ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

NĂM HỌC 2018 – 2019

-------*-------

BÁO CÁO BÀI TẬP LỚN

MÔN : KỸ THUẬT SỐ NÂNG CAO

GVHD : TRẦN HOÀNG LINH

SVTH : VŨ ĐĂNG KHOA

MSSV: 1611645

TP HCM, 04/2019

I) Mục tiêu

Mục tiêu của bài tập lớn này là xây dựng một máy tính dấu chấm động FPU (Floating Point Unit) đơn giản, thực hiện các phép toán (+, -, *, /) giữa hai số float (định dạng IEEE754, single precision).
Ngõ vào: 2 số float 32-bit (A, B) và 2 bit lựa chọn phép toán (+, -, *, /).
Ngõ ra: Kết quả phép toán ở định dạng IEEE754, single precision.
Yêu cầu: Không được sử dụng các phép toán có sẵn trong Verilog/VHDL (như +, -, *, / và <<, >>), chỉ được sử dụng các lệnh logic (AND, OR, XOR, NOT).

II) Giới thiệu

1) Số float

Số float được dùng để chỉ một hệ thống biểu diễn số mà trong đó sử dụng một chuỗi chữ số (hay chuỗi bit) để biểu diễn một số hữu tỉ.
Thuật ngữ dấu phẩy động xuất phát từ chỗ hệ thống dấu phẩy động có dấu phẩy cơ số (tức là dấu phẩy thập phân trong trường hợp dùng hệ thập phân thường ngày hoặc là dấu phẩy nhị phân trong trường hợp dùng bên trong máy tính) không cố định mà có thể thay đổi vị trí của nó bất kỳ đâu trong các chữ số có nghĩa của số cần được biểu diễn. Vị trí này được mô tả một cách độc lập trong biểu diễn cụ thể của từng số. Đã có nhiều hệ thống dấu phẩy động khác nhau được dùng trong máy tính; tuy nhiên, vào khoảng hai mươi năm trở lại đây thì hầu hết các máy tính đều dùng cách biểu diễn tuân thủ theo chuẩn IEEE 754.

2) Chuẩn IEEE754

Số float theo chuẩn IEEE754 được biểu diễn thành một chuỗi bit. Chuỗi bit này là tập hợp của ba phần tử: phần bit dấu (sign bit), phần bit mũ (exponent bit), phần bit định trị (mantissa/significand bit). Độ chính xác đơn (single precision) của chuẩn IEEE754 được biểu diễn theo như hình 1.

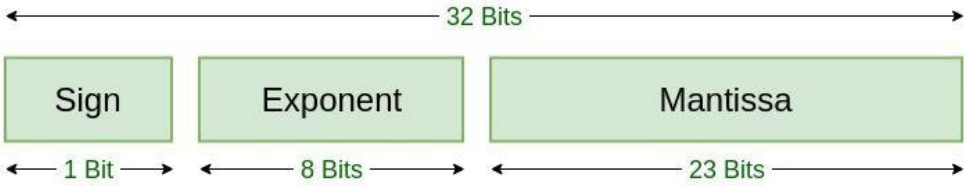

Figure 1: Biểu diễn dấu chấm động chính xác đơn

Công thức biểu diễn single precision:

$$n = (-1)^s 2^{e-127}(1.f) \qquad (1)$$

Với 0 < e < 255 và $f \neq 0$.

3) Bộ ALU

Bộ ALU (Arithmetic Logic Unit) là bộ logic thực hiện tất cả các phép toán số học và phép toán logic. Bộ ALU có vai trò quan trọng trong tất cả các CPU. ALU dựa vào chuỗi bit phép toán (operation bit) mà thực hiện phép toán và đưa ra kết quả. Ngoài ra, bộ ALU còn đưa ra các status bit cho những trường hợp ngoại lệ khi thực hiện phép toán. Ví dụ như: kết quả bằng 0, tràn trên (overflow), tràn dưới (underflow), chia cho 0, v.v.

4) Pipelining

Trong việc xây dựng bộ FLU, nhóm đã chọn thiết kế theo hướng pipelining. Có nghĩa là các module được sắp xếp theo một trật tự mà kết quả đầu ra của một module này là đầu vào của một module khác, cho đến khi ra kết quả cuối cùng. Pipelining giúp đẩy nhanh tốc độ thực hiện phép toán, giảm thiểu delay và dễ dàng hơn khi nâng cấp. Ngoài ra, pipelining còn có thể được sử dụng để thực hiện nhiều phép toán cùng một lúc song song với nhau (nếu threaded).

III) Thiết kế

1) Thiết kế tổng quát

Ngõ vào: 2 số float 32 bits (A, B) và 2 bit lựa chọn phép toán (Op), và xung clock.
Ngõ ra: 1 số float 32 bits (Out)
Thiết kế tổng quát được biểu diễn như theo hình 2.

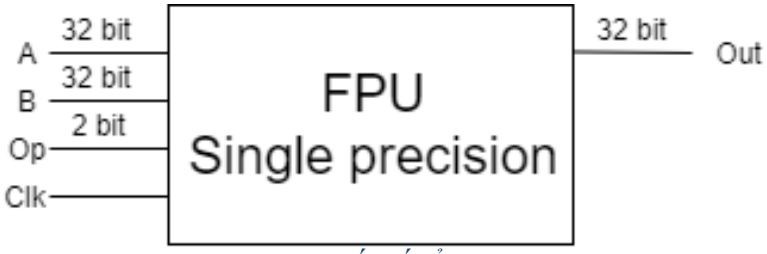

Figure 2: Thiết kế tổng quát của bộ FPU

2) Top module

Top module là module đầu tiên của bộ FPU. Module này dựa vào 2 bit Op để chọn phép toán và từ đó xử lý các trường hợp ngoại lệ (exceptions). Nếu có trường hợp ngoại lệ xảy ra, module ngay lập tức cho ra kết quả cuối cùng (giảm thiểu tốc độ tính toán).
Bảng giá trị của Op được biểu diễn như theo bảng 1.

Operation bits [1:0]	ALU Operation
00	Cộng [Addition] (+)
01	Trừ [Subtraction] (-)
10	Chia [Division] (/)
11	Nhân [Multiplication] (*)

Table 1: Bảng giá trị Operation bits

Nếu các trường hợp ngoại lệ xảy ra, module sẽ cho ra kết quả dựa theo code đã viết sẵn mà không raise các status bit.
Nếu không có trường hợp ngoại lệ xảy ra, module sẽ cho đưa kết quả cho module tính toán tiếp theo dựa theo operation bits.
Thiết kế của top module được biểu diễn như theo hình 3.

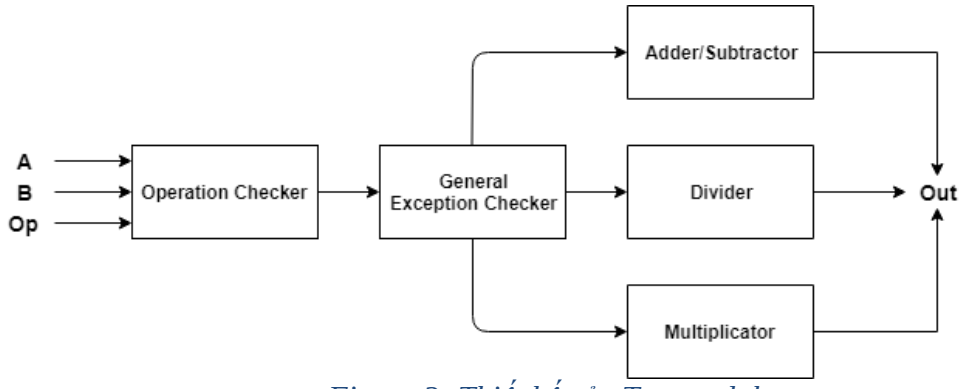

Figure 3: Thiết kế của Top module

3) Module adder/subtractor

Vì phép toán cộng và trừ là gần giống nhau nên chúng được gộp chung lại thành một module, module này sẽ đơn giản được gọi là adder.

i) *Thuật toán*

Thuật toán cho module adder được biểu diễn như hình 4.

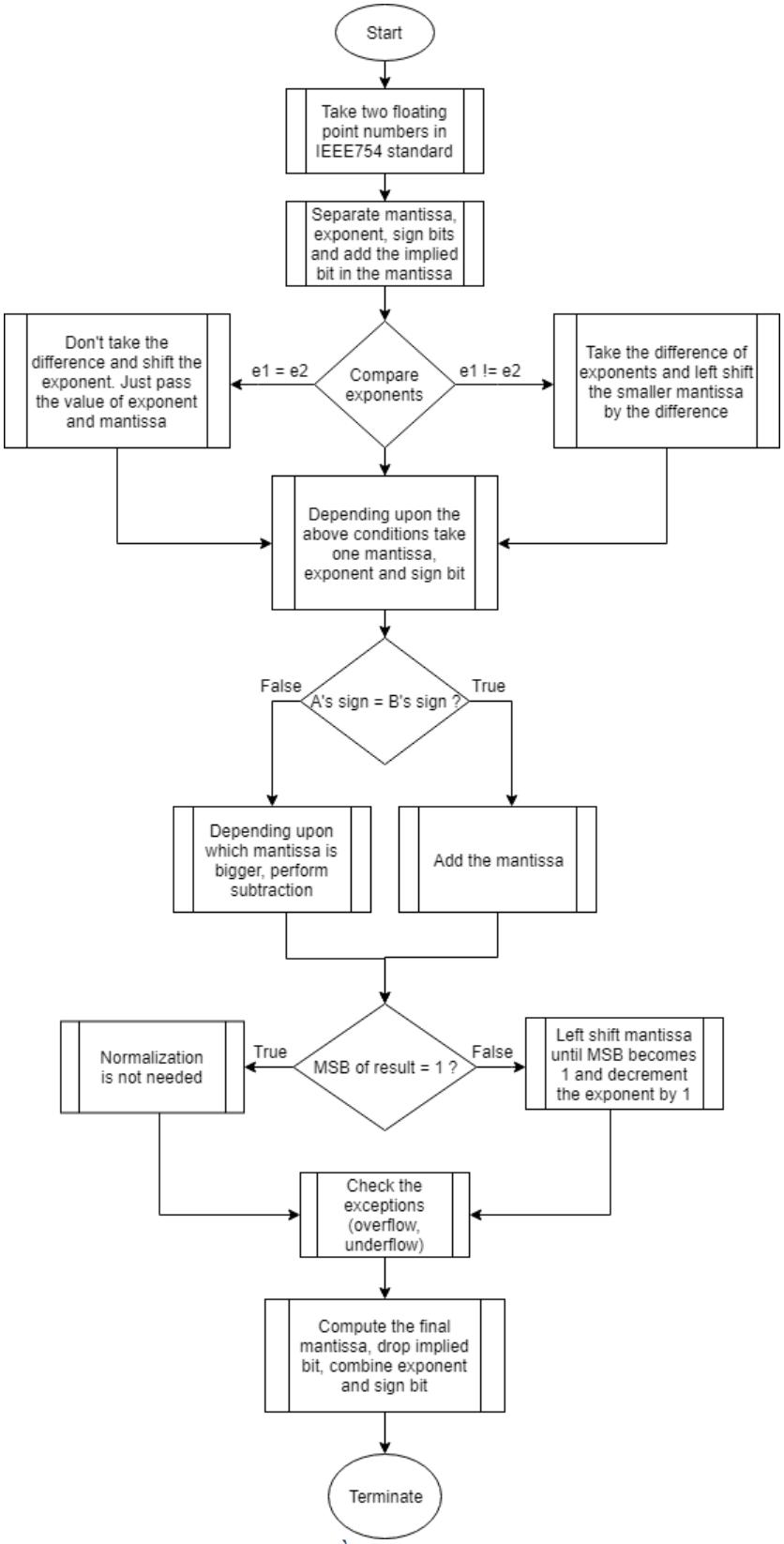

Figure 4: Sơ đồ thuật toán bộ cộng/trừ

ii) *Các bước thực hiện*

- Unpack module
  Module này tách các phần sign bit, exponent bit, và mantissa bit, đồng thời thêm impiled bit vào phần mantissa.
- Comparator và Barrel shifter
  Module này so sánh phần exponent và dịch bit phải exponent nhỏ hơn bằng độ lệch của chúng.
  - Barrel shifter: Bộ barrel shifter dịch bits hoàn toàn nhờ hệ tổ hợp sử dụng các bộ mux 2x1.
    VD: Bộ barrel shift 8 bit: sẽ có hai tín hiệu trung gian để dịch 4 bits và 2 bits, hai tín hiệu này phụ thuộc 2 giá trị S[2] và S[1]. Tín hiệu này sau đó sẽ được dịch lần nữa nhờ vào điều khiển của S[0].

```
int1  = IN       , if S[2] == 0
      = IN   << 4, if S[2] == 1
 int2  = int1     , if S[1] == 0
      = int1 << 2, if S[1] == 1
 OUT   = int2     , if S[0] == 0
      = int2 << 1, if S[0] == 1
```

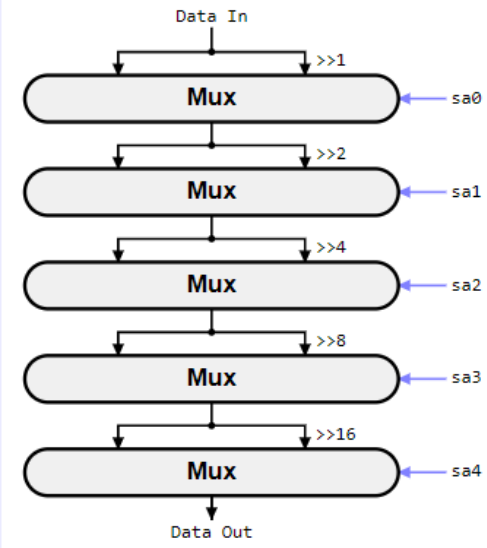

Figure 5: Bộ barrel shifter 32 bit

- Add/sub module
  Module này cộng hoặc trừ dựa vào bit dấu sử dụng bộ cộng toàn phần 24 bits.
- Normalize module
  Trong module này, kết quả được normalize bằng cách: nếu MSB của kết quả bằng 0, phần mantissa được dịch trái cho đến khi MSB bằng 1, và phần exponent bị trừ đi 1.
- Exception checker
  Module này kiểm tra các trường hợp overflow, underflow trước khi xuất ra kết quả cuối cùng.

- Packer
  Module này tổng hợp sign bit, exponent bit, mantissa bit, bỏ đi impiled bit và xuất ra kết quả.

4) Module divider

i) *Thuật toán*

Sơ đồ thuật toán cho module divider được biểu diễn như hình 6.

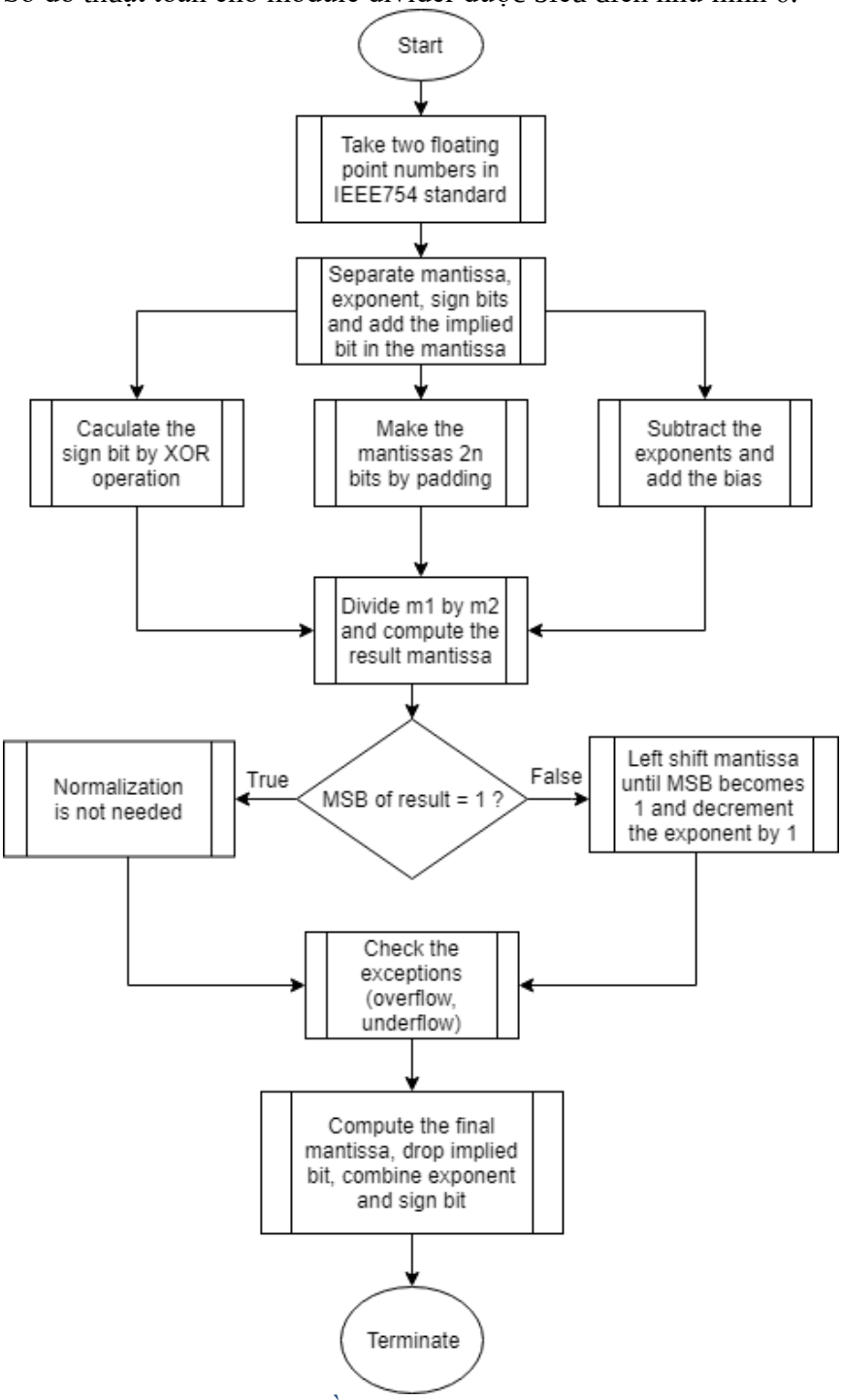

Figure 6: Sơ đồ thuật toán bộ chia

ii) *Các bước thực hiện*

- Unpack module
  Module này tách các phần sign bit, exponent bit, và mantissa bit, đồng thời thêm impiled bit vào phần mantissa.
- Divide module
  Module này thực hiện phép chia bằng phương pháp hồi phục (restore method). Đầu tiên cho thương (Quotient – Q) và phần dư (Remainder – R) bằng 0. Tiếp theo đặt LSB của R bằng MSB của A, và tính R – B. Nếu carry của bộ trừ bằng 1 (tức kết quả dương), thì trả về kết quả R – B, nếu carry của bộ trừ bằng 0 (tức kết quả âm), thì hồi phục lại kết quả, tức trả về R. Kết quả carry của bộ trừ được đặt vào LSB của Q và sau đó dịch trái 1 bit Q và R.
  Thực hiện phương pháp này 48 lần (cho single precision), chúng ta được kết quả cuối cùng out = Q[24:1], bit 1 của Q là bit guard, LSB của Q là bit sticky. Dựa vào 2 bit này có thể thực hiện làm tròn để được kết quả tốt hơn. Tuy nhiên trong việc xây dựng bộ divider này, nhóm đã quyết định sử dụng phương pháp Truncation, tức là bỏ qua làm tròn và chấp nhận sai số.
- Module trừ exponent
  Module này trừ phần exponent cho nhau và cộng lại cho bias (127 cho single precision)
- Sign calculator
  Module này tính bit dấu cho kết quả bằng cách XOR hai bit sign của hai operands.
- Normalize module
  Trong module này, kết quả được normalize bằng cách: nếu MSB của kết quả bằng 0, phần mantissa được dịch trái cho đến khi MSB bằng 1, và phần exponent bị trừ đi 1.
- Exception checker
  Module này kiểm tra các trường hợp overflow, underflow trước khi xuất ra kết quả cuối cùng.
- Packer
  Module này tổng hợp sign bit, exponent bit, mantissa bit, bỏ đi impiled bit và xuất ra kết quả.

5) Module multiplicator

i) *Thuật toán*

Sơ đồ thuật toán cho module multiplicator được biểu diễn như hình 7.

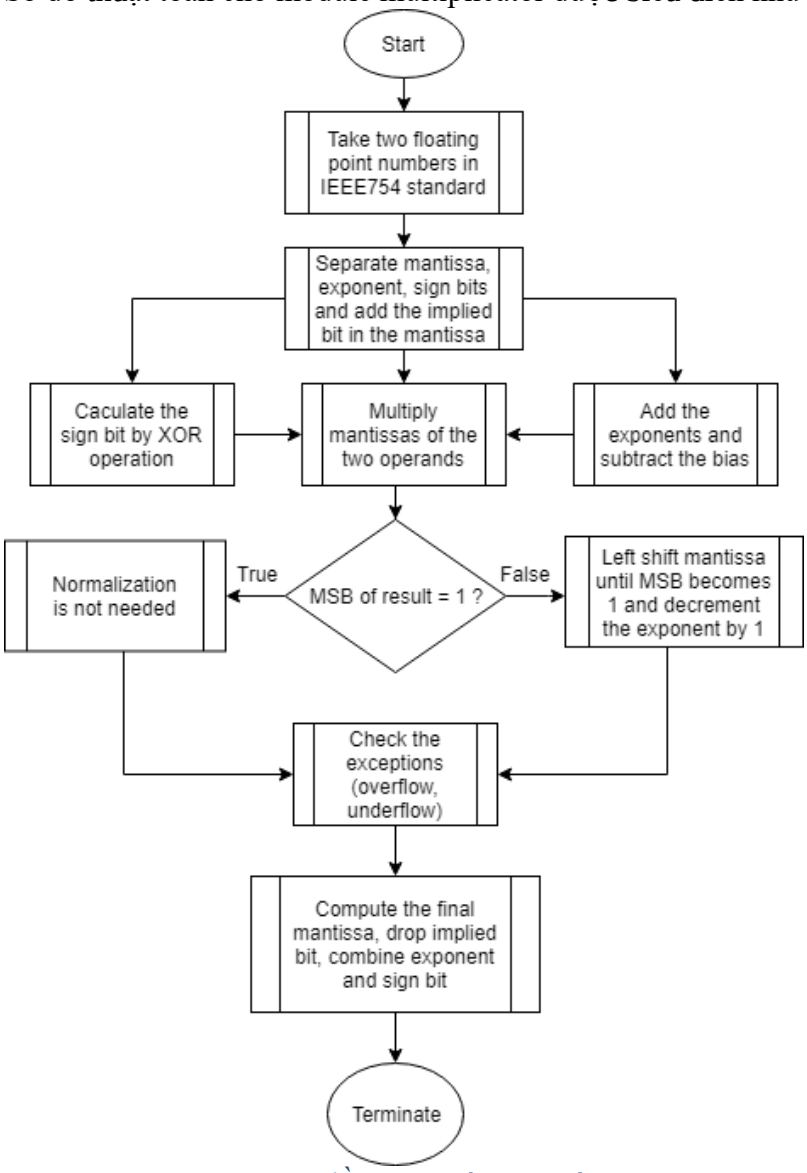

Figure 7: Sơ đồ thuật toán bộ nhân

ii) *Các bước thực hiện*

- Unpack module
  Module này tách các phần sign bit, exponent bit, và mantissa bit, đồng thời thêm impiled bit vào phần mantissa.
- Sign calculator
  Module này tính bit dấu cho kết quả bằng cách XOR hai bit sign của hai operands.
- Module cộng exponent
  Module này cộng hai phần exponent lại với nhau và trừ đi cho bias (127 cho single precision)

- Multiplier module
  Module này dùng thuật toán Dadda để nhân 2 số 24 bits lại với nhau, cho kết quả là số 48 bits.
  - Dadda multiplier
    Bộ Dadda multiplier được thiết kế bởi Luigi Dadda vào năm 1965. Nó gần giống với bộ Wallace multiplier nhưng nhanh hơn và cần ít cổng logic hơn.
    Bộ Dadda multiplier có ba bước chính giống như bộ Wallace multiplier:
    1. Nhân (cổng AND) từng bit của w1 với w2, được kết quả l1 và l2, gộp lại dùng weight theo từng cột.
    2. Giảm số được partial product từng stages sử dụng các bộ half-adder và full-adder cho đến khi chỉ còn lại 2 bits của mỗi weight.
    3. Cộng 2 bit lại sử dụng bộ cộng thông dụng.

    Hình 8 biểu diễn sơ đồ thuật toán Dadda multiplier.

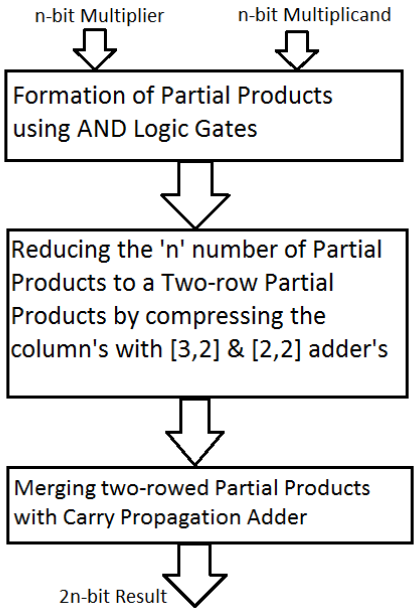

Figure 8: Sơ đồ thuật toán Dadda multiplier

Khác với bộ Wallace multiplier cố gắng giảm thiểu partial product theo từng stages thì bộ Dadda multiplier cố gắng giảm thiểu số lượng cổng logic sử dụng và thời gian delay giữa input và ouptut.

- Normalize module
  Trong module này, kết quả được normalize bằng cách: nếu MSB của kết quả bằng 0, phần mantissa được dịch trái cho đến khi MSB bằng 1, và phần exponent bị trừ đi 1.
- Exception checker
  Module này kiểm tra các trường hợp overflow, underflow trước khi xuất ra kết quả cuối cùng.
- Packer
  Module này tổng hợp sign bit, exponent bit, mantissa bit, bỏ đi impiled bit và suất ra kết quả.

IV) Code design và testbench:

Xem các file đính kèm: fpu.v, fpu_tb.v, multiplier.v, testgen.py

V) Waveform result:

Sử dụng phần mềm Modelsim 10.4a Student Edition (đã có hỗ trợ chuyển sang dạng float32/float64) để mô phỏng một số trường hợp được kết quả biểu diễn ở các hình 9, 10, 11, 12.

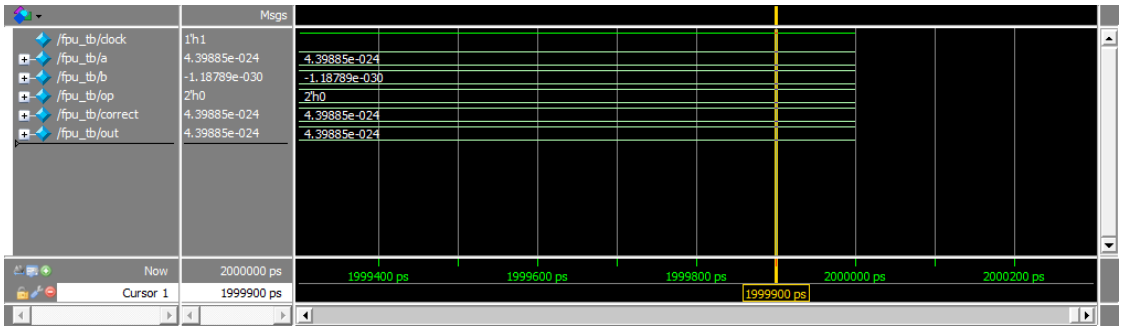

Figure 9: ADD Waveform

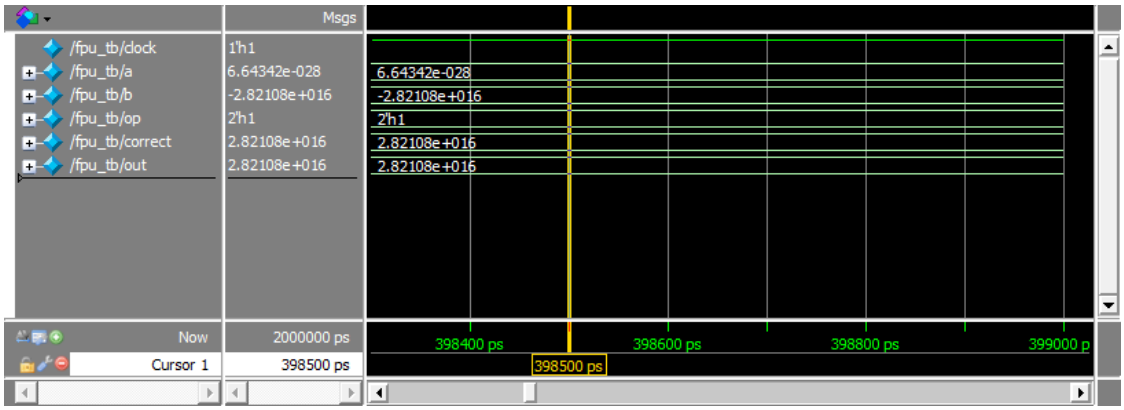

Figure 10: SUB Waveform

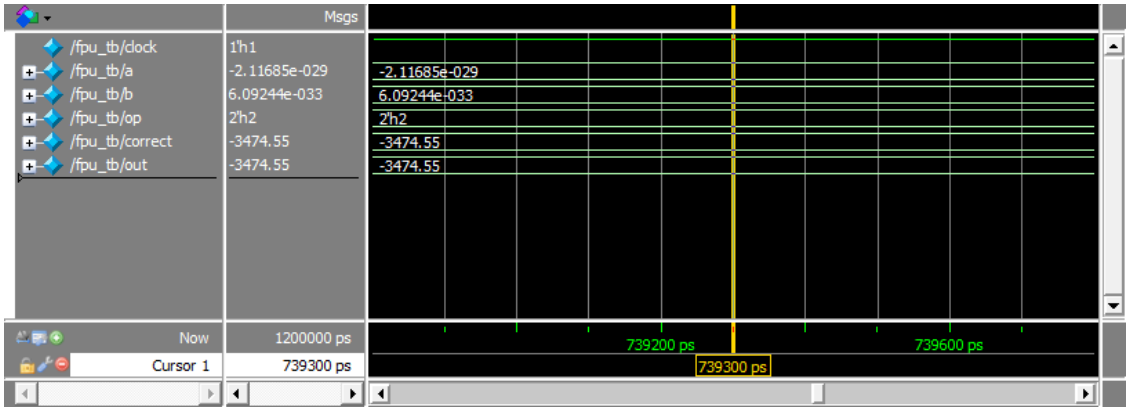

Figure 11: DIV Waveform

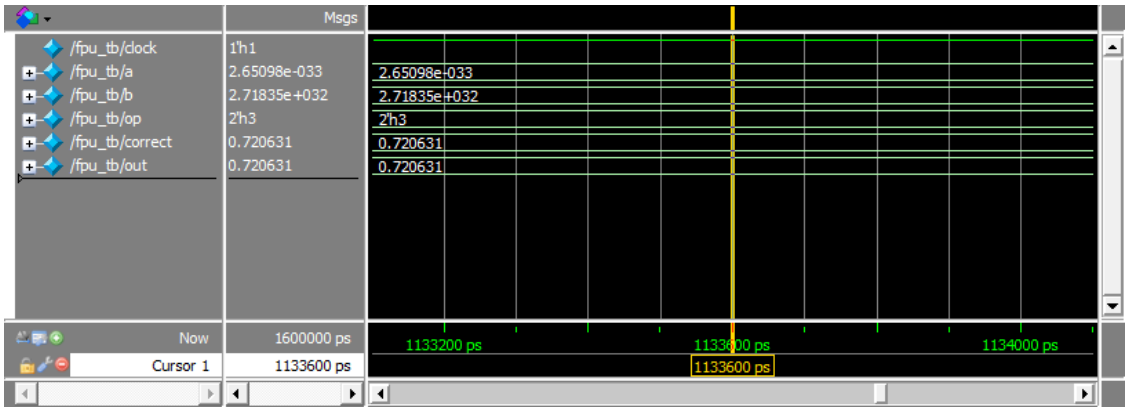

Figure 12: MULT Waveform

VI) Kết luận

Bộ FPU thực hiện thành công 4 phép tính cộng trừ nhân chia và cho kết quả tương đối chính xác. Code design và code testbench đã đạt được mục tiêu đặt ra ban đầu.