Name: Suraj Salian ID:2018AAPS0253G

Final Submission: Floating Point Adder:

Modules Developed:

1. Exponent Compare (Small ALU)
2. 24-bit MUX
3. Shift Right Fraction
4. 8-bit MUX (For Exponent)
5. Input Compare
6. Big ALU
7. Rounding/ Normalizing Hardware
8. Main Module
9. Input Exception Handling
10. Test-Bench

Assumptions:

1. Inputs to the given model are always in IEEE-754 Format, and inputs should be in normalized form.
   i.e, Binary number Of Form: $1. < \cdots \ldots \ldots \text{Fraction Part} \ldots \ldots > * \; 2^{y} \quad (\text{y} - \text{exponent})$
   Inputs: A, B (IEEE 754 single precision format- 32 Bits)

2. The input number cannot be NaN, it should be valid number of $\pm\infty$
3. The model outputs result in normalized IEEE-754 format.
4. Edge Cases: $If \ (any \ one \ of \ input \ or \ both \ inputs) = \pm\infty \rightarrow \ output = \pm\infty.$
   $If \ \ Input \ A = \pm\infty \ and \ Input \ B = \mp\infty \rightarrow output = 0$

Module Description & Design Flow:

- **Exponent Compare (small ALU)**: The exponents of two inputs are compare and their absolute difference is passed by the wire '$SHAMT$' (Shift Amount).
  Pseudo Code Logic:

```
$If \ Exponent_A > Exponent_B \rightarrow EXP\_INDICATE = 0;$
$Else \rightarrow EXPINDICATE = 1;$
$SHAMT = | \, Exponent_A - Exponent_B | \quad //Shift \ \ Amount$
```

- **24-Bit Mux**: The 24-bit MUX is used to select the fraction part of the input. $muxout_x$ is the output of the mux. If number is non -zero then the 24 bits (MSB) '1' is prefixed, else '0' is the prefix
  Pseudo Code Logic:

```
$If \ SmallerInput \ != \ 0; \qquad //Smaller \ Input \ \ is \ non-zero$
    $muxout_1 = \{1, Fraction_{smaller \ input}\}$
$Else \ If \ SmallerInput == 0; \quad //Smaller \ Input \ \ is \ zero$
    $muxout_1 = \{0, Fraction_{smaller \ input}\}$

$If \ BiggerInput \ != \ 0; \qquad //Bigger \ Input \ \ is \ non-zero$
    $muxout_2 = \{1, Fraction_{bigger \ input}\}$
$Else \ If \ BiggerInput == 0; \quad //Bigger \ Input \ \ is \ zero$
    $muxout_2 = \{0, Fraction_{bigger \ input}\}$
```

- **Shift Right Fraction**: The output from the 24-bit MUX is shifted right by amount specified by '$SHAMT$' (shift Amount).
  Pseudo Code Logic:
  ```
  $Fraction_{out} = muxout_1 \gg SHAMT \quad //Right\ \ Shift$
  ```

- **8-Bit Mux**: The MUX is used to select the higher Exponent value of the two input exponents.
  Pseudo Code Logic:
  ```
  $If\ \ EXPINDICATE == 0 \rightarrow EXP_{OUT} = EXPONENT_A;$
  $Else \rightarrow EXP_{OUT} = EXPONENT_B;$
  ```

- **Input Compare:** The absolute value of two inputs A & B are compared.
  Pseudo Code Logic:
  ```
  $If\ A[30{:}0] \geq B[30{:}0] \rightarrow bigIndiccate = 0;$
  $Else \rightarrow bigIndiccate = 1;$
  ```

- **Intermediate Processing**
  Pseudo Code Logic:
  ```
  $assign\ A_s = (expIndicate)?\ Fraction_{out}{:}muxout2;$
  $assign\ B_s = (expIndicate)?\ muxout2{:}Fraction_{out};$
  ```

- **Big ALU:** Add and Subtraction operation is performed on the processed inputs. $A_s\ and\ B_s$ are the two 24 bits inputs to Big ALU. The output of Big ALU is $sum_{out}[24{:}0]$ 25 bits. $sum_{out}[24] = carry\ out$
  Pseudo Code Logic:
  ```
  $If\ inputs\ are\ opposite\ sign$:
      $sum_{out} = A_s + B_s$
      $sign_{out} = sign_A$
  $Else\ If\ inputs\ are\ opposite\ sign$:
      $sum_{out} = |A_s - B_s|$
      $sign_{out} = (bigIndiccate)?\ sign_B{:}sign_A;$
  ```

- **Rounding/ Normalizing Hardware**: The output from BIG ALU is converted to normalized IEEE 754 format. It also handles output exceptions like **overflow** and **underflow** cases.
  Pseudo Code Logic:
  ```
  $Case\ 1$:        $If\ (sum_{out} == 0)$:        $//Result\ \ is\ \ zero$
                        $sum_{out} = 0$
                        $EXP_{out} = 0$
                        $Sign_{out} = 0$
  ```

```
$Case\ 2$:        $If\ Carry\ Generated$:        $//Carry\ is\ generated$
                      $sum_{out} = sum_{out} \gg 1$
                      $EXP_{out} = EXP_{out} + 1$

$Case\ 3$:        $If\ (carry\ not\ Generated)$:     $//\ Sum\ is\ not\ normalized$
                      $While((sum_{out}[23] == 0))$:
                          $sum_{out} = sum_{out} \ll 1;$
                          $EXP_{out} = EXP_{out} - 1$

$Case\ 4$:        $If\ (EXP_{out} == 255)$:
                  $//\ Overflow\ Condition\ Output\ is\ \infty$
                          $sum_{out} = 0$
                          $EXP_{out} = 0xFF$

$Case\ 5$:        $If\ (EXP_{out} == 0)$:
                  $//\ Underflow\ Condition\ Output\ is\ 0$
                          $sum_{out} = 0$
                          $EXP_{out} = 0$
                          $Sign_{out} = 0$
```

Input Exception Handling:

$$If\ (any\ one\ of\ input\ or\ both\ inputs) = \pm\infty \rightarrow output = \pm\infty.$$

$$If\ \ Input\ A = \pm\infty\ and\ Input\ B = \mp\infty \rightarrow output = 0$$

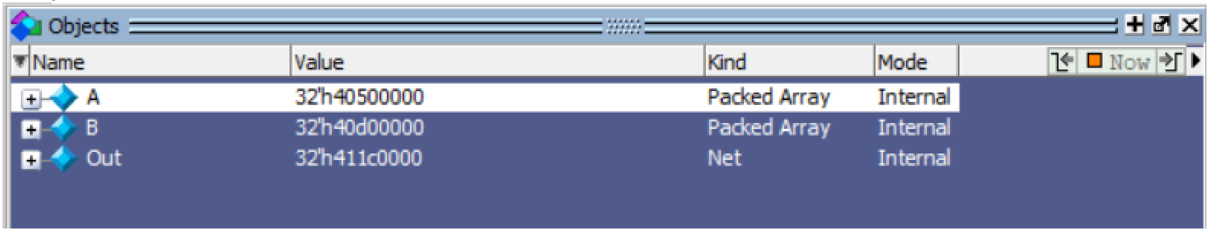
Testcase -1 : A = 3.25 B = 6.5 Output = 9.75

IEEE-754 Representation:

A = 32'b01000000010100000000000000000000; // 0x40500000

B = 32'b01000000110100000000000000000000; //0x40d00000

Output = 32’b01000001000111000000000000000000; //0x411c0000



```
Objects
Name    Value          Kind           Mode
A       32'h40500000   Packed Array   Internal
B       32'h40d00000   Packed Array   Internal
Out     32'h411c0000   Net            Internal
```

Testcase -2 : A = 4 B = 6 Output = 10

IEEE-754 Representation:

A = 32'b01000000100000000000000000000000; // 0x40800000

B = 32'b01000000110000000000000000000000; // 0x40c00000

Output = 32’b01000001001000000000000000000000; //0x41200000

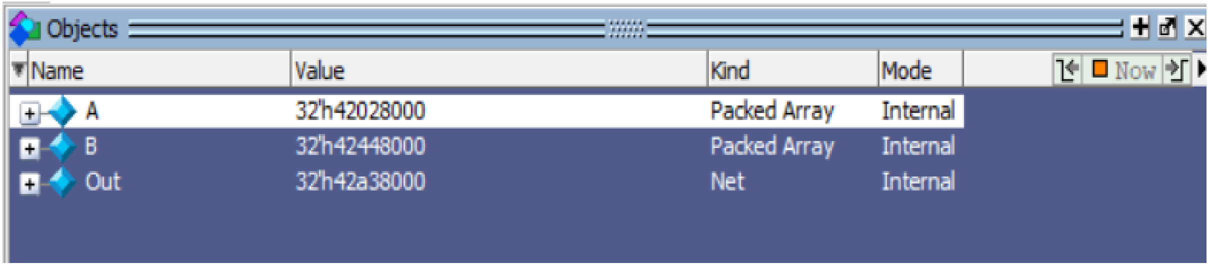
Testcase -3 : A = 32.625 B = 49.125 Output = 81.75

IEEE-754 Representation:

A = 32'b01000010000000101000000000000000; // 0x42028000

B = 32'b 01000010010001001000000000000000; // 0x42448000

Output = 32’b 01000010101000111000000000000000; // 0x42a38000



```
Objects
Name    Value          Kind           Mode
A       32'h42028000   Packed Array   Internal
B       32'h42448000   Packed Array   Internal
Out     32'h42a38000   Net            Internal
```

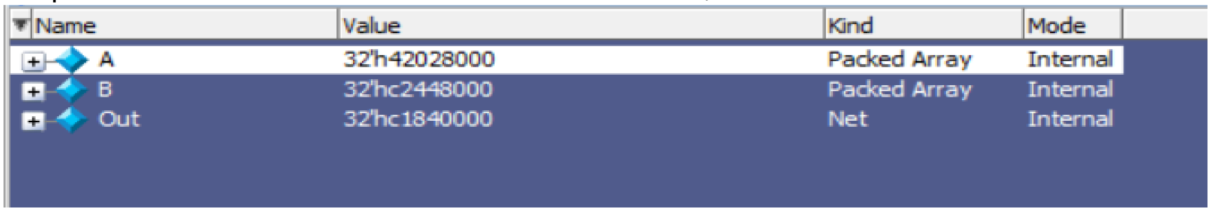
Testcase -4 : A = 32.625 B = -49.125 Output = -16.5

IEEE-754 Representation:

A = 32'b01000010000000101000000000000000; // 0x42028000

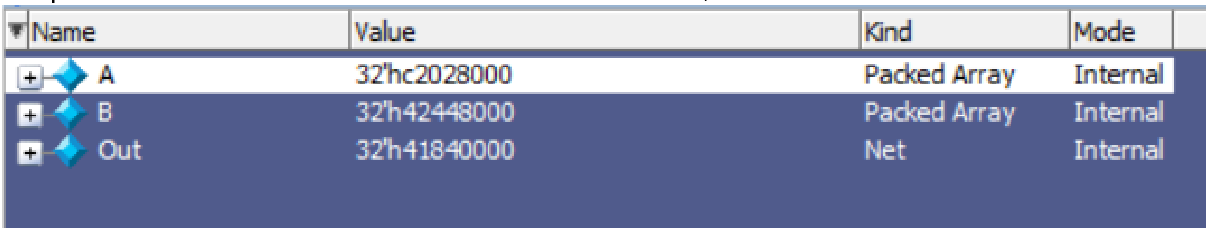
B = 32'b 11000010010001001000000000000000; // 0xc2448000

Output = 32’b11000001100001000000000000000000; // 0xc1840000



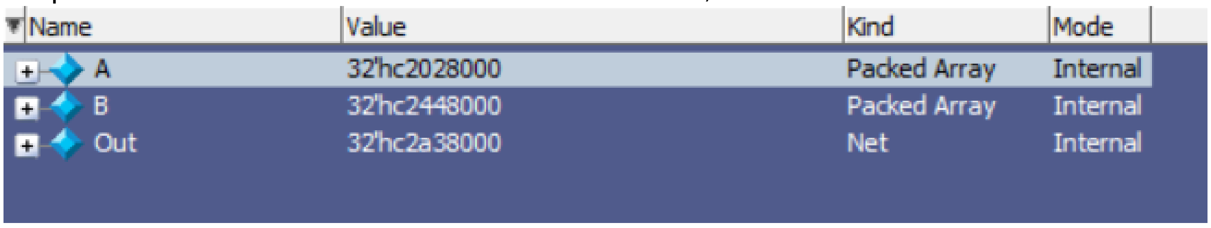
```
Name    Value          Kind           Mode
A       32'h42028000   Packed Array   Internal
B       32'hc2448000   Packed Array   Internal
Out     32'hc1840000   Net            Internal
```

Testcase -5 : A = -32.625 B = 49.125 Output = 16.5
IEEE-754 Representation:
A = 32'b11000010000000101000000000000000; // 0xc2028000
B = 32'b 01000010010001001000000000000000; // 0x42448000
Output = 32'b01000001100001000000000000000000; // 0x41840000



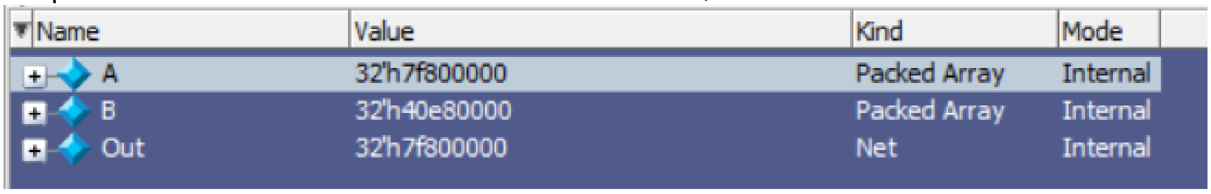
Name	Value	Kind	Mode
A	32'hc2028000	Packed Array	Internal
B	32'h42448000	Packed Array	Internal
Out	32'h41840000	Net	Internal

Testcase -6: A = -32.625 B = -49.125 Output = -81.75
IEEE-754 Representation:
A = 32'b11000010000000101000000000000000; // 0xc2028000
B = 32'b 11000010010001001000000000000000; // 0xc2448000
Output = 32'b01000001100001000000000000000000; // 0xc2a38000



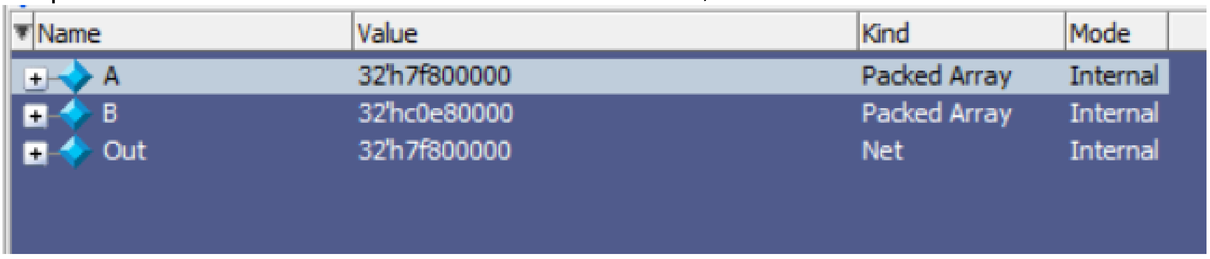
Name	Value	Kind	Mode
A	32'hc2028000	Packed Array	Internal
B	32'hc2448000	Packed Array	Internal
Out	32'hc2a38000	Net	Internal

Edge Cases
Testcase -7: A = ∞ B = 7.25 Output = ∞
IEEE-754 Representation:
A = 32'b01111111100000000000000000000000; // 0x7f800000
B = 32'b'01000000111010000000000000000000; //0x40e80000
Output = 32'b01111111100000000000000000000000; // 0x7f800000



Name	Value	Kind	Mode
A	32'h7f800000	Packed Array	Internal
B	32'h40e80000	Packed Array	Internal
Out	32'h7f800000	Net	Internal

Testcase -8: A = ∞ B = -7.25 Output = ∞
IEEE-754 Representation:
A = 32'b01111111100000000000000000000000; // 0x7f800000
B = 32'b'11000000111010000000000000000000; // 0xc0e80000
Output = 32'b01111111100000000000000000000000; // 0x7f800000

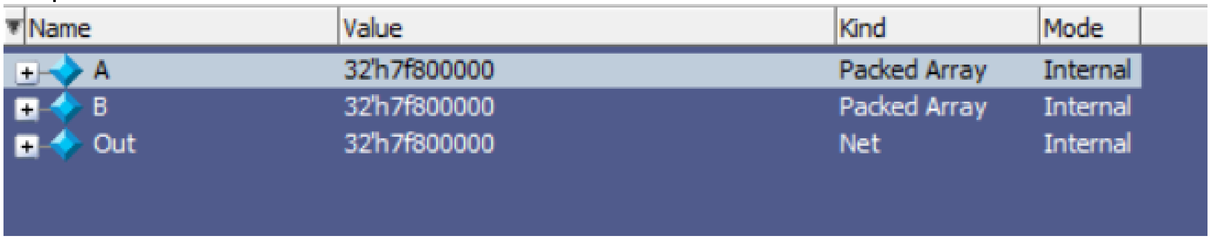


Name	Value	Kind	Mode
A	32'h7f800000	Packed Array	Internal
B	32'hc0e80000	Packed Array	Internal
Out	32'h7f800000	Net	Internal

Testcase -9: A = ∞ B = ∞ Output = ∞

IEEE-754 Representation:

A = 32'b01111111100000000000000000000000; // 0x7f800000
B = 32'b01111111100000000000000000000000; // 0x7f800000
Output = 32'b01111111100000000000000000000000; // 0x7f800000

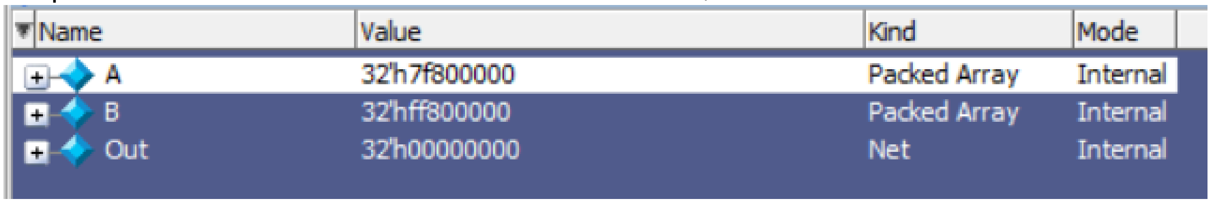


Name	Value	Kind	Mode
A	32'h7f800000	Packed Array	Internal
B	32'h7f800000	Packed Array	Internal
Out	32'h7f800000	Net	Internal

Testcase -10: A = ∞ B = −∞ Output = 0

IEEE-754 Representation:

A = 32'b01111111100000000000000000000000; // 0x7f800000
B = 32'b11111111100000000000000000000000; // 0xff800000
Output = 32'b00000000000000000000000000000000; // 0x00000000

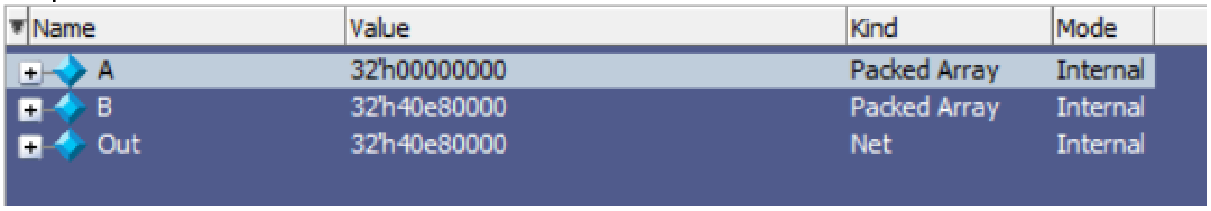


Name	Value	Kind	Mode
A	32'h7f800000	Packed Array	Internal
B	32'hff800000	Packed Array	Internal
Out	32'h00000000	Net	Internal

Testcase -11: A = 0 B = 7.25 Output = 7.25

IEEE-754 Representation:

A = 32'b 00000000000000000000000000000000; // 0x00000000
B = 32'b' 01000000111010000000000000000000; // 0x40e80000
Output = 32'b' 01000000111010000000000000000000; // 0x40e80000

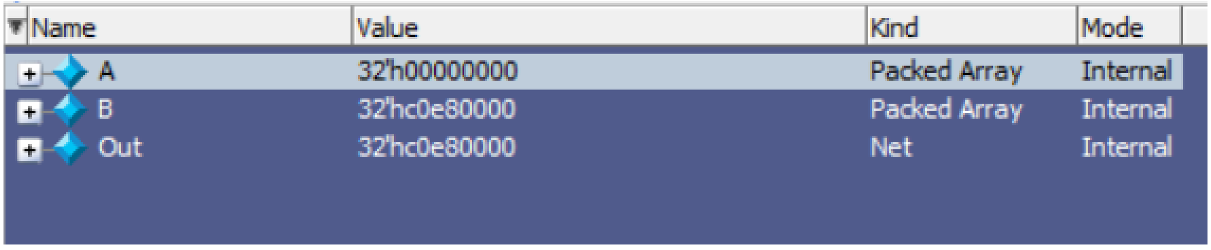


Name	Value	Kind	Mode
A	32'h00000000	Packed Array	Internal
B	32'h40e80000	Packed Array	Internal
Out	32'h40e80000	Net	Internal

Testcase -12: A = −7.25 B = 0 Output = -7.25

IEEE-754 Representation:

A = 32'b' 11000000111010000000000000000000; // 0x c0e80000
B = 32'b 00000000000000000000000000000000; // 0x00000000
Output = 32'b' 11000000111010000000000000000000; // 0x c0e80000



Name	Value	Kind	Mode
A	32'h00000000	Packed Array	Internal
B	32'hc0e80000	Packed Array	Internal
Out	32'hc0e80000	Net	Internal

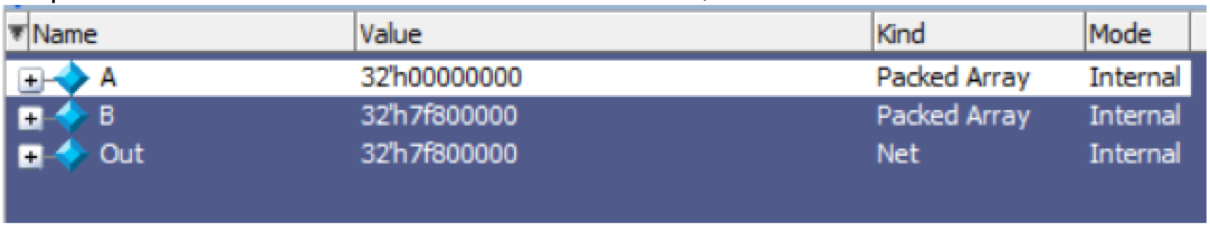
Testcase -13: A = 0 B =-∞ Output = −∞

IEEE-754 Representation:

A = 32'b 00000000000000000000000000000000; // 0x00000000

B = 32'b11111111100000000000000000000000; // 0xff800000

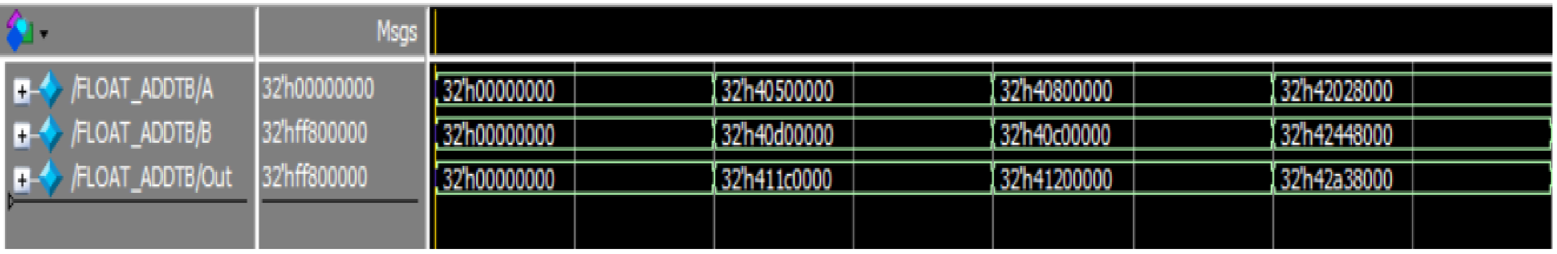
Output = 32'b11111111100000000000000000000000; // 0xff800000



Name	Value	Kind	Mode
A	32'h00000000	Packed Array	Internal
B	32'h7f800000	Packed Array	Internal
Out	32'h7f800000	Net	Internal

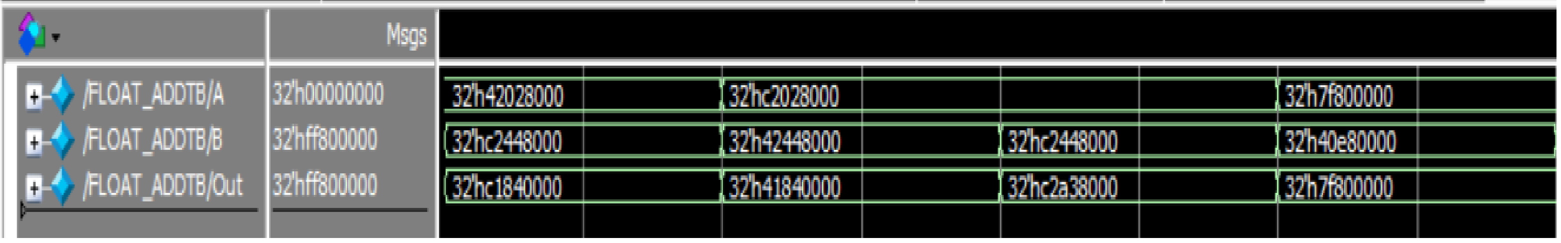
Testbench Run:

Test Cases 0-3



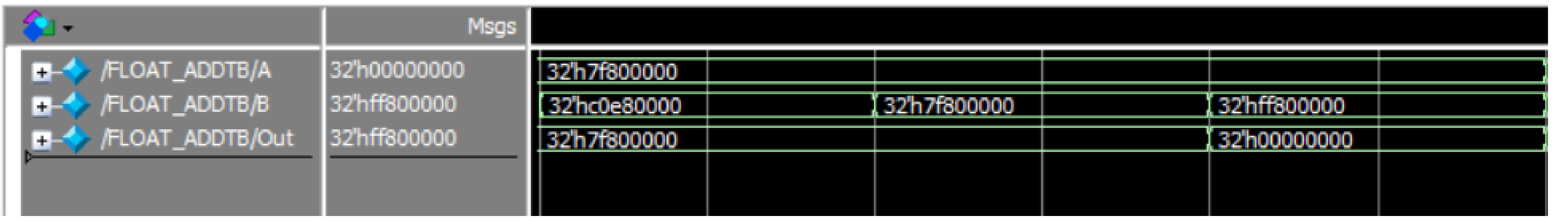
```
                 Msgs
/FLOAT_ADDTB/A   32'h00000000   32'h00000000  32'h40500000  32'h40800000  32'h42028000
/FLOAT_ADDTB/B   32'hff800000   32'h00000000  32'h40d00000  32'h40c00000  32'h42448000
/FLOAT_ADDTB/Out 32'hff800000   32'h00000000  32'h411c0000  32'h41200000  32'h42a38000
```

Test Cases 4-7



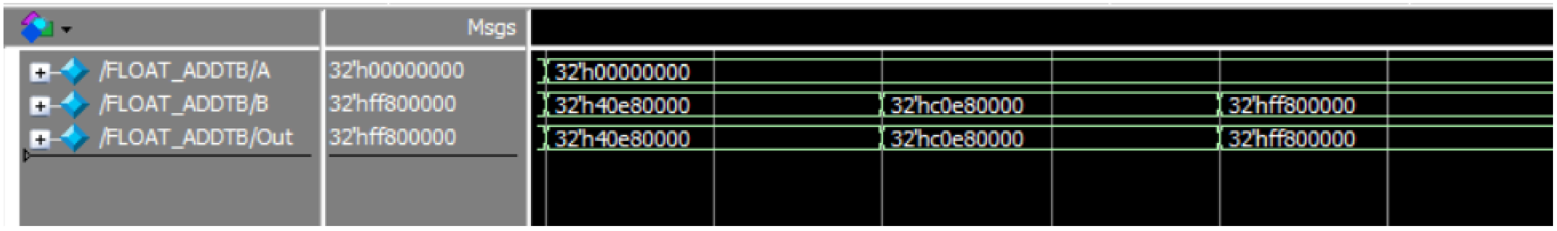
```
                 Msgs
/FLOAT_ADDTB/A   32'h00000000   32'h42028000  32'hc2028000                32'h7f800000
/FLOAT_ADDTB/B   32'hff800000   32'hc2448000  32'h42448000  32'hc2448000  32'h40e80000
/FLOAT_ADDTB/Out 32'hff800000   32'hc1840000  32'h41840000  32'hc2a38000  32'h7f800000
```

Test Cases 8-10



```
                 Msgs
/FLOAT_ADDTB/A   32'h00000000   32'h7f800000
/FLOAT_ADDTB/B   32'hff800000   32'hc0e80000  32'h7f800000  32'hff800000
/FLOAT_ADDTB/Out 32'hff800000   32'h7f800000                32'h00000000
```

Test Cases 11-13



```
                 Msgs
/FLOAT_ADDTB/A   32'h00000000   32'h00000000
/FLOAT_ADDTB/B   32'hff800000   32'h40e80000  32'hc0e80000  32'hff800000
/FLOAT_ADDTB/Out 32'hff800000   32'h40e80000  32'hc0e80000  32'hff800000
```