Tahmini Ders İçeriği
(Tentative Couse Schedule – Syllabus)

1. Hafta: Sayı sistemleri, onluk/ikilik taban sayı gösterimleri, mantıksal kapılar, computer system overview, başarım (performance)

2. Hafta: 2'lik tabanda işaretli sayılar, mikroişlemci tarihi, benchmarking, başarım,

3. Hafta: Başarım, Amdahl yasası, RISC-V development Environment, Verilog HDL ile Birleşik (Combinational) devreler

4. Hafta:, Verilog HDL ile sıralı (sequential) mantıksal devre ve sonlu durum makinası tasarımı, timing analysis

5. Hafta: Aritmetik devre tasarımları: Toplama, çıkarma, çarpma, bölme, trigonometri, square-root, hyperbolic, exponential, logarithm

6. Hafta: Fixed ve Floating-Point sayı gösterimleri

7. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

8. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

9. Hafta: Tek-çevrim işlemci tasarımı (single-cycle CPU)

10. Hafta: Çok-çevrim işlemci tasarımı (multi-cycle CPU)

11. Hafta: Boruhatlı işlemci tasarımı (pipelined CPU)

12. Hafta: Bellek sistemi ve hiyerarşisi

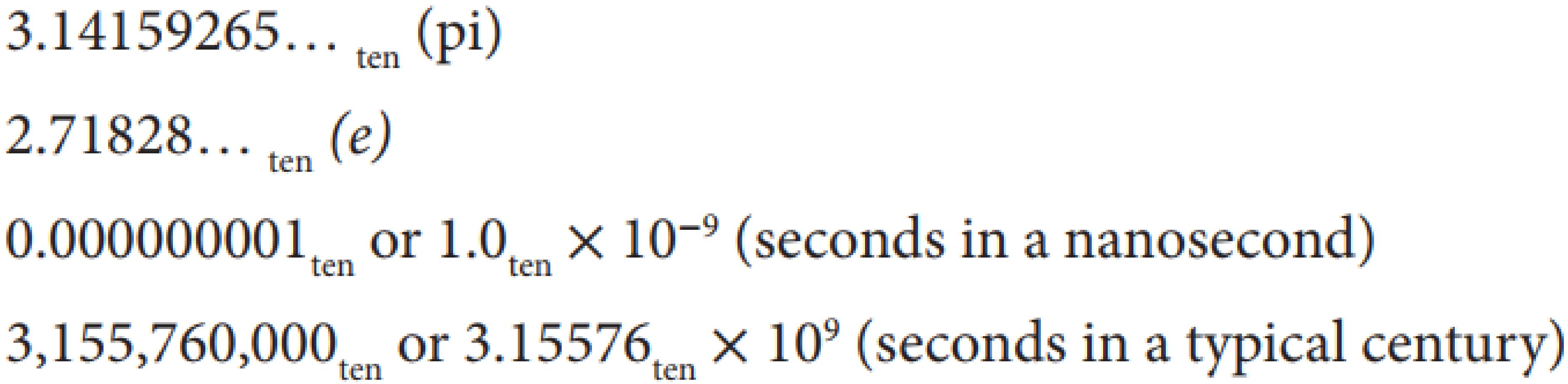
13. Hafta: İleri mimari konuları: Branch prediction, superscalar cpu, out-of-order execution, multi-core systems

14. Hafta: Gömülü sistemler, mikrodenetleyiciler, SoCs

Sabit Noktalı x Kayan Noktalı
(Fixed-Point vs Floating Point)

Doğal sayılar: Unsigned   Tam Sayılar: Signed (2’s complement for negative numbers)

Reel (real) sayılar: ???



$3.14159265\ldots_{\text{ten}}$ (pi)

$2.71828\ldots_{\text{ten}}$ (e)

0.000000001_{ten} or $1.0_{\text{ten}} \times 10^{-9}$ (seconds in a nanosecond)

$3{,}155{,}760{,}000_{\text{ten}}$ or $3.15576_{\text{ten}} \times 10^{9}$ (seconds in a typical century)

Taban Dönüşümü (Ondalıklı Sayılar)

$$(11010.11)_2 = (?)_{10}$$

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26.75$$

$$(0.6875)_{10} = (?)_2$$

Sayı	Çarpım	Elde	Bit Değer	Bit Sıra
0.6875	0.375	1	1	-1
0.375	0.75	0	0	-2
0.75	0.5	1	1	-3
0.5	0	1	1	-4

Ondalıklı kısım için sürekli 2 ile çarp, elde değeri bit değerini ifade eder. Bit sırası -1,-2,... şeklindedir. Çarpım ‘0’ olana kadar devam et.

$$(0.6875)_{10} = (0.1011)_2$$

Taban Dönüşümü (Base Conversion)

$$(0.6875)_{10} = (?)_2$$

Sayı	Çarpım	Elde	Bit Değer	Bit Sıra
0.6875	0.375	1	1	-1
0.375	0.75	0	0	-2
0.75	0.5	1	1	-3
0.5	0	0	1	-4

Ondalıklı kısım için sürekli 2 ile çarp, elde değeri bit değerini ifade eder. Bit sırası -1,-2,… şeklindedir. Çarpım ‘0’ olana kadar devam et.

$$(0.6875)_{10} = (0.1011)_2$$

0.1011 → 2 ile çarp: $\boxed{1}.\boxed{011}$

1.011 → 2 ile çarp: $\boxed{10}.\boxed{11}$

10.11 → 2 ile çarp: $\boxed{101}.\boxed{1}$

101.1 → 2 ile çarp: $\boxed{1011}$

<u>İkilik tabanda</u>

Sola kaydırma → 2 ile çarpma
Sağa kaydırma → 2’ye bölme

<u>Onluk tabanda</u>

Sola kaydırma → 10 ile çarpma
Sağa kaydırma → 10’a bölme

Taban Dönüşümü (Ondalıklı Sayılar)

$$(0.513)_{10} = (?)_2$$

Sayı	Çarpım	Elde	Bit Değer	Bit Sıra
0.513	0.026	1	1	-1
0.026	0.052	0	0	-2
0.052	0.104	0	0	-3
0.104	0.208	0	0	-4
0.208	0.416	0	0	-5
0.416	0.832	0	0	-6
0.832	0.664	1	1	-7
0.664	0.328	1	1	-8

Eğer ondalıklı kısım için 8-bit ayrıldıysa 8 kere iterasyon devam ettirilir.

$$(0.513)_{10} = (0.10000011)_2$$

$$\Downarrow$$

$$(0.10000011)_2 = (0.51171875)_{10}$$

Bilgi kaybı yaşandı!

Eğer ondalıklı kısım için sadece 4-bit ayrılmış olsaydı 0.513 sayısı ancak 0.5 olarak ifade edilebilecekti!

Sabit Noktalı x Kayan Noktalı (Fixed-Point vs Floating Point)

$$(326.513)_{10} = (?)_2$$

Sabit-Noktalı Gösterim: Tamsayı ve ondalıklı sayılar için sabit uzunlukta yer (bit) ayrılmıştır. Sayının tamsayı ya da ondalıklı sayı kısmının büyüklüğü, bu sabit uzunluğu değiştirmez. Örnek olarak tamsayı 10, ondalıklı 8 bit olursa:

$$(326.513)_{10} = (0101000110.10000011)_2$$

Kayan-Noktalı Gösterim: Tamsayı ve ondalıklı sayılar için sabit uzunlukta yer (bit) ayrıl**ma**mıştır. Sayının tamsayı ve ondalıklı kısımlarına göre noktanın yeri değişir, kayar. Bilgisayarlarda ondalıklı sayıları ifade edebilmek için IEEE-754 Single Precision ve Double Precision standartları oluşturulmuştur.

Single Precision : 32-bit
Double Precision : 32-bit

IEEE-754 standard ve fixed-point hakkında detaylı bilgi ileriki haftalarda anlatılacak

Signed Fixed-Point

$(s\ i_2\ i_1\ i_0\ .\ f_1\ f_2\ f_3\ f_4)_2$ Q4.4 : 4 integer bits (sign included), 4 fraction bits

Sign Bit (İşaret Biti): if s = 0 positive, if s = 1 negative

$$i_2 * 2^2 + i_1 * 2^1 + i_0 * 2^0 + f_1 * 2^{-1} + f_2 * 2^{-2} + f_3 * 2^{-3} + f_4 * 2^{-4}$$

2’s complement:

$$-s * 2^3 + \ i_2 * 2^2 + i_1 * 2^1 + i_0 * 2^0 + f_1 * 2^{-1} + f_2 * 2^{-2} + f_3 * 2^{-3} + f_4 * 2^{-4}$$

Ex: -2.375

<u>Sign Bit:</u>

Önce 2.375 hesapla → 0010.0110
Sonra işaret bitini 1 yap → 1010.0110

Ex: -2.375

<u>2’s complement:</u>

Önce 2.375 hesapla → 0010.0110
Sonra 1’e tümleyen al (tersle) → 1101.1001
1 ekle (en sağa) → 1101.1010

1101 = -8+4+1 = -3   .1010 = +0.5 + 0.125 = 0.625

-3+0.625 = -2.375

Signed Fixed-Point

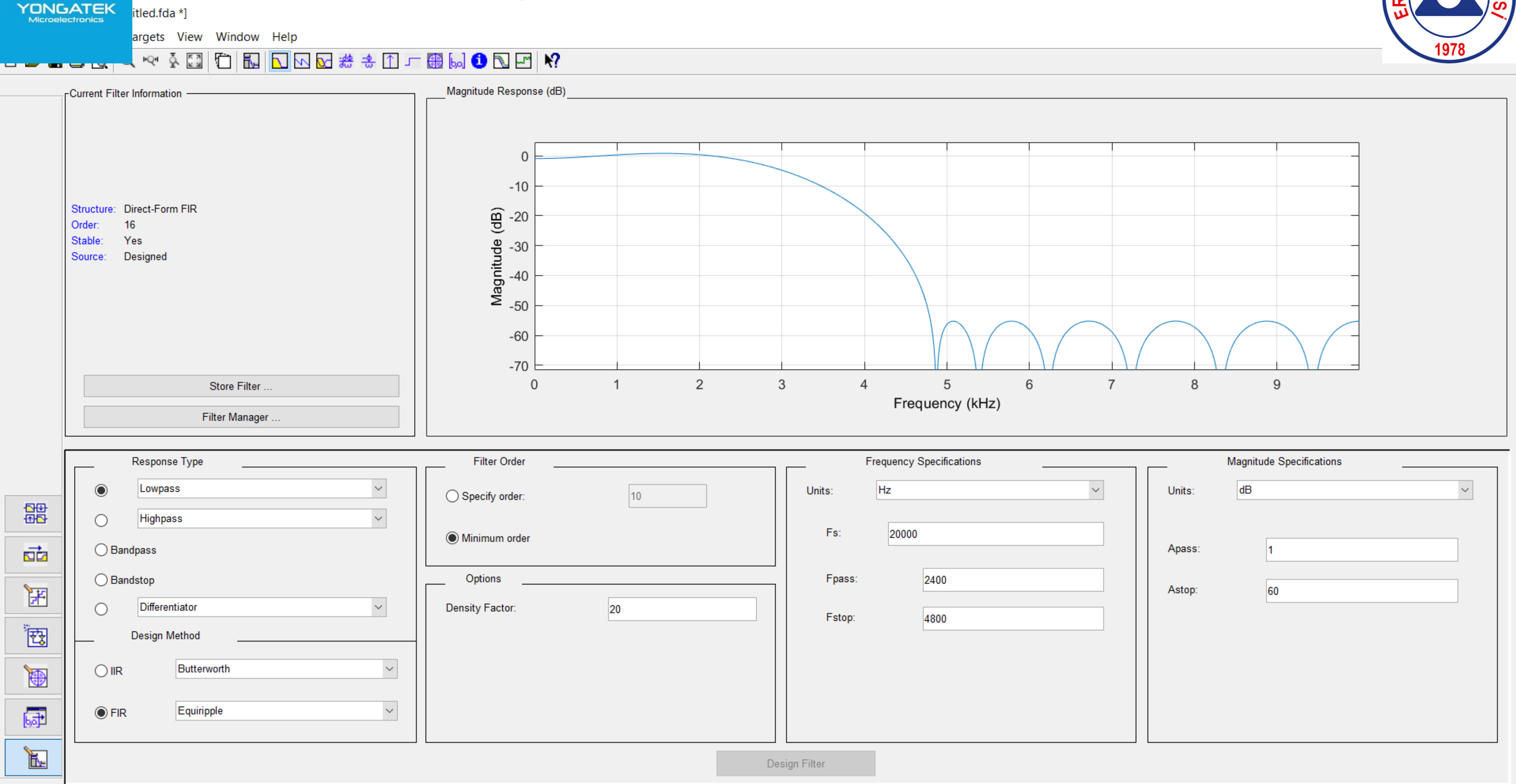
Ex: 0.75 + (-0.625) in Q4.4 form

0.750 → 0000.1100
-0.625 → 0.625 = 0000.1010 → 1’s complement = 1111.0101 → +1 rightmost = 1111.0110

```
   0000.1100
 + 1111.0110
 -----------
   0000.0010
```

Fixed-point number systems are commonly used in digital signal processing (DSP), graphics, and machine learning applications because the computations are faster and consume less power than they would in floating-point systems. Q1.15 (also known as Q15) is the most common format, storing signed numbers in the range (−1, 1) with 15 bits of precision. Q1.31 (also called just Q31) is sometimes used for higher precision intermediate results, such as in a Fast Fourier Transform. U8.8 is sometimes used for sensor readings sampled by analog/digital converters (ADCs). Note that all of these formats pack into 16- or 32-bit words for efficient storage in computer memories, which are typically a power of 2 in width. (Harris & Harris)

DSP | Low-Pass Filter Example



Current Filter Information

Structure: Direct-Form FIR
Order: 16
Stable: Yes
Source: Designed

Response Type

● Lowpass
○ Highpass
○ Bandpass
○ Bandstop
○ Differentiator

Design Method

○ IIR Butterworth
● FIR Equiripple

Filter Order

○ Specify order: 10
● Minimum order

Options

Density Factor: 20

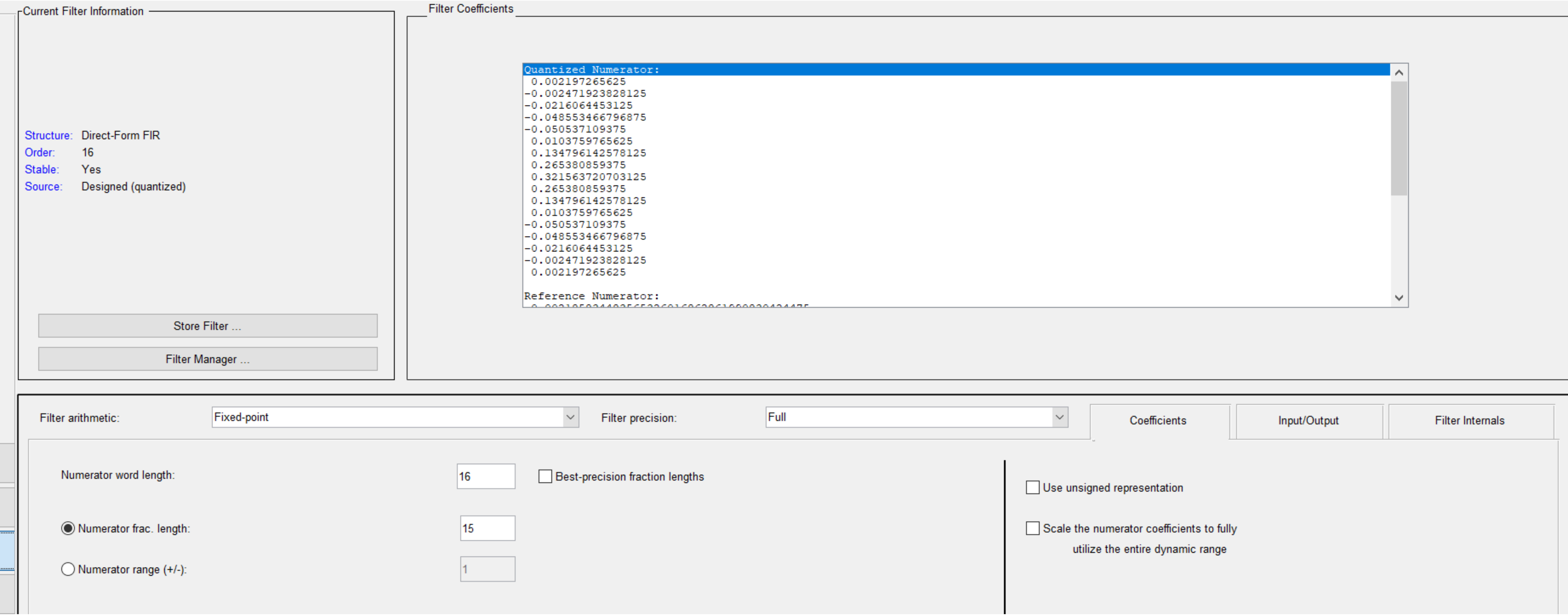
Frequency Specifications

Units: Hz
Fs: 20000
Fpass: 2400
Fstop: 4800

Magnitude Specifications

Units: dB
Apass: 1
Astop: 60

DSP | Low-Pass Filter | Fixed-Point



Current Filter Information

Structure: Direct-Form FIR
Order: 16
Stable: Yes
Source: Designed (quantized)

Filter Coefficients

```
Quantized Numerator:
 0.002197265625
-0.002471923828125
-0.0216064453125
-0.048553466796875
-0.050537109375
 0.0103759765625
 0.134796142578125
 0.265380859375
 0.321563720703125
 0.265380859375
 0.134796142578125
 0.0103759765625
-0.050537109375
-0.048553466796875
-0.0216064453125
-0.002471923828125
 0.002197265625

Reference Numerator:
```

Filter arithmetic: Fixed-point

Filter precision: Full

Numerator word length: 16

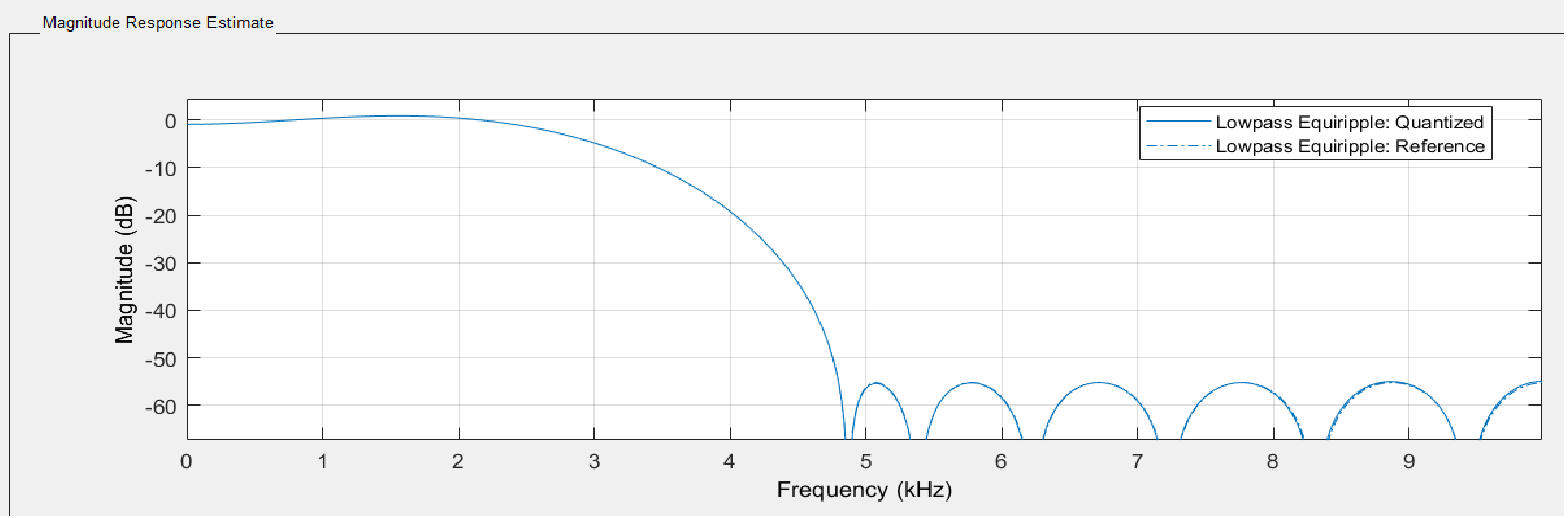
□ Best-precision fraction lengths

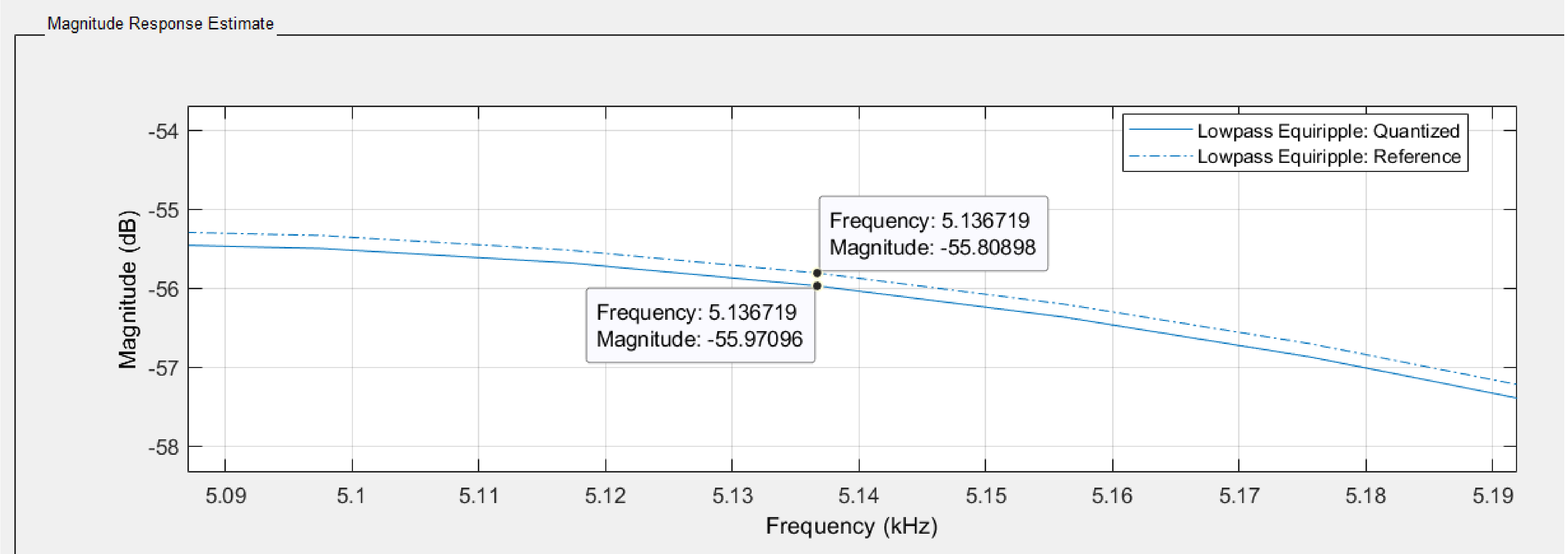
● Numerator frac. length: 15

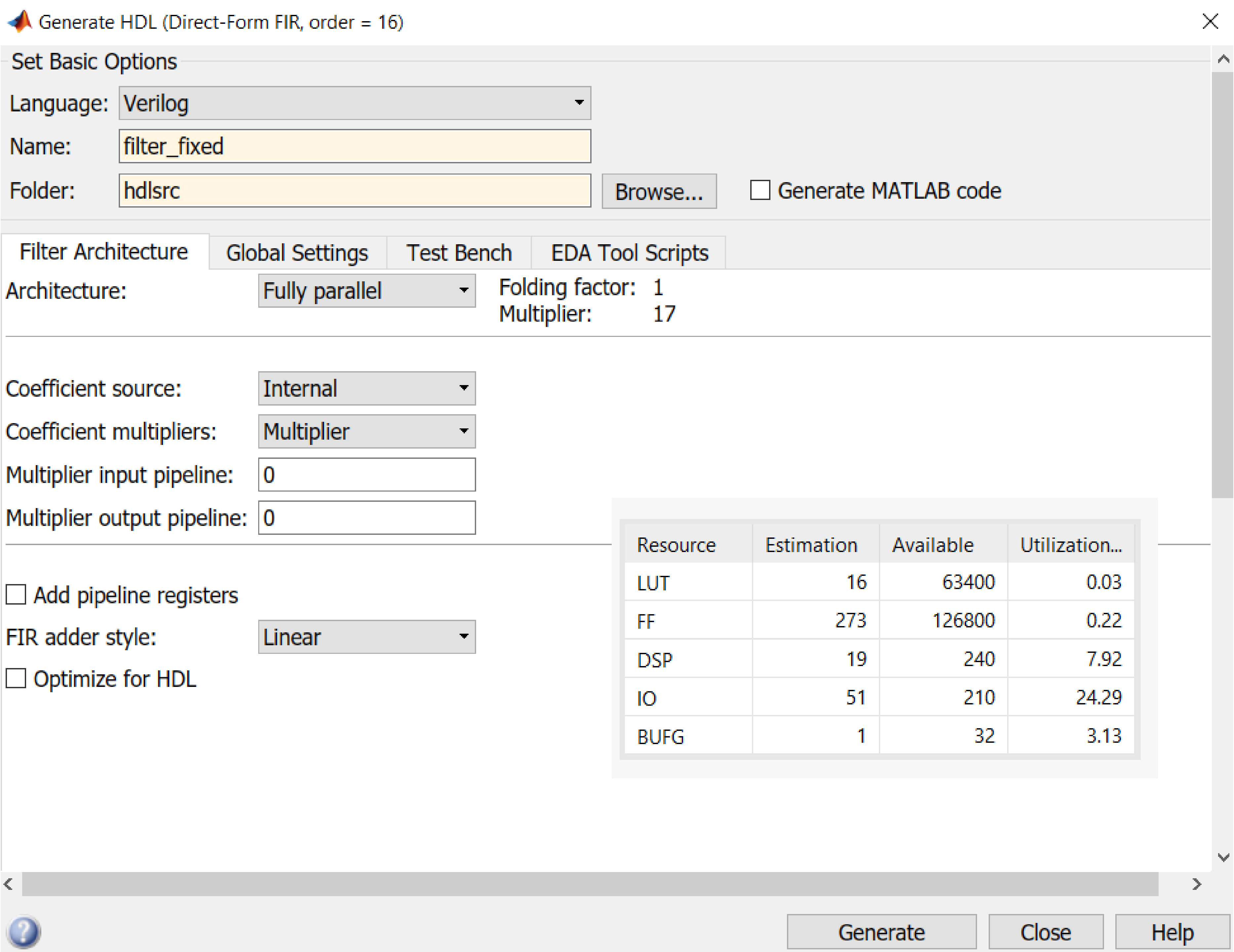
○ Numerator range (+/-): 1

□ Use unsigned representation

□ Scale the numerator coefficients to fully utilize the entire dynamic range


Magnitude Response Estimate
Magnitude (dB)
0
-10
-20
-30
-40
-50
-60
0
1
2
3
4
5
6
7
8
9
Frequency (kHz)
Lowpass Equiripple: Quantized
Lowpass Equiripple: Reference


Magnitude Response Estimate
Magnitude (dB)
-54
-55
-56
-57
-58
5.09
5.1
5.11
5.12
5.13
5.14
5.15
5.16
5.17
5.18
5.19
Frequency (kHz)
Lowpass Equiripple: Quantized
Lowpass Equiripple: Reference
Frequency: 5.136719
Magnitude: -55.80898
Frequency: 5.136719
Magnitude: -55.97096



Generate HDL (Direct-Form FIR, order = 16)

Set Basic Options

Language: Verilog

Name: filter_fixed

Folder: hdlsrc

□ Generate MATLAB code

Filter Architecture

Architecture: Fully parallel

Folding factor: 1

Multiplier: 17

Coefficient source: Internal

Coefficient multipliers: Multiplier

Multiplier input pipeline: 0

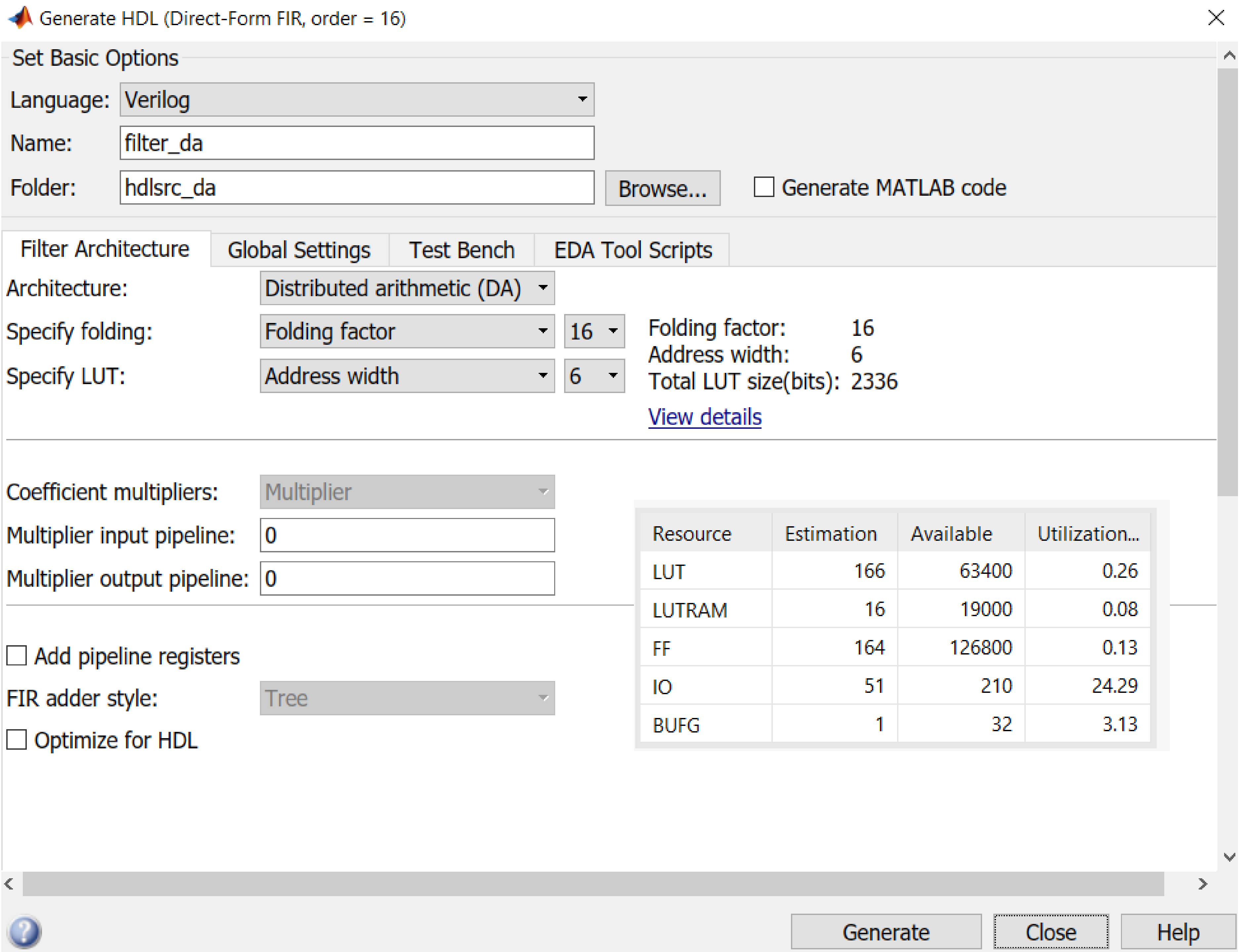
Multiplier output pipeline: 0

□ Add pipeline registers

FIR adder style: Linear

□ Optimize for HDL

Resource	Estimation	Available	Utilization...
LUT	16	63400	0.03
FF	273	126800	0.22
DSP	19	240	7.92
IO	51	210	24.29
BUFG	1	32	3.13



Generate HDL (Direct-Form FIR, order = 16)

Set Basic Options

Language: Verilog

Name: filter_da

Folder: hdlsrc_da

☐ Generate MATLAB code

Architecture: Distributed arithmetic (DA)

Specify folding: Folding factor 16

Specify LUT: Address width 6

Folding factor: 16
Address width: 6
Total LUT size(bits): 2336

Coefficient multipliers: Multiplier

Multiplier input pipeline: 0

Multiplier output pipeline: 0

☐ Add pipeline registers

FIR adder style: Tree

☐ Optimize for HDL

Resource	Estimation	Available	Utilization...
LUT	166	63400	0.26
LUTRAM	16	19000	0.08
FF	164	126800	0.13
IO	51	210	24.29
BUFG	1	32	3.13

Verilog Fixed-Point

Özel bir şey yapmaya gerek yok

Addition: +

Subtraction: -

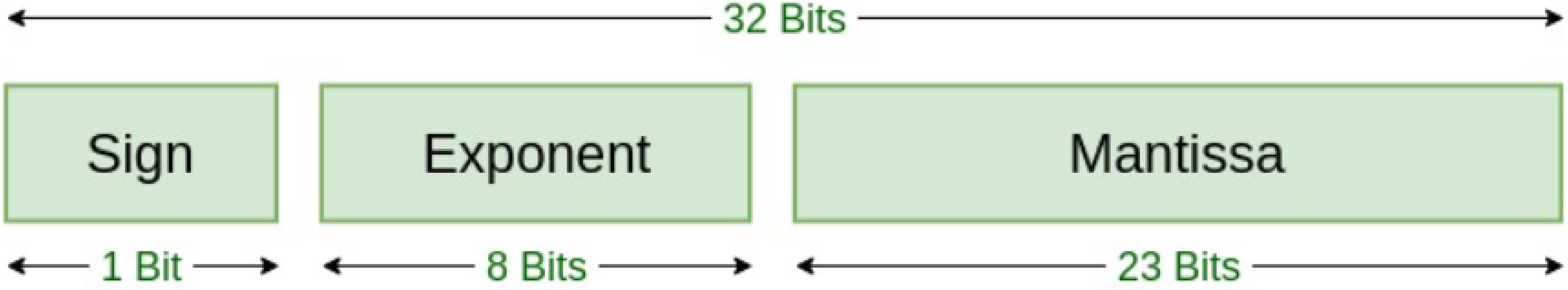
Multiplication: *

Çarpma yaparken input ve output’u “signed” olarak tanımlamayı unutmamak lazım

Floating-Point | IEEE-754

IEEE-754 floating-point standard (first in 1985, current version 2019)

Single precision (32-bit)



32 Bits

Sign

Exponent

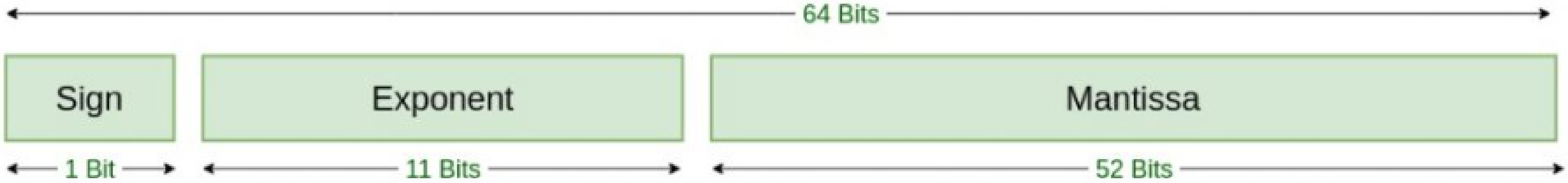
Mantissa

1 Bit

8 Bits

23 Bits

Double precision (64-bit)



64 Bits

Sign

Exponent

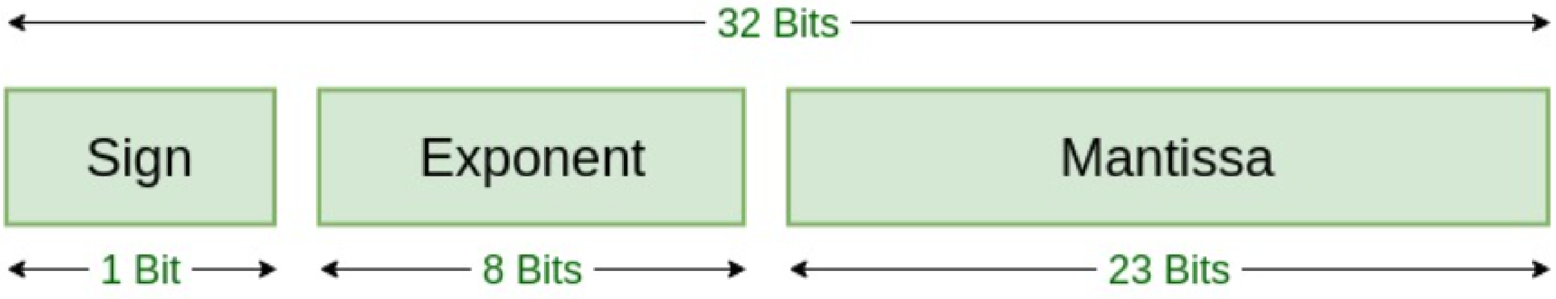
Mantissa

1 Bit

11 Bits

52 Bits

Floating-Point | IEEE-754



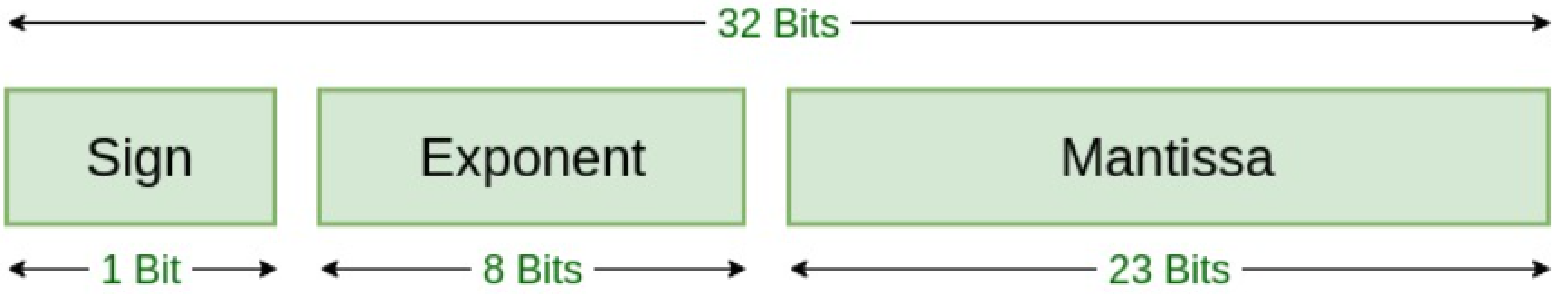
Sign : İşaret biti, ‘0’ pozitif, ‘1’ negatif
Exponent (Üs) : Biased (127) exponent.
Mantissa : Ondalıklı kısım

Ex: 85.125

85 = 1010101 0.125 = 001 85.125 = 1010101.001
Scientific Notation → 1.010101001 x 2^6

Sign : 0
Biased Exponent : 127+6 (2^6’nın altısı) = 133 = 10000101
Mantissa : 010101001 (23 bite tamamlarsak) = 01010100100000000000000
: [0][10000101][01010100100000000000000] → 0x42AA4000

Floating-Point | IEEE-754



Sign : İşaret biti, ‘0’ pozitif, ‘1’ negatif
Exponent (Üs) : Biased (127) exponent.
Mantissa : Ondalıklı kısım

Ex: - 0.75

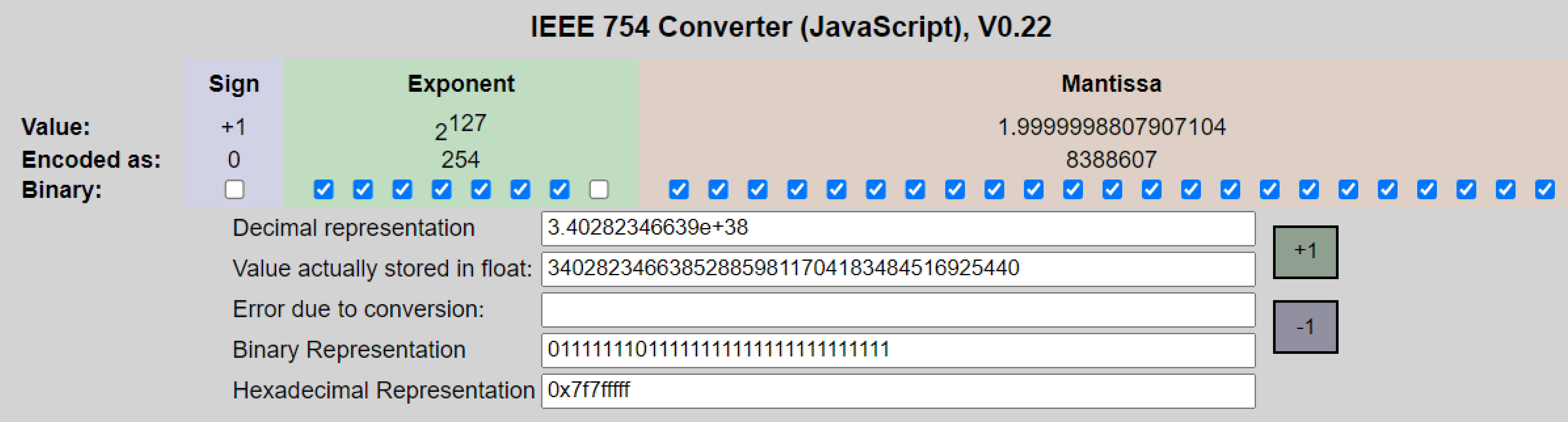
0 = 0 0.75 = 110 0.75 = 0.110
Scientific Notation → 1.10 x 2^-1

Sign : 1
Biased Exponent : 127-1 (2^-1’in -1) = 126 = 01111110
Mantissa : 100000000 (23 bite tamamlarsak) = 10000000000000000000000
: [1][01111110][10000000000000000000000] → 0xBF400000

Floating-Point | IEEE-754

Number	Sign	Exponent	Fraction
0	X	00000000	00000000000000000000000
∞	0	11111111	00000000000000000000000
−∞	1	11111111	00000000000000000000000
NaN	X	11111111	Non-zero

Format	Total Bits	Sign Bits	Exponent Bits	Fraction Bits	Bias
Single	32	1	8	23	127
Double	64	1	11	52	1023
Quad	128	1	15	112	16363



IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^{127}	1.9999998807907104
Encoded as:	0	254	8388607
Binary:	☐	☑ ☑ ☑ ☑ ☑ ☑ ☑ ☐	☑ ☑

Decimal representation: 3.40282346639e+38

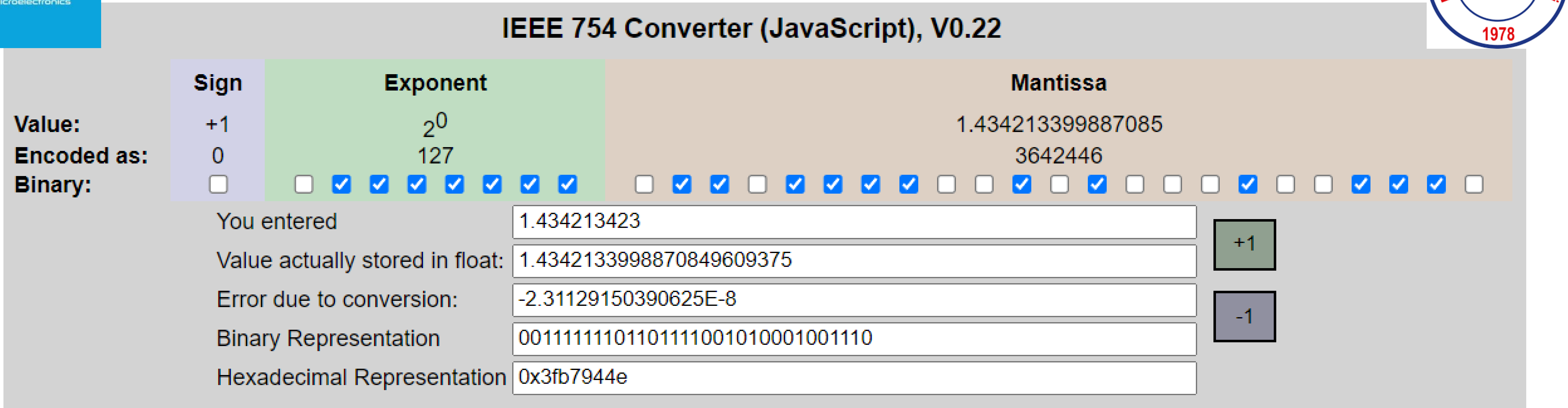
Value actually stored in float: 340282346638528859811704183484516925440

Error due to conversion:

Binary Representation: 01111111011111111111111111111111

Hexadecimal Representation: 0x7f7fffff

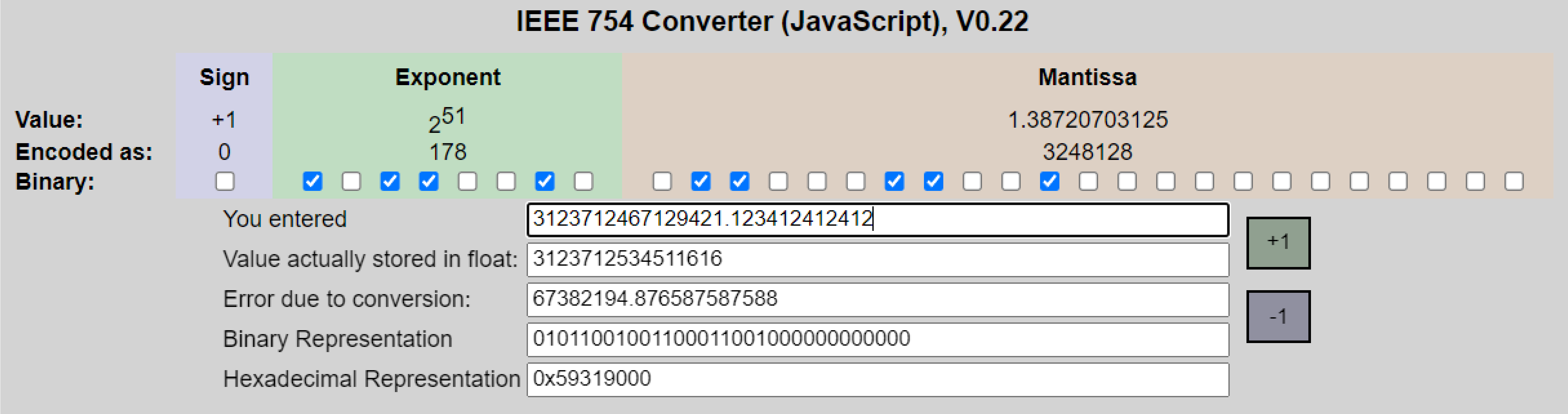
Floating-Point | IEEE-754 | Rounding



IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^{0}	1.434213399887085
Encoded as:	0	127	3642446
Binary:	☐	☐☑☑☑☑☑☑☑	☐☑☑☐☑☑☑☑☐☐☑☐☑☐☐☐☑☐☐☑☑☑☐

You entered	1.434213423
Value actually stored in float:	1.4342133998870849609375
Error due to conversion:	-2.31129150390625E-8
Binary Representation	00111111101101111001010001001110
Hexadecimal Representation	0x3fb7944e

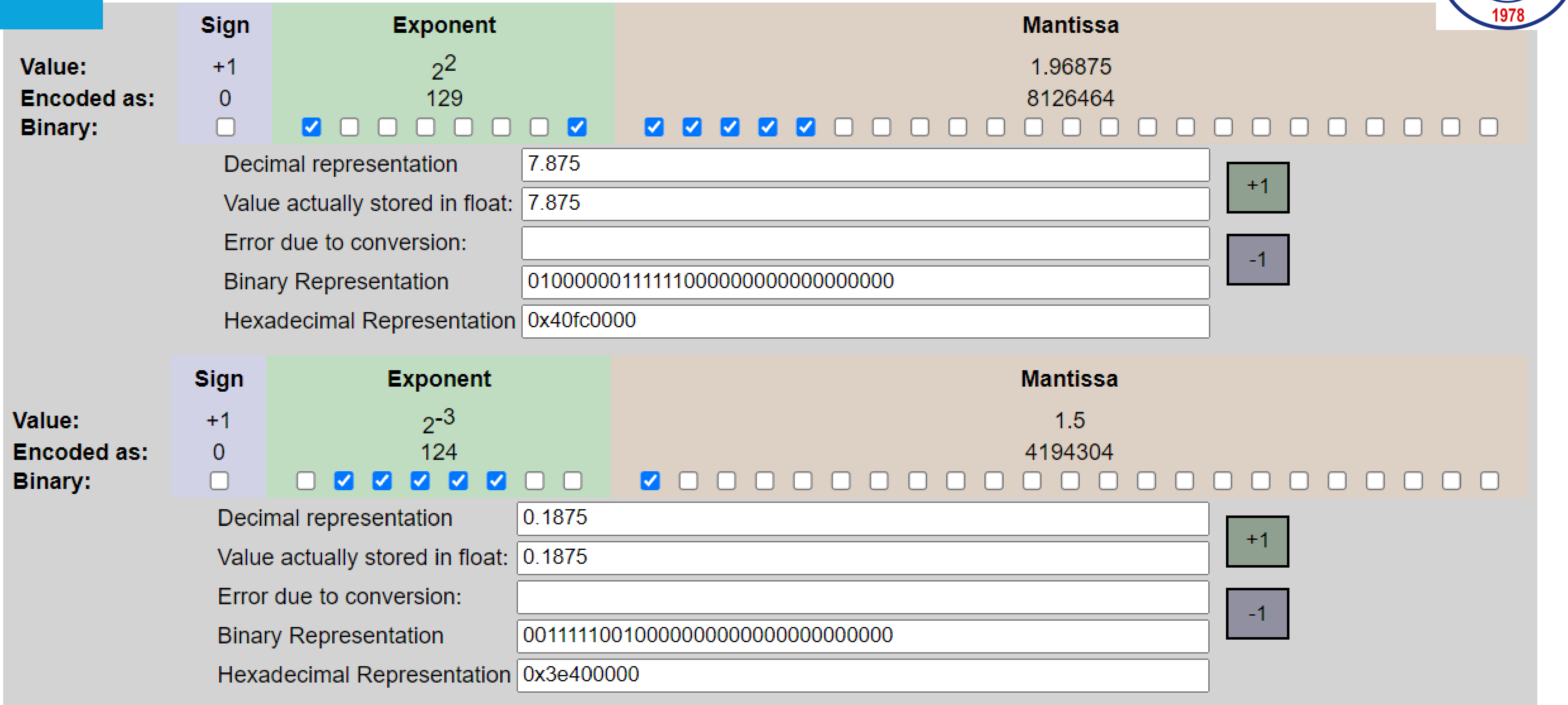


IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^{51}	1.38720703125
Encoded as:	0	178	3248128
Binary:	☐	☑☐☑☑☐☐☑☐	☐☑☑☐☐☐☑☑☐☐☑☐☐☐☐☐☐☐☐☐☐☐☐

You entered	3123712467129421.123412412412
Value actually stored in float:	3123712534511616
Error due to conversion:	67382194.876587587588
Binary Representation	01011001001100011001000000000000
Hexadecimal Representation	0x59319000

Floating-Point | IEEE-754 | Addition



	Sign	Exponent	Mantissa
Value:	+1	2^{2}	1.96875
Encoded as:	0	129	8126464
Binary:	□	☑ □ □ □ □ □ □ ☑	☑ ☑ ☑ ☑ ☑ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

Decimal representation: 7.875

Value actually stored in float: 7.875

Error due to conversion:

Binary Representation: 01000000111111000000000000000000

Hexadecimal Representation: 0x40fc0000

	Sign	Exponent	Mantissa
Value:	+1	2^{-3}	1.5
Encoded as:	0	124	4194304
Binary:	□	□ ☑ ☑ ☑ ☑ ☑ □ □	☑ □

Decimal representation: 0.1875

Value actually stored in float: 0.1875

Error due to conversion:

Binary Representation: 00111110010000000000000000000000

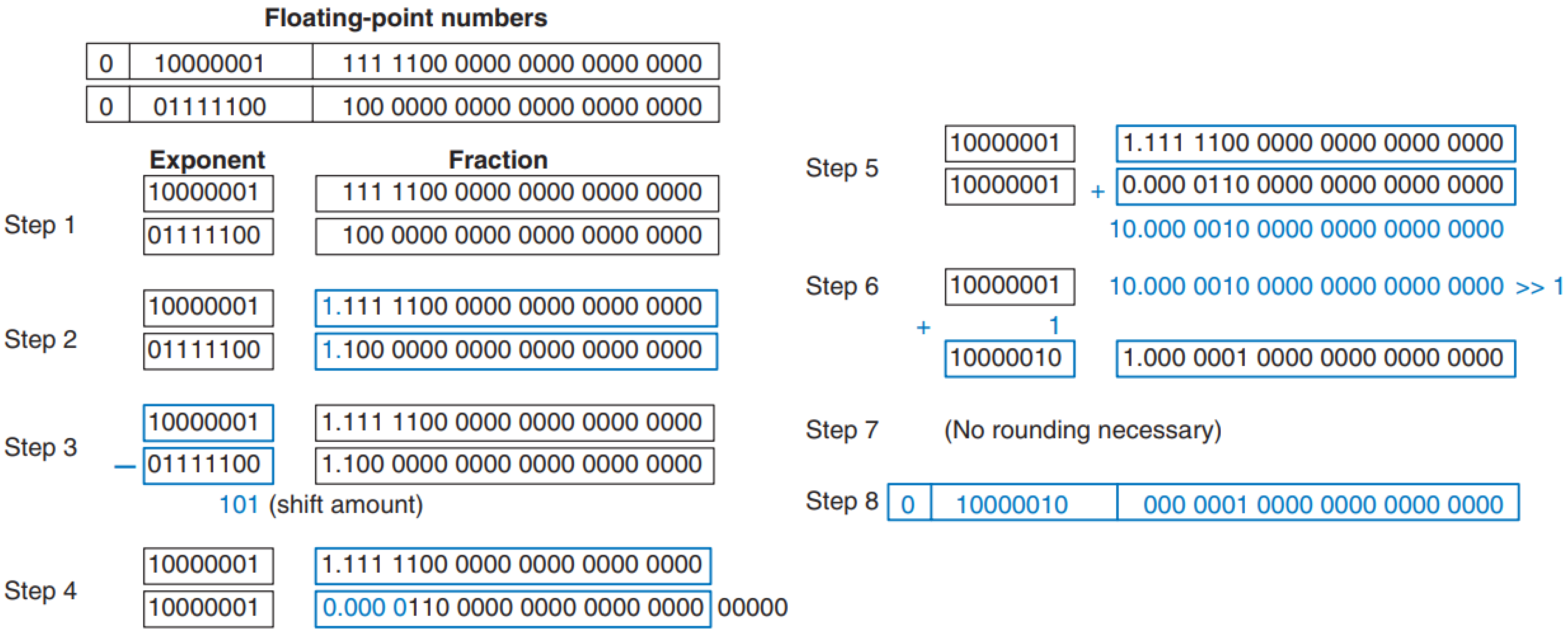
Hexadecimal Representation: 0x3e400000

Floating-Point | IEEE-754 | Addition

7.875 + 0.1875 = ???

7.875 : 7 = 111, 0.875 = 0.111 → 111.111 = 1.11111 x 2^2

0.1875 : 0 = 0, 0.1875 = 0.0011 → 0.0011 = 1.1 x 2^-3



Floating-point numbers

0	10000001	111 1100 0000 0000 0000 0000
0	01111100	100 0000 0000 0000 0000 0000

	Exponent	Fraction
Step 1	10000001	111 1100 0000 0000 0000 0000
	01111100	100 0000 0000 0000 0000 0000
Step 2	10000001	1.111 1100 0000 0000 0000 0000
	01111100	1.100 0000 0000 0000 0000 0000
Step 3	10000001	1.111 1100 0000 0000 0000 0000
	– 01111100	1.100 0000 0000 0000 0000 0000
	101 (shift amount)	
Step 4	10000001	1.111 1100 0000 0000 0000 0000
	10000001	0.000 0110 0000 0000 0000 0000 00000

Step 5	10000001	1.111 1100 0000 0000 0000 0000
	10000001	+ 0.000 0110 0000 0000 0000 0000
		10.000 0010 0000 0000 0000 0000
Step 6	10000001	10.000 0010 0000 0000 0000 0000 >> 1
	+ 1	
	10000010	1.000 0001 0000 0000 0000 0000
Step 7	(No rounding necessary)	

Step 8

0	10000010	000 0001 0000 0000 0000 0000

Floating-Point | IEEE-754 | Multiplication

0.5 * (-0.4375) = ???

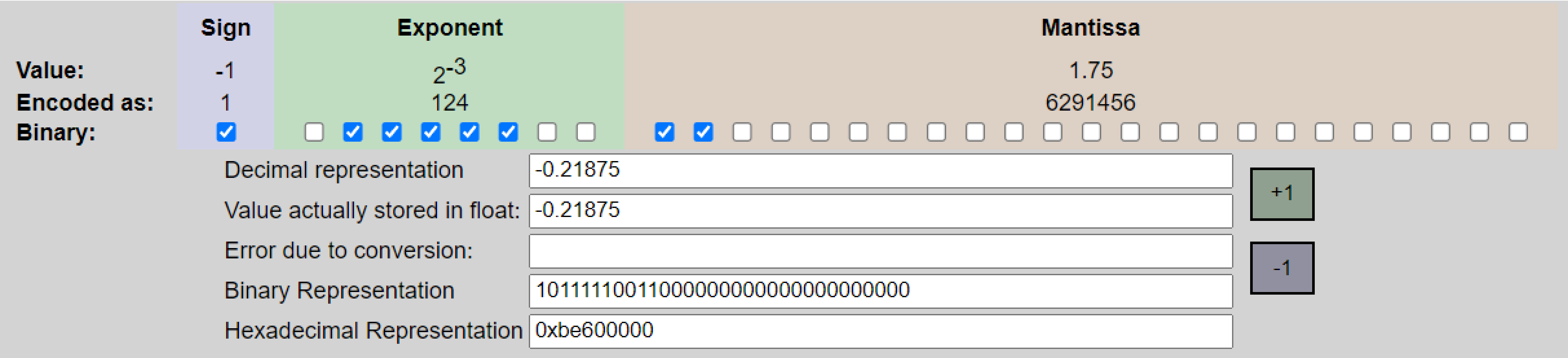
0.5	: 0 = 0, 0.5 = 0.1	→ 0.1	= 1.0 x 2^-1
0.4375	: 0 = 0, 0.4375 = 0.0111	→ 0.0111	= -1.11 x 2^-2

Add exponents: -1-2 = -3

Multiply fractions: 1.0 * 1.11 = 1.11

→ -1.11 * 2^-3 = -0.21875

[1][01111100][11000000000000000000000] → 0xBE600000



	Sign	Exponent	Mantissa
Value:	-1	2^{-3}	1.75
Encoded as:	1	124	6291456
Binary:	☑	☐ ☑ ☑ ☑ ☑ ☑ ☐ ☐	☑ ☑ ☐

Decimal representation: -0.21875

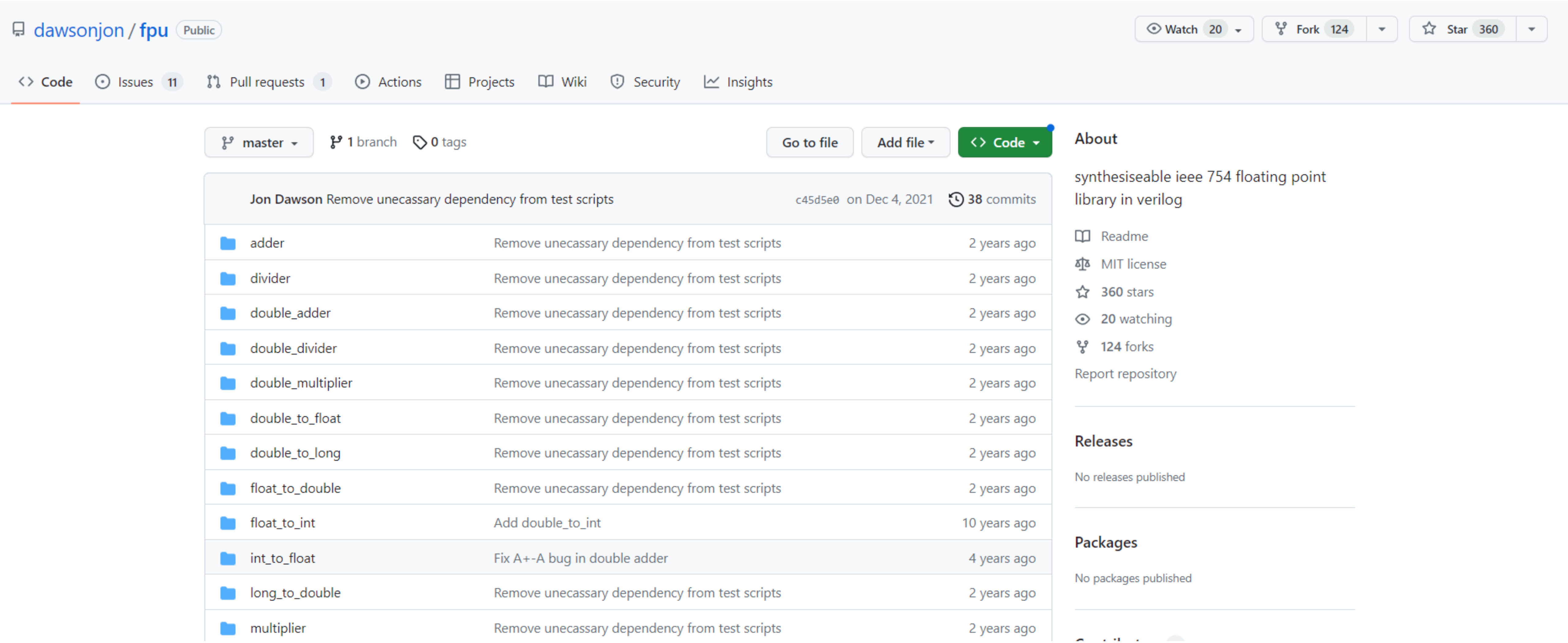
Value actually stored in float: -0.21875

Error due to conversion:

Binary Representation: 10111110011000000000000000000000

Hexadecimal Representation: 0xbe600000

Floating-Point Arithmetic IP Cores



Jon Dawson Remove unecassary dependency from test scripts c45d5e0 on Dec 4, 2021 38 commits

adder	Remove unecassary dependency from test scripts	2 years ago
divider	Remove unecassary dependency from test scripts	2 years ago
double_adder	Remove unecassary dependency from test scripts	2 years ago
double_divider	Remove unecassary dependency from test scripts	2 years ago
double_multiplier	Remove unecassary dependency from test scripts	2 years ago
double_to_float	Remove unecassary dependency from test scripts	2 years ago
double_to_long	Remove unecassary dependency from test scripts	2 years ago
float_to_double	Remove unecassary dependency from test scripts	2 years ago
float_to_int	Add double_to_int	10 years ago
int_to_float	Fix A+-A bug in double adder	4 years ago
long_to_double	Remove unecassary dependency from test scripts	2 years ago
multiplier	Remove unecassary dependency from test scripts	2 years ago

About

synthesiseable ieee 754 floating point library in verilog

Readme

MIT license

360 stars

20 watching

124 forks

Report repository

Releases

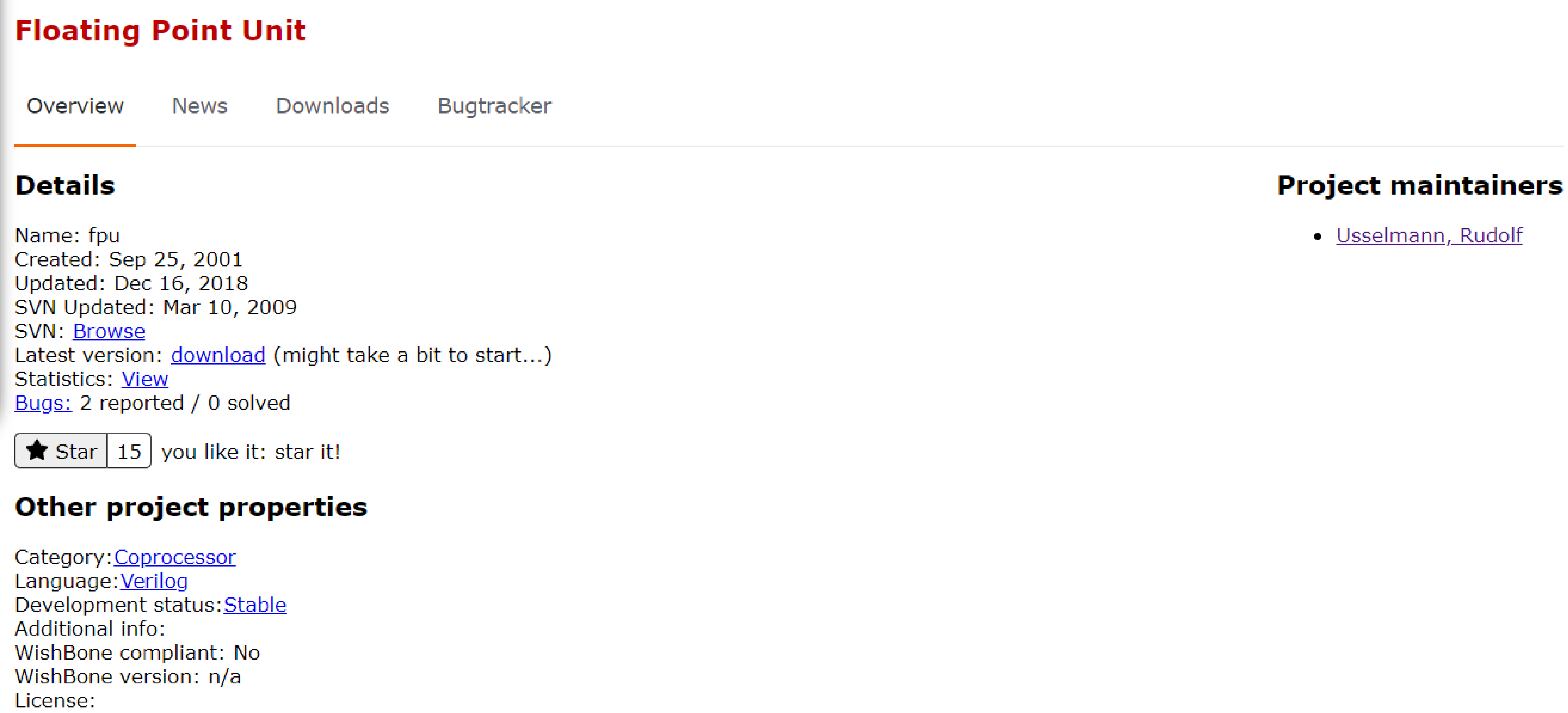
No releases published

Packages

No packages published

https://github.com/dawsonjon/fpu

Floating-Point Arithmetic IP Cores



Floating Point Unit

Details

Name: fpu
Created: Sep 25, 2001
Updated: Dec 16, 2018
SVN Updated: Mar 10, 2009
SVN: Browse
Latest version: download (might take a bit to start...)
Statistics: View
Bugs: 2 reported / 0 solved

Project maintainers

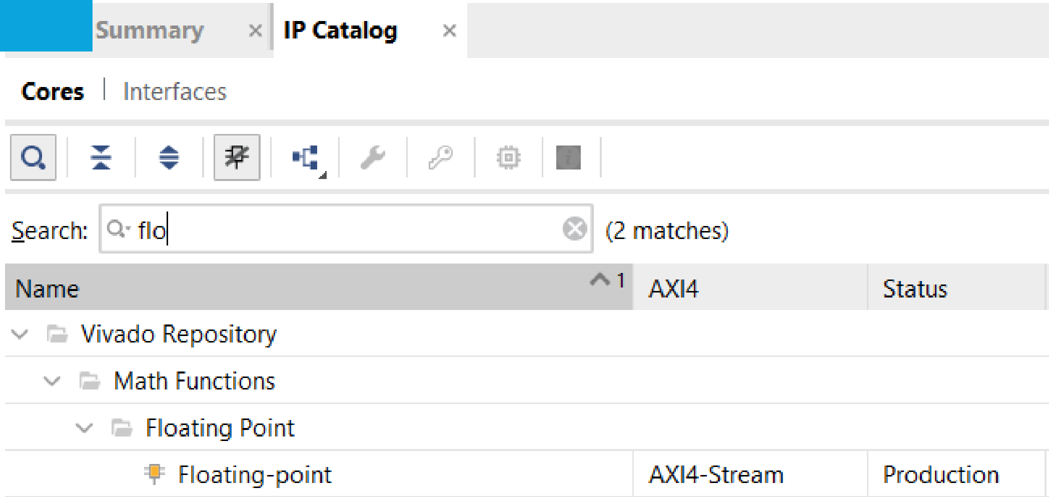
- Usselmann, Rudolf

Other project properties

Category:Coprocessor
Language:Verilog
Development status:Stable
Additional info:
WishBone compliant: No
WishBone version: n/a
License:

https://opencores.org/projects/fpu

Floating-Point Arithmetic IP Cores

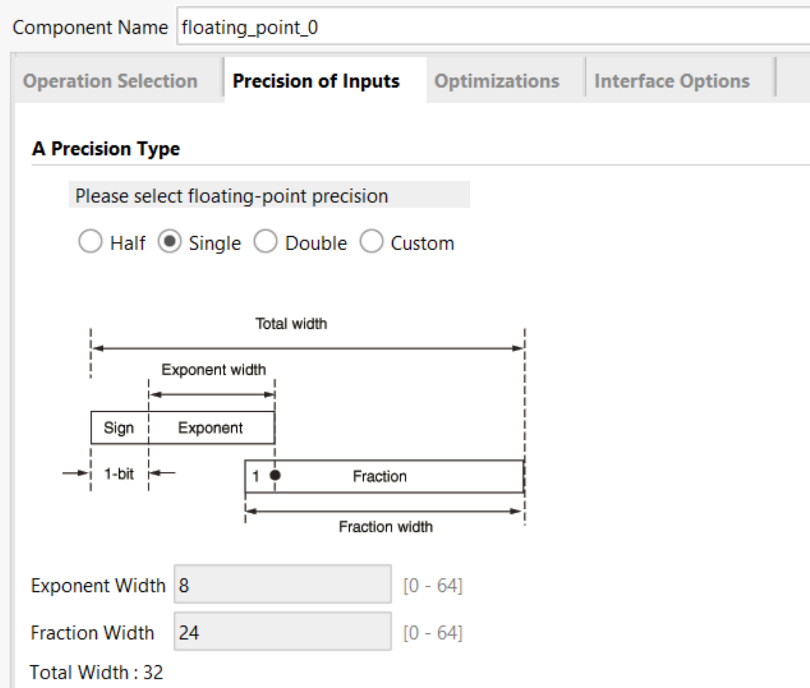


Summary × IP Catalog ×

Cores | Interfaces

Search: flo (2 matches)

Name	AXI4	Status
Vivado Repository		
Math Functions		
Floating Point		
Floating-point	AXI4-Stream	Production



Component Name: floating_point_0

Operation Selection | **Precision of Inputs** | Optimizations | Interface Options

A Precision Type

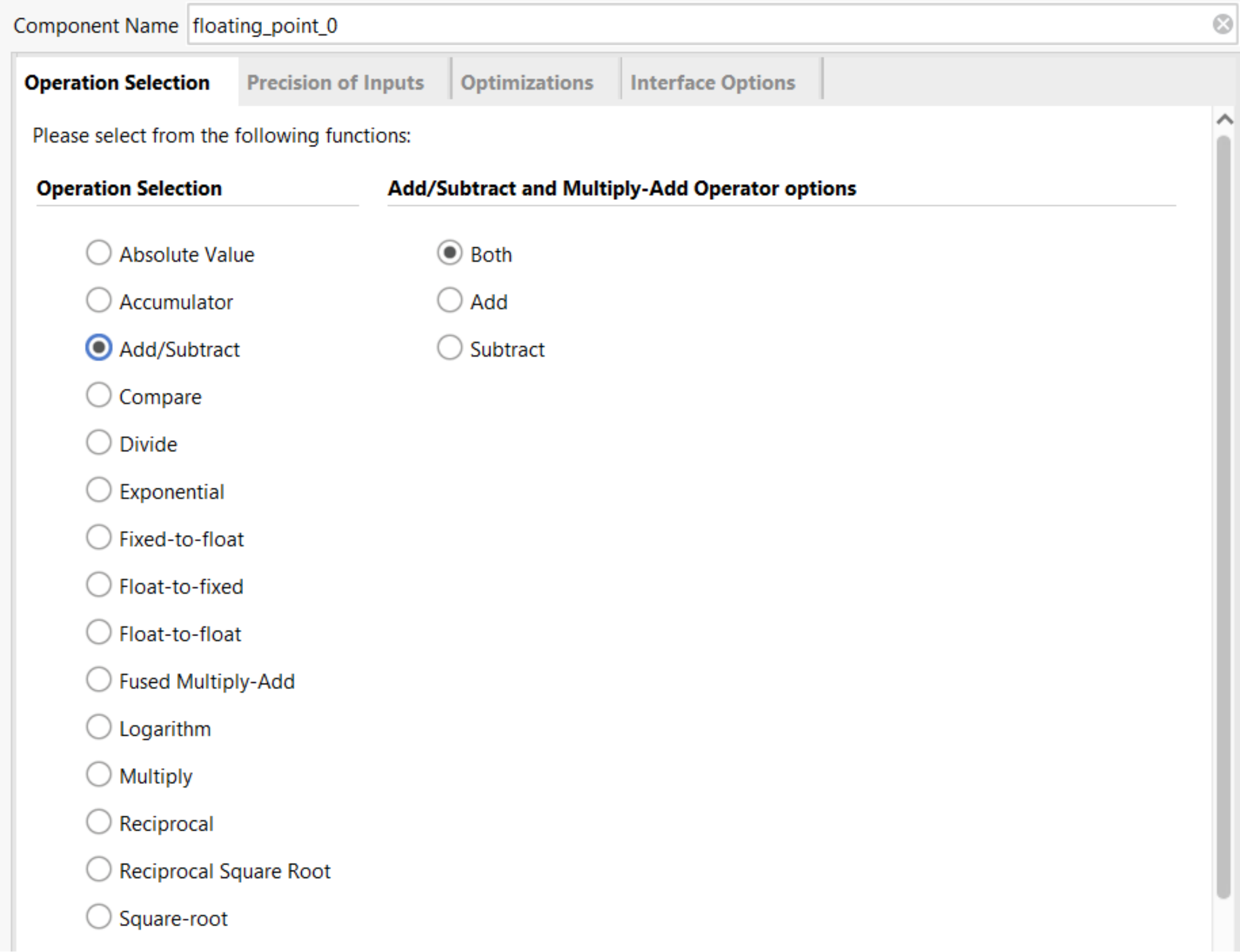
Please select floating-point precision

○ Half ● Single ○ Double ○ Custom

Exponent Width: 8 [0 - 64]

Fraction Width: 24 [0 - 64]

Total Width : 32



Component Name: floating_point_0

Operation Selection | Precision of Inputs | Optimizations | Interface Options

Please select from the following functions:

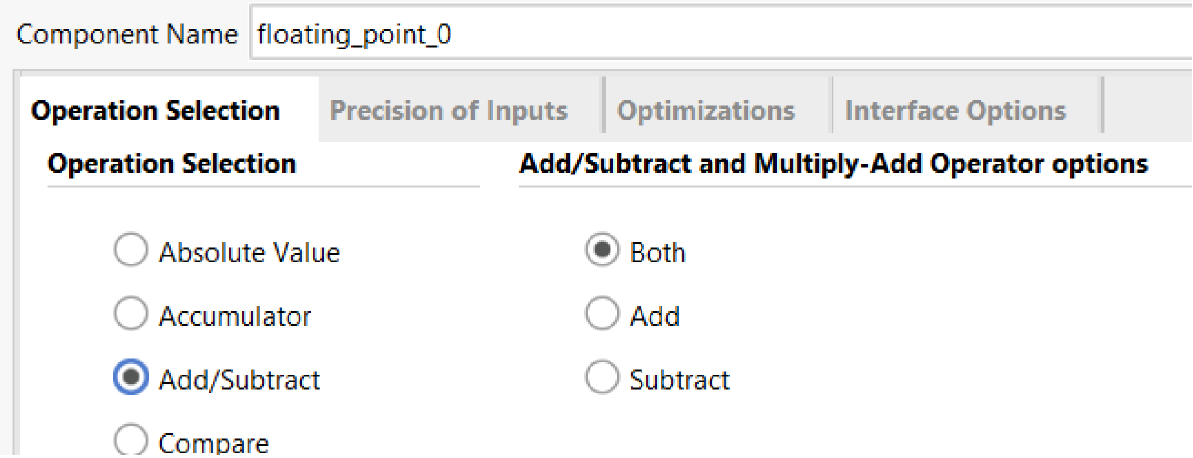
Operation Selection

○ Absolute Value
○ Accumulator
● Add/Subtract
○ Compare
○ Divide
○ Exponential
○ Fixed-to-float
○ Float-to-fixed
○ Float-to-float
○ Fused Multiply-Add
○ Logarithm
○ Multiply
○ Reciprocal
○ Reciprocal Square Root
○ Square-root

Add/Subtract and Multiply-Add Operator options

● Both
○ Add
○ Subtract

Floating-Point Arithmetic IP Cores



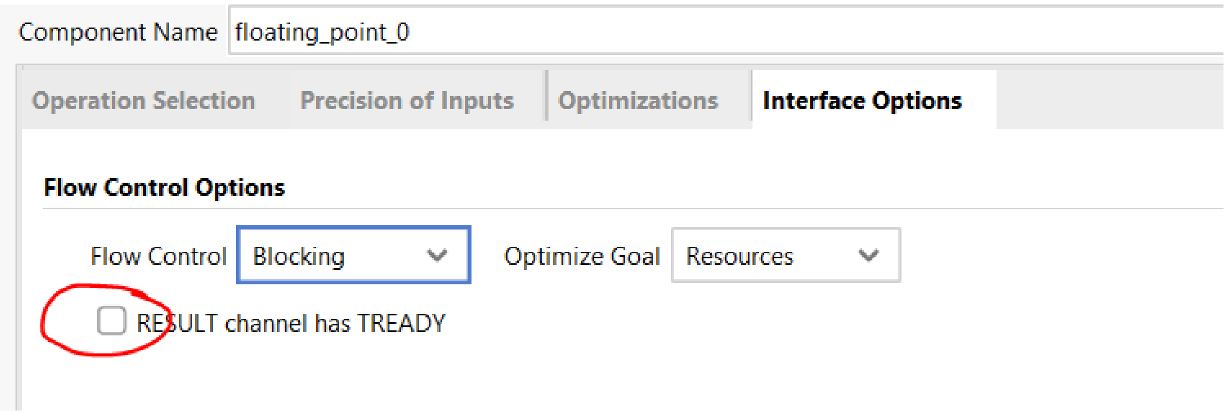
Component Name: floating_point_0

Operation Selection

- ○ Absolute Value
- ○ Accumulator
- ● Add/Subtract
- ○ Compare

Add/Subtract and Multiply-Add Operator options

- ● Both
- ○ Add
- ○ Subtract



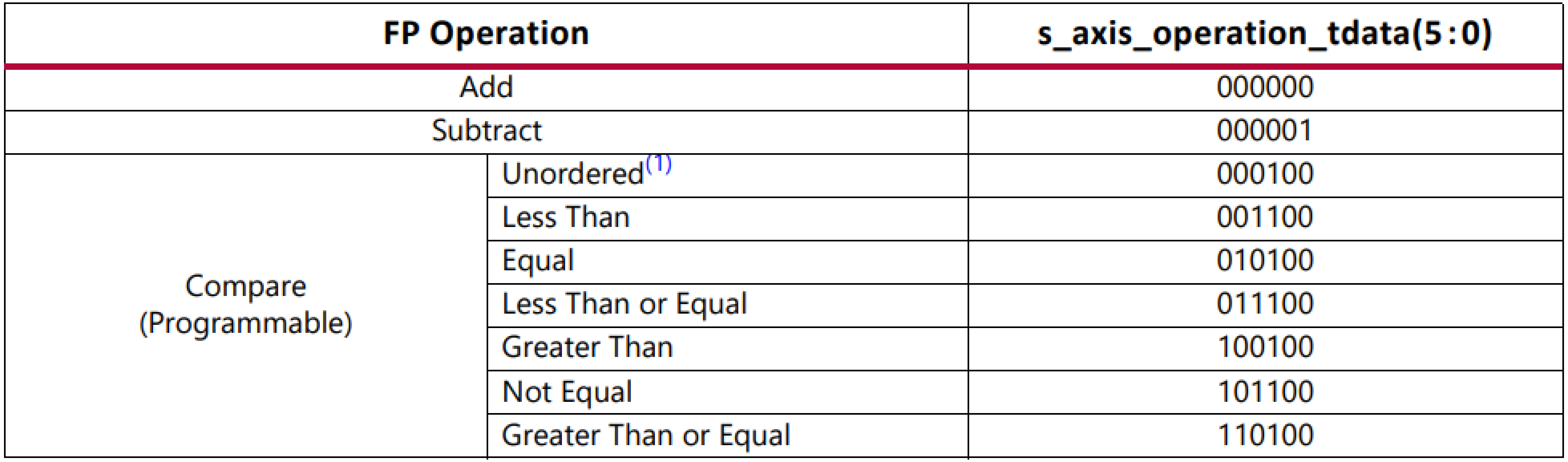
Component Name: floating_point_0

Flow Control Options

Flow Control: Blocking

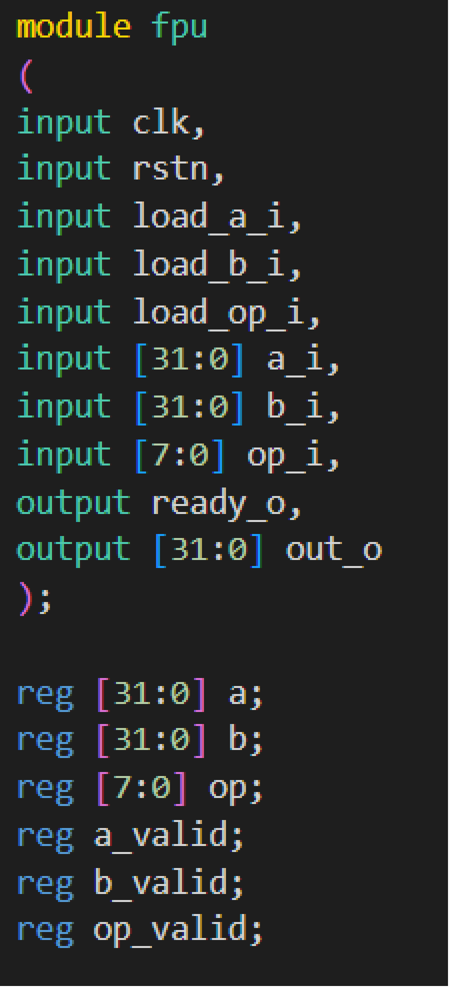
Optimize Goal: Resources

□ RESULT channel has TREADY

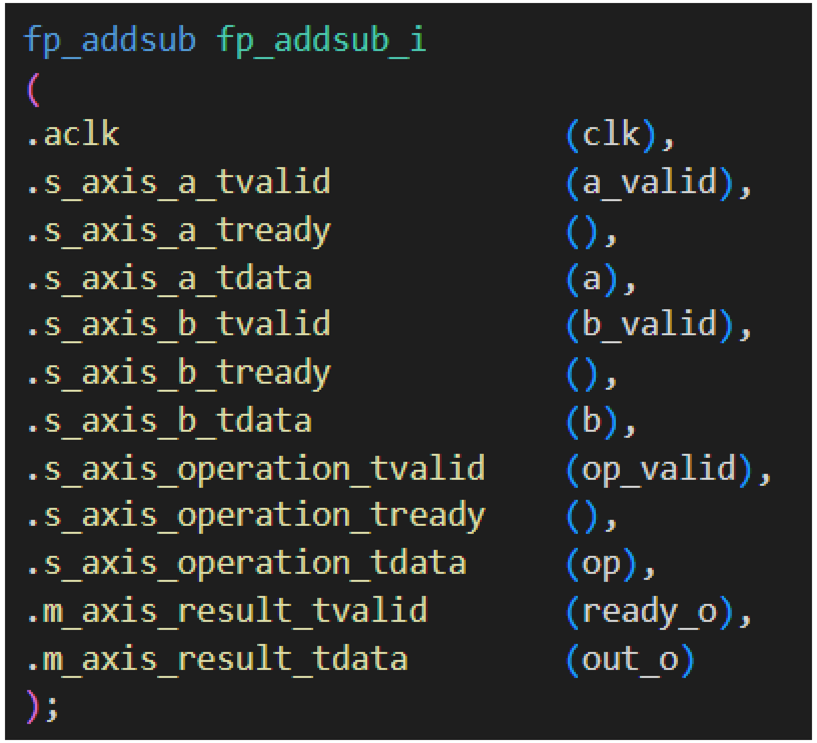


FP Operation		s_axis_operation_tdata(5:0)
Add		000000
Subtract		000001
Compare (Programmable)	Unordered[1]	000100
	Less Than	001100
	Equal	010100
	Less Than or Equal	011100
	Greater Than	100100
	Not Equal	101100
	Greater Than or Equal	110100

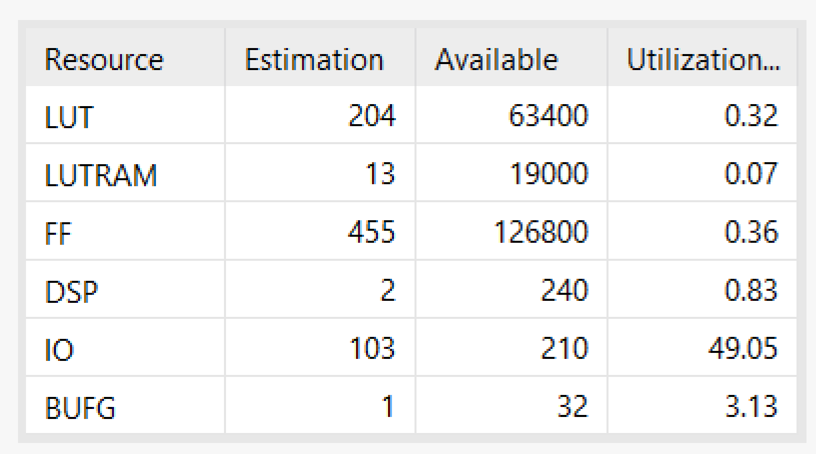
Floating-Point Arithmetic IP Cores



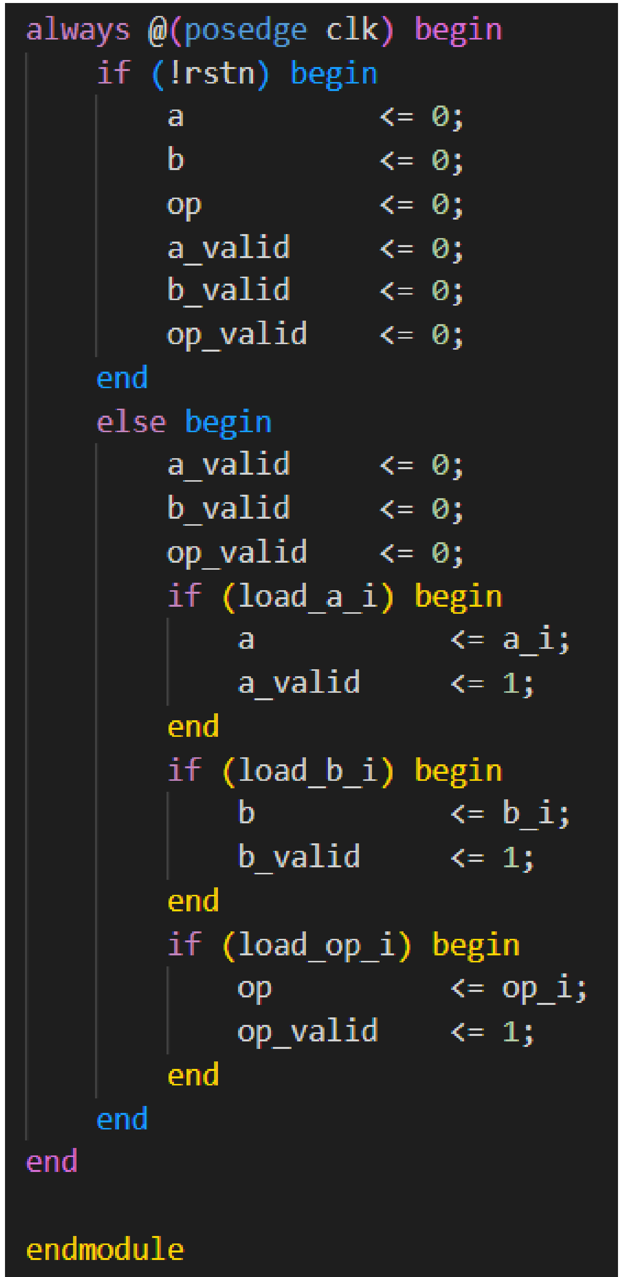
```
module fpu
(
input clk,
input rstn,
input load_a_i,
input load_b_i,
input load_op_i,
input [31:0] a_i,
input [31:0] b_i,
input [7:0] op_i,
output ready_o,
output [31:0] out_o
);

reg [31:0] a;
reg [31:0] b;
reg [7:0] op;
reg a_valid;
reg b_valid;
reg op_valid;
```



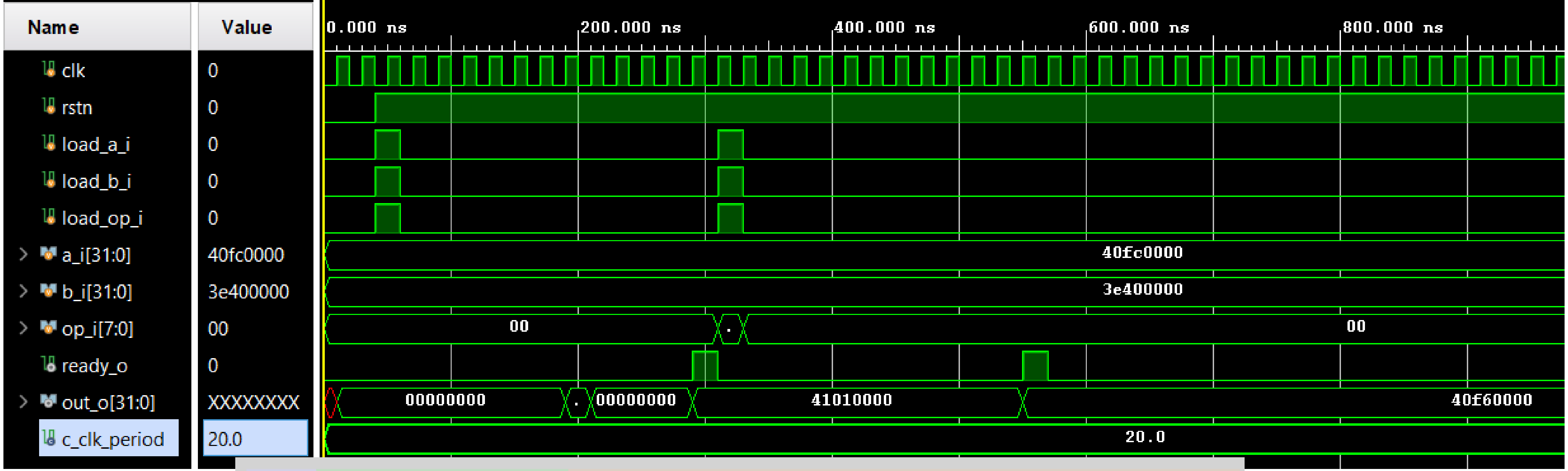
```
fp_addsub fp_addsub_i
(
.aclk                       (clk),
.s_axis_a_tvalid            (a_valid),
.s_axis_a_tready            (),
.s_axis_a_tdata             (a),
.s_axis_b_tvalid            (b_valid),
.s_axis_b_tready            (),
.s_axis_b_tdata             (b),
.s_axis_operation_tvalid    (op_valid),
.s_axis_operation_tready    (),
.s_axis_operation_tdata     (op),
.m_axis_result_tvalid       (ready_o),
.m_axis_result_tdata        (out_o)
);
```



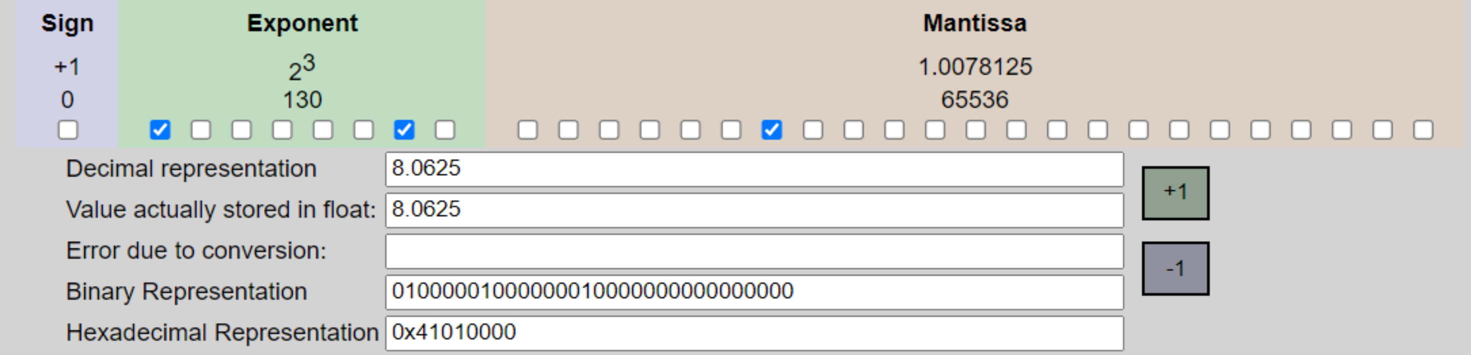
Resource	Estimation	Available	Utilization...
LUT	204	63400	0.32
LUTRAM	13	19000	0.07
FF	455	126800	0.36
DSP	2	240	0.83
IO	103	210	49.05
BUFG	1	32	3.13



```
always @(posedge clk) begin
    if (!rstn) begin
        a           <= 0;
        b           <= 0;
        op          <= 0;
        a_valid     <= 0;
        b_valid     <= 0;
        op_valid    <= 0;
    end
    else begin
        a_valid     <= 0;
        b_valid     <= 0;
        op_valid    <= 0;
        if (load_a_i) begin
            a           <= a_i;
            a_valid     <= 1;
        end
        if (load_b_i) begin
            b           <= b_i;
            b_valid     <= 1;
        end
        if (load_op_i) begin
            op          <= op_i;
            op_valid    <= 1;
        end
    end
end

endmodule
```



Name	Value
clk	0
rstn	0
load_a_i	0
load_b_i	0
load_op_i	0
a_i[31:0]	40fc0000
b_i[31:0]	3e400000
op_i[7:0]	00
ready_o	0
out_o[31:0]	XXXXXXXX
c_clk_period	20.0



Sign	Exponent	Mantissa
+1	2^3	1.0078125
0	130	65536
□	☑ □ □ □ □ □ ☑ □	□ □ □ □ □ □ ☑ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

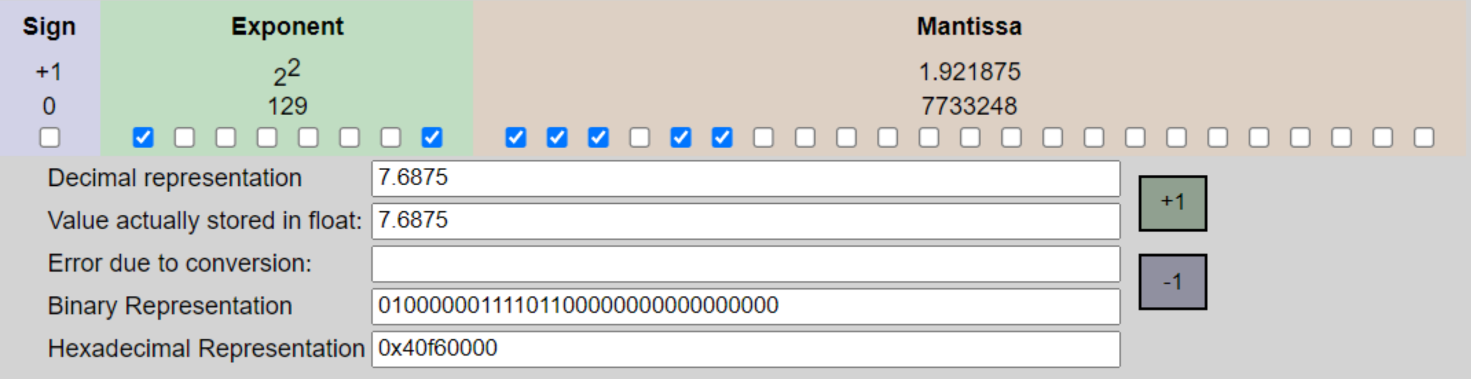
Decimal representation: 8.0625

Value actually stored in float: 8.0625

Error due to conversion:

Binary Representation: 01000001000000010000000000000000

Hexadecimal Representation: 0x41010000



Sign	Exponent	Mantissa
+1	2^2	1.921875
0	129	7733248
□	☑ □ □ □ □ □ □ ☑	☑ ☑ ☑ □ ☑ ☑ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

Decimal representation: 7.6875

Value actually stored in float: 7.6875

Error due to conversion:

Binary Representation: 01000000111101100000000000000000

Hexadecimal Representation: 0x40f60000