25-SUMMER SNU INTERNSHIP

LayerNorm Approximation FPGA

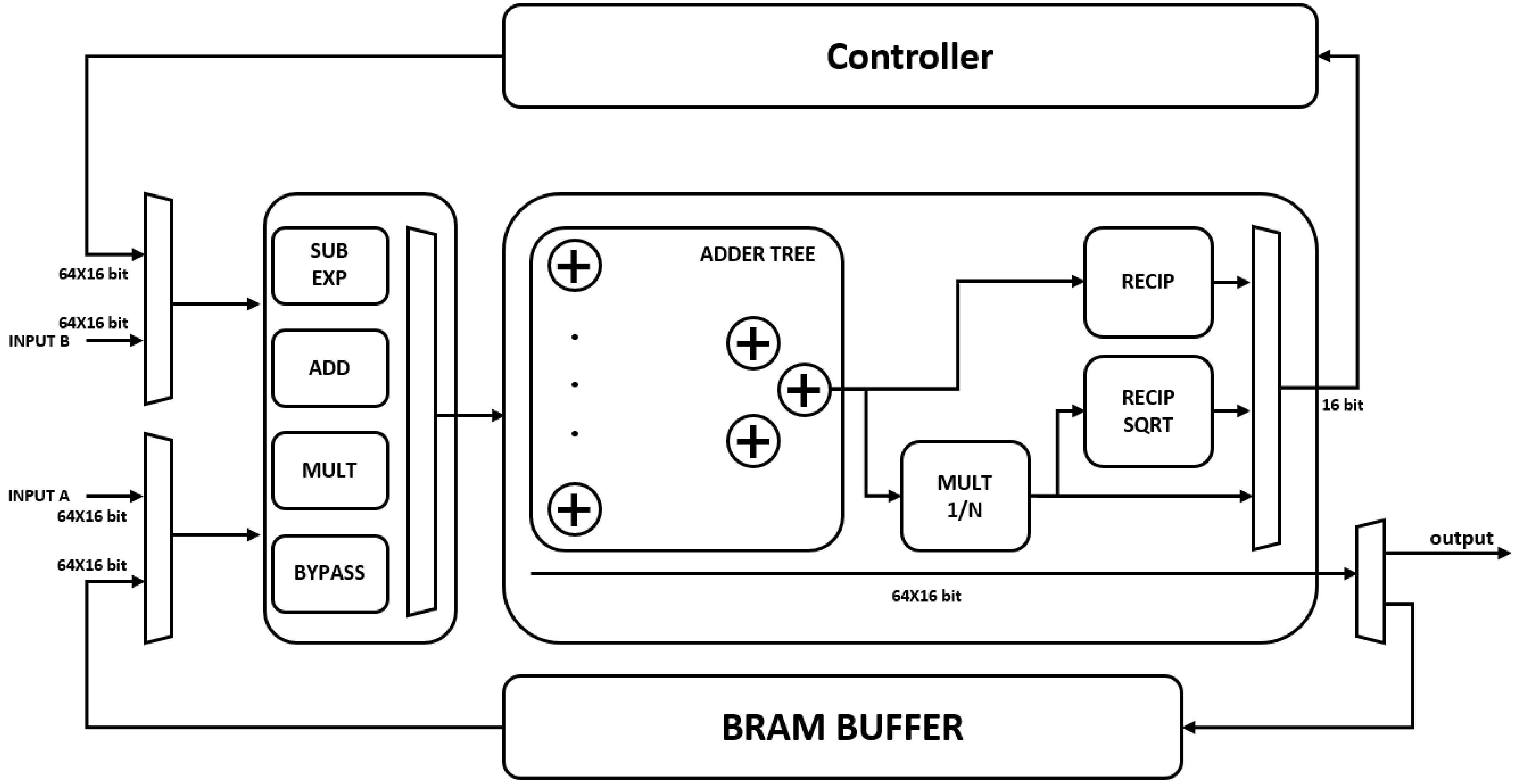
2025.7.28.

Sangyun Kim
Eunju Kim

CONTENTS

1. **Standard LN Model**
2. **LN Approximation Model**
3. **Revised LN Approximation Model**
4. **Compare LN Approximation Model**
5. **Apply FPGA**
6. **Conclusion**

Standard LN Model



Problems of Standard Layer Normalization (LN)

1. Sequential Processing
→ Mean → Variance → Normalize
(3 steps, not parallelizable)

2. Hardware Inefficiency
→ Involves square root & division
(slow on FPGA/GPU)

3. High Latency
→ 52 cycles for 512 inputs
→ 9–16% of total runtime in real inference

LN Approximation Model

Improvements in Pairwise LN

1. Fully Pipelined

→ Group-wise parallel computation

→ No waiting between stages

2. Fast & Accurate

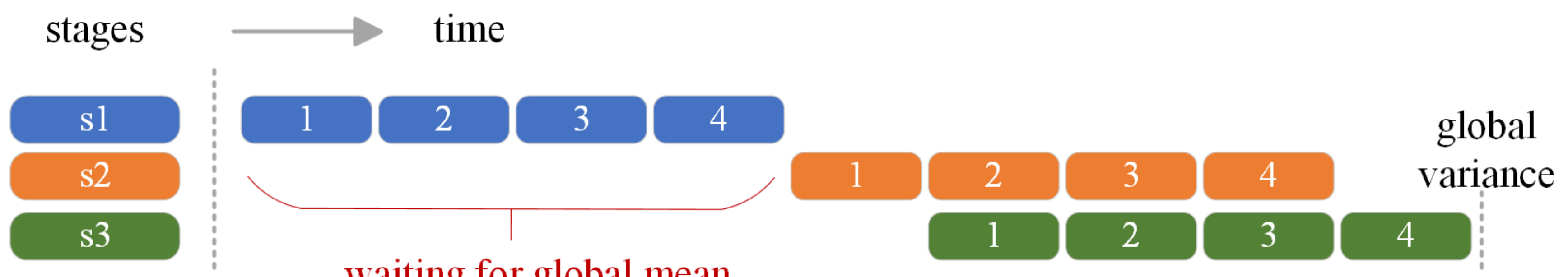
→ Reduces latency by **27%** (52 → 38 cycles)

→ Maintains accuracy (≤ **1.09%** drop)

3. Efficient Hardware Use

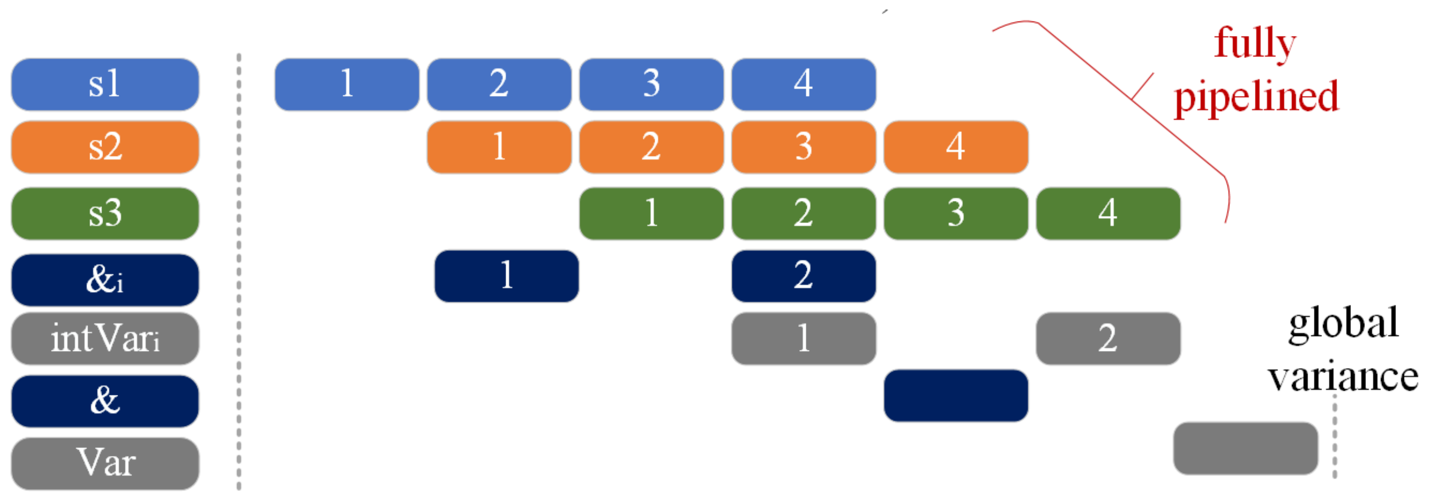
→ Best **Throughput per LUT/DSP** among all methods

Standard LN Model



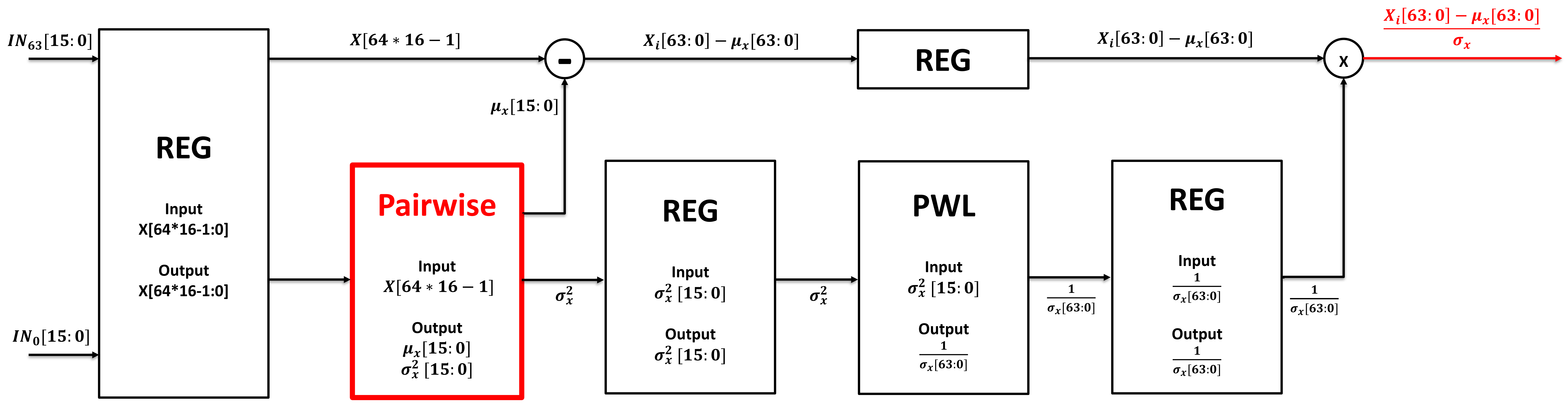
a)

LN Approximation Model



b)

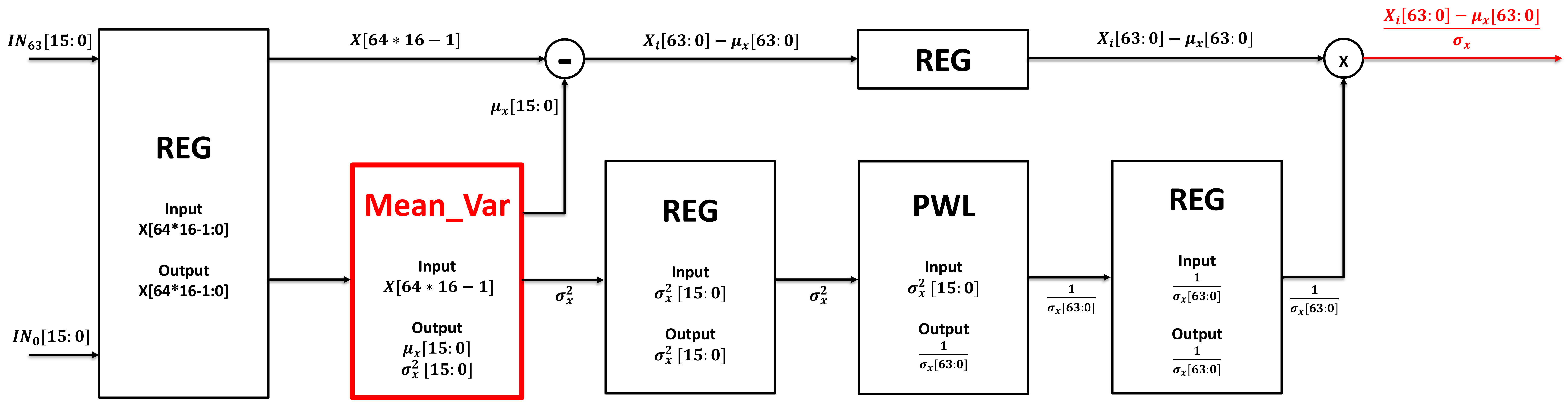
LN Approximation Model



Using Mult(4 Latency), Add(1 Latency), Sub(1 Latency) IP → when Layer Normalization.

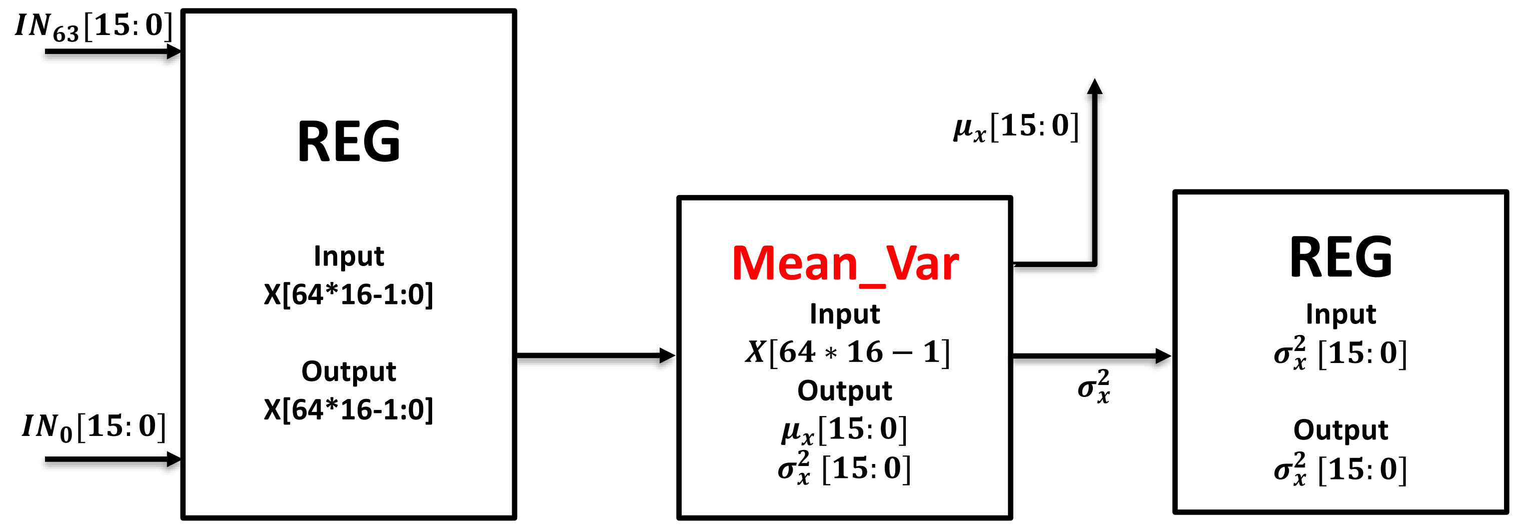
Error due to **excessive resource usage.**
Occurred during **pairwise computation.**

Revised LN Approximation Model

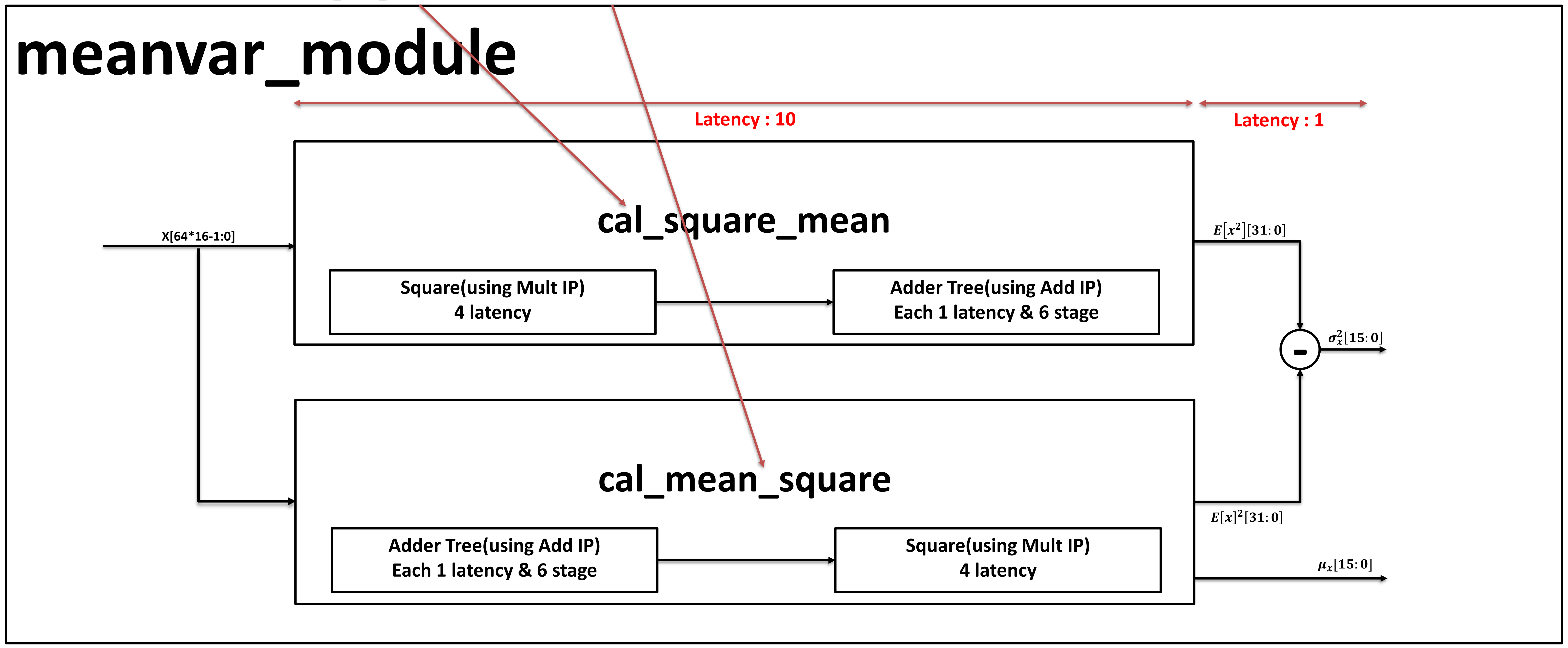


We resolved the error by **replacing the pairwise module with our revised Mean_var module.**

Revised LN Approximation Model



$$\sigma_x^2[15:0] = E[x^2][31:0] - E[x]^2[31:0]$$



Revised LN Approximation Model

Cal_square_mean module

Input data : X[64*16-1:0]

Output data : $E\left[x^2\right][31:0]$

```
wire [15:0] x [0:63];
wire [31:0] x_sq [0:63];
```

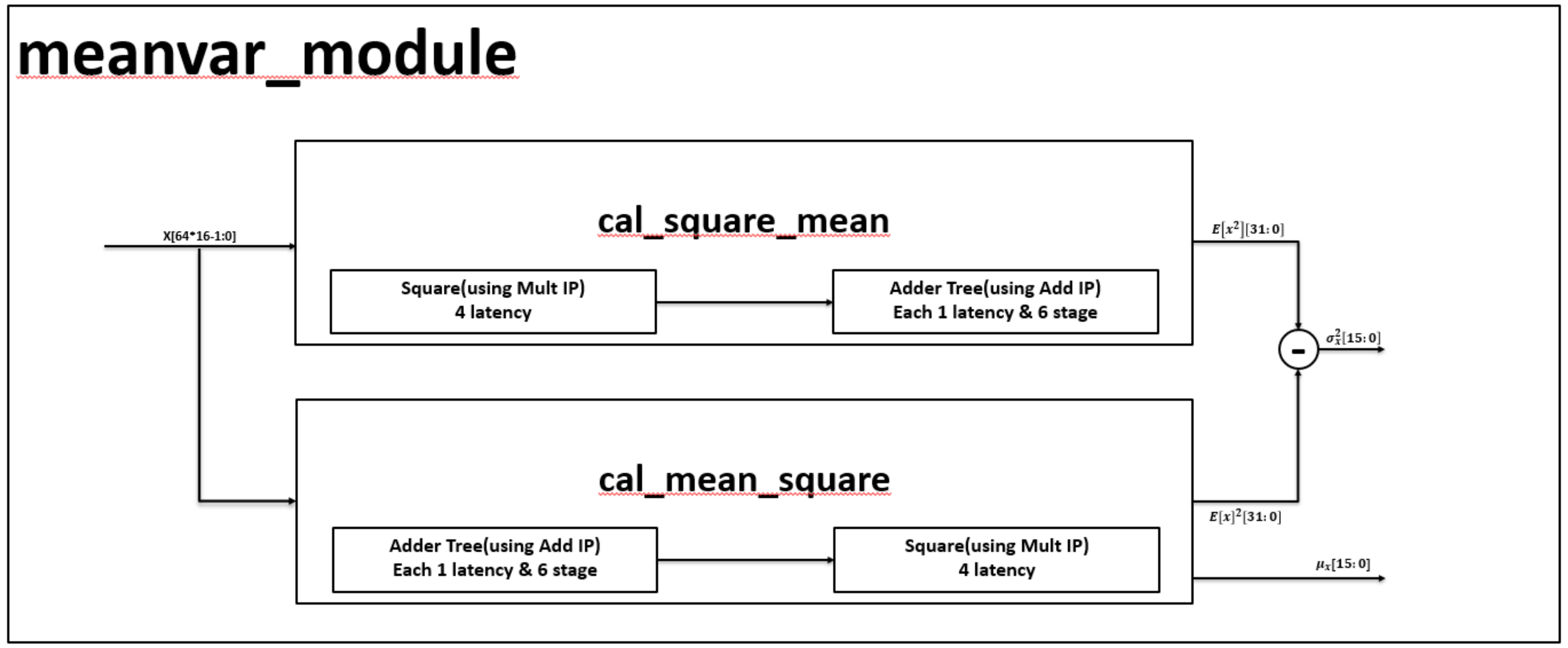
```
genvar i;
generate
    for (i = 0; i < 64; i = i + 1) begin : MUL
        assign x[i] = a_in[i*16 +: 16];
        mult_gen_mult mul_inst (
            .A(x[i]),
            .B(x[i]),
            .CLK(clk),
            .P(x_sq[i])    // Q16.16
        );
    end
endgenerate
```

$\rightarrow x^2$

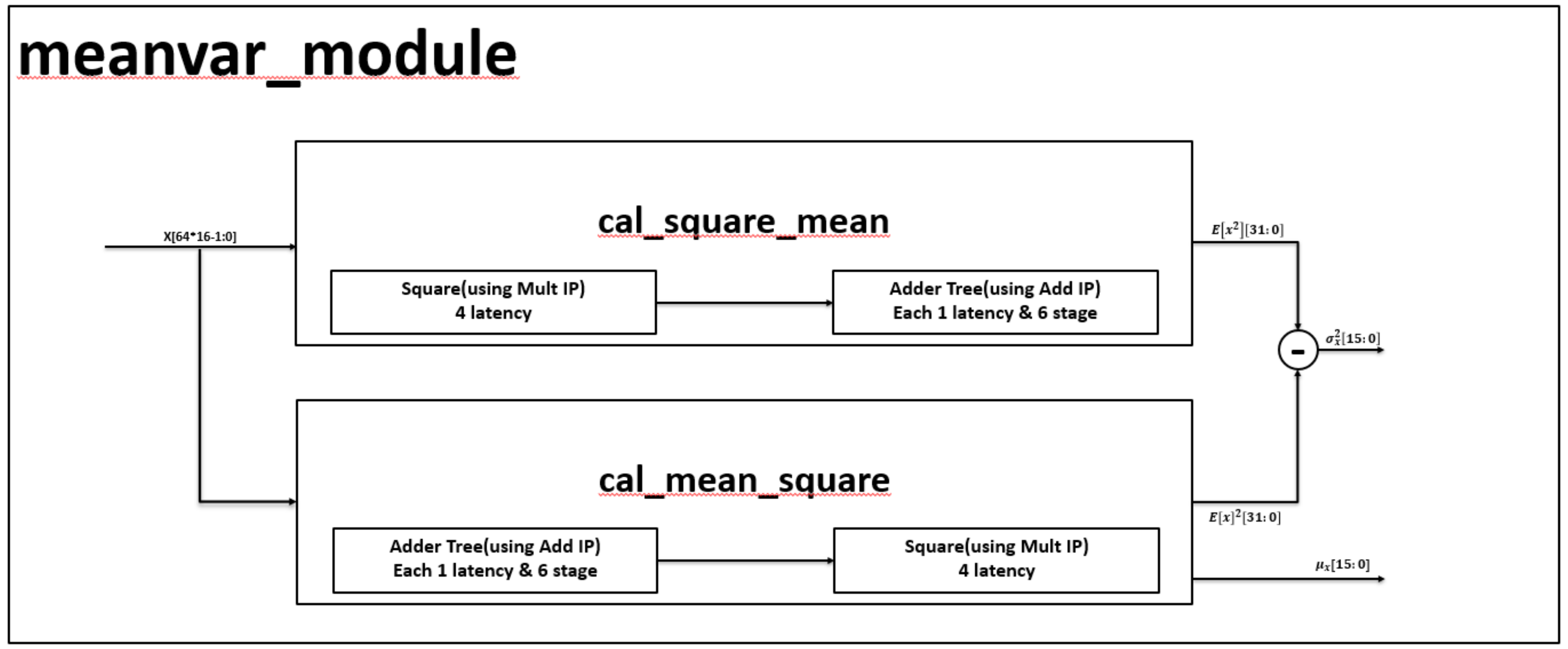
```
generate
    for (i = 0; i < 32; i = i + 1)
        c_addsub_add32 add1 (.A(x_sq[2*i]), .B(x_sq[2*i+1]), .CLK(clk), .CE(1'b1), .S(add_stage1[i]));
    for (i = 0; i < 16; i = i + 1)
        c_addsub_add32 add2 (.A(add_stage1[2*i]), .B(add_stage1[2*i+1]), .CLK(clk), .CE(1'b1), .S(add_stage2[i]));
    for (i = 0; i < 8; i = i + 1)
        c_addsub_add32 add3 (.A(add_stage2[2*i]), .B(add_stage2[2*i+1]), .CLK(clk), .CE(1'b1), .S(add_stage3[i]));
    for (i = 0; i < 4; i = i + 1)
        c_addsub_add32 add4 (.A(add_stage3[2*i]), .B(add_stage3[2*i+1]), .CLK(clk), .CE(1'b1), .S(add_stage4[i]));
    for (i = 0; i < 2; i = i + 1)
        c_addsub_add32 add5 (.A(add_stage4[2*i]), .B(add_stage4[2*i+1]), .CLK(clk), .CE(1'b1), .S(add_stage5[i]));
endgenerate

c_addsub_add32 final_add (.A(add_stage5[0]), .B(add_stage5[1]), .CLK(clk), .CE(1'b1), .S(sum_sq));
```

$\rightarrow sum\_sq \gg 6 = E\left[x^2\right][31:0]$


meanvar_module
X[64*16-1:0]
cal_square_mean
Square(using Mult IP) 4 latency
Adder Tree(using Add IP) Each 1 latency & 6 stage
$E[x^2][31:0]$
$\sigma_x^2[15:0]$
cal_mean_square
Adder Tree(using Add IP) Each 1 latency & 6 stage
Square(using Mult IP) 4 latency
$E[x]^2[31:0]$
$\mu_x[15:0]$

Revised LN Approximation Model


meanvar_module
X[64*16-1:0]
cal_square_mean
Square(using Mult IP) 4 latency
Adder Tree(using Add IP) Each 1 latency & 6 stage
$E[x^2][31:0]$
-
$\sigma_x^2[15:0]$
cal_mean_square
Adder Tree(using Add IP) Each 1 latency & 6 stage
Square(using Mult IP) 4 latency
$E[x]^2[31:0]$
$\mu_x[15:0]$

Cal_mean _square module

Input data : X[64*16-1:0]

Output data : $E[x]^2[31:0]$ **/** $E[x][15:0]$

```
generate
    for (i = 0; i < 32; i = i + 1) begin : ADD1
        c_addsub_add add_inst (.A(x[2*i]), .B(x[2*i+1]), .CLK(clk), .CE(1'b1), .S(add_stage1[i]));
    end
    for (i = 0; i < 16; i = i + 1) begin : ADD2
        c_addsub_add add_inst (.A(add_stage1[2*i]), .B(add_stage1[2*i+1]), .CLK(clk), .CE(1'b1), .S(add_stage2[i]));
    end
    for (i = 0; i < 8; i = i + 1) begin : ADD3
        c_addsub_add add_inst (.A(add_stage2[2*i]), .B(add_stage2[2*i+1]), .CLK(clk), .CE(1'b1), .S(add_stage3[i]));
    end
    for (i = 0; i < 4; i = i + 1) begin : ADD4
        c_addsub_add add_inst (.A(add_stage3[2*i]), .B(add_stage3[2*i+1]), .CLK(clk), .CE(1'b1), .S(add_stage4[i]));
    end
    for (i = 0; i < 2; i = i + 1) begin : ADD5
        c_addsub_add add_inst (.A(add_stage4[2*i]), .B(add_stage4[2*i+1]), .CLK(clk), .CE(1'b1), .S(add_stage5[i]));
    end
endgenerate

c_addsub_add add_inst_final (.A(add_stage5[0]), .B(add_stage5[1]), .CLK(clk), .CE(1'b1), .S(sum_x));
```

$$sum\_x \gg 6 = E[x][15:0]$$

```
wire [15:0] mean_q8 = sum_x >> 6;          // calculate mean
wire [31:0] mean_sq_out;
wire        mult_valid;

mult_gen_mult mult_inst (
    .A(mean_q8),
    .B(mean_q8),
    .CLK(clk),
    .P(mean_sq_out)
);
```

$$E[x]^2[31:0]$$

Revised LN Approximation Model

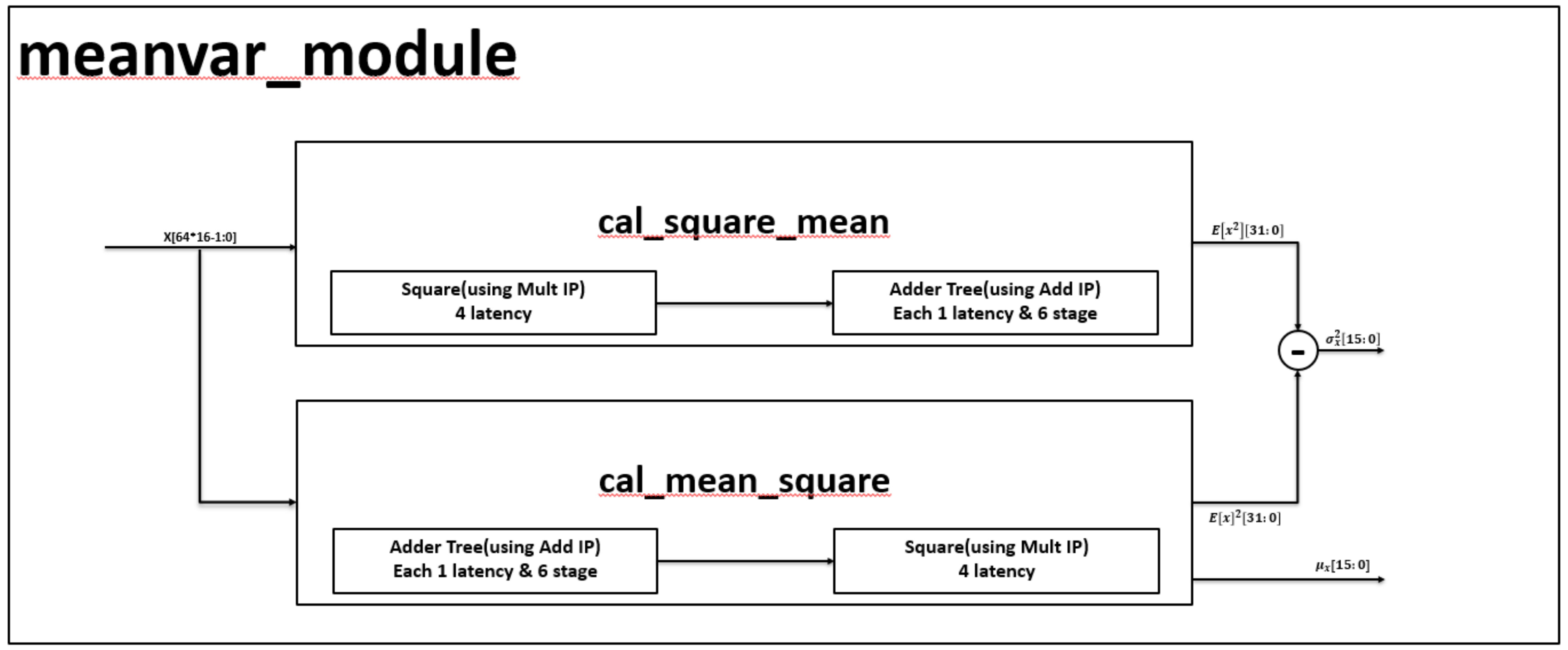
Meanvar_module

Input data : $E[x^2][31:0]$ / $E[x]^2[31:0]$

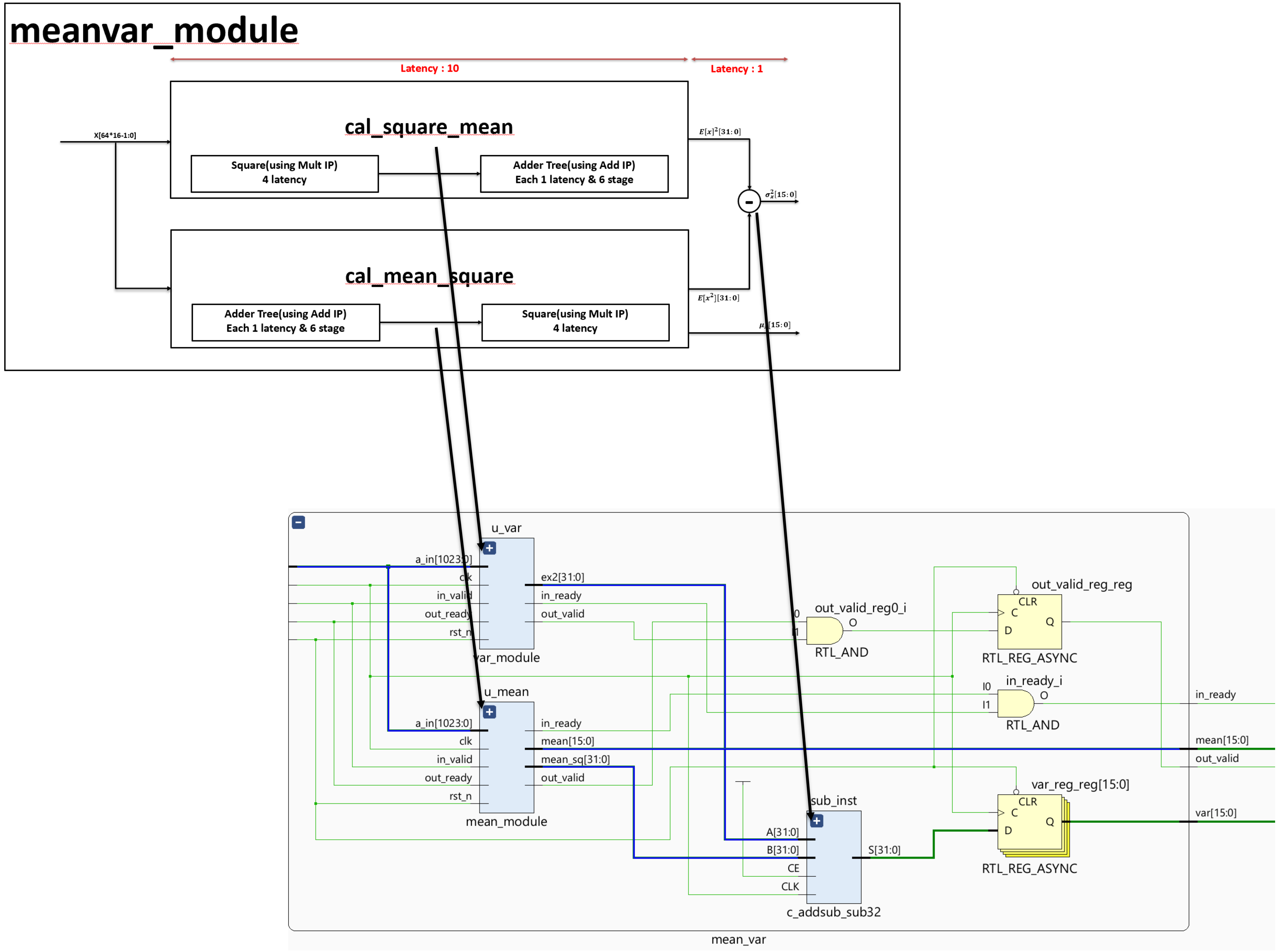
Output data : $\sigma_x^2[15:0]$

```
c_addsub_sub32 sub_inst (
    .A(ex2),
    .B(mean_sq),
    .CLK(clk),
    .CE(1'b1),
    .S(var_q16)
);
```

$$\rightarrow \sigma_x^2[15:0]$$


meanvar_module
cal_square_mean
X[64*16-1:0]
Square(using Mult IP) 4 latency
Adder Tree(using Add IP) Each 1 latency & 6 stage
$E[x^2][31:0]$
$\sigma_x^2[15:0]$
cal_mean_square
Adder Tree(using Add IP) Each 1 latency & 6 stage
Square(using Mult IP) 4 latency
$E[x]^2[31:0]$
$\mu_x[15:0]$

Revised LN Approximation Model



Compare LN Approximation Model

Pairwise_module

Module name	Bit	IPs used	Latency
addsub_add(sub)	16bit	2 * 63 = **126**	**1**
addsub_add(sub)	32bit	2 * 63 = 126	**1**
mult_gen_mult	32bit	63= 63	**4**

Suppose IP Catalog when implement Pairwise

63EA : 32 + 16 + 8 + 4 + 2 + 1 (Pairwise Structure)

Pairwise_module

Total Latency : 15 Latency

Mean_Var_module

Module name	Bit	IPs used	Latency
addsub_add(sub)	16bit	63(Tree) = **63**	**1**
addsub_add(sub)	32bit	63(Tree) + 1(sub) = **64**	**1**
mult_gen_mult	32bit	64(x^2) + 1(μ^2) = **65**	**4**

Using IP Catalog when implement Mean_Var

63EA : 32 + 16 + 8 + 4 + 2 + 1 (Adder Tree)

Mean_Var_module

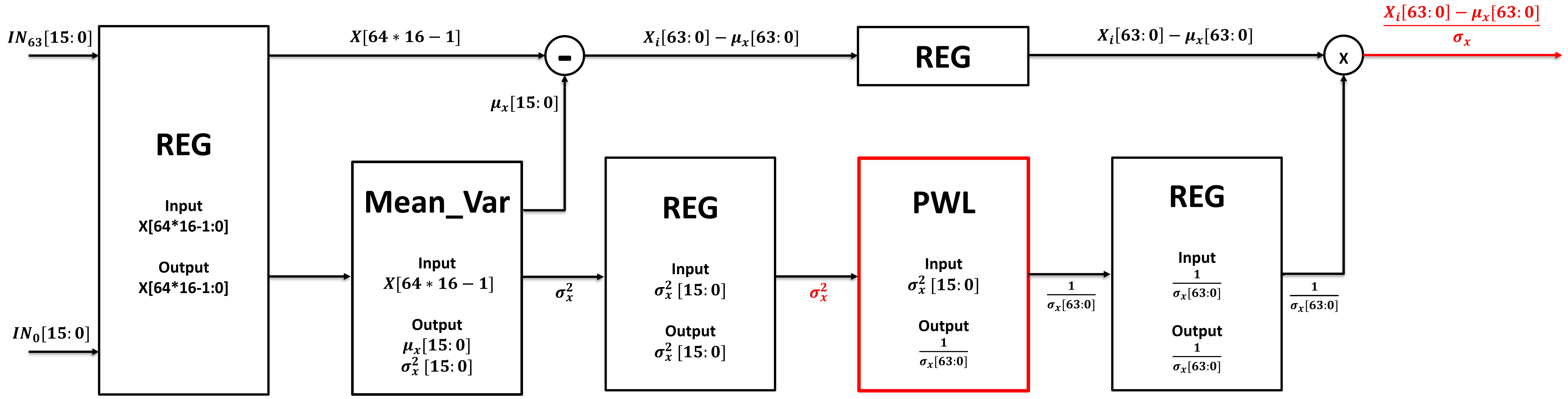
each stage : 10(6+4) + 1 Latency
Total stage : 64 → 1

Total Latency : 11 Latency

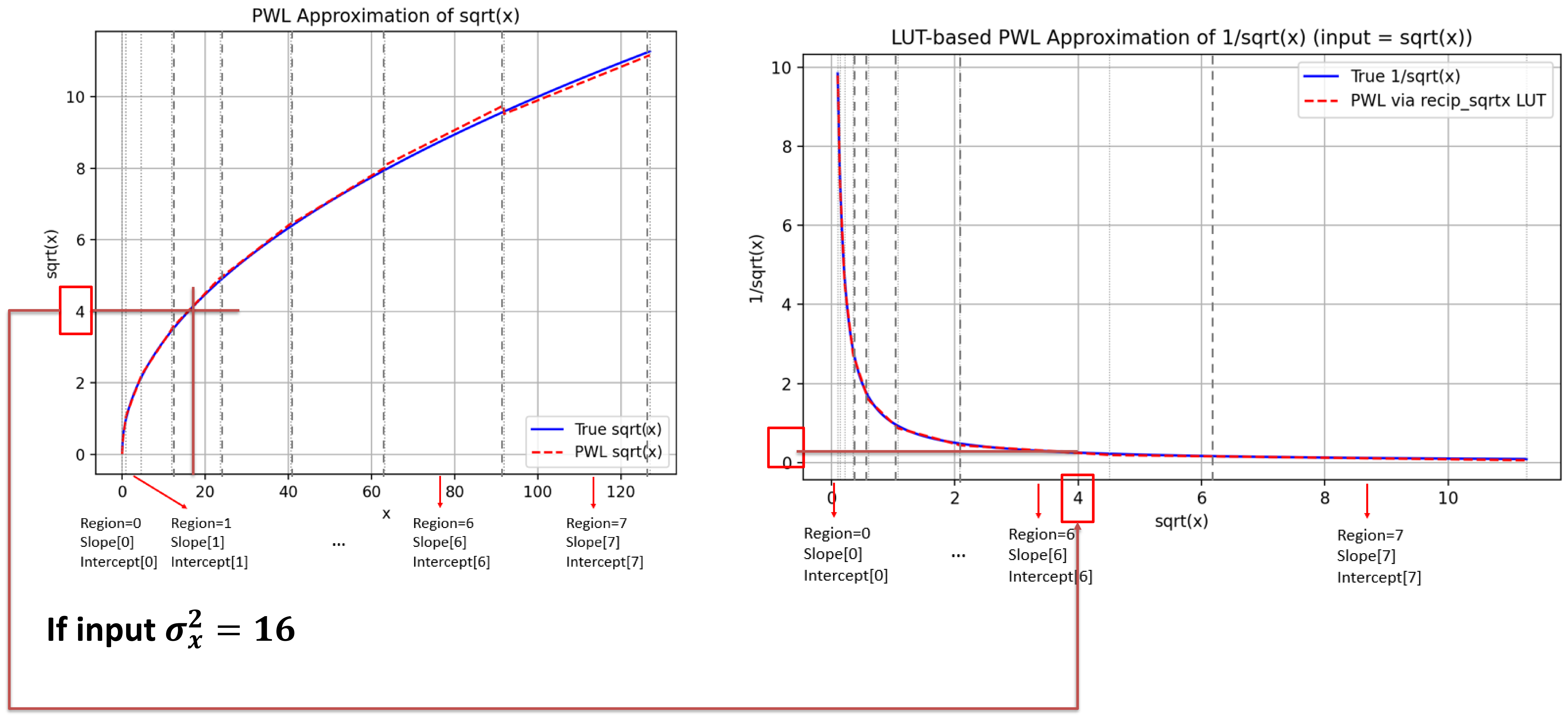
*** Same Latency when calculate $E[x^2][31:0], E[x]^2[31:0]$**

$$\begin{tikzcd}
\boxed{\text{Square / 4 latency}} \arrow[r] & \boxed{\text{Adder Tree / 6 latency}} \\
\boxed{\text{Adder Tree / 6 latency}} \arrow[r] & \boxed{\text{Square / 4 latency}}
\end{tikzcd}$$

LN Approximation Model



LN Approximation Model

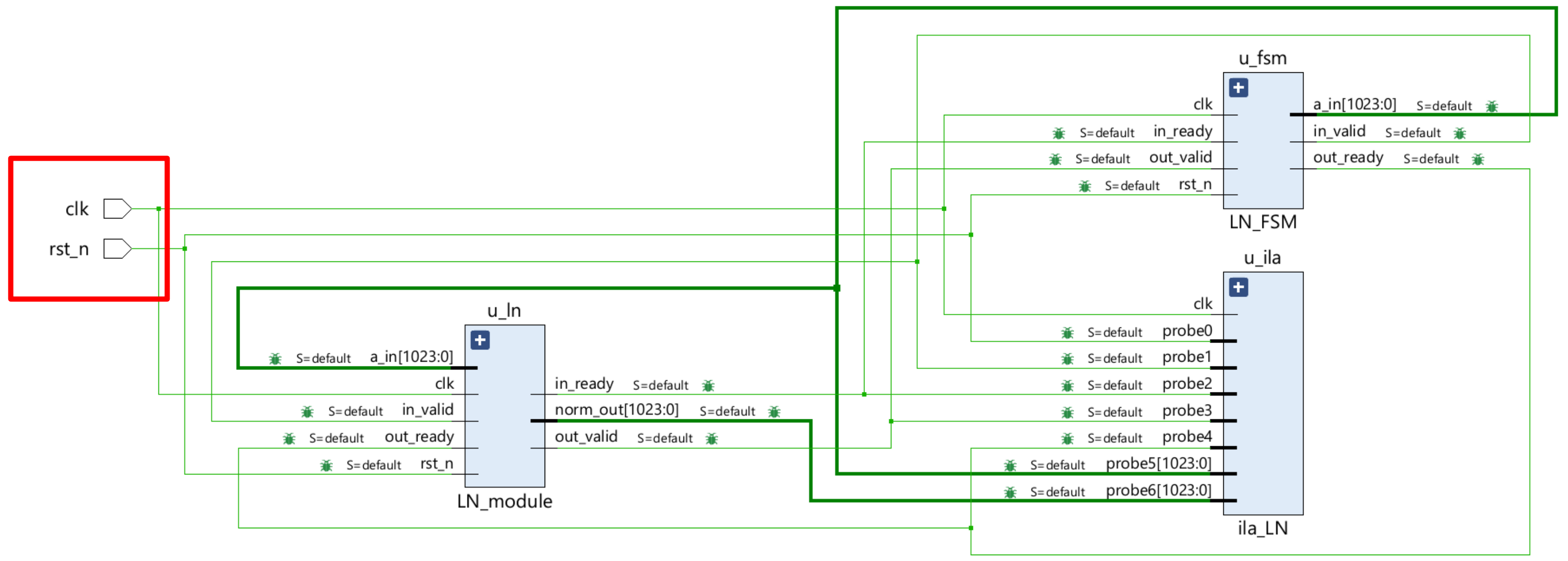


If input $\sigma_x^2 = 16$

Apply FPGA

Top_module : include LN_module, FSM, ILA.

LN_module : Performs normalization on input data
FSM : Used to verify correct operation on FPGA before **applying BRAM**
ILA : Used to observe signal values and **confirm** correct **operation** on **FPGA**



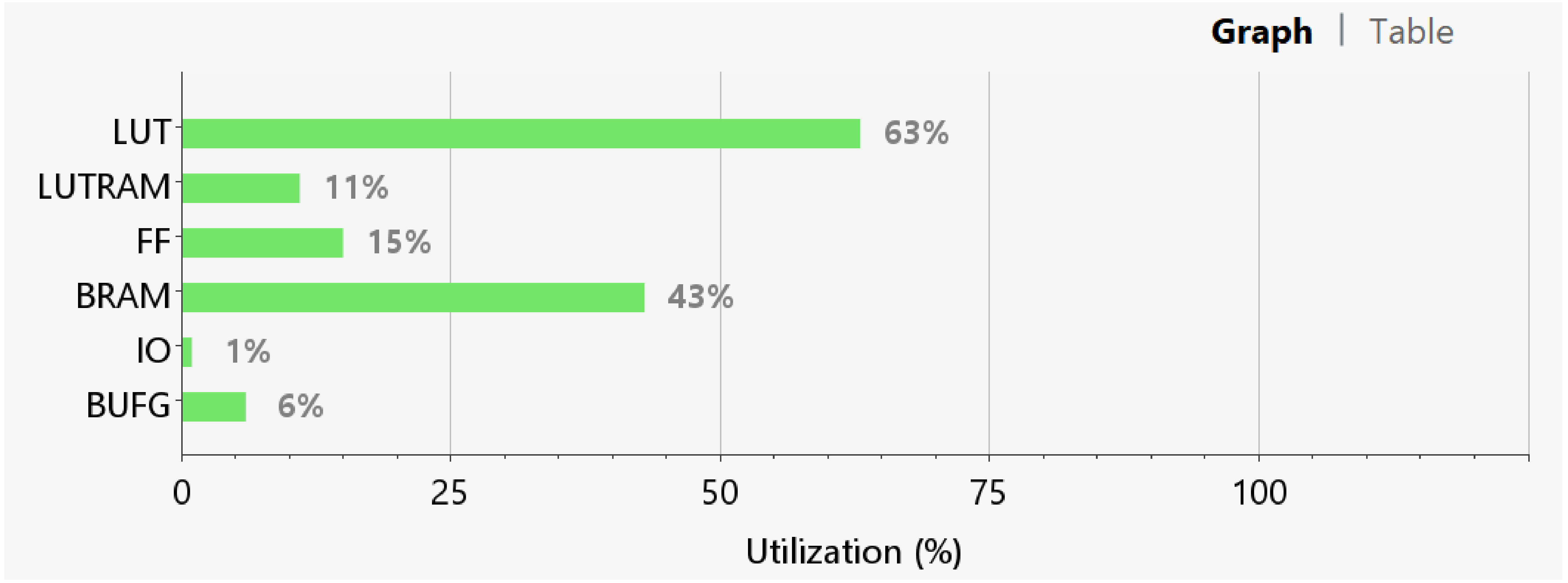
Using clk, rst_n pin -> set up .xdc file

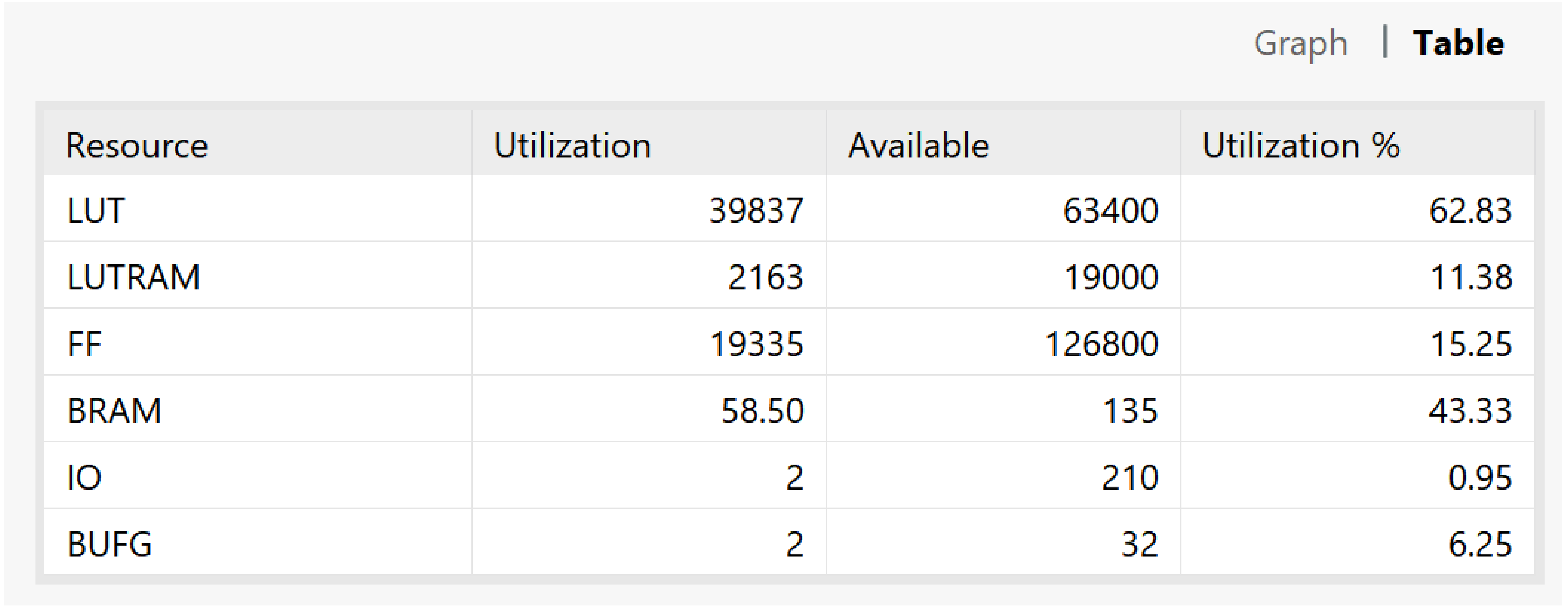
```
## Clock: 50MHz
set_property PACKAGE_PIN E3 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -period 20.000 -name sys_clk -waveform {0 5} [get_ports clk]

## Reset
set_property PACKAGE_PIN D4 [get_ports rst_n]
set_property IOSTANDARD LVCMOS33 [get_ports rst_n]
```

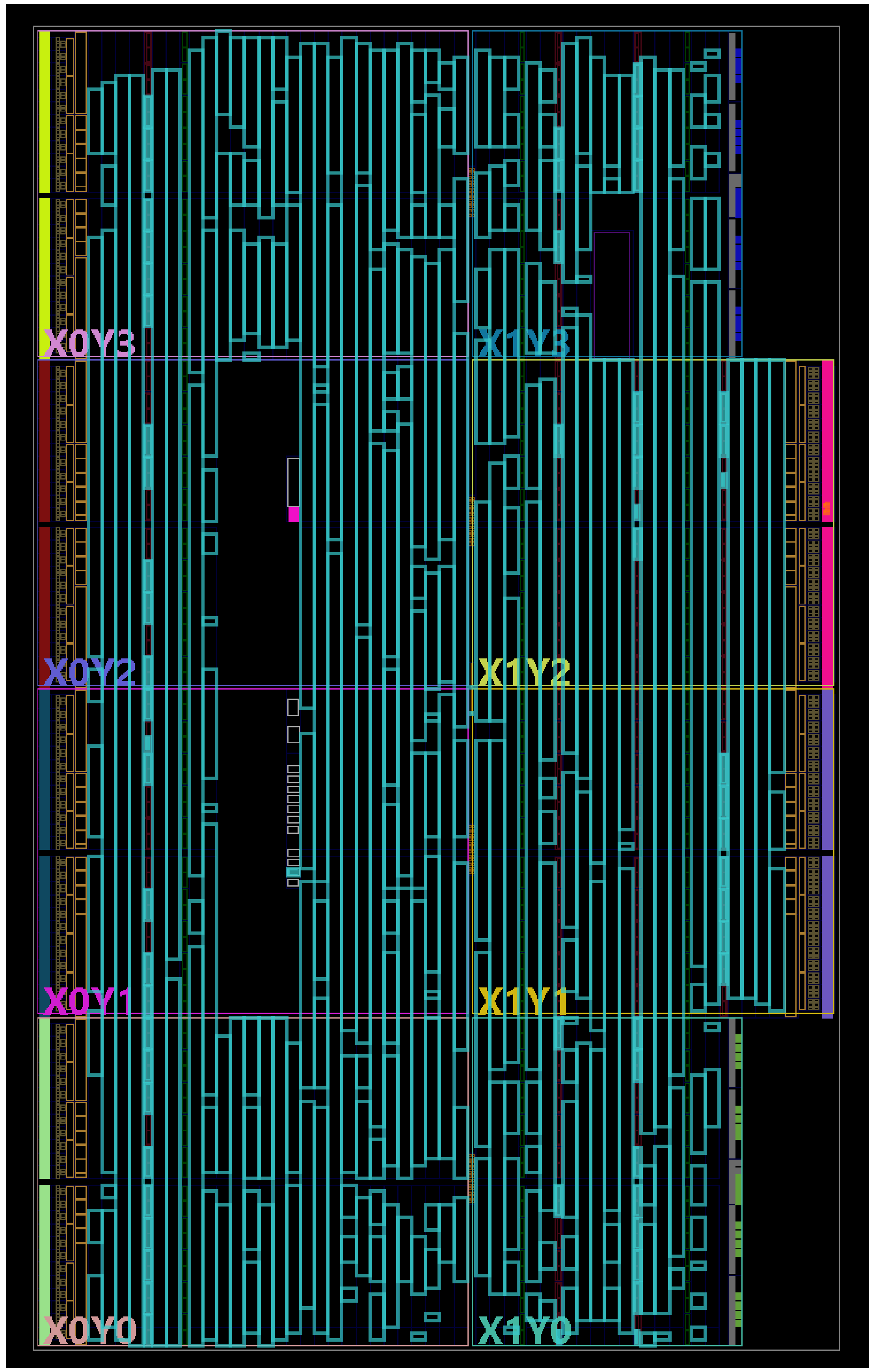
Frequency : **50MHz**(Period : 20.0ns)
duty ratio : 50%

Apply FPGA(Implementation)



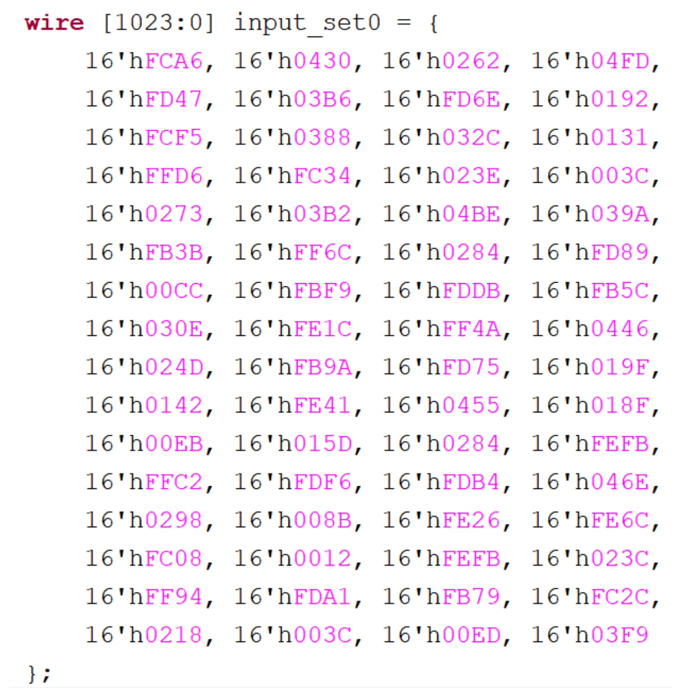

Graph | Table

Resource	Utilization	Available	Utilization %
LUT	39837	63400	62.83
LUTRAM	2163	19000	11.38
FF	19335	126800	15.25
BRAM	58.50	135	43.33
IO	2	210	0.95
BUFG	2	32	6.25



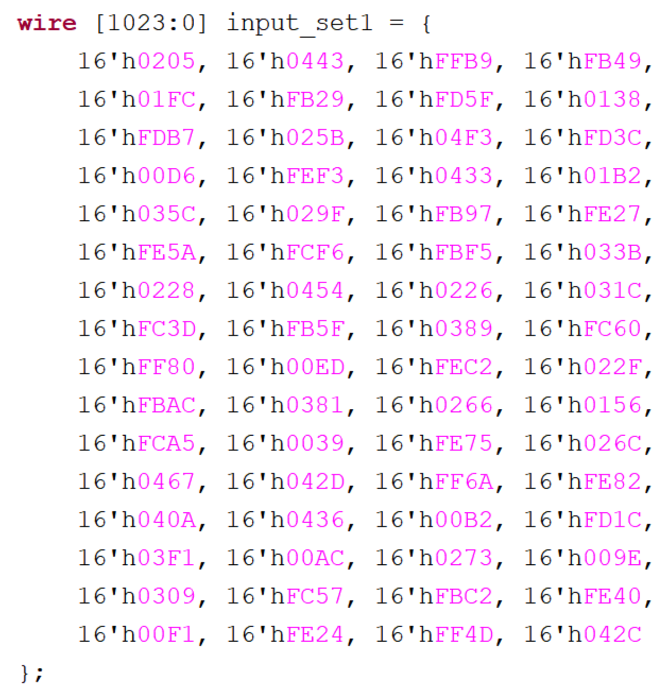
Verified FPGA operate

Input data : random data (-5 ~ 5) 2set (each set has 64)



```
wire [1023:0] input_set0 = {
    16'hFCA6, 16'h0430, 16'h0262, 16'h04FD,
    16'hFD47, 16'h03B6, 16'hFD6E, 16'h0192,
    16'hFCF5, 16'h0388, 16'h032C, 16'h0131,
    16'hFFD6, 16'hFC34, 16'h023E, 16'h003C,
    16'h0273, 16'h03B2, 16'h04BE, 16'h039A,
    16'hFB3B, 16'hFF6C, 16'h0284, 16'hFD89,
    16'h00CC, 16'hFBF9, 16'hFDDB, 16'hFB5C,
    16'h030E, 16'hFE1C, 16'hFF4A, 16'h0446,
    16'h024D, 16'hFB9A, 16'hFD75, 16'h019F,
    16'h0142, 16'hFE41, 16'h0455, 16'h018F,
    16'h00EB, 16'h015D, 16'h0284, 16'hFEFB,
    16'hFFC2, 16'hFDF6, 16'hFDB4, 16'h046E,
    16'h0298, 16'h008B, 16'hFE26, 16'hFE6C,
    16'hFC08, 16'h0012, 16'hFEFB, 16'h023C,
    16'hFF94, 16'hFDA1, 16'hFB79, 16'hFC2C,
    16'h0218, 16'h003C, 16'h00ED, 16'h03F9
};
```

Actual mean	: 0.2278
Actual Variance	: 7.8088

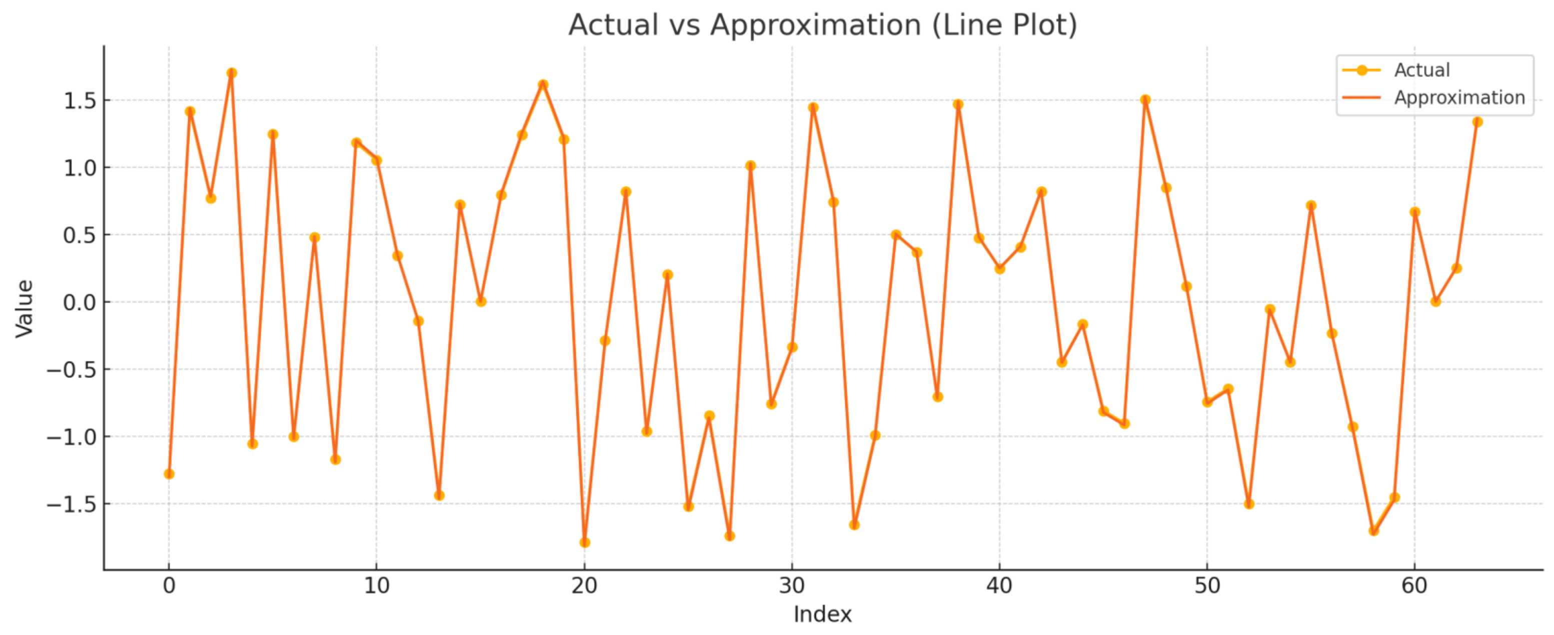


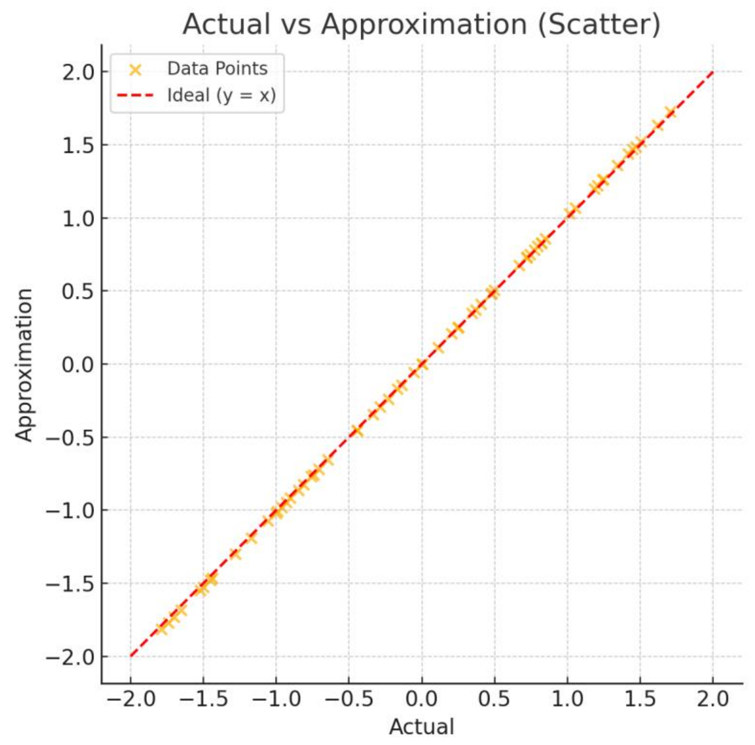
```
wire [1023:0] input_set1 = {
    16'h0205, 16'h0443, 16'hFFB9, 16'hFB49,
    16'h01FC, 16'hFB29, 16'hFD5F, 16'h0138,
    16'hFDB7, 16'h025B, 16'h04F3, 16'hFD3C,
    16'h00D6, 16'hFEF3, 16'h0433, 16'h01B2,
    16'h035C, 16'h029F, 16'hFB97, 16'hFE27,
    16'hFE5A, 16'hFCF6, 16'hFBF5, 16'h033B,
    16'h0228, 16'h0454, 16'h0226, 16'h031C,
    16'hFC3D, 16'hFB5F, 16'h0389, 16'hFC60,
    16'hFF80, 16'h00ED, 16'hFEC2, 16'h022F,
    16'hFBAC, 16'h0381, 16'h0266, 16'h0156,
    16'hFCA5, 16'h0039, 16'hFE75, 16'h026C,
    16'h0467, 16'h042D, 16'hFF6A, 16'hFE82,
    16'h040A, 16'h0436, 16'h00B2, 16'hFD1C,
    16'h03F1, 16'h00AC, 16'h0273, 16'h009E,
    16'h0309, 16'hFC57, 16'hFBC2, 16'hFE40,
    16'h00F1, 16'hFE24, 16'hFF4D, 16'h042C
};
```

Actual mean	: 0.3224
Actual Variance	: 8.8496

Verified FPGA operate

Input data : random data (-5 ~ 5) 2set (each set has 64)
Output data : normalized input data
Set1 → mean : 0.2278 / Variance : 7.8088



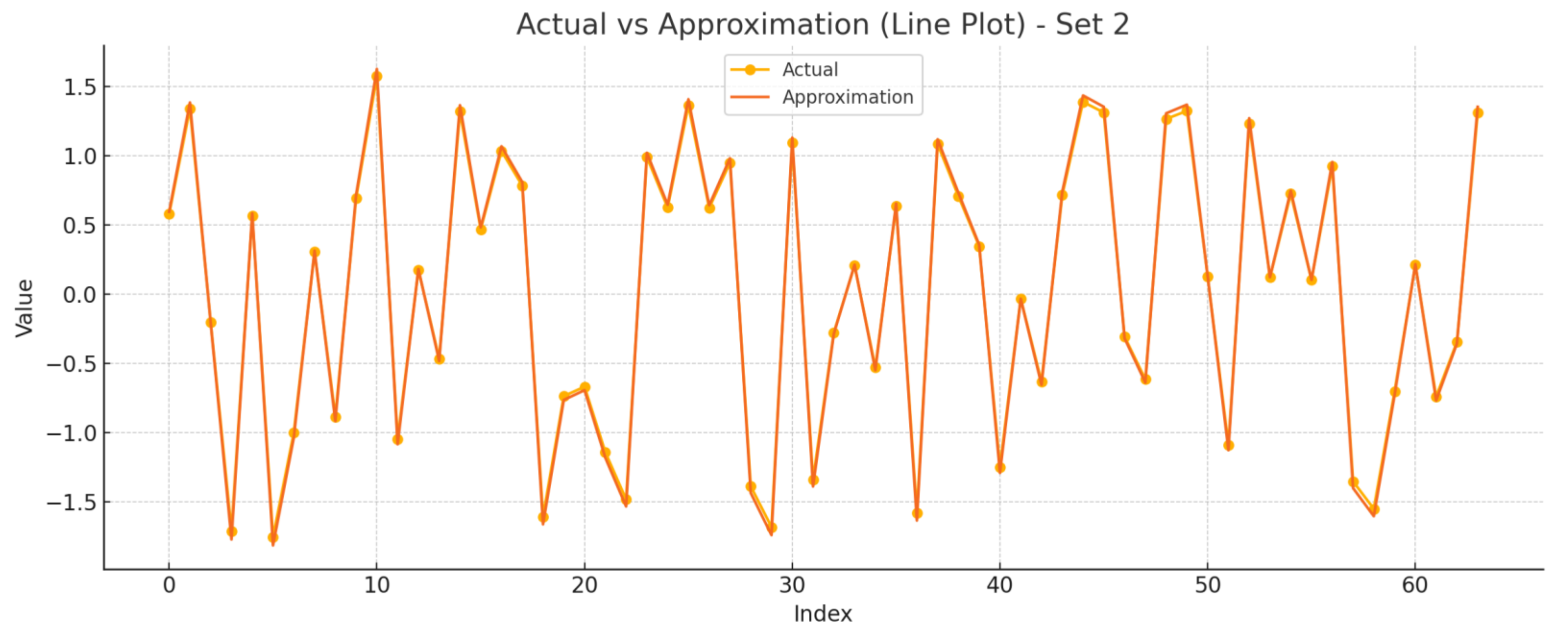


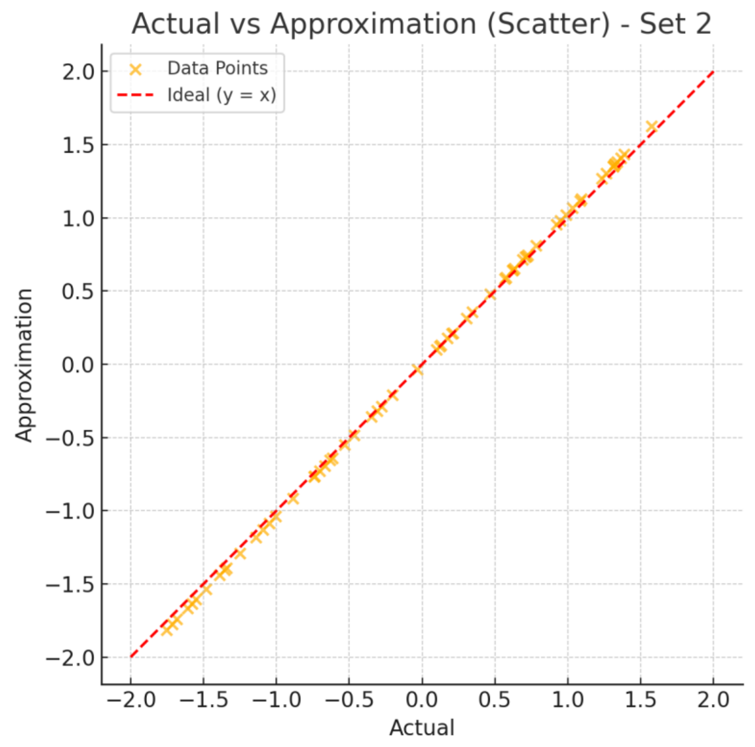
Accuracy(95% ↑)	: 62 / 64
Average Accuaracy	: 98.47%

Accuracy(95% ↓)		
Actual	Approximation	Accuracy
0.0023	0	0%
0.0023	0	0%

Verified FPGA operate

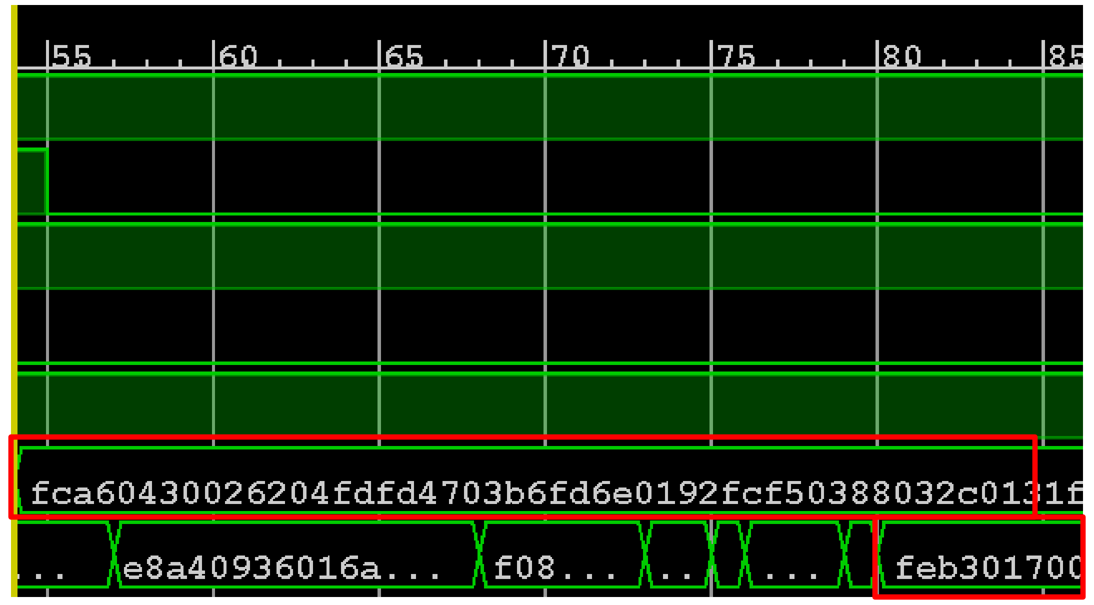
Input data : random data (-5 ~ 5) 2set (each set has 64)
Output data : normalized input data
Set2 → mean : 0.3224 / Variance : 8.8496





Accuracy(95% ↑)	: 64 / 64
Average Accuaracy	: 96.63%

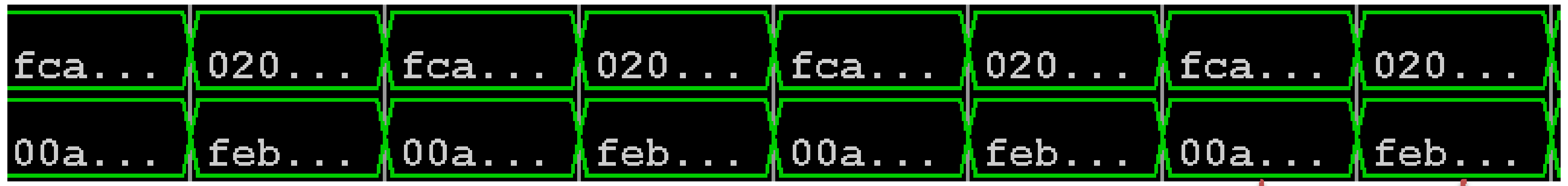
Verified FPGA operate



Total Latency : 26 Latency

Each module Latency

mean_var	: $10(mean \,/\, mean^2) + 1(var)$	= 11 Latency
PWL	: $root(4(a_i) + 1(b_i)) + recip(5)$	= 10 Latency
MULT	: 4	= 4 Latency
SUB	: 1	= 1 Latency



feb3017000c801bafeed0144fefb007cfecf013301110059ffdbfe8900b

00a60177ffd1fe3400a3fe28fef6005cff1600c501b7fee90038ff890171008

The input data set is applied continuously **on each clock cycle**.

Conclusion

1. LN is a major bottleneck in Transformer models due to high computation and latency.
2. Approximation models were applied to key operations such as **root and devide**.
3. Instead of accumulator, **adder tree structure** and **equations($\boldsymbol{E\left[x^2\right][31:0] - E[x]^2[31:0]}$)** were applied.
4. Used **Q8.8 format** instead of IEEE float, to enable lightweight and acceptable approximations.

Result

→ Total latency to **26 cycles**

→ **Accuracy** maintained **above 95%**