    32'b0_0111_00000000000000000000000000, // b[5]
    32'b0_0111_10000000000000000000000000, // b[4]
    32'b1_0111_00000000000000000000000000, // b[3]
    32'b0_0001_00000000000000000000000000, // b[2]
    32'b0_0001_00000000000000000000000000, // b[1]
    32'b1_0001_00000000000000000000000000  // b[0]
};
```

Ambil contoh output [5]:

output[5] = {z[0] × w[0][5]} + {z[1] × w[1][5]} + b[5]

output[5] = {1 × (-7)} + {(-1) × 7.5} + 7

output[5] = -7.5

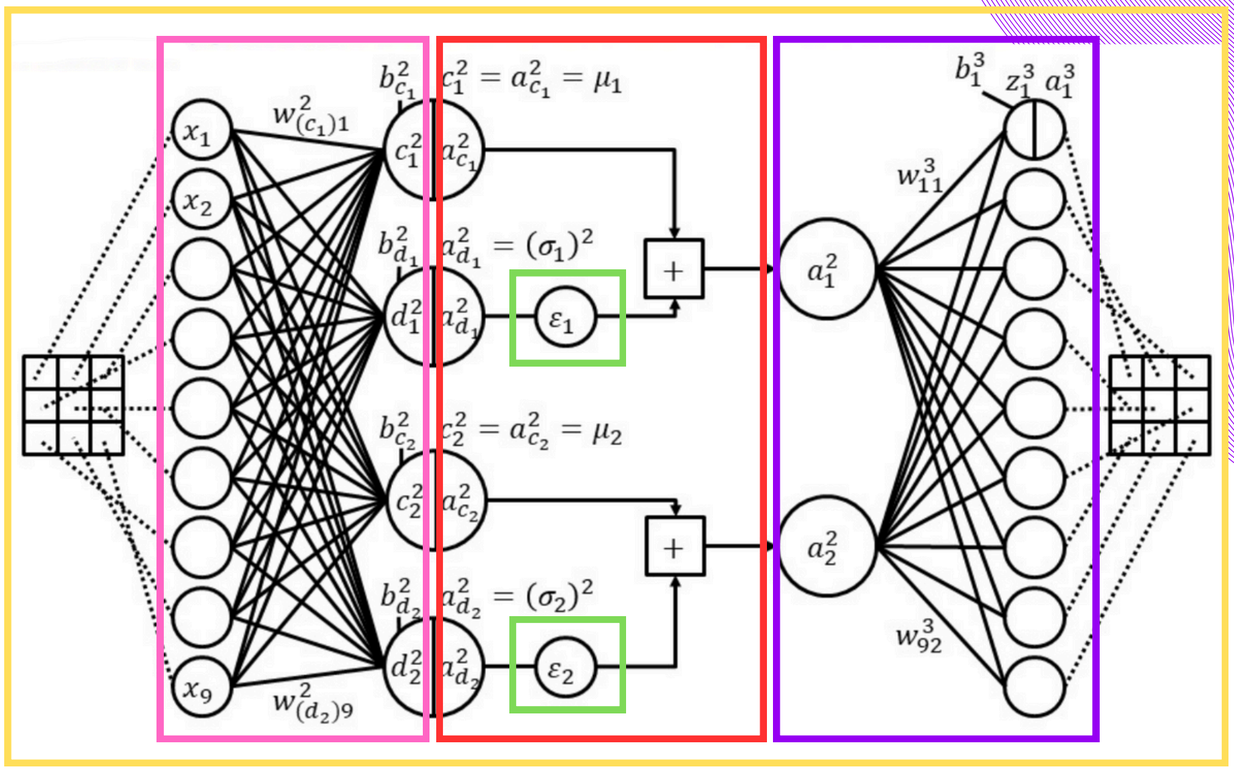
Hasil Simulasi

```
Output[5] in Hex: bc000000, in Binary: 10111100000000000000000000000000
```

Top Level

Pembuatan top level

- Untuk integrasi




```
module top_vae #(
    parameter N_input_enc = 9,        // Input size of encoder
    parameter M_output_enc = 4,       // Output size of encoder
    parameter N_input_sampling = 2,   // Input sampling dari output enc dibagi 2
    parameter M_output_sampling = 2, // besar input output sama
    parameter N_input_dec = 2,        // Input size of decoder (latent space)
    parameter M_output_dec = 9,       // Output size of decoder
    parameter BITSIZE = 32            // Fixed Point 32-bit
)(
    input wire rst,
    input wire signed [N_input_enc*BITSIZE-1:0] x,           // Input for encoder
    output reg signed [M_output_dec*BITSIZE-1:0] sigmoid_out // Final sigmoid output
);
```

Top Level

Isi top level

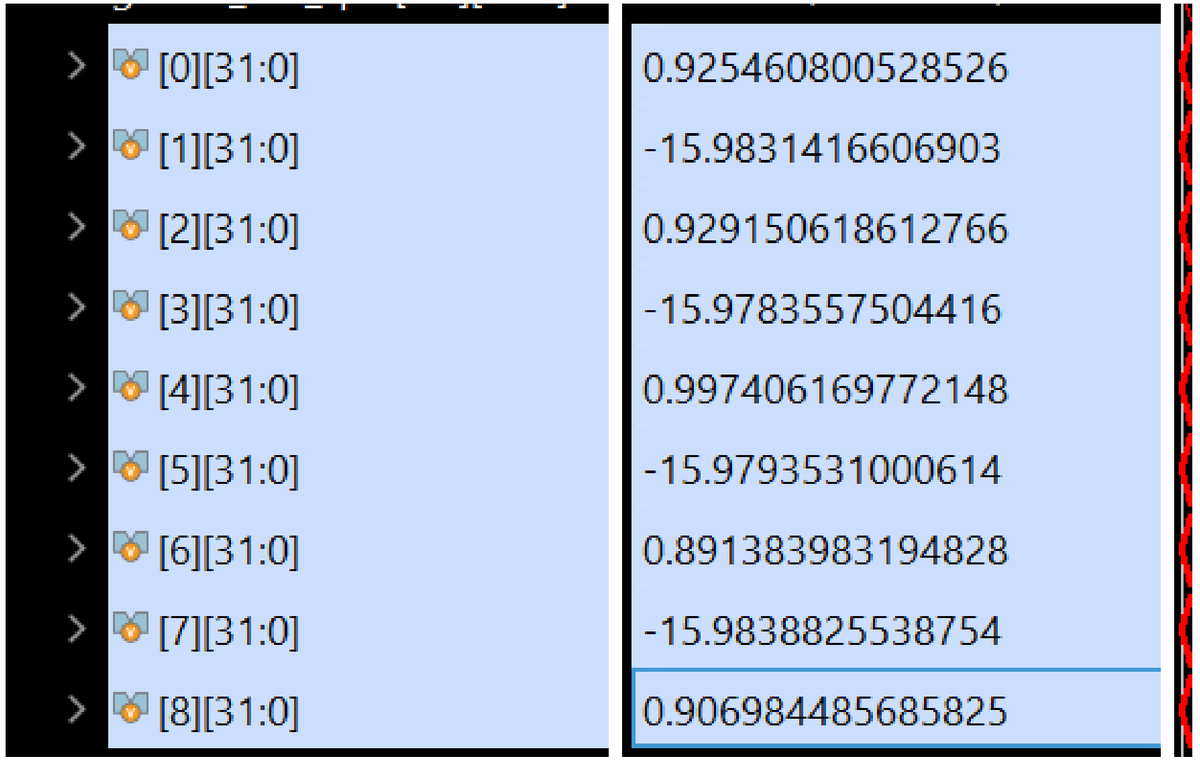
- Deklarasi weight dan bias konstan
- Interkoneksi antar block

```
// Initialize constants
initial begin
    // Encoder weights
    w_enc = {
        32'b1_0000_00000000110110000100010110,
        32'b0_0000_00000000101010100110010110,
        32'b1_0000_00000000111001010110000010,
        32'b0_0000_00000000101000111101011000,
        32'b1_0000_00000000100100000010110111110,
        32'b0_0000_00000000101111100000110111,
```

```
// Instantiate encoder
encoder_fixed_point #(
    .N_input(N_input_enc),
    .M_output(M_output_enc),
    .BITSIZE(BITSIZE)
) encoder_inst (
    .x(x),
    .w(w_enc),
    .b(b_enc),
    .out(encoder_out)
);

// Instantiate softplus for [2*BITSIZE-1:1*BITSIZE-
softplus_8slice softplus_2_inst (
    .data_in(encoder_out[2*BITSIZE-1:1*BITSIZE]),
    .data_out(softplus_out_2)
);
```

Simulasi Top Level

```
tb_x[0*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[1*BITSIZE +: BITSIZE] = 32'b0_0000_00000000000000000000000000;
tb_x[2*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[3*BITSIZE +: BITSIZE] = 32'b0_0000_00000000000000000000000000;
tb_x[4*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[5*BITSIZE +: BITSIZE] = 32'b0_0000_00000000000000000000000000;
tb_x[6*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[7*BITSIZE +: BITSIZE] = 32'b0_0000_00000000000000000000000000;
tb_x[8*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
```

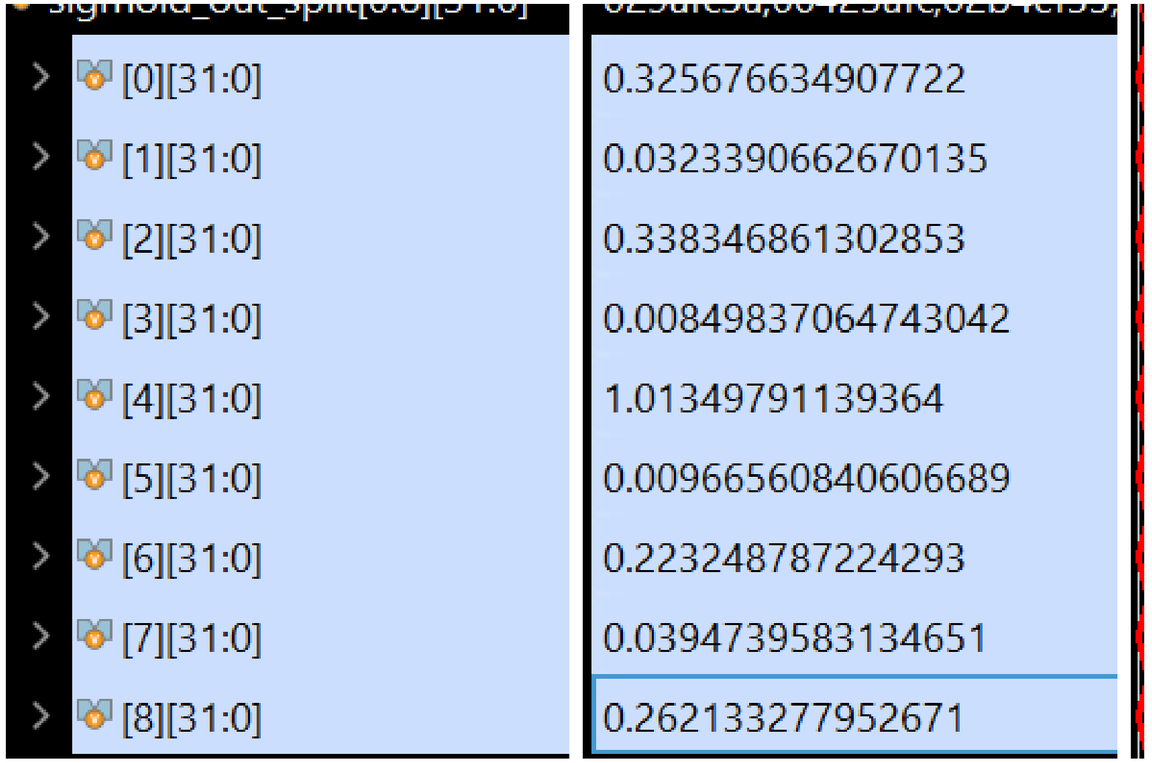


Simulasi

- Input X
- Output -15... merupakan 0. Vivado interpretasi dengan cara 2's complement
- Tipe data custom menggunakan 1's

Simulasi Top Level

```
tb_x[0*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[1*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[2*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[3*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[4*BITSIZE +: BITSIZE] = 32'b0_0000_00000000000000000000000000;
tb_x[5*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[6*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[7*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
tb_x[8*BITSIZE +: BITSIZE] = 32'b0_0001_00000000000000000000000000;
```



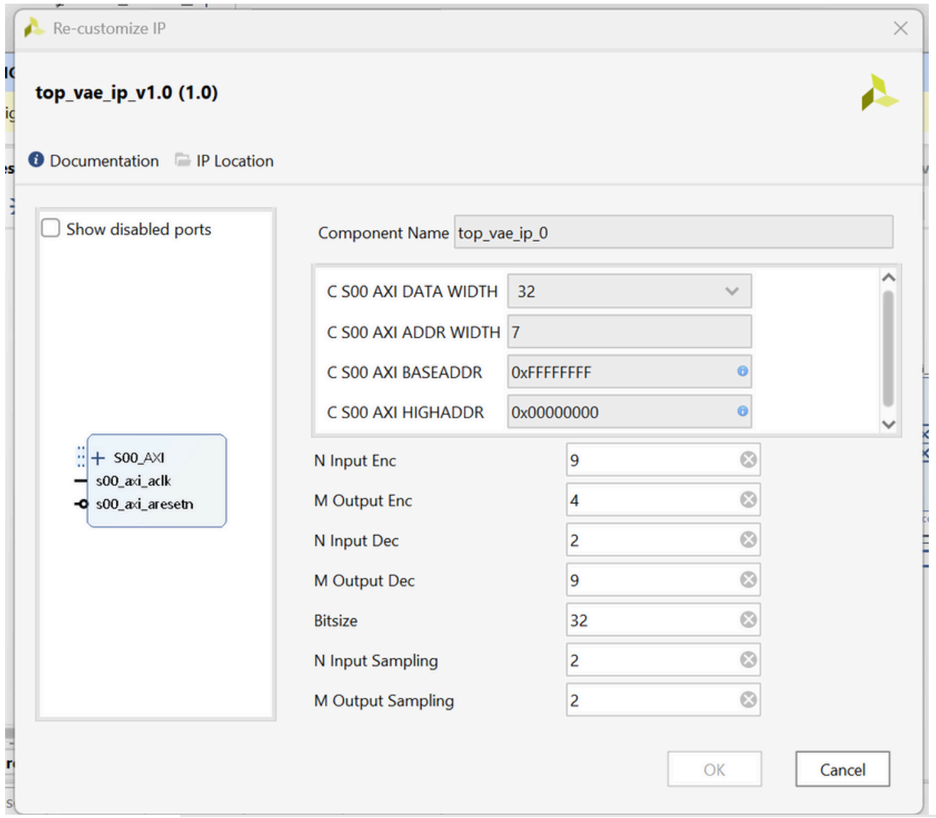
Simulasi

- Input 0
- Output -15... merupakan 0. Vivado interpretasi dengan cara 2's complement
- Tipe data custom menggunakan 1's

Pembuatan AXI-4 Lite

AXI

- Untuk implementasi ke FPGA
- Configurable N, M, BITSIZE
- Terdiri atas minimal 18 Register i/o (9 input dan 9 output)

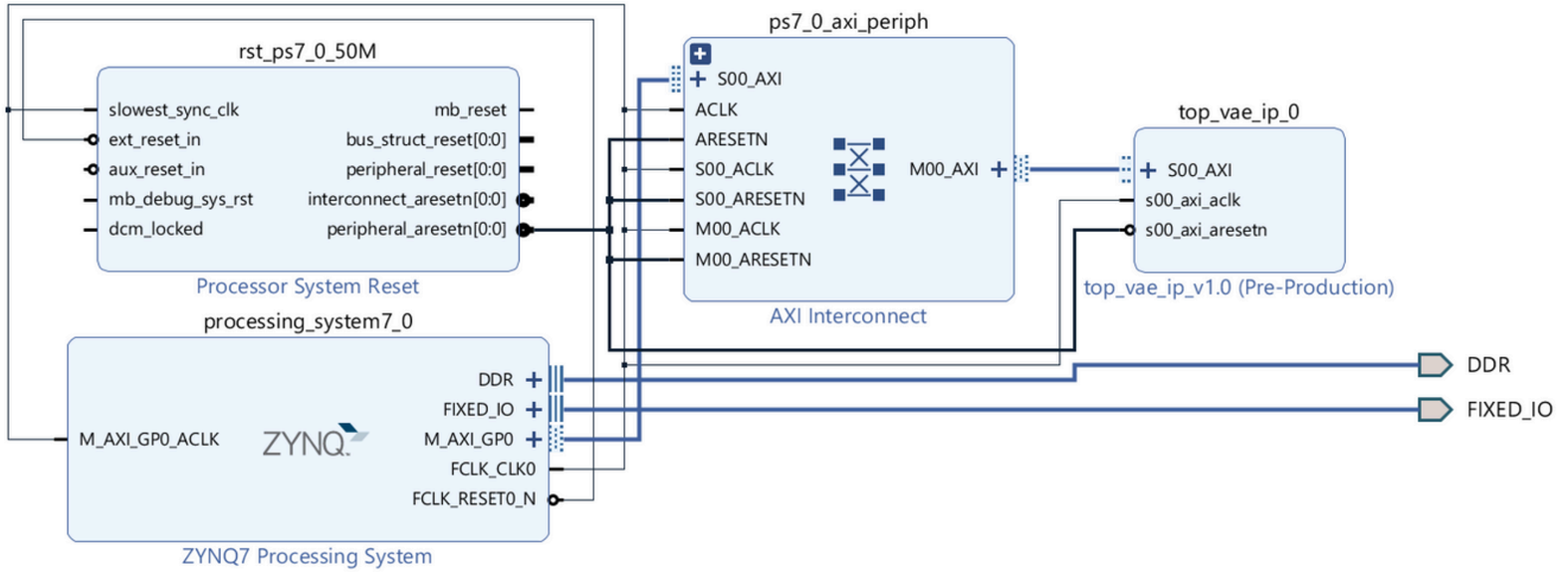



```
// Add user logic here
wire rst = ~S_AXI_ARESETN;
wire [N_input_enc*BITSIZE-1:0] tb_x;
assign tb_x = {slv_reg0, slv_reg1, slv_reg2, slv_reg3, slv_reg4, slv_reg5, slv_reg6, slv_reg7, slv_reg8};

top_vae #(
    .N_input_enc(N_input_enc),
    .M_output_enc(M_output_enc),
    .N_input_sampling(N_input_sampling),  // Input sampling dari output enc dibagi 2
    .M_output_sampling(M_output_sampling), // besar input output sama
    .N_input_dec(N_input_dec),
    .M_output_dec(M_output_dec),
    .BITSIZE(BITSIZE)
) top_vae_1 (
    .rst(rst),
    .x(tb_x),              // Connect the input signal tb_x
    .sigmoid_out(tb_sigmoid_out) // Connect the output signal tb_sigmoid_out
);
```

Block Design

Block Design

- Interaksi IP dengan PS



Synthesis Top Level

Zybo

- Gagal karena design membutuhkan LUT yang terlalu banyak
- Versi -7010 hanya memiliki 17600 LUT dan 80 DSP

LUT	FF	BRAMs	URAM	DSP
37062	1434	0.0	0	72

Synthesis Top Level

Pynq-Z1

- Pynq kapasitasnya lebih besar
- rangkaian masih fully combinational, timing buruk

17399	**1433**	**0.0**	**0**	**196**
16122	**1705**	**0.0**	**0**	**196**

WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
							37062	1434	0.0	0	72
							17399	**1433**	**0.0**	**0**	**196**
-117.834	**-32680.273**	**0.041**	**0.000**	**0.000**	**1.910**	**0**	**16122**	**1705**	**0.0**	**0**	**196**

Setup

Worst Negative Slack (WNS):	-117.834 ns
Total Negative Slack (TNS):	-32680.273 ns
Number of Failing Endpoints:	288
Total Number of Endpoints:	4462

Hold

Worst Hold Slack (WHS):	0.041 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	4462

Pulse Width

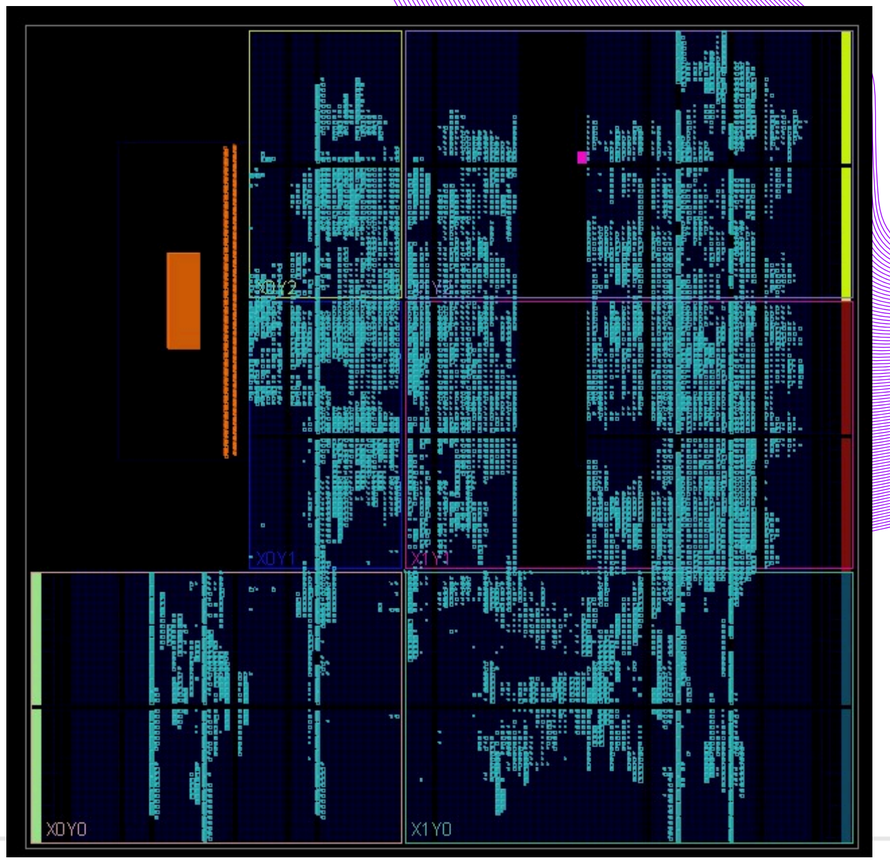
Worst Pulse Width Slack (WPWS):	9.020 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	1770

Timing constraints are not met.

Synthesis Top Level

Pynq-Z1

- Utilitas dan daya



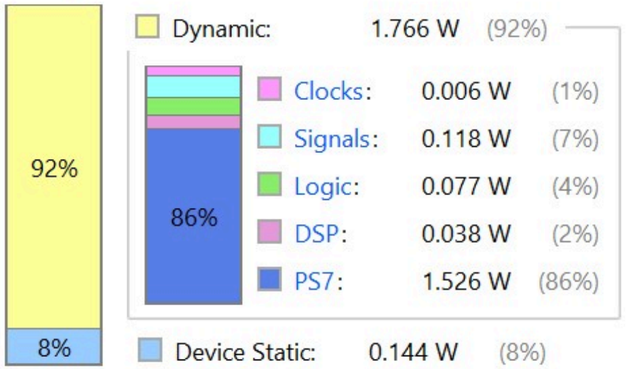
Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.91 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**47.0°C**
Thermal Margin:	38.0°C (3.2 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power



Pengujian di FPGA

Pynq-Z1

- Input X

```
# input array
ol.top_vae_ip_0.write(0,float_to_custom_binary(1))
ol.top_vae_ip_0.write(4,float_to_custom_binary(0))
ol.top_vae_ip_0.write(8,float_to_custom_binary(1))
ol.top_vae_ip_0.write(12,float_to_custom_binary(0))
ol.top_vae_ip_0.write(16,float_to_custom_binary(1))
ol.top_vae_ip_0.write(20,float_to_custom_binary(0))
ol.top_vae_ip_0.write(24,float_to_custom_binary(1))
ol.top_vae_ip_0.write(28,float_to_custom_binary(0))
ol.top_vae_ip_0.write(32,float_to_custom_binary(1))
```

In [42]:

```
addresses = [36, 40, 44, 48, 52, 56, 60, 64, 68]
for addr in addresses:
    print(binary_to_float(pad_binary_to_32bit(bin(ol.top_vae_ip_0.read(addr))[2:])))
```

```
0.9069844856858253
-0.01611744612455368
0.891383983194828
-0.020646899938583374
0.9974061697721481
-0.02164424955844879
0.9291506186127663
-0.016858339309692383
0.9254608005285263
```

Pengujian di FPGA

Pynq-Z1

- Input 0

In [43]:
```
# input array
ol.top_vae_ip_0.write(0,float_to_custom_binary(1))
ol.top_vae_ip_0.write(4,float_to_custom_binary(1))
ol.top_vae_ip_0.write(8,float_to_custom_binary(1))
ol.top_vae_ip_0.write(12,float_to_custom_binary(1))
ol.top_vae_ip_0.write(16,float_to_custom_binary(0))
ol.top_vae_ip_0.write(20,float_to_custom_binary(1))
ol.top_vae_ip_0.write(24,float_to_custom_binary(1))
ol.top_vae_ip_0.write(28,float_to_custom_binary(1))
ol.top_vae_ip_0.write(32,float_to_custom_binary(1))
```

In [45]:
```
addresses = [36, 40, 44, 48, 52, 56, 60, 64, 68]
for addr in addresses:
    print(binary_to_float(pad_binary_to_32bit(bin(ol.top_vae_ip_0.read(addr))[2:])))
```

```
0.26213327795267105
0.03947395831346512
0.22324878722429276
0.009665608406066895
1.0134979113936424
0.00849837064743042
0.33834686130285263
0.03233906626701355
0.3256766349077225
```

Pengujian di FPGA

Pynq-Z1

- Input X with 1 bit error

```
ol.top_vae_ip_0.write(0,float_to_custom_binary(1))
ol.top_vae_ip_0.write(4,float_to_custom_binary(0))
ol.top_vae_ip_0.write(8,float_to_custom_binary(0))
ol.top_vae_ip_0.write(12,float_to_custom_binary(0))
ol.top_vae_ip_0.write(16,float_to_custom_binary(1))
ol.top_vae_ip_0.write(20,float_to_custom_binary(0))
ol.top_vae_ip_0.write(24,float_to_custom_binary(1))
ol.top_vae_ip_0.write(28,float_to_custom_binary(0))
ol.top_vae_ip_0.write(32,float_to_custom_binary(1))
```

In [7]:

```
addresses = [36, 40, 44, 48, 52, 56, 60, 64, 68]
for addr in addresses:
    print(binary_to_float(pad_binary_to_32bit(bin(ol.top_vae_ip_0.read(addr))[2:])))
```

```
0.9771797880530357
0.0033284202218055725
0.972045972943306
-0.0008230060338973999
0.8340208753943443
-0.0017200857400894165
0.9836511611938477
0.002651803195476532
0.982573114335537
```

Pengujian di FPGA

Pynq-Z1

- Input O with 1 bit error

```
# input array
ol.top_vae_ip_0.write(0,float_to_custom_binary(1))
ol.top_vae_ip_0.write(4,float_to_custom_binary(1))
ol.top_vae_ip_0.write(8,float_to_custom_binary(1))
ol.top_vae_ip_0.write(12,float_to_custom_binary(1))
ol.top_vae_ip_0.write(16,float_to_custom_binary(0))
ol.top_vae_ip_0.write(20,float_to_custom_binary(1))
ol.top_vae_ip_0.write(24,float_to_custom_binary(1))
ol.top_vae_ip_0.write(28,float_to_custom_binary(1))
ol.top_vae_ip_0.write(32,float_to_custom_binary(0))
```

In [20]:

```
addresses = [36, 40, 44, 48, 52, 56, 60, 64, 68]
for addr in addresses:
    print(binary_to_float(pad_binary_to_32bit(bin(ol.top_vae_ip_0.read(addr))[2:])))
```

```
0.7499014958739281
-0.010361678898334503
0.6987052634358406
-0.015254572033882141
0.9902450069785118
-0.016290143132209778
0.7926401123404503
-0.01115599274635315
0.7855302840471268
```

Thank You!