
CHALMERS
UNIVERSITY OF TECHNOLOGY

# Energy-Efficient Computation of TensorFloat32 Numbers on an FP32 Multiplier



Citation for the original published paper (version of record):
Larsson-Edefors, P. (2025). Energy-Efficient Computation of TensorFloat32 Numbers on an FP32 Multiplier. IEEE/IFIP International Conference on VLSI and System-on-Chip, VLSI-SoC. http://dx.doi.org/10.1109/VLSI-SoC64688.2025.11421761

# Energy-Efficient Computation of TensorFloat32 Numbers on an FP32 Multiplier

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

***Abstract*—Several new shorter floating-point formats have been proposed to match requirements of emerging application workloads. To simplify hardware development in the presence of an increasing number of formats, one practical design option is to use as much as possible preexisting hardware, such as standard 32-bit IEEE-754 (FP32) floating-point units, to handle emerging, less complex formats. We evaluate the case where we use an FP32 multiplier to run Nvidia TensorFloat32 data. While the FP32 multiplier area is not as small as a dedicated TensorFloat32 multiplier, we show that energy per operation scales well with the mantissa width reduction and that smart pin assignment can leverage uneven input vector switching activities to significantly decrease energy for reduced precisions.**

## I. Introduction

Machine learning is but one example of important applications that fuel a trend towards floating-point formats with reduced computing precision. The key rationale behind this trend is that shorter formats lead to less complex compute and memory circuits and less interconnects linking them. Choice of format and precision of data representations has a direct impact on resource and energy usage of floating-point units. A case in point is the integer multiplier—a core component of floating-point units—whose complexity decreases quadratically as the precision is reduced.

A number of floating-point formats with reduced precision and/or dynamic range have recently been proposed: IEEE-754 half-precision format (binary16 or FP16), IBM's DLFloat, AMD's fp24, Google's bfloat16, and Nvidia's TensorFloat32 (TF32) are examples of formats which relax the demands on the hardware in comparison to a IEEE-754 single-precision (binary32 or FP32) solution. Requirements on precision and dynamic range vary not only across application domains, but also between workload phases of some applications. Thus, processor floating-point units that can handle mixed-precision workloads, which can be found in machine learning and signal processing, have been proposed [1]–[3].

A circuit implementation customized to one reduced-precision format clearly will perform more efficiently than standard FP32 hardware, which is bound to have higher circuit complexity and area. But the latter one-size-fits-all hardware solution is practical as it can handle many types of formats, as long as they are not more complex than FP32. As we will show in this work, the energy dissipation of FP32 when executing shorter formats can be reduced substantially, to levels closer to FP16 operation than to FP32. We will specifically investigate how TF32 [4], [5] multiplications can be efficiently run on a standard FP32 multiplier.

## II. Background

The IEEE-754 standard for floating-point arithmetic [6] defines a number as $(-1)^S\, 2^{exponent-bias}\, 1.mantissa$, where $S$ (the sign bit) indicates the sign, $exponent$ represents the $w$-bit exponent, and $mantissa$ represents the $t$-bit mantissa (significand). For normalized numbers, the resulting exponent is defined as the subtraction of the coded $exponent$ and a $bias$ which depends on the format: $2^{(w-1)}-1$. Additionally, there exists an implicit 1 which is leading the mantissa bits and makes the resulting $p$-bit mantissa, where $p=t+1$, take on numbers in an interval of [1, 2). The IEEE-754 standard supports also denormalized numbers, in which case an implicit 0 leads the mantissa bits and the exponent is set to 0.

As shown in Fig. 1, IEEE-754's FP32 format uses three different fields with 23 mantissa bits, 8 exponent bits, and one sign bit. It turns out that the half-precision FP16 format, with its 10 mantissa bits and 5 exponent bits, provides sufficient precision for many workloads. The TF32 format proposed by Nvidia is a compromise between FP16 and FP32 [4], [5]: It has the same 10-bit mantissa field as FP16, but shares the 8-bit exponent field of FP32, thus offering the same dynamic range as FP32, with the precision of FP16.

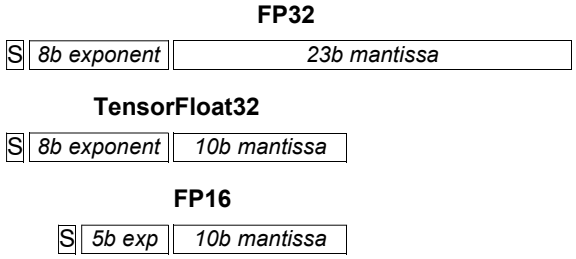


Fig. 1: Floating-point formats under consideration.

As a baseline we also consider fixed-point arithmetic, which is often based on two's complement numbers. Here, an $n$-bit number is defined as $-x_{n-1}2^{n-1}+\sum_{i=0}^{n-2} x_i 2^i$. We can scale this number any way we prefer, using an implicit binary point. This is in contrast to floating point, where the mantissa bits are directly trailing an implicit 0 and 1 for denormalized and normalized numbers, respectively.

## III. Evaluation Method

Floating-point multipliers are implemented for different formats: FP32, FP16, and TF32. We also implement 24-bit and 11-bit fixed-point multipliers as comparison baselines for FP32 and FP16. Their precisions consequently should mirror those of their floating-point counterparts; thus, we let $n=p=t+1$.

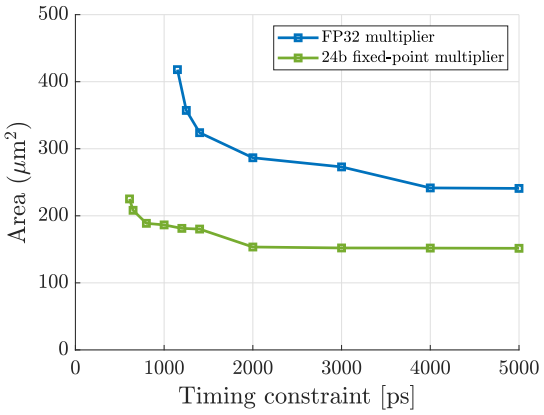


Fig. 2: Area of an FP32 multiplier vs a 24-bit fixed-point multiplier, as function of timing constraint.

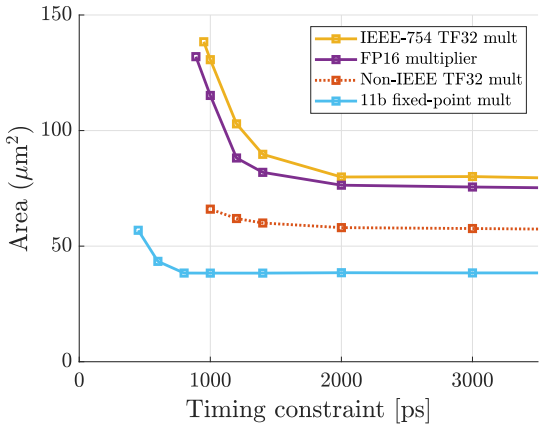


Fig. 3: Area of IEEE754-compliant TF32 and FP16 floating-point multipliers vs non-IEEE-compliant TF32 and 11-bit fixed-point multipliers, as function of timing constraint.

We use Cadence Genus [7] to synthesize the HDL code under different timing constraints. We use regular-VT cells from the open-source ASAP7 library [8], which was developed by Arizona State University and ARM Ltd. to represent a predictive 7-nm FinFET process technology. In general, ASIC logic synthesis strives to optimize area under a timing constraint. Thus, as the timing target is gradually reduced, the area starts to grow when the synthesizer struggles to handle the longest logic paths of the circuit. To make the results as general as possible, we choose to make all circuits combinational.

The implementation metric of area is retrieved after synthesis, but the metric of energy per operation requires more evaluation steps: We generate in Matlab input vector sets for all number formats. For floating point, each set is made up of 10,000 random IEEE-754-compliant numbers, representing normals and denormals, but not NaNs. The three different floating-point fields are created independently and concatenated at the bit level. The fixed-point inputs are more straightforward to generate as there is only one single $n$-bit field. Additionally, the results of reference multiplications performed in Matlab are stored in output vector sets, for fixed-point and IEEE-754-compliant floating-point multiplications, in all formats. All gate netlists are verified for logic functionality using Cadence Xcelium [9], which compares the netlist outputs to the ground-truth reference of the stored output vector sets.

During the above netlist verification, we save switching data for each circuit node and backannotate this to the netlist. Using the system clock rate $f$ as reference, we define $\alpha_i$ as the per node switching activity. This is the fraction of clock cycles when a circuit node $i$ with capacitance $C_i$ switches from 0 to 1. Assuming $N$ nodes, the switching power is defined as $P_{sw} = f\,{V_{DD}}^2 \sum_{i=1}^{N}(C_i\,\alpha_i)$, where $V_{DD} = 0.7\,\mathrm{V}$ is the supply voltage of the cell library. For all energy evaluations, we assume a constant $f = 200\,\mathrm{MHz}$ which corresponds to a relaxed system operating point that satisfies every timing constraint used in our evaluations.

While the use of random input data will overestimate the power dissipation that we can expect in most practical scenarios, in which signal switching is significantly lower, this assumption works here: We focus on comparing energy per operation for multipliers that have the same input switching activity profile. Energy per operation is calculated as $P_{tot}/f$, in which $P_{tot}$ includes an insignificant level of static power.

## IV. Implementation Area

Fig. 2 contrasts the area of an FP32 multiplier with that of a 24-bit fixed-point multiplier, which has the same precision as FP32. As expected, the fixed-point multiplier has a smaller area and it can meet tighter timing constraints. Fig. 3 shows the area of the multiplier implementations of the TensorFloat32 and IEEE's FP16 formats. As comparison we include the area of an 11-bit fixed-point multiplier. The lowest timing constraint of the TF32 multiplier is longer than that of the FP16 multiplier. This is due to TF32's exponent field being wider than FP16. The wider exponent field is also the reason the TF32 multiplier is slightly larger than the FP16 multiplier. The graphs show that the delay overhead of using floating-point over fixed-point for multipliers is close to 2X.

We consider multipliers which are compliant with IEEE-754. But as shorter floating-point formats are being pursued, some features in conventional formats may be removed to reduce hardware complexity. For example, in bfloat16 [10] denormals are not supported but input and output numbers are flushed to zero. To illustrate this option, we include in Fig. 3 also a variant of TF32 which is not compliant with IEEE-754, but lacks leading zero anticipation and normalization features.

## V. Energy per Operation

The previous section showed area results for several multiplier implementations. A widespread assumption is that power dissipation scales linearly with area, but there are several caveats to this simplified view. For example, glitching power, due to unwanted spurious signal transitions in gates with unbalanced input arrival times, will make total switching power depend also on logic functions and computing workload.

XOR-intensive arithmetic circuits are known to have excessive glitching power dissipation. As shown in [11], the glitching power can constitute more than half the total power of a 16-bit fixed-point complex-valued multiplier. Adding pipeline registers to an arithmetic circuit is a known remedy to glitches as this stops them from propagating [12]. However,

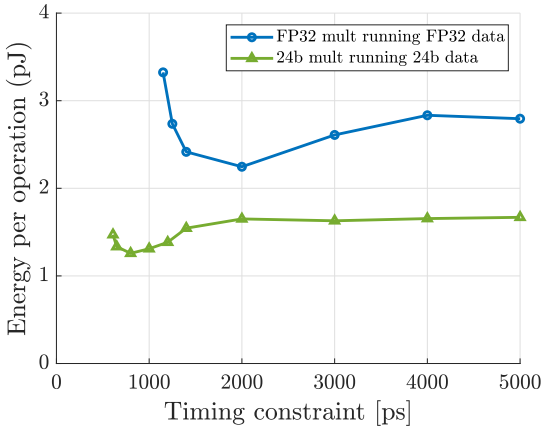


Fig. 4: Energy per operation of FP32 and 24-bit fixed-point multipliers, as function of timing constraint.

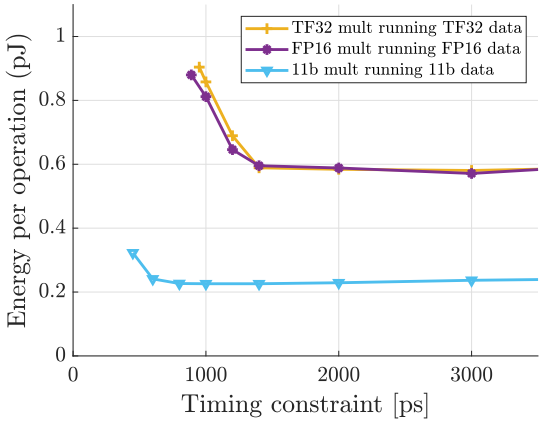


Fig. 5: Energy per operation of TF32 and FP16 floating-point multipliers and an 11-bit fixed-point multiplier.

using an extra pipeline stage is not straightforward in latency-restricted architectures and systems using feedback loops.

Fig. 4 shows the energy per operation for an FP32 multiplier operating on FP32 numbers and a 24-bit fixed-point multiplier operating on 24-bit two's complement numbers. As shown in this figure, the energy per operation decreases for tighter timing constraints, before the area begins to grow fast (Fig. 2). The reason we observe the decreasing energy dissipation is because for tighter timing constraints, the logic paths become more balanced in terms of timing: In logic gates where inputs arrive simultaneously, the generation of glitches is inhibited.

Fig. 5 shows the energy per operation for two floating-point multipliers (TF32 and FP16) and an 11-bit fixed-point multiplier. Here, TF32 operates on TensorFloat32 numbers, the FP16 multiplier on FP16 data, and the fixed-point multiplier on 11-bit two's complement numbers. There is a small but clear area difference between TF32 and FP16 in Fig. 3. However, except for the tightest timing constraints, their energy per operation is almost identical. This is because in floating-point multipliers, the exponent computation is less complex and less power dissipating than the mantissa computation [13].

## VI. Using Reduced-Precision Workload Data

We will again perform an energy evaluation, but now we will include cases where we apply data with shorter floating-point formats to an FP32 multiplier. Fig. 6 shows the energy per operation for an FP32 multiplier operating on TF32 and FP16 data, respectively. We have also included two curves from previous graphs, viz. FP32 running FP32 data from Fig. 4 and TF32 running TF32 data from Fig. 5.

Clearly input data with lower precision (mantissa) and dynamic range (exponent) leads to significantly lower energy dissipation. As shown in Fig. 1, TF32 numbers have a narrower mantissa field than FP32 numbers, whereas FP16 numbers have both narrower mantissa and exponent fields. Since the two cases where FP32 is running shorter formats yield almost the same energy per operation, we can make a useful observation when using workloads with shorter floating-point formats: Reducing the number of mantissa bits in the input data has a greater effect on energy dissipation than reducing the number of exponent bits. This is because the exponent bias changes, which impacts the bit information in this field.

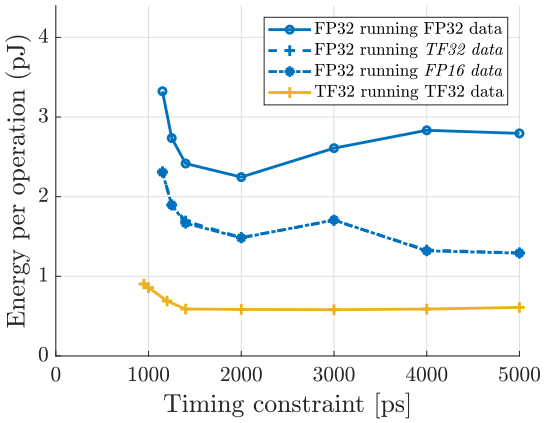


Fig. 6: Energy per operation strongly depends on input data precision.

### A. Pin Assignment for Reduced-Activity Data

It is well known that when input data to fixed-point multipliers have different dynamic ranges, the power dissipation can be reduced if we assign data to the 'right' input pins. As shown previously [14], [15], a lower dynamic integer range (which leads to longer strings of consecutive '0' or '1' in the most significant bit positions) can be exploited in fixed-point multipliers where Booth recoding of bit patterns like '000' and '111' lead to partial products which are evaluated to zero. Since they are implemented in a different way, floating-point multipliers cannot exploit dynamic range reductions the same way fixed-point multipliers can. But as shown in Fig. 6, a reduced mantissa precision, which implies consecutive '0' in the *least* significant bit positions of the mantissa integer, leads to reduced energy.

In many applications, multiplications are performed on data input vectors which are different in how frequently their bits switch. This is often the case where weights or coefficients are changing more slowly than the data they operate on. This difference in switching activities has been previously used for low-power design of multipliers [16], FIR filters [17]

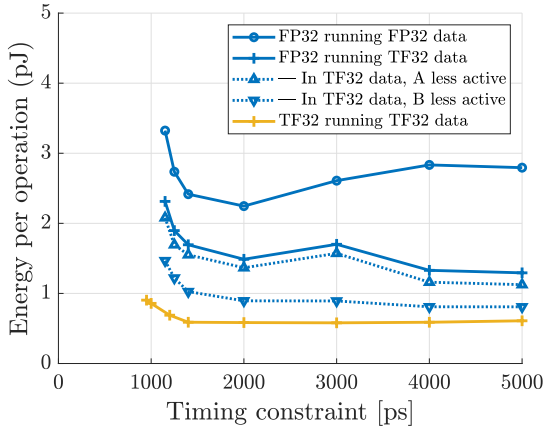


Fig. 7: Energy per operation depends on input switching activity.

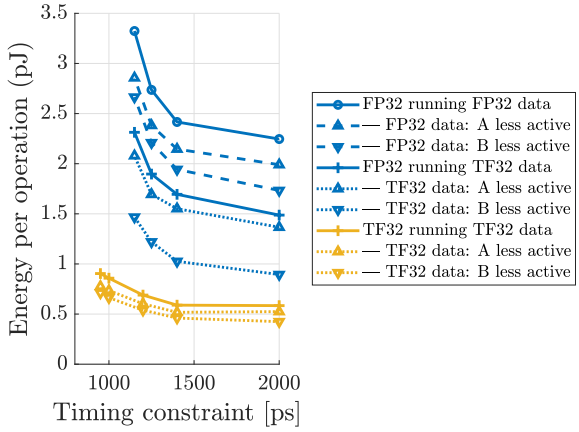


Fig. 8: Impact of pin assignment on energy per operation for performance-oriented implementations with strict timing targets.

and FFTs [18]. In [19], we unify the two approaches of pin assignment—using dynamic range and switching activity—for fixed-point multipliers for complex numbers.

The effect of pin assignment on an FP32 multiplier running TF32 data is shown in Fig. 7. We show as references (top and bottom) the previous curves (Fig. 6) for the FP32 multiplier running FP32 data and the TF32 multiplier running TF32 data. In addition, we show as baseline the energy per operation for the FP32 multiplier as it operates on TF32 (Fig. 6). Then we add two curves which show the effect of reducing the switching activity of one of the input data vectors by 4X: We assign this reduced-activity data to, in the first simulation, multiplier pin A. In the subsequent simulation run, we swap the inputs so that the reduced-activity data goes to pin B. As shown, we substantially reduce energy per operation when the slowly changing data is assigned to pin B.

Pin assignment can also be applied to the reference cases. Fig. 8 includes the result of pin assignments also for the FP32 multiplier running FP32 data and the TF32 multiplier running TF32 data. Optimal pin assignment has a favorable impact on energy, but the gain is the largest when we reduce precision, making an FP32 multiplier running TF32 data approach the energy efficiency of a dedicated TF32 multiplier.

## VII. Conclusion

We evaluate using standard FP32 floating-point multipliers for TensorFloat32 workloads which use less precision than the FP32 format. While using a one-size-fits-all FP32 multiplier also for numbers with less complex floating-point representations comes with an area overhead, we show that energy per operation substantially decreases with a reduced mantissa precision in the workload. Additionally, we show that optimally assigning input data that have different switching activities to the input pins has a large impact on energy dissipation when the precision of the workload is reduced.

## Acknowledgement

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

## References

[1] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "LNPU: a 25.3TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2019, pp. 142–144.

[2] A. Nannarelli, "Variable precision 16-bit floating-point vector unit for embedded processors," in *IEEE 27th Symp. Computer Arithmetic (ARITH)*, 2020, pp. 96–102.

[3] H. Tan, G. Tong, L. Huang, L. Xiao, and N. Xiao, "Multiple-mode-supporting floating-point FMA unit for deep learning processors," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 2, pp. 253–266, 2023.

[4] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 tensor core GPU: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[5] D. Stosic and P. Micikevicius, "Accelerating AI training with TF32 tensor cores," https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/, Nvidia.com, 2021, Accessed Aug. 27, 2025.

[6] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[7] *Cadence® Genus®, v. 18.14*, Cadence Design Systems, Inc., 2019.

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

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

[10] *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.

[11] P. Larsson-Edefors and E. Börjeson, "Implementation evaluation of fixed-point multipliers for complex numbers," in *IEEE 32nd Symp. Computer Arithmetic (ARITH)*, 2025, pp. 81–84.

[12] A. Chandrakasan and R. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, no. 4, pp. 498–523, 1995.

[13] L. Bertaccini, G. Paulin, T. Fischer, S. Mach, and L. Benini, "MiniFloat-NN and ExSdotp: An ISA extension and a modular open hardware unit for low-precision training on RISC-V cores," in *IEEE 29th Symp. Computer Arithmetic (ARITH)*, 2022, pp. 1–8.

[14] P.-M. Seidel, "Dynamic operand modification for reduced power multiplication," in *Asilomar Conf. on Signals, Systems and Computers*, vol. 1, 2002, pp. 52–56.

[15] N.-Y. Shen and O.-C. Chen, "Low-power multipliers by minimizing switching activities of partial products," in *IEEE Int. Symp. Circuits and Systems*, vol. 4, 2002, pp. 93–96.

[16] P. Larsson-Edefors and E. Börjeson, "Activity-based input operand assignment for reduced multiplier power dissipation," in *IEEE Latin American Symp. Circuits and Systems (LASCAS)*, 2025.

[17] C. J. Nicol and P. Larsson, "Low power multiplication for FIR filters," in *Int. Symp. Low Power Electronics and Design*, 1997, pp. 76–79.

[18] O. Meteer and M. J. G. Bekooij, "Low-power Booth multiplication without dynamic range detection in FFTs for FMCW radar signal processing," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conf.*, 2021, pp. 44–48.

[19] P. Larsson-Edefors and E. Börjeson, "Low-power complex multiplier pin assignment based on spatial and temporal signal properties," in *IEEE Int. Symp. Circuits and Systems (ISCAS)*, 2025.