A CMOS Floating Point Unit

17th Annual Student VLSI Design Contest
Experienced Class Entry

Michael J. Kelley
Matthew A. Postiff
Advisor: Richard B. Brown

University of Michigan
May 13, 1997

Abstract

In conjunction with the High-Performance Microprocessor Project at The University of Michigan, a PowerPC-compatible floating point unit (FPU) has been implemented. This paper presents the FPU, which is implemented in Hewlett Packard's HP14B 0.5 μm, 3M, 3.3V CMOS process. In order to achieve high performance, the FPU has a nine-stage pipeline and implements a subset of the most commonly used double-precision floating point instructions. Unimplemented instructions will be handled in software by the compiler and the operating system. The FPU achieves a 75 MHz clock frequency. The critical path is through a 54-bit shift unit. A full 64-bit multiplier tree based on 4:2 compressors computes a 106-bit sum and carry within one clock cycle. The 330,000 transistors in the FPU take up 56 mm^2. Power dissipation is estimated to be less than 8.5 W. This paper outlines the architecture of the floating point unit and describes some of our design methodology. Implementation issues and detailed statistics are given.

1. System Overview

The floating point chip described in this paper is designed to be paired with a companion fixed point unit, either on the same die or on a multi-chip module. The fixed point unit provides instruction fetch and data retrieval capability so that the majority of the area on the floating point chip can be devoted to units specific to floating point calculations.

To achieve a high clock frequency, the floating point unit is designed with a long pipeline (9 stages) and includes several unique features to increase performance. The tree multiplier is composed of custom standard cell 4:2 compressors which employ transmission gate logic. The wide compressors reduce the number of logic levels needed in the tree compared to a traditional Wallace tree design based on 3-2 adders. To further increase performance, we studied how to reduce the number of pipeline stages and still achieve a high clock rate. Our design implements a dual-path design for addition operations that allows the pipeline to be one stage shorter than it otherwise would have been.

The rest of this paper is divided into sections that describe the chip's architecture, various trade-offs made during the design, and implementation and testing issues. We then conclude and suggest some areas for future work.

2. Implementation and Engineering Considerations

2.1. Functional Specification

The FPU is designed to be PowerPC compatible with support from both the compiler and operating system, so it implements a small subset of the PowerPC floating point instruction set. Instructions with a high dynamic usage frequency in the SPECFP95 benchmarks were included in the implemented instruction set. The instructions that are rarely used or that can be handled easily in software by the compiler or the operating system have been removed from the instruction set. Table 1 summarizes the implemented FPU instructions. These instructions implement over 75%

Table 1: FPU Instruction Set Architecture

Mnemonic	Description	Mnemonic	Description
fadd	Add	fabs	Absolute value
fsub	Subtract	fmr	Move register
fmul	Multiply	fnabs	Negate absolute value
fmadd	Multiply-add	fneg	Negate
fmsub	Multiply-subtract	lfd	Load
fnmadd	Multiply-add negate	lfdx	Load indexed
fnmsub	Multiply-subtract negate	stfd	Store

of the instructions that occur in the SPECFP95 benchmarks, for example. Of the remaining 25%, 18% of those can easily be emulated by other instructions.

Further details of the requirements of a PowerPC floating point architecture can be found in [2].

2.2. Pipeline Organization

The pipeline which implements the above instructions utilizes a simple, deeply pipelined architecture in order to achieve high performance. The FPU is composed of 9 pipeline stages as shown in Figure 1. Stage 1 and stage 2 of the FPU are shared with a companion FXU described in [17]. Stage 1 increments the current program counter

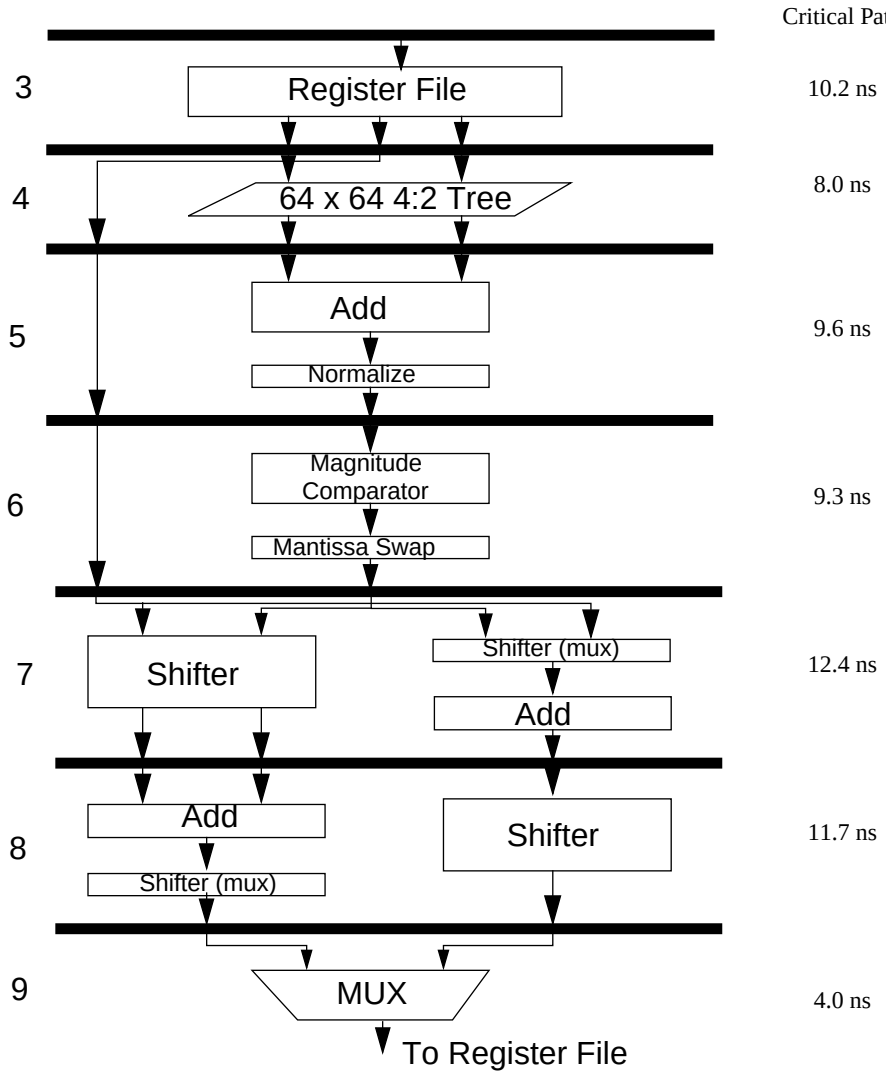

Figure 1. FPU block diagram. The heavy black lines represent pipeline latches.

(PC) and selects PC for the next instruction. Stage 2 accesses the shared L1 instruction cache.

Stages 3 through 9 compose the stages that execute floating point instructions. In stage 3, the register file is

accessed and the instruction is decoded. In addition, a simple scoreboard checks for data dependencies; if a dependency is detected, the FPU stalls until the dependent instruction finishes execution.

The remainder of the pipeline is similar to the IBM 603e floating point unit in that its operation is based on the flow of the fused multiply-add operation [19]. There are several advantages of the fused multiply-add operation which result from the operation not doing an intermediate round between the multiply and the add. Perhaps most obvious is that the result is more accurate because of the so-called *infinite-precision* intermediate result. This is important in applications that do, for instance, matrix multiplication where there is a long sequence of multiplications and additions. More subtle is the fact that performance is improved because of fewer overall rounding steps. Another advantage of the multiply-add operation is that it reduces overall register port pressure (there is one combined instruction instead of two, i.e. one write operation instead of two), though this advantage is lessened by the fact that the multiply-add instruction requires 3 read ports instead of the traditional 2 read ports.

Stage 4 of our pipeline contains the multiplier tree and exponent adder. The 53-bit significands are multiplied to the point where two 106-bit partial products remain. In stage 5, these final two partial products are added to form the result of the multiply which is also 106 bits in length. In order to calculate the sign and perform proper additions of floating point numbers, the operands to the floating point adder have to be aligned. Because this pipeline computes floating point results in one's complement arithmetic, stage 6 is necessary to compare the operands and possibly swap them if the operation is a subtract. Stages 7 and 8 perform the add/subtract operation by aligning the floating point operands through shifters, adding the aligned operands, and then normalizing the final result (how this works is described in the next paragraph). The final stage selects the correct value to write back to the register file.

A simple specification of the floating point pipeline for addition operations would call for three stages: a large preshift stage to align the summands, the add stage, and then a post-normalization stage. Closer inspection of the floating point addition operation reveals that the addition cannot have both a large pre-shift alignment (greater than 1 bit position) and a large post-shift normalization. The proof is by cases and is shown in Figure 2. Thus, by duplicating the adder and adding two one-bit shifters (muxes) we can implement the addition portion of the pipeline in two stages instead of three, because the small shift operations can be absorbed into the stage with the adder. As shown in Figure 1, stages 7 and 8 compute both cases and stage 9 selects the correct result. Implementing the addition this way only requires one additional 54-bit adder plus some muxes to do small shifts (shift of 0 or 1 bit for alignment

or normalization), resulting in a modest increase in transistor count and one less pipeline stage.

Case 1: Exponents equal	Case 2: Exponents differ by 1	Case 3: Exponents differ by > 1
1.fffff + 1.fffff --------- 11.fffff	1.fffff + 0.1ffff --------- 11.fffff	1.fffff + 0.001ff --------- 10.fffff
In this case, no pre-normalizing shift is necessary. The addition results in a floating point number which requires at most a 1-bit right shift.	Here, only a 1-bit right shift is needed to normalize the operands. If a carry propagates to the most significant bit, a 1-bit right shift may be necessary to normalize the result.	In this case, a large aligning preshift is necessary before the addition takes place. In the worst case, a carry could propagate to the most significant end and require a 1-bit right shift to normalize the result.

Figure 2. Proof that IEEE 754 floating point addition does not require both a large aligning preshift and a large normalizing post-shift. Inputs are assumed to be normalized. The proof for subtraction is analogous.

2.3. Architectural Trade-offs

One unique feature of the architecture is that the multiplier tree is implemented as a 64-bit tree as opposed to a 54-bit multiplier tree. The primary reason for implementing a larger tree is for future additions of SIMD instructions similar to Intel's MMX and Sun's VIS instructions [18]. In [20] we describe how a 64-bit multiplier can be slightly modified by produce multiple byte- and halfword-results simultaneously, the essence of SIMD operation. This type of addition to the architecture can increase the performance of audio, graphics, and video applications with proper device driver support. Since increasing the multiplier tree to 64-bits does not increase the number of levels of 4:2 compressors (5 levels), the delay through the tree will be roughly the same. It does increase transistor count and area.

Another trade-off had to be made concerning floating point subtractions. Using the IEEE 754 floating point standard, all floating point numbers are represented in a signed magnitude notation. Sign magnitude numbers are difficult to perform subtraction on, either requiring a separate subtract unit, or a conversion to a different binary representation of the number. Since an additional 53-bit subtraction unit would increase the area of the chip, the FPU converts all negative numbers to one's complement notation. One's complement requires a mantissa swap before subtraction. The area of the extra comparator is much smaller than an additional subtract unit.

In addition to the above trade-offs, the FPU also does not contain any forwarding paths. The majority of the

instructions in our target instruction mix (multiply-add instructions) require the full length of the pipeline to execute so forwarding busses will not increase performance. There are several short latency instructions (absolute value and move instructions in particular) which could benefit from short-circuiting, but we opted to leave the necessary 64-bit busses out of the pipeline to decrease area and reduce complexity. The additional muxing logic that would be needed in stage 4 of the pipeline for instructions dependent on these operations would also increase the clock period.

2.4. Design Methodology

Design entry for the FPU was in Verilog. Both behavioral and structural implementations were coded and tested by comparing results to known-good results computed on an existing IEEE-754 compliant floating point unit. For all tests, randomly chosen inputs were used. When this data was used as input to a randomized stream of instructions, the results frequently went out of bounds (either to +/- infinity) and thus we needed to inject new random (but well-formed) values into the simulation every few tens of cycles to avoid producing too many edge cases. We are still running millions of random test vectors through the models to catch any errors that were not caught by out focused tests. Upon completing these tests, we will also run real application code through the FPU to pick up any remaining errors.

Cascade's EPOCH toolset provided the 0.5 µm cell library and automated placement and routing. (The FPU is implemented in Hewlett Packard's HP14B process, a 0.5 µm, 3.3 V, 3 metal CMOS process). Mentor Graphics ICstation was used to design a custom 4:2 compressor which was imported into the EPOCH framework. The EPOCH static timing analyzer, TACTIC, was used to analyze delays through each pipeline stage. This allowed us to identify critical paths and reduce the delays through those paths. Cascade's Floorplanner was used to manually place some cell blocks for a more compact layout. The Cascade tool PDABS provided power driven buffer sizing: given our target clock cycle, this tool iterated over the cell blocks, attempting to properly size buffers so that the target would be met.

Once we are sure that the FPU functions correctly, final layout will be generated based on this "bug free" design. Finally, DRC, ERC, and LVS checks will be run on the entire circuit.

The remainder of this section describes some of the implementation issues that we examined during this design.

2.5. Multiplier Tree

The single-cycle 64-bit multiplier tree is similar in architecture to a traditional Wallace Tree, but is actually implemented with five levels of 4:2 compressors. The mux-based implementation of the compressor was adapted from [1], with each mux composed of two transmission gates and two inverters. The 4:2 compressor cell was laid out full-custom using Mentor Graphics ICStation (see Figure 3). Each compressor consists of 48 transistors. In-cell routing was confined to layer 1 and 2 metal so that metal 3 could be used for overcell routing. Transistor sizing is based on HSPICE simulations. A sample HPSICE run is shown in Figure 4. The critical path through the mux-based compressor is three 2-input mux delays; this simulation shows the critical path to be about 750 ps, or about 250 ps per mux. The inputs are driven by inverters (not ideal current sources) and all three outputs of the simulated compressor are loaded by inputs of three other compressors.

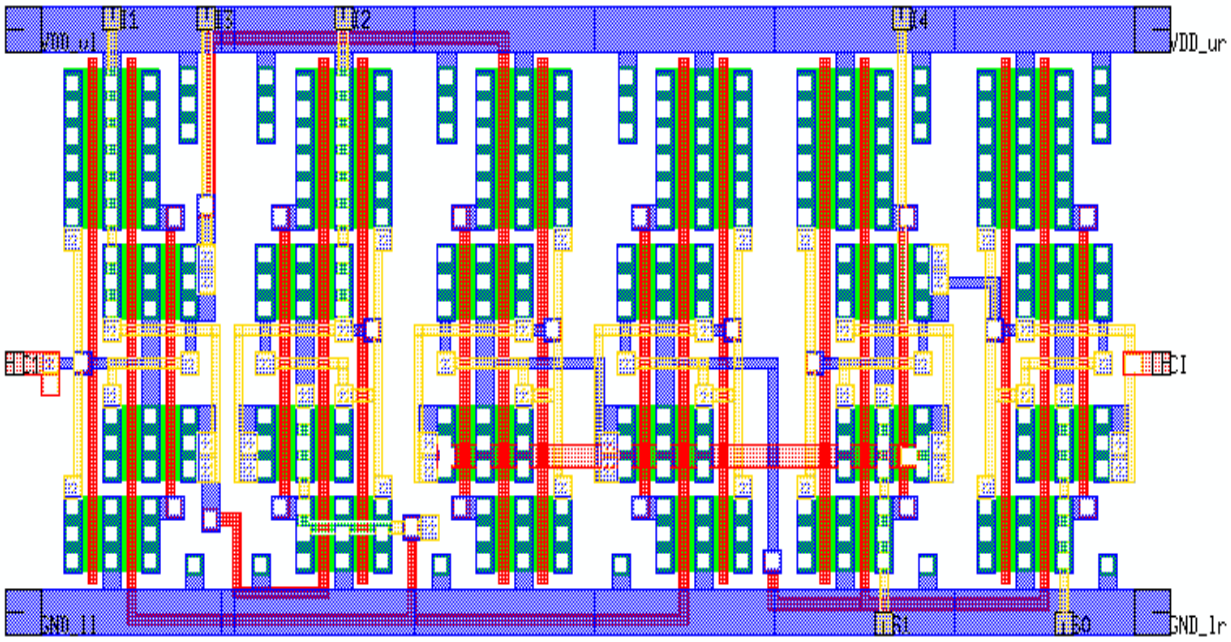

Figure 3. 4:2 Compressor Layout

In addition to the 4:2 compressors, the multiplier requires an AND plane which generates the partial products of the multiply. A custom 4:2 compressor with ANDing function was designed to combine both functions into one standard cell. The compressor cells were imported into EPOCH for placement and routing. The 64-bit multiplier tree consists of 2273 compressors and 4096 AND gates, for a total of 133,680 transistors. Due to the large number on interconnections within the multiplier tree, the majority of the area is consumed by the interconnect. The area of the tree is 5.545 mm x 3.681 mm (20.411 mm^2). The multiplier tree makes up nearly 36.4% of the total chip area. The

tree is estimated to dissipate 5.6 W and have a worse case delay of 7 ns. The multiplier tree layout can be seen at the top of the chip layout in Figure 5.

Note that the bit of each multiplicand fans out to 54 inputs in the tree. The capacitance on each net is therefore very large and a multi-stage buffer tree is required to drive the large fanout.

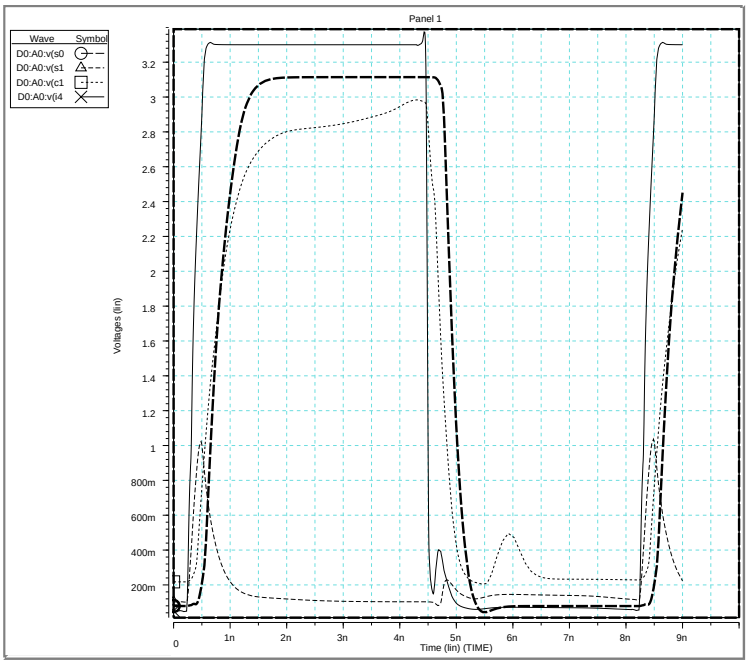

Figure 4. A spice simulation of the 4:2 compressor. The bold dashed curve shows the output response of the sum output s0 stimulated by a fall in the input i4 shown as the thin solid curve. The delay is about 750 ps.

2.6. Clocking and Timing

The FPU is clocked by a single-phase, 50% duty-cycle clock. All storage in the pipeline is based on positive edge triggered flip-flops. To prevent race conditions, short paths through pipeline stages have been delayed by the addition of buffers between the flip-flops. This effectively eliminates the short paths and thus reduces the possibility of race conditions occurring.

Initially, we had hoped for a fast clock rate (200 to 250 MHz), but shortly into the design, we noticed that our cell library components did not meet speed requirements. There are basically two critical paths that determine the clock frequency of the FPU. First is the path through the register file. This path goes through some decode logic, the register file, and then a mux. The delay through this path is 10.2 ns. The register file contributes 8.5 ns of that delay.

The other critical path is through the large pre-shift stage (7). This path goes through a small 11-bit adder and then through a 54-bit shifter. The delay through this logic is 12.4 ns. This sets the clock rate at 75 MHz (13 ns period) which is substantially below our target clock rate. We are currently redesigning the large pre-shift stage of the FPU to balance this delay with the others in the pipeline; we anticipate that the pipeline will run close to 100 MHz after this enhancement. Figure 1 summarizes the critical path delays through each stage of the FPU.

2.6.1. Design for Test and Testing Environment

The most important test element of the FPU is the full scan chain. All registers (excluding the register file) are connected to a scan chain. Using this scan chain, known inputs to all the stages can be scanned into the pipeline registers and after one clock tick, test results can be scanned back out of the FPU.

The packaged FPU will be testing using Hewlett Packard test equipment. The HP 82000 IC tester is capable of providing 240 signals at 200 MHz. An HP 80000 pulse generator will supply our clock signal.

2.7. Chip Statistics

Figure 5 shows the final layout of the FPU. The chip's critical statistics are shown in Table 2.

Table 2: Chip Statistics

Category	Value	Category	Value
Die size	7.5 mm x 9.0 mm	Core density	5980 transistors / mm^2
Core area	56 mm^2	Clock rate	75 MHz
Power Dissipation	8.3 W	Number of pins	244
Number of Transistors	334,903	Voltage supply	3.3V

3. Summary

This paper presented a 330,000 transistor floating point unit which implements a subset of the PowerPC floating point ISA. In designing the unit, we made use of transmission-gate logic and wide compressors in the multiplier tree, and showed how to implement the addition portion of the pipeline in two stages instead of the normal 3. While the part did not meet our speed goals, the layout area and power dissipation were satisfactory.

3.1. Future Work

Our primary items for work in the near future are to finish testing and increase the operating frequency of the part. A new cell library has just become available with faster cells. If those are not fast enough, we will revert to cus-

tom cell designs to increase the speed.

In addition to increasing the clock rate, there are additional instructions and components that were left out of this initial project that will be added back in. For instance, single precision operations and integer conversions can be added to the current pipeline with little modification. The FPU currently implements one rounding mode (round towards zero), the IEEE specification calls for 4 rounding modes. Finally, exception handling for NaNs, underflow and overflow needs to be added.

4. References

[1] Ohkubo N., Suzuki M., Shinbo T., Yamanaka T., Shimizu A., Sasaki K., and Nakagome Y., “a 4.4-ns CMOS 54X54-b Multiplier Using Pass-transistor Multiplexer,” IEEE 1994 Custom Integrated Circuits Conference, pp. 599-602, 1994.

[2] IBM and Motorola. “PowerPC Microprocessor Family: The Programming Environments.” MPRPPCFPE-01. IBM.

[3] IBM and Motorola. “PowerPC 603 RISC Microprocessor Users’s Manual.” MPR603UMU-01. IBM.

[4] IBM and Motorola. “PowerPC 604 RISC Microprocessor Users’s Manual.” MPR604UMU-01. IBM.

[5] T. Huff. “Architectural and Circuit Issues for a High Clock Rate Floating-Point Processor.” Technical Report No. SSEL-RBB-1-95. University of Michigan.

[6] H. Kim and T. Strong, “A Complementary GaAs 53-bit Parallel Array Floating Point Multiplier.” EECS 627 class Report. University of Michigan. December 9, 1995.

[7] M. Postiff, “PUMA Instruction Set Architecture.” Version 1.0.

[8] M. Riepe, T. Huff, T.N. Mudge, “Implementing IEEE Rounding in Parallel-Array Floating-Point Multipliers.” Submitted to 12th IEEE Symposium on Computer Arithmetic.

[9] M. Riepe, T. Huff, M. Upton, T.N. Mudge, R.B. Brown, “A 7-ns 53x53-bit Parallel Array Multiplier Implemented in 0.6um GaAs DCFL.” Submitted to ISCA 1994. Dept. of Electrical Engineering and Computer Science, University of Michigan.

[10] N.T. Quach, M.J. Flynn, High Speed Addition in CMOS,” Computer Systems Laboratory, Standord University, technical report: CSL-TR-90-415, February 1990.

[11] The PowerPC Architecture: A Specification for a New Family of RISC Processors,” International Business Machines, Inc. Morgan Kaufmann, SF, CA. 1994.

[12] Israel Koren, Computer Arithmetic Algorithms. Prentice Hall, Englewood Cliffs, NJ. 1993.

[13] M. Kelley and M. Postiff, “The FPU SUSPENS Model,” 9/23/96.

[14] T.D. Strong, M. Postiff, M. Kelley, “PUMA Floating Point Architecture Specification.” Version 0.2. 10/2/96.

[15] Weste N., and Eshrraghian K., Principles of CMOS VLSI Design, Addison-Wesley, 1985

[16] Rabaey J., Digital Integrated Circuits, Prentice Hall, 1996

[17] Dundas J., “A Description of the FXU”, version 1.2, 1996

[18] Gwennap L., “Intel’s MMX Speeds Multimedia”, Microprocessor Report, March 5, 1996.

[19] R.M. Jessani and C.H. Olson, “The floating-point unit of the PowerPC 603e microprocessor.” IBM Journal of Research and Development, Vol. 40 No. 5, September 1996. Pages 559-566.

[20] M. Kelley and M. Postiff, “Limited SIMD Instruction Set Architecture Extensions.” Available at http://www-personal.umich.edu/~postiffm/papers/papers.html.

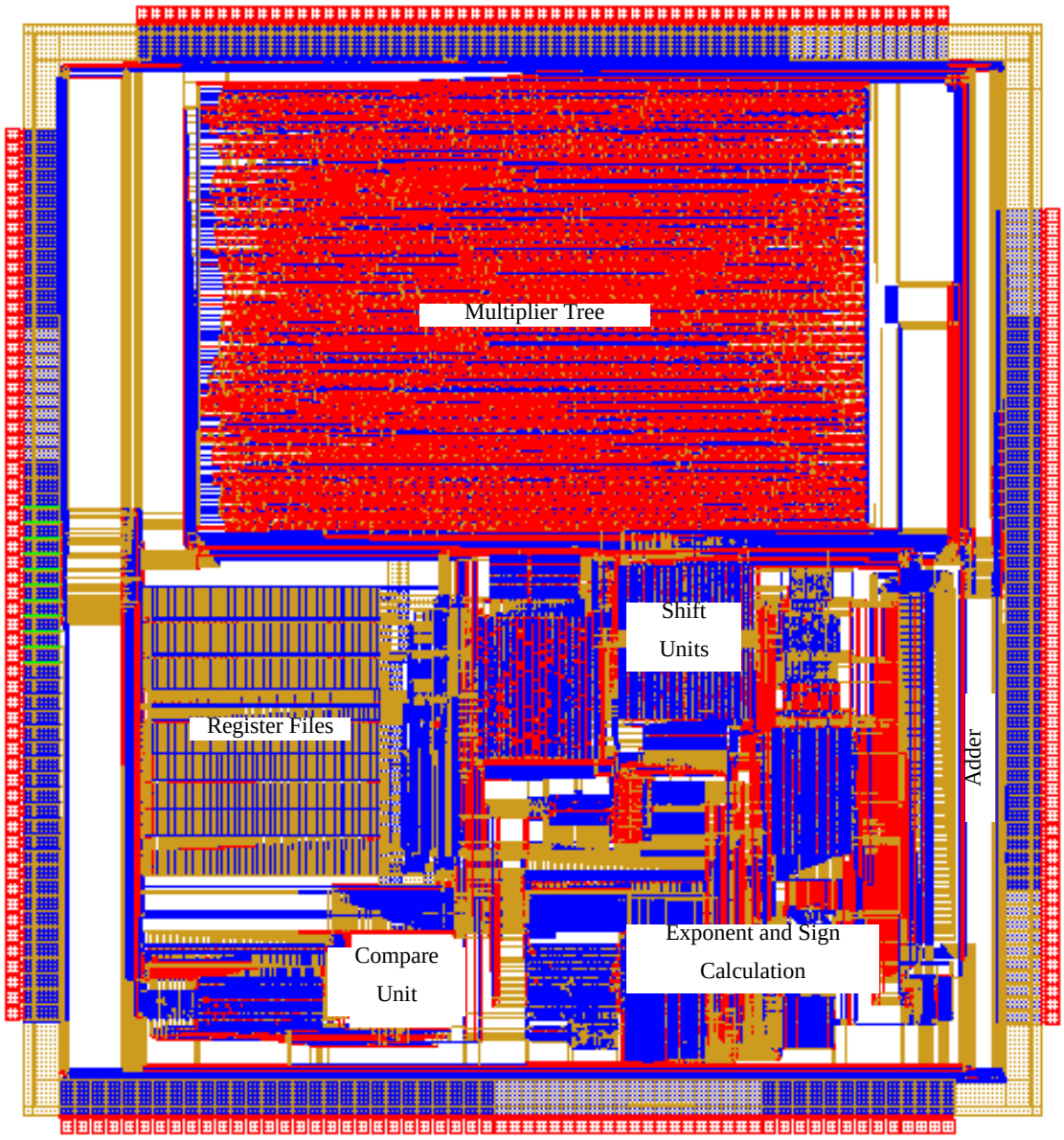

Figure 5. The floating point chip layout.