Efficacy of Pipelining to Reduce Energy of Floating-Point Adders and Multipliers

Per Larsson-Edefors
Chalmers University of Technology
Gothenburg, Sweden
perla@chalmers.se

Erik Börjeson
Chalmers University of Technology
Gothenburg, Sweden
erikbor@chalmers.se

***Abstract*—Because of the way arithmetic circuits are logically implemented, spurious transitions (glitches) can dominate the energy dissipation of combinational implementations. Insertion of pipeline registers effectively suppresses glitches, but this comes with an energy overhead. To identify in what design situations pipelining reduces energy, we consider adders and multipliers for three different floating-point formats: One 32-bit (FP32) and two 16-bit formats (FP16 and bfloat16) that have different exponent and significand field widths. Each circuit is implemented in a predictive 7-nm FinFET technology and evaluated in terms of gate and signal-transition distributions and energy-delay products. Our evaluations show that, regardless of format, adders benefit more from pipelining than multipliers. Assuming constant timing constraints, pipelining can reduce total adder energy by half for formats with longer significands, but less, around 40%, for bfloat16 which has a short significand. Since combinational adders have larger logic depths than multipliers, an additional positive effect of pipelining is that the timing of the adder becomes more balanced with the multiplier.**

I. Introduction

Differences in input logic path delays may cause a gate's output to make several spurious logic signal transitions (glitches) before the output settles to its final steady-state value. Once a glitch has been generated, it may be propagated to downstream logic, leading to a growing number of transition events in downstream logic levels [1]. The energy dissipated by glitches depends on the logic circuit structure and it is well known that arithmetic circuits are glitch intensive [2], [3].

Pipeline registers effectively suppress glitch propagation. For arithmetic circuits of higher complexity, like floating-point arithmetic, the effect of pipelining on energy dissipation can be significant. Even though the insertion of extra register cells incurs an energy overhead, which is exacerbated by the addition of clock wires and buffers, the total energy dissipation may decrease after pipelining is introduced.

In some systems, pipelining may be used to balance timing of logic stages in order to increase clock rate. But in other systems, for example, those with tight feedback loops, additional pipeline stages cause system performance degradations which may not be acceptable. What is not well known is in what design scenarios it makes sense to introduce pipelining with the main purpose to *reduce energy*. It may be possible to introduce one level of registers and an additional cycle of latency, if this leads to an overall energy reduction.

To build design insights for situations where pipelining may be an effective tool to reduce energy, we implement and analyze two key computing circuits: floating-point adders and multipliers. To enable robust observations, we consider three widespread formats which correspond to different design targets in terms of computing precision and dynamic range.

II. Formats

The IEEE-754 standard for floating-point arithmetic [4] has successfully provided a unified definition for floating-number representations, however, this feature-rich representation leads to complex implementations. With the advent of new applications like machine learning, shorter floating-point formats have been proposed. Some of these formats abandon IEEE-754 to reduce hardware complexity: For example, the 16-bit formats of bfloat16 [5] and DLFloat [6] share properties, in that they have no support for denormals and use fixed rounding modes. The former, however, uses round nearest to even, while the latter uses round nearest up. While simplifications like these make new formats have some shortcomings, these are deemed not to be important enough to warrant the implementation of significantly more complex circuits.

The FloPoCo framework [7] supports the Nfloat representation which is a simplification over IEEE-754. Beside sign, exponent and significand input bits, two additional bits are used to define Nfloat's operation mode, where a normal mode is augmented with zero, Inf and NaN modes.

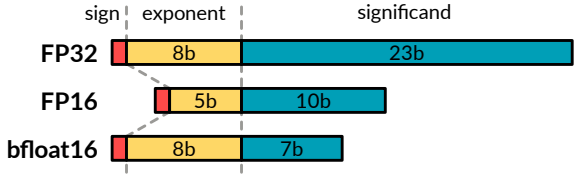
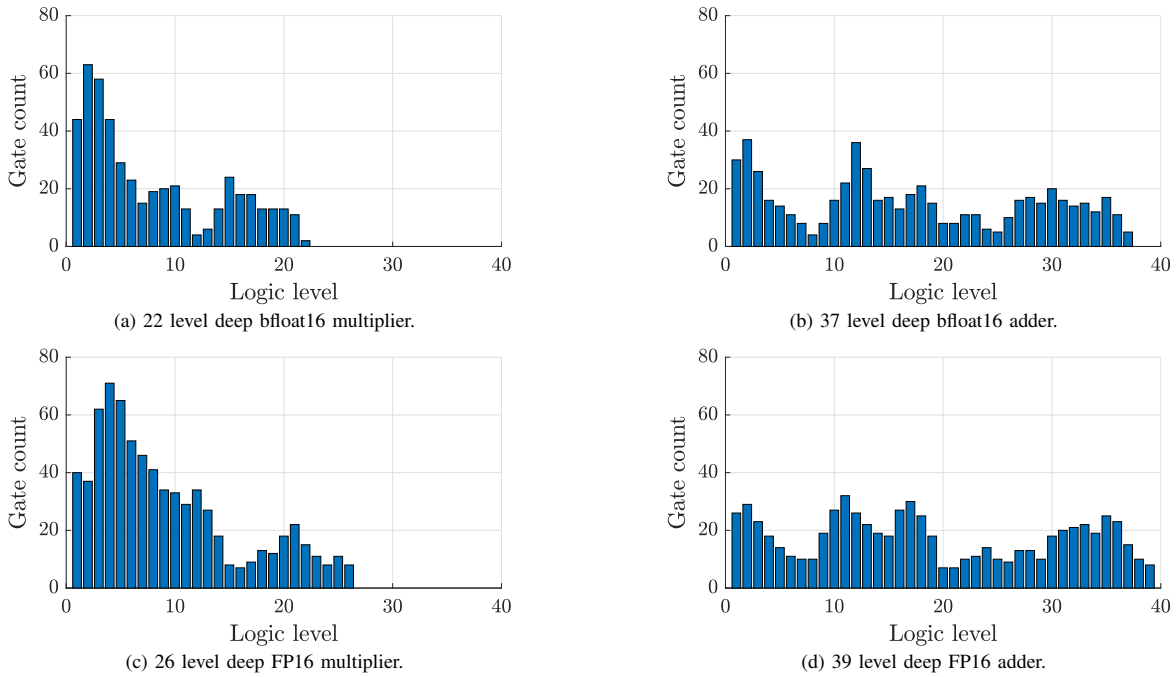


Fig. 1: Floating-point formats under consideration.

The circuit implementation complexity of floating-point adders and multipliers depends on exponent and significand field widths. Interestingly adders and multipliers have very different dependencies on these two fields. To address implementations representative of practical applications, we will consider the three formats shown in Fig. 1: 32-bit floating-point (FP32), 16-bit floating-point (FP16) and bfloat16. While



(a) 22 level deep bfloat16 multiplier.

(b) 37 level deep bfloat16 adder.

(c) 26 level deep FP16 multiplier.

(d) 39 level deep FP16 adder.

Fig. 2: Total number of gates per logic level of arithmetic netlist implementations.

FP32 has a wide exponent and significand, FP16 uses a shorter exponent but a relatively long significand, and bfloat16 uses as long exponent as FP32 but a shorter significand.

III. Method

The FloPoCo framework [7] targets FPGA systems, but the HDL code that can be generated is also applicable to ASIC designs. We generate HDL code for the three selected formats. To isolate the impact of logic gates on the main computational logic paths, we target Nfloat implementations for which we statically set the mode bits to normal mode ('01'). During synthesis, this leads to all logic associated with zero, Inf and NaN modes being trimmed away during logic optimization. Similarly, we discard the mode output bits so that these do not incur any hardware cost.

Synthesis in Cadence Genus [8] and verification in Cadence Xcelium [9] largely follow the workflow described in [10]. For each of the three formats, we generate a large number of custom input vectors to drive the synthesized netlists in logic simulation. Additionally, a reference output is generated in Matlab and used to verify netlist functionality.

The generated vectors are used also as inputs to the netlists during power analysis: If we assume N circuit nodes at a supply voltage of V_{DD}, then the total switching power dissipation is defined as $P_{sw} = f\,{V_{DD}}^2 \sum_{i=1}^{N}(C_i\,\alpha_i)$. Here, f is the system clock rate, while the switching activity α_i describes the fraction of clock cycles when a circuit node i with capacitance C_i switches from 0 to 1. Randomly switching signals have an α of 0.25, but this is not a realistic assumption on signals in practical systems. Since we will consider pipelined circuit in which registers are inserted, capacitive loads driven directly by the clock will be introduced. Therefore it is important to use a realistic ratio of clock and logic signal switching activity. All bits of the generated vectors that drive the primary inputs of the netlists are designed to have an average switching activity α of 0.1, which is in line with assumptions in EDA tools [11]. Switching events for all *internal* nodes are captured during logic simulation in value change dump (VCD) files.

The target of synthesis is the predictive ASAP7 7-nm FinFET technology developed at Arizona State University with support from Arm Ltd. [12]. We use regular-VT cells at the typical corner, which means static power is insignificant.

IV. Gate Distribution and Energy-Delay Product

The current drawn by a circuit correlates with the switching events that take place as signals are evaluated and propagated through different logic paths. Thus, it is instructive to study the number of gates that are located at different logic levels in floating-point adders and multipliers. Now, we assume the primary inputs represent logic level 0. For each k-input gate in the netlist, we define its logic level by taking the maximum of the logic levels for the k gates which drive its inputs and incrementing this number by one. As a result of micro-architectural design decisions and synthesis area optimizations, some gate output signals skip logic levels and jump from level i to j where $j > i+1$ or even $j \gg i+1$.

Figure 2 shows gate distributions for four different circuits that are synthesized for their respective *minimal timing constraint*, obtained for the lowest constraint that has a non-negative timing slack. Our gate distribution analysis shows clearly that floating-point adder circuits have larger logic

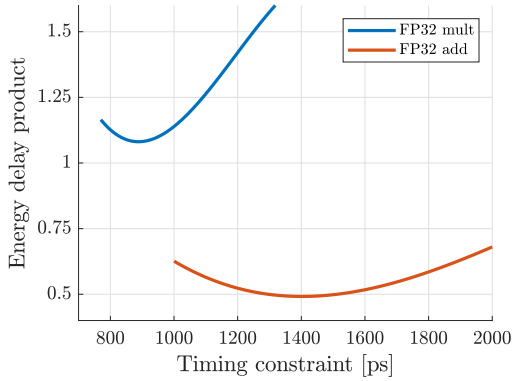


Fig. 3: Energy-delay product (EDP) for FP32 arithmetic.

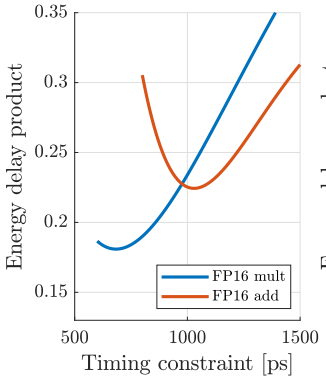


Fig. 4: EDP for FP16.

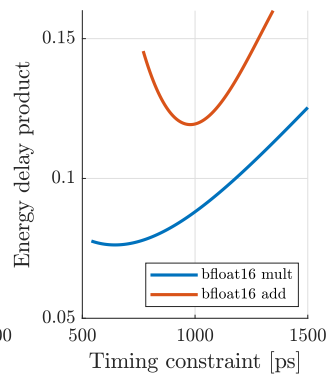


Fig. 5: EDP for bfloat16.

depths than multipliers. On the other hand, multipliers have a greater number of gates per logic level.

The adder type that we use here is an area-efficient single-path design. Since floating-point adders are known to be slow [13], approaches to parallelize some computations can speed up operation. Using dual-path FloPoCo adders reduces delay by around 10% for all formats, but area increases by approximately the same amount. An exception is for formats with shorter exponents, like FP16 and its 5-bit exponent field, for which the area overhead is small.

The circuit's energy-delay product (EDP) is a metric which helps designers identify at what timing constraint the energy efficiency of an arithmetic circuit is maximized. We synthesize and analyze each circuit for a range of timing constraints with relatively short increments. Then we combine the timing constraint (delay) with the obtained energy per operation to create EDP curves based on curve fitting. As shown in Fig. 3, the EDP curves for floating-point adders and multipliers are quite different. If we consider the gate distributions for the floating-point adder, we can notice that they have large logic depths but relatively few gates per level. This is the reason the EDP curve of the adder is flat in comparison to the EDP of the multiplier.

Figures 4 and 5 show the EDP curves for the less complex formats of FP16 and bfloat16. Here it is noticeable that the adder dissipates more energy than the multiplier for shorter significands (bfloat16). For 16-bit multipliers, it makes sense from an EDP perspective to push the implementations very close to their minimal timing constraints.

A final observation relevant to this study is that, regardless of format and variations in field widths, Figs 3, 4 and 5 show that floating-point adders cannot support as high clock rates as floating-point multipliers can. At least, this is true as long as pipelining is not used.

V. Logic Gate Distribution and Current Profile

As they describe how many gates per logic level are involved in the creation of signal transitions, gate distributions correlate with current and power profiles. As shown in Figs 6 and 7, the distribution of logic gates and the current profile from simulation share some features. One major reason they are not perfectly matching is that gates have different delays: The delay of each individual gate depends on logic function, gate drive strength, input rise and fall times, and output capacitive load. The latter depends on gate fanout and wire loads and, thus, is not possible to fully determine until place & route (P&R) has been completed. Since glitch energy dissipation is a sensitive function of delay modeling accuracy [1], netlist-based logic simulations will have accuracy limitations caused by missing P&R information. But P&R requires very long runtimes, so we use post-synthesis data for all implementations and separately consider P&R aspects in Section IX.

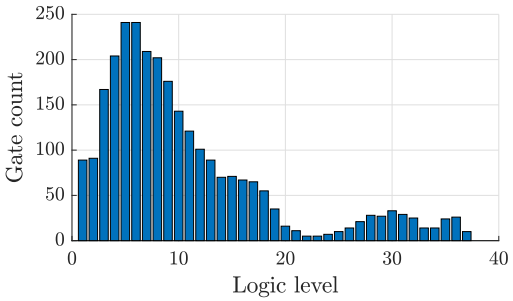


Fig. 6: Total number of gates per logic level in an FP32 multiplier.

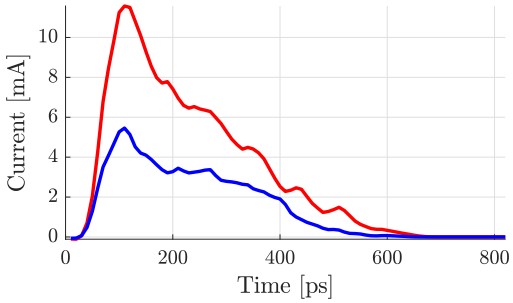
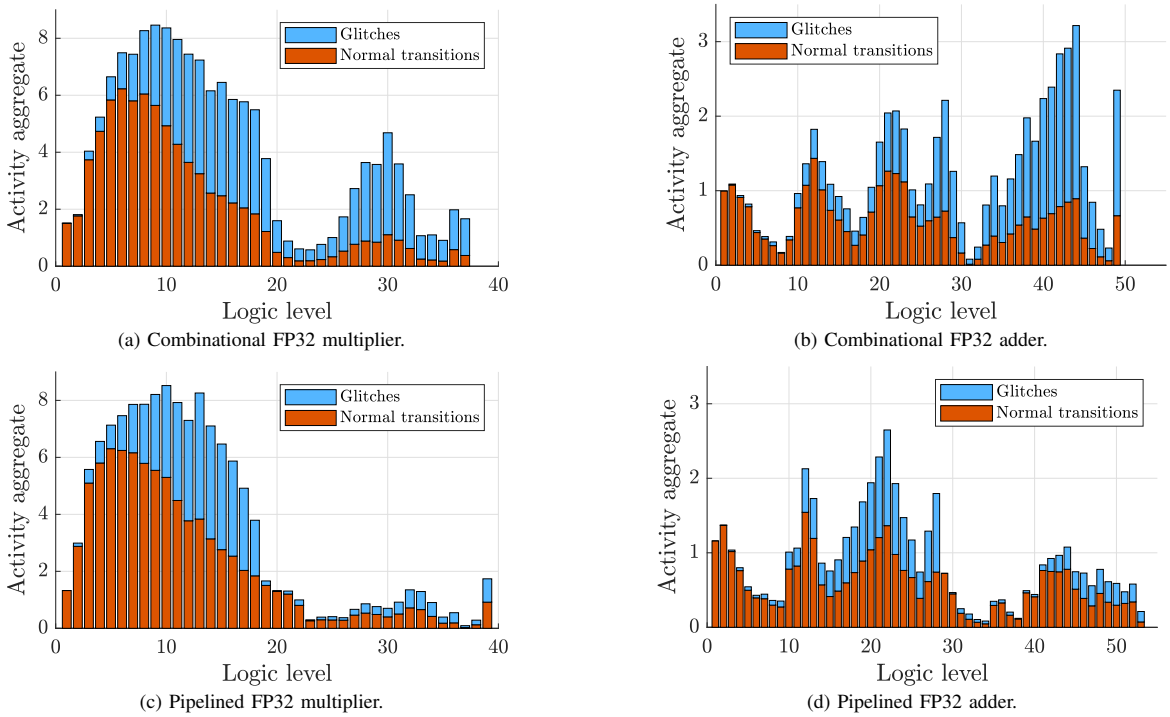


Fig. 7: Current profile of two (red vs blue) different computational cycles in an FP32 multiplier implementation. It is straightforward to estimate the energy dissipated in one clock cycle: For an average current of, say, 1 mA drawn over a 1-ns cycle, 1 pJ is dissipated.



(a) Combinational FP32 multiplier.

(b) Combinational FP32 adder.

(c) Pipelined FP32 multiplier.

(d) Pipelined FP32 adder.

Fig. 8: Total switching activity (α) per logic level in FP32 arithmetic circuits synthesized at each respective minimal timing constraint. To enhance readability, results for multipliers and adders use different ranges on X and Y axes.

In the power analysis, the input vector sets are developed to have a switching activity α of 0.1 to mirror practical workloads [11]. But the vectors are randomly generated and will cause each cycle to have a unique current profile. In fact, two different current profiles are shown in Fig. 7: One is for the cycle which has the highest power dissipation, the other is for the cycle with minimal power. These two cycles are only different in what data the multiplier operates on. Fig. 7 illustrates that the circuit's power dissipation is not only depending on which logic gates are used, but on which input data vectors we use in our analysis.

VI. Logic Signal Transition Distribution

Based on the gate distributions, we can represent the switching events on signals throughout the circuits in activity profiles. We annotate the VCD transition information obtained from logic simulations, both for normal transitions[1] and glitches, to each respective gate. To illustrate signal transitions per logic level, we take the α factor (Section III) for each gate and add all α for gates located in the same logic level. Here, it is useful to recall that α for the clock is 1.

The activity profile for the combinational FP32 multiplier and adder is shown in Fig. 8a and Fig. 8b, respectively. To show the impact of glitches, we stack the bars so that normal transitions are shown as a base contribution on top of which spurious transitions are added. Clearly glitches make up a significant portion of transitions and as logic signals travel through the logic circuit, this portion increases drastically. This behavior is especially pronounced in the adder, where glitches by far outnumber normal transitions in the later logic levels, from level 40 and downstream. This behavior is inherent to the single-path floating-point adder.

Since glitches cause extra energy to be dissipated, the large number of unnecessary transitions is a real problem. Now, pipelining the combinational circuit with an extra level of registers will partly solve this problem: since each register samples its input data, it will effectively suppress all glitches that arrive on its input and provide a glitch-free output.

VII. Pipelining to Suppress Glitches

We now add registers at the output of each arithmetic component and let the synthesis tool perform timing-driven retiming, which entails moving the registers inside the combinational circuit. FloPoCo supports pipelined arithmetic components, but they are optimized for FPGAs which have a more coarse-grained logic substrate than ASICs. As it turns out, our approach leads to more efficient retimed ASIC implementations than if we start from FloPoCo's pipelined HDL.

The result of pipelining on logic signal transitions is shown in the activity profiles for the pipelined FP32 multiplier and adder in Fig. 8c and Fig. 8d, respectively. Both circuits are implemented for their respective minimal timing constraint and, clearly, this leads to an effective suppression of glitches for both multipliers and adders. In the downstream logic following

[1] A normal transition is one which flips to the steady-state value of a specific signal node.

the registers, relatively few new glitches are generated. By contrasting the pipelined activity profiles with Figs 8a and 8b, we can conclude that the majority of glitches in the later logic levels of the combinational implementations are propagated from gates located in the early logic levels.

We initially add 32 or 16 registers at the outputs, however, as retiming is carried out, the actual number of used pipeline registers depends on which timing constraint is enforced. For example, when implementing a pipelined FP32 multiplier for its minimal timing constraint, more than 100 pipeline registers are required. If we relax the timing constraint to that of a combinational FP32 multiplier, the number of registers drops to around 70. The actual register count is decided by how many internal combinational signal paths that need to be pipelined. While the exact delay of signal paths is not only decided by logic level, gate distributions such as Fig. 2 serve to illustrate that some logic levels have a greater number of gates (and, thus, output signals) than other levels.

Now, assume that we keep the timing constraint constant and equal to the minimal timing constraint of the combinational circuit. This is the scenario considered when using pipelining to reduce energy. Introducing pipelining means that the inserted pipeline registers cut long combinational paths into two sets of shorter paths, so the logic portions located before and after pipeline registers will have their individual timing constraints relaxed by almost 2X. In general, this timing relaxation means that the gate count per level will drop, because synthesis strives to minimize area. From an energy point of view, it is positive that fewer and less powerful gates are used. However, relaxed timing constraints lead to an increasing spread in signal arrival times for gates and this has the negative consequence that more glitches are generated.

Figure 9 shows the activity profile for a pipelined FP32 multiplier whose timing constraint is relaxed. We can see that in comparison to the profile in Fig. 8c, the total logic depth increases as a result of area optimizations possible under relaxed timing, that suppression of glitches still is effective, but that more glitches are generated in the integer multiplier.

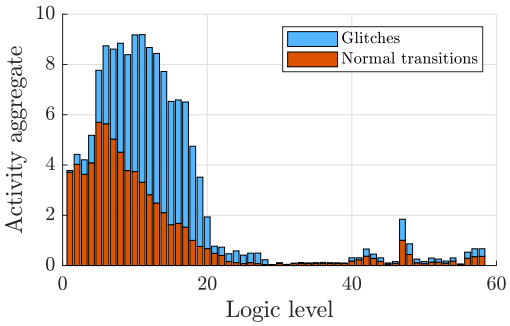


Fig. 9: Activity profile for a pipelined FP32 multiplier implemented for the same timing constraint as the combinational circuit.

VIII. Pipelining for Energy Reductions

The pipelined implementations can work at significantly higher clock rates, under much tighter timing constraints. But if we are going to consider pipelining as a means to reduce energy dissipation, we must compare the pipelined implementation with the combinational implementation at the same performance level in terms of throughput, that is, at the same clock rate, under identical timing constraints.

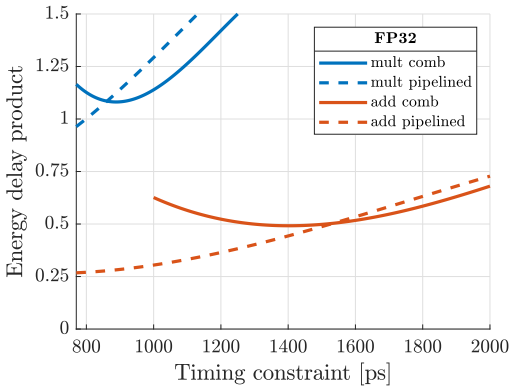


Fig. 10: EDP for combinational and pipelined FP32 circuits.

We show in Fig. 10 the EDP for both combinational and pipelined FP32 multipliers and adders. Clearly pipelined implementations can accept much shorter timing constraints than their combinational counterparts but here we only focus on reduced energy for the same timing performance. As illustrated with the lower dashed line, the EDP curve of the pipelined adder has its minimum at significantly stricter timing constraints. This has the consequence that in the timing region where the EDPs of combinational implementations are minimal, the EDPs of pipelined implementations are quite linear: The reason is that the two combinational logic portions of the pipelined implementations each gets to be synthesized under relaxed timing constraints. From a design point of view, it is in these regions (around 800 and 1000 ps for the multiplier and the adder, respectively) that it makes sense to consider a drop-in replacement, substituting a combinational circuit with an energy-efficient pipelined circuit.

The key observation we can make from Fig. 10 is that pipelining of floating-point adders can lead to a larger energy reduction than pipelining of multipliers. If we use a pipelined adder implementation to make a drop-in replacement for a combinational adder, then we can reduce energy by around 50% when operating close to the timing limits of the combinational implementation. For the multiplier, the energy reduction is limited to 18% and this limitation is in large parts due to generated glitches in the integer multiplier caused by the relaxed timing constraint as shown in Fig. 9.

Figures 11 and 12 show EDP for the implementations of the shorter floating-point formats. As observed already in Fig. 5, for short significand and relatively long exponent field widths, the adder dissipates more energy than the multiplier. This is true also for the pipelined circuits, although the difference is diminishing for stricter timing constraints. For formats like FP16, with a shorter exponent but longer significand, replacing a combinational adder with a pipelined one saves half the

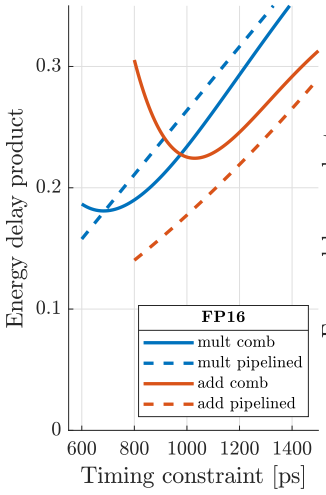


Fig. 11: Pipelined FP16.

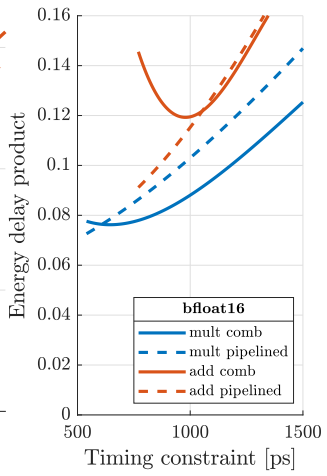


Fig. 12: Pipelined bfloat16.

energy, but for blfloat16 the energy reduction is limited to 40%. For multipliers, the energy savings are not as dramatic but for the three formats considered in this study, we reduce energy by 7–18%.

IX. Impact of Place & Route and CTS

We consistently use post-synthesis data, but we have also explored P&R solutions for different netlists. Since wiring becomes defined, P&R has an impact on many circuit aspects, not least through additional wire capacitances and resistances and their impact on delays. But P&R evaluations are not straightforward as assumptions on neighboring logic, utilization targets, pin placements, etc. have an impact on the results.

The biggest impact P&R has on our comparisons is caused by the clock tree synthesis (CTS) phase, which generates a clock tree with buffers. Since the clock has the highest switching activity, the energy overhead of clocking must not be neglected when considering the pipelined circuits. In our estimates, clock power (buffers and wiring) for pipelining is 1–4% of arithmetic logic power. For strict timing constraints, the number and sizes of clock buffers increase, but so does buffering and sizing of the arithmetic logic.

As discussed in Section III, the ratio of switching activity of logic signals and clock impacts the efficacy of pipelining for energy reductions. For example, if the input data workload has a lower switching activity than our $\alpha = 0.1$ assumption, then the energy cost for the clock tree becomes relatively larger.

X. Conclusion

The aim of this study has been to elucidate some key design tradeoffs involved in energy-aware implementation of floating-point adders and multipliers. Pipelining is known to suppress energy-dissipating glitches, but to our knowledge no design guidelines exist as to what is the best strategy to pipeline floating-point units to achieve high energy efficiency.

We use as metric the energy-delay product (EDP), which captures energy efficiency by combining energy dissipation with performance in terms of delay. For the three different floating-point formats we consider, the multiplier implementations dissipate more energy for formats which use relatively long significand fields and short exponent fields. However, for bfloat16 which uses an 8-bit exponent and a 7-bit significand, the adder dissipates more energy than the multiplier.

For all formats, our EDP results show that pipelining of floating-point adders leads to substantially larger energy reductions than pipelining of multipliers. When we consider how the logic signal transitions are distributed across logic levels, we can make some observations: Multipliers have lower logic depths and many of the signal transitions happen in the first "half" of the circuit, in the integer multiplier. Adders on the other hand have long logic paths and we show that glitches dominate signal transitions in later logic levels of combinational adder circuits.

Pipelining is a common technique to raise system throughput. In this work, we use pipelining to reduce energy. This assumes that the timing constraint for the pipelined circuit is equal to or greater than the minimal timing constraint of the combinational circuit. It is possible to use the timing slack of the pipelined circuit to enhance throughput, but this require an increasing clock rate and leads to higher power dissipation.

Acknowledgement

This project is financially supported by the Swedish Foundation for Strategic Research (SSF).

References

[1] F. N. Najm and M. Y. Zhang, "Extreme delay sensitivity and the worst-case switching activity in VLSI circuits," in *Design Automation Conf. (DAC)*, 1995, pp. 623–627.

[2] A. Raghunathan, S. Dey, and N. Jha, "Register transfer level power optimization with emphasis on glitch analysis and reduction," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 8, pp. 1114–1131, 1999.

[3] H. Eriksson and P. Larsson-Edefors, "Glitch-conscious low-power design of arithmetic circuits," in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, vol. 2, 2004, pp. 281–284.

[4] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std. 754-2019 (Revision of IEEE 754-2008), 2019.

[5] *BFLOAT16 — Hardware Numerics Definition White Paper*, https://software.intel.com/sites/default/files/managed/40/8b/bf16-hardware-numerics-definition-white-paper.pdf, Intel Corp., 2018.

[6] A. Agrawal, S. M. Mueller, B. M. Fleischer, X. Sun, N. Wang, J. Choi, and K. Gopalakrishnan, "DLFloat: A 16-b floating point format designed for deep learning training and inference," in *IEEE 26th Symp. on Computer Arithmetic (ARITH)*, 2019, pp. 92–95.

[7] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, Jul. 2011.

[8] *Cadence® Genus®, v. 23.10*, Cadence Design Systems, Inc., 2024.

[9] *Cadence® Xcelium®, v. 22.09*, Cadence Design Systems, Inc., 2023.

[10] P. Larsson-Edefors, "Energy-efficient computation of TensorFloat32 numbers on an FP32 multiplier," in *IFIP/IEEE 33rd Int. Conf. on Very Large Scale Integration (VLSI-SoC)*, 2025.

[11] *Synopsys® Power Compiler® User Guide, v. 2022.12*, Synopsys, Inc., 2022.

[12] V. Vashishtha, M. Vangala, and L. T. Clark, "ASAP7 predictive design kit development and cell design technology co-optimization," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 992–998.

[13] P.-M. Seidel and G. Even, "How many logic levels does floating-point addition require?" in *Int. Conf. on Computer Design (ICCD)*, 1998, pp. 142–149.