DOI: 10.14529/jsfi190205

Development of a RISC-V-Conform Fused Multiply-Add Floating-Point Unit

Felix Kaiser[1]***, Stefan Kosnac***[2]***, Ulrich Brüning***[2]

Despite the fact that the open-source community around the RISC-V instruction set architecture is growing rapidly, there is still no high-speed open-source hardware implementation of the IEEE 754-2008 floating-point standard available. We designed a Fused Multiply-Add Floating-Point Unit compatible with the RISC-V ISA in SystemVerilog, which enables us to conduct detailed optimizations where necessary. The design has been verified with the industry standard simulation-based Universal Verification Methodology using the Specman e Hardware Verification Language. The most challenging part of the verification is the reference model, for which we integrated the Floating-Point Unit of an existing Intel processor using the Function Level Interface provided by Specman e. With the use of Intel's Floating-Point Unit we have a "known good" and fast reference model. The Back-End flow was done with Global Foundries' 22 nm Fully-Depleted Silicon-On-Insulator (GF22FDX) process using Cadence tools. We reached 1.8 GHz over PVT corners with a 0.8 V forward body bias, but there is still a large potential for further RTL optimization. A power analysis was conducted with stimuli generated by the verification environment and resulted in 212 mW.

Keywords: floating-point, multiply-add, risc-v, hardware-design, verification, uvm, synthesis, asic, gf22fdx, ieee754.

Introduction

The open-source RISC-V Instruction Set Architecture (ISA) has gotten great attention in the last years and continues to thrive. However, it has yet to enter the realm of High-Performance Computing (HPC). To enable high-performance processors based on RISC-V, it is crucial to provide fast hardware Floating-Point Units (FPUs). Arguably, the most considered ranking of HPC systems is the TOP500 [17]. The criterion for this list is the floating-point-performance based on the benchmark LINPACK [12]. LINPACK is a numerical library for linear algebra. Therefore, floating-point multiplication with subsequent additions need to be performed. This corresponds to the function of the so called Fused Multiply-Add (FMA) units. FMA units implement a multiplication and a consecutive addition without intermediate rounding in hardware. Consequentially, the high throughput and energy efficient FMA units count to the essentials in HPC hardware. That is the reason why we are focussing on them within this paper.

Due to the standardization of floating-point called IEEE 754-2008, modular FPUs and especially FMAs have already been on the market for decades. The need for a new RISC-V-specific implementation lies in the nature of that standard: For historic reasons not each statement it contains is unique. To keep the design of the ISA clean and the results of different implementations reproducible, RISC-V makes these decisions fixed, but different from the existing implementations [20].

The latter enables the possibility to develop a universal verification environment for FPUs. This is another challenging point we are tackling within this work. Due to the high number of possible input patterns, we applied a simulation-based approach following the industry standard Universal Verification Methodology (UVM). UVM intends to generate constraint-random stimuli

[1]EXTOLL GmbH, Mannheim, Germany
[2]Heidelberg University, Heidelberg, Germany

for the Design Under Test (DUT). The same stimuli is distributed to one or more reference models to generate the expected result that is to be compared with the DUT output. As a key component of this approach a reference model has to be "known good". So we used Intel Intrinsics to get low level access to the Intel FMA unit of the processor running the verification tasks. Since some details were not covered by this model, we decided to integrate Berkeley SoftFloat, which is a software implementation of the IEEE floating-point standard.

Besides the Register Transfer Level (RTL) further optimizations for speed and power can be done in the Semi-Custom part of the flow. For the last step of the implementation, we performed this Back-End design flow for Global Foundries' 22 nm Fully-Depleted Silicon-On-Insulator (FDSOI) process. This includes synthesis, floorplanning, placement, scan insertion, clock tree synthesis and routing. Therewith, we are able to analyze which target frequencies are reachable with and without Forward Body Bias (FBB) and estimate the expected power consumption.

This article is organized in four parts. Section 1 gives an overview of the state of the art, followed by Section 2, which presents the FMA unit architecture. In Section 3 we discuss our verification approach, and in Section 4 the synthesis results are presented. Last but not least a conclusion summarizes our work.

1. State of the Art

As the presented work consists of three major parts, namely the design, the implementation, and the verification, different explorations need to be done. For the goal of a high-performance unit, design and implementation have to go hand in hand. Hence, the exploration is split up into development, which consists of design and implementation, and verification.

First of all, it has to be mentioned that not each IEEE 754-2008-conform FPU can be compared with every other one. This comes from the fact that IEEE 754-2008 leaves some decisions to the designer [2]. The RISC-V Foundation decided to avoid differences in functionality between different RISC-V compliant FPUs by making these decisions fixed within their standard. Following that, we only take other RISC-V-conform FPUs in consideration.

1.1. Development

The arguably most known implementation of a RISC-V FPU is the HardFloat of the University of California, Berkeley (UC Berkeley) [19]. It is used within different cores, or core generators, like the System-on-Chip (SOC) generator Rocket [4] and the Out-of-Order core Berkeley Out-of-Order Machine (BOOM) [8]. The fastest Rocket Implementation is SiFives' U54 Rocket on the TSMC 28 nm HPC process with 1.5 GHz [7]. Due to its multiple usages and even Tape-Outs, HardFloat can be counted as reliable. However, when it comes to high-performance, it has disadvantages. HardFloat is developed using the high-level hardware generation language Chisel [5]. Generally, Chisel does not take the opportunity to optimize a design completely, but in case of HardFloat a descriptive approach instead of an optimized architecture is chosen. Following that, the whole optimization is done within the Back-End. This restricts the potential of optimizing at the RTL.

Another open-source RISC-V-conform FPU is from Parallel Ultra Low Power (PULP) [11]. PULP is a platform of the ETH Zürich, where a set of RISC-V cores and peripherals they developed are provided. There is also an FPU designed in SystemVerilog [16]. Unfortunately,

it does not provide double-precision. It is also missing the rounding mode roundTiesToAway, which is obligatory for RISC-V.

1.2. Verification

The common issue with verification is that not each possible input and state can be tested. The provided design does not have a complex state, but, due to the wide inputs, it still can not be verified by iterating through all possible values. Realizable approaches are formal or simulation-based verification. Although there have been formal verifications of FPUs in the last years [10, 14], the proposed design is verified simulation-based, due to the larger amount of time needed for the corner cases in the execution of the formal methods [6, 9].

Since a verification environment for an FMA unit does not have to react to the internal state, it can be verified by generating the stimuli statically. Also it can be generated offline, which both increases the performance of the tests. An example of such a generator is IBMs FPGen [3], which works using a constraint solver. Unfortunately, the actual generator, or constraint solver, is not open-source, only a set of pre-generated single-precision inputs. Another approach, which is even a part of the RISC-V ecosystem, is the so called TestFloat [15]. TestFloat is a similar approach, that makes use of the SoftFloat model. SoftFloat is a software implementation of the IEEE 754-2008. Even though TestFloat would work for our design, we are using an UVM-based approach as it enables an efficient integration into system- respectively chip-level testbenches.

2. FMA Unit Design

Currently, the FMA unit supports all four double-precision fused operations defined in the RISC-V ISA (Tab. 1) as well as add, subtract, and multiply. It supports all rounding modes required by the IEEE 754-2008 standard and additionally roundTiesToAway, which is mandatory for RISC-V. Divide and square root will be implemented in the future using a Newton-Raphson algorithm.

Table 1. Supported RISC-V floating-point instructions

Instruction	Description	Operation
FADD	Add	$A + C$
FSUB	Subtract	$A - C$
FMUL	Multiply	$A \cdot B$
FMADD	Fused Multiply-Add	$A \cdot B + C$
FMSUB	Fused Multiply-Subtract	$A \cdot B - C$
FNMSUB	Negative Fused Multiply-Subtract	$-A \cdot B + C$
FNMADD	Negative Fused Multiply-Add	$-A \cdot B - C$

Figure 1 shows the interface and architecture of the FMA unit, which is based on [18]. It comprises three 64-bit inputs `port_a`, `port_b`, and `port_c` for the operands, and the 64-bit wide output `port_res` for the result. The type of operation is determined by `op`, the rounding mode by `rm` and exceptions are signaled at the output `exception_flags`. A forward flow control (not shown here) is implemented via `valid_in` and `valid_out`. `valid_in` can also be used for clock-gating inside the FMA unit. In the following, the main components of the design are described in more detail. The Sign-/Exponent Transformation transforms the operand exponents from the

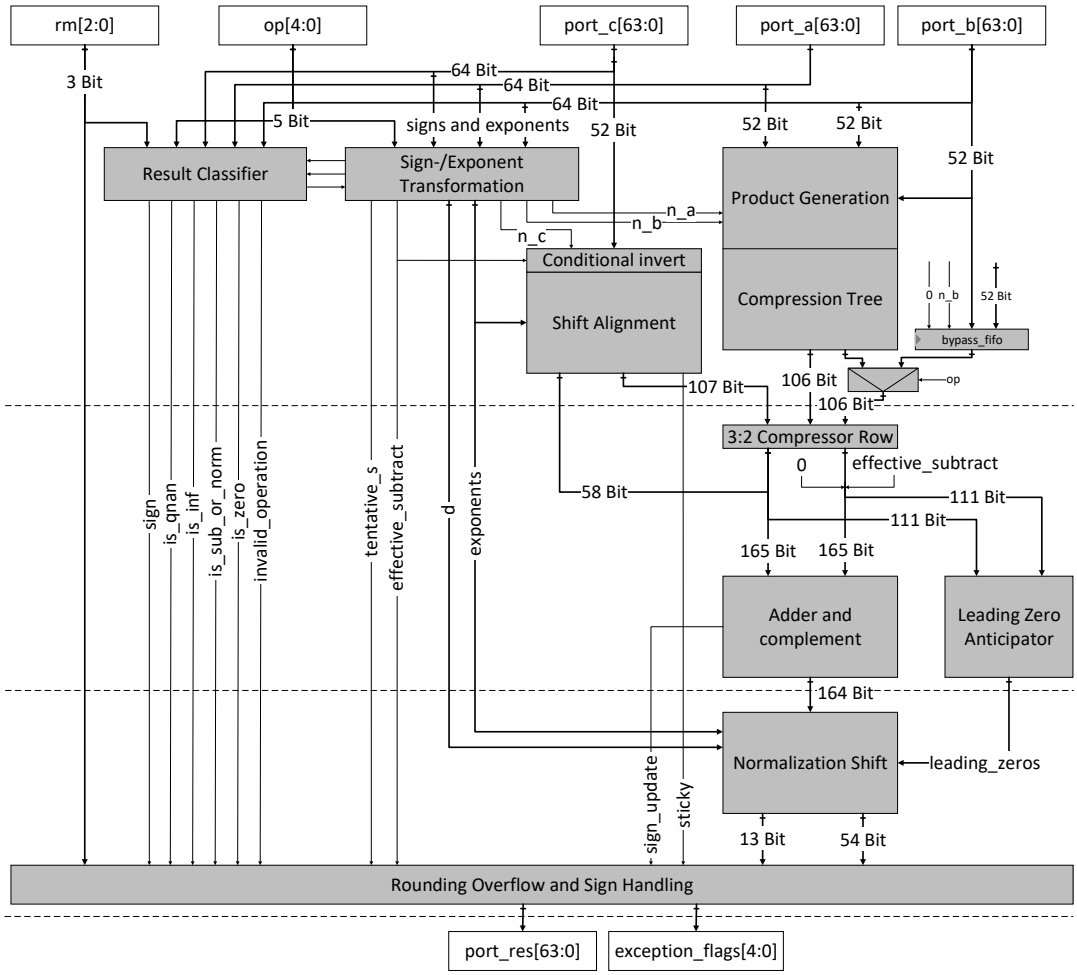


Figure 1. FMA unit architecture (the dashed lines represent the three pipeline stages)

biased representation into 2's complement and checks if the operands are normal, i.e. have no special values. Furthermore, it calculates the difference between the products' exponent (e_A+e_B) and the addends exponent (e_C). It also generates an effective subtraction bit indicating if the absolute values of $A \cdot B$ and C are added or subtracted. The Result Classifier handles operations with special values, such as qNaN, sNaN, zero, infinity, and subnormal numbers. Product Generation and Compression Tree perform the main part of the multiplication. Therefore, they take the mantissas of operands A and B and provide the product in carry-save representation. This allows to take the additional 3:2 Compressor Row to add another operand (mantissa of C) at low cost and is one of the reasons to perform FMA operations at all. For a floating-point addition, it is necessary to align the addends according to their exponent by shifting one of them relative to the other. This is done by the Shift Alignment in parallel to the compression tree. There is the case where one addend is much larger than the other, so the smaller one is completely absorbed and does not change the result. The Adder and Complement resolves the carry-save representation, as well as the following 2's complement into an 1's complement intermediate result. In parallel to the Adder, a so called Leading Zero Anticipator estimates the leading zeros of the intermediate result for the normalization. The latter is then done by the Normalization Shift. Afterwards the result is finalized by the Rounding, Overflow and Sign Handling unit, which determines if there is an overflow and performs rounding based on this information.

3. FMA Unit Verification

The verification of an FMA unit is a challenging task since a reference model is not easily developed, and not all input combinations can be tested within a reasonable time. The latter is mitigated with the simulation-based UVM, which we applied for the FMA unit. To avoid missing test cases that would show an erroneous behavior, the test cases are not predefined but instead generated constrained-random, which additionally facilitates automation. To keep track of which parts have been tested, code coverage, as well as functional coverage is used. The last important aspect is the checking. Due to its many special behaviors, the most challenging task in the verification of an FMA unit in general is to automatically generate the answer to the question whether a behavior is correct or not. Behavioral models, which are the common way to solve that issue, are usually developed by a verification engineer for the specific design. Since floating-point is standardized by IEEE 754-2008, other units can be used for this purpose.

One attempt of getting a reference for floating-point operations is to execute them in the applied language for the testbench. In last instance, such an operation maps onto the FPU within the utilized CPU. Since a higher level programming language and the instructions executed by a processor are separated by abstraction layers, this introduces a lack of controllability. For instance, the operation $D = A \cdot B + C$ may be compiled to a single multiplication followed by an addition or to an FMA operation. In our approach, we force the processor to execute the intended operation by using the programming language C and implementing the operations as intrinsics [1]. Therewith, we get the reliability of a "known good" Intel FPU.

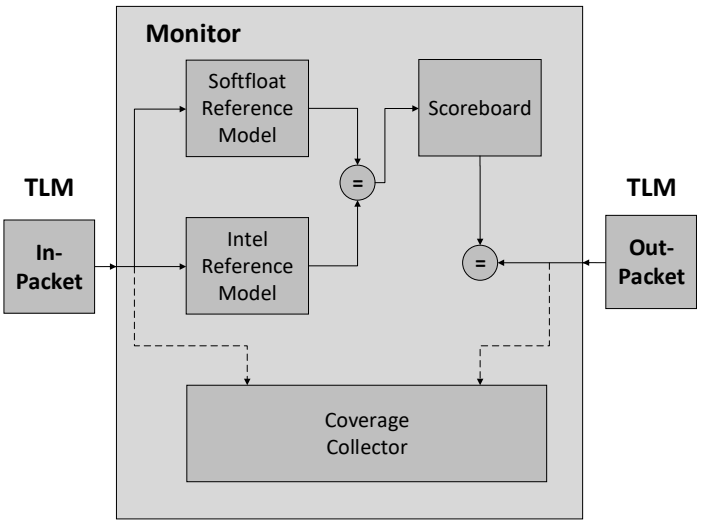


Figure 2. UVM Monitor applying the Intel Intrinsics reference model and the SoftFloat reference model

This approach alone is not sufficient as a reference model for a RISC-V FPU. As mentioned before, IEEE 754-2008 left some decisions to the implementer and Intel and the RISC-V Foundation took different choices. For these situations, we integrated another reference model, the already mentioned SoftFloat model [15]. Instead of just filling the holes using this model, we integrated both models in parallel and checked them for common cases against each other as shown in Fig. 2. Functional coverage is collected concurrently.

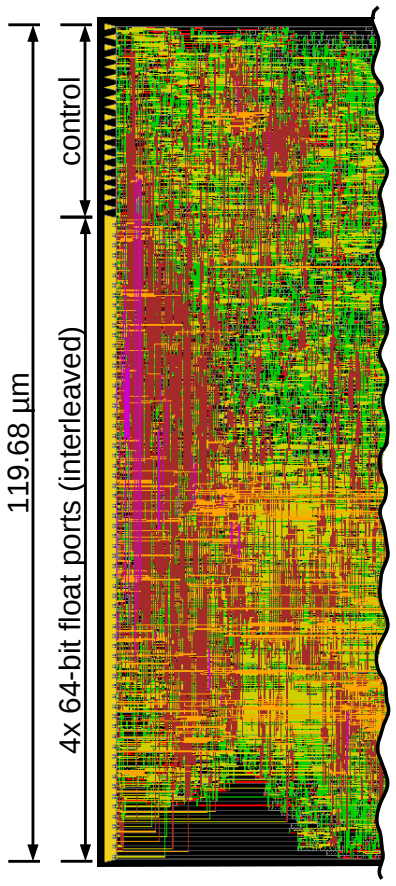
4. FMA Unit Back-End

For the Back-End implementation, we chose a recent technology: Global Foundries' 22 nm FDSOI process. It applies an additional insulator layer to remove the diodes between drain/source and the substrate. Furthermore, the channel is fully depleted, i.e. it is not weakly doped, to reduce leakage current. The insulator layer acts like a back-gate, which can be used to modify the transistor's threshold voltage. A back-gate bias voltage generator can later be used to apply a voltage to the back-gate, called Body Bias (BB). This allows to tune the circuit for either more performance or lower leakage or compensate process corners. The latter already emphasizes that this bias needs to be treated like process, voltage and temperature in static timing analysis. The GF22FDX process offers four types of transistors for different applications: high (HVT), regular (RVT), low (LVT), and super low (SLVT) threshold voltage devices. Since we are looking for performance, we decided to use the SLVT standard cell library.

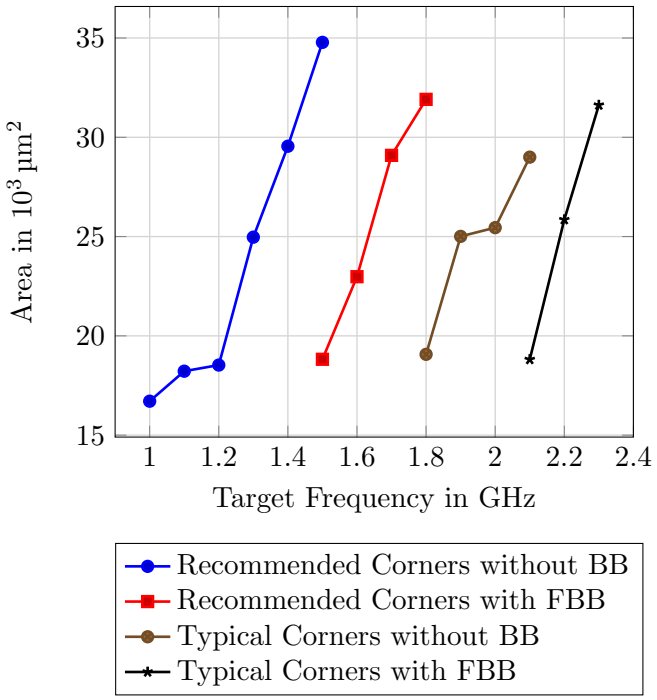
4.1. Synthesis Results

The floorplan was kept rather simple for this first implementation. We only defined the height to be 119.68 µm. The length was adjusted to yield a utilization of 80%. A more detailed placement will be part of future work, when more submodules are described at a lower abstraction level. This will allow for more control about what is synthesized. Pin placement was done with a later application in a RISC-V processor implementation in mind. RISC-V suggests a dedicated floating-point register file, which will need three read- and one write-port to provide the operands for fused operations. Assuming the register file will be located left of the FPU in a pipeline, we placed the operand and result pins on the left side in an interleaved manner using metal layers 3 to 6 and a spacing of 0.35 µm. The remaining pins are also placed on the left side with a spacing of 1.4 µm on metal 3 following the pins of `port_a`. This is shown in Fig. 3a. Depending on the exact register file size, this spacing may need to be changed in the future. To get a realistic timing we already performed scan insertion for this first synthesis run. This replaces all Flip-Flops (FFs) with Scan Flip-Flops (SFFs) that have a multiplexer in the datapath to switch between the regular input (D) and the scan input (SI). The additional multiplexer delay reduces the time available for other logic, but a scan chain is needed for chip testing. The effect of the clock distribution was also considered by performing Clock Tree Synthesis (CTS). This adds a buffer tree to the design to distribute the clock to all clock inputs and assures that the rising clock edge reaches every FF within a defined time window. Subsequent to CTS the design was routed. After routing the timing was met for a cycle time of 666 ps, i.e. 1.5 GHz over all recommended implementation corners without using FBB. Figure 3b shows more details of our results.

The synthesis results for the lower frequencies (blue curve) were obtained using the recommended corners for setup and hold analysis. These are slow (SS) and fast (FF) process, 10% voltage deviation around the nominal voltage of 0.8 V and a temperature of −40 °C and 125 °C. From these recommended corners the tools identified the combination (SS, 0.72 V, 125 °C, RC max) to be most timing critical. The currently only partially optimized design suffers from a significant area increase with rising target frequency. Figure 3b shows that the area roughly doubles from 1.2 GHz to 1.5 GHz. Despite 1.5 GHz being our target frequency for now, we conducted some tests for higher frequencies. For a real design, we could apply a FBB to increase the performance. So we switched the corners to the corresponding recommended corners with



a) Pin placement



b) Area over target frequency

Figure 3. Synthesis results

FBB. This allowed us to reach frequencies up to 1.8 GHz (red curve). To see where the design performs typically, we synthesized with typical (TT) corners only (brown curve) and also with their forward body biased versions (black curve). Typical corners are available for 25 °C and 85 °C. Here the tools identified (TT, 0.8 V, 85 °C, RC nominal) to be most timing critical. This allowed us to reach up to 2.3 GHz. The tools used were Cadence Genus and Innovus.

Another observation is, that we can reach higher frequencies with only typical corners, than with the recommended corners using FBB. This shows it is not possible, at least for this design, to compensate a worst case corner completely by using FBB.

4.2. Power Analysis

For all points presented in Fig. 3b, a power analysis based on a value change dump (vcd) file containing stimuli was conducted. The stimuli were generated with a modified test from our verification environment using Cadence Xcelium. The testbench contained the netlist, derived after all the previously described synthesis steps where executed, and a clock signal of the corresponding target frequency. As a side effect we also got some confidence in the synthesis procedure by running some stimuli through the implemented netlist. The test itself applied new operands every clock cycle and performed a random operation with a random rounding mode. The total power consumption for every design is shown in Fig. 4. The values were calculated for the (TT, 0.8 V, 85 °C, RC nominal) corner for all points, with the ones implemented with FBB

having their power determined with the corresponding FBB corner. We chose this corner for the power analysis, since all parameters are closest to real operating conditions. The clock tree makes up between 0.77 % and 1.56 % of the total power from high to low target frequencies, which seems plausible for a small design. The power analysis was done with Cadence Innovus/Voltus.

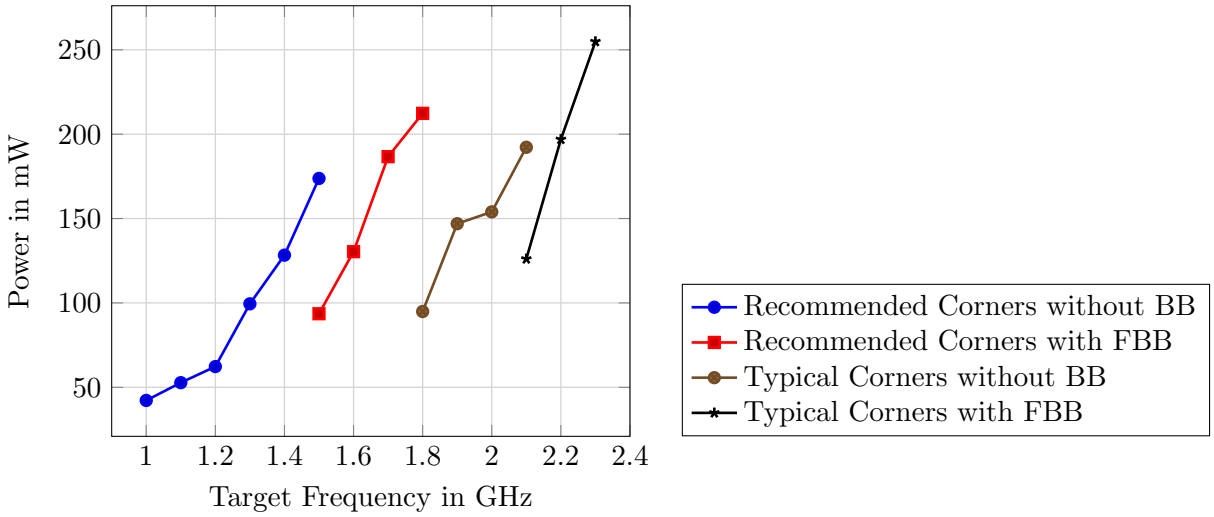


Figure 4. Power over target frequency for different corner setups

Figure 4 shows that the power scales with area and frequency. Figure 5 shows the power consumed by the 2.3 GHz implementation operating at frequencies from 1.0 GHz to 2.3 GHz. The linear rise of power with frequency is expected, but the values also show that a faster design, which uses more area, also consumes more power at lower operating frequencies than a design implemented for that particular target frequency. Besides the total power (black curve), Fig. 5 also shows the three parts which make up the total power. There is the switching power representing the loading/unloading of nets and the power used internally in the standard cells. They make up the linear part. The third part (brown curve) is the leakage power, which is constant at 33.6 mW over the operating frequency. This high leakage current is caused by using SLVT standard cells and FBB.

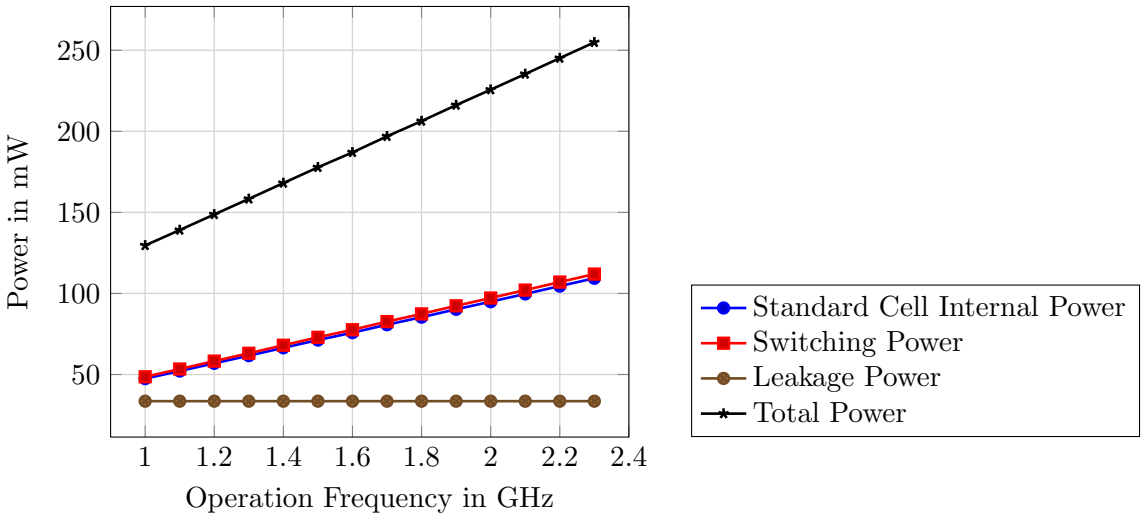


Figure 5. Power of the 2.3 GHz implementation over operation frequency

To evaluate our synthesis results we compared them to [13]. Table 2 compares the two closest FMA implementations in terms of the metrics used in [13]. We calculated the performance of our design by assuming two floating-point operations per clock cycle, i.e. assuming the maximum throughput possible. This was also done in [13]. Their design runs with 1.81 GHz in a 45 nm technology using low threshold devices and six pipeline stages. Compared to our synthesis result at 1.8 GHz for a typical process with super-low threshold devices, we have a similar power per performance but are roughly a factor of 3 smaller in terms of area per performance. The latter is contributed to technology scaling. The former is probably due to the low pipeline depth of three versus six. A lower number of pipeline stages makes it harder to achieve timing, thus requires the synthesis tool to use additional logic to fit the combinational logic into the cycle time. Furthermore, our design is not optimized for power in any way yet.

Table 2. Comparison of our synthesis results with [13]

Property	45 nm FMA [13]	Our 22 nm Design
V_{th}	low	super low
V_{DD} in V	0.9	0.8
Pipeline Depth	6	3
Frequency in GHz	1.81	1.8
Area in µm^2	49839	19066
W/GFLOPS	0.0253	0.0264
mm^2/GFLOPS	0.0145	0.0053
W/mm^2	1.75	4.98

Another interesting fact is seen in Fig. 6, which shows power per area over target frequency. Firstly, power density scales linearly with frequency as expected. But the second observation is that using FBB keeps the power density constant, whereas going from slow to typical corners reduces power more than one would expect from the area shrink alone. This shows again, that FBB is not enough to compensate worst case corners.

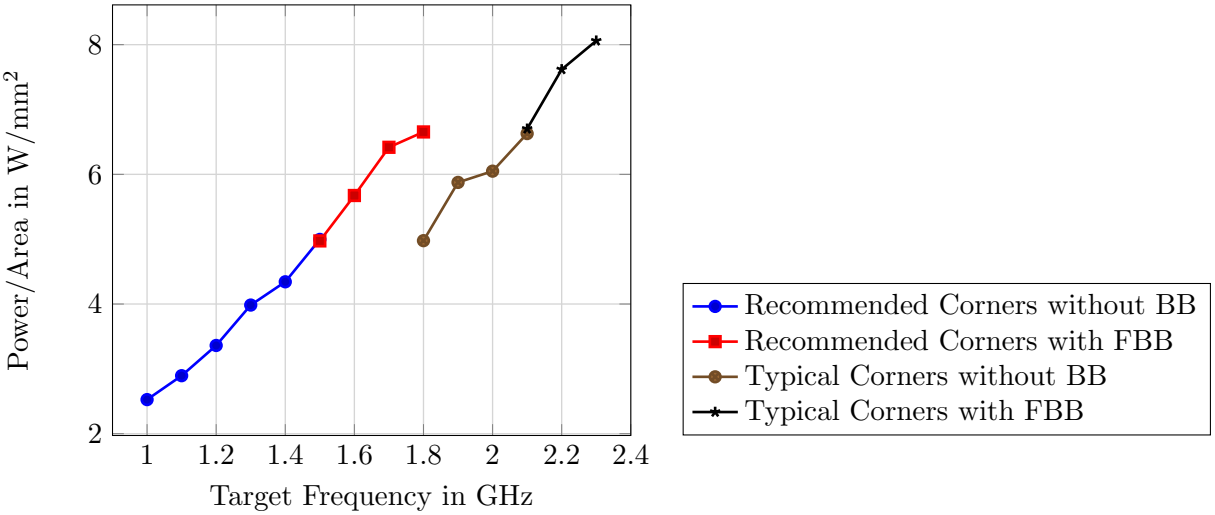


Figure 6. Power/Area over target frequency for different corner setups

Conclusion

This paper presents a design of a RISC-V- and IEEE 754-2008-conform FMA unit, which passed the complete ASIC design flow. This work's target is to lay the foundation for a high-speed FPU. Although it is still work in progress, it reaches 1.8 GHz using FBB - which is faster than the HardFloat of the U54 Rocket chip [7]. However, we do not have information on how fast other FPUs could be implemented standalone, without the corresponding CPU core. In terms of power, we still have room to improve, but power efficiency was not our goal. Still, we will certainly have a closer look at it in the future.

Additional to design and Back-End flow, we ensured functional correctness by performing a verification in which we checked the FMA unit against Intel's FPU and the Softfloat reference model. We also performed a small number of tests on the gate level during generation of the vcd file used for the power analysis.

References

1. Intel C++ Intrinsics Reference. `http://www.info.univ-angers.fr/pub/richer/ens/l3info/ao/intel_intrinsics.pdf` (2007), accessed: 2019-06-21

2. IEEE Standard for Floating-Point Arithmetic. IEEE Std 754-2008 pp. 1–70 (2008), DOI: 10.1109/IEEESTD.2008.4610935

3. Aharoni, M., Asaf, S., Fournier, L., Koifman, A., Nagel, R.: FPgen - a test generation framework for datapath floating-point verification. In: Eighth IEEE International High-Level Design Validation and Test Workshop 2003, 12-14 November 2003, San Francisco, California, USA. pp. 17–22 (2003), DOI: 10.1109/HLDVT.2003.1252469

4. Asanović, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser, J., Izraelevitz, A., et al.: The Rocket Chip Generator. EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2016-17 (2016)

5. Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawrzynek, J., Asanović, K.: Chisel: Constructing hardware in a Scala embedded language. In: The 49th Annual Design Automation Conference 2012, DAC 2012, 3-7 June 2012, San Francisco, California, USA. pp. 1212–1221 (2012), DOI: 10.1145/2228360.2228584

6. Beyer, S., Jacobi, C., Kröning, D., Leinenbach, D., Paul, W.J.: Putting it all together – Formal verification of the VAMP. International Journal on Software Tools for Technology Transfer 8(4), 411–430 (2006), DOI: 10.1007/s10009-006-0204-6

7. Celio, C., Chiu, P.F., Nikolic, B., Patterson, D., Asanović, K.: BOOM v2 an open-source out-of-order RISC-V core. `https://content.riscv.org/wp-content/uploads/2017/12/Wed0936-BOOM-v2-An-Open-Source-Out-of-Order-RISC-V-Core-Celio.pdf` (2017), accessed: 2019-06-21

8. Celio, C., Patterson, D.A., Asanović, K.: The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor. EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2015-167 (2015)

9. Clarke, E.M., German, S.M., Zhao, X.: Verifying the SRT division algorithm using theorem proving techniques. In: Computer Aided Verification. pp. 111–122. Springer Berlin Heidelberg, Berlin, Heidelberg (1996), DOI: 10.1007/3-540-61474-5_62

10. Clarke, E.M.: The Birth of Model Checking. In: 25 Years of Model Checking: History, Achievements, Perspectives, pp. 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), DOI: 10.1007/978-3-540-69850-0_1

11. Conti, F., Rossi, D., Pullini, A., Loi, I., Benini, L.: Energy-efficient vision on the PULP platform for ultra-low power parallel computing. In: 2014 IEEE Workshop on Signal Processing Systems, SiPS 2014, 20-22 October 2014, Belfast, United Kingdom. pp. 1–6 (2014), DOI: 10.1109/SiPS.2014.6986099

12. Dongarra, J.J.: The LINPACK Benchmark: An explanation. In: Supercomputing, ICS 1987, 1st International Conference Athens, Greece, June 8–12, 1987. pp. 456–474. Springer Berlin Heidelberg, Berlin, Heidelberg (1988), DOI: 10.1007/3-540-18991-2_27

13. Galal, S., Horowitz, M.: Energy-Efficient Floating-Point Unit Design. IEEE Transactions on Computers 60(7), 913–922 (2011), DOI: 10.1109/TC.2010.121

14. Harrison, J.: Formal verification of ia-64 division algorithms. In: Theorem Proving in Higher Order Logics, 13th International Conference, TPHOLs 2000 Portland, OR, USA, August 14–18, 2000. pp. 233–251. Springer Berlin Heidelberg, Berlin, Heidelberg (2000), DOI: 10.1007/3-540-44659-1_15

15. Hauser, J.: The softfloat and testfloat packages. `http://www.jhauser.us/arithmetic/` (2015), accessed: 2019-06-21

16. Li, L.: PULP-Platform - FPU. `https://github.com/pulp-platform/fpu` (2017), accessed: 2019-06-21

17. Meuer, H., Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: Top 500 list. `https://www.top500.org/lists/2018/11/` (2018), accessed: 2019-06-21

18. Muller, J.M., Brisebarre, N., De Dinechin, F., Jeannerod, C.P., Lefevre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S.: Handbook of floating-point arithmetic. Springer Science & Business Media (2009)

19. Waterman, A., Lee, Y., Hauser, J.: Berkeley Hardware Floating-Point Units. `https://github.com/ucb-bar/berkeley-hardfloat` (2018), accessed: 2019-06-21

20. Waterman, A.S.: Design of the RISC-V Instruction Set Architecture. Ph.D. thesis, EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2016-1 (2016)