CAD for VLSI Systems (CS6230)

Project: Pipelined double precision(fp64) floating point adder

-EE20b072 & EE20B121

Theory:

IEEE 754 double-precision binary floating-point format:
Double-precision binary floating-point is a commonly used format on PCs, due to its wider range over single-precision floating point, in spite of its performance and bandwidth cost.

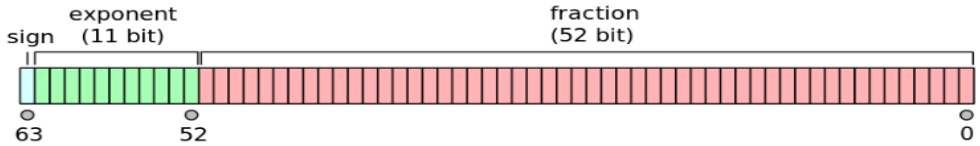
The IEEE 754 standard specifies a binary64 as having:

- Sign bit: 1 bit
- Exponent: 11 bits
- Significand precision: 53 bits (52 explicitly stored)

The sign bit determines the sign of the number.

The exponent field is an 11-bit unsigned integer from 0 to 2047, in biased form: an exponent value of 1023 represents the actual zero. Exponents range from −1022 to +1023 because exponents of −1023 (all 0s) and +1024 (all 1s) are reserved for special numbers.

The 53-bit significand precision gives from 15 to 17 significant decimal digits precision ($2^{-53} \approx 1.11 \times 10^{-16}$).



The real value assumed by a given 64-bit double-precision datum with a given biased exponent e and a 52-bit fraction is

$$(-1)^{\text{sign}}(1.b_{51}b_{50}\ldots b_0)_2 \times 2^{e-1023}$$

Here, the 52-bit fraction is called mantissa, and form 1.mantissa is called the normalized form.

Addition of floating-point numbers:

- Choose the number with smallest exponent, shift its mantissa right by number of steps equal to the difference in exponents.
- Set the exponent of the result as the larger exponent.
- Perform addition on mantissas and determine the sign of the result as sign of the number with larger absolute value.
- Normalize the result mantissa if needed.

Code:

The above logic is implemented in the code with 4 pipelining stages as:

Stage 1: Compare the exponents and determine the number of shifts required to align the mantissa to make the exponents equal. Then, right-shift the mantissa of the smaller exponent by the required amount (alignment).

Stage 2: Compare the two aligned mantissas and determine which is the smaller of the two. Take 2's complement of the smaller mantissa if the signs of the two numbers are different. Then, add the two mantissas. (addition).

Stage 3: Determine the number of shifts required and the corresponding direction to normalize the result (normalization-1).

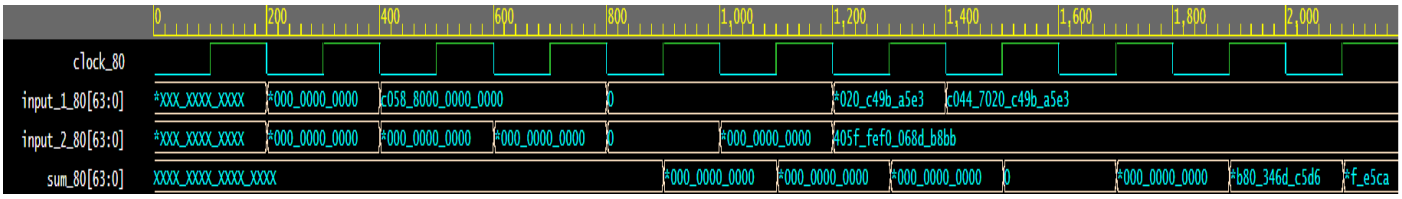
Stage 4: Shift the mantissa to the required direction by the required amount. Adjust the exponent accordingly, and check for any exceptional condition (normalization-2).

Results:

The outputs (with time) for the test cases from testbench respectively are shown below:

```
   0sum=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 901sum=0100000000110100000000000000000000000000000000000000000000000000
1101sum=0011111111100000000000000000000000000000000000000000000000000000
1301sum=1100000001011011000000000000000000000000000000000000000000000000
1501sum=0000000000000000000000000000000000000000000000000000000000000000
1701sum=1100000000100100000000000000000000000000000000000000000000000000
1901sum=0100000001100101000110111000000000110100011011011100010111010110
2101sum=0100000001010101110001101101111110100100001111111110010111001010
```

It can be noticed that there is a significant reduction in the time taken for the display of output because of pipelining in the design.



Results from synthesis in OpenLane:

=== mkFP_Adder64 ===

Number of wires: 5027

Number of wire bits: 5216

Number of public wires: 696

Number of public wire bits: 885

Number of memories: 0

Number of memory bits: 0

Number of processes: 0

Number of cells: 5083

Chip area for module '\mkFP_Adder64': 51384.281600 (in um^2)

Results from floorplan in OpenLane:

Initial floor plan core area: 5.52 10.88 331.66 334.56

Initial floor plan die area: 0.0 0.0 331.615 342.335