FPU Description

The FPU was designed by splitting the process of addition/multiplication into several modules and then implementing them using verilog. First, floating points A and B are unpacked, outputting the sign, exp, and mantissa of each float. The unpacker also outputs signals indicating when A/B are either NaN or Infinity.

For addition, there are modules for alignment, addition/subtraction, normalization, rounding, and result adjustment. Normalization and rounding have signals for overflow while add/sub and normalization have signals for underflow. These signals (in addition to the NaN and infinity signals from the unpacker) determine the final result for addition in the result adjustment module.

For multiplication, the process is split into modules for multiplication, normalization, rounding, and result adjustment. The rounding module used is the same one used for addition. The multiplier has signals for overflow and underflow, while normalization has a signal for overflow. The result adjustment takes in these signals and additionally checks for a denormal output and adjusts the result accordingly.

A single mux selects the final float value from the two result adjustment modules.

FPU Block Diagram

See fpuDiagram.pdf

Area/Delay

Area / Timing reports submitted on gradescope

Result Accuracy

```
narochon@ece006:~/private/18340/project3b$ ./nn_result_check.py golden_output/gold.txt nn_sim_results.txt
No. of errors: 0 / 150
narochon@ece006:~/private/18340/project3b$
```

Possible Improvements

With infinite time, one possible way to improve the design is by optimizing the adders used in the FPU. Pretty much all the adders used in the design were ripple carry adders, as such other adder designs such as carry-save or conditional sum could be used to optimize for either area or delay.

What I Learned

One interesting thing I learned when designing my FPU was when designing the adders used to add 1 to either the exponent in normalization/rounding or 1 to the significand when rounding. Since we are only adding 1 to the number, we can design a ripple-carry adder-like design using only half adders instead of full adders. This leads to reduced delay and area, since we are only using half as many gates for these specialized adders.

Extra Credit

To achieve an area smaller than 10000, multiple optimizations were made. First, the rounding modules were removed from the FPU to reduce area, this meant guard/sticky bits were no longer present for addition/subtraction. After this, the result adjustment modules for both addition and multiplication were removed to reduce area. Since we weren't adjusting the result anymore, Signals, for NaN, Infinity, Overflow, etc could be removed from the design, further reducing area. To significantly reduce area, computations were only performed with the 15 most significant bits of the significand (ie, bits [22:8] in a float of size [31:0]). The other 8 bits of the significands were ignored. This led to less accuracy but significantly reduced area and timing.

Extra Credit Area

Hierarchical cell	Global cell area: Absolute Total	Global cell area: Percent Total	Local cell area: Combinational	Local cell area: Noncombinational	Local cell area: Black-boxes	Design
FPU	9814.5000	100.0	2.0000	970.0000	0.0000	FPU
aNorm	897.0000	9.1	0.0000	0.0000	0.0000	nomalizeSum
aNorm/exp1	68.0000	0.7	0.0000	0.0000	0.0000	addOne8_1
aNorm/exp1/genblk1[0].genblk1.h0	8.5000	0.1	0.0000	0.0000	0.0000	halfAdder_60
aNorm/exp1/genblk1[0].genblk1.h0/a0						

Extra Credit Timing

```
clock network delay (ideal)                              0.00      100.00
Y_reg_reg[9]/clk (dff_1x)                                0.00      100.00 r
library setup time                                      -0.19       99.81
data required time                                                  99.81
--------------------------------------------------------------------------
data required time                                                  99.81
data arrival time                                                   -8.56
--------------------------------------------------------------------------
slack (MET)                                                         91.25
```

Extra Credit Accuracy

25 errors / 150 vectors = .166 = 83.3% Accuracy

```
Vector 37 neuron 7: Correct result 0.9141258001327515; Your result: 0.5
Vector 37 neuron 8: Correct result 0.12981659173965454; Your result: 0.46317392587661743
Vector 39 neuron 8: Correct result 0.12098684161901474; Your result: 0.46317392587661743
Vector 41 neuron 8: Correct result 0.11852546036243439; Your result: 0.46317392587661743
Vector 42 neuron 7: Correct result 0.7088561058044434; Your result: 0.5
Vector 42 neuron 8: Correct result 0.3600114583969116; Your result: 0.46317392587661743
Vector 44 neuron 8: Correct result 0.11624797433614731; Your result: 0.46317392587661743
Vector 46 neuron 7: Correct result 0.9302830696105957; Your result: 0.5
Vector 46 neuron 8: Correct result 0.1324843466281891; Your result: 0.46317392587661743
Vector 48 neuron 8: Correct result 0.11968895047903061; Your result: 0.46317392587661743
Vector 49 neuron 8: Correct result 0.11808428913354874; Your result: 0.46317392587661743
No. of errors: 25 / 150
```