Extension of TangramFP MAC unit from FP16 into FP32

Abdelmojeb Mohammednour

Supervisor Yuan Yao

1. Abstract

As energy consumption is a big concern currently, Tangram FP16 was developed. It is an optimized MAC unit where the multiplication process of the operation (a*b + c) is optimized according to how much expected rounding will happen after the addition operation. It has shown significant improvement in performance and power consumption while maintaining after-synthesis hardware utilization. In this project the model was extended to operate on 32-bit floating-point numbers. In the first part the model of MAC was mimicked in c code to test numerical accuracies along different operation modes. The accuracy performance was measured in Unit in the Last Place ULP. The simulation reported 99.9989% of the operations had ULP < 0.5. The model of the MAC unit was designed in HDL and simulated for the same number of test operations. Static power analysis was reported for each mode of multiplication with power reduction 2.27% at Skip_BD, 13.64% at AC_only and 36.36% at Skip mode.

2. Introduction

In modern AI applications significant amount of multiply-add operations lose some of the multiplication result accuracy due to shift alignment of the addend as a result of exponent differences, moreover large part of this loss squash way the multiplication result because the added to number (the sum) in the accumulation process is too big that the addition operation is just the sum itself with no change which renders the total operation useless at that point. [1]

Many strategies were applied to utilize such phenomena to tackle another major problem in the deep learning field, which is the higher demand of power required by deep learning network during training and inference processes. Many solutions were applied in accelerator MAC designs to tackle the problem. The high cost of data movement led to various solutions involving increasing on-chip memory, however the energy cost of computation increases. [1]

As detailed in the TangramFP for 16-bit implementation, there are multiple approaches to optimizing power consumption in computation. Either through parallelism or applying approximation in the serial bit computation, as the parallel approach lacks flexibility with dynamic change in the precision and the serial implementation has a major drawback in performance due to cyclic operation and data dependency.

TangramFP proposed an approach of detecting the ineffectual computation parts of the multiplication omits the corresponding multiplication on runtime. The idea of Tangram stems from the property in the floating-point numbers that Big + Small = Big i.e. when two floating point numbers are added the exponent of the two numbers are compared to be aligned and the mantissa of the smaller number is shifted to the right with the amount of the exponent difference. The result of the shift and exponent alignment will reduce the precision of the number. In the case that the smaller number is the result of the multiplication then part of the multiplication

product will be wasted and the energy to produce it will not be for effective result. In normal multiplication there is no way to separate the part that will later be shifted out in the addition stage, but in Tangram the multiplication of the two mantissas is divided into smaller partial products added together to produce the result as shown in figure 2. The Tangram first compares the exponents of the multiplication result through the operands a and b by adding them and the accumulated result then determines which multiplication operation is to be activated, consequently the corresponding partial product is skipped in the multiplication. This approach does not just reduce power consumption but is flexible and seamless to be applied in various ways of implementation in serial design or parallel without requiring deep change in the existing design.

TangramFP 16-bit demonstrated adherence to the same bounds as the standard IEEE FP16 measured in ULP and delivered better precision than a state of the art approach based on bit-serial truncated multiplication aimed at eliminating ineffectual computations in DNNs. Tangram is a drop-in module having approximately the same mean ULP error, the same area and latency, while achieved up to 36.57% dynamic power savings.

In this work the aim is to apply the same design into 32-bit scale with the same structure of the tangram and test its performance simulation and its power saving in the hardware design.

3. Tangram for 32-bit

in the TangramFP the floating-point representation of the IEEE-754 is adopted for 16-bit is in this report the same standard is used for FP-32. The IEEE-754 standard stores the floating-point numbers in three sections, 1 bit for the sign of the number, the second section stores the exponent with a bias depending on the precision of the number to have the range form 0 up to double the maximum exponent that the FP-format covers. In the 16 bit the exponent ranges from 0 up to 30 in 5 bits length with a bias value of 15 so the FP16 covers exponent range from -14 to 15. In addition to special cases, first when the exponent is all 1s meaning the number either NaN or infinity based on the state of the mantissa, second when the exponent is 0 the exponent is interpreted as -14 with no hidden significant bit in the mantissa. The mantissa of the IEEE-754 represents the fraction of the floating point number after put in the binary form 1.### where the whole number 1 is removed and considered as hidden significant bit.

In FP32 the exponent bit width is 8, giving an exponent range from -126 up to 127 with subnormal values with exponent -126.

In Tangram the choice of IEEE-754 format gives a relatively long bit width for the mantissa which gives an opportunity to utilize the tangram process to increase the precision in the approximation of the multiplication while minimizing the power consumption.

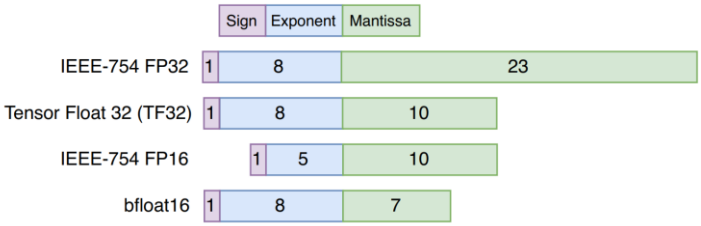

Figure 1FP formats for DNN training. [1]

The main idea of tangram is to divide the mantissa of the two multiplication operands into smaller parts. X * Y where X is segmented into 1:A:B and Y into 1:C:D where the length of A,C is p and the length of B,D is q then the multiplication operation is performed as

$$2^{2(p+q)} + (X+Y) * 2^{p+q} + A * C * 2^{2q} + (A * D + B * C) * 2^{q} + B * D$$

Each term in the equation above corresponds to a mode of operation representing an expected alignment shift that will affect the term or part of it. When the exponent difference between the multiplication results and the added to value of the sum is higher than 0 but less than first threshold then the B*D term is discarded and its multiplication is skipped, the mode of operation is called Skip_BD, and when the difference is between threshold 1 and 2 the B*D and (A*D + B*C) are skipped with the mode AC_only. Whenever the difference is greater than threshold 2 then the entire multiplication is skipped as the result will likely yield a small number when added to the sum it results in the sum itself.

TangramFP proposes different forms of segmentation of the mantissa with levels for modes of operation for the approximation

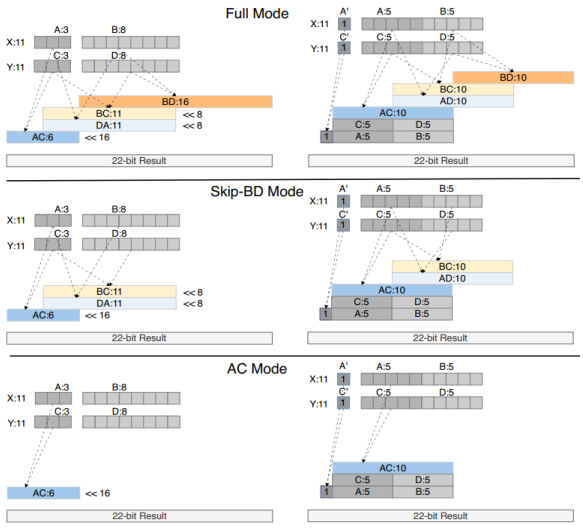

Figure 2 3:8 and 1:5:5 splits featuring three modes of operation: i) Full Mode; ii) Skip-BD Mode; iii) AC Mode. [1]

In Tangram32 the same mechanism is implemented with dividing the mantissa into two parts in addition to the hidden significant 1 with p = 12 and q = 11 as close extension to the split 1:5:5 in the 16 bit Tangram. The following figure illustrates the multiplication stages

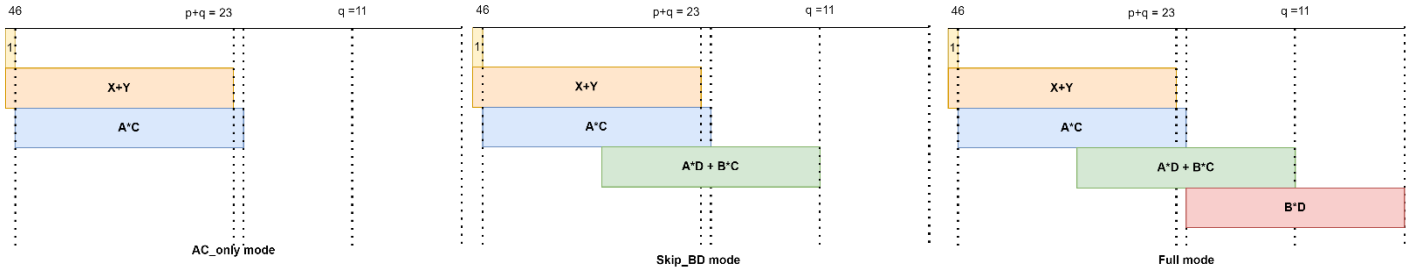

Figure 3 Addition stage of the first three modes

Every term in the equation is multiplied selectively and separately in a DADDA reduction matrix then added as illustrated in the figure above.

When the input of the operation a*b + c in which a or b is a subnormal number, the multiplication the result will likely depend on the other input so the subnormal number can be approximated to be the lowest normal number in the floating-point standard by replacing the zero exponent with 1 representing the lowest exponent. Then the mantissa is replaced by zero meaning the old mantissa is rounded up. This conversion adds up the hidden significant 1, and the operation mode then is determined according to the difference between the a*b exponent and the c exponent. On the other hand, if the c is the subnormal number, then it will be converted to the lowest normal number in the same way above, but the mode of multiplication will be Full mode as the result totally depends on the result of the a*b multiplication.

When the exponent of any of the inputs is all ones this indicates that either the number is NaN or infinity in any way this means that the whole operation will result in a NaN value in that case the the whole operation multiplication and addition is skipped and the NaN directed to the output, similarly if the exponent of the multiplication is exceeding the maximum exponent covered by the standard FP such as above 254 in FP32 that means the final result will be infinity. In this case the mode skip will be selected, and the addition is avoided, and the MAC unit will give back infinity as an output.

4. Simulation of the MAC unit

The purpose of the simulation is to test the numerical viability of the Tangram32 organization, especially when approximation modes are applied. The simulation is written in C to mimic the hardware MAC unit process in multiplying and adding floating point numbers.

4.1. Simulation structure

the main parts are MAC unit and testing and evaluation part, where the MAC is the part responsible for processing the numbers as if it's the hardware module while testing and evaluation is responsible for feeding the unit with data and receives the result then it computes the ULP error in addition to statistics of each mode of multiplication that is triggered in the MAC unit along with average ULP of that mode.

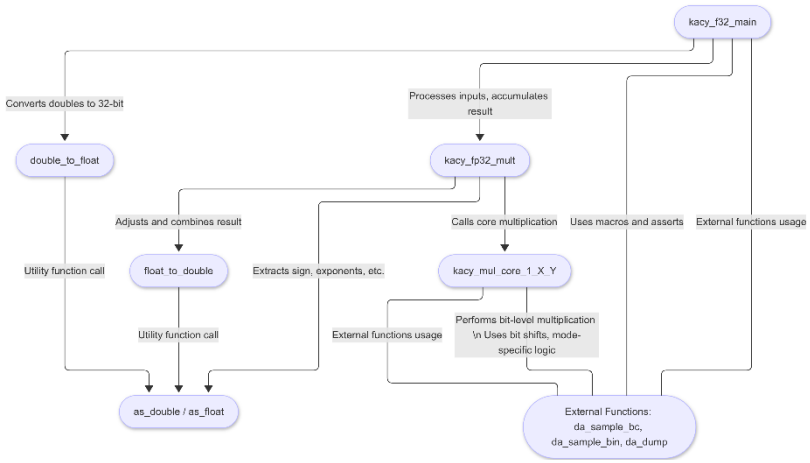

Figure 4 structural diagram of the MAC multiplier simulation

The MAC unit is organized as follows

The main function takes in the a, b, c inputs in the form of vectors to compute a dot product then test each elements status ; if zero then a 1 is added to the zero vector corresponding to the input in the same position of the element for later mode selection, similarly the elements are checked for the normal/subnormal type, if an element is subnormal then the exponent is replaced with 1 and the mantissa is set to zero. If any of the elements are infinity or NaN then the code terminates as the whole operation will result in a NaN value.

For testing the accuracy of the multiplication along various modes there was no need for performing dot products to calculate the ULP although it was performed in a testing stage in the development. Individual operation where each time a, b and c were selected gave more flexibility into testing various scenarios including edge cases, also calculating ULP for each operation independently gives more accurate picture about the performance of the MAC and can indicate directly into the accuracy of the operation which enables looking at the contributing numbers

into a specific result rather than be buried in the crowed of the other elements of the two input vectors and the initial sum input.

After the first round of checking the numbers, the maximum exponent is determined from the a, b vectors and the initial sum input. This maximum exponent is used each time two numbers are extracted from their vectors for multiplication to compute exponent difference by subtracting the

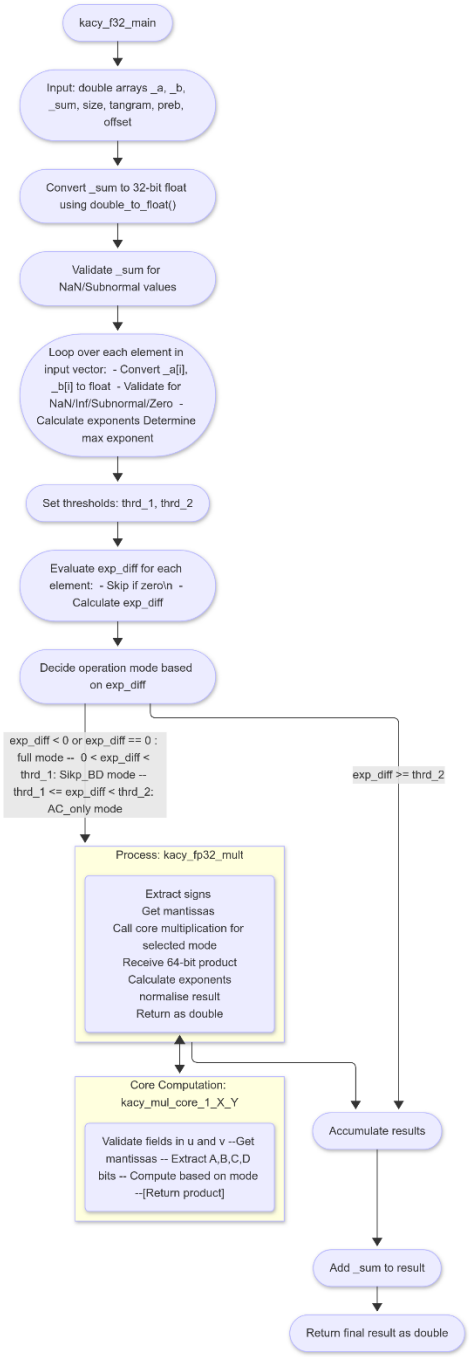

Figure 5 flow chart describing floating point multiplication

a*b exponent from the max exponent to determine the multiplication mode. Then according to

the difference, the multiplication function is called and the a, b, mode and the operating cut is passed. the cut defines the length of the segmentation of the mantissa for example in the TangramFP 16-bit it could 5 meaning the Tangram formation of the mantissa will be 1:5:5 for 1:A:B or 8 then the formation will be 1:3:8. Here the cut is set to be 11 leading to 1:A:B be 1:12:11.

The multiplication function "*kacy_fp32_mult*" extracts the signs, exponent and the mantissas of the two numbers then computes the sign of the result and the exponent of the result. For the mantissa operation it adds the hidden significant 1 to the two mantissas increasing the length into 24 bits then pass them into the core multiplication function "*kacy_mul_core_1_X_Y*" with the mode and the cut for the Tangram multiplication. The core function segments the two mantissas then selects which terms will be multiplied and combined according to the mode the returns the result to the parent function.

The multiplication function "*kacy_fp32_mult*" takes the product and normalizes it based on the leading zeros before the first one, which in this case will be either in the 47^{th} bit or the 46^{th} bit because all input were normal FP. The leading 1 is removed from the mantissa to prepare for standard IEEE-754 floating point construction. The function also adjusts the result exponent accordingly. After normalization the floating-point result is reconstructed according to the standard *sign:exponent:mantissa; 1:11:52*.

It is possible to generate the result in the FP32 form rather than FP64 but there is a good reason for that, that is to defer the approximation after the addition or defer it after the dot product is finished in both ways enhance the precision of the result and will not require significant increase in the hardware. Otherwise downgrading the precision at each multiplication will result in repeated approximation as the dot product progresses which might counter the benefit of the Tangram or at least reduce the precision that could be gained.

4.2. Testing and evaluation

To test and evaluate the MAC a testing module was developed. It generates random numbers of a, b, sum with constraints for a that it is in the range of FP32 while b is constrained to be a value when multiplied with a does not exceed the range of FP 32 on the sum value the constraint was to have a value that allows for different multiplication modes but not to have the freedom to get more values that result in a difference that is greater than the mantissa length to limit the probability of having Skip mode always or dominating the mode selection to give a fair chance of for the other modes to happen.

In each iteration the three numbers are generated in FP32 then an expected value is calculated, and the generated numbers are fed into the MAC unit. After the MAC returns the result, the error is computed in ULP unit with dedicated function that calculates the ULP step for each result. Another function is dedicated to count each mode will be triggered by the MAC and logs the count and accumulated ULP for mode separately. After that in each iteration if the ULP is higher than 0.5 for a single operation the result with the inputs and the expected value are logged for individual check manually. The test runs in a loop iterating for the intended number tests. The

overall average ULP, statistics of the mode occurrence and the average ULP for each mode are computed and logged as the result for the test.

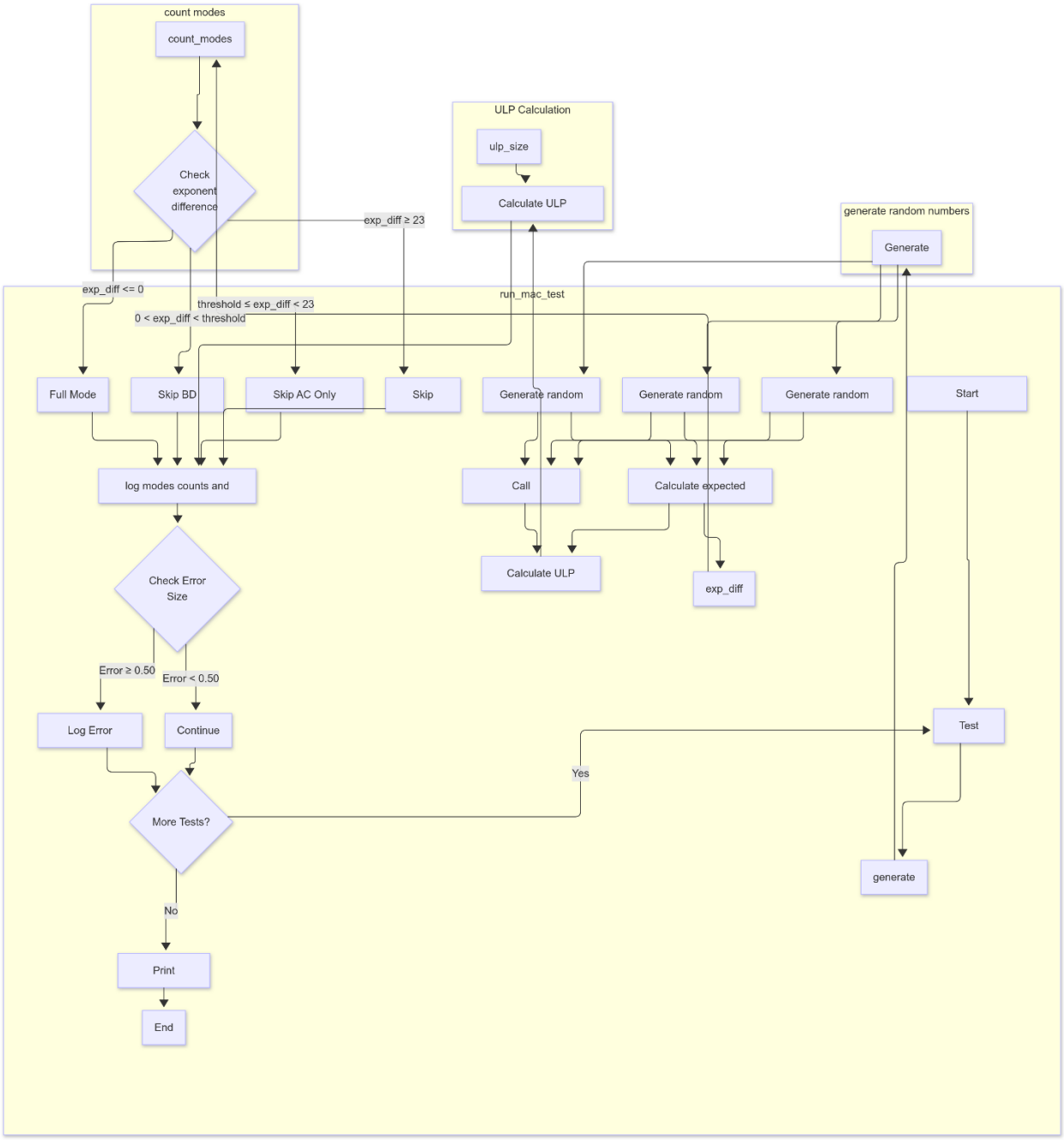

Figure 6 test and evaluation code flow chart

4.3. Simulation results

The test was run for 1000000 iterations each time 3 numbers are generated and fed into the MAC unit. The result shows promising performance with an overall ULP average 0.4 detailed in the following table

Test Summary:

Number of tests	Maximum ULP error	Average ULP error	Number of errors>0.5
1000000	1910.00	0.04	11

ULP per modes

mode	Full	Skip_BD	Only_AC	Skip
ULP per mode	0.088802	6.952e-310	0.031913	0.076431
Count mode	494348	158013	189829	157810

Error distribution:

0-0.5 ULP	99.9989%
>0.5 ULP	0.0011%

mode	Test	ULP Error	exponent diff
Skip_BD	777997	1910.00	1
Expected	1.093597712387151e-10		
Actual	1.0933326466400217e-10		
Test Inputs			
a	-62434175690423140352		
b	3.5440654687822812 2102e-26		
sum	2.2125987015897408127785e-06		
a_mantissa	10110001001110010110000		
b mantissa	01011110111111001011110		
ulp size	1.3877787807814456755295e-17		
actual - expected	2.6506574712925612402614e-14		

As shown in the tables above the results are quite impressive with an average of less than 0.5, however it is important to discuss the occasions that the ULP exceeded 1. There are 11 points out of 1000000 in all these occasions the mode was Skip_BD. In all cases the mode was Skip_BD with exponent difference = 1. In the last table the maximum ULP occasion is examined. The high ULP comes from the multiple factors. First, since the cut value is 11 bits and activated when the difference is > 0 and less than 11, in this case the difference = 1 Skip_BD mode activates. Second, the mantissa lower bits in both numbers have significant contributing information hence discarding BD term (multiplication of the two red colored bits) will increase the discrepancy. Third, the multiplication result is small value and added to a small value which will result in a floating-point number in a range with high representation of numbers which makes a difference in 1 bit result in a different number. The ULP is calculated with the formula

abs(actual - expected)/ulp_size

The ulp_size is determined from the expected value as it depends on the exponent of the reference number. The ulp_size in this case is very small value even with small difference between the actual and the expected still the result ULP is very high. For more soft errors above

ULP unit 0.5 or 1 Tangram can be tweaked to adjust the cut or the thresholds with which these modes are triggered, this is handy by choosing different values for the cut or setting the offset to a value higher than 0. In this test neither the cut was changed, nor the offset was given a value other than 0. This is left for future work to investigate the best configuration of the Tangram32.

5. Hardware design

The design of the Tangram32 hardware followed the same organization of the simulated MAC in the software. In the core multiplier a DADDA multiplier is designed for its efficiency and

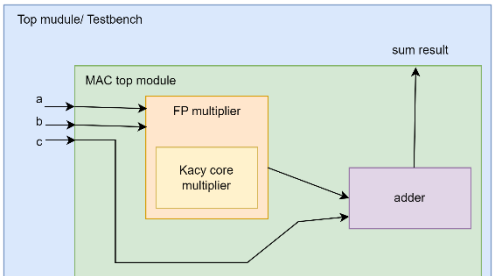

Figure 7 MAC unit block diagram

performance in multiplication. Each term in the Tangram terms is associated with a separate DADDA multiplier which is activated when the mode requires its result. Additionally, to DADDA multipliers there are adders to combine the result, some of these adders also are activated only when their associated mode is selected. The core multiplier processes only the mantissa multiplication directs its result to the higher entity in the structure where the other parts of the floating-point number are processed and the exponent with the mantissa is normalized.

5.1. Hardware structure

1- MAC top module: this module receives the input from the wrapping interface then it performs preliminary operations to check for subnormal, zero, NaN/infinity values in the inputs and sets the corresponding flags. Then determines the mode of the multiplication and passes the inputs to the FP multiplier. It calculates the ab exponent to define the mode and based on the flags decides whether to pass input to the multiplier, adder or bypass any or all of them. If a or b is 0 then the multiplier is bypassed and the result is the sum, if the other way around then the multiplication is the result and the adder is bypassed. If any of the inputs in NaN/infinity or the ab exponent exceeded FP range, then the all-sub modules are bypassed, and the result is NaN.
After the multiplication is computed it takes the normalized result with the sum to the adder then returns the accumulated result to the interface/testbench

2- FP multiplier: receives the inputs form the MAC in the forms of ab exponent and the mantissa with the hidden significant 1added to it in addition to the mode. The FP multiplier passes the mantissas to the core multiplier and receives the result from it. When it receives the result, it counts the leading 0 to normalize the mantissa and the ab exponent and return them to the MAC.

3- Adder the adder receives the exponents, signs and the mantissas of the ab and the input sum then determines if the operation is addition or subtraction based on the sign of both

inputs. When addition is chosen, the result is normalized based on the carry existence in the case of a carry the mantissa is shifted and leading 1 is removed with adjustment to the exponent. However, if the subtraction is chosen the situation of an overflow borrow is avoided by ordering the subtracted and the subtrahend then there may be chance for variable number of leading zeros. The result of the subtraction is passed to a leading zero counter LZC sub module to count how many leading zeros in the mantissa before the first one. The adder takes counter value to normalize the result and construct the FP number.

5.2. Design development

The design was approached from bottom-up approach with DADDA and associated adders are designed first and verified with testbench then FP multiplier and the adder, finally the top MAC unit with each module is tested separately and with its instantiated sub modules

During the hardware design the principle of choosing the efficient architecture was followed by, in this point it is important to note that efficient for the synthesizer tool to choose lower number of consecutive LUTs is not the same as choosing best architecture for ASIC designer as for the synthesizer requires a style of coding that takes into account the available resources in the board and the max size of the LUT. This is obvious when designing, for example the LZC and choosing the carry look ahead adders/subtractors. Despite the look ahead adders are considered efficient, especially when adding more than two numbers it has been discovered it is better to allow the synthesizer to choose the best way to implement addition or subtraction then it will utilize available carry chain resulting in a shorter data path.

Since the design was intended for high speed a higher frequency clock is chosen, this led into reconsideration of the hardware timing design as it was intended to produce a result in one clock cycle. But the complexity and density of the combinational data processing at different levels leads to a long critical path. This is mitigated by splitting the design time into two stages multiplication stage in one clock cycle and the addition performed in another clock cycle with pipeline in which result production takes two clock cycles, and the new result comes out every clock cycle.

The simulation of the adder and the multipliers core and FP were tested for numerical accuracy. The top MAC unit testing followed the same approach that was used in the software simulation where three random numbers are generated and passed to the MAC DUT followed by comparing the result to an expected result. However, since there is limitation in the HDL when it comes to floating point manipulations and precision options, the numerical test was conducted for the basic and obvious mistakes for debugging the design. While the testbench primary advantage to the testing is to verify the timing and functional correctness of the module under test.

The design was synthesized and simulated post synthesis for functional and timing simulations. Additionally simulated post implementation for only functional correctness.

The main target after hardware design was to seek information about dynamic energy consumption of the Tangram multiplier.

5.3. Power Analysis

After implementing the design, a static power analysis is conducted by implementing the whole design then running static power analysis in Vivado and reporting the dynamic power consumption then repeating the operation with removing the parts that are connected to produce the BD, AC_only, skip terms to simulate power reduction when these modes are activated.

The power reports are summarized in the following table

Mode	**Total On-Chip Power**	**Dynamic Power**	**Device Static Power**	**Clocks Power**	**Signals Power**	**Logic Power**	**I/O Power**
FULL_MODE	0.203 W	0.044 W (22%)	0.158 W (78%)	0.002 W (4%)	0.010 W (23%)	0.011 W (24%)	0.022 W (49%)
SKIP_BD	0.201 W	0.043 W (21%)	0.158 W (79%)	0.002 W	0.009 W (21%)	0.010 W (24%)	0.022 W (51%)
AC_ONLY	0.196 W	0.038 W (19%)	0.158 W (81%)	0.002 W (4%)	0.007 W (19%)	0.008 W (20%)	0.021 W (57%)
SKIP_ALL	0.186 W	0.028 W (15%)	0.158 W (85%)	0.001 W (5%)	0.003 W (12%)	0.003 W (10%)	0.020 W (73%)

The power reduction for each mode with respect to the full mode is detailed in the table below

Mode	Total Power Reduction	Dynamic Power Reduction
SKIP_BD	0.99%	2.27%
AC_ONLY	3.45%	13.64%
SKIP_ALL	8.37%	36.36%

The static power analysis shows significant reduction in power consumption especially with AC_only and Skip modes. Considering deep learning training with its power intensive computations where these modes are not rare this result shows a considerable improvement in power consumption is expected. The device's static power remains constant at 0.158w because the design is intended to be deployed on FPGA board particularly Kintex-7. In the FPGA board the leakage power or the device's static power depends on the FPGA board regardless of how much the utilization of the board resources.
It is needless to say that the static power analysis is not accurate and more accurate analysis with suitable hardware is necessary to get more accurate analysis about the reduction in power consumption would be achieved from activating these modes. Nevertheless, this analysis shows that Tangram32 is promising in power consumption performance. And more accurate analysis is left for future work.

6. Conclusion

This work successfully extends the TangramFP MAC unit from FP16 to FP32, maintaining its core principles while adapting to the larger bit-width format. By leveraging the segmented multiplication approach, Tangram32 preserves numerical accuracy while achieving notable reductions in power consumption. Simulation results demonstrate that 99.9989% of operations

maintain a ULP error below 0.5, highlighting the precision of the proposed design. Additionally, static power analysis indicates reductions of up to **36.36%** in dynamic power consumption when utilizing the Skip mode, further proving the effectiveness of this approach for energy-efficient computation.

While the results are promising, further work is needed to refine the mode selection thresholds and optimize hardware synthesis for improved power and performance trade-offs. Moreover, dynamic power analysis with real-world workloads would provide a more accurate assessment of energy savings in practical scenarios. Future research can also explore adapting Tangram32 for custom AI accelerators and other low-power computing applications.

Ultimately, this study confirms that the Tangram approach remains a viable and scalable solution for efficient MAC operations, paving the way for its broader adoption in high-performance and energy-conscious computing architectures. *

7. References

[1] Y. Yao, X. Chen, H. Atmer and S. Kaxiras, "TangramFP: Energy-Efficient, Bit-Parallel, Multiply-Accumulate for Deep Neural Networks," in *2024 IEEE 36th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Hilo, HI: IEEE, 2024, pp. 1-12.

* Written with assistance with ChatGPT.