DSP Architecture Design
Midterm Project Presentation – Float16 Arithmetic Computation Unit Circuit Design

Speaker：Sheng-Wei Huang

國立臺灣師範大學
NATIONAL TAIWAN NORMAL UNIVERSITY

Outline

- Background – Float16 Datatype
- Circuit Design
  - Float16 Adder
  - Log-scale Multiplier
  - Log-scale Divider

Background – Float16 Datatype

- Float16 datatype has been widely used in AI computations in recent years.
- Pros: Using fewer bits compared to the Float32 datatype, reduce resource consumption in hardware design
- Cons: Representable numerical range and precision are significantly lower, reduce computational precision compared to Float32
- Commonly used datatypes in hardware design for floating-point computation:

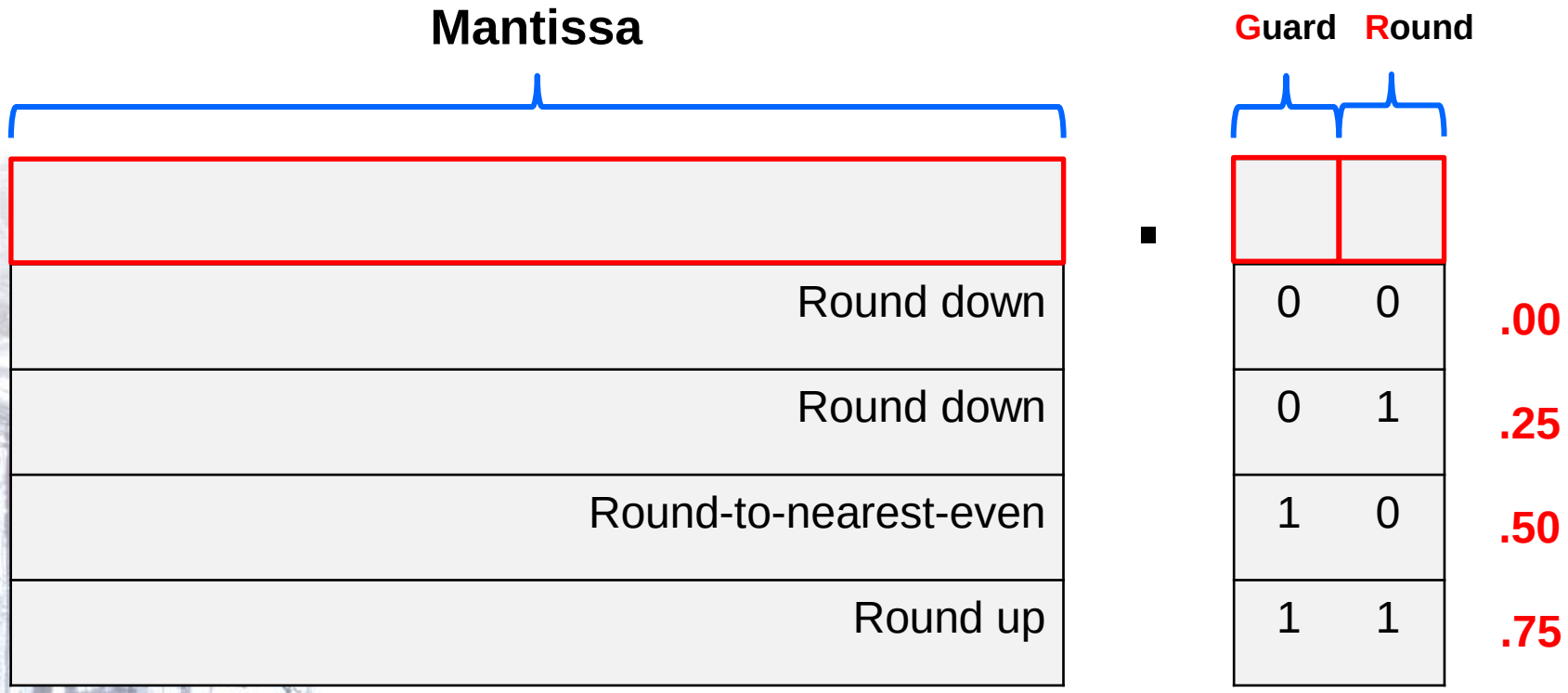
	Sign bit	Exponent bits	Mantissa bits
BFloat16	1	8	7
Float16	1	5	10
Float32	1	8	23

Float16 Adder

- Float16 Addition Algorithm
  - Separate input to sign, exponent, mantissa
  - Shift smaller number right to align the bigger one
  - Mantissa addition and normalization
  - Round-to-Nearest-Even rounding
  - Output result

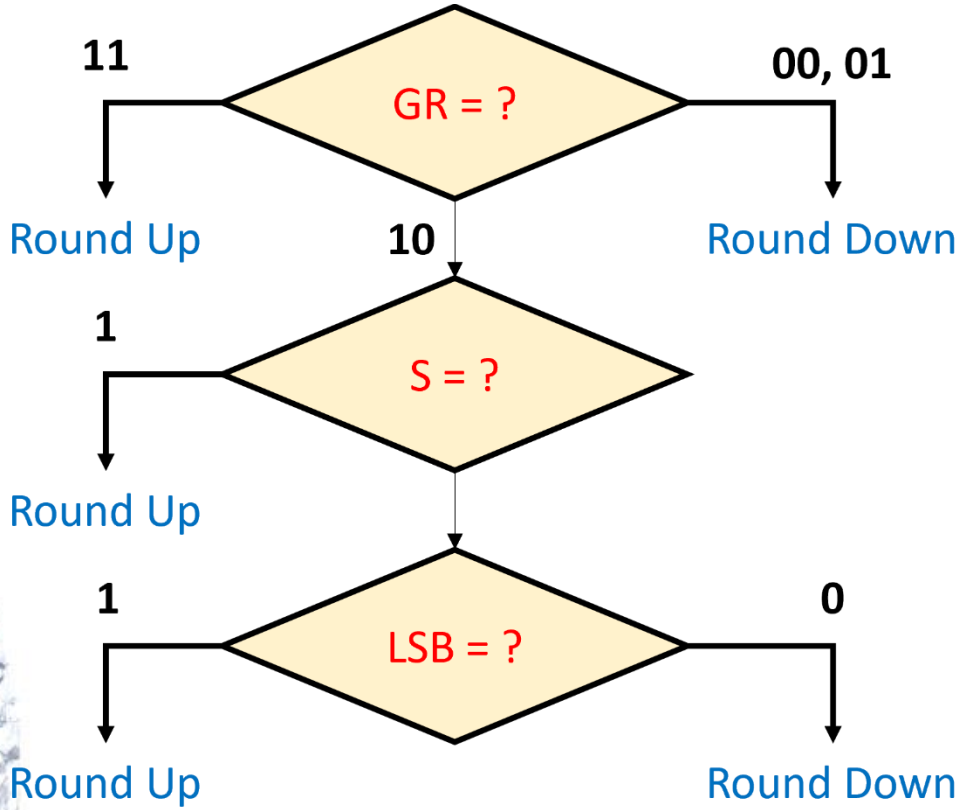
Float16 Adder

- Round-to-Nearest-Even rounding

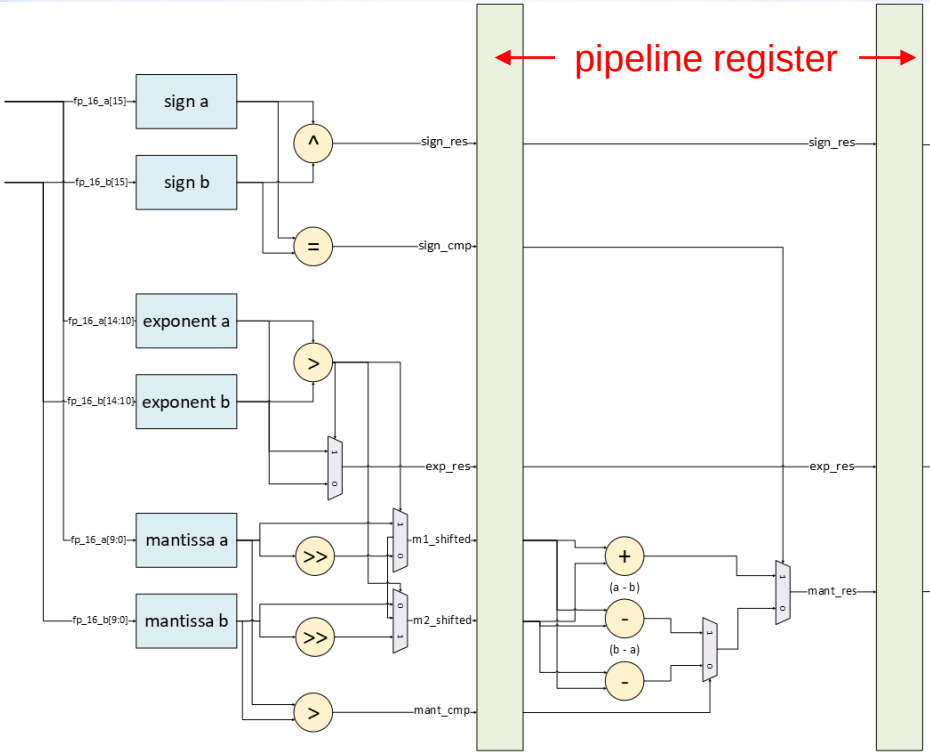


Float16 Adder

- Round-to-Nearest-Even rounding

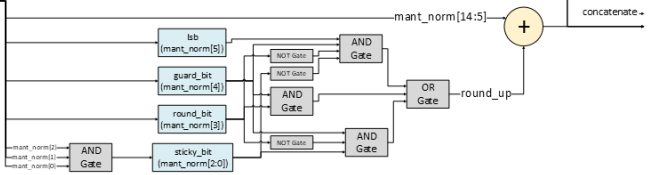


Float16 Adder



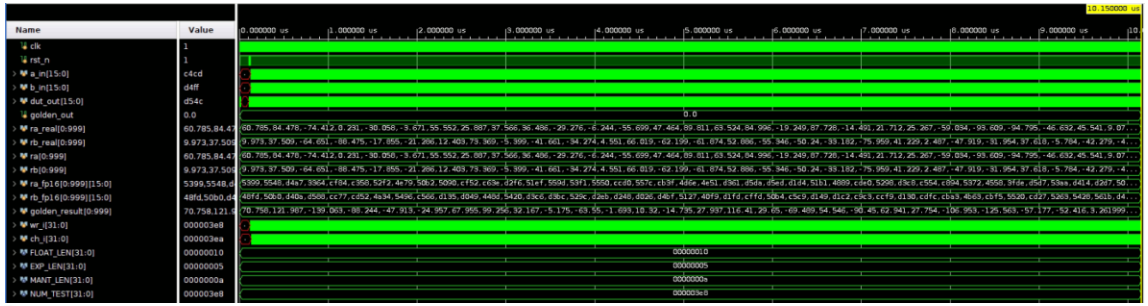

```
// For Round to nearest, ties to even
always_comb begin
    casez (mant_res_r)
        17'b1?????????????????: mant_norm = mant_res_r[15:1];
        17'b01????????????????: mant_norm = mant_res_r[14:0];
        17'b001???????????????: mant_norm = mant_res_r[13:0] << 1;
        17'b0001??????????????: mant_norm = mant_res_r[12:0] << 2;
        17'b00001?????????????: mant_norm = mant_res_r[11:0] << 3;
        17'b000001????????????: mant_norm = mant_res_r[10:0] << 4;
        17'b0000001???????????: mant_norm = mant_res_r[9:0] << 5;
        17'b00000001??????????: mant_norm = mant_res_r[8:0] << 6;
        17'b000000001?????????: mant_norm = mant_res_r[7:0] << 7;
        17'b0000000001????????: mant_norm = mant_res_r[6:0] << 8;
        17'b00000000001???????: mant_norm = mant_res_r[5:0] << 9;
        17'b000000000001??????: mant_norm = mant_res_r[4:0] << 10;
        17'b0000000000001?????: mant_norm = mant_res_r[3:0] << 11;
        17'b00000000000001????: mant_norm = mant_res_r[2:0] << 12;
        17'b000000000000001???: mant_norm = mant_res_r[1:0] << 13;
        17'b0000000000000001??: mant_norm = mant_res_r[0] << 14;
        default: mant_norm = 15'b0;
    endcase
end
```

Priority Encoder
(Leading-one detector)




```
always_comb begin
    casez (mant_res_r)
        17'b1?????????????????: exp_norm_tmp = exp_res_sec_r + 1;
        17'b01????????????????: exp_norm_tmp = exp_res_sec_r;
        17'b001???????????????: exp_norm_tmp = exp_res_sec_r - 1;
        17'b0001??????????????: exp_norm_tmp = exp_res_sec_r - 2;
        17'b00001?????????????: exp_norm_tmp = exp_res_sec_r - 3;
        17'b000001????????????: exp_norm_tmp = exp_res_sec_r - 4;
        17'b0000001???????????: exp_norm_tmp = exp_res_sec_r - 5;
        17'b00000001??????????: exp_norm_tmp = exp_res_sec_r - 6;
        17'b000000001?????????: exp_norm_tmp = exp_res_sec_r - 7;
        17'b0000000001????????: exp_norm_tmp = exp_res_sec_r - 8;
        17'b00000000001???????: exp_norm_tmp = exp_res_sec_r - 9;
        17'b000000000001??????: exp_norm_tmp = exp_res_sec_r - 10;
        17'b0000000000001?????: exp_norm_tmp = exp_res_sec_r - 11;
        17'b00000000000001????: exp_norm_tmp = exp_res_sec_r - 12;
        17'b000000000000001???: exp_norm_tmp = exp_res_sec_r - 13;
        17'b0000000000000001??: exp_norm_tmp = exp_res_sec_r - 14;
        default: exp_norm_tmp = 5'b0;
    endcase
end
```

Float16 Adder



```
PASS at 995
  a       = -91.542000 (0x51f2)
  b       = 35.476000 (0x5619)
  DUT     = 0xd303 (-56.093750)
  GOLDEN = -56.066000
  ERROR = 0.027750
PASS at 996
  a       = 10.002000 (0x5145)
  b       = -79.655000 (0x54b7)
  DUT     = 0xd45a (-69.625000)
  GOLDEN = -69.653000
  ERROR = 0.028000
PASS at 997
  a       = 47.547000 (0xc4cd)
  b       = 97.566000 (0xd4ff)
  DUT     = 0x5889 (145.125000)
  GOLDEN = 145.113000
  ERROR = 0.012000
PASS at 998
  a       = 42.157000 (0xc4cd)
  b       = 75.446000 (0xd4ff)
  DUT     = 0x575a (117.625000)
  GOLDEN = 117.603000
  ERROR = 0.022000
PASS at 999
  a       = -4.800000 (0xc4cd)
  b       = -79.957000 (0xd4ff)
  DUT     = 0xd54c (-84.750000)
  GOLDEN = -84.757000
  ERROR = 0.007000

 Test Finished. Total: 1000. Failures: 0
```

Resource	Estimation	Available	Utilization %
LUT	287	871680	0.03
FF	103	1743360	0.01
IO	50	416	12.02
BUFG	1	672	0.15

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.948 ns	Worst Hold Slack (WHS):	-0.057 ns	Worst Pulse Width Slack (WPWS):	4.725 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	-2.204 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	43	Number of Failing Endpoints:	0
Total Number of Endpoints:	43	Total Number of Endpoints:	43	Total Number of Endpoints:	104

Timing constraints are not met.

Clock period = 10.000ns
Clock rate = 100MHz
Setup Time = 10ns – 8.948ns = 1.052ns

ERROR =
DUT Answer – Golden Answer < 0.2

Log-scale Multiplier

- Since Float16 has fewer mantissa bits compared to Float32, it offers less precision. When using traditional floating-point multiplication, this limited precision restricts the margin for correction during the normalization stage.
- Transforming the multiplication operation into the logarithmic domain：

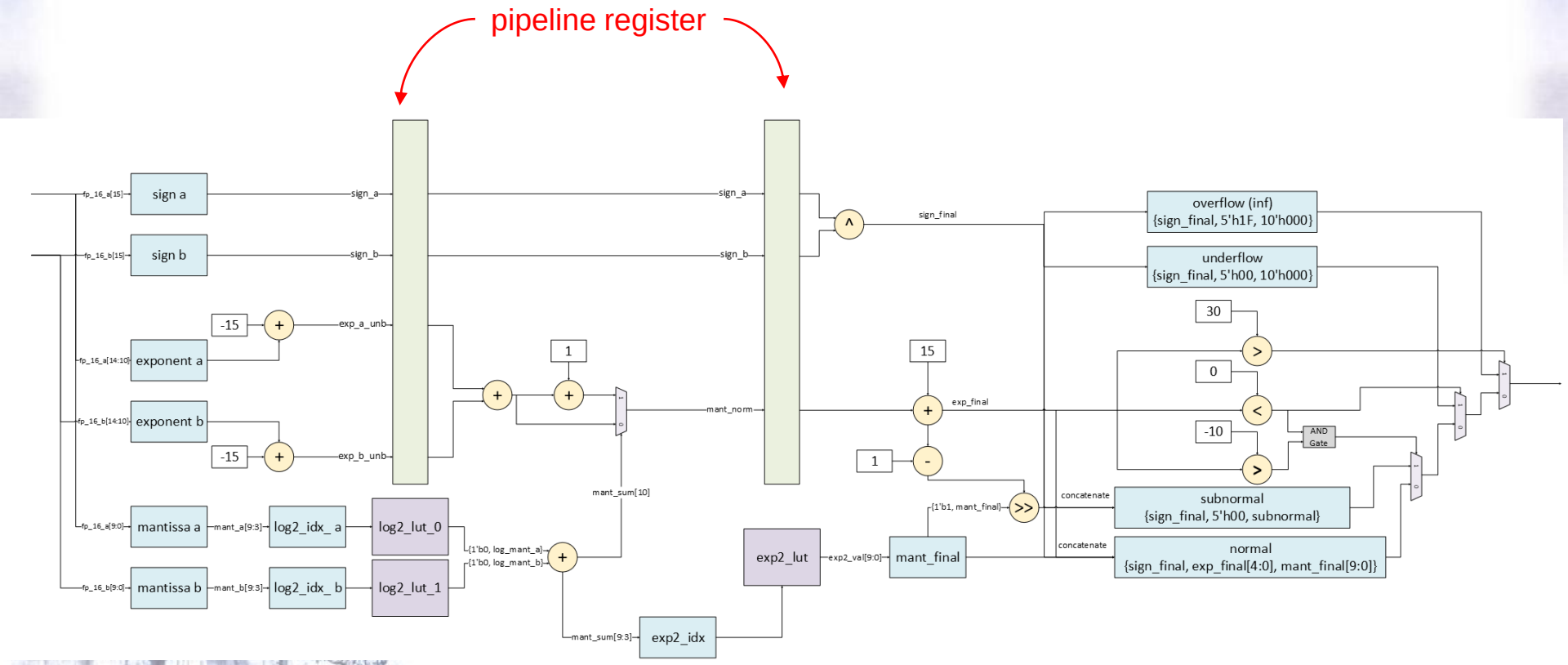
$$a \times b = 2^{\log_2(a \times b)} = 2^{\log_2 a + \log_2 b}$$

- Use Look Up Table(LUT) to replace log2 & exp2 computation, log2 LUT size = (10 bits) * 128, exp2 LUT size = (16 bits) * 128

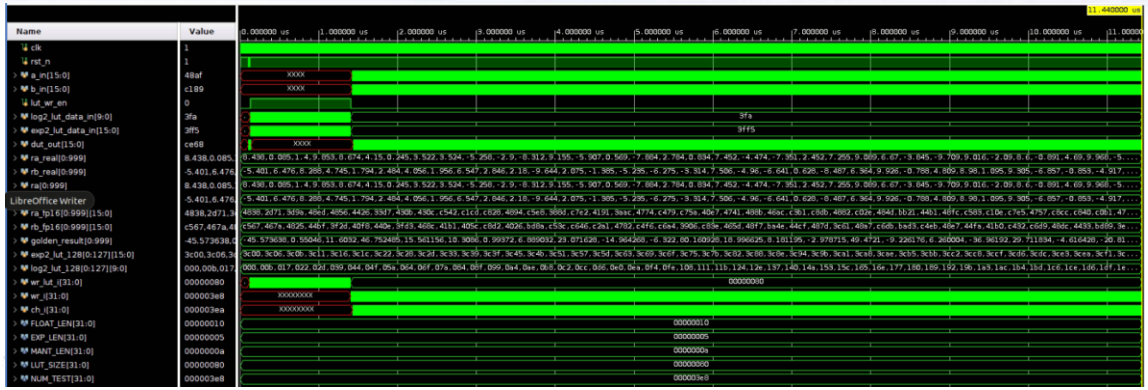
Log-scale Multiplier

- Log-scale Multiplication Algorithm
  - Load log2 & exp2 look up table
  - Separate input to sign, exponent, mantissa, get the unbiased exponent and get the index to look up log2 table
  - Look up log2's mantissa
  - Exponent & Mantissa addition and normalization
  - Get the index to look up exp2 table
  - Compute sign, exponent, mantissa and concatenate them

Log-scale Multiplier



Log-scale Multiplier



```
PASS at 996
  a       = -4.038000 (0x3266)
  b       = -2.888000 (0x4838)
  DUT     = 0x49c8 (11.562500)
  GOLDEN = 11.661744
  ERROR = 0.099244
  ERROR RATE = 0.008510
PASS at 997
  a       = -5.201000 (0x48af)
  b       = 8.661000 (0xc189)
  DUT     = 0xd191 (-44.531250)
  GOLDEN = -45.045861
  ERROR = 0.514611
  ERROR RATE = 0.011424
PASS at 998
  a       = 0.200000 (0x48af)
  b       = 8.441000 (0xc189)
  DUT     = 0x3eba (1.681641)
  GOLDEN = 1.688200
  ERROR = 0.006559
  ERROR RATE = 0.003885
PASS at 999
  a       = 9.370000 (0x48af)
  b       = -2.768000 (0xc189)
  DUT     = 0xce68 (-25.625000)
  GOLDEN = -25.936160
  ERROR = 0.311160
  ERROR RATE = 0.011997

 Test Finished. Total: 1000, Failures: 0
```

Resource	Estimation	Available	Utilization %
LUT	172	871680	0.02
LUTRAM	80	403200	0.02
FF	72	1743360	0.01
IO	75	416	18.03
BUFG	1	672	0.15

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.545 ns	Worst Hold Slack (WHS):	-0.078 ns	Worst Pulse Width Slack (WPWS):	4.468 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	-30.214 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	502	Number of Failing Endpoints:	0
Total Number of Endpoints:	612	Total Number of Endpoints:	612	Total Number of Endpoints:	153

Timing constraints are not met.

Clock period = 10.000ns
Clock rate = 100MHz
Setup Time = 10ns – 8.545ns = 1.455ns

ERROR RATE=
abs(ERROR) – abs(Golden Answer) < 1.9%

Log-scale Divider

- Since Float16 has fewer mantissa bits compared to Float32, it offers less precision. When using traditional floating-point multiplication, this limited precision restricts the margin for correction during the normalization stage.
- Transforming the division operation into the logarithmic domain：

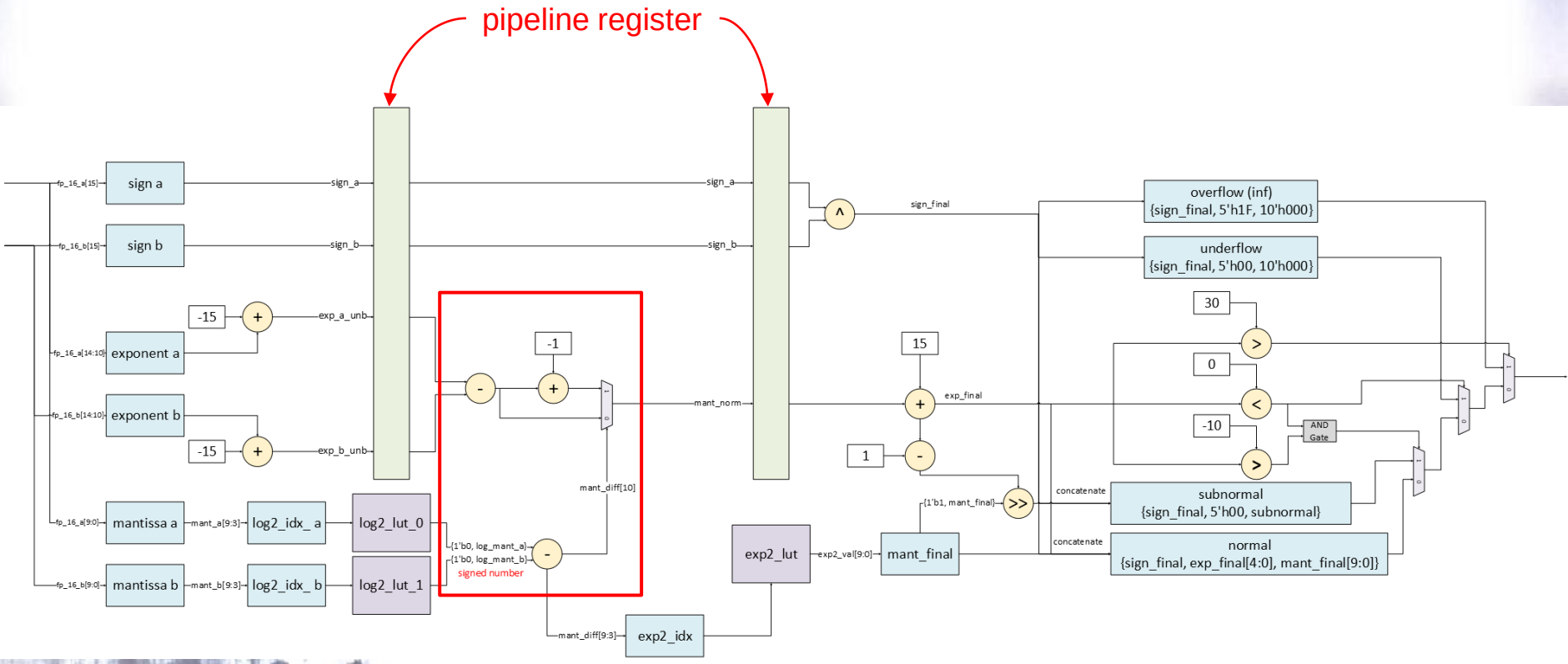
$$a/b = 2^{\log_2(a/b)} = 2^{\log_2 a - \log_2 b}$$

- Use Look Up Table(LUT) to replace log2 & exp2 computation, log2 LUT size = (10 bits) * 128, exp2 LUT size = (16 bits) * 128

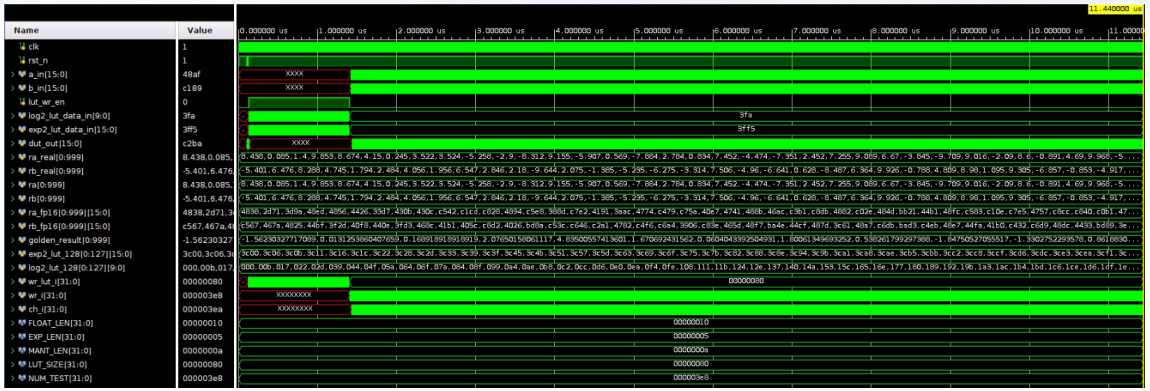
Log-scale Multiplier

- Log-scale Multiplication Algorithm
  - Load log2 & exp2 look up table
  - Separate input to sign, exponent, mantissa, get the unbiased exponent and get the index to look up log2 table
  - Look up log2's mantissa
  - Exponent & Mantissa subtraction and normalization
  - Get the index to look up exp2 table
  - Compute sign, exponent, mantissa and concatenate them

Log-scale Multiplier



Log-scale Multiplier



```
PASS at 996
  a       = -4.038000 (0x3266)
  b       = -2.888000 (0x4838)
  DUT     = 0x3d99 (1.399414)
  GOLDEN  = 1.398199
  ERROR   = 0.001215
  ERROR RATE = 0.000869
PASS at 997
  a       = -5.201000 (0x48af)
  b       = 8.661000 (0xc189)
  DUT     = 0xb8cf (-0.601074)
  GOLDEN  = -0.600508
  ERROR   = 0.000566
  ERROR RATE = 0.000943
PASS at 998
  a       = 0.200000 (0x48af)
  b       = 8.441000 (0xc189)
  DUT     = 0x2609 (0.023575)
  GOLDEN  = 0.023694
  ERROR   = 0.000119
  ERROR RATE = 0.005024
PASS at 999
  a       = 9.370000 (0x48af)
  b       = -2.768000 (0xc189)
  DUT     = 0xc2ba (-3.363281)
  GOLDEN  = -3.385116
  ERROR   = 0.021834
  ERROR RATE = 0.006450

 Test Finished. Total: 1000. Failures: 0
```

Resource	Estimation	Available	Utilization %
LUT	170	871680	0.02
LUTRAM	80	403200	0.02
FF	72	1743360	0.01
IO	75	416	18.03
BUFG	1	672	0.15

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.729 ns	Worst Hold Slack (WHS):	-0.099 ns	Worst Pulse Width Slack (WPWS):	4.468 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	-30.365 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	502	Number of Failing Endpoints:	0
Total Number of Endpoints:	612	Total Number of Endpoints:	612	Total Number of Endpoints:	153

Timing constraints are not met.

Clock period = 10.000ns
Clock rate = 100MHz
Setup Time = 10ns – 8.729ns = 1.271ns

ERROR RATE=
abs(ERROR) – abs(Golden Answer) < 1.1%