Approximate Least Squares Accelerator

MSc Thesis

Alexander Krapukhin

Committee:
Dr.ir. A.B.J. Kokkeler
S.G.A. Gillani
Ir. J. Scholten

Computer Architecture for Embedded Systems
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
Enschede
The Netherlands

January 2019

Посвящаю бабушке Анете

Contents

Abstract **4**

Introduction **5**

1. Literature review **7**

1.1. Error resilience analysis. 7

1.2. Approximate computing techniques. 12

2. Multiply-accumulate unit and its approximation **15**

2.1. Least Squares Problem 15

2.2. Multiplier-accumulator (MAC) 16

2.3. Unit gate model 18

2.4. Errors and error metrics. 19

2.5. Mitigation of errors. 23

2.5.1. Mean error balancing. 23

2.5.2. Using two multipliers with opposite errors. 23

2.5.3. Accumulator initialization. 24

2.6. Approximate units against careful data sizing. 25

2.7. Approximate multipliers for MAC. 26

2.7.1. Recursive approximate multipliers 26

2.7.2. Dynamic Range Unbiased Multiplier (DRUM). 34

2.7.3. Low-Power Approximate MAC Unit. 36

2.7.4. Approximate Booth multiplier. 37

3. Comparison of approximation methods **39**

3.1. Experimental setup for comparisons. 39

3.2. Comparison results. 44

3.2.1. M1 against M2 type 45

3.2.2. MSE minimization against ME minimization. 47

3.2.3. MSE minimization against variance minimization. 47

3.2.4. Leading M3 combinations against M2 combinations 50

3.2.5. Truncation of partial products against truncation of inputs 51

3.2.6. DRUM 52

3.2.7. Low-power approximate MAC. 54

3.2.8. Approximate Booth multiplier against truncated Booth multiplier. 55

3.2.9. Truncation of partial products against other approximate techniques. 57

3.2.10. Effectiveness of using 2x2 multipliers. 61

3.2.11. Additional experiments for the 16x16 case. 64

3.3. Comparison conclusions. 66

4. Least-squares approximation applied to radio astronomy calibration. 67
4.1. Radio astronomy calibration. 67
4.2. Floating-point to fixed-point conversion. 69
4.3. Approximation of the Stefcal algorithm. 73
4.3.1. Truncation of partial products 75
4.3.2. Truncation of inputs 78
4.3.3. DRUM. 81
4.3.4. OR-compression. 82
4.4. Overview. 86
Conclusions. 87
Future work. **88**
References. 89
Appendix A. Approximate MAC comparison plots. **92**

Abstract.

Approximate computing allows to reduce power, area or increase the speed of a circuit by simplifying its logic. This simplification introduces errors in computations. Some applications can tolerate a certain degree of inaccuracy and the correct computations are not necessary to produce acceptable results. In this work, approximate computing methods are applied to the least squares problem. In terms of hardware required, the least squares computation consists of a multiplier-accumulator (MAC), squarer-accumulator (SAC) and divider. As the SAC is a special simplified version of the MAC, and the division is typically done only once at the end of the computation, the MAC unit approximation is the most important to analyze. Several approximate techniques are applied to the MAC and their effectiveness is compared. The results of this MAC analysis are used to approximate the more complex least squares unit. As a case study where the approximation of the least squares is applied, radio-astronomy calibration is used. The calibration is performed by iteratively solving the least squares problem to estimate complex antenna gains. This algorithm does not require the same precision during its computation, which allows to map some number of initial iterations to a lower-precision (or approximate) hardware in order to decrease the energy consumption.

Introduction.

Approximate computing is an emerging paradigm for the area, power and delay reduction. The reduction of the power consumption is one of the main challenges in computing today, both for high-performance computing and for embedded systems. The core idea behind approximate computing is to simplify the logic of the circuit to achieve savings at the cost of accuracy reduction. Some applications are particularly tolerant to errors. These include audio and video applications as human end users are not able to notice small deviations, machine learning and artificial intelligence as these algorithms often can deal with erroneous data, digital signal processing as the real-world inputs are noisy. Many of these applications are resource-/power-hungry, and their error resilience can be exploited by using approximations to save energy, area, and increase performance at the cost of acceptable quality degradation.

Adders and multipliers are the main building blocks of computing units. In the recent years, many novel approximate adders and multipliers have been proposed in the literature. All of them introduce certain simplifications to circuits in order to decrease area, power, or critical path. The optimization of power and area is not something new, however. For example, when a floating-point algorithm needs to be mapped on hardware, it is desirable to use the fixed-point hardware as the logic circuits of fixed-point hardware are much simpler and the power consumption is smaller compared to those of floating-point hardware. In the process of mapping to fixed-point hardware, the main challenge is to optimize the bit widths of signals such that the total cost of the required hardware is minimized and at the same time the performance is satisfied. In this process the signals are truncated or rounded, introducing errors in the computation. In this case the arithmetic units stay accurate, but they operate on smaller signals. Also, to avoid the bit growth, truncated multipliers are often used in digital signal processing applications. The logic of these multipliers either stays accurate but the result is truncated, or the internal logic is simplified as well. These traditional methods can be thought of as belonging to the approximate computing domain as well, as they also enable a tradeoff between the accuracy and cost. In that sense, it is not clear what is the advantage of the new approximate computing methods in comparison to the traditional fixed-point truncation methods, as these new architectures are typically not compared with the truncation methods.

One of the fields where the reduction of power is a critical challenge is the radio astronomy signal processing. An example is the upcoming Square Kilometer Array (SKA) which is going to have enormous power requirements. This array will have a large number of antennas, and each antenna element will receive noise dominated data. Approximate computing methods can be helpful in reducing the power consumption of the required computing units. One of the computations which can be approximated is the calibration of antennas, which can be performed by iteratively solving linear least squares problems. In this work, the possibility of applying hardware approximate computing techniques to the least squares problem is investigated.

The research questions are formulated as follows.

- Comparison of various approximate computing techniques present in literature to investigate how the novel approximate methods perform compared to the traditional truncation methods, and evaluation of their applicability to the least squares computation.
- Application of approximations to the radio astronomy calibration algorithm to determine how much energy can be saved by using different approximate computing methods.

The work is divided into the following chapters. Chapter 1 is a literature review of the approximate computing field. Chapter 2 introduces the selected approximation methods and explains their operation. Chapter 3 presents a comparison of several approximate computing techniques applied to the

multiplier-accumulator unit. Chapter 4 describes the application of the approximate methods to the radio astronomy calibration algorithm.

The main contributions of this work are the comparison of different approximation methods and the application of these methods to a real radio astronomy application. Such comparisons performed in a systematic way are rare in the literature. The application of these methods to a real-world application is also not straightforward. An approach to find an effective approximation configuration for the radio astronomy calibration using a small number of simulations is described in chapter 4. Also, in the process of this work, different scripts were implemented which automate the comparison process, and many models and code generators were built to effectively compare and apply these approximate methods. This code can be useful in the future research.

1. Literature review.

To efficiently apply approximate computing to a specific application, the error resilience of the application needs to be analyzed and an appropriate approximate computing technique should be chosen according to the error analysis. Error resilience analysis allows to identify instructions/kernels/data of a program where approximations could be allowed. Approximations then can be introduced by using software and hardware approximate computing techniques. In this chapter various error resilience analysis methodologies and different approximate computing techniques available in literature are considered.

1.1. Error resilience analysis.

Error resilience of applications can be attributed to several factors [9]:

- Noisy input – inputs from real world (from sensors for example) always contain some noise, and applications processing this data already know how to handle that noise. Errors introduced by approximate techniques can be regarded as additional noise.
- Redundant input data.
- Perceptual limitations – an image processing application can produce an image which contains some errors, but they are hardly noticed by a human user.
- Statistical, self-healing computation patterns – applications may employ computation patterns (such as statistical aggregation and iterative refinement) which intrinsically attenuate or correct errors.
- range of outputs are equivalent – i.e. no unique golden output exists

To effectively apply approximate computing, it is important to analyze error resilience of applications. An application always contains error-tolerant parts and error-sensitive parts. The goal of error resilience analysis is to identify the error-tolerant parts and the amount of approximations which can be applied on them, as well as to get insights into what kind of approximation techniques can be used. On the other hand, error-sensitive parts must be kept accurate as they include pointer arithmetic, conditions, control instructions – all these can lead to crashes and unacceptable results if they are approximated.

The main principle in the error resilience analysis is to inject errors (according to some chosen error model) into different parts of the application and monitor the output to see how the errors affect the result. To quantify the quality degradation, a quality function has to be defined which indicates how far the approximate result is from the exact one. In principle, any function which produces a measure of the difference between accurate and approximate outputs can be used. Examples of such functions include mean square error, mean percentage error, peak signal-to-noise ratio, bit-error rate and so on [16]. The selection of quality metric is based on the analyzed application. The range of errors for which the quality function is satisfied can be regarded as the approximation space of the application.

Over the past few years, many works have been presented to assess the applications for intrinsic error resilience. In this section some of the most relevant papers are reviewed and main ideas and results are described.

Analysis and Characterization of Inherent Application Resilience for Approximate Computing.

Chippa et al. [9] propose a systematic framework for Application Resilience Characterization (ARC). This framework partitions an application into resilient and sensitive parts and characterizes the resilient parts using approximation models that abstract a wide range of approximate computing techniques. The error resilience analysis is divided into three steps:

1) Profiling – distinguish dominant kernels. Kernels which run for more than 1% of the application overall time are chosen for analysis. Other kernels are considered to be not promising for approximations as they constitute only a small fraction of computations.
2) Identify error resilience – random errors are injected to the outputs of the dominant kernels and the overall output is checked against a relaxed quality function. This step is needed to partition kernels into sensitive and resilient. The error model is simple in this case (just random bitflips) and the quality function is relaxed.
3) Characterize error resilience – resilient kernels are analyzed by introducing errors according to the statistical approximation model (SAM) or according to a technique-specific approximation model (TSAM). The result is validated with the actual quality function provided by the user.

As can be seen, steps 2 and 3 are identical. However, step 2 uses a simple error injection model and a relaxed quality function as this step is only needed to identify potentially resilient kernels and drop the ones which are clearly not appropriate for approximations.

The detailed analysis of the chosen kernels is performed in step 3. In that step errors are introduced according to the statistical approximation model (SAM) or a technique-specific approximation model (TSAM). SAM injects errors from a normal distribution based on three parameters: EM (error mean), EP (error predictability) and ER (error rate). This is a high-level approximation model as it does not represent any specific approximation technique, but rather indicates the amount of possible approximations which can be applied to the analyzed computation. For example, such analysis can show that the application is tolerant to an error in some computation, with EM=0, EP=0.1, ER=1, which would mean that the computation output can have deviations of +-10% from its exact result and the application would produce acceptable output (according to the defined quality function). On the other hand, TSAM introduces errors in a more specific way. For example, it can introduce errors based on characteristics of some specific approximate adder according to its bit-error profile. Or it can model effects of bit truncation. On the algorithm level, it can skip some iterations in a loop, which corresponds to a software approximate technique called loop perforation.

Overall, given an application, input data and quality function, ARC produces a list of resilient kernels and the results of applying various approximation models on them (SAM or TSAM). This information can be used to choose which computations to approximate, and what approximate techniques to use. Such error profile also helps to reduce the available design space to choose the best possible quality-cost design option (e.g., which approximate multiplier to use). They apply ARC to 12 widely used applications from the domains of recognition, data mining, and search. They used SAM in their analysis. On average, these applications spent 83% of their run-time in resilient kernels, out of which 74% belong to one dominant kernel. In their work they only apply approximations to this most dominant kernel, so they don't analyze how errors introduced to different kernels can interact with each other.

Improving Error Resilience Analysis Methodology of Iterative Workloads for Approximate Computing.

Gillani et al. [4] improved the ARC framework described above by introducing adaptive statistical approximation model (ASAM). In addition to the original three parameters of SAM, namely error mean (EM), error predictability (EP) and error rate (ER), they use a new parameter, number of approximate iterations (NAI). The model allows to divide an iterative workload into exact and approximate iterations. They apply ASAM to a radio astronomy application and show that the approximation space obtained by ASAM is significantly larger than that of SAM. For this application, they show that the first 23% of iterations can be made approximate with certain EM, EP, ER, while the remaining iterations must be accurate. This allows to better exploit accuracy configurable and heterogeneous architectures, as approximate iterations can be assigned to inexact cores/modes, while sensitive iterations to exact counterparts. They also demonstrate that the original quality function may become inadequate in the error resilience analysis procedure, which requires defining an additional quality function to serve the purpose.

Quality of Service Profiling.

Misailovic et al. [5] introduce a quality of service (QoS) profiler which allows to identify sub-computations that can be replaced with less accurate sub-computations which deliver increased performance with acceptable quality of service loss. The profiler uses loop perforation (which transforms loops to perform fewer iterations than the original loop) to obtain implementations with different performance and quality of service characteristics. Sub-computations that consume a significant amount of computation time, demonstrate tolerance to loop perforation with acceptable QoS loss, and show significant increase in performance can be regarded as the best optimization targets. Such computations can be perforated to increase performance, or they can be replaced by other optimized computations. They argue that optimizable computations often contain loops that perform extra iterations, and that removing iterations, then observing the resulting effect on the quality of service, is an effective way to identify such optimizable sub-computations. To quantify the quality of service, the profiler works with a developer-provided quality of service metric. They apply the profiler to a number of applications and show that loop perforation can increase the performance by a factor of between two or three (the perforated applications run between two and three times faster than the original applications) with quality degradation of less than 10%.

iACT: A Software-Hardware Framework for Understanding the Scope of Approximate Computing.

Mishra et al. [6] present iACT (Intel's Approximate Computing Toolkit). It allows to analyze and study the scope of approximations in applications. The toolkit allows to apply three approximate computing techniques: precision reduction, noisy ALU and memoization. The key idea is when the programmer writes a program, he/she annotates the approximation amenable functions (code segments) with high-level pragmas and also provides a quality function.

pragma_axc – with the *pragma_axc* annotation to a C function declaration, the tool simulates a noisy hardware – noisy arithmetic instructions that operate on floating point values and noisy memory loads and stores. The tool supports several different parameterized noise models – probability based, operand bit-position based, bit-width based, etc.

axc_precision_reduce – downconverts all the floating-point values in the function to 16-bit width precision.

pragma axc_memoize – the tool creates a table with inputs and outputs of an approximated function. If during the execution the input values are within specified range with table inputs, then the result is read from the table skipping the function computation. Otherwise, the function is executed, and input/output values are written into the table.

As an example on how to use this toolkit, they include three different applications and analyze the scope of approximate computing in them. Applying precision reduction to a bodytracking application provides 22% dynamic energy reduction with less than 4% quality degradation. With the approximate memoization scheme applied to a Sobel filter, they obtain dynamic energy savings of up to 22% with 10% quality degradation. Finally, the effect of random bit failures is shown on the accuracy of a classification algorithm. Random bit failures are representative of the timing failures which can happen at low-voltage or high-frequency operation modes. They show that even at high probability of these failures (up to 0.5) the quality degradation is less than 5%.

ASAC: Automatic Sensitivity Analysis for Approximate Computing.

Roy et al. [7] propose ASAC – a framework which automatically identifies approximable data in an application. The main component of this framework is a specialized sensitivity analysis using statistical methods. Variables are systematically perturbed, and the output sensitivity is observed. A hypothesis test generates scores for each variable to quantify the variable's contribution to the output of the program. Based on the scores, variables are classified as approximable or non-approximable. ASAC achieves 86% accuracy when compared to a manual annotation, which shows that this method can be used for large programs where a manual annotation is infeasible.

They evaluate their method by applying it to several benchmark applications. After identifying approximable variables, they apply bit-flip errors (by choosing a random bit among 16 lower bits and toggling it) to these variables and demonstrate that the applications are indeed amenable to the approximations of these variables at the cost of acceptable quality degradation. For example, when bit-flip errors are injected to all the approximable variables in an FFT application, the QoS loss is around 3%. On the other hand, when they apply bit-flip errors to non-approximable variables, the output becomes unacceptable or the applications crash.

PAC: Program Analysis for Approximation-aware Compilation.

Roy et al. [8] (same authors as in the previous method) propose a framework similar to ASAC, but in contrary to ASAC, PAC is a static tool which allows to analyze applications without running them, significantly reducing the time required for the analysis. Another distinction is that the variables are not just classified as approximable or non-approximable but are also assigned with a degree of accuracy (DoA) required to satisfy a quality function. The DoA is a number between 0 and 1, it quantifies the degree of approximation that can be applied to a specific variable. As an example, the DoA number can be translated to the number of bits which can be approximated by a configurable approximate arithmetic circuit such as an adder or a multiplier. The key idea of their method is to propagate the required accuracy of the output (quality of service) to all the program variables. The DoAs are propagated using influence relations among the variables.

In comparison with ASAC, PAC demonstrates a more conservative behavior, i.e. some of the variables identified as approximable by ASAC are classified as non-approximable by PAC. The main advantage of PAC is the runtime overhead of the analysis – PAC is 3 orders of magnitude faster than ASAC. They evaluate PAC in a similar way as ASAC, by injecting bit-flip errors to approximable variables in a number of applications. On average, about one third of the variables in the tested applications are classified as approximable (they assumed that variables with DoA less than 0.5 are approximable and the rest are not). Applying bit-flip errors to these variables causes the QoS loss of 3.4% on average.

Error Resilience Analysis for Systematically Employing Approximate Computing in Convolutional Neural Networks.

Hanif et al. [10] address the question of how to systematically employ approximate computing in Convolution Neural Networks (CNNs). They divide the error resilience analysis into hardware level and software level analysis. There are two possible hardware approximation techniques that can be used for improving the efficiency of CNNs, quantization (floating point operations are transformed to fixed point and the word sizes of the activations and weights are reduced) and approximate hardware components (adders, multipliers, memory units). They argue that in both cases errors are introduced at multiple locations in a network and therefore the error resilience of a network can be simulated by introducing Random Gaussian or White Gaussian Noise (RGN or WGN) at particular locations in a network (because errors from multiple sources, when added together, generate a Gaussian distribution). They introduce WGN individually at the output of convolutional layers and observe the effects on the output accuracy of the network. They apply this analysis to an image classification network and show that the network has a different level of error tolerance for the same error in different convolutional layers. For the software level analysis, they propose a technique which computes the significance of a filter in a layer. Filters with low significance can be pruned at the cost of low quality loss.

Algorithmic-level Approximate Computing Applied to Energy Efficient HEVC Decoding.

Nogues et al. [11] describe a method for applying approximate computing at the level of a complete application. The method decomposes the application into processing blocks and identifies the classes of approximate computing techniques each block may tolerate. They divide approximate computing techniques which can be applied to a signal processing block into processing-oriented and data-oriented techniques. The processing-oriented class consists of techniques altering computations. Examples include computation skipping (a block is skipped permanently or periodically), and computation approximation (complex processing block is replaced by a simpler one with lower accuracy). The data-oriented class includes techniques that modify data characteristics. This class is further divided into dataset reduction techniques which reduce the number of processed data samples (for example, music can be sampled at 32kHz instead of 48kHz), and data format optimization (using fixed-point data, bit-width optimization). In order to determine which kind of technique to apply to each block, they define criteria according to the type of data generated by the signal processing block – control-oriented and signal-oriented data blocks are distinguished. They apply their method to a HEVC decoder and obtain energy reduction of up to 40% with a slight degradation of application quality.

1.2. Approximate computing techniques.

Approximate computing techniques can be broadly divided into software techniques and hardware techniques.

Software techniques:

- computation skipping – some computations are skipped at the cost of acceptable quality loss. Examples include loop perforation, filter pruning, memoization, reducing the number of iterations of an iterative process.
- computation approximation – computations can be replaced by less complex approximate alternatives with lower accuracy. For example, the order of a filter can be reduced, or a Finite Impulse Response (FIR) filter can be replaced by an Infinite Impulse Response (IIR) filter [11].

Hardware techniques:

- circuit pruning – combinational logic can be made simpler by reducing the number of logic gates at the cost of introducing errors. For example, arithmetic elementary circuits such as full adder and 2x2 multiplier can be made smaller at the cost of making some of the entries in their truth tables to be erroneous.
- Data sizing – usage of less accurate data representation to reduce the complexity of arithmetic operations and storage requirements. For example, fixed-point representation with optimal bit-width satisfying quality requirements can be used instead of floating point numbers. Another example is quantization – 32-bit result of multiplying two 16-bit numbers can be truncated or rounded to 16 bits for further processing in the datapath.
- Voltage overscaling – the supply voltage is reduced to the point at which occasional timing errors occur and the circuit starts producing errors. These timing errors affect critical paths which are usually involved in the computation of the most significant bits, which means that voltage overscaling is likely to lead to large errors. This makes it hard to achieve smooth degradation in accuracy as voltage decreases. It can also be hard to predict which errors will be produced at which voltages, as this can depend on a lot of factors such as layout and manufacturing process.

In this section circuit pruning and data sizing are considered and some of the important work done in this area is discussed.

Low-Power Approximate MAC Unit.

Esposito et al. [1] present a low-power approximate multiply-accumulate unit. They keep the accumulate part (adder) accurate and approximate the partial product matrix (PPM) using two methods:

- Approximate counters – if there are two partial products x2y8 and x8y2 in a column, their sum x2y8 + x8y2 can be approximated by the OR-gate: x2y8 OR x8y2. So, two partial products are reduced into a single term. In this case only one input pattern out of 16 produces an error. When x2=x8=y2=y8=1 the correct sum is 2, while the OR-gate produces result 1 in this case. If the height (the column with maximum number of partial products) of the original PPM is N, this approach allows to halve the PPM height making in N/2 by using OR-gates in the columns which have height larger than N.
- PPM columns deletion - selected columns of the partial product terms are not formed to further save energy.

They also compute the error introduced by the column deletion and approximate compression and improve the MAC accuracy by initializing the accumulation register with a compensation term equal to the computed mean error multiplied by the number of multiplications. The proposed MAC units with different configurations show area and power improvements ranging, respectively, from 39% to 69% and 40% to 71%. They also use their MAC in an image filtering application and show significant power reduction with tolerable image quality degradation.

Approximate 1-bit full-adders.

Several papers describe approximate 1-bit full-adders [2] [3] [15]. In these works, the logic of full-adders is simplified at the cost of introducing errors in the truth tables. These adders differ in the number of logic gates they use and in error patterns they introduce (number of erroneous outputs, magnitudes of errors).

Approximate 2x2 multipliers.

Kulkarni et al. [17] present an underdesigned multiplier architecture which consists of approximate elementary 2x2 multiplier blocks that generate partial products. The 2x2 multiplier produces only 1 error when all four inputs are 1 (3x3 is equal to 7). Rehman et al. [2] describe other 2x2 multiplier designs with different structures and various error characteristics (error magnitudes, error probabilities).

Architectural-Space Exploration of Approximate Multipliers.

Rehman et al. [2] propose a methodology to generate and explore the architectural space exploration of large-sized multipliers using the following design parameters:

- different types of elementary approximate 2x2 multiplier modules
- different types of elementary 1-bit full adder modules for summing the partial products
- selection of bits for approximation – how many LSBs need to be approximated in the adder tree?

The design space grows very fast with the width of the operands. They show that even for the 4x4 multiplier, there are 7500 possible configurations. They propose a depth-first search algorithm which starts using the highest approximation available and moves towards a more accurate solution. It stops at a certain configuration (of adder, multiplier and bit-width) when the provided quality function is satisfied. They apply a subset of design points to a JPEG application and show corresponding power/area/quality results.

Approximate multipliers for MAC.

Verstoep in his bachelor's thesis [12] uses 2x2 multipliers to build an 8x8 multiplier with a near-to-zero mean error in order to improve the overall output quality of the multiply-accumulate unit. As state-of-the-art 2x2 multipliers produce smaller results than the accurate multiplier (erroneous entries in the truth-table are smaller than exact results), he introduces 2x2 multipliers with larger results for the purpose of error balancing. For example, along with using the multiplier with 3x3=7 case, he also

adds to the design space a multiplier with 3x3=11 entry. By using different combinations of 2x2 multipliers it's possible to construct a multiplier with the mean error close to zero. The output of the whole multiply-accumulate unit will also have zero-mean error. Using an exhaustive search algorithm, he finds a configuration with the best quality for a cost constraint, or a configuration with the lowest cost for a quality constraint.

DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications.

Hashemi et al. [18] propose a novel approximate multiplier with a dynamic range selection scheme and an unbiased error distribution. The main idea of their method is to use an exact multiplier but with smaller operand widths. If the operands to multiply have width n, they use a kxk-bit multiplier ($k<n$) and choose the k bits from each operand by detecting the leading 1 in the bit pattern and selecting k bits starting from there. It means that instead of approximating the multiplication process, they approximate the operands while using an exact multiplier. The $2k$-bit result is then shifted to the left by a certain number of bits depending on the positions of the leading ones in the original operands to get a $2n$-bit result, while placing zeros at the least-significant bits. To make the truncation error to have near-zero mean when selecting k bits from the operands, 1 is always placed at the least-significant bit of the newly formed k-bit operand. This allows the multiplier to be unbiased and have a near-zero average error. As a result, when using this multiplier in real applications involving numerous multiplications some errors potentially cancel each other rather than accumulate in the final result of the computation.

They compare their multiplier with other two approximate multipliers including [17] in stand-alone manner as well as in three different applications from the domains of computer vision, image processing and data classification, and demonstrate a better overall power/quality tradeoff of the DRUM.

The Hidden Cost of Functional Approximation Against Careful Data Sizing.

Barrois et al. [14] argue that a fair comparison between the usage of approximate circuits and classical fixed-point arithmetic with bit-width optimization has never been performed. They select a number of existing approximate adders and multipliers and demonstrate that they tend to be dominated by truncated or rounded fixed-point ones. They argue that if the values below a certain bit position can be made approximate, it's usually better to not compute the approximate bits at all while saving energy and reducing the bit-width of the data. The reduced bit-width then also have a positive impact on the other operators in the datapath and the storage of the data. Approximated data, on the other hand, require larger bit-width and contain a greater amount of costly useless information. Accuracy reduction is obtained by simplifying the operator structure but not by reducing the operator output bit-width. This reduces the energy of the considered operator but does not have a positive impact on the other operators, as for fixed-point.

2. Multiply-accumulate unit and its approximation.

2.1. Least Squares Problem

Least squares is a method of finding approximate solutions of over-determined systems of linear equations [31]. Over-determined systems are systems with more equations than unknowns. Consider a system of linear equations $Ax = b$ with an mxn matrix A $(m > n$ as it's overdetermined$)$, n-vector x, and m-vector b. Such a system has a solution only if b is a linear combination of the columns of A. If a solution does not exist, i.e., there is no such x that satisfies $Ax = b$, it is possible to find a vector x that minimizes the error vector $r = Ax - b$ which is called the residual vector. In this case x almost satisfies the linear equations and $Ax \approx b$. Minimizing the residual vector means making its length as small as possible. The length of a vector is given by its norm which is equal to the square root of the sum of the squares of its elements:

$$\|r\| = \|Ax - b\| = \sqrt{r_1^2 + r_2^2 + \cdots + r_m^2} \tag{1}$$

The problem of minimizing the norm $\|r\|$ is the same as the problem of minimizing the square of the norm $\|r\|^2 = r_1^2 + r_2^2 + \cdots + r_m^2$. The least squares problem therefore can be formulated as follows:

$$\min_x \|Ax - b\|^2 \tag{2}$$

If the columns of A are linearly independent, the solution of the least squares problem is given by:

$$x = (A^T A)^{-1} A^T b \tag{3}$$

If A is a vector instead of a matrix and denoted as a, the least squares problem is reduced to finding a scaling x which makes ax as close as possible to b. The solution x is computed by dividing the dot product of a and b by the dot-product of a with itself:

$$x = \frac{a^T b}{a^T a} \tag{4}$$

This problem can also be formulated as the problem of finding the projection of b onto a. An example for two-dimensional vectors is shown in Fig. 1. The goal is to find a scaling factor x such that ax is a projection of b onto vector a. In this case vector ax has the smallest possible distance from b among all vectors with the same direction as a as $r = ax - b$ is perpendicular to the direction of vector a.

If $a = (1, 2)$ and $b = (8, 3)$, the scaling is computed as follows: $x = \frac{a^T b}{a^T a} = \frac{1 \cdot 8 + 2 \cdot 3}{1^2 + 2^2} = 2.8$.

Such a computation can be mapped onto a circuit which consists of three hardware units: a multiplier-accumulator (MAC), a squarer-accumulator (SAC) and a divider.

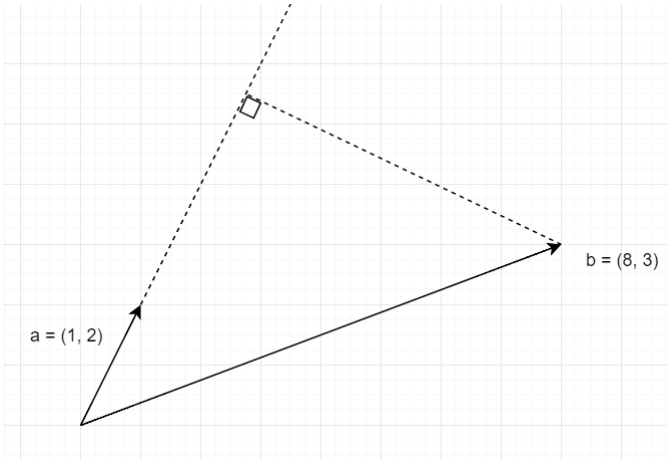


Figure 1. Least squares problem in a simple case with two vectors.

2.2. Multiplier-accumulator (MAC)

MAC and SAC are the most computation-intensive units of the least squares computation. The vectors in applications can be very large, consisting of hundreds or thousands of elements and the computation of the two dot-products is the dominant part of the least squares algorithm. Division is done only once at the end of the computation when the numerator and denominator are known. For this reason, in this work the division unit is assumed to be accurate as approximating it would produce a large error for a minimal reduction in cost. By cost, the area, power or latency of a circuit is meant. As the SAC is a special case of the MAC with simplified logic due to the equal operands, this section describes a more general MAC unit, but the discussion is relevant to the SAC as well.

A multiplier-accumulator consists of a multiplier, an adder, and registers which store an intermediate result. All three can be made approximate by using approximate multipliers, adders, or approximate storage techniques. However, the multiplier part typically has the largest cost in terms of hardware and power required, which makes it the first target for approximation attempts and the focus of this work. An example of an accurate 8x8 MAC unit in dot notation (similar to [20]) can be seen in Fig. 4. The operation of a MAC unit can be divided into five stages:

1. Partial products generation – in Fig. 3 the partial products are generated by 64 AND gates. Each row of the partial product matrix represents the multiplicand value multiplied by 0 or 1 depending on the multiplier bit. This value is also shifted such that the least significant bit (LSB) has the same position as the multiplier bit. In this case each row is either 0 or $1a$ and there are n rows for an nxn multiplier. If the modified Booth algorithm is used, the number of rows is half as much, $n/2$, as each row can represent 0, $\pm 1a$, $\pm 2a$. In case of a squarer such matrix is significantly simplified as some of the AND gates have equal inputs, allowing to significantly reduce the number of rows [20].
2. Partial products reduction – the partial products have to be summed. To speed up this process and avoid long carry propagation, the matrix is typically reduced to a height of two by using carry-save addition. The reduction is done by using full adders and half adders. In Fig. 4 full adders are represented by long rectangles, and half adders by small rectangles. Fig. 2 demonstrates the operation of the full adders and half adders. The full adder applied to three dots in a column allows to compress these dots to one dot in this column and one dot in the next column. Similarly, the half adder compresses two dots in one column into one dot in the current column and one dot in the next column. This process leads to a gradual reduction of the height of the partial products matrix to the height of two. There are two main methods to

reduce the matrix: Wallace tree and Dadda tree. In the Wallace method, the partial products are reduced as soon as possible, and in the Dadda tree the reduction is done by using the smallest possible number of adders to reduce the height to the next optimal value (for an 8x8 case, these values are 6-4-3-2). The Dadda tree is slightly faster and requires fewer gates [21]. In Fig. 4 the Dadda method is used. The height is reduced from 8 to 2 in four stages: 8-6-4-3-2. The Dadda method is also used to estimate the cost of various designs later in the work.

3. Addition after reduction – the resulting two operands are then added by a carry-propagate adder to produce the final multiplication result. It can be a simple ripple-carry adder in which case the addition after the reduction can be seen as continuation of the partial products reduction as it is done by using half adders and full adders to reduce the height from two to one. But this addition can also be done by using a faster adder, for example the carry-lookahead adder.
4. Accumulation – the multiplication result is extended and added to the running sum which is stored in the accumulator registers. The accumulator has a larger bit width as it has to store the sum of a large number of multiplications. The size is chosen depending on the number of elements in the vectors and the distributions of their values to avoid an overflow.
5. Storing the result – the multiply-accumulate value is stored into registers and the process is repeated for the next pair of inputs.

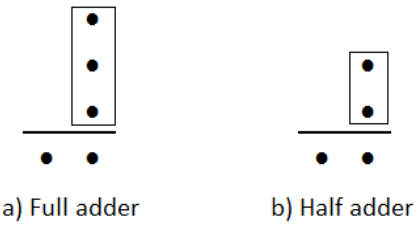


Figure 2. Operation of full adders and half adders.

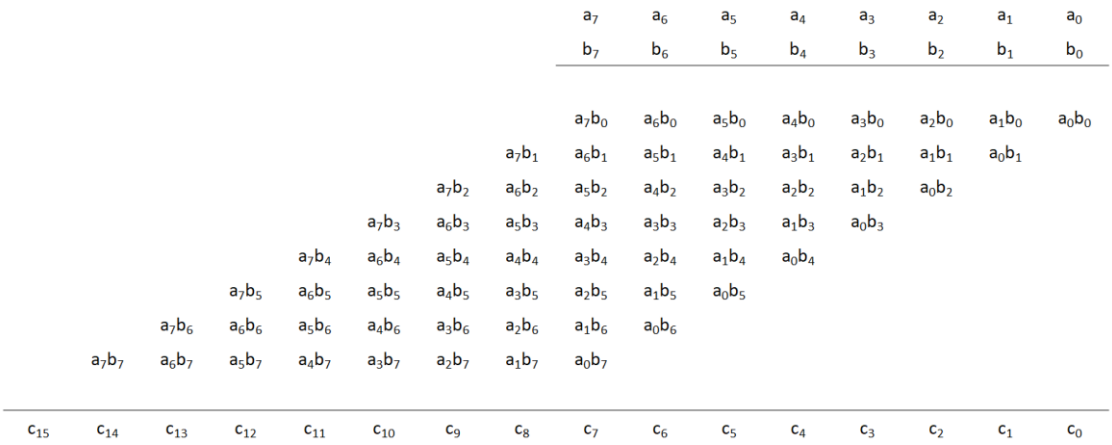


Figure 3. Generation of 64 partial products in an 8x8 multiplier.

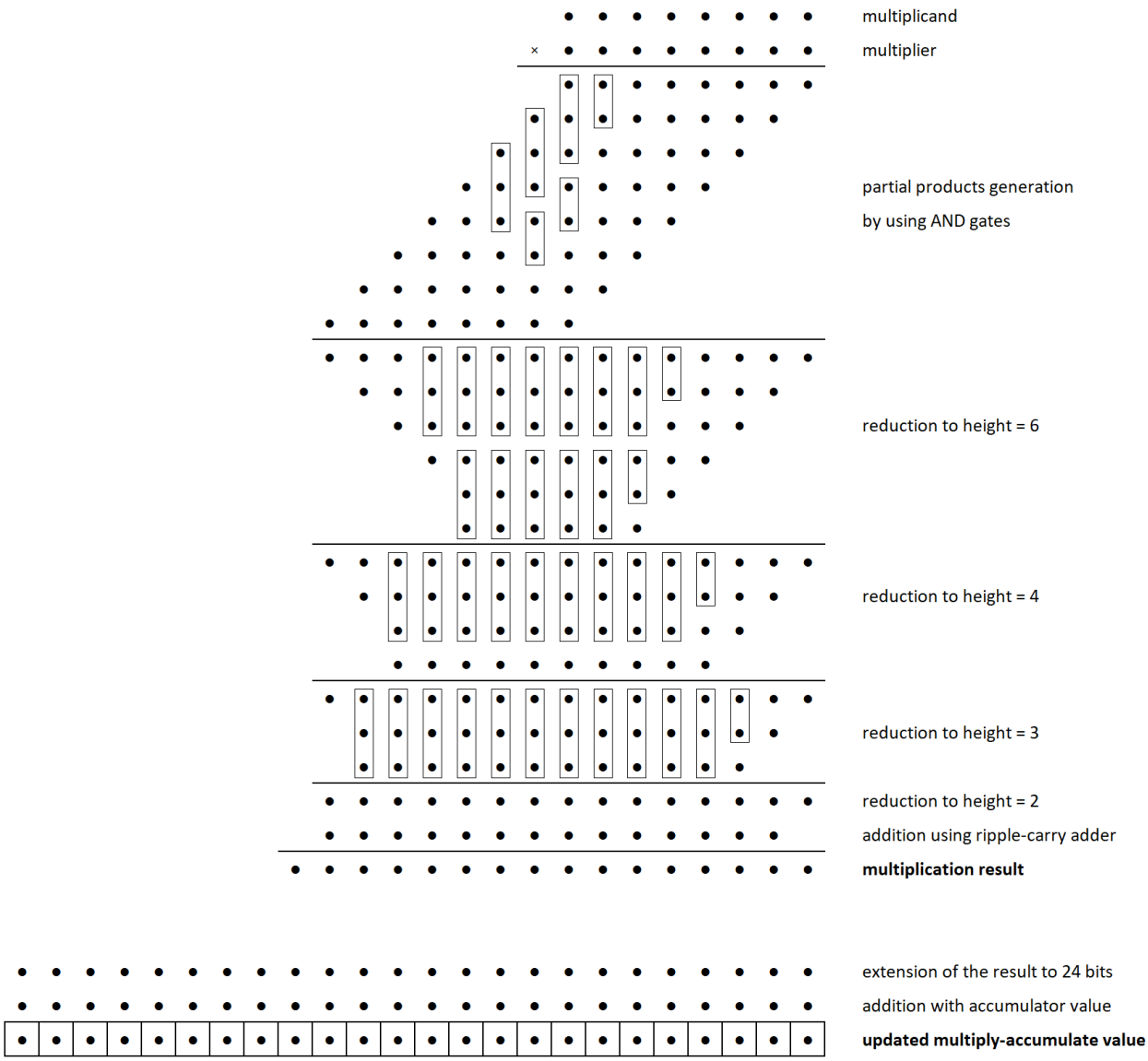


Figure 4. Dot diagram of an 8x8 MAC unit.

2.3. Unit gate model

In this work a simple unit gate model [22] is used to estimate the area of various designs and make comparisons. In this model the NOT gate is equal to 0.5 gates, two-input gates AND, OR, NAND, NOR are assigned a value of 1, and the XOR gate has a cost of 2 gates. Areas of some other components according to this model are shown in Table 1.

Component	Area in gates
NOT	0.5
AND, OR, NAND, NOR	1
XOR	2
Full adder	7
Half adder	3
Register (master–slave D flip-flop)	9

Table 1. Unit gate model.

Table 2 shows the results of applying this simple model to the MAC unit. As can be seen, the multiplier part (the first three stages) has more area than the accumulation and registers combined. However, these values depend on the architectures of the components (in this case, for example, the adder is assumed to be a ripple-carry adder). Also, the sizes of the accumulation stage and registers can be made smaller or larger depending on the overflow behavior.

MAC stage	Area in gates	Percent of total area
Partial products generation	64	8%
Partial products reduction	266	33%
Addition after reduction	94	12%
Accumulation	164	20%
Storing the result	216	27%

Table 2. Area of MAC computed using simple gate model.

2.4. Errors and error metrics.

Errors introduced in the multiplier part of a MAC unit lead to an error at the output of the multiplier-accumulator. As the errors are accumulated in the accumulation stage, the error profile of a MAC unit can be very different from the error profile of the multiplier. As an example, Fig. 5 shows an error distribution of an approximate recursive multiplier consisting of M1-type 2x2 multipliers (the details of this approximation technique will be explained later). The histogram is computed by iterating through all possible 65536 input combinations of two 8-bit operands and taking the difference between the approximate result and the accurate computation. Fig. 6 shows the errors produced by the MAC unit which uses this approximate multiplier to compute the dot-product of uniformly distributed random vectors with 500 elements. The simulation is run 100000 times and the histogram is computed by taking the differences between the accurate output and the result of the approximate MAC unit. In this case the introduced errors can be seen as independent (as each multiplication has independent inputs) identically distributed (error distribution of each multiplication is the same as in Fig. 5) random variables and the sum of a large number of such errors tends towards a normal distribution according to the central limit theorem, even though each of these errors individually are not normally distributed [23]. The red curve in Fig. 6 shows a normal distribution with the mean and standard deviation computed from the MAC error distribution. It shows that the errors of the MAC unit closely follow the normal distribution. In this case, however, each multiplication has the same input distribution (uniform) and each multiplication has operands which are uncorrelated between each other. Distributions of real applications can lead to different shapes as such distributions can lead to the violation of the central limit theorem's assumption of independent identically distributed random variables.

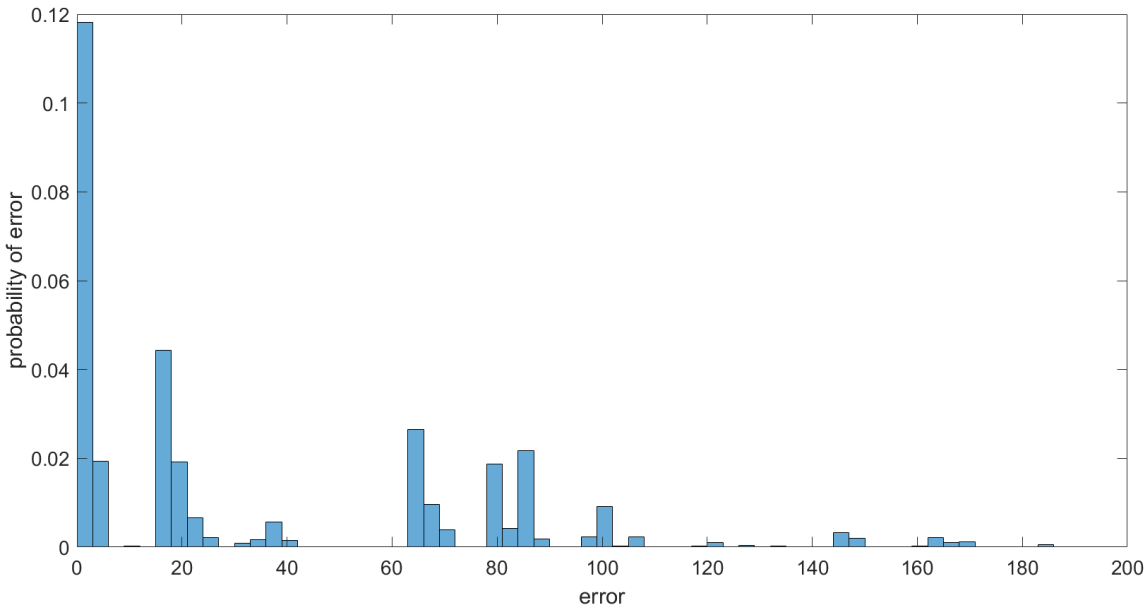


Figure 5. Error distribution of an 8x8 approximate multiplier.

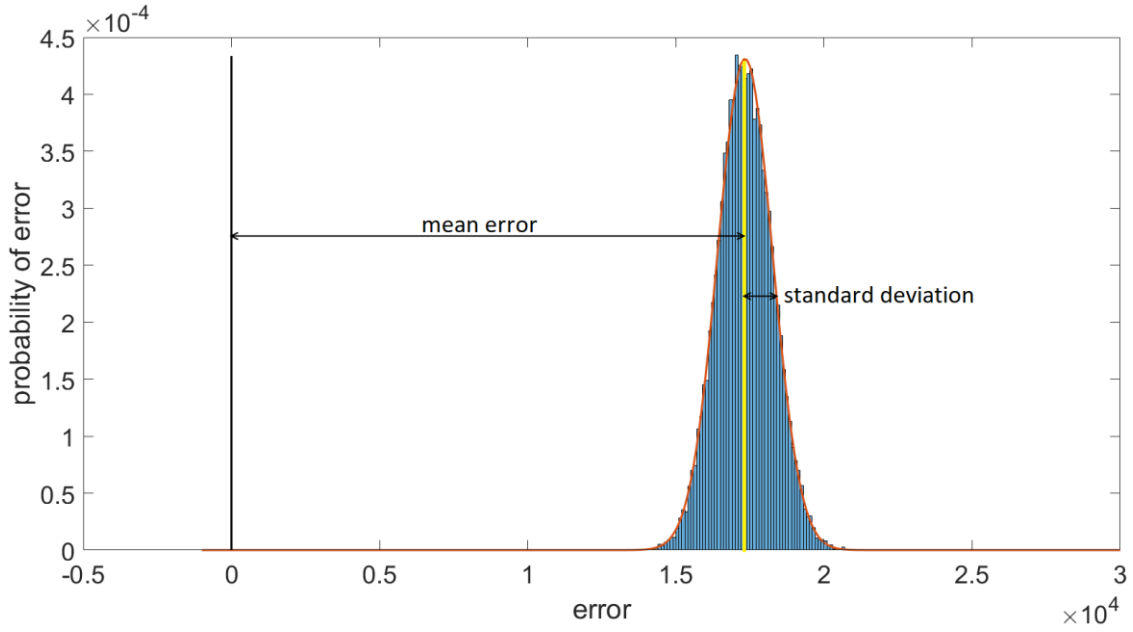


Figure 6. Error distribution of a MAC with an approximate 8x8 multiplier.

Figure 7 shows the effect of increasing the size of vectors. If the vectors have only one element, the MAC error distribution is the same as the multiplier error distribution. As the size grows, the distribution shape becomes more and more bell-shaped.

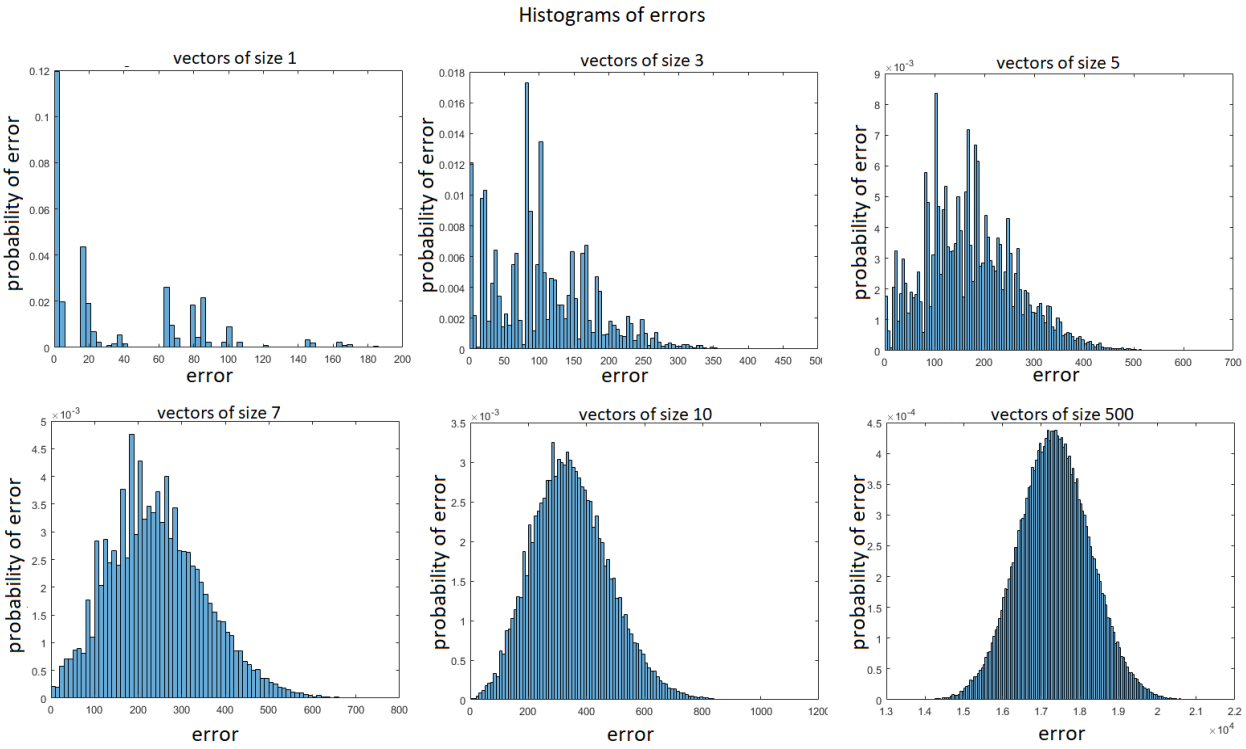


Figure 7. MAC error distribution as the size of vectors increases.

To quantify the error distribution at the output of the MAC the position of the distribution and the width of the distribution must be considered. The position can be described by the distance from 0 to the center of the distribution, which is equal to the mean error (ME), i.e., sum of errors divided by the number of outputs.

$$ME_{mac} = \frac{1}{K}\sum_{i=1}^{K} e_i \tag{5}$$

where K is the number of MAC results ($K = 100000$ in Fig. 5) and e_i is the error of an individual MAC result. The simplest way to describe the width is to measure the distances from the mean to the farthest errors to the left and right of the mean, the maximum deviation. However, this measure depends only on the extreme values and does not reflect the fact that typically these extreme errors are rare and most of the errors are gathered around the mean value. There are many error combinations which lead to the mean error and there are only few combinations producing large errors. A more reliable measure of the width of a distribution that depends on all errors is the standard deviation σ:

$$\sigma_{mac} = \sqrt{\frac{1}{K}\sum_{i=1}^{K}(e_i - ME_{mac})^2} \tag{6}$$

Sometimes it is more convenient to use the square of the standard deviation, the variance, as a measure of the distribution width:

$$Var_{mac} = \frac{1}{K}\sum_{i=1}^{K}(e_i - ME_{mac})^2 \tag{7}$$

The mean error or variance alone are not sufficient to describe errors. If only the mean error is used, there is no information about the spread of the errors from the average error. It may be that the mean error is equal to 0, but the spread is large, and the error becomes significant. It means that it's not possible to conclude that one approximate design is better than another just by comparing the mean errors. Similarly, if the variance is close to 0, but the distribution is far from 0, i.e., the mean error is large, the total error will also be significant. Therefore, the variance alone cannot be used to compare approximate MAC designs either.

Error metrics exist which combine the position and width of an error distribution into a single value [23]. An example is the widely used mean squared error (MSE).

$$MSE_{mac} = \frac{1}{K}\sum_{i=1}^{K}(e_i)^2 = ME_{mac}^2 + Var_{mac} \tag{8}$$

It incorporates both the mean error and the variance and thus can be used to compare approximate MAC designs. However, it is not possible to compute ME and variance if only the MSE number is given. This fact makes it hard to use the MSE metric to understand the origin of the errors, i.e., are the errors coming from a large mean error or a large variance? The answer to this question determines error mitigation possibilities as will be described in the next section.

The mean absolute error (MAE) is similar to MSE [23]. It also combines the position and width of an error distribution and can be used for comparisons. The difference is that in the MSE metric larger errors have more weight as they are squared.

$$MAE_{mac} = \frac{1}{K}\sum_{i=1}^{K}|e_i| \tag{9}$$

Fig. 6 contains information about the magnitudes and signs of errors between accurate and approximate MAC results. There is no information about the relative errors. For example, if there are two accurate outputs of MAC which are equal to 5 and 105, and two corresponding approximate results 10 and 110, in both cases these errors will be plotted as -5 on an error histogram similar to Fig 6. However, in the first case there is a percentage error of $100 \cdot (10-5)/5 = 100\%$, while in the second case it's only 4.76%. To quantify this relative error at the output of the MAC the mean absolute percentage error (MAPE) can be used [23]. The main disadvantage of this metric is that if the exact result is zero, it's undefined, which makes it difficult to use this metric for signed numbers.

$$MAPE_{mac} = \frac{100}{K}\sum_{i=1}^{K}\left|\frac{e_i}{exact\ result}\right| \tag{10}$$

Overall, one-value metrics like the MSE, MAE and MAPE of the MAC output can be used to quantify the error introduced by an approximate design and compare different designs. The mean error and variance together can be used to better understand the nature of the errors.

2.5. Mitigation of errors.

The concepts of the mean error and variance described in the previous section can be used to understand what can be done to reduce the errors at the output of a MAC unit. If an exact multiplier is used, the bias is zero and variance is also zero, which corresponds to the straight vertical line at zero in Fig. 6. If an approximate multiplier is used, a certain bias and variance are introduced at the output of the MAC which depend on the input distribution, error profile of the multiplier and the size of the vectors.

2.5.1. Mean error balancing.

The first approach is to make the mean error of the approximate multiplier to be zero for a given distribution. If the expected error of the multiplier is zero, the expected error of the MAC will also be zero as the expected value of a sum is equal to the sum of expected values of individual variables (multiplier errors in this case).

$$ME_{mac} = N \cdot ME_{mult} \tag{11}$$

where N is the number of multiplications needed to compute the dot-product, i.e., the vector size. This approach can only be used by certain approximate computing techniques which allow such mean error balancing. For example, this method can be used in recursive multipliers which consist of smaller 2x2 elementary multipliers (described in a later section). By using a certain combination of 2x2 multipliers with different positive and negative errors the mean error can be made smaller. This method is considered in [12]. The main advantage is that in theory the vector size is not important because if the mean error is 0, addition of any number of multiplier results will produce the expected mean error of zero at the output of the MAC unit. In practice, however, it is often not possible to find a combination with mean error exactly equal to zero as there are a limited number of possible approximate 2x2 designs and their combinations. Therefore, the mean error will be non-zero and the bias at the MAC output will be farther and farther away from zero as the length of the vectors grows.

Another problem is that this approach ignores the variance. Low mean error does not guarantee anything about the error behavior at the MAC output, it only says that the distribution is centered at zero, but the width of it is unknown. It is therefore impossible to predict the MSE or other error metrics which take into account both the width and bias of the distribution. The MSE of a MAC can be computed using the mean error and variance of the multiplier as follows:

$$MSE_{mac} = ME_{mac}^2 + Var_{mac} = (N \cdot ME_{mult})^2 + N \cdot Var_{mult} \tag{12}$$

Therefore, to minimize the MSE of a MAC, both the mean error and variance of the multiplier must be minimized. Mean error balancing, however, introduces a trade-off between the mean error and variance. It might be that the mean error is minimized at the expense of increased variance. For example, a design with mean error zero can use more approximate 2x2 blocks than a design with non-zero mean error, which means that the variance of the zero mean error design is larger and that can lead to larger errors at the output of the MAC unit. Also, this approach is hardware oriented and if the input distribution is changed, the hardware must be changed as well to remove the bias.

2.5.2. Using two multipliers with opposite errors.

If half of the multiplications in the dot-product are done by a multiplier with a positive bias $+\delta$ and another half with the opposite negative bias $-\delta$, then the resulting mean error will be zero assuming that the input distributions of these multipliers are the same. The variance will be the same as if only

one multiplier is used for all multiplications, as the error profile of both multipliers is the same except the signs. This approach is described in [19]. The main advantage of this method is that the mean error is zero independent of the size of the vectors. Also, if the distribution is changed, the mean error will stay at zero as the mean error of one multiplier will be balanced by the opposite multiplier. The main disadvantage is that two multipliers are required, making this method inapplicable to area optimization. Also, it can be hard to design a multiplier with an error profile which is opposite of a given multiplier and it can be less efficient in terms of error/cost tradeoff. In [19] the mirror multiplier with $+\delta$ bias always has larger area and power compared to the multiplier with $-\delta$.

2.5.3. Accumulator initialization.

Another approach is to place a compensation initial value in the accumulator register, i.e., instead of starting with 0 inside the accumulator, the accumulation register is initialized with some predefined compensation value. This value is equal to the mean error of the multiplier, multiplied by the size of the vectors. The position of the distribution can be controlled by this initial offset and shifted to zero. This method can be used with any approximate multiplier as the unbiasing is done outside of the multiplier. Another advantage of this method is that the unbiasing value can be adjusted according to the current distribution and therefore it is not needed to change the hardware if the distribution is changed. The main disadvantage is that the size of the vectors must be known to compute the required compensation term. However, in most applications this size is known before the computation. Also, this compensation term is a constant which is added to all the results independent of the real values, which can lead to large errors in certain applications, especially if the distribution is not stable and changes frequently.

This method allows to optimize the MSE of the MAC unit, as the variance and the mean error in Eq. 12 can be minimized independently and therefore there is no tradeoff between the two. The multiplier part in this method is optimized to have the smallest possible variance without considering the mean error of the multiplier, as this mean error will be compensated in the accumulator. The mean error of the MAC will be zero after unbiasing. The variance will be equal to the variance of the multiplier times the number of multiplications if errors of individual multiplications are uncorrelated, as the variance of a sum of independent random variables is equal to the sum of the variances of these variables. If they are correlated, then the covariance between each pair of multiplications has to be computed and added to this sum. However, the sum of individual variances is a good approximation to the total MAC variance if this correlation is not large.

$$Var_{mac} = N \cdot Var_{mult} \tag{13}$$

This method can also be described by the following analogy. If there is a measurement instrument with a certain systematic error which is known, the instrument can still be used by compensating the measured value. It can also be calibrated to remove this systematic error. In the error analysis terminology, the mean error is called accuracy and the variance of an instrument is called precision. The measurements will be more accurate on average in this case. The precision of the instrument, however, will be the same as it is inherent in the design, quality and the working principle of the instrument. In this analogy, the multiplier can be seen as an instrument, and the accumulator initialization as a calibration/adjustment mechanism.

2.6. Approximate units against careful data sizing.

The work of [14] have compared a number of approximate circuits with careful data sizing. By careful data sizing they mean the choosing of the right bit-widths for the signals. The results suggest that the benefits of using approximate functional units seem to be minimal or even nonexistent for the considered set of real-world applications. One of the researchers in the field even claims that it is a closed problem in the approximate computing research and we should stop designing approximate circuits because exact units with a narrower bit width are at least as good as approximate ones [24].

Choosing a smaller bit-width for a functional unit introduces an error due to truncation, but it has a positive impact on the functional units which use the output as they also can be made smaller. Approximate functional units, on the other hand, have the same number of output bits as their accurate counterparts. The hardware cost of the particular unit can be decreased by introducing errors and simplifying its logic, but the subsequent units which are using the computed approximate result still have to operate on the same bit-widths.

It seems that if the datapath after the approximated unit is long enough, it is always better to use truncation (smaller bit-widths). To illustrate this, suppose that a multiplier is approximated. If an approximation technique is used which does not reduce the number of output bits, the cost is saved only in the multiplier itself but not in the subsequent circuits as the output bit-width is not changed. The error is propagated, and the exact units spend energy and area processing erroneous data. On the other hand, if a multiplier is approximated in such a way that some of its least-significant bits are always zero, it may save less cost in the multiplier, but these zeros are propagated to the whole datapath, including registers, adders, other multipliers, multiplexers and so on. If this datapath is long enough (an IIR/FIR filter, for example), the benefits will outweigh the savings in just one multiplier. Errors from the truncated unit are also propagated to the datapath, but the cost is saved as the subsequent units are also truncated, so the errors are propagated in the form of truncation.

Some of the units after an approximated unit can also be made approximate by introducing more errors on top of the already propagated errors, but not all circuits are easily approximable (consider registers or multiplexers as an example) and the nets and buses between the units cannot be simplified. Also, the error profile in such a multi-level approximation scheme can be more difficult to analyze and predict than truncation. Another possibility is to truncate the output of an approximate unit. In this case two types of approximation are used – the approximate result with an introduced error has the same number of bits as the accurate one, but then only some of the computed bits are used by the subsequent units.

These issues seem to be the main limitation of approximate functional units and it is not considered in many papers proposing such units. Typically, approximate adders and multipliers are evaluated in isolation and there is no comparison with the simplest approximation – truncation. Such units are compared with their accurate counterparts and the achieved cost/error trade-off is reported. However, it may be that a similar or even better tradeoff can be achieved simply by using truncation techniques.

The MAC (SAC) unit, however, is a simple circuit and using an approximate multiplier may lead to a better trade-off compared to truncation, as truncation in this case has a relatively small effect on the short part after the multiplier. As the MAC unit is an essential part of the least squares algorithm, an evaluation of the applicability of approximate techniques to the multiplier in the form of a comparison with truncation techniques can be helpful. In the rest of the chapter an attempt is made to perform a comparison between a number of approximate multiplier techniques and truncation methods.

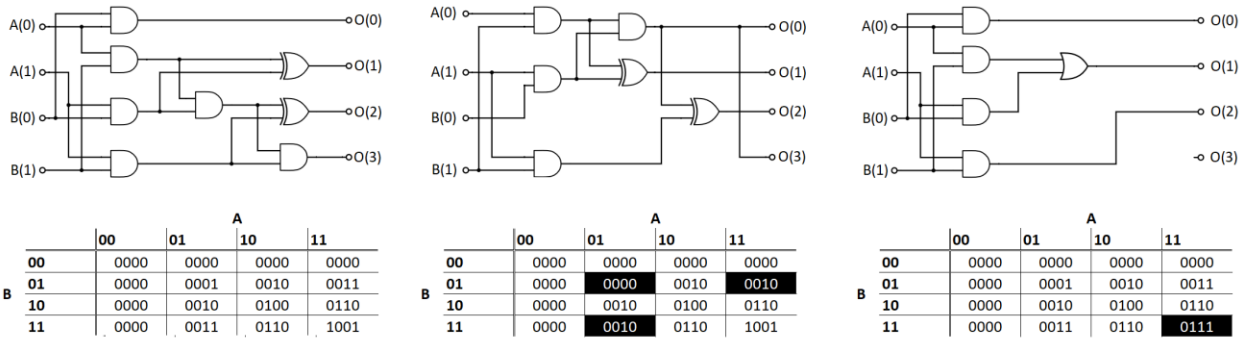
2.7. Approximate multipliers for MAC.

In this section several techniques for approximating a multiplier are described. The selected techniques have different approximation principles and their comparison with truncation should help to understand their effectiveness compared to truncation methods.

2.7.1. Recursive approximate multipliers

Recursive multipliers are multipliers consisting of 2x2 elementary multipliers. It is a hard task to design an arbitrary nxn approximate multiplier, but a 2x2 multiplier is a simple circuit which can be made approximate by removing some gates and modifying its truth table. These small multipliers can be used as building blocks for larger multipliers. Several approximate 2x2 designs are available in the literature with various error and cost properties. A combination of accurate and different approximate 2x2 blocks can be used to construct approximate multipliers with certain error properties based on a given input distribution.

Five 2x2 designs are considered, one accurate and four approximate (Fig. 8). Approximate designs are smaller than the accurate one in terms of cost, but they introduce various errors which can be seen in the corresponding truth tables. An example of an 8x8 multiplier consisting of sixteen 2x2 multipliers can be seen in Fig. 9. The mean errors and variances (assuming uniform distribution) of the multipliers are shown in Table 3. As the errors are of different magnitudes, signs and frequencies, these elementary 2x2 blocks can be combined to achieve the best cost-error trade-off for a given input distribution. Different ways of combining these 2x2 blocks are considered next.



B \ A	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	1001

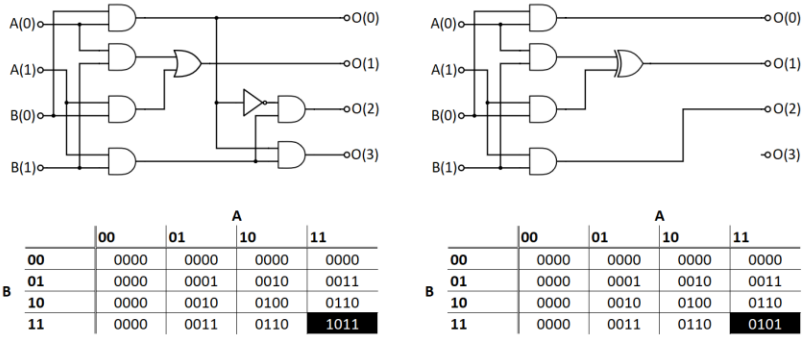
a) M0 - accurate 2x2 multiplier

B \ A	00	01	10	11
00	0000	0000	0000	0000
01	0000	**0000**	0010	**0010**
10	0000	0010	0100	0110
11	0000	**0010**	0110	1001

b) M1 - approximate 2x2 multiplier with 3 errors

B \ A	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	**0111**

c) M2 - approximate multiplier with 3x3=7



B \ A	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	**1011**

d) M3 - approximate multiplier with 3x3=11

B \ A	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	**0101**

e) M4 - approximate multiplier with 3x3=5

Figure 8. 2x2 multipliers.

2x2 type	Mean error	Variance
M1	-0.1875	0.15234375
M2	-0.125	0.234375
M3	0.125	0.234375
M4	-0.25	0.9375

Table 3. Mean error and variance of 2x2 multipliers assuming uniform distribution.

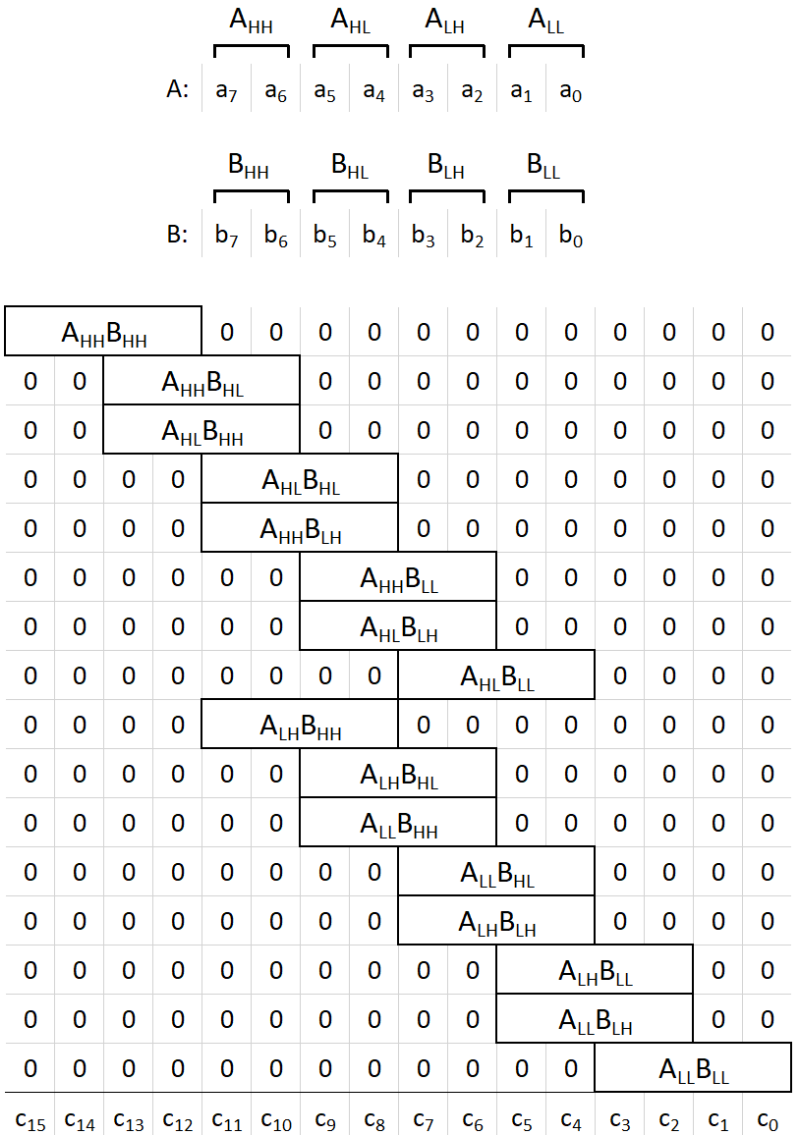


Figure 9. 8x8 multiplier consisting of 2x2 multipliers [12].

Low mean-error combinations:

In this approach the 2x2 blocks are combined in such a way that the mean error of the multiplier is as close as possible to zero for a given distribution and cost. This is possible by using 2x2 multipliers with different error signs. As an example, the M2 type produces a negative error -2 ($3x3 = 7$) while M3 has a positive error for the same 3x3 entry. The main disadvantage of this method is that the variance is ignored, and the error of a design combination is assumed to be equal to its mean error. However, according to eq. (12), small mean error will not necessarily lead to a small mean squared error.

Another disadvantage is that for finding the best design for a given cost (generation of the trade-off) an exhaustive search over a large design space is required. It is feasible for 4x4 multipliers, as the design space includes only $5^4 = 625$ combinations (the accurate design plus four approximate designs, raised to the power which is equal to the number of required 2x2 blocks, four in this case). But the design space grows exponentially with the multiplier size. For an 8x8 multiplier consisting of 16 2x2 multipliers, there are $5^{16} = 152{,}587{,}890{,}625$ possible combinations. For this reason, exhaustive search is only used for 4x4 multiplier tradeoff generation. For 8x8 case, the design is partitioned into four 4x4 multipliers. For each of these four multipliers a separate tradeoff is generated considering all 625 possibilities, from which only 60 pareto-optimal and close-to-pareto-optimal combinations are taken. This leads to $60^4 = 12{,}960{,}000$ combinations from which the final tradeoff is generated.

The approach for the tradeoff generation for an 8x8 multiplier can be described as follows:

1. Construct multipliers consisting exclusively of a certain type, M2 for example. Measure the power and area of the multiplier and divide it by the number of 2x2 blocks (16 in the 8x8 case). This way a certain weight is assigned for each type. Typically, M0 has the largest weight as it's accurate, and M2 has the smallest weight.
2. For each of the four 4x4 multipliers compute the cost (based on the determined weights) and the error according to an error function (ME, MSE, or variance) and the given input distribution. This way each of the 625 combinations is assigned a cost and an error.
3. Select a number of Pareto-optimal designs from each of the 4x4 tradeoffs, 60 for example. The larger the number, the more optimal the tradeoff, but the complexity of the search is also increased.
4. Merge all the selected combinations and find the best designs.

This approach is used for finding the best designs among the low mean-error combinations, and in each of the combinations considered next as well, except those which consist of only one type where the tradeoff generation is trivial.

Low variance combinations (not including covariance):

The variance of a multiplier can be computed by summing the variances of all 2x2 multipliers and covariances between each pair of the blocks:

$$Var(mul) = Var(mul_1) + Var(mul_2) + \cdots + Var(mul_n) + 2\sum_{i \neq j} Covar(mul_i, mul_j) \qquad (14)$$

The covariance term is not equal to zero as some blocks are correlated, i.e., if there is an error in one of the blocks then the probability of an error in a block with the same input is not independent. As an example, if there is an error in an M2 approximate multiplier $A_{LL}B_{LL}$ in Fig. (9), then both A_{LL} and B_{LL} are equal to 3. Higher order blocks with inputs A_{LL} and B_{LL} are more likely to produce errors. If block $A_{LL}B_{LH}$ is also of type M2, then the error probability is not $1/16$ but 1/4. However, this covariance term is small and can be ignored for the trade-off generation. This allows to significantly reduce the design space exploration as the next more approximate design can be chosen just by making the next higher-order 2x2 block with smallest variance to be of an approximate type. The disadvantage of this approach is that the mean error is ignored, the error of a combination is assumed to be equal to its variance. However, the mean error of a design can be unbiased by the accumulator initialization. In this case the variance can be seen as the MSE after unbiasing, so the error of a combination can be predicted. In the previous approach the MSE of a combination cannot be predicted as there is no

information about the variance in the ME metric. The low variance approach with unbiasing, however, allows to minimize the width of the error distribution and then shift it to zero with unbiasing to minimize errors.

This method only uses combinations with the M1 and M2 multiplier types. The M3 type is not useful for such combinations because it has the same variance as M2, but higher cost, so M2 is always better to use (if covariance is ignored). M4 is always worse than M2 as they both have an error for the same 3x3 case, but the error of M4 is twice as large which means that it always has a larger variance. Moreover, the area is also larger as M4 has an XOR-gate instead of an OR-gate in M2.

Low MSE combinations

These combinations minimize the MSE of the MAC unit by minimizing both the mean error and variance in Eq. 12. For a 4x4 case, the MSE of each combination out of 625 is computed and the pareto-optimal combinations are selected. For an 8x8 case, similar to low mean error combinations, the design is divided into four 4x4 blocks, the best 60 designs are chosen to form a design space of $12{,}960{,}000$ combinations, and then the pareto-optimal designs are included in the tradeoff. Contrary to the low-variance approach, this method introduces a tradeoff between the mean error and variance. Therefore, all 2x2 multiplier designs are present in such combinations. It is expected that these combinations should produce a more effective tradeoff compared to low mean error and low variance combinations if the unbiasing is not performed, but low variance combinations with unbiasing should produce better results compared to low MSE with unbiasing.

M2 combinations

There is a significant difference between the M2 and M4 types and the two other approximate 2x2 multipliers. M2 and M4 produce only 3 bits as output, which means that there are fewer partial products to add in the adder tree (Fig. 10). Therefore, these 2x2 blocks save cost at the partial product generation stage and also reduce the adder tree cost. M1 and M3, on the other hand, have 4 bits at the output, the same as the accurate type M0. When using M1 and M3, savings are made only in the partial product generation step. This issue is not considered in the papers which propose M1 [2] and M3 [19]. The partial product generation cost increases quadratically with the multiplier size, but the adder tree cost presumably grows faster. Table 4 shows the costs of partial products generation (number of AND gates) and the costs of the required Dadda trees (number of FAs and HAs) to sum them. If the unit gate model is used which counts a full adder as 7 gates and a half adder as 3 gates, the relative cost of the partial product generation decreases with the multiplier size. If these estimations reflect the real grow rate, the usage of M2 multipliers becomes more effective with larger multipliers compared to other elementary types.

Considering that M2 has the smallest area and it reduces the addition tree, it can be expected that this type is the most efficient 2x2 multiplier among the considered types. The only drawback is a larger variance compared to M1. The tradeoff is very simple as there is only one type. It starts with all blocks of the accurate type M0 and the next block with the smallest expected error is made approximate. This process is repeated until all the blocks are of the type M2.

The M4-only combinations are always less efficient than M2-only combinations as the M4 type has a larger variance and cost. Therefore, M4 combinations are not included in the comparison.

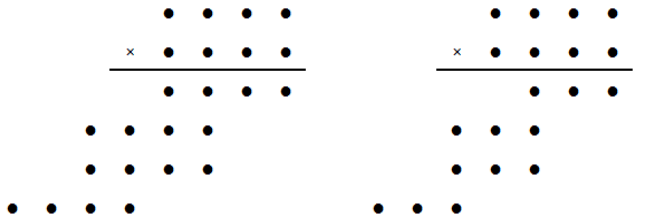


Figure 10. Partial products produced by 2x2 multipliers when they are used in a 4x4 multiplier.

Multiplier size	AND gates for partial products	Full adders for Dadda tree	Half adders for Dadda tree	Dadda tree cost estimation in gates	Partial products cost divided by Dadda tree cost
4x4	16	3	1	24	0.67
8x8	64	34	6	256	0.25
16x16	256	194	14	1400	0.18
32x32	1024	898	30	6376	0.16
64x64	4096	3842	62	27080	0.15

Table 4. Costs of partial products and Dadda trees for multipliers of different sizes according to the simple gate model.

M1 combinations

Combinations consisting of only M1 and M0 types are also included in the comparison as M1 has the lowest variance among the considered 2x2 multipliers (if the input distribution is uniform). M1, however, does not have a positive impact on the compression tree, so it is expected that the M1-only approach is worse than the M2-only combinations. The tradeoff generation is the same as for M2.

Leading M3 combinations (covariance aware combinations)

In the low-variance approach, the covariance is ignored. If a design does not contain an M3 type multiplier, the covariance can be ignored as in such combinations all the correlations are positive and the total variance is at least the sum of the individual variances, taking into account the covariance can only make the variance larger. If, however, there is an M3 type multiplier in the combination, it can reduce the variance. As an example, if there is an error in a 2x2 M2 type block, and there is an M3 block with shared inputs, when the M2 type has an error, there is a higher probability that the M3 type will also produce an error, but of the opposite sign, which can partially reduce the variance of such a combination compared to a combination consisting of only the M2 type.

As an example, Fig. 11 shows the combination M0-M0-M3-M2, which will be labeled as 0032 for short. The most significant block is accurate, M0, and the least significant block is of type M2. M3 and M2 blocks have a shared input A_L. If the covariance is ignored, the variance in case of the uniform distribution can be estimated as $Var(0032) \approx Var(M2) + Var(M3) = 0.234375 + 16 \cdot 0.234375 = 3.984375$. If the covariance is included, the variance can be computed precisely as $Var(0032) = Var(M2) + Var(M3) + 2 \cdot Covar(M2, M3) = 0.234375 + 16 \cdot 0.234375 + 2 \cdot (-0.1904296875) = 3.603515625$. So, this positive correlation between the two blocks reduces the

variance. If there is an error in M2, there is a high probability of partially cancelling this error by the M3 block.

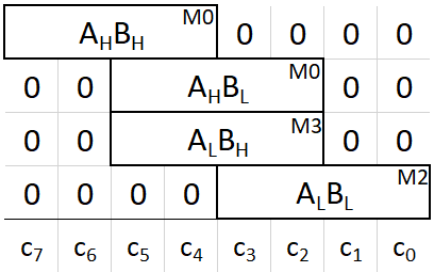


Figure 11. 0032 combination.

In comparison, if the combination is 0022, the variance becomes larger. If there is an error in one block, there is a higher probability of error in another. The variance of this combination is increased by the covariance term: $Var(0022) = 0.234375 + 16 \cdot 0.234375 + 2 \cdot 0.1904296875 = 4.365234375$. As can be seen, the covariance of the 0032 combination is smaller than of the 0022 combination. However, M3 is less effective compared to M2. To understand the impact of this effect, combinations consisting of M3 type at the most significant block followed by M2 blocks are also included in the comparison.

Truncation of partial products

Truncation of partial products is normally done by removing AND gates (Fig. 12). If this approach is used here, however, it will lead to a slightly different architecture compared to the recursive multipliers. Recursive multipliers use small 2x2 multipliers to produce four partial products. Essentially an accurate 2x2 multiplier compresses the four AND gates with two half adders such that the results have the height of one (Fig. 13a). The four AND gates can be seen in the scheme of the accurate 2x2 multiplier (Fig. 8). The two half adders can also be recognized there. One half adder consists of an XOR and an AND gate, and two XOR and two AND gates are visible in Fig. 8. Usage of half-adders for such initial reduction leads to a non-optimal adder tree. An example of this for a 4x4 multiplier is shown in Fig. 14. In Fig. 14a, 16 AND gates are used to generate the partial products, then 3 HA and 3 FA are used for compression, and finally 1 HA and 5 FA (a ripple-carry adder) produce the 8-bit result. In Fig. 14b, the operation of a recursive multiplier consisting of four 2x2 accurate multipliers is shown. As can be seen, it requires 16 AND gates in the first stage, 9 HA and 3 FA for compression, and the final addition can be done by 2 HA and 4 FA. The recursive multiplier has one less full adder but seven more half adders, which is a significant overhead resulting from this non-optimal reduction.

Table 5 shows the costs of an optimal 4x4 multiplier which does not use 2x2 multipliers, an accurate multiplier consisting of 2x2 blocks (0000), and approximate multipliers in which all 4 blocks are approximate with the same type. Out of the four 2x2 approximate multipliers, only M2 and M4 types lead to a reduced cost compared to the Dadda optimal multiplier depicted in Fig. 14a as instead of using half adders an OR (or XOR) gate is used (Fig. 13b). The 4x4 multipliers consisting of the M1 and M3 types have a larger cost compared with the Dadda optimal multiplier, which makes their usage inefficient if a multiplier is assembled from small 2x2 multipliers. This issue is not considered in the literature. Presumably, synthesis tools can combine 2x2 multipliers and optimize the whole structure. If it's not the case, it might happen that the effects of truncation are combined with the effects of a better compression of the partial products and it can be difficult to compare truncation with recursive multipliers. For this reason, to make the truncation method to have a similar architecture as recursive

multipliers, truncated 2x2 multipliers are used (Fig. 15). M7 has the LSB equal to zero. M6 has two LSBs equal to zero, and M5 computes correctly only the MSB bit. The principle stays the same as in Fig. 12a. By using three truncation types, the columns of partial products are gradually removed and as a result the LSB bits are always zero. Such truncation has a positive effect on all the MAC stages, leading to reduction in cost. The tradeoff is generated by gradually removing the partial products, so there is no adjustment made to a specific distribution.

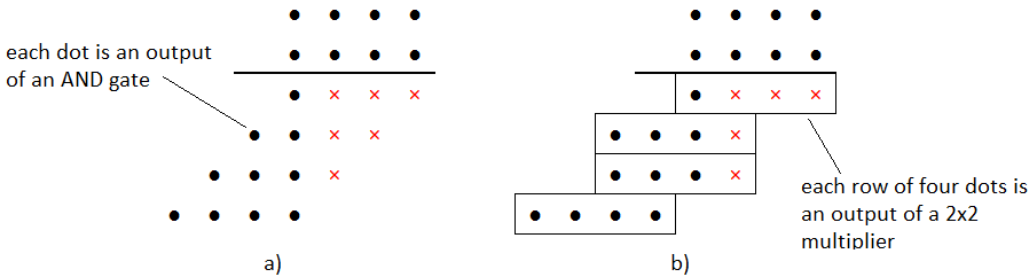


Figure 12. Usual truncation of a 4x4 multiplier (a) and truncation of a recursive multiplier using truncated 2x2 multipliers (b).

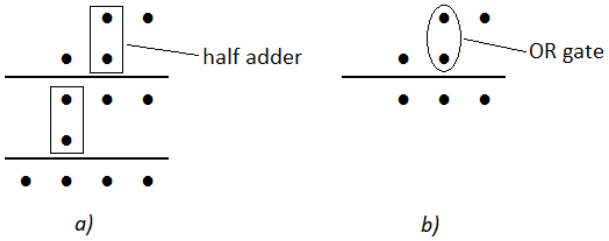


Figure 13. Accurate 2x2 multiplier (a) and M2 2x2 multiplier (b).

Design of a 4x4 multiplier	**Partial products generation cost**	**Reduction cost**	**Final addition cost**	**Total cost**
Dadda optimal	16	30	38	84
0000	40	24	34	98
1111	32	24	34	90
2222	20	13	27	60
3333	30	24	34	88
4444	24	13	27	64

Table 5. Cost of 4x4 multipliers.

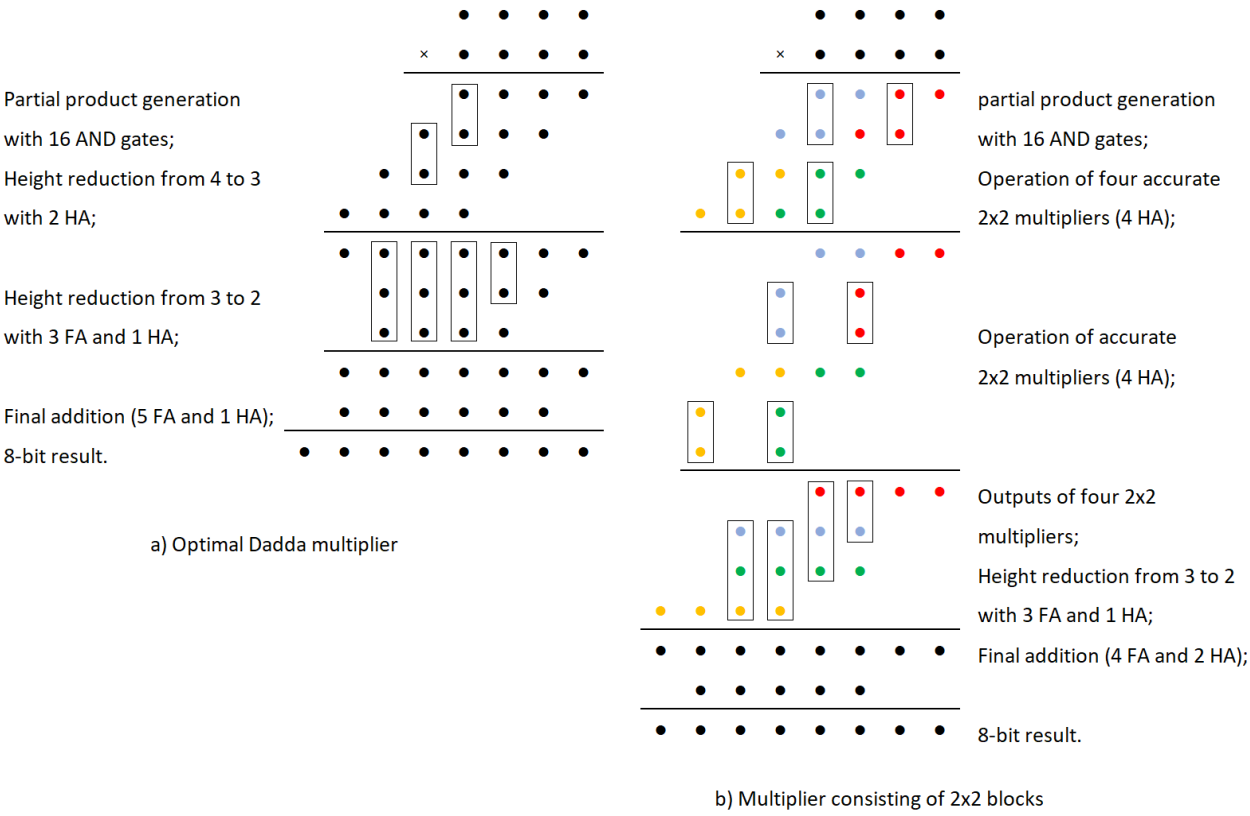


Figure 14. 4x4 multiplier with optimal Dadda tree (a); recursive multiplier using 2x2 blocks (b).

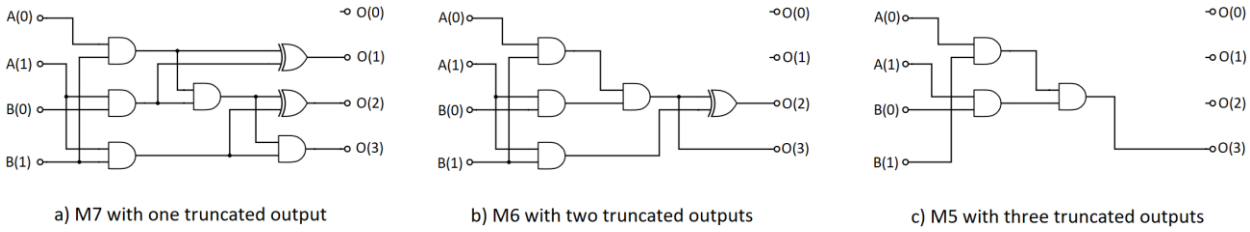


Figure 15. Truncated 2x2 multipliers equivalent to truncation of partial products.

Truncation of inputs

Another truncation technique is to make the LSB bits of the inputs to be zero. The corresponding AND gates can be removed (Fig 16a). Again, to make the architecture similar to the recursive multipliers, this method is implemented by using approximate 2x2 types, Fig. 17. The effect is the same in both cases. In the recursive case, the LSB zeros are propagated to the corresponding 2x2 blocks and these blocks can be simplified or removed (Fig. 16b). If two LSB bits are truncated, the least significant 2x2 multiplier can only have two possible values, 0 or 4. This multiplier takes as its inputs the two least significant bits of each operand, and when the bits are truncated, the inputs are either equal to 2 or to 0. This corresponds to the rectangle with one dot and three crosses in Fig. 16b. Similarly, the next two multipliers can only have values 0, 2, 4 or 6 (rectangles with two dots and two crosses). The most significant 2x2 multiplier is accurate in this case. The tradeoff is generated by gradually truncating the least significant bits.

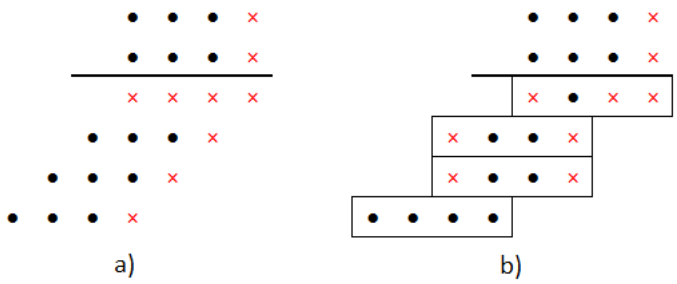


Figure 16. Usual truncation of inputs in a 4x4 multiplier(a) and truncation of inputs in a recursive multiplier using truncated 2x2 blocks (b).

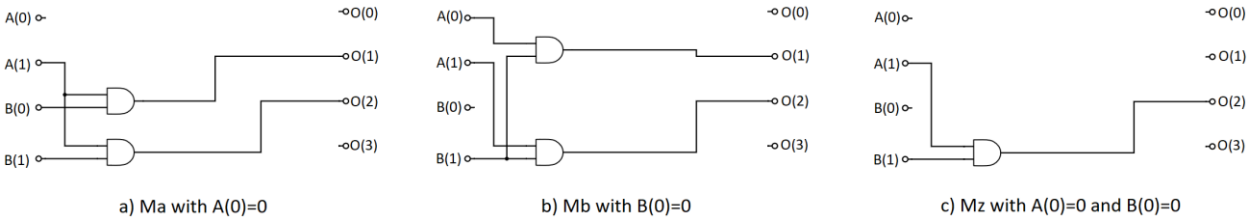


Figure 17. Truncated 2x2 multipliers equivalent to the truncation of inputs.

2.7.2. Dynamic Range Unbiased Multiplier (DRUM).

A different approach to approximate multiplication is proposed by [18]. This method in some sense is similar to truncation, as the operands are truncated to a smaller size and then multiplied by a smaller multiplier. The smaller result of the multiplication must be shifted by the number of bits truncated from each of the operands. This truncation depends on the value of the operand, as it chooses a certain number of bits starting from the first detected 1 (scanning from left to right, i.e., from the MSB to the LSB). If a number of bits are just dropped after some point, then there is a biased error introduced, because the truncated number is always smaller in expectation. To remove this negative bias, the LSB of the reduced operand is always set to 1. An example is described in Table 6. If the operand has eight bits and two bits are truncated, the error can be from 0 to -3, leading to the mean error of -1.5 if these three bits are uniformly distributed. If the most significant truncated bit is assumed to be one, the error range is fro minus 3 to 4, leading to the mean error of 0.5. This is an example where the mean error is made smaller at the expense of introducing higher variance as the negative errors are balanced by the positive errors, but now the error case of +4 is possible.

Exact value	Value if two bits are truncated	Error	Value with unbiasing	Error with unbiasing
xxxxx000	xxxxx000	0	xxxxx100	+4
xxxxx001	xxxxx000	-1	xxxxx100	+3
xxxxx010	xxxxx000	-2	xxxxx100	+2
xxxxx011	xxxxx000	-3	xxxxx100	+1
xxxxx100	xxxxx100	0	xxxxx100	0
xxxxx101	xxxxx100	-1	xxxxx100	-1
xxxxx110	xxxxx100	-2	xxxxx100	-2
xxxxx111	xxxxx100	-3	xxxxx100	-3

Table 6. Truncation and its unbiasing.

The operation of a DRUM 6x6 multiplier applied to 16x16 multiplication is illustrated in Fig. 18. The 16-bit unsigned numbers 5965 and 346 are multiplied by an accurate 6x6 multiplier. Initially, the leading ones of both operands are detected. Starting from them, 5 bits are taken directly from the operand, and the sixth bit is always set to one for unbiasing. The bits after that are assumed to be zero. As can be seen, instead of multiplying 5965x346, the multiplication 47x43 is performed. The result is then shifted by the number of truncated bits, ten in this case. The total effect is that instead of multiplying 5965x346, the multiplication 6016x344 is performed, leading to an error.

The cost is saved by using a smaller accurate multiplier, however some overhead is introduced to detect the leading ones in each operand, select the required bits, determine the amount of shifting after the multiplication and then shift it. The main limitation of this method is that it cannot be directly used for signed multiplication, as it requires the leading one detection, and 2's complement negative numbers always have a 1 at the MSB. The operands have to be converted to the absolute value, multiplied, and then the result has to be converted back in the 2's complement representation. Also, the unbiasing is valid only if the truncated bits are uniformly distributed. Other distributions can still introduce some bias.

Instead of an accurate multiplier any multiplier architecture can be used. The tradeoff can be generated by using multipliers of different sizes. For example, for 8x8 multiplication, the accurate multiplier can be a 7x7, 6x6, …, 2x2 multiplier. For the comparison, this smaller multiplier is assumed to be consisting of accurate 2x2 blocks. In this case it can be compared to truncation in the same way as with recursive multipliers. However, to observe the effect of using these 2x2 blocks which may lead to non-optimal multipliers, also multipliers chosen by the synthesis tool are included, i.e., multipliers generated by the tool from a behavioral description like `A*B` in VHDL. Also, the leading one detection logic and the logic which determines the shifting value (by how many bits to shift the result of the smaller multiplier) can be implemented differently. For example, the shifting value can be determined by using multiplexers, but also a small adder can be used. For this reason, several different architectures are implemented for the comparison.

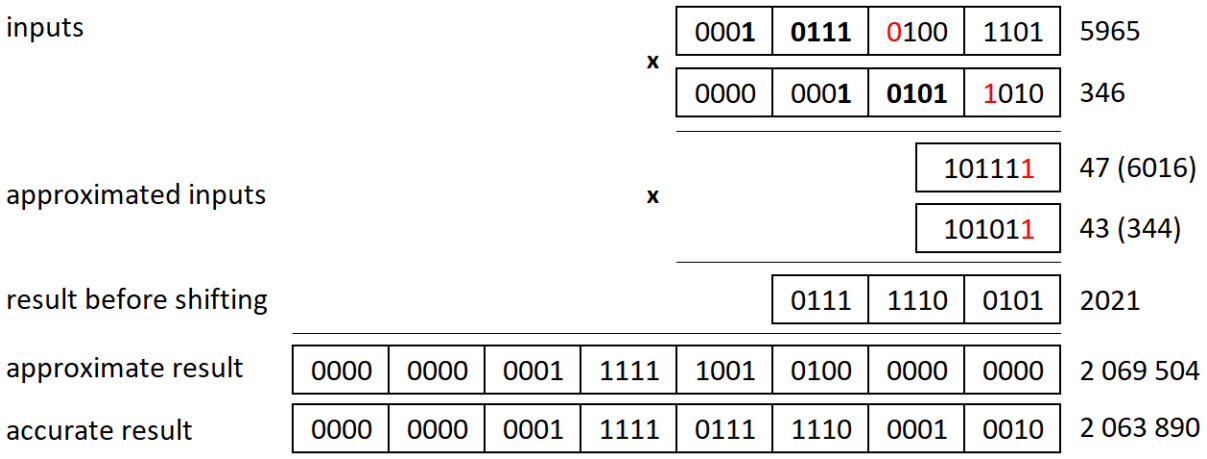


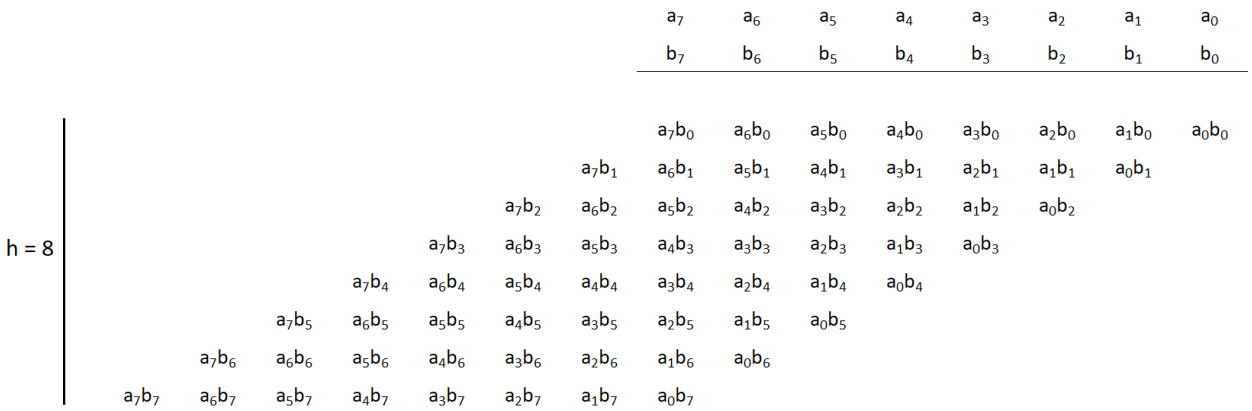
Figure 18. DRUM multiplier operation [18].

2.7.3. Low-Power Approximate MAC Unit.

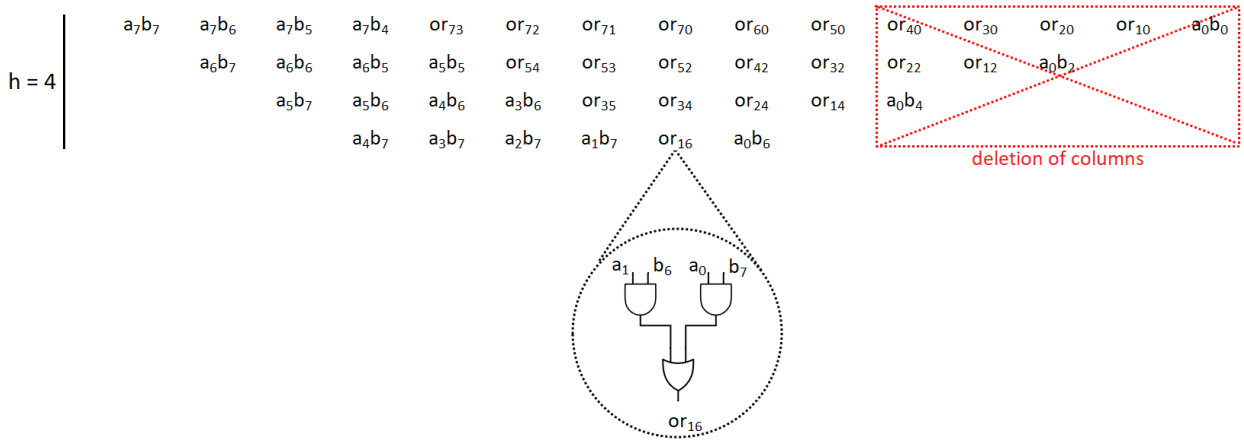
This technique is based on two types of approximations:

- Sum of two partial products is approximated by an OR-gate
- A number of least significant columns is deleted

This method is similar to using 2x2 multipliers of the type M2. In M2 the sum of two partial products is also approximated by an OR-gate. In this method, however, the most significant bits of 2x2 blocks are also OR-ed with least significant bits of other 2x2 blocks to enable more compression and reduce the height of the partial product matrix by a factor of 2. Such compression leads to savings in the adder tree. However, this compression requires approximating high order partial products and the least approximate design of this method has a very large error. As can be seen in Fig. 19, the OR-compression must be used for the columns 0-10. This least approximate design can be approximated further by deleting the least significant columns. This work does not compare compression with column deletion (truncation) separately, truncation is used only in combination with approximate compression. It might be that using only truncation produces a better trade-off. As this method essentially uses the M2 multiplier, it can also be constructed by using 2x2 blocks and then OR the outputs before summation. This way it can be compared with truncation as well. The unbiasing in this method is done by placing an initial value in the accumulation register. The tradeoff is generated by gradually truncating the columns of the initial approximate design.



a) partial product matrix of an accurate 8x8 multiplier



b) resulting partial product matrix after OR-compression and column deletion

Figure 19. Low-Power Approximate MAC Unit [1].

2.7.4. Approximate Booth multiplier.

This approximation method is based on the modified Booth algorithm [25]. In the radix-4 modified Booth algorithm three bits of the multiplier are used to determine how many multiplicands to add, 0, ± 1 or ± 2. The partial product matrix of this method applied to an 8x8 multiplication is shown in Fig. 20. If the multiplier has eight bits, there are four groups of three bits which determine the four rows to be added by the adder tree. There are also compensation terms equal to one at some rows, and sign factors.

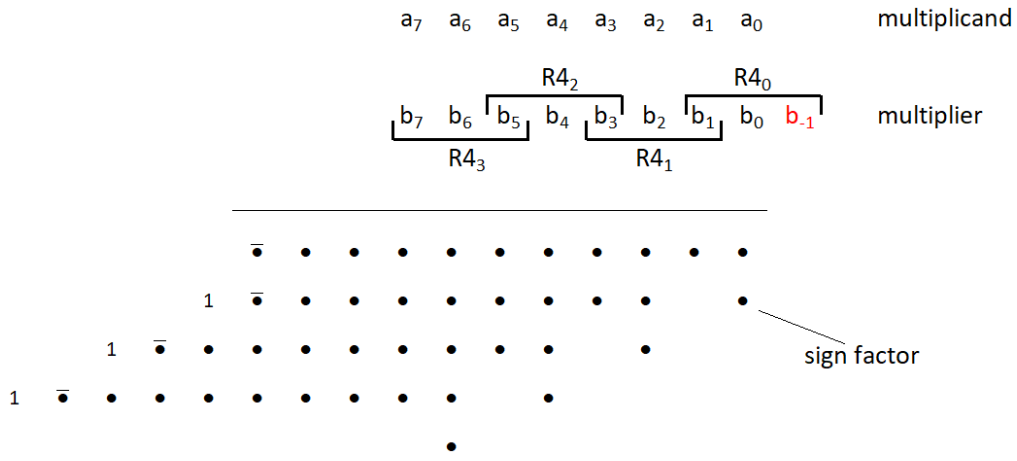


Figure 20. Accurate 8x8 multiplier using the radix-4 Booth algorithm.

This algorithm allows to reduce the height of the partial product matrix, as instead of 8 rows there are only four rows, but the partial products generation requires more complex logic compared to AND gates.

In this approximation method, along with the radix-4 encoding, the radix-16 encoding is also used for the least significant five bits of the multiplier. In this case the five bits are used to determine whether to add 0, $\pm 1, \pm 2, \pm 3, \pm 4, \pm 5, \pm 6, \pm 7$ or ± 8 (Fig. 21). In comparison with the radix-4 Booth algorithm which uses only shifting to get ± 2, the radix-16 method has non-power of 2 cases which are more difficult and require additional adder. In this method these non-power-of-two factors are mapped to the nearest power-of-2 factors, and the partial products are generated by using only shifting as in the radix-4 case. This mapping approach introduces errors, but it allows to reduce the height of the matrix as there are only three rows in the partial product matrix instead of the four rows as in the accurate radix-4 case.

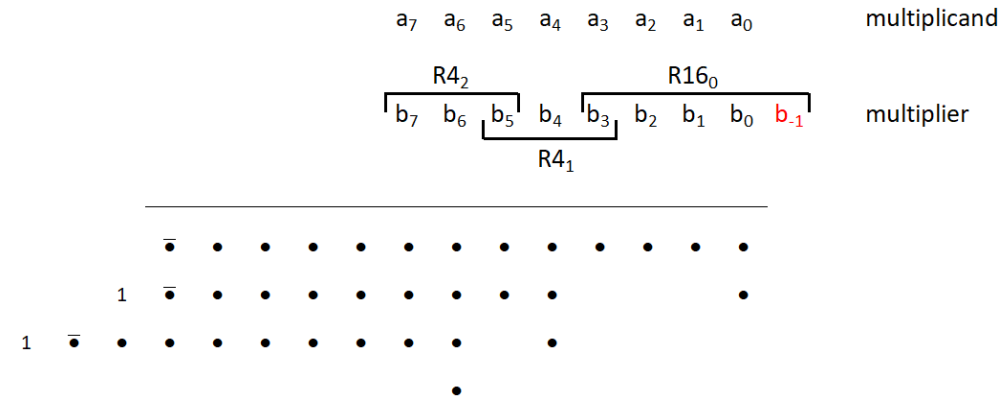


Figure 21. Approximate 8x8 multiplier using hybrid R4/R16 encoding.

The work which proposes this type of approximation does not compare it with an accurate Booth multiplier with truncated least significant partial products. As this multiplier is the only one in the comparison which cannot be constructed using 2x2 elementary multipliers, the comparison with truncation is done in a different way. Truncated designs are using the accurate radix-4 encoding, but some of the least significant partial products are not formed (Fig. 22). This multiplier assumes two's complement numbers as inputs, so it's included only in the two's complement comparison.

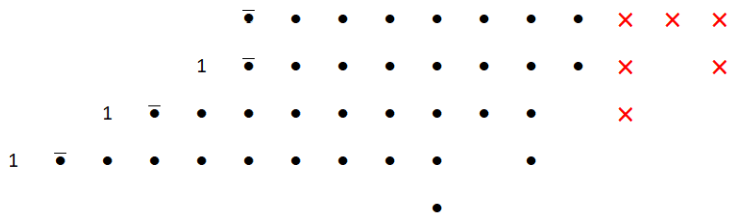


Figure 22. Truncation of partial products of the accurate 8x8 radix-4 Booth multiplier.

3. Comparison of approximation methods.

3.1. Experimental setup for comparisons.

The experimental setup for comparisons can be seen in Fig. 23. All designs are implemented in VHDL and synthesized by Synopsys Design Compiler using the TSMC 40nm Low Power (TCBN40LP) technology library. The logic synthesis produces an area report for each design, and also a gate-level netlist and standard delay file (SDF) which can be used by simulation software (Questasim). A simulation is using an SDF file, a synthesized design and provided input data to record the switching activity of the circuit (SAIF file). The input in each simulation consists of two vectors with 10000 elements which are multiplied and accumulated. The produced SAIF file is then used by Synopsis Power Compiler to perform power estimation.

For assessing the error behavior of approximate designs, the corresponding MATLAB models are implemented. The VHDL and MATLAB models are cross-validated to make sure that the models work identically. 60000 vectors (30000 for both operands) with 500 elements are used to compute 30000 accurate and approximate MAC results. The error is then computed by comparing the accurate and approximate results according to an error metric (mean squared error, absolute mean error and mean absolute percentage error).

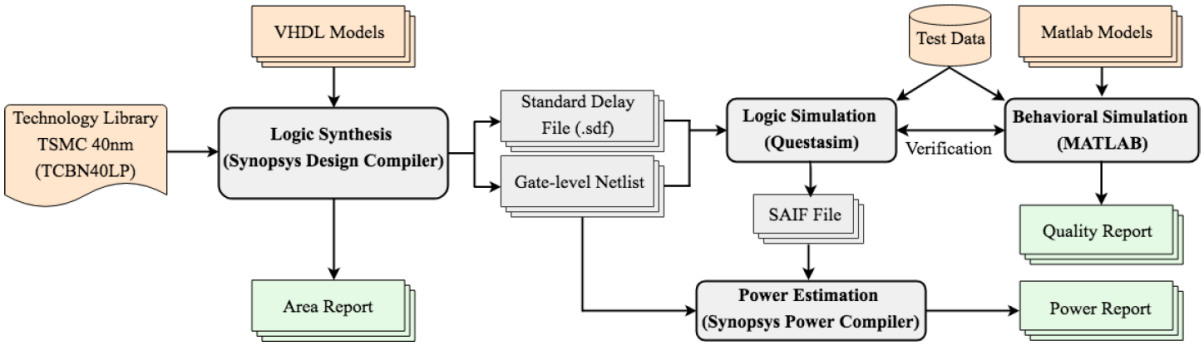


Figure 23. Experimental setup for quality-efficiency tradeoff comparison between different approximate methods [19].

The following approximation methods are included in the comparison:

1. Recursive multipliers
    a. Low mean error combinations (M0, M1, M2, M3, M4)
    b. Low variance combinations (M0, M1, M2)
    c. Low MSE combinations (M0, M1, M2, M3, M4)
    d. M2 combinations (M0, M2)
    e. M1 combinations (M0, M1)
    f. Leading M3 combinations (M0, M2, M3)
    g. Truncation of partial products (M0, M5, M6, M7)
    h. Truncation of inputs (M0, Ma, Mb, Mz)
2. Dynamic range unbiased multiplier (DRUM)
    a. With small multiplier consisting of 2x2 blocks
    b. With small multiplier chosen by the synthesis tool
3. Low power approximate MAC unit
4. Approximate hybrid radix Booth multiplier
5. Truncation of the Booth partial product matrix

6. Truncation of partial products applied to the partial product matrix (only for 16x16 experiments)
7. Truncation of inputs applied to the partial product matrix (only for 16x16)
8. OR-compression applied to the partial product matrix (only for 16x16)

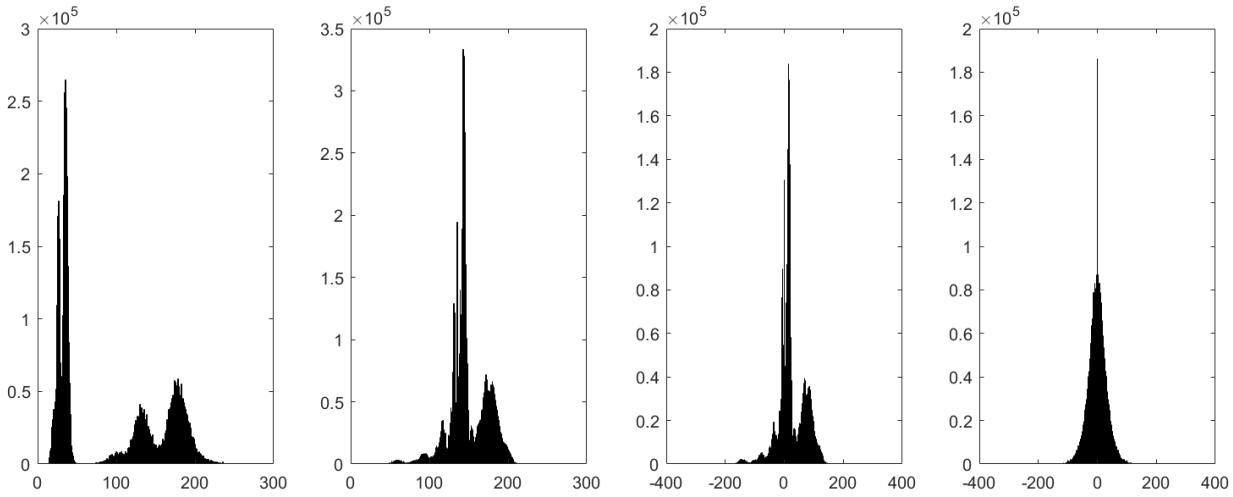
The comparisons are made for the following bit sizes, data representations and input distributions:

1. 4x4 case
    a. Unsigned data representation
        i. Uniform distribution
        ii. Normal distribution with mean=8 and std=2.5 for both inputs
        iii. Radio-astronomy calibration input (Stefcal_CE)
    b. Signed magnitude representation
        i. Uniform distribution
        ii. Normal distribution with mean=0 and std=5 for both inputs
        iii. Radio-astronomy calibration input (Stefcal_ET)
2. 8x8 case
    a. Unsigned data representation
        i. Uniform distribution
        ii. Normal distribution with mean=128 and std=40 for both inputs
        iii. Radio-astronomy calibration input (Stefcal_CE)
    b. Signed magnitude representation
        i. Uniform distribution
        ii. Normal distribution with mean=0 and std=80 for both inputs
        iii. Radio-astronomy calibration input (Stefcal_ET)
    c. Two's complement representation
        i. Uniform distribution
        ii. Normal distribution with mean=0 and std=40 for both inputs
        iii. Radio-astronomy calibration input (Stefcal_ET)
3. 16x16 case
    a. Unsigned data representation
        i. Uniform distribution
        ii. Normal distribution with mean=32768 and std=10240 for both inputs
        iii. Radio-astronomy calibration input (Stefcal_CE)
    b. Signed magnitude representation
        i. Uniform distribution
        ii. Normal distribution with mean=0 and std=20480 for both inputs
        iii. Radio-astronomy calibration input (Stefcal_ET)
    c. Two's complement representation
        i. Uniform distribution
        ii. Normal distribution with mean=0 and std=10240 for both inputs
        iii. Radio-astronomy calibration input (Stefcal_ET)

As a large number of designs have to be synthesized and simulated for different data representations and distributions, it would be infeasible to perform this work manually. Therefore, a script has been made which takes a design identifier, changes the corresponding VHDL files, synthesizes the design, performs a simulation, estimates the power and writes the area and power numbers to a text file. An identifier is a combination in case of recursive multipliers (for example, 0000_0000_0000_0002 represents a combination with all 2x2 multipliers accurate except the least significant block, which is of type M2), or a name of the approximate method followed by a parameter (for example, lowpowmac_1 represents the low-power approximate MAC unit with 1 deleted least-significant column). In this way, more than 1000 designs have been synthesized and simulated.

Uniform and normal distributions are standard distributions to apply to an approximate design. As a multiplier takes two inputs, these inputs in general can have different distributions, and may be correlated or uncorrelated. In this comparison, for the uniform and normal cases both operands of the multiplier are sampled from the same distribution and are uncorrelated.

To test the designs on more realistic distributions, the distributions from the radio-astronomy calibration application are also used (Fig. 24). The values in this application are larger than 4/8 bits, so they are scaled to fit in 4-bit and 8-bit operands. The Stefcal_CE distribution is used for unsigned comparisons and denotes the distributions of the signals C and E of the Stefcal algorithm, which will be described in chapter 4. In the Stefcal algorithm the signal E has both positive and negative values, so this distribution is shifted to the right for the unsigned comparisons to have only positive values. The Stefcal_ET distribution is used for the signed comparisons. In this case the signal E has the same shape as for the unsigned version but is not shifted and has both positive and negative values. In these distributions the two operand values have different distributions and have some degree of correlation between them. The Pearson correlation coefficient for the signals E and T is equal to 0.02, which means that there is a slight correlation between the two signals. For comparison, two uniformly distributed inputs have the correlation coefficient of 0.0001. The signals C and E have the correlation coefficient of 0.45 which denotes a significant correlation between the two signals. The signal E is computed using four signals, and one of them is the signal T, which explains this high correlation. These Stefcal distributions are not used to determine the right approximation strategy which can be applied to the Stefcal algorithm, as the Stefcal has a large number of signals, the real signals require more bits, and the computation is iterative, so the results are not directly translated to the approximation of the whole Stefcal algorithm. These distributions are added to test the effect of approximations on some distributions which are different from the idealistic uniform and normal distributions and represent data from a real application.



a) Stefcal_CE distribution scaled to 8 bits (unsigned representation)

b) Stefcal_ET distribution scaled to 9 bits (signed-magnitude representation)

Figure 24. Stefcal distributions.

To test the designs on signed numbers, the signed magnitude and two's complement representations are used. In the signed magnitude representation, the most significant bit represents the sign of an operand. In these experiments, the sign bit is added to a 4/8-bit operand and make it a 5/9-bit operand. The multiplication is performed on the absolute value and then the result is adjusted according to the signs of the operands. The sign of a multiplication result is equal to the exclusive-OR (XOR) of the sign bits. The multiplication is performed on the 4/8-bit absolute values which are unsigned.

Two's complement representation is also included in the comparison as it is the most common representation used for representing signed numbers. However, some 2x2 types cannot be directly used for the multiplication of two's complement numbers. In the existing literature they are used only for the unsigned multiplication. The partial products matrix for two's complement numbers is shown in Fig. 25a. As can be seen, some boundary partial products have negative weights. To remove these negative signs, the Baugh-Wooley method can be used [20], Fig 25b. In this method, certain bits are inverted and compensation terms are added. The 2x2 accurate multipliers which correspond to the modified partial products have to be adjusted, and if the considered approximate multipliers are used at their place, the errors for the M1 and M3 types are much higher, which makes them infeasible to use at these boundary positions. This is demonstrated in Fig. 26 where the truth tables of the 2x2 multipliers are shown when the inputs to them are inverted according to the Baugh-Wooley modified matrix of Fig. 25b. The M2 and M4 types have a similar behavior as in the unsigned case, as they have the same error magnitudes of -2 and -4, but differing error cases. The 4x4 case does not include two's complement representation, as the three out of four 2x2 multipliers are affected by these modifications. In the 8x8 case, seven out of sixteen are affected. These seven 2x2 multipliers are therefore kept accurate in this comparison, and at the other nine positions the considered 2x2 approximate multipliers can be used. The DRUM multiplier works only with unsigned operands, so the operands are converted from two's complement to the unsigned absolute values, the multiplication is performed on the unsigned values, and then the result is converted back to two's complement. The low-power MAC method can be used with the Baugh-Wooley method as well. The approximate Booth multiplier can be used directly on the 2's complement numbers without any modifications.

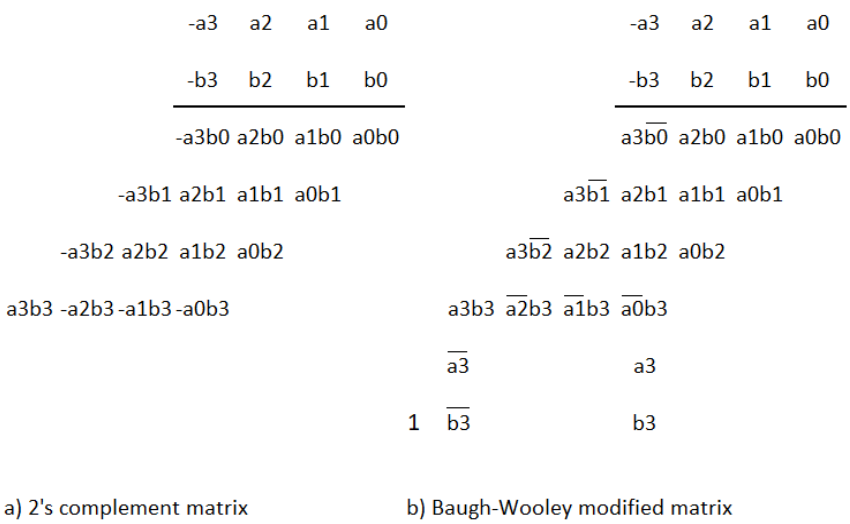
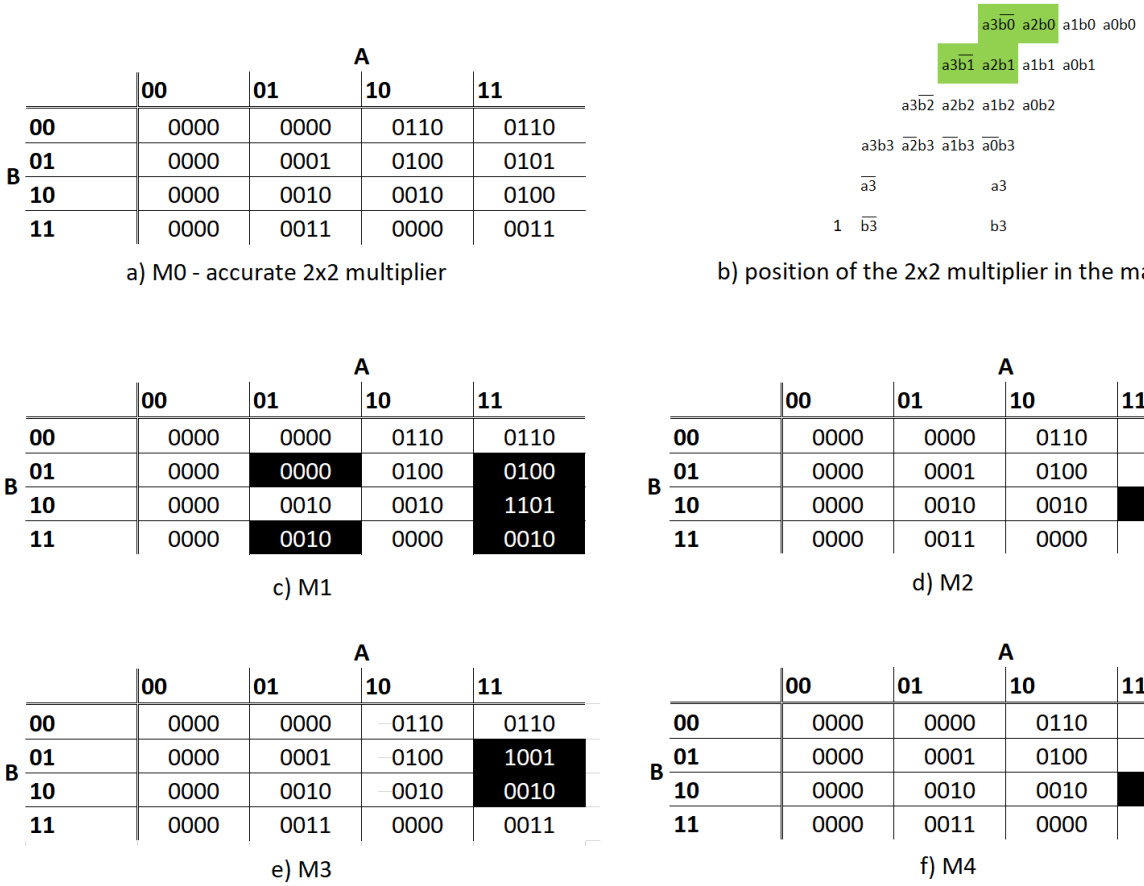


Figure 25. Two's complement multiplication [20].



A

B	00	01	10	11
00	0000	0000	0110	0110
01	0000	0001	0100	0101
10	0000	0010	0010	0100
11	0000	0011	0000	0011

a) M0 - accurate 2x2 multiplier

b) position of the 2x2 multiplier in the matrix

A

B	00	01	10	11
00	0000	0000	0110	0110
01	0000	0000	0100	0100
10	0000	0010	0010	1101
11	0000	0010	0000	0010

c) M1

A

B	00	01	10	11
00	0000	0000	0110	0110
01	0000	0001	0100	0101
10	0000	0010	0010	0010
11	0000	0011	0000	0011

d) M2

A

B	00	01	10	11
00	0000	0000	0110	0110
01	0000	0001	0100	1001
10	0000	0010	0010	0010
11	0000	0011	0000	0011

e) M3

A

B	00	01	10	11
00	0000	0000	0110	0110
01	0000	0001	0100	0101
10	0000	0010	0010	0000
11	0000	0011	0000	0011

f) M4

Figure 26. 2x2 multiplier truth tables for 2s complement representation when used at positions with inverted input bits.

3.2. Comparison results.

In this section the main conclusions that can be made from the comparison are discussed. Here only the relevant plots are presented. All the plots can be found in appendix A.

It can be hard to use the Synopsys Design Compiler for comparisons of approximate designs. Results for different frequencies and different compilation strategies can vary significantly. The synthesis tool considers all designs as individual instances and can apply different techniques and efforts for their optimization. The experimental results are often scattered, and it can be difficult to draw conclusions. For this reason, a model which computes the number of gates for each design combination was implemented (only for recursive multipliers). Also, as the synthesis tool treats the timing constraint as the most important to satisfy, it can be helpful to remove this latency constraint by using an unrealistic frequency of 1Hz. All approximate designs are synthesized for realistic frequencies ranging from 400 MHz to 1 GHz. The model and 1Hz predictions can be used to better understand the tradeoffs and see if a certain result is true in general or it is specific to a certain frequency. For this reason, along with the realistic frequencies, sometimes the model and 1Hz results are also included. Interestingly, model predictions usually closely follow the 1Hz results.

The results will be presented in the way shown in Fig. 27. On the y-axis, the mean-squared error represents the error associated with the designs. During the comparison the mean absolute percentage error (MAPE) and the mean absolute error (MAE) were computed as well. However, there is no significant difference between these three metrics, and the MAPE for some designs is undefined because of the division by zero, so here only the MSE is presented. On the x-axis the percentage of the nominal power is used. The nominal power is the power of the accurate design consisting of the M0 type 2x2 multipliers. The area numbers were also computed, but as the main goal of the Stefcal approximation is the energy reduction, only the power results are presented here.

Each approximation method has multiple designs with different error-power pairs, representing the tradeoff between the accuracy and power savings. The best designs are the ones which lie on the Pareto front. The Pareto front can consist of points corresponding to different approximation methods. The point B in Fig. 27 is not part of the Pareto front as it is dominated by the point A which has lower power and a smaller error.

The results are shown in two versions, the biased version and the unbiased one. The biased results are the ones in which the accumulator is initialized with the zero value, so the mean error is not compensated, and the error distribution may be positioned away from zero. In the unbiased results the accumulator is initialized with a compensation term which is equal to the mean error of the multiplier, multiplied by the number of elements in the vectors. The error distributions are shifted to zero and the MSE becomes equal to the variance.

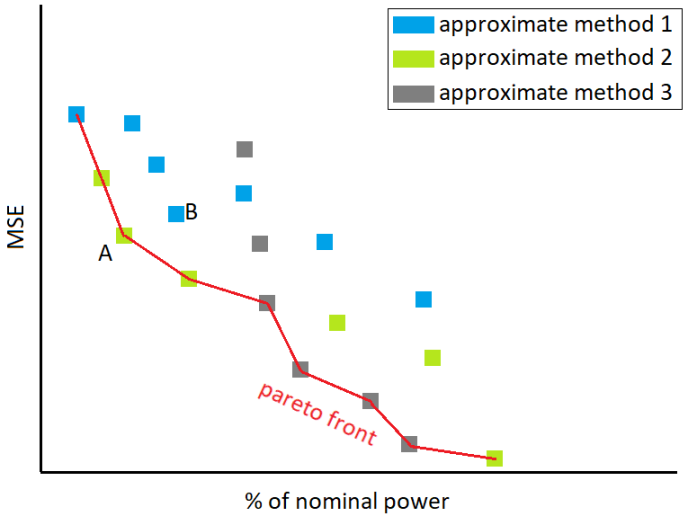


Figure 27. Pareto optimality.

3.2.1. M1 against M2 type

All experiments show that using only the M2 type is more effective than using only the M1 type. The M1 multiplier has a lower variance, but it saves less area and power compared to the M2 type. A typical tradeoff example can be seen in Fig. 28.

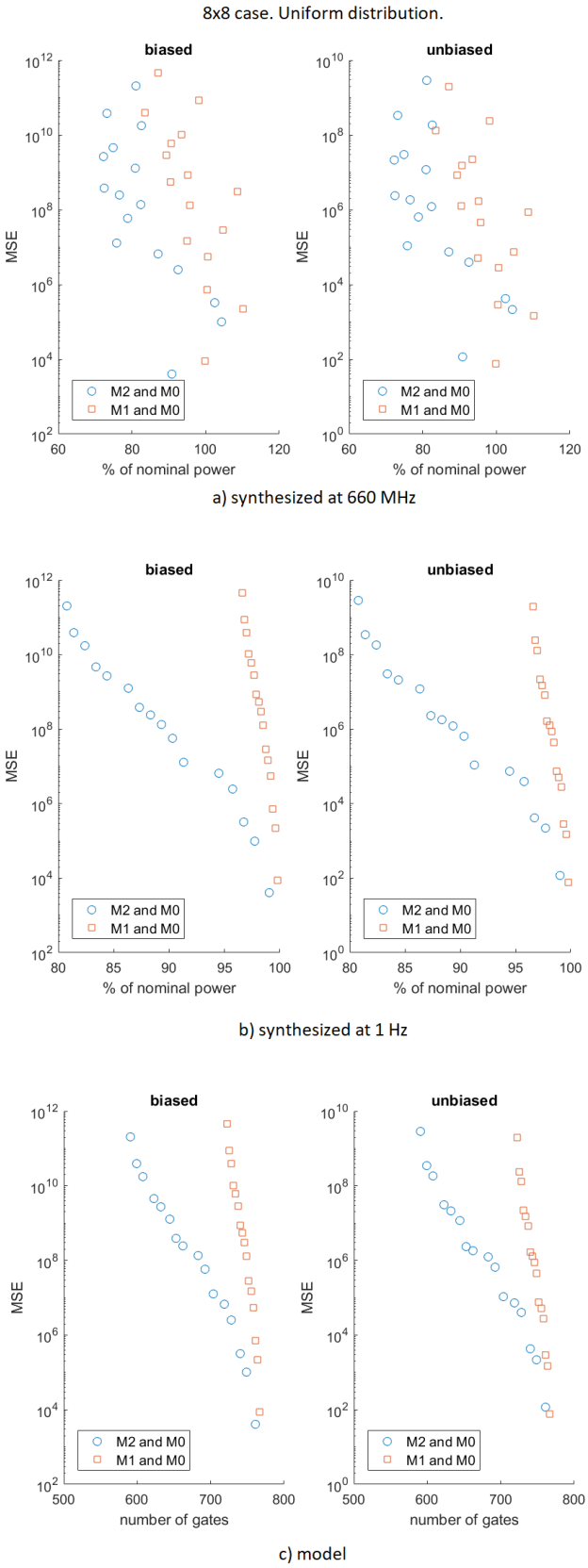


Figure 28. M1 vs M2 comparison.

3.2.2. MSE minimization against ME minimization.

As can be predicted by eq. 12, minimization of the mean error alone is less effective than the minimization of the mean squared error. The MSE minimization takes into account both the variance and the mean error of a combination leading to a better tradeoff. An example can be seen in Fig. 29.

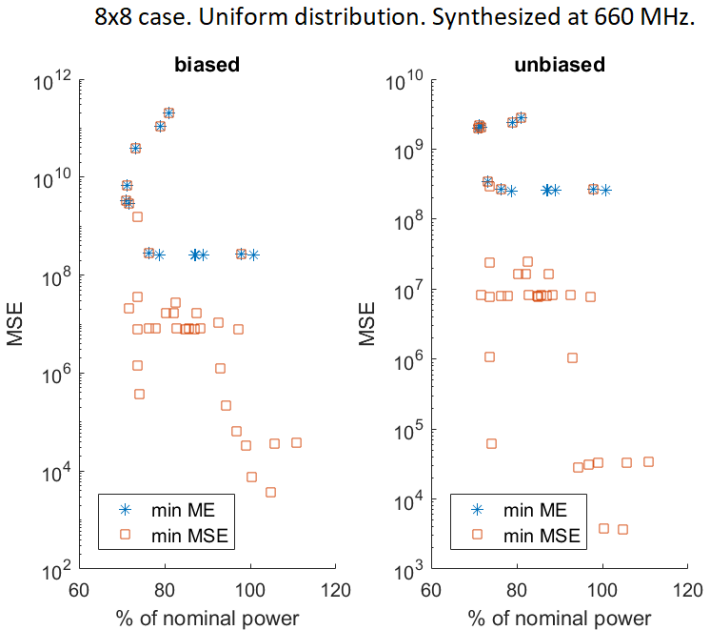


Figure 29. Minimization of MSE compared to minimization of ME.

3.2.3. MSE minimization against variance minimization.

Eq. 12 suggests that MSE minimization should produce better results compared to variance-only minimization if there is no unbiasing. If the unbiasing is used however, variance minimization is supposed to be more effective as in this case both the variance and mean error are optimized independently. Fig. 30 demonstrates the comparison. At 660 MHz the scattering of results is quite large and there is one result which stands out among the minimal variance combinations. This design corresponds to 0000_0000_0000_0002 combination (all blocks are accurate except the least significant one which is of the type M2). This design has lower power than expected. However, the larger effect of unbiasing on low variance combinations can still be seen in Fig. 30a and it is easily observed in Fig. 30 b) and c). The majority of low-variance combinations on the left plots are above the MSE-combinations, but with unbiasing they are shifted down more than the low-MSE combinations, making them slightly more effective.

The variance minimization combined with unbiasing is essentially the best way of using recursive multipliers consisting of 2x2 elementary blocks. As such combinations only use M1 and M2 types, it allows to significantly reduce the design space and simplify the design space exploration. Even though the M1 type is less effective than the M2 type, their combination can produce a better tradeoff compared to the only-M2 method (Fig. 31).

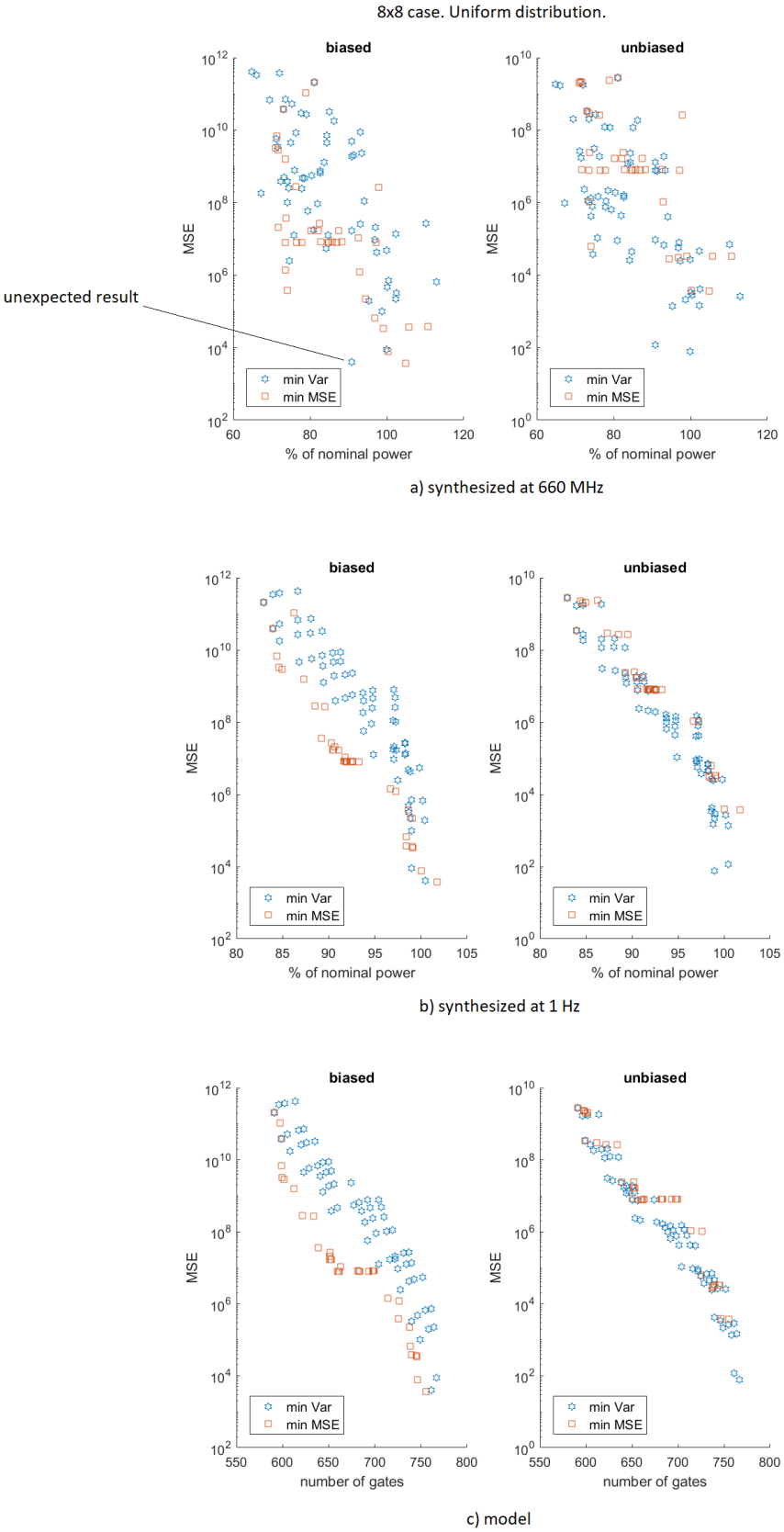


Figure 30. Minimization of MSE compared with minimization of variance.

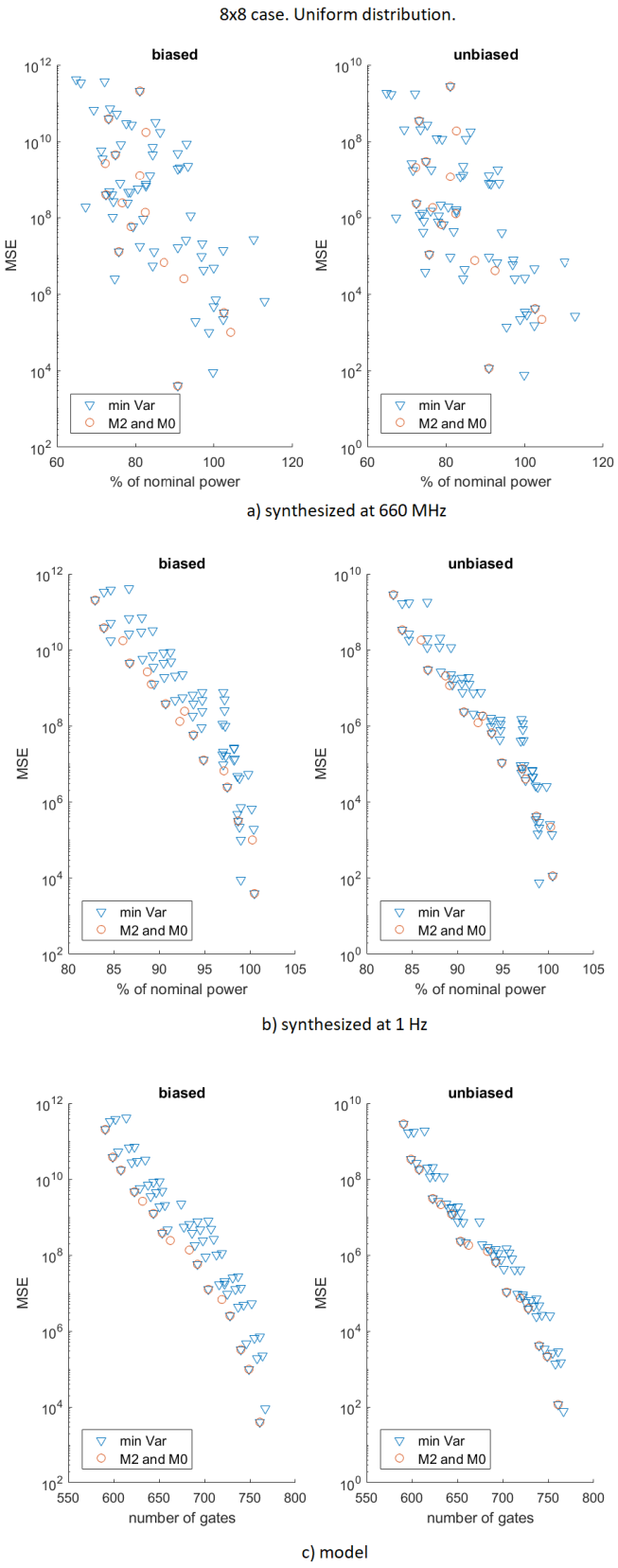


Figure 31. Comparison of M2-only combinations with M1/M2 combinations.

3.2.4. Leading M3 combinations against M2 combinations

Combinations with the M3 type at the most significant position show better results than M2 combinations if the unbiasing is not performed (Fig. 32a). However, as the results with unbiasing suggest (Fig. 32b), this effect is due to the mean error balancing rather than the variance balancing. As M3 has a positive error (3x3=11), the mean error of such combinations is closer to zero than in the M2-only combinations. This effect can be demonstrated as follows. If there are two combinations of a 4x4 multiplier, one with the M2 type at all positions (2222), and another one with the M3 type at the most significant position and the M2 type at the other positions (3222), the mean error of the (2222) combination in case of the uniform distribution is equal to $ME_{2222} = -2 \cdot 2^0 - 2 \cdot 2^2 - 2 \cdot 2^2 - 2 \cdot 2^4 = -50$. The mean error of the (3222) combination is $ME_{3222} = -2 \cdot 2^0 - 2 \cdot 2^2 - 2 \cdot 2^2 + 2 \cdot 2^4 = 14$. As can be seen, the mean error for the first three blocks is the same in both cases as they have the same M2 at their positions. Then, the most significant block M2 of the (2222) combination makes the mean error even smaller, whereas the M3 error has the same absolute error but with the opposite sign. Therefore, the absolute mean error of the combinations with the M3 at the most significant position and the M2 type at others is always smaller or equal to the M2-only combinations for any distribution. This explains a better performance in the biased case. However, if the results are unbiased, the gains are lost which means that the covariance does not have a large effect on the design combinations.

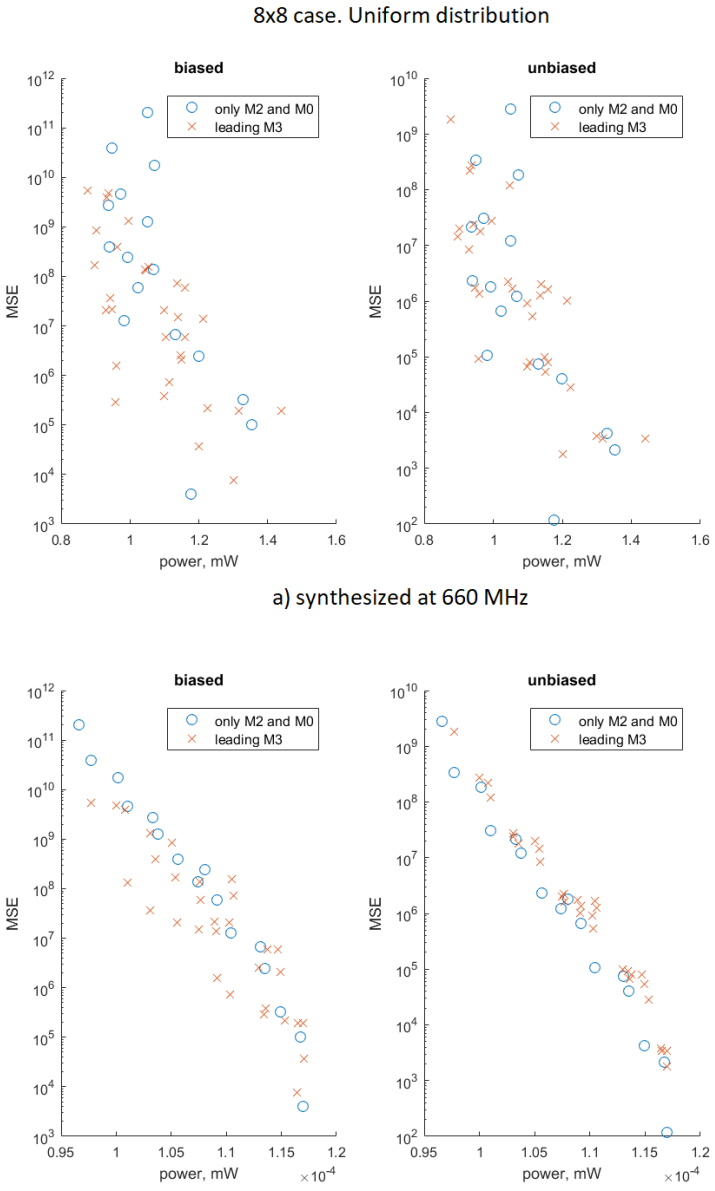


Figure 32. Leading M3 against M2.

3.2.5. Truncation of partial products against truncation of inputs

Truncation of partial products is more effective compared to truncation of inputs. As examples in Fig. 12a and 16a show, for a similar number of deleted partial products, the error introduced by the truncation of inputs is larger because more significant bits are removed. The synthesized tradeoffs can be seen in Fig. 33.

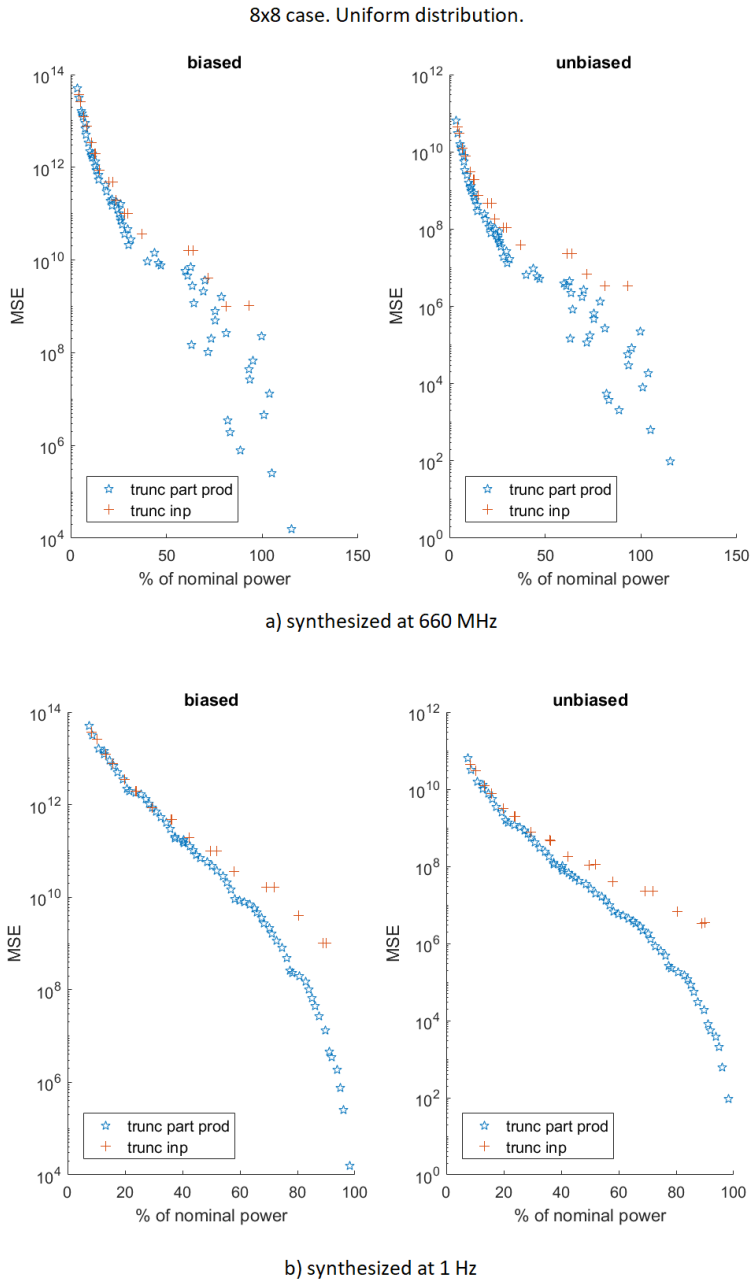


Figure 33. Truncation of partial products against truncation of inputs.

3.2.6. DRUM

The DRUM performance highly depends on the input distribution, as the errors introduced depend on the position of the leading 1 in the operands. In the uniform distribution a large fraction of numbers has a leading one at the most significant positions leading to large errors. However, because the DRUM is unbiased, it can be slightly more efficient than the truncation methods which have a negative bias. If the truncation results are unbiased as well, the advantage is lost. An example is shown in Fig. 34. One of the designs on the left is slightly better compared to truncation of partial products as the Pareto front includes one DRUM design. After unbiasing this advantage is not present.

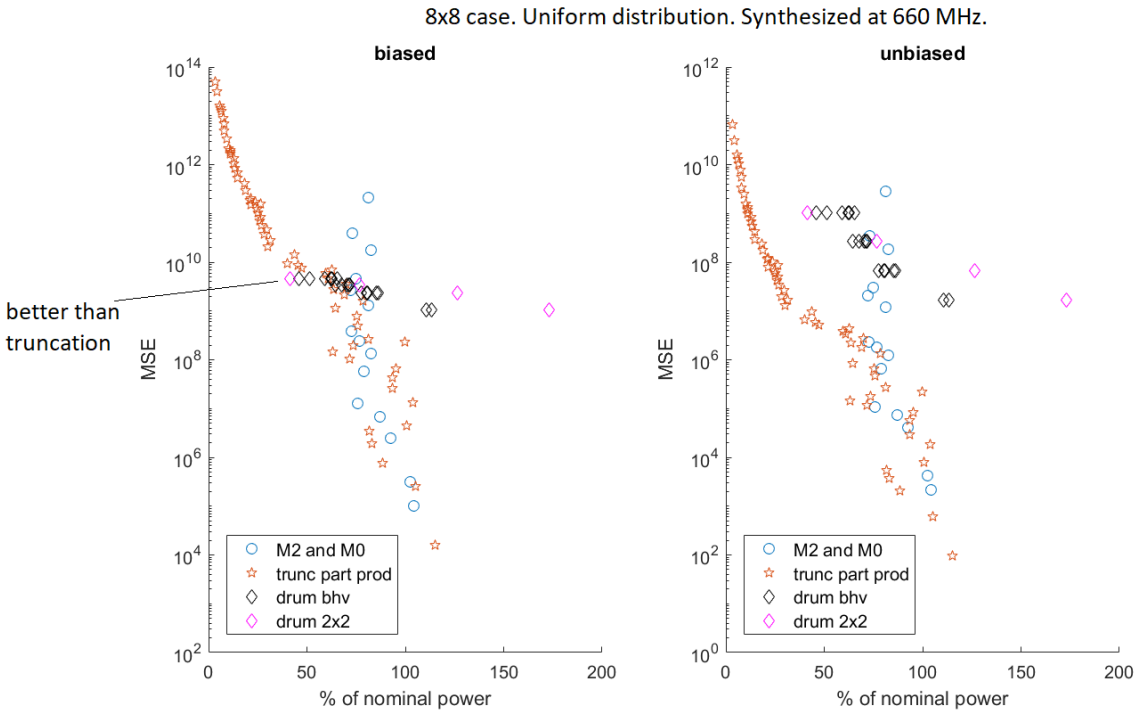
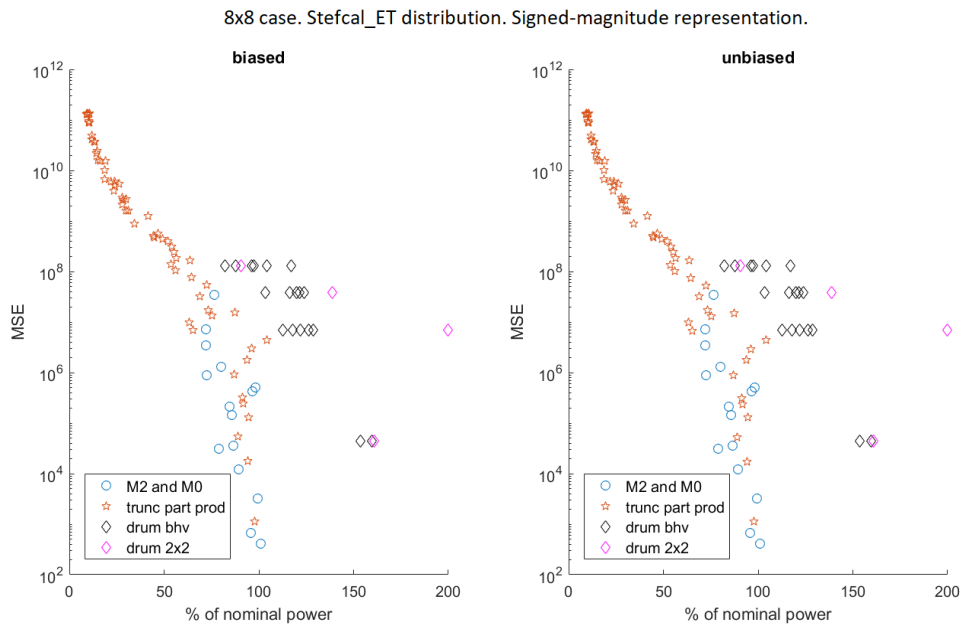
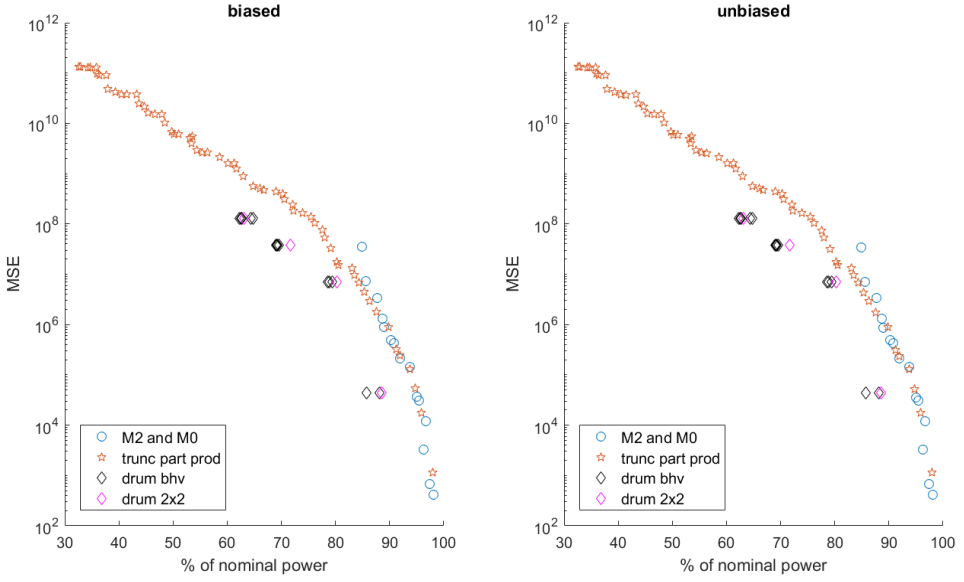


Figure 34. DRUM tradeoff (drum bhv – multiplier is selected by the synthesis tool; drum 2x2 – multiplier consists of 2x2 blocks)

All other experiments do not show advantages of DRUM compared to the partial products truncation. However, one distribution for which the DRUM performance stands out in the experiments is the stefcal_ET distribution. This distribution has inputs located around zero with only a small fraction of large numbers. The results show an advantage of the DRUM only in this distribution and only for a frequency of 1 Hz. An example is shown in Fig. 35.



a) synthesized at 660 MHz.



b) synthesized at 1 Hz.

Figure 35. DRUM performance on Stefcal distribution.

In general, the results suggest that the overhead of the DRUM due to the leading one detection and shifting of the result is quite significant for the 4x4 and 8x8 cases. Additional experiments are needed with larger bit widths to understand how this overhead scales with the size of the operands.

3.2.7. Low-power approximate MAC.

The smallest approximation of this method leads to a significant error as the higher order bits are approximated. The range of the tradeoff in this method is rather short for this reason. Typically, the tradeoff starts at a position above the partial product truncation curve, and as more and more columns are deleted it starts to follow the truncation tradeoff. An example is shown in Fig. 36.

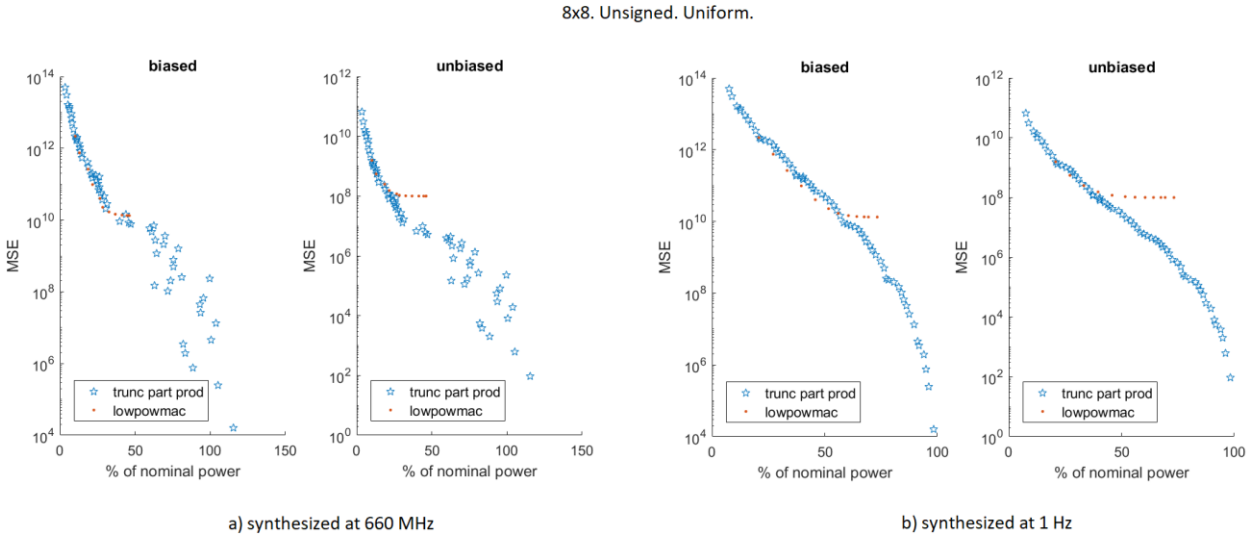


Figure 36. Low-power MAC tradeoff.

One of the advantages of this method is that the height of the partial product matrix is reduced by a factor of two which leads to a simplification of the required adder tree. In the partial products truncation the height stays the same up to the middle of the partial product matrix and then starts to gradually decrease. This fact may explain why in certain data representations and distributions some points of the low-power MAC method are better. An example can be seen in Fig. 37. Some points in the middle of the tradeoff demonstrate better performance compared to the truncation of partial products.

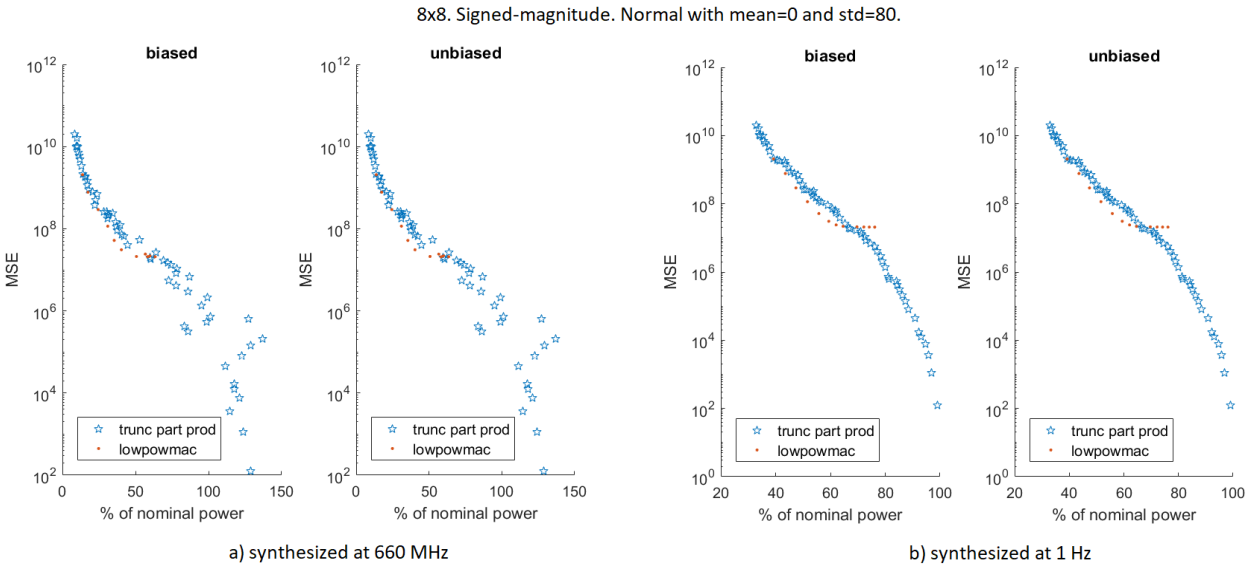


Figure 37. Example of Low-power MAC advantage.

3.2.8. Approximate Booth multiplier against truncated Booth multiplier.

For the 8x8 case, the approximate Booth multiplier has only one design point as 8 bits is the minimal operand size for which it can be used. This single approximate design is compared with the accurate Booth multiplier with one and two columns deleted from the partial product matrix. This truncation is similar to the truncation of AND gates described earlier. The comparison is shown in Fig. 38.

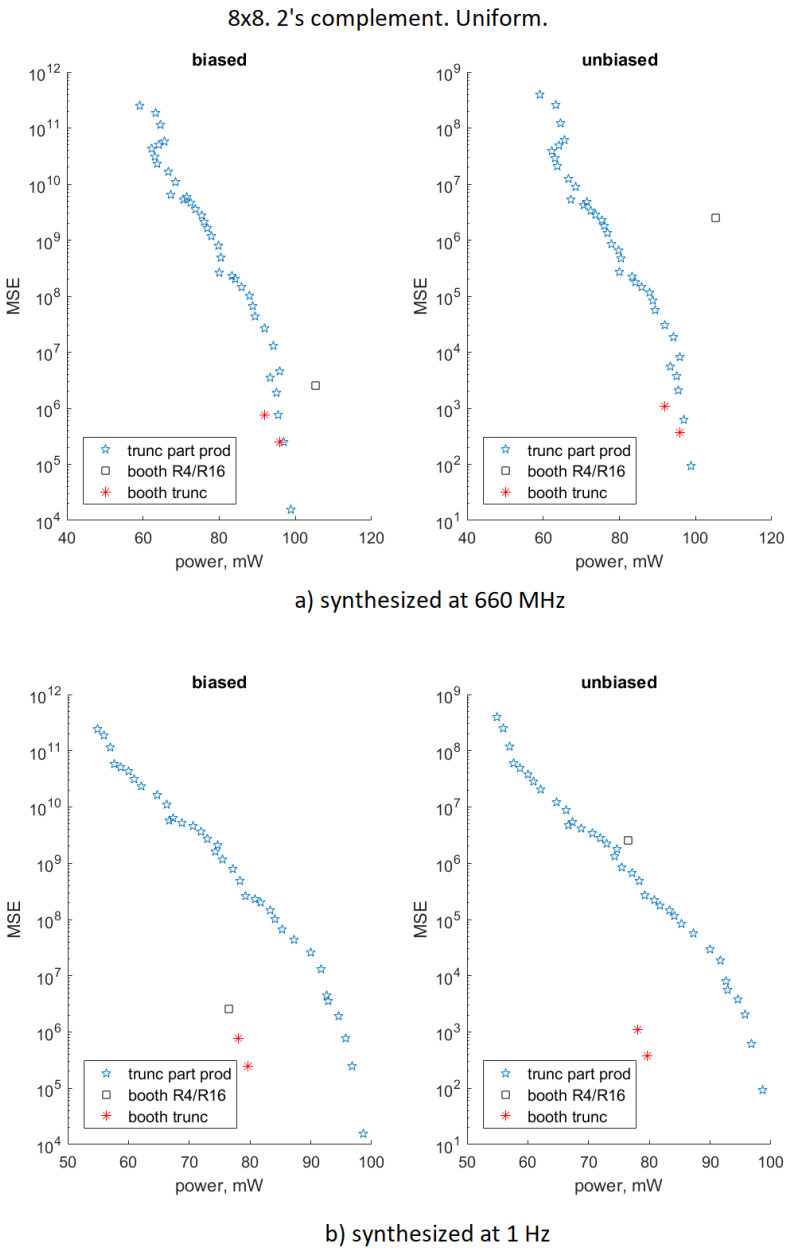


Figure 38. Approximate Booth multipliers comparison.

As can be seen in the realistic 660 MHz case, there is no advantage of using the approximate Booth multiplier compared to the Booth multiplier with truncated partial products. In the 1 Hz case, it gives one additional pareto-optimal point when the truncation is not unbiased. The approximate Booth method is unbiased initially because the non-power-of-two multiplicands are mapped to the closest power-of-two multiplicands, so sometimes the approximate result is larger and sometimes smaller allowing to have the mean error closer to zero compared to the truncation in which the result is always smaller and have a negative bias. When the truncation is unbiased, the performance of the approximate Booth becomes even worse.

One interesting observation is that truncated Booth is a bit more efficient for 660 MHz and considerably better in the 1 Hz case compared to the truncation of partial products applied to the Baugh-Wooley 2's complement matrix. It does not mean that Booth truncation (yellow markers) is better

than Baugh-Wooley truncation (blue markers) as the principle is the same in both cases. It may suggest that using the Booth multipliers is more efficient in certain conditions, presumably because the height of the partial product matrix is smaller by a factor of two, which leads to a simpler adder tree. The generation of the partial products in case of Booth multiplier is more complex, however. In the experiments the accurate Booth multiplier has a smaller area and power compared to the accurate Baugh-Wooley multiplier.

Additional experiments with larger operands have to be made to understand the tradeoff as one design point is not enough to draw conclusions.

3.2.9. Truncation of partial products against other approximate techniques.

The main purpose of this comparison is to evaluate the efficiency of simple truncation methods which are usually used for hardware optimization. As was described earlier, the truncation of partial products is more effective compared to the truncation of inputs. In this section the truncation of partial products is compared with all other methods.

The partial products truncation inherently has a negative bias making it less competitive in the biased case. When unbiasing is used however, the efficiency is improved significantly. A typical tradeoff can be seen in Fig. 39. In the biased case there are many points which are more efficient than the truncation. Synthesis at 660 MHz, at 500 MHz, at 1 Hz and the model – all have a number of points which are better than the truncation (Figures 39 and 40). However, when the results are unbiased, the truncation results are generally better. There are occasional design points which are more effective as can be seen in Fig. 39a where one point in the unbiased case stands out. This point is not present in the model prediction and in the synthesis at 1 Hz, so it may be valid only for this particular frequency but not in general.

The tradeoff of Fig. 39 is typical across all experiments except one case that should be mentioned. In the signed-magnitude representation with the Stefcal_ET distribution, the truncation of partial products is significantly less efficient than usage of recursive approximate multipliers (Fig. 41). This distribution has both inputs centered at zero with only a small fraction of numbers with a large magnitude. The truncation of least-significant bits leads to a large error and the hardware corresponding to the most significant bits is used only occasionally. As this issue is visible in the 1 Hz case and in the model prediction, this situation seems to be general for this distribution.

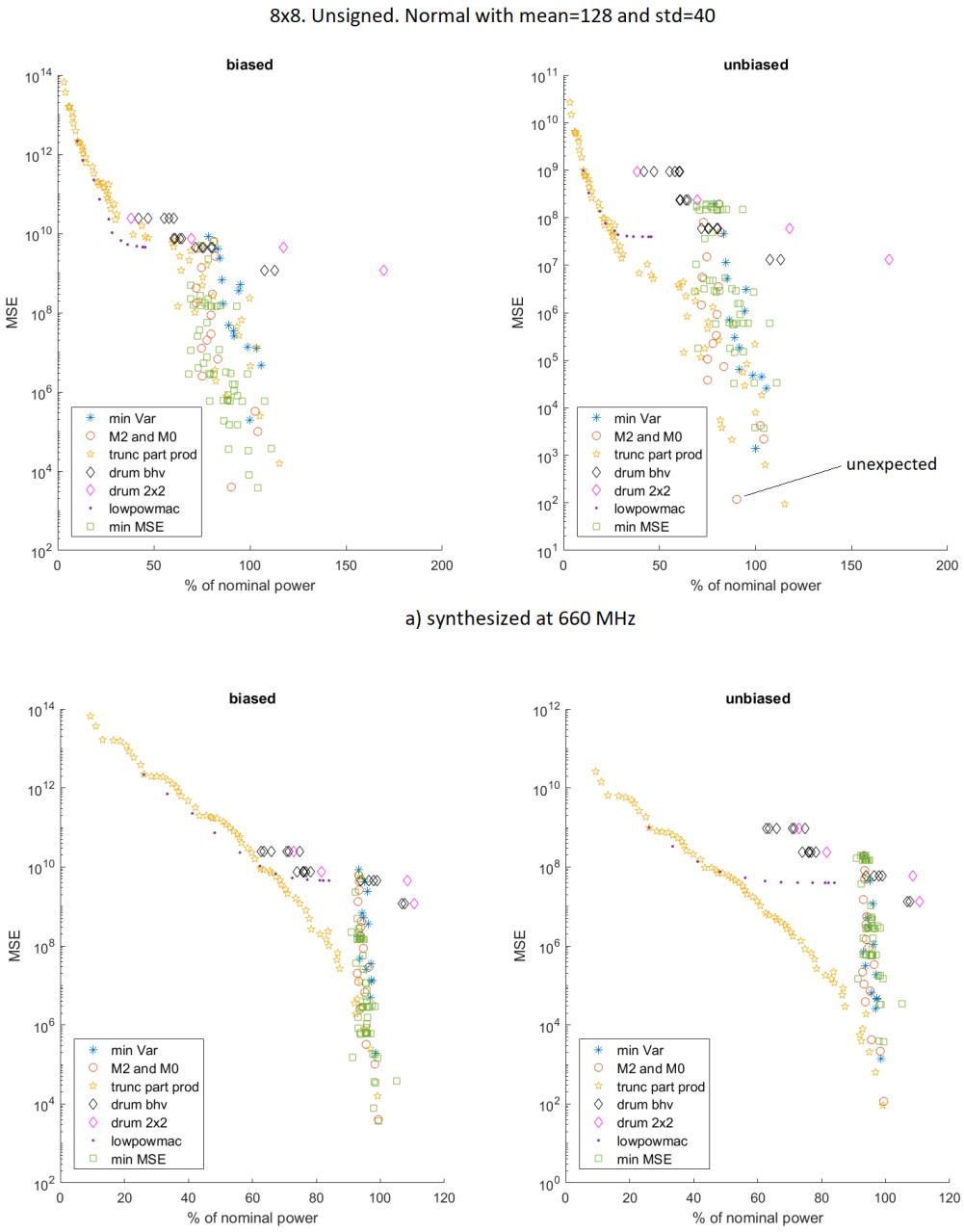
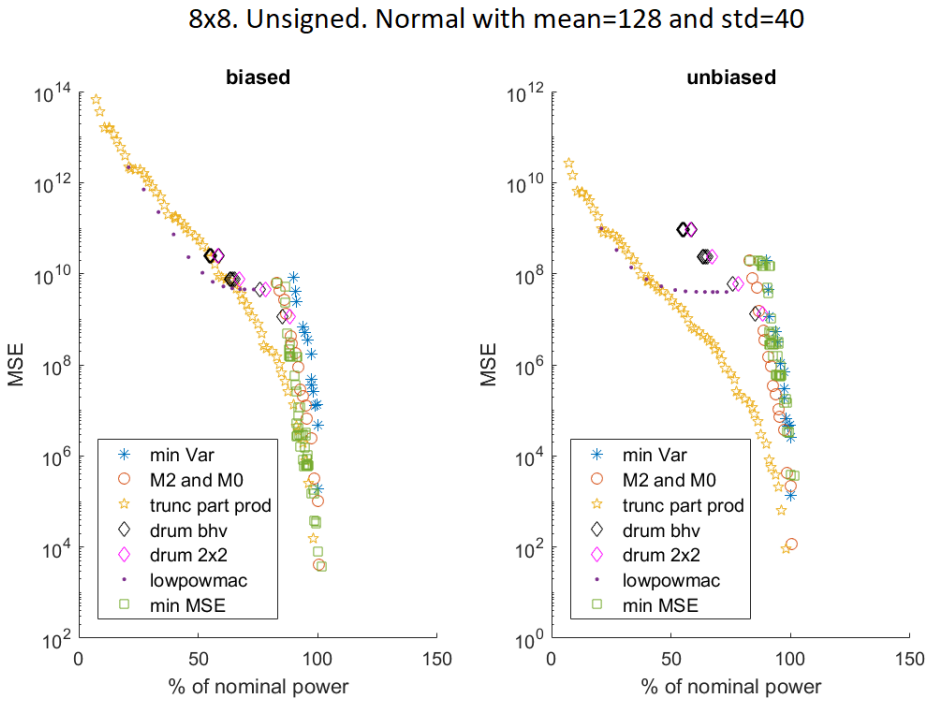
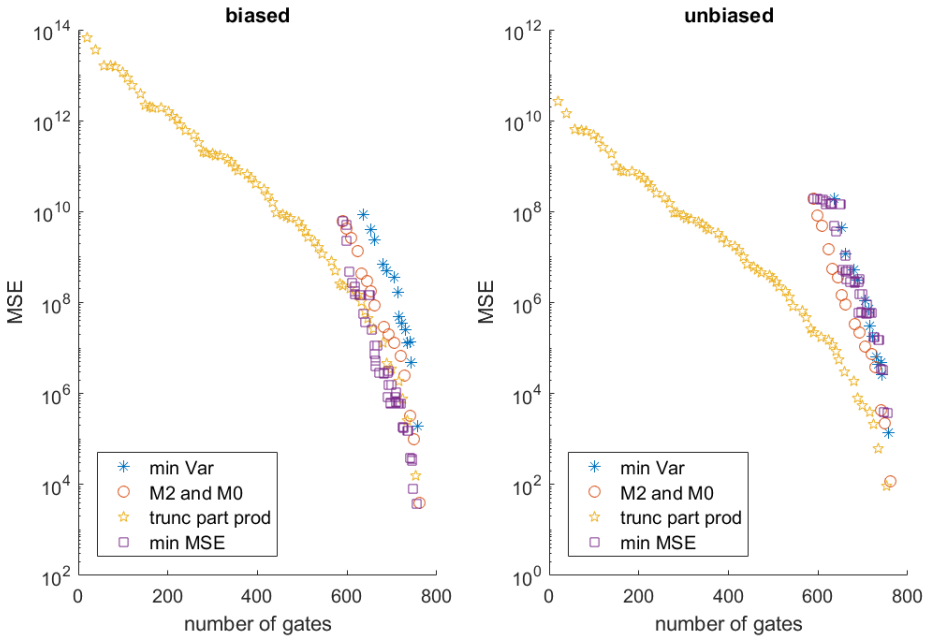


Figure 39. Truncation of partial products compared with other approximations.



a) synthesized at 1Hz



b) model

Figure 40. Truncation of partial products compared with other approximations. 1Hz and model results.

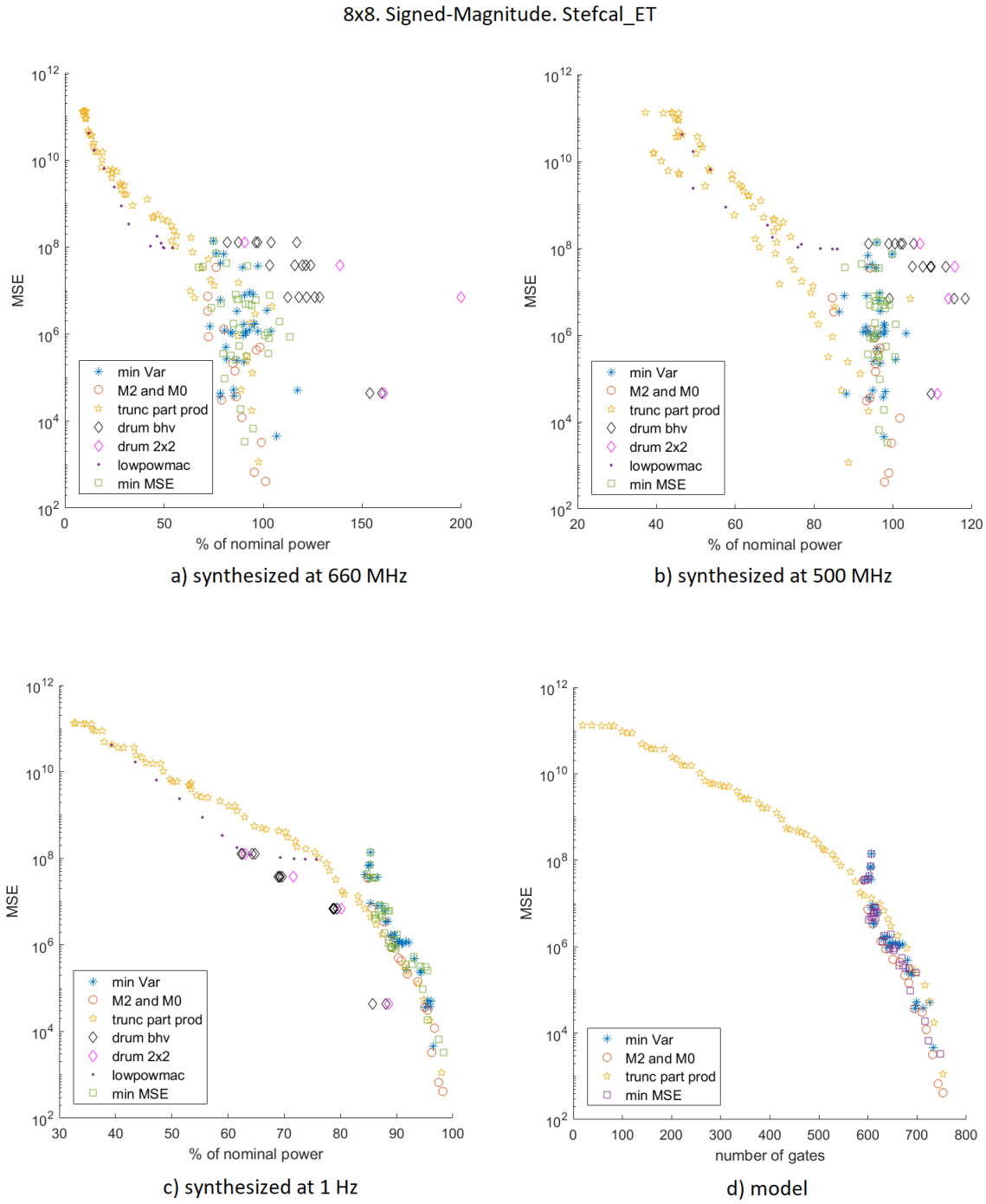


Figure 41. Truncation of partial products applied to Stefcal distribution in signed-magnitude representation.

Examination of combinations shows that all of them have the M2-type at the most significant position. As an example, one of the pareto-optimal points has the following combination: 2000_0000_0000_0002. Both the most significant and least significant blocks are approximated. It turns out that the most significant block can be made approximate without introducing any error as the error case (3x3=7) is not present in this distribution. This error case corresponds to a situation when both inputs have 1 at the MSB. Both inputs individually can have 1 at the MSB in this distribution, but they are never equal to 1 at the same time. Interestingly, if the same distribution is represented using the 2's complement representation, the truncation of partial products becomes the best method again as in this representation negative numbers are represented by using the most significant bits as

well and the corresponding hardware cannot be made approximate without introducing a large error (Fig. 42).

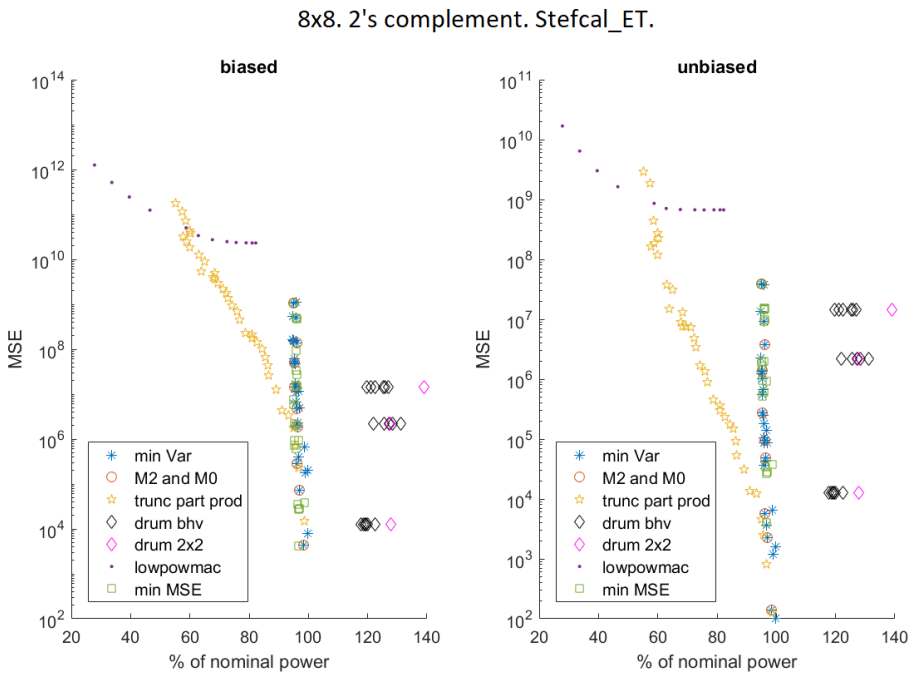
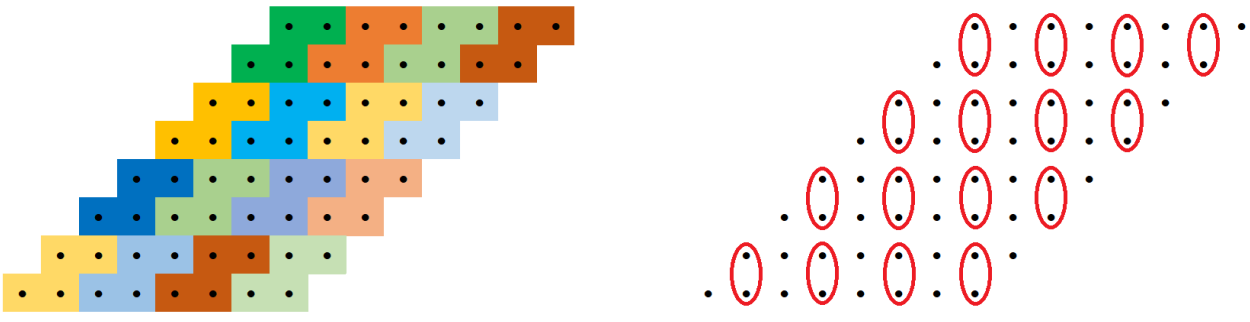


Figure 42. Truncation of partial products applied to Stefcal distribution in 2's complement representation.

3.2.10. Effectiveness of using 2x2 multipliers.

As was discussed in section XX, the usage of 2x2 multipliers can lead to a non-optimal adder tree. To understand this effect, another experiment was made. As the M2 type is simply an OR-gate placed after two partial products (AND-gates), exactly the same behavior can be implemented without using 2x2 multipliers. Two tradeoffs were implemented for the comparison. In case of 2x2 multipliers, an accurate 8x8 multiplier consists of sixteen 2x2 blocks. In the tradeoff these blocks are made of the M2 type one by one, starting from an accurate 8x8 multiplier where all the blocks are of the type M0, and gradually moving towards the maximum approximation where all the blocks are of the type M2. An equivalent tradeoff is generated without using 2x2 multipliers, by placing OR gates at the same places in the partial product matrix (PPM) (Fig. 43). The OR gates are depicted as ellipses.



a) partial product matrix consisting of 2x2 multipliers

b) partial product matrix consisting of AND gates

Figure 43. OR compression applied to 2x2 multipliers (a) and directly to the partial product matrix (b).

The comparison results are shown in Fig. 44. For each recursive multiplier (blue marker) there is a corresponding equivalent usual multiplier (red marker) with OR gates placed at the corresponding places of its PPM. The error for each pair is the same, only the power differs. The 500 MHz results show that the multipliers consisting of 2x2 blocks consume less power, which suggests that the synthesis tool is able to optimize the structure and avoid the mentioned non-optimal adder tree. However, the reason why the corresponding designs with OR gates in the PPM perform worse is not clear. This is presumably a flaw of the synthesis tool as if it is able to find a certain optimization for an 8x8 multiplier consisting of the M0 and M2 types, then exactly the same optimization can be applied to the PPM as well, because the 2x2 way of constructing multipliers can be seen as a subset of the PPM constructed of AND gates. Each 2x2 multiplier has four AND gates (dots) in their structure, corresponding to the same four AND gates in the PPM. Then, in the 2x2 case these four dots are compressed to one row. The corresponding PPM can be compressed in the same way, or in a more effective way, so it always includes the 2x2 way of compressing. The results for 1 Hz are the opposite.

One limitation of using the M2 type which is not mentioned in the reviewed literature becomes evident when looking at Fig. 43. Only the even columns are approximated by the OR gates, the odd columns remain accurate. The 16 OR gates can be placed in a more optimal way (Fig. 44), and more approximation is enabled as the most significant columns can be approximated as well. The M2 type allows to place only 16 OR gates in the multiplier, but it is possible to place 12 more OR gates. Considering that the M2 type is the most effective among the considered 2x2 multipliers, this raises a question of whether using 2x2 multipliers and its combinations is an efficient way of approximation.

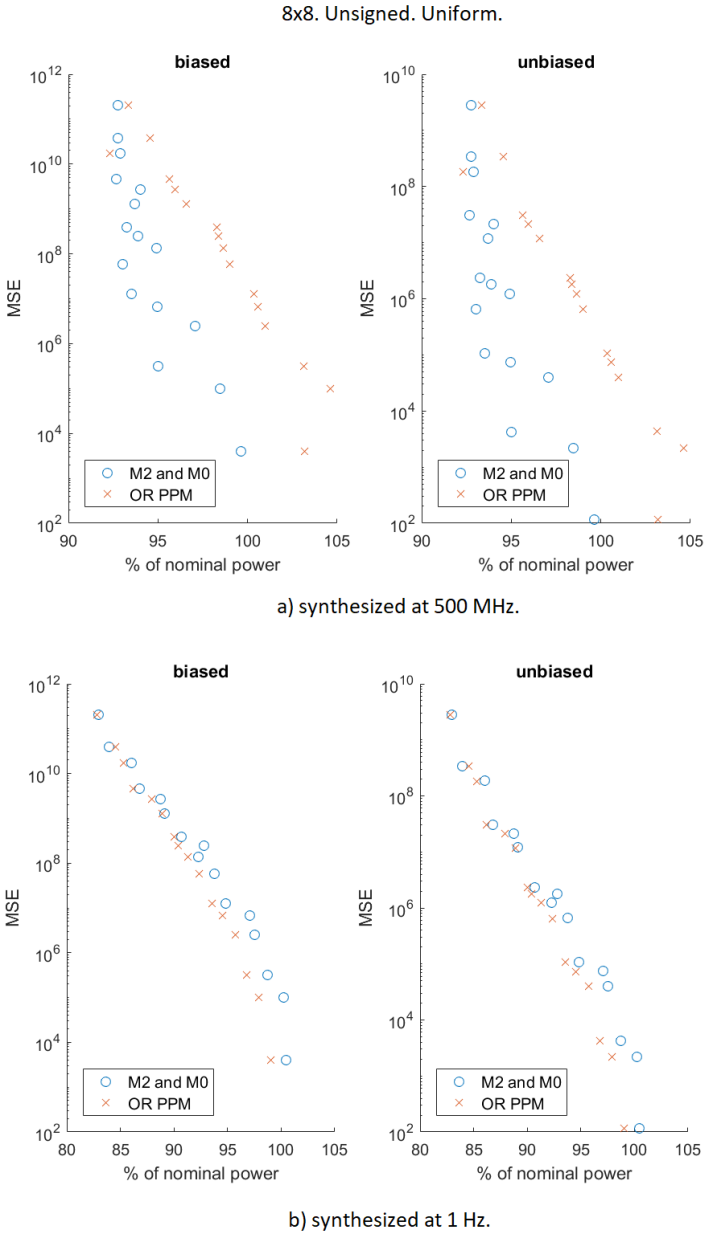


Figure 44. M2 type against the OR partial product matrix (PPM) compression.

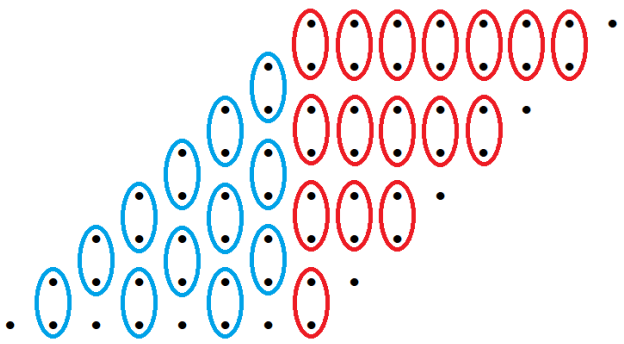


Figure 45. A more effective way of using OR gates compared to the M2 type.

3.2.11. Additional experiments for the 16x16 case.

The 8x8 case is not sufficient to understand the behavior of the DRUM and the hybrid Booth approximate multipliers. Additional experiments were made where the operands have the size of 16 bits. The 2x2 multipliers are not used in this comparison, so all approximations are applied directly to the partial product matrix. Overall, for this comparison 6 methods were used which are considered next.

Reference accurate multiplier

The reference accurate multiplier is implemented by using AND gates to form the 256 partial products. The partial products are combined into 16 rows which are summed by an accurate adder tree. The adder tree implementation is described in a behavioral way such that its implementation is chosen by the synthesis tool.

Truncation of inputs

In this method, the 16-bits inputs are gradually truncated such that the maximum approximation has only one-bit operands. The corresponding AND gates are removed, leading to a simplification of the required adder tree.

Truncation of partial products

The 31 columns of the PPM are gradually truncated column by column such that in the end there is only one column with the AND gate corresponding to the two most significant bits of the operands.

OR-compression of partial products

2x2 blocks are not used in this comparison. However, their performance can be predicted by using OR gates directly on the PPM. As the M2 type is the most effective, and it uses the OR gates in a non-optimal way, it can be expected that the usage of 2x2 combinations will not be better than the optimal OR compression. The tradeoff is generated by "OR-ing" the PPM column by column in the way shown in Fig. 45.

DRUM

The DRUM tradeoff is generated by using smaller multipliers ranging from 15x15 to 3x3. These smaller multipliers are implemented in the same way as the reference one, but with smaller operands.

Approximate hybrid encoding Booth method

In the 8x8 case there was only one possible design, radix-4/radix-16 hybrid encoding. In the 16x16 case, three approximate options are used. The encodings R4/R64, R4/R256 and R4/R1024 reduce the height of the PPM from 8 to 6, 5 or 4 dots respectively, introducing more and more errors.

Truncation of the Booth partial product matrix

This method is added mainly for the comparison with the approximate hybrid Booth method. The PPM partial products are deleted column by column. Due to the difficulty of designing, this tradeoff stops at the truncation of 12 columns, but it is enough to observe the trend.

The results for the unsigned representation with the uniform distribution are shown in Fig. 46. The best method for both the biased and unbiased cases is the truncation of partial products, the Pareto front consists completely of this method. The DRUM method with the 13x13, 14x14 and 15x15 smaller multipliers consumes more power than the reference architecture and becomes useful only starting

from the 12x12 size. As this method is the only one which is unbiased initially, in the biased case it performs better than the truncation of inputs. All the other methods have a negative bias as their results are always smaller than the accurate ones. After unbiasing the DRUM curve stays at the same place, whereas the other curves are shifted down. The OR-compression is not effective as it is dominated by the truncation of partial products. Starting from about 82% it is dominated by all other methods. This result suggests that the 2x2 combinations might be not effective as well. The maximum saving which can be achieved by using OR gates is around 23%. This number is not achievable by any of the 2x2 combinations, as the maximum power saving in the 2x2 method is achieved when all the 2x2 blocks are of the M2 type, which uses a smaller number of OR gates not optimally.

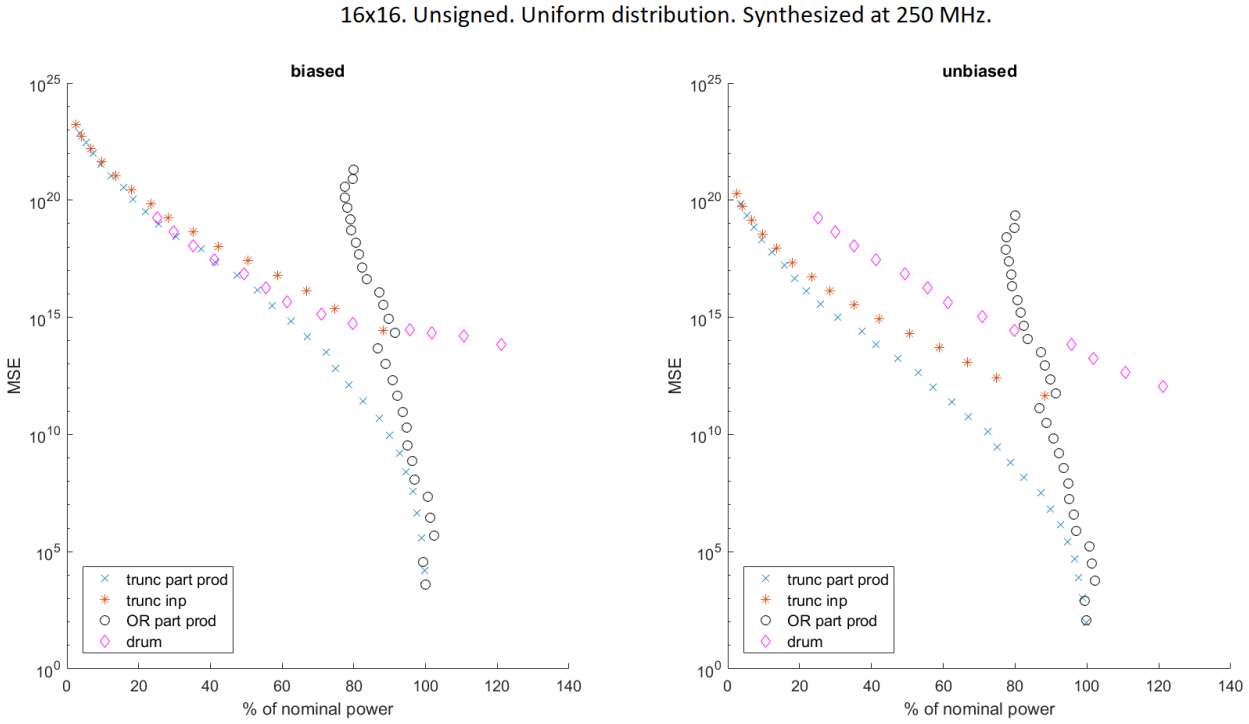


Figure 46. 16x16 experiment for the unsigned representation.

The results for the 2's complement representation with the uniform distribution are shown in Fig. 47. The Booth hybrid encoding method does not show an advantage compared to the truncation methods. Interestingly, the truncation of the Booth PPM is more effective than the usual partial product truncation. The accurate Booth multiplier consumes 0.9334 mW, whereas the refence design described above consumes 1.008 mW, which might suggest that the Booth multiplier is more power efficient. Another interesting observation is that the truncation of inputs in the biased case performs better than the truncation of partial products. This is possibly because in the 2's complement matrix some partial products are inverted. When the inputs are truncated, they become equal to 1. Whereas when the partial products are truncated, these terms become equal to 0. This allows to reduce the negative bias of the truncation of inputs. In the unbiased case these gains are lost.

These experiments were made for the same data representations (unsigned, signed-magnitude, 2's complement) and the same distributions (uniform, normal, Stefcal) as for the 8x8 case. However, the behaviour in these cases is the same as in the two figures presented here. The plots can be found in the appendix.

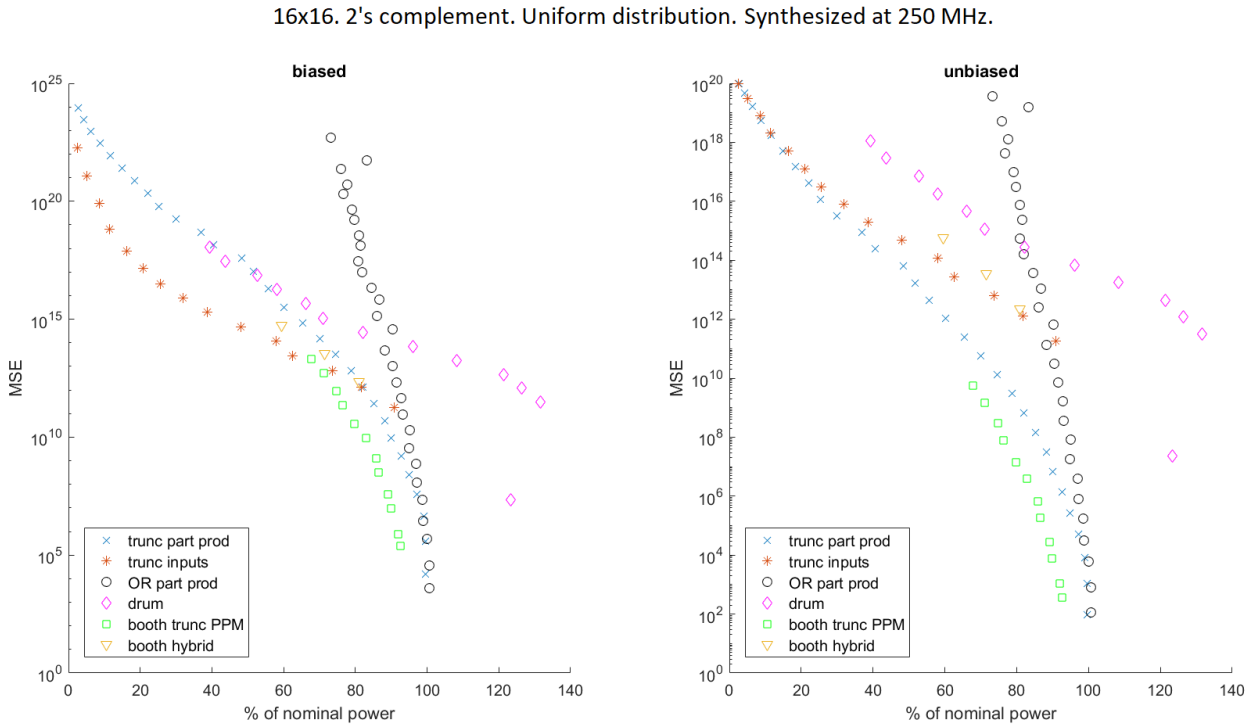


Figure 47. 16x16 experiment for the 2's complement representation.

3.3. Comparison conclusions.

Overall, the truncation methods perform well compared to the other novel methods. For certain distributions other methods can be more efficient, especially because the truncation methods have a negative bias, but in general they are among the best techniques as can be concluded from the experiments. One of important advantages is that they have a large tradeoff range compared to other methods. For example, the largest approximation for the recursive multipliers is achieved by making all the blocks to be approximate, like the 2222_2222_2222_2222 combination. If a larger approximation is desirable, it has to be combined with some other approximation. On the other hand, the truncation of partial products starts with minimal approximation (one AND gate is removed) and ends with the largest approximation (only one most significant AND gate is used) covering the full possible range of approximations and allowing to save from 0% to 100% of the power and area.

However, for different distributions, data representations, frequencies and compilation strategies various approximation methods can be advantageous. It should be noted that the truncation of inputs and the truncation of partial products is not adjusted to the distributions in these experiments, whereas the 2x2 recursive multipliers can be adjusted by using different combinations, which may explain their better performance in certain cases. It is not clear how the truncation of inputs can be adjusted to a certain distribution. But the truncation of partial products is possible to adjust by deleting the AND gates in a certain order depending on the expected errors produced by these gates for a given distribution.

4. Least-squares approximation applied to radio astronomy calibration.

4.1. Radio astronomy calibration.

In radio telescope arrays, the gain of the main beam of each telescope and the phase difference between the telescopes have to be estimated to enhance the quality of astronomical sky images. This process is called gain calibration [13]. The gains combine the effects of atmospheric disturbances, telescope geometry, receiver characteristics and so on. Atmospheric disturbances can vary within minutes, hence the calibration of these arrays (like Square Kilometer Array) have to be done online. This process is computationally expensive and consumes a lot of energy.

Gain calibration can be performed by observing a known bright source in the sky for which a matrix of model visibilities M is known. Matrix V is a measured signal of this source. To estimate the gains, the difference (15) has to be minimized by finding the matrix of gains G

$$\|V - GMG\|^2 \tag{15}$$

$G = diag(g)$ – complex antenna gains. In this work, the gains are 124 complex numbers, so the matrix G has size 124x124 with gains on the diagonal and zeros at other entries.

V – measured visibilities, a matrix with size 124x124x4 as there are four channels.

M – model visibilities, size is the same as V, 124x124x4.

Finding the optimal solution to (15) with these large matrices is a difficult problem and it is typically solved by heuristics. One of the algorithms performing the calibration process is called StefCal (statistically efficient and fast calibration) [13]. In the Stefcal calibration algorithm, it is solved by iteratively solving a least squares problem. The algorithm starts with an initial guess for the 124 gains $g^{[0]}$. To compute the first gain ($p = 1$), all elements of g are multiplied by elements of the first column of the matrix M element-wise (16) to compute vector z.

$$z = M_{:,p} \circ g^{[i-1]} \tag{16}$$

Then, a linear least squares problem is solved to find a scaling of z such that it is as close as possible to the first column of V (17) to find $g_1^{[1]}$

$$g_p^{[i]} = \frac{V_{:,p}^H \cdot z}{z^H \cdot z} \tag{17}$$

The symbol H denotes the Hermitian transpose, which means that the vector is transposed, and the complex conjugate of each element is taken. This process is repeated 124 times to get the vector $g^{[1]}$ which is then used in the next iteration to compute a better estimate. The algorithm stops when the improvement between iterations is small enough, i.e., when a convergence criterion is reached. In Stefcal the convergence criterion is the norm of the difference of two solution vectors divided by the norm of the current solution vector (18). Usually this process takes around 100 iterations.

$$\frac{\left\|g^{[i]} - g^{[i-1]}\right\|}{\left\|g^{[i]}\right\|} \leq 10^{-6} \tag{18}$$

The algorithm can be divided into four stages: element-wise product, multiply-accumulate, square-accumulate and division (Fig. 48).

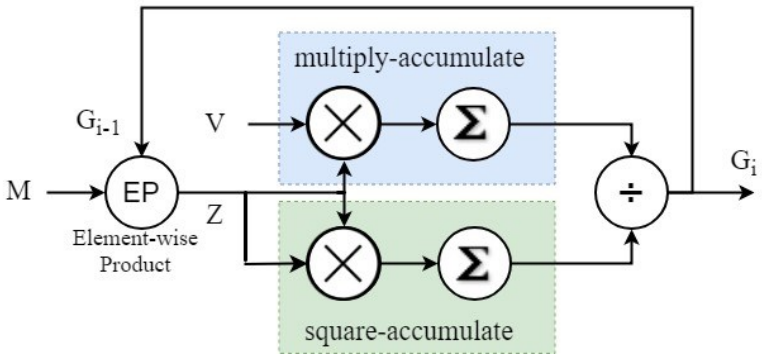


Figure 48. Least-squares calibration block diagram.

Multiplication of two complex numbers requires four multiplications, one addition and one subtraction.

$$(a + ib)(c + id) = ac - bd + i(ad + bc) \tag{19}$$

A datapath to which the algorithm can be mapped is shown in Fig. 49. To compute one gain in the current iteration, the first element of $g = a + ib$ is multiplied by the corresponding element from a column of matrix M, $m = c + id$. This number $z = e + if$ is then multiplied by the corresponding element from matrix V, $v = h + it$, the real and imaginary parts are stored in two registers. Also, $z = e + if$ is multiplied by the complex conjugate of itself, which is equal to $e^2 + f^2$, and the result is stored in one register as the imaginary part is eliminated. This process is repeated 496 times as there are 124 gains and 4 channels, and the running sum is stored in the registers. After the accumulation is complete, the numbers stored in real and imaginary registers of the MAC part are divided by the real number computed by the SAC part to obtain the gain for next iteration. To compute the next gain, 496 computations are done again with the next columns of M and V and the current vector g. The gains in the current iteration can be computed in parallel by 124 structures as in Fig. 49, or all the gains can be computed in a serial way by one such structure.

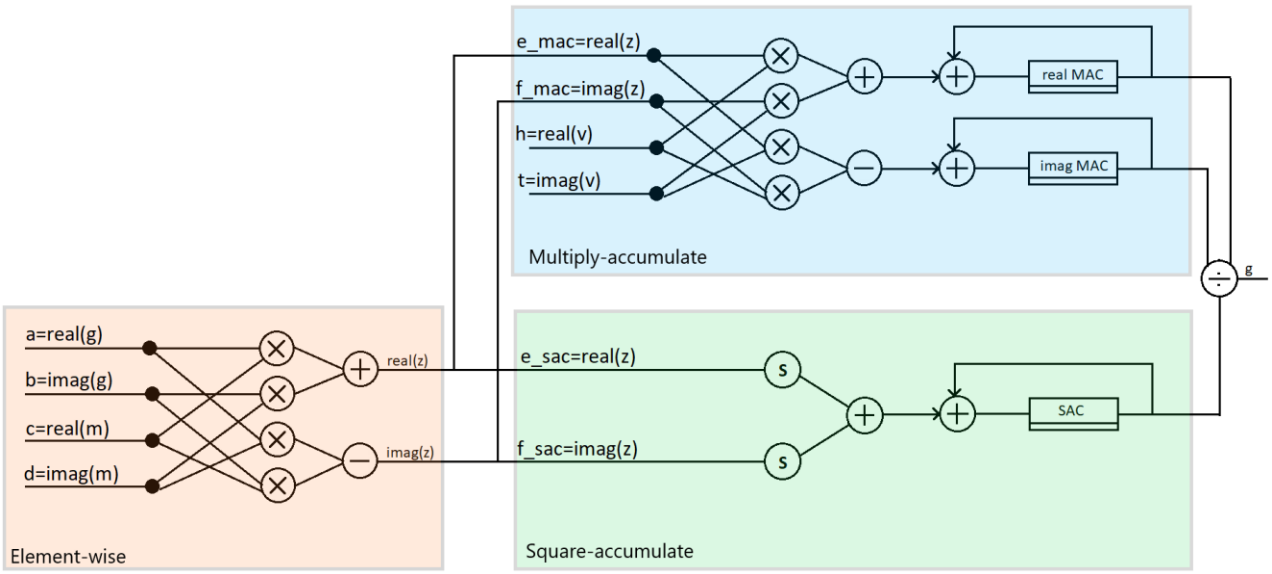


Figure 49. Datapath for Stefcal algorithm.

4.2. Floating-point to fixed-point conversion.

As the algorithm is given in floating-point double precision format, it has to be converted to fixed-point format as fixed-point hardware is more efficient in terms of area and power. This process consists of two parts. First, integer parts of each signal are determined, i.e., how many bits are needed to avoid overflow. Secondly, the required fractional parts are determined to satisfy precision requirements of the algorithm. Integer part of this process is trivial as it requires observing the ranges of all signals from which the number of integer bits for each signal can be determined. Optimal selection of the number of fractional bits for all signals is a difficult process and it can be done by using heuristics.

In this work, an approach described in [26] is used. In this approach, all signals are kept in floating-point format except one which is converted into fixed-point. The minimum fractional length of this signal which satisfies the performance is found. This process is repeated for all signals to get the minimum fractional length for each signal. Then, all signals are converted to fixed-point and if the performance is satisfied, the optimization is finished. However, typically the fractional widths have to be increased by 1-2 bits from their minimal found widths. For that, the work of [26] uses an exhaustive combinatorial algorithm which tries all possible combinations to find an optimal combination. It starts with increasing one signal by one bit, and all signals are tried. If the performance is not satisfied, it increases two signals by two bits in sum, for all possible combinations. This approach requires a large number of simulations and is only feasible if the number of signals is not very large, not more than six as the work suggests. If the number of signals is larger, the related and similar signals can be grouped to simplify the search. As the Stefcal algorithm has a large number of signals, they are grouped into six groups as shown in Table 7. All the signals can be seen directly in Fig 49 or traced from the name of the signal. For example, the eh_plus_ft signal corresponds to the output of the subtractor which computes the difference between the outputs of the two multipliers computing the eh and ft signals. Each group can have their signals untouched, increased by one bit, or increased by two bits, so there are three options for each group. As there are six groups, $3^6 = 729$ simulations are required, which is feasible for the Stefcal algorithm. The results of applying this approach to the Stefcal algorithm are provided in Table 8.

a, b
c, d
h, t
ac, bd, ad, bc, e_mac, f_mac
e_sac, f_sac, esq, fsq, esq_plus_fsq, SAC
eh, ft, et, fh, eh_plus_ft, et_minus_fh, mac_real, mac_imag

Table 7. Grouping of signals for the optimal bitwidth search.

Signal	Minimal integer length	Minimal fractional length	Optimal fractional length	Optimal lengths in WL.FL format	Non-optimal uniform WL=27
a	9	13	14	23.14	27.18
b	8	13	14	22.14	27.19
c	-9	24	25	16.25	27.36
d	-10	24	25	15.25	27.37
h	6	11	12	18.12	27.21
t	6	11	12	18.12	27.21
ac	-2	23	25	23.25	27.29
bd	-4	23	25	21.25	27.31
ad	-3	24	26	23.26	27.30
bc	-2	24	26	24.26	27.29
e_sac	-2	21	23	21.23	27.29
f_sac	-2	21	22	20.22	27.29
esq	-6	26	28	22.28	27.33
fsq	-6	26	28	22.28	27.33
esq_plus_fsq	-6	26	28	22.28	27.33
sac	1	22	23	24.23	27.26
e_mac	-2	23	25	23.25	27.29
f_mac	-2	24	26	24.26	27.29
eh	3	23	25	28.25	27.24
ft	1	23	25	26.25	27.26
et	2	24	26	28.26	27.25
fh	1	24	26	27.26	27.26
eh_plus_ft	3	23	25	28.25	27.24
et_minus_fh	2	24	26	28.26	27.25
mac_real	7	16	18	25.18	27.20
mac_imag	6	16	18	24.18	27.21

Table 8. Fixed-point optimization of Stefcal algorithm.

In this work, the performance of the Stefcal algorithm is assumed to be satisfied if the number of iterations to converge is smaller or equal to the floating-point version and the relative difference in length between the fixed-point solution vector and floating-point answer is not more than 10^{-5}, similar to [4].

$$Diff\_rel = \frac{\left\| g_{fixed} - g_{float} \right\|}{\left\| g_{float} \right\|} \leq 10^{-5} \tag{3.6}$$

The signal widths are presented in the WL.FL format which is used in the Fixed-point designer toolbox of MATLAB. In this format the integer length is equal to the word length minus the fractional length. Some of the signals in Table 8 have negative integer lengths. This means that the signals have small

values. For example, the signal d has a range of [-0.000347 : 0.000347], so all its bits are fractional, and the most significant bit has the weight of 2^{-11} (Fig. 50).

As can be seen in Table 8, the optimal word-lengths of the signals are not the same. This approach allows to significantly reduce the hardware cost compared to the uniform word-length approach (the last column in Table 8). If the uniform word-length is used, 27 bits for all signals are required. If the gate model described in the previous chapter is applied to the datapath, it requires 52137 gates. The optimization allows to reduce the gate count to 28247 gates, 46% reduction. The comparison between the floating-point and optimized fixed-point versions can be seen in Fig. 51.

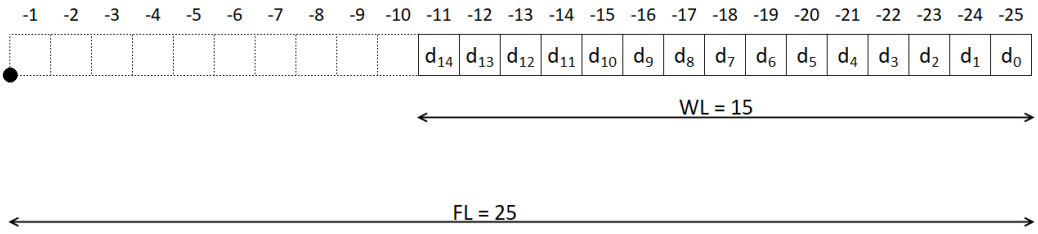


Figure 50. signal d in the WL.FL format.

On the left figure the convergence behavior is plotted. Both versions converge to 10^{-6} in 92 iterations. On the right, the difference $\|V - GMG\|$ is plotted. Both versions converge to almost the same solution.

The comparison between the computed gains is shown in Fig. 52, demonstrating identical results for the 124 complex gains.

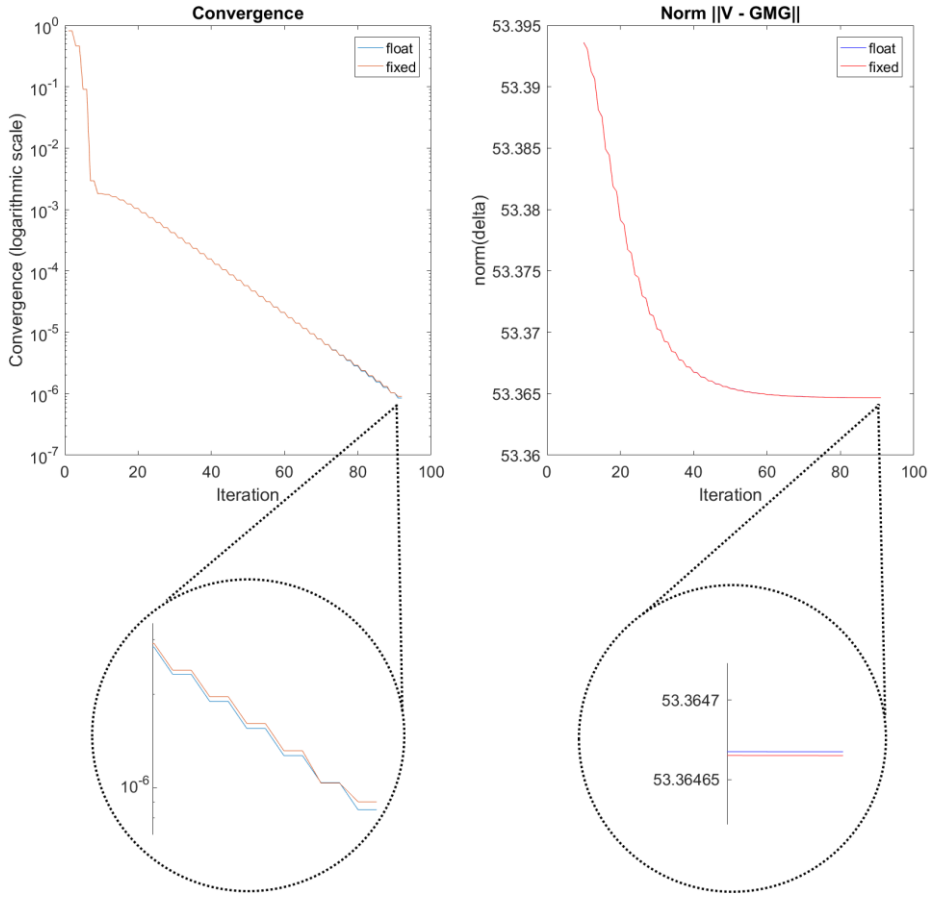


Figure 51. Floating-point and fixed-point Stefcal behavior.

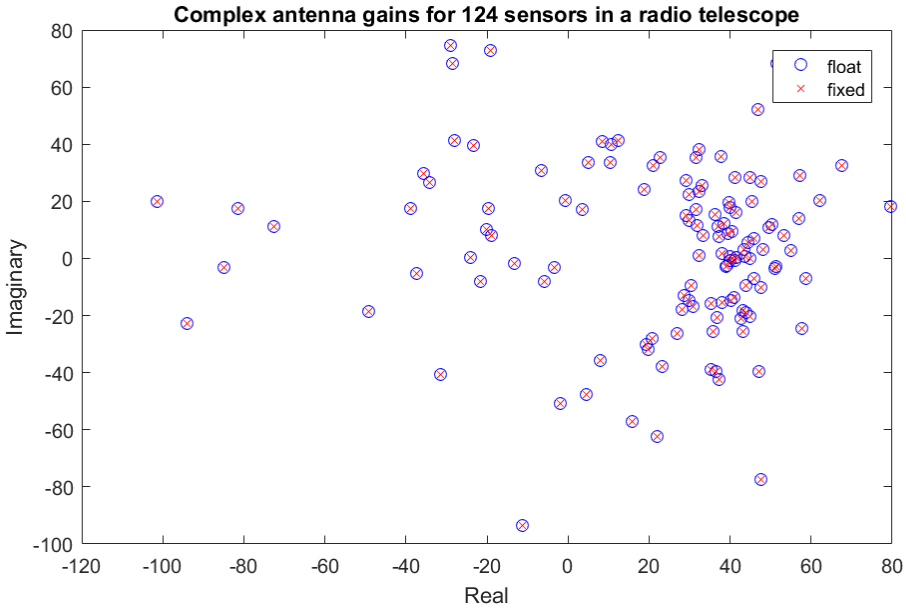


Figure 52. Gains computed by floating-point and fixed-point algorithms.

4.3. Approximation of the Stefcal algorithm.

The optimized fixed-point version of Stefcal specifies the minimum hardware requirements if exact arithmetic units are used. Simplifying it further will violate the iteration count criterion or the distance from the floating-point reference result. However, a better optimization can be achieved by using approximate arithmetic units. As demonstrated in [4], the algorithm does not require uniform precision in the course of computation. It suggests that some number of initial iterations can be mapped to a simplified less accurate hardware in order to reduce energy consumption. In this section, different approximation methods described in the previous chapter will be applied to the optimized fixed-point version of the Stefcal algorithm.

The optimized fixed-point version and all the approximations are implemented in the Fixed-Point Designer Toolbox of MATLAB. This toolbox allows to build the model of hardware and run bit-true simulations. As it directly supports only the 2's complement representation, the approximations will be applied to the 2's complement version of the algorithm. It is still possible to use this toolbox for the signed-magnitude, but it requires more work, and in general it cannot be said that the 2's complement of the signed-magnitude representations are more more power efficient [20], it depends on the particular application and it is difficult to predict.

Application of approximations to the Stefcal algorithm is not straightforward. Even in the simplest approximation (the truncation of inputs) it is a difficult task to find the optimal truncation configuration which minimizes the energy consumption. It requires to determine which signals to truncate, by how many bits, and for how many iterations. To simplify this problem, some assumptions will be made. First, the element-wise product will be kept accurate with the parameters determined in the previous section, approximations will be applied only to the MAC and SAC units. Second, two hardware units are assumed, one accurate (optimized fixed-point architecture from the previous section) and one approximate which has to be found, and to which some number of initial iterations will be mapped.

Even truncation of only one signal has to be optimized. If a signal can be truncated by 3 bits and 40 iterations or by 2 bits and 60 iterations, it is not clear which option saves more energy. The answer can be obtained by synthesizing the design and performing a power simulation. However, it would be infeasible for such a complex structure as it will require significant amount of time for design, compilation and simulation. For this reason, to analyze the effect of approximations and understand how much cost is saved for each approximation, the unit gate model is used. This gate model counts the number of gates required for each hardware configuration. If an approximation is applied in such a way that all the logic after an approximated multiplier is simplified, these effects are also included in the model. The gate model outputs a single number which corresponds to the number of gates required by the whole datapath – all multipliers, squarers, adders and registers in the design. The savings achieved are computed by the following expression:

$$cost\ saved = \ 100 \cdot \frac{\left(cost_{opt} - \ cost_{ax}\right) \cdot num_{axiter}}{cost_{opt} \cdot num_{iter}} \% \qquad (20)$$

The difference in cost between the approximate and optimal hardware is multiplied by the number of approximate iterations and divided by the total cost of the optimal fixed-point computation.

$cost_{opt}$ is a constant number which is equal to 28247, the cost of the optimal hardware.

num_{iter} is also a constant, which is equal to the number of iterations required by the floating-point algorithm.

Every approximation applied will change $cost_{ax}$ to some number smaller than 28247, and it will also change num_{axiter}, as the Stefcal algorithm can withstand different approximations for different number of iterations. There is a tradeoff between $cost_{ax}$ and num_{axiter}, as smaller $cost_{ax}$ corresponds to more approximations, leading to smaller number of iterations which can survive this approximation. Therefore, the optimization goal is to find an approximate architecture which minimizes $cost_{ax}$ and maximizes num_{axiter} in such a way that the $cost\ saved$ in (20) is maximized. Overall, the approach can be described as follows:

1. Apply an approximation.
2. Try to run the algorithm for $num_{axiter} = 90$ iterations on the approximate architecture and then switch to the optimal fixed-point architecture. If the results are not acceptable, decrease the number of approximate iterations by 10.
3. Compute the $cost\ saved$ and if there are some other approximations to apply, repeat steps 1-2.
4. Implement the architecture with the smallest cost in VHDL, synthesize the design and perform the power simulation to see the real power and area numbers.

As the unit gate model computes the area and the optimization goal is the reduction of power consumption, an assumption is made that the smallest architecture will have the smallest power. In that sense the purpose of the model is to find the minimum, the real percentage of saving may be different.

The reference design is synthesized for 50 MHz. The power is equal to 3.5530 mW. The area 27023 μm^2. The reference design is implemented without using behavioral descriptions for the multipliers and squarers in VHDL. They are implemented by constructing the partial product matrix consisting of AND gates which are then summed by an adder tree selected by the synthesis tool, and all the approximations are applied directly to the partial product matrix.

The previous chapter compared different approximations and their performance by using the MSE metric. However, this algorithm is iterative, it includes division and a large number of computations. It is difficult to predict which one is the most efficient for it. Therefore, four approximations will be applied. The truncation of partial products is the most promising and applied first. Then, the truncation of inputs is applied as it also showed good performance in the comparison and is easy to implement. The DRUM is also tried, as the input distribution in the Stefcal multipliers is similar to the normal distribution, and the DRUM can be efficient for such distributions as they have a large fraction of small numbers and only occasionally the shifting is required to multiply them. And finally the OR-compression is also implemented. It would be difficult to apply the recursive multipliers as the structure has many multipliers and they are much larger than the 4x4 and 8x8 cases considered previously. However, as the M2 type with unbiasing is among the best recursive combinations, the OR-compression can be used to predict what can be expected is the recursive multipliers are applied, as the OR-compression is using the OR gates more efficiently compared to the recursive multipliers. In all cases the approximations are applied directly to the partial product matrix, which allows to compare the effectiveness of methods.

4.3.1. Truncation of partial products

In this section, the truncation of partial products is applied to the four multipliers of the MAC unit and to the two squarers of the SAC unit. The truncation is applied in a column by column way, so the approximation step is the truncation of one column of the partial product matrix.

First, the multipliers and squarers are truncated one by one, while the other remain accurate. This way an optimal truncation number for each of the units individually is determined. Then, the multipliers and squarers are truncated in pairs, around the determined numbers. Finally, the MAC and SAC units are truncated together. This approach will be demonstrated for the truncation of partial products, and it will be applied to the other three approximations. It does not guarantee to find the optimal approximation configuration, but it allows to reduce the number of time-consuming simulations and hopefully find a near-optimal solution for each method.

SAC truncation

The squarers which compute e_sac^2 and f_sac^2 are truncated one by one first. The e_sac^2 squarer is truncated, and all the other multipliers and squarers are kept accurate. The same is repeated for the f_sac^2 squarer. The corresponding squarers can be seen in Fig. 49. The results of truncating the squarers are shown in Fig. 53. This plot shows how many columns can be truncated if only the corresponding squarer can be approximated. Truncation of 16 columns in the e_sac^2 squarer allows to save around 1.7%.

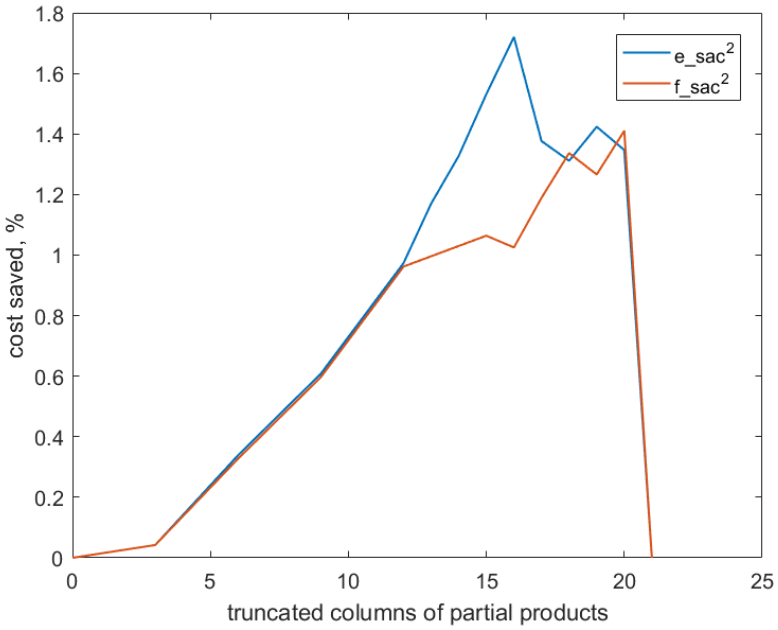


Figure 53. Truncation of squarers.

The next step is to truncate both squarers at the same time. For that, in the e_sac^2 squarer the number of truncated columns ranges from 14 to 20, and in the f_sac^2 the number of truncated columns ranges from 17 to 20. Therefore, 28 combinations are simulated, 7 for e_sac^2, multiplied by 4 for f_sac^2. The results are shown in Fig. 54. It can be concluded that the most effective way of truncating the squarers is to truncate 20 columns in the e_sac^2 squarer and 18 columns in the f_sac^2 squarer.

It allows to save about 3% of cost. The number of approximate iterations is not shown in these plots. In this case, the Stefcal survives these approximations if they are applied for 60 iterations.

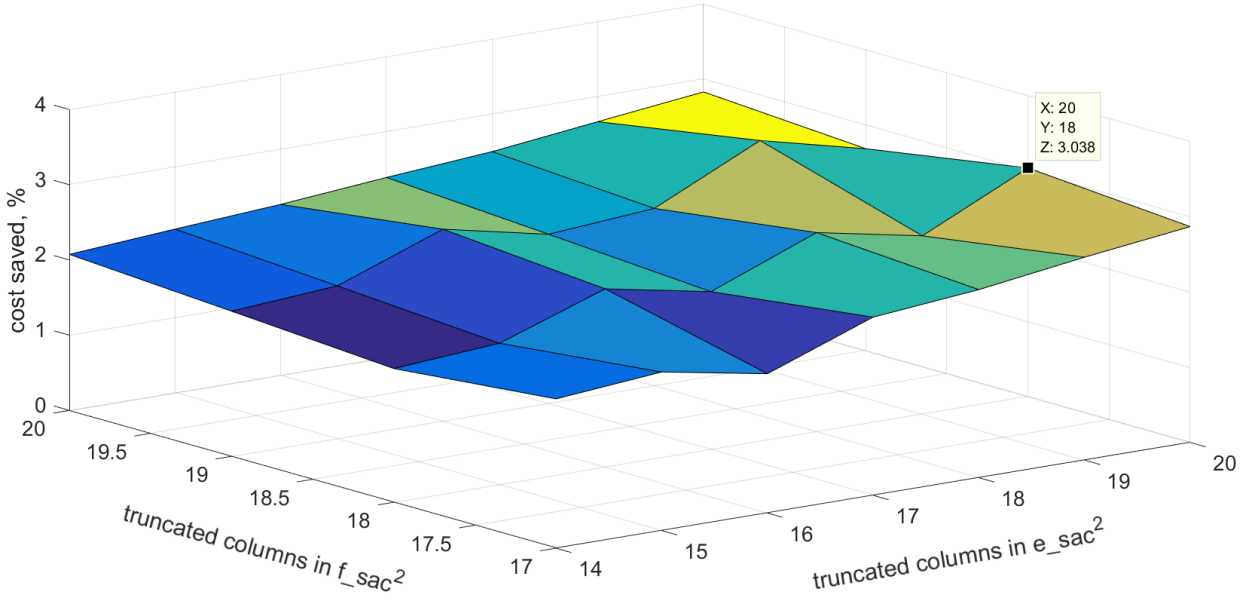


Figure 54. Truncation of columns in the two squarers.

MAC truncation

In a similar way, the four multipliers eh, ft, et, fh of the MAC unit are truncated individually. The results are shown in Fig. 55.

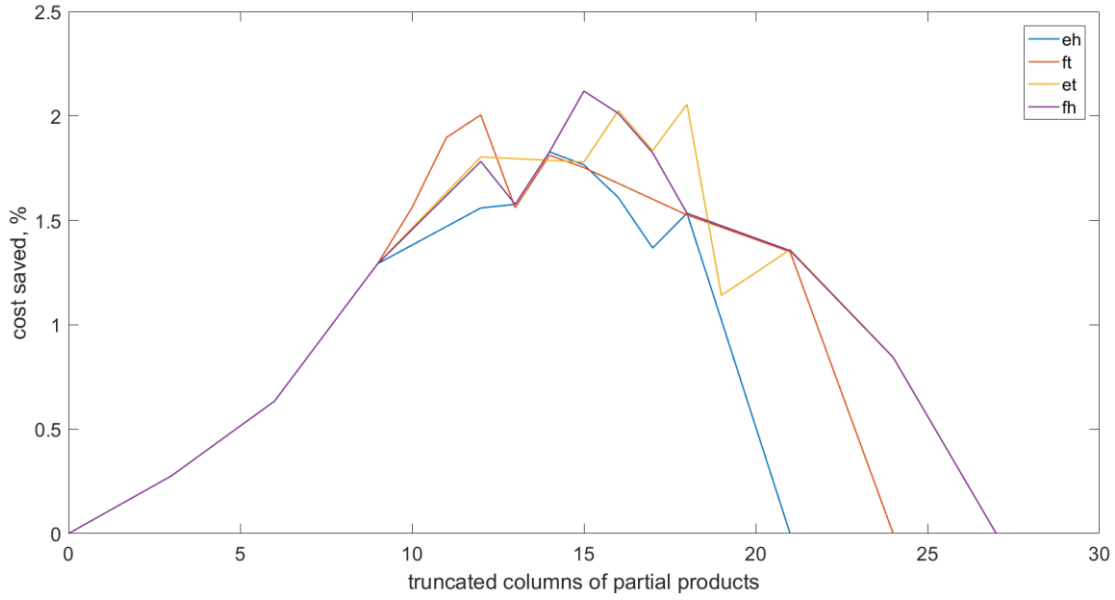


Figure 55. Truncation of multipliers in the MAC unit.

In the next step, the two multipliers eh and ft are truncated together. In the eh multiplier 13, 14, 15 and 16 columns are truncated, and the corresponding range for the ft multiplier is from 11 to 15.

Therefore, 20 simulations are performed, and the optimal numbers of truncated columns are 13 for eh and 14 for ft for 60 iterations, allowing to save 3.4% of cost. The same process is repeated for the pair et and fh, truncating from 14 to 18 columns in each multiplier. The optimal parameters are 15 for et and 16 for fh.

Then, all four multipliers are truncated. The starting parameters are the ones determined for each pair. To test more configurations, the pairs are deviated from the found parameters. Deviation Δ_1 is applied to one pair, and deviation Δ_2 to another: $(eh = 13 + \Delta_1, ft = 14 + \Delta_1)$ and $(et = 15 + \Delta_2, fh = 16 + \Delta_2)$.

Both Δ_1 and Δ_2 range from -2 to 2. Therefore, 25 simulations are performed, and the following parameters are found for the whole MAC unit: $eh = 13, ft = 14, et = 15, fh = 16$. So, the starting parameters are the most effective. The Stefcal survives this approximation of the MAC unit for 60 iterations, allowing to save 8% of the cost in gates of the unit gate model.

SAC and MAC truncation

At this final step, both the MAC and SAC units are truncated. The starting parameters are the ones determined for the MAC and SAC individually, and again the deviations Δ_1 and Δ_2 ranging from -2 to 2 are applied. The found parameters are summarized in Table 8. The Stefcal is able to withstand the truncation of partial products with these parameters for 51 iterations, allowing to save 10.7% of the reference cost.

Multiplier (or squarer)	**# of truncated columns**
e_sac^2	21
f_sac^2	19
eh	14
ft	15
et	16
fh	17

Table 8. Truncation of partial products. Determined parameters.

Unbiasing

The truncated architecture has a negative bias as all the deleted partial products are equal to zero. The unbiasing is performed by placing three different initial values in each of the three accumulators. These values are determined by performing two simulations in parallel, one with the optimized accurate structure, and one with the approximate structure. The mean of the differences between the corresponding values is computed for each accumulator and the value is added to the corresponding accumulator. The effect of unbiasing is that the Stefcal is able to withstand the truncation of partial products for 61 iterations, ten iterations more compared to the biased case, and save 12.8% instead of 10.7%. The effect of the applied approximation on the convergence process and solution can be seen in Fig. 56. The behavior until iteration 51 (61 in the unbiased case) deviates from the floating-point version. After switching from the truncated approximate version to the optimal fixed-point version, the solution converges to the same value in the same number of iterations.

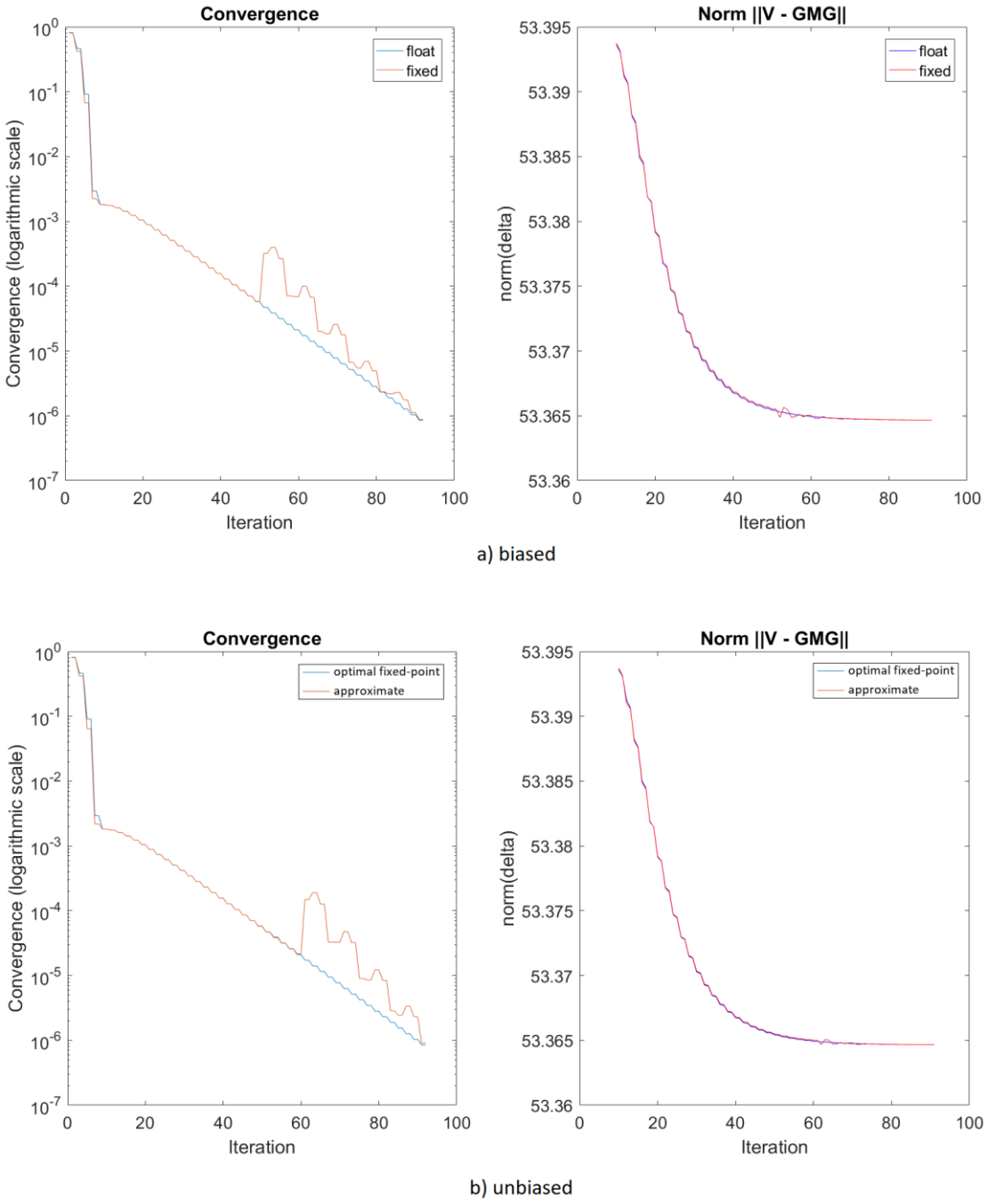


Figure 56. Effect of partial product truncation for biased and unbiased cases.

Synthesis

The approximate design is synthesized, and the power is estimated. The power is equal to 2.2280 mW (the reference architecture has the power of 3.5530 mW). The area is equal to 22523 μm^2 (the reference architecture has the area of 27023 μm^2. Overall, by increasing the area by 83%, 20% of energy can be saved in the biased case, and 24.7% if the unbiasing is enabled.

4.3.2. Truncation of inputs

The approach for the approximation of inputs is identical to the one described for the truncation of partial products, so the full process is not described here, only the determined parameters are

reported. The parameters are shown in Table 9. As can be seen, the signals h and t remain its original bit width. This can be explained by the fact that these signals are already the shortest in the optimal fixed-point architecture. The number of approximate iterations for this approximation is equal to 52 iterations.

Signal	# of bits truncated
e_sac	8
f_sac	8
e_mac	8
f_mac	12
h	0
t	0

Table 9. Truncation of inputs. Determined parameters.

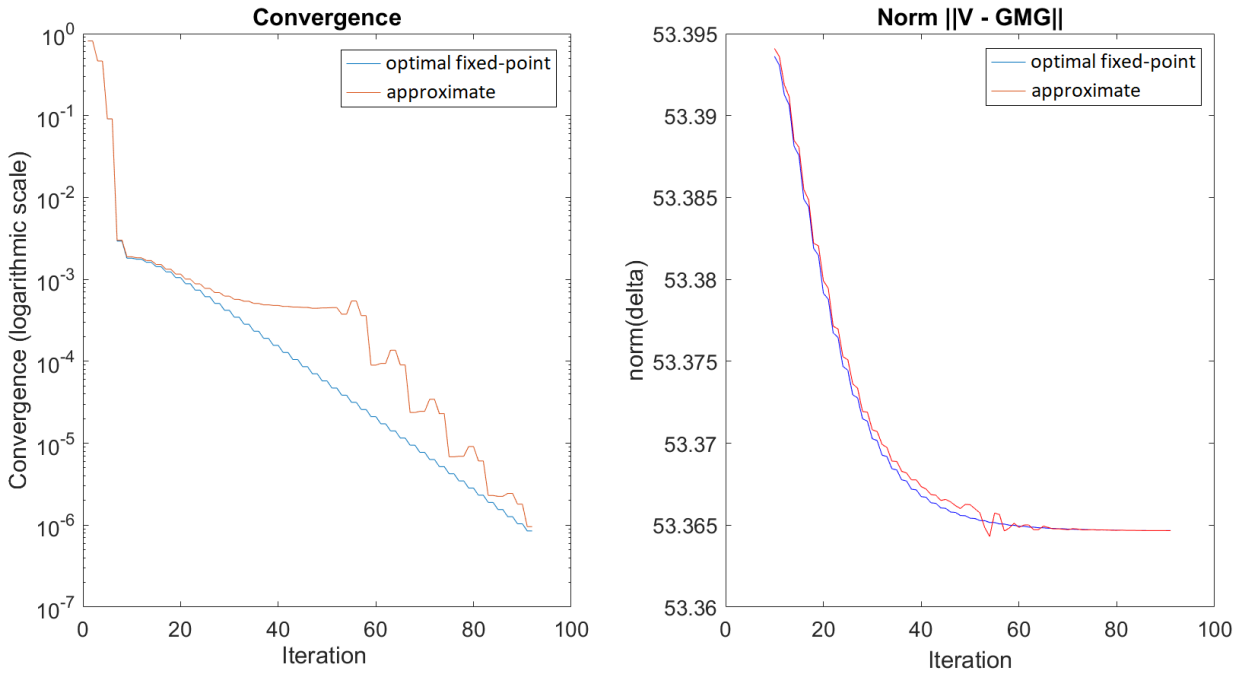
Unbiasing

The unbiasing by the accumulator initializations turns out to be ineffective for this method. Both the number of iterations becomes larger and $Diff\_rel > 10^{-5}$. However, if the unbiasing is done by setting the least significant bits of the truncated signals to 1, similar to what is done in the DRUM multiplier, the performance is significantly improved, allowing to use the approximation for 64 iterations, 12 more than in the biased case. This unbiasing is not a constant term added to all the results, it is supposed to be more effective as for each multiplication the unbiasing depends on the operands. The LSB of one operand which is set to 1 is multiplied by all the bits of another operand, which are differing for various operands.

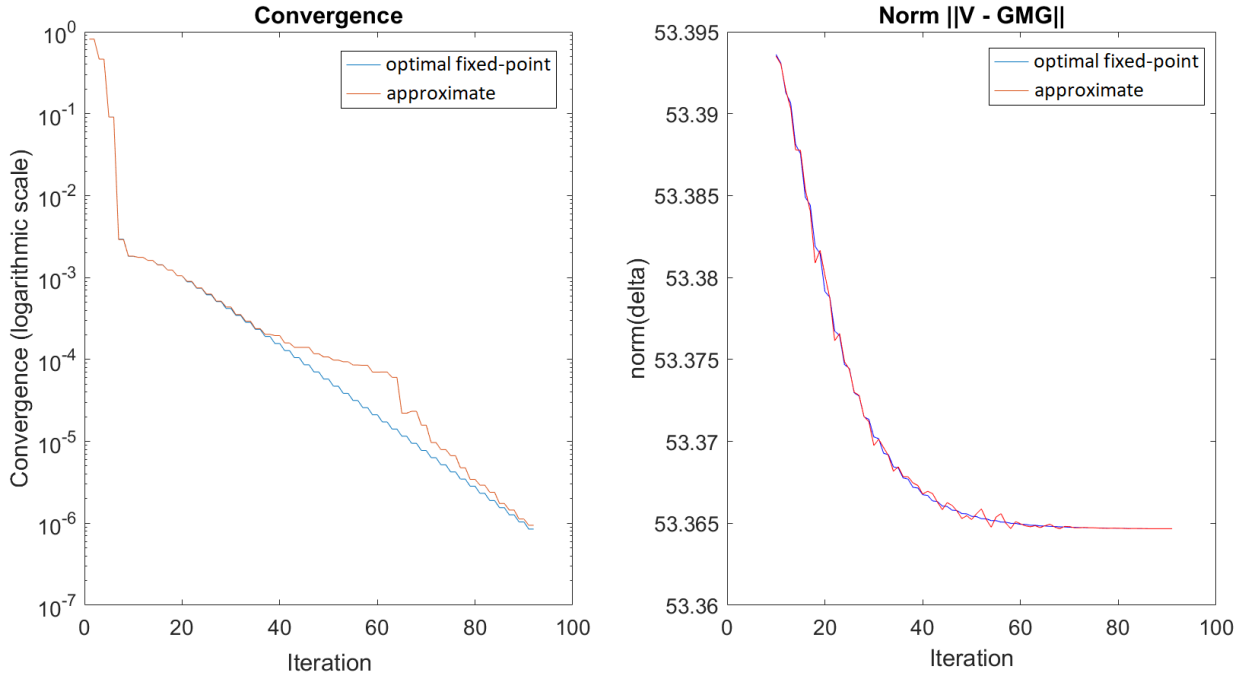
The convergence behavior is depicted in Fig. 57.

Synthesis

The power of the approximate design is equal to 2.0821 mW. The area is equal to 20604 μm^2. Therefore, by increasing the area by 76%, 24% of the energy can be saved in the biased case, and 28.8% in the unbiased case, making this method more effective than the truncation of partial products, which was not expected beforehand, as the performance of the partial products truncation showed better results in the comparisons of chapter 3.



a) biased



b) unbiased

Figure 57. Effect of the truncation of inputs for the biased and unbiased cases.

4.3.3. DRUM.

The determined parameters of the DRUM method are presented in Table 10. The approximation can be applied for 60 initial iterations.

Multiplier (or squarer)	Size of the small DRUM multiplier
e_sac^2	10x10
f_sac^2	9x9
eh	10x10
ft	10x10
et	8x8
fh	8x8

Table 10. DRUM parameters.

Unbiasing

Unbiasing by the accumulator initialization is not effective for this method, presumably because it is already unbiased, as can be suggested by the fact that it can be applied for 60 iterations, more than the truncation methods in the biased case.

It should be noted that in this work the smaller accurate multiplier is assumed to have the operands of the same size. It's possible that if the smaller multipliers can have different sizes, the savings can be increased. However, it would introduce one additional optimization parameter, significantly increasing the number of required simulations.

The effect of this approximation method on the convergence process can be seen in Fig. 58.

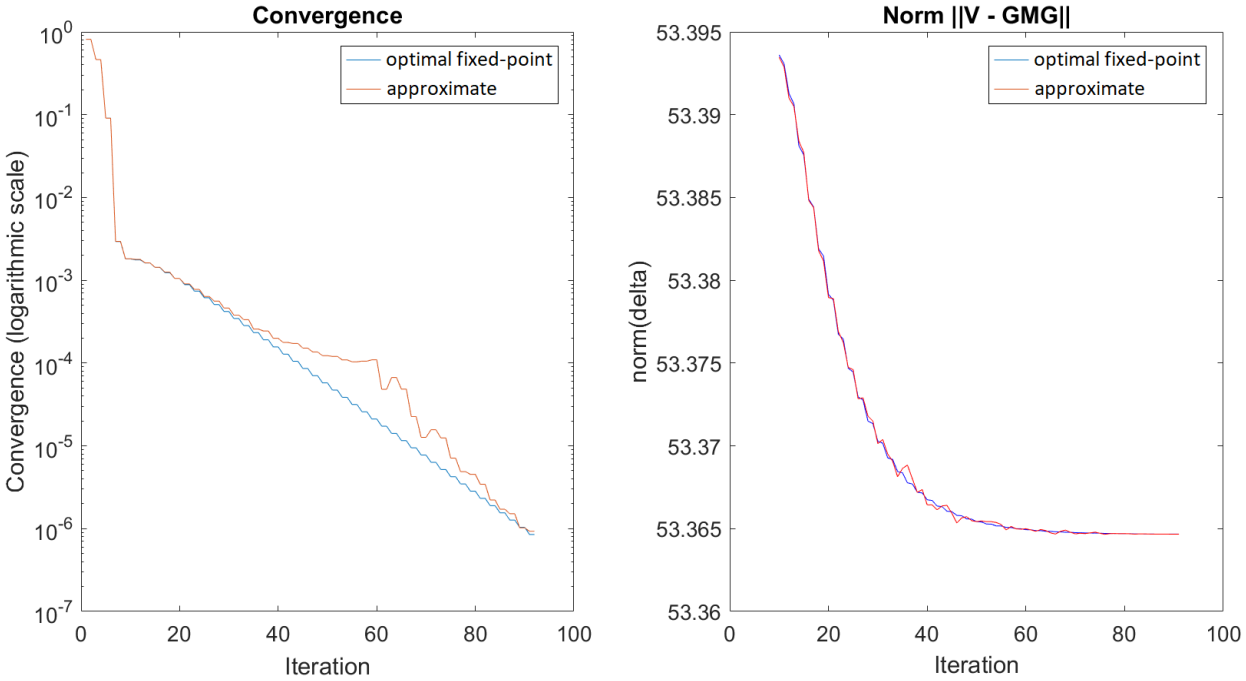


Figure 58. Effect of the DRUM method applied to the Stefcal.

Synthesis

The power of the approximate design is equal to 2.7114 mW. The area is 20835 μm^2. By introducing an area overhead of 77%, 15% of the energy can be saved, significantly smaller than both the truncation methods.

4.3.4. OR-compression.

This method is similar to the Low-power MAC method, and also to the M2 recursive type. The OR-compression is applied to the partial product matrix in a column-by-column way, similar to the truncation of partial products. Initially it was planned to apply the M2 type in a recursive way, however after obserbing that this method uses OR gates in an inefficient way, it was decided to apply the OR-compression directly on the partial product matrix. However, the analysis of the errors introduced by the M2 type was made which allows to understand how to apply the OR-compression. Fig. 59 shows the partial product matrix consisting of 2x2 multipliers for the ft and fh multipliers which have the same size of 24x18. The structure consists of 108 2x2 multipliers.

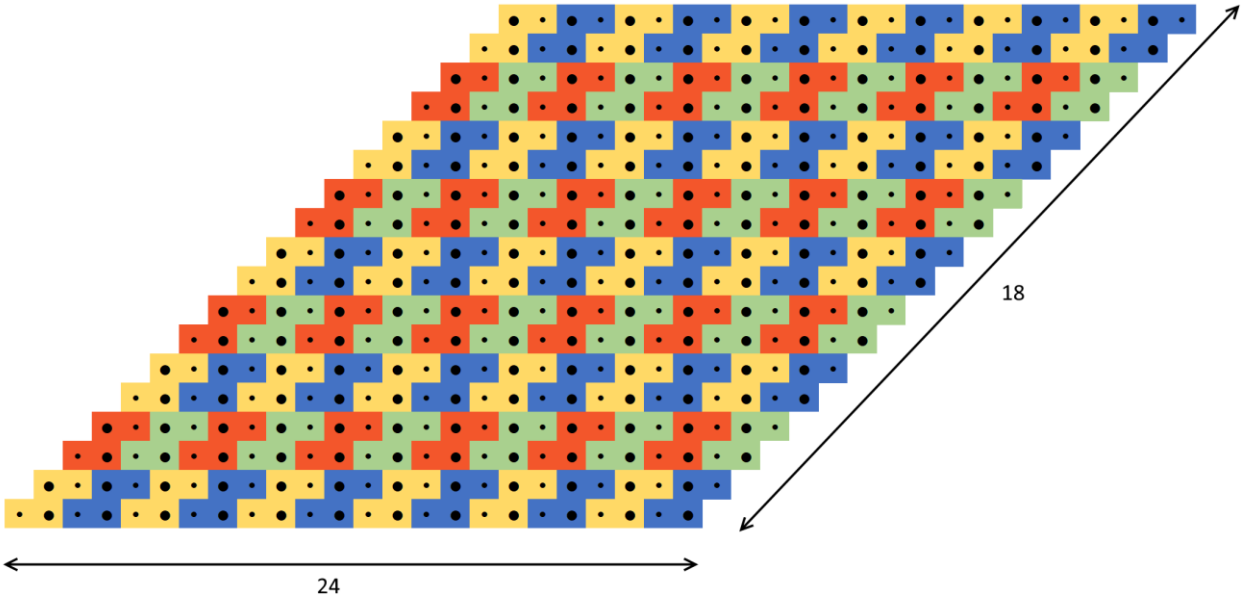


Figure 59. 24x18 multiplier matrix consisting of 108 2x2 multipliers.

								530.56	206.12	76.90	30.04	15.40	7.91	3.94	1.96	0.98	0.49	0.25	0.12
							1067.93	415.11	152.57	59.84	30.89	15.83	7.91	3.96	1.97	0.99	0.49	0.25	
						2126.97	826.12	305.61	121.88	61.80	31.70	15.85	7.89	3.95	1.98	0.99	0.49		
					4186.03	1669.63	615.36	243.48	122.91	63.34	31.34	15.71	7.87	3.93	1.96	0.98			
				8436.65	3285.60	1216.43	487.14	247.61	126.56	62.54	31.57	15.78	7.90	3.95	1.97				
			16817.33	6578.39	2428.59	974.33	482.97	252.07	125.66	62.26	31.22	15.64	7.84	3.90					
		24947.55	19321.60	6647.93	2563.02	1307.30	674.43	336.93	167.86	84.12	41.98	20.99	10.52						
	556.04	54467.41	19561.70	7737.84	3900.47	2022.02	1003.68	500.96	250.84	125.54	62.73	31.36							
743215.24	34275.35	26324.62	16167.04	8024.08	3984.75	1997.81	1006.90	503.39	251.48	125.36	62.74								
743215.24	17415.69	35246.53	17966.92	7484.98	3391.41	1616.36	770.57	370.64	169.54	80.19	39.36	18.32	8.26	3.94	1.97	0.98	0.49	0.25	0.12

Figure 60. Expected errors of 2x2 multipliers of the M2 type. The last row is the mean of the corresponding columns.

Figure 60 shows the probabilities of errors multiplied by the weights of the columns in which the 2x2 multipliers reside, so the numbers correspond to the expected errors of each of the 2x2 multipliers. All multipliers are of the M2 type. The 2x2 approach is not used for the Stefcal, but the error matrix in Fig. 60 shows that the OR-compression applied in a column-by-column way is a reasonable way of approximation, as the error increases gradually if the approximation step is one column.

As the Low-power MAC is a special case of this method, it was tried as well. However, the Stefcal algorithm cannot survive even the minimal approximation of this method. Fig. 61 shows the effect of applying the Low-power MAC method just to the eh multiplication for only 20 initial iterations. The number of iterations is larger and $Diff\_rel > 10^{-5}$.

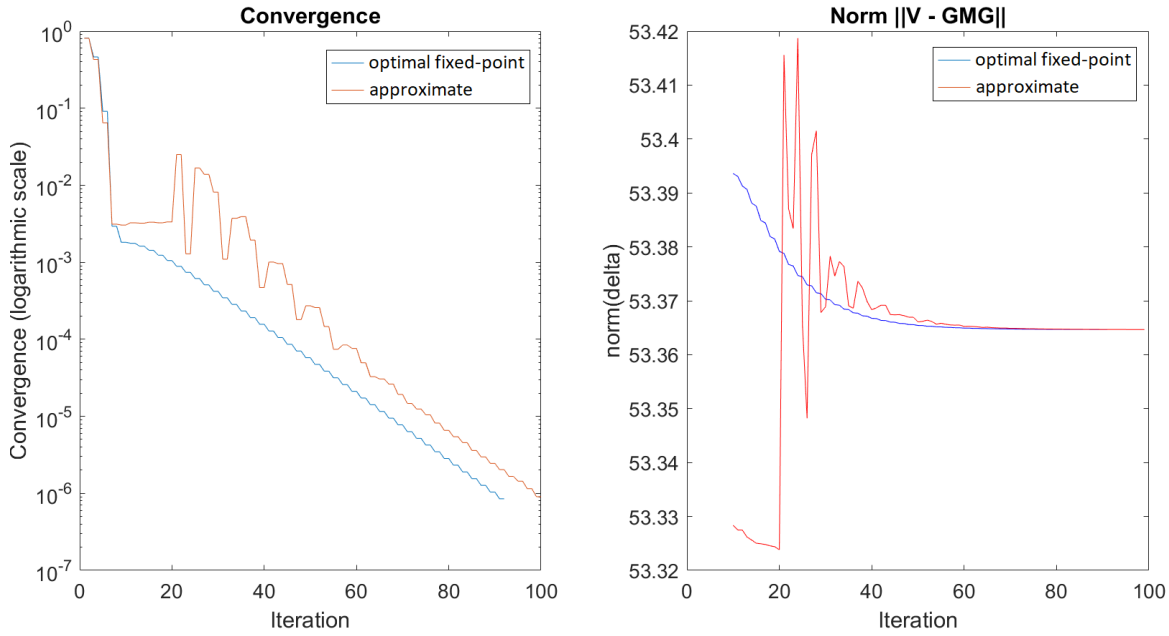


Figure 61. Low-power MAC applied to Stefcal.

The parameters for each multiplier and squarer for the OR-compression of the columns are presented in Table 11. The method is applied for 60 iterations, and the unit gate model predicts the savings of only 5.5%.

Multiplier (or squarer)	**# of compressed columns**
e_sac^2	21
f_sac^2	20
eh	15
ft	16
et	18
fh	19

Table 11. OR-compression parameters.

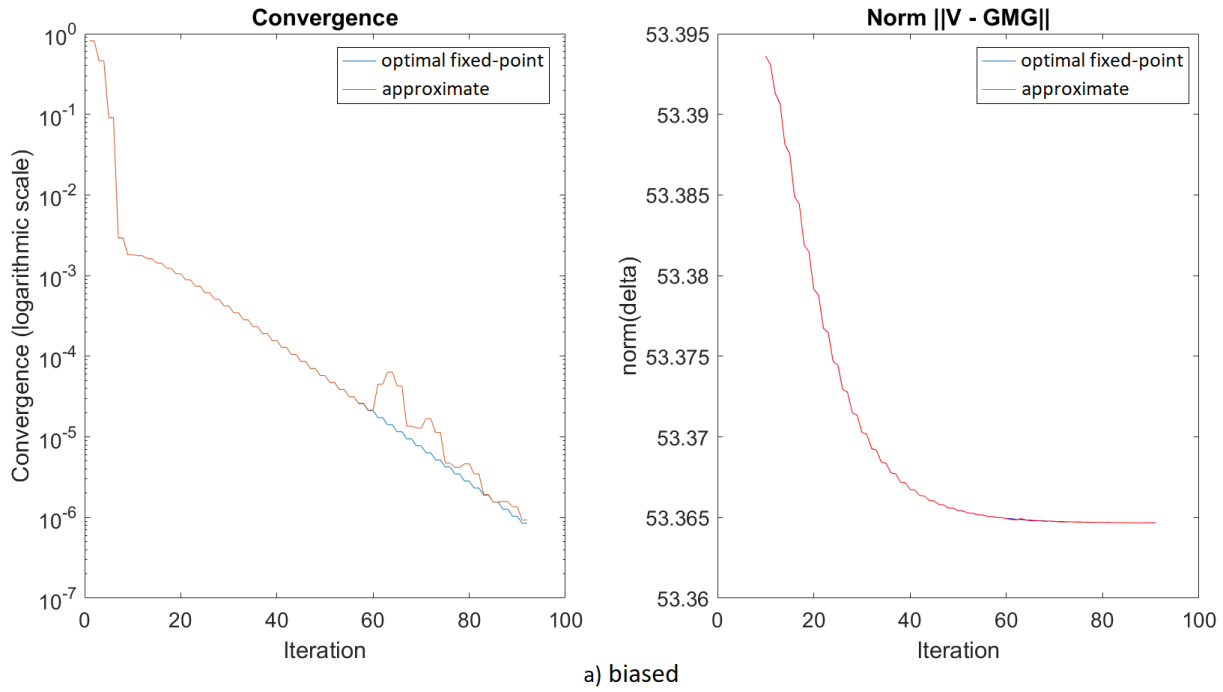
Unbiasing

Unbiasing by the accumulator initialization allows to increase the number of approximate iterations from 60 to 66.

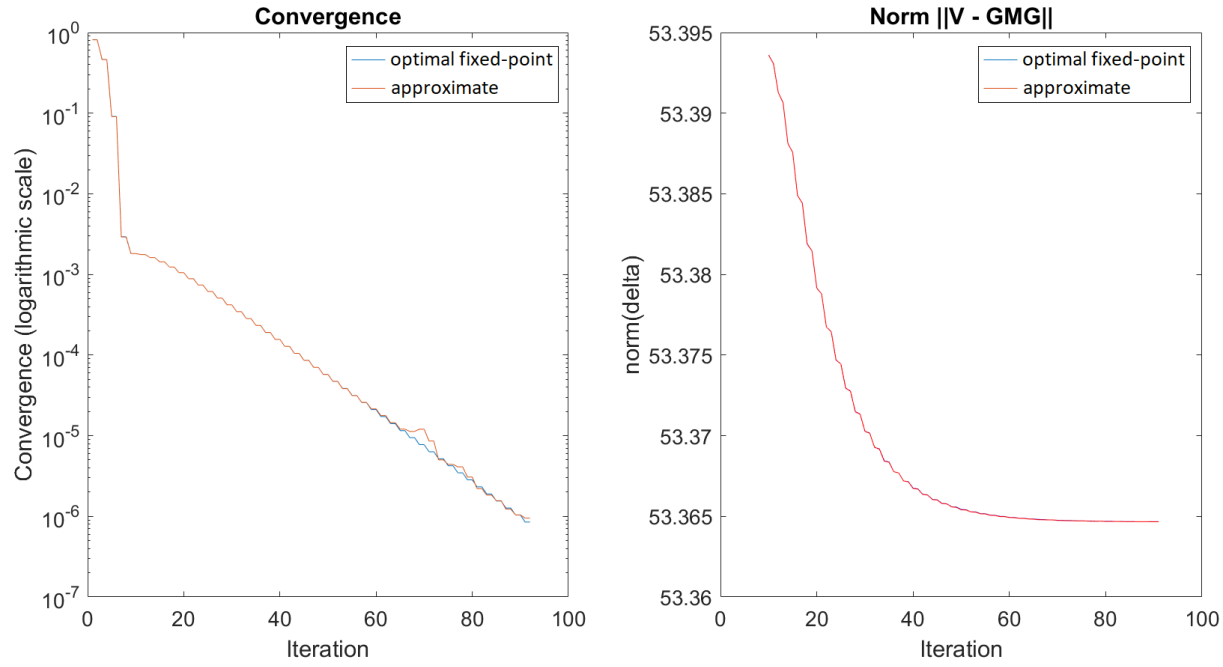
Synthesis

The power is equal to 3.2757 mW and the area is 25490 μm^2. This method allows to save only 5% of the energy in the biased case and 5.6% in the unbiased case by introducing an area overhead of 94%. This is clearly the least efficient method of approximation among the applied ones.

The convergence process can be seen in Fig. 62.



a) biased



b) unbiased

Figure 62. Effect of the OR-compression method applied to the Stefcal.

4.4. Overview.

Table 12 presents an overview of all the applied methods and the savings which can be achieved by them.

Method	**# of approximate iterations (biased/unbiased)**	**Energy saved, % (biased/unbiased)**	**Area overhead, %**
Truncation of partial products	51/61	20/24.7	83
Truncation of inputs	52/64	24/28.8	76
DRUM	60/60	15/15	77
OR-compression	60/66	5/5.6	94

Table 12. Approximate methods comparison.

The truncation of inputs is the most effective method to apply to the Stefcal algorithm. It allows to save more than the other methods while having the smallest area overhead. The reason why the truncation of inputs is more effective than the truncation of partial products is not clear. It might be related to the way the 2's complement matrix is truncated. If the approximations are applied to the signed-magnitude representation, the truncation of partial products may become the best method. The OR-compression is extremely inefficient. The savings are the smallest, and the area overhead is the largest. It is reasonable to expect that the possible savings achieved by using the 2x2 multipliers in different combinations will not be significantly better than the OR-compression.

Conclusions.

In this work, a comparison of approximate computing techniques applied to the MAC unit was performed. The comparison included the novel approximation methods as well as the traditional fixed-point truncation methods. For most of the cases considered, these simple truncation methods are the best approximations. However, no definitive conclusion can be made as to which approximation is the most efficient, as for various distributions and data representations different techniques can be advantageous. Also, the synthesis tool proves to be difficult to use for these comparisons as for different frequencies the results can differ significantly, which make these results look like random at times. To better understand these results, 1 Hz frequency was used which removes the critical path constraint from the designs, and also a model was implemented which computes the number of gates for each design. Results which are present in all the three can be given more credit.

The least squares algorithm performing the radio astronomy gain calibration was approximated in a systematic way by applying four different approximations. The two truncation methods show the best results allowing to save more energy compared to the DRUM approximation and the OR-compression. The best result is demonstrated for the truncation of inputs, 28.8% of the energy can be saved compared to the optimal fixed-point implementation by introducing an area overhead of 76%. The first two thirds of iterations can be mapped to this approximate truncated architecture, switching to the optimal fixed-point architecture only for the final third.

Future work.

The element-wise product part of the Stefcal algorithm was not approximated in this work. It can be expected that its approximation will allow to save more energy.

The approach to approximation in this work introduces an area overhead as two architectures are required, an approximate structure for the initial iterations, and an exact one with optimized bit widths for the final iterations. The truncation of inputs, apart from demonstrating the best results for the Stefcal approximation, has another important advantage over other approximations. It can be used in a dynamic way. Instead of using two different architectures, one optimal architecture can be used in which the least significant bits of the multiplier inputs are set to zero for the initial iterations by using multiplexers. Then, for the final part of the computation these multiplexers propagate all the bits, increasing the precision. This way the multiplier part corresponding to these “frozen” bits has no switching activity and saves power. The input bits can be activated gradually, smoothly increasing the precision. For the maximum savings, this approach would require to determine the optimal function according to which the bits are activated depending on the current iteration number. It can also be possible that the overhead introduced by the multiplexers and their control will nullify the potential savings.

The Stefcal approximations were applied to the 2’s complement version of the algorithm. It can be possible that the signed-magnitude representation is more power efficient. It would require repeating the full process again starting from the floating-point to fixed-point conversion as for the signed-magnitude representation the optimal word lengths might differ from the 2’s complement representation. This is because in 2’s complement data the truncation of both positive and negative values introduces a negative bias, making the values smaller. The truncation of signed-magnitude signals is different as it makes the positive values smaller and the negative values larger, which may lead to no bias at all depending on the data distribution.

The 16x16 comparison in this work doesn’t includes the recursive multipliers. It could be helpful to see their performance for the 16x16 case as well.

Results for the 16x16 case suggest that the truncation of the Booth multiplier partial products can be more power efficient as the partial product matrix of the Booth multiplier is smaller.

All this potential future work can be done faster by using the models and scripts developed in the process of writing this work.

References.

[1] D. Esposito, A. G. M. Strollo and M. Alioto. 2017. Low-power approximate MAC unit. In *13th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), Giardini Naxos, 2017, pp. 81-84.*

[2] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar and J. Henkel. 2016. Architectural-space exploration of approximate multipliers. In *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM, 80.

[3] V. Gupta, D. Mohapatra, A. Raghunathan and K. Roy. 2013. Low-Power Digital Signal Processing Using Approximate Adders. In *IEEE Trans. CAD of Integrated Circuits and Systems, 32(1), pp. 124-137*.

[4] G.A. Gillani and A.B.J. Kokkeler. 2017. Improving Error Resilience Analysis Methodology of Iterative Workloads for Approximate Computing. In *Proceedings of CF'17, Siena, Italy, May 2017.*

[5] S. Misailovic, S. Sidiroglou, H. Hoffmann and M. Rinard. 2010. Quality of service profiling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1. 25–34.*

[6] A. Mishra, R. Barik and S. Paul. 2014. iACT: A software-hardware framework for understanding the scope of approximate computing. In *Workshop on Approximate Computing Across the System Stack (WACAS).*

[7] P. Roy, R. Ray, C. Wang and W. Wong. 2014. ASAC: Automatic sensitivity analysis for approximate computing. In *ACM SIGPLAN Notices.* ACM, 95–104.

[8] P. Roy, J. Wang and W. Wong. 2015. PAC: program analysis for approximation-aware compilation. In *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. IEEE Press, 69–78.

[9] V. Chippa, S. Chakradhar, K. Roy and A. Raghunathan. 2013. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*. ACM, 113.

[10] M. A. Hanif, R. Hafiz and M. Shafique. 2018. Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2018, pp. 913-916.*

[11] E. Nogues, D. Menard and M. Pelcat. 2016. Algorithmic-level Approximate Computing Applied to Energy Efficient HEVC Decoding. In *IEEE Transactions on Emerging Topics in Computing.*

[12] B. Verstoep. 2018. Approximate multipliers for MAC.

[13] S. Salvini and S. Wijnholds. 2014. Fast gain calibration in radio astronomy using alternating direction implicit methods: Analysis and applications. *Astronomy & Astrophysics* 571 (2014), A97.

[14] B. Barrois, O. Sentieys and D. Menard. 2017. The hidden cost of functional approximation against careful data sizing — A case study. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, Lausanne, pp. 181-186.*

[15] V. Gupta, D. Mohapatra, S.P. Park, A. Raghunathan. 2011. IMPACT: IMPrecise adders for low-power approximate computing. *International Symposium on Low Power Electronics and Design (ISLPED), pp. 409 – 414, 2011.*

[16] J. Han and M. Orshansky. 2013. Approximate computing: An emerging paradigm for energy-efficient design. In *Proceedings of the 18th IEEE European Test Symposium (ETS)*.

[17] P. Kulkarni, P. Gupta, M. Ercegovac. 2011. Trading Accuracy for Power with an Underdesigned Multiplier Architecture. *24th International Conference on VLSI Design (VLSI Design), pp. 346 – 351, 2011.*

[18] S. Hashemi, R. I. Bahar and S. Reda. 2015. DRUM: A Dynamic Range Unbiased Multiplier for approximate applications. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, 2015, pp. 418-425.*

[19] G.A. Gillani, M.A. Hanif, M. Krone, S.H. Gerez, M. Shafique and A.B.J. Kokkeler. 2018. SquASH: Approximate Square-Accumulate with Self-Healing. *IEEE Access.*

[20] B. Parhami. 2012. Algorithms and design methods for digital computer arithmetic. Oxford University Press.

[21] W. Townsend et al. A Comparison of Dadda and Wallace Multiplier Delays. 2003. In *SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations XIII.*

[22] A. Tyagi. A reduced-area scheme for carry-select adders. 1993. In *IEEE Trans. Comput., vol. 42, no. 10, pp. 1163–1170, Oct. 1993.*

[23] I. Hughes and T. Hase. 2010. Measurements and their uncertainties. A practical guide to modern error analysis. Oxford University Press.

[24] A. Sampson. 2017. Approximate Computing Is Dead; Long Live Approximate Computing. http://www.cs.cornell.edu/~asampson/blog/closedproblems.html

[25] V. Leon et al. 2018. Approximate Hybrid High Radix Encoding for Energy-Efficient Inexact Multipliers. In *IEEE transactions on VLSI systems, vol. 26, no. 3, march 2018.*

[26] S. Bhattacharyya et al. 2018. Handbook of signal processing systems. Third edition. Springer.

[27] J. Oedzes. 2018. Approximate signed multipliers for multiply-accumulate circuits.

[28] Q. Xu et al. Approximate computing: a survey. 2015. In *IEEE Design & Test.*

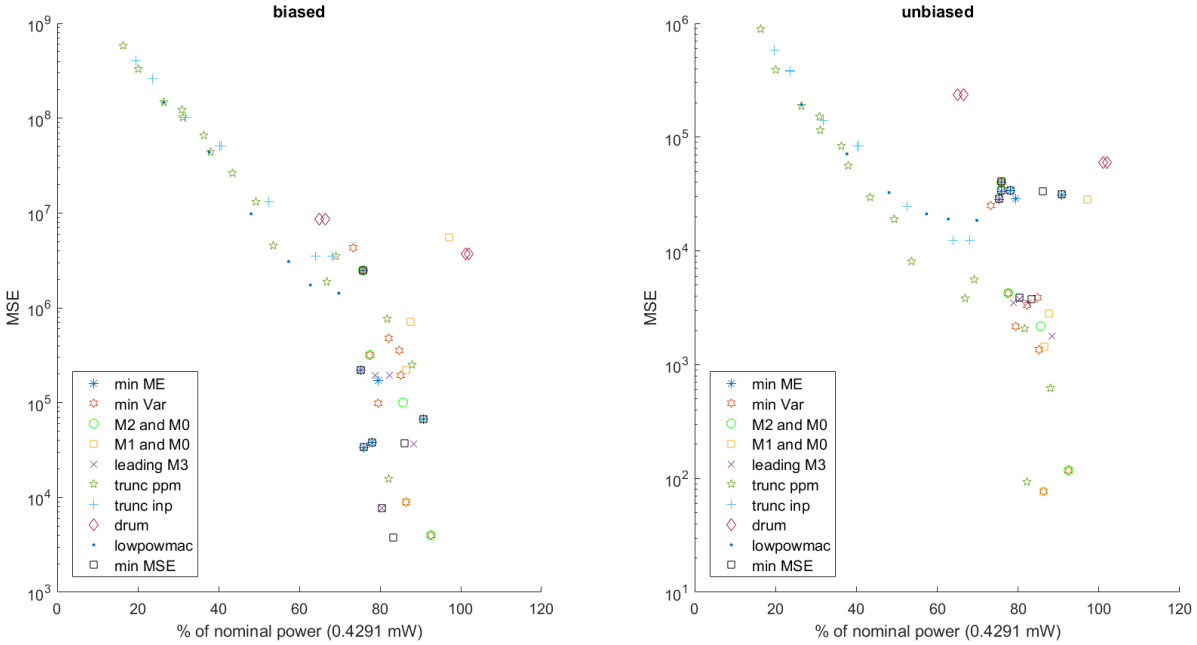
[29] J. Han et al. 2013. Approximate Computing: An Emerging Paradigm for Energy-Efficient Design. In *18th IEEE European Test Symposium (ETS).*

[30] S. Mittal. 2015. A survey of techniques for approximate computing. *20xx. ACM Comput. Surv. a, b, Article 1 (2015), 34 pages.*

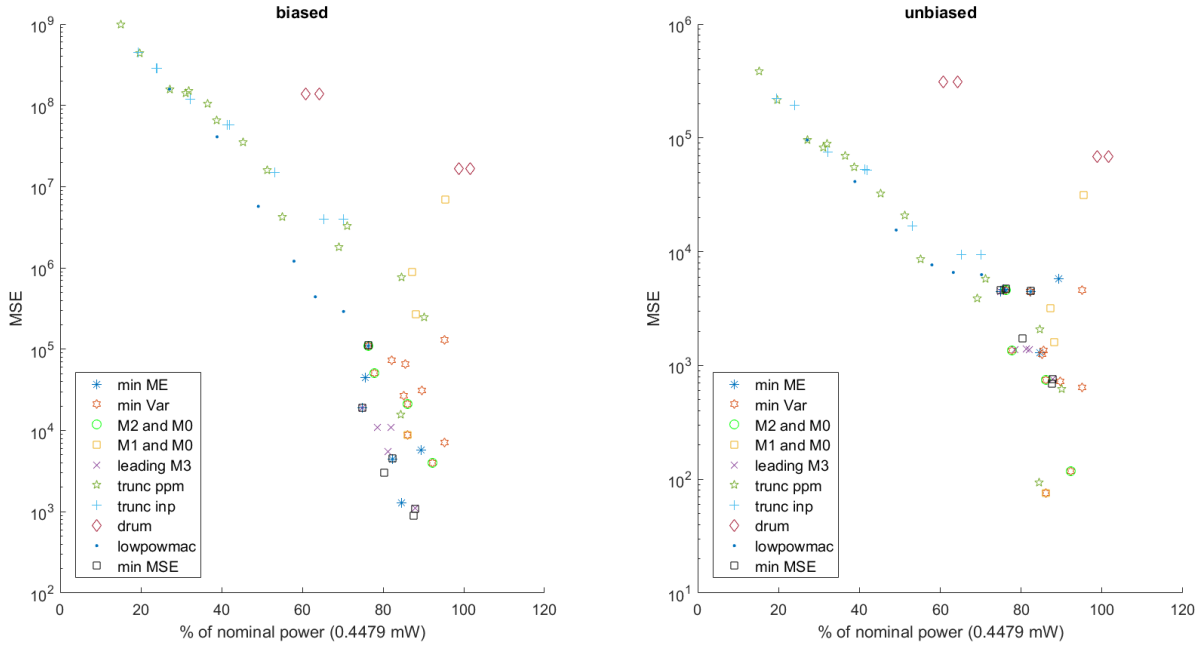
[31] S. Boyd. 2018. Introduction to applied linear algebra. Vectors, matrices, and least squares. Cambridge university press.

Appendix A. Approximate MAC comparison plots.

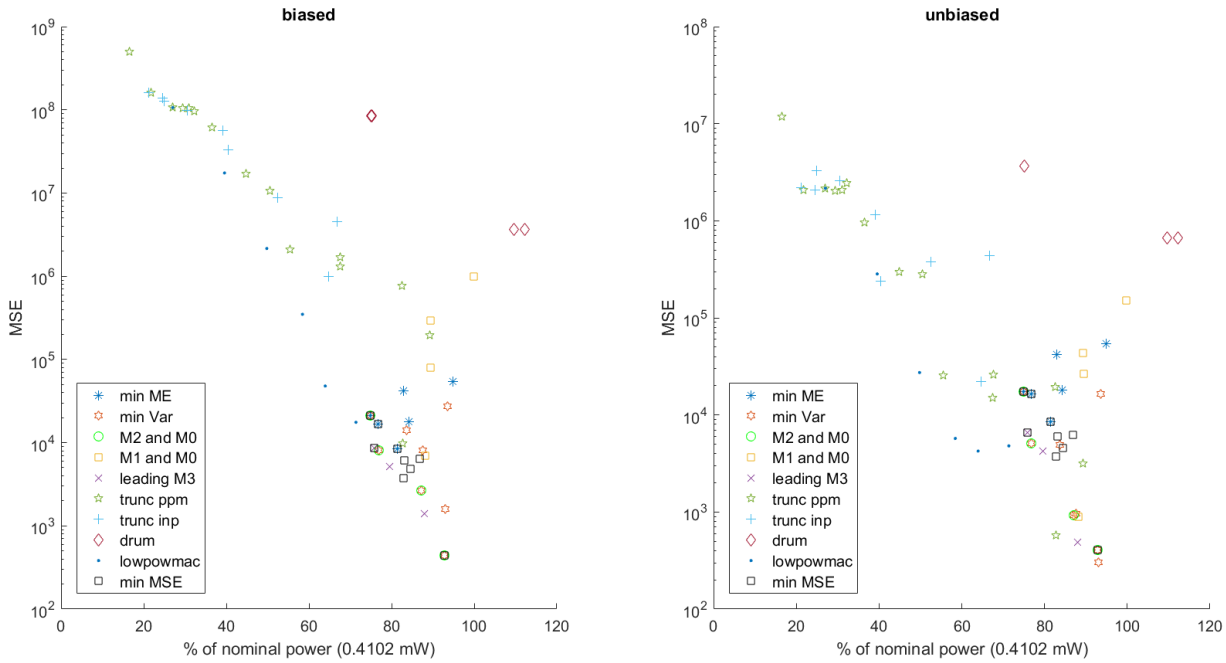
4x4. Unsigned.
Synthesized at 1.05 GHz.
compile -map_effort high; compile_ultra



a) uniform

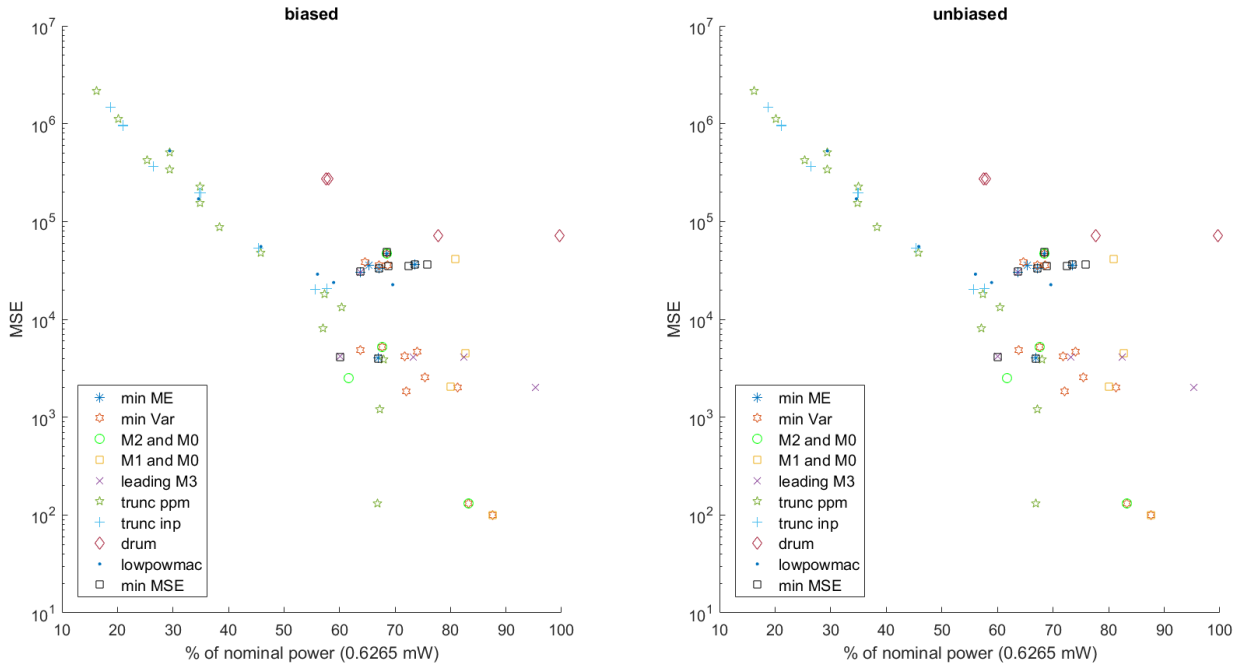


b) normal with mean=8 and std=2.5

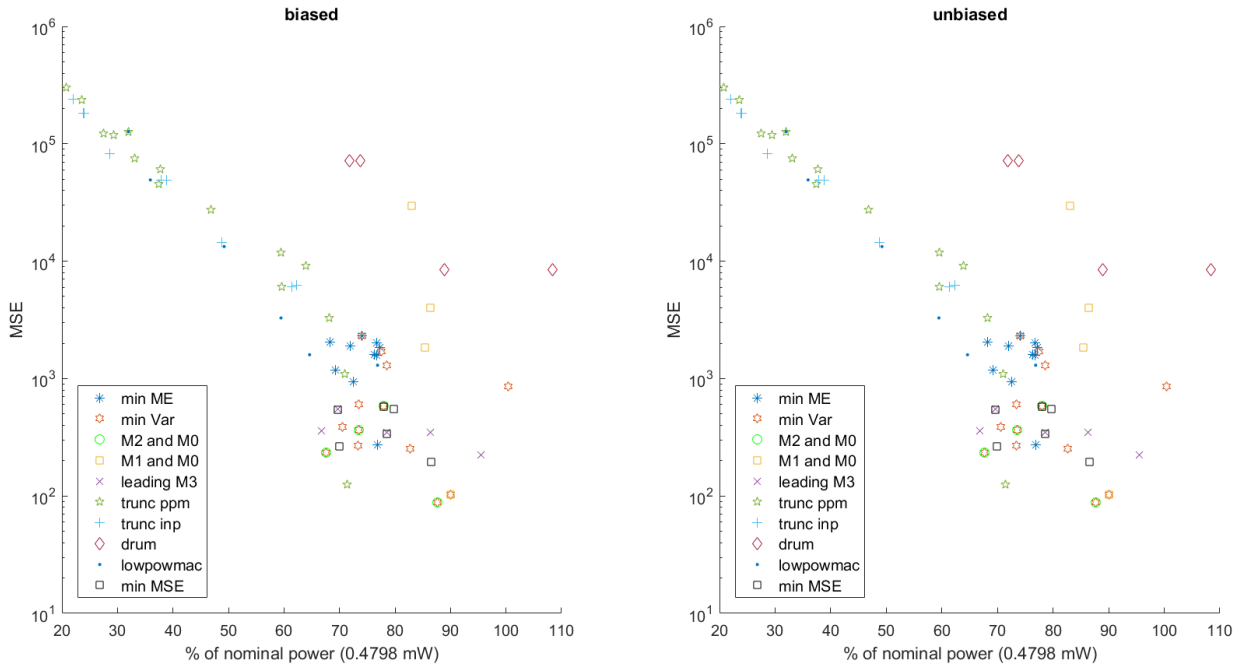

biased
unbiased
MSE
% of nominal power (0.4102 mW)
min ME
min Var
M2 and M0
M1 and M0
leading M3
trunc ppm
trunc inp
drum
lowpowmac
min MSE

c) Stefcal_CE

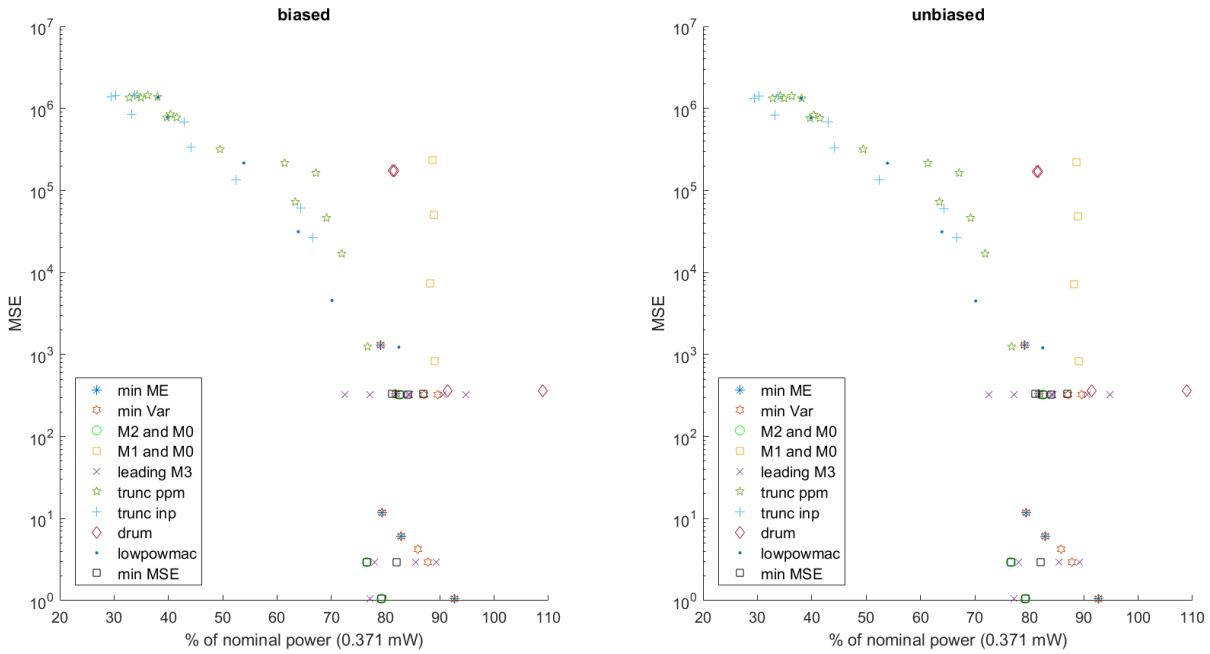
4x4. Signed magnitude.
Synthesized at 900 MHz.
compile -map_effort high; compile_ultra



a) uniform

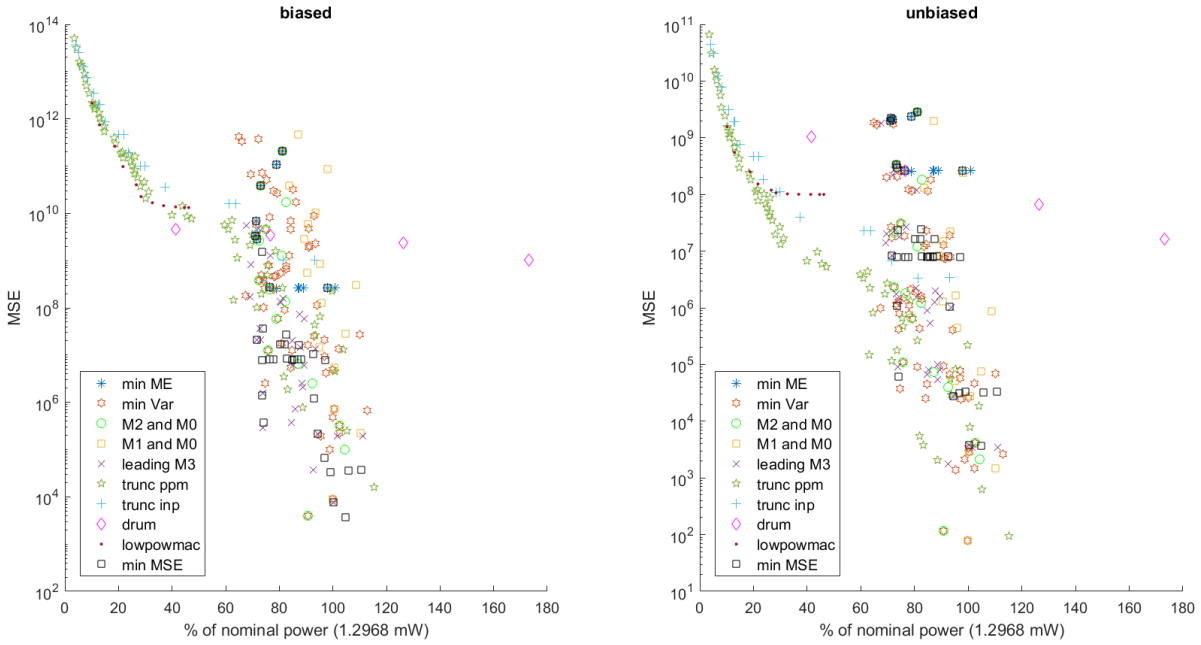


b) normal with mean=0 and std=5

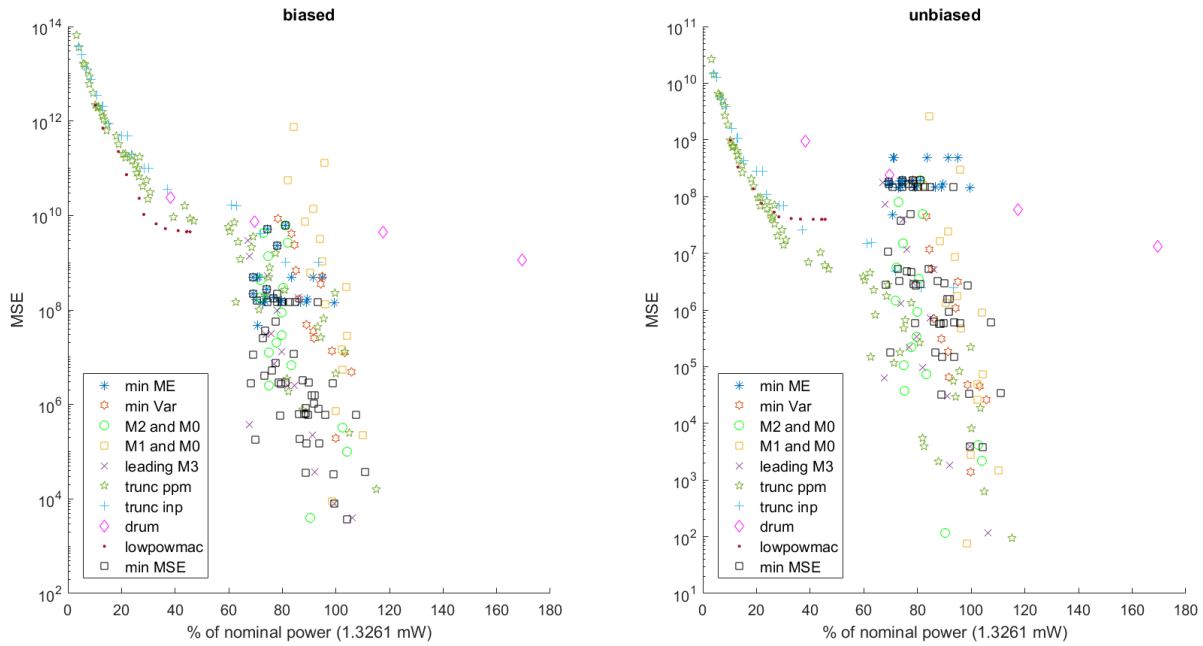

biased
unbiased
MSE
% of nominal power (0.371 mW)
min ME
min Var
M2 and M0
M1 and M0
leading M3
trunc ppm
trunc inp
drum
lowpowmac
min MSE

c) Stefcal_ET

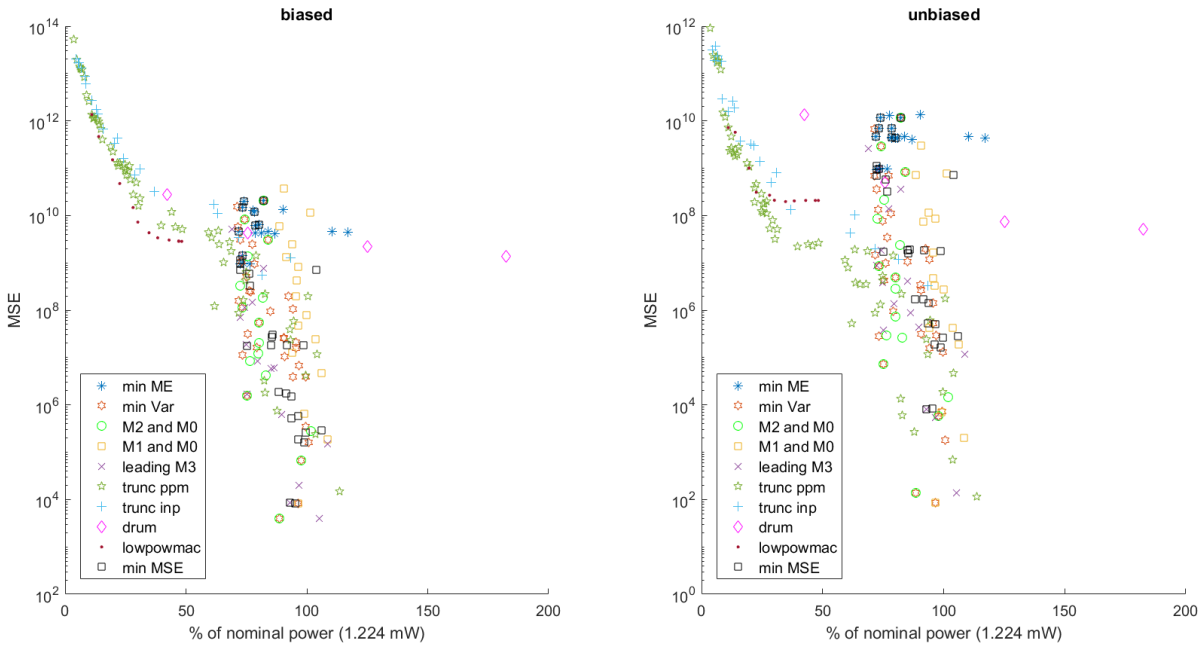
8x8. Unsigned.
Synthesized at 660 MHz.
compile -map_effort high; compile_ultra



a) Uniform

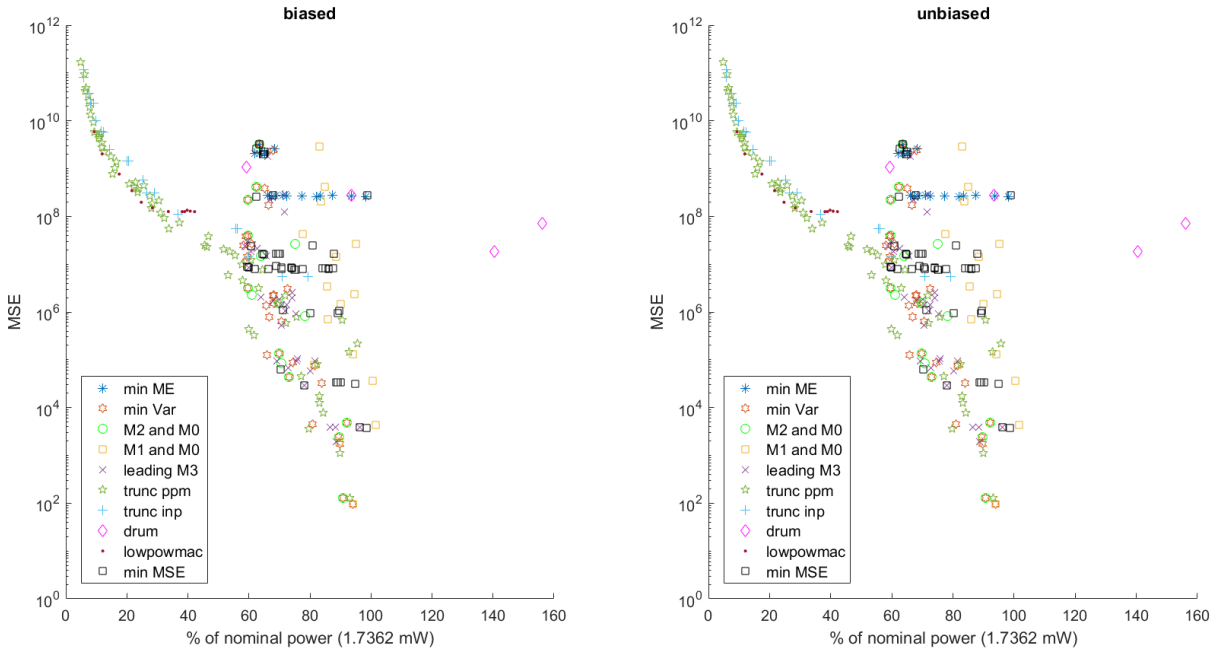


b) normal with mean=128 and std=40

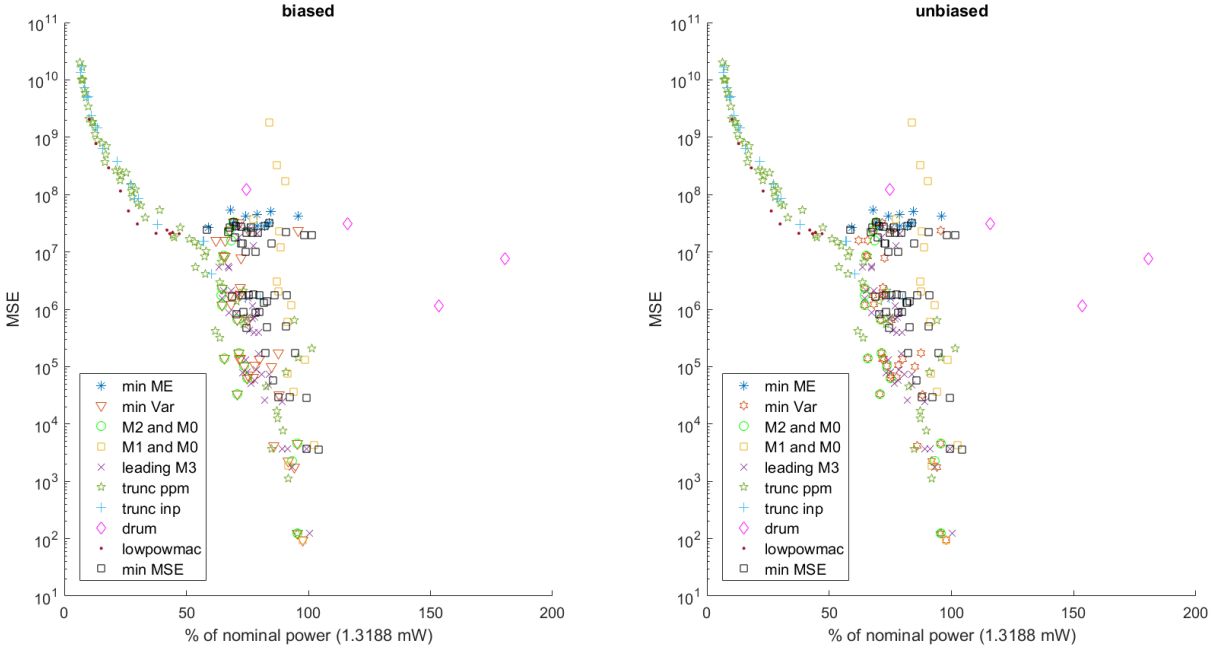

biased
unbiased
MSE
% of nominal power (1.224 mW)
min ME
min Var
M2 and M0
M1 and M0
leading M3
trunc ppm
trunc inp
drum
lowpowmac
min MSE

c) Stefcal_CE

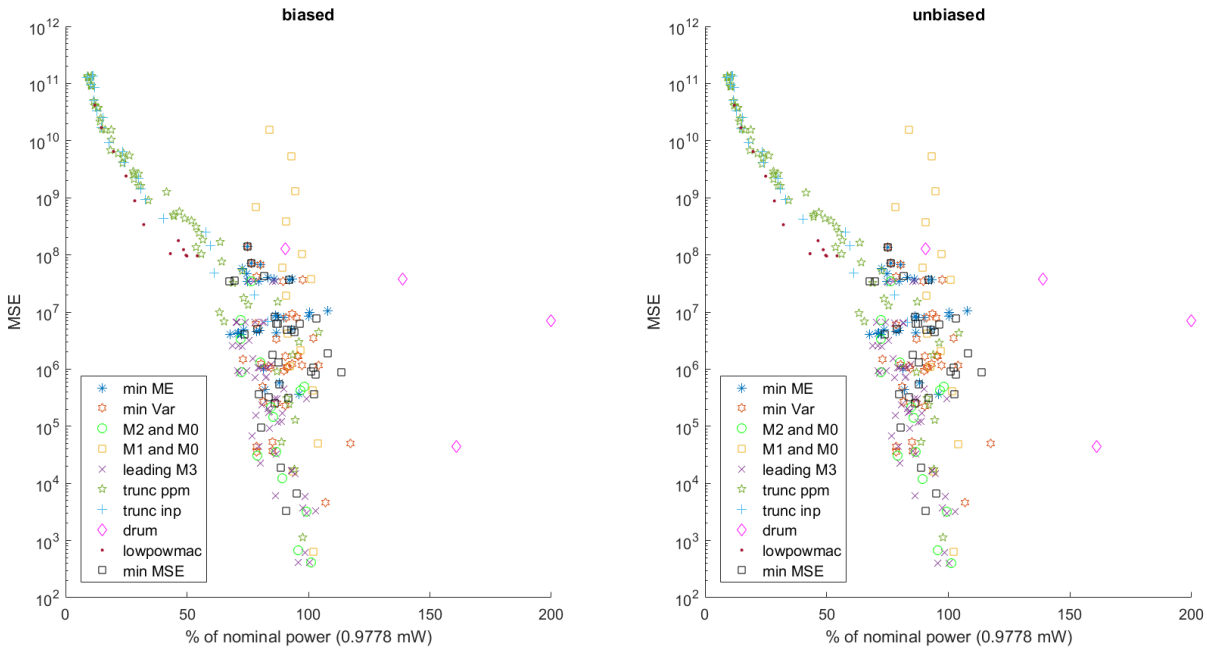
8x8. Signed magnitude.
Synthesized at 666 MHz.
compile -map_effort high; compile_ultra



a) uniform

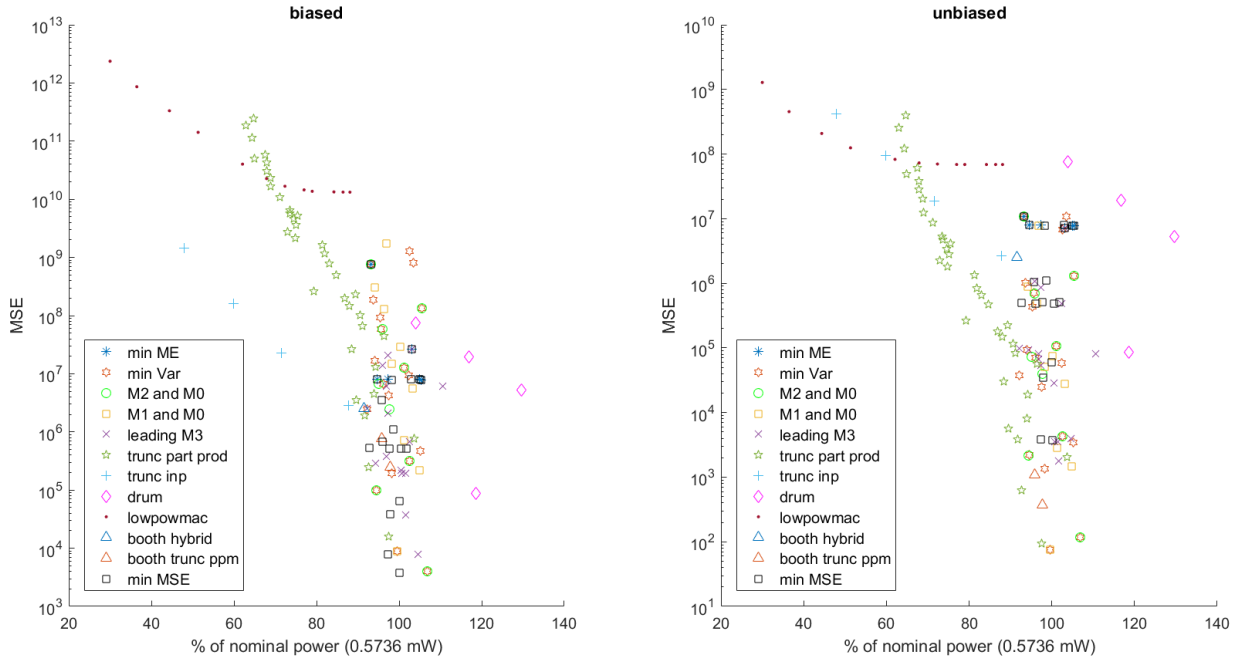


b) normal with mean=0 and std=80

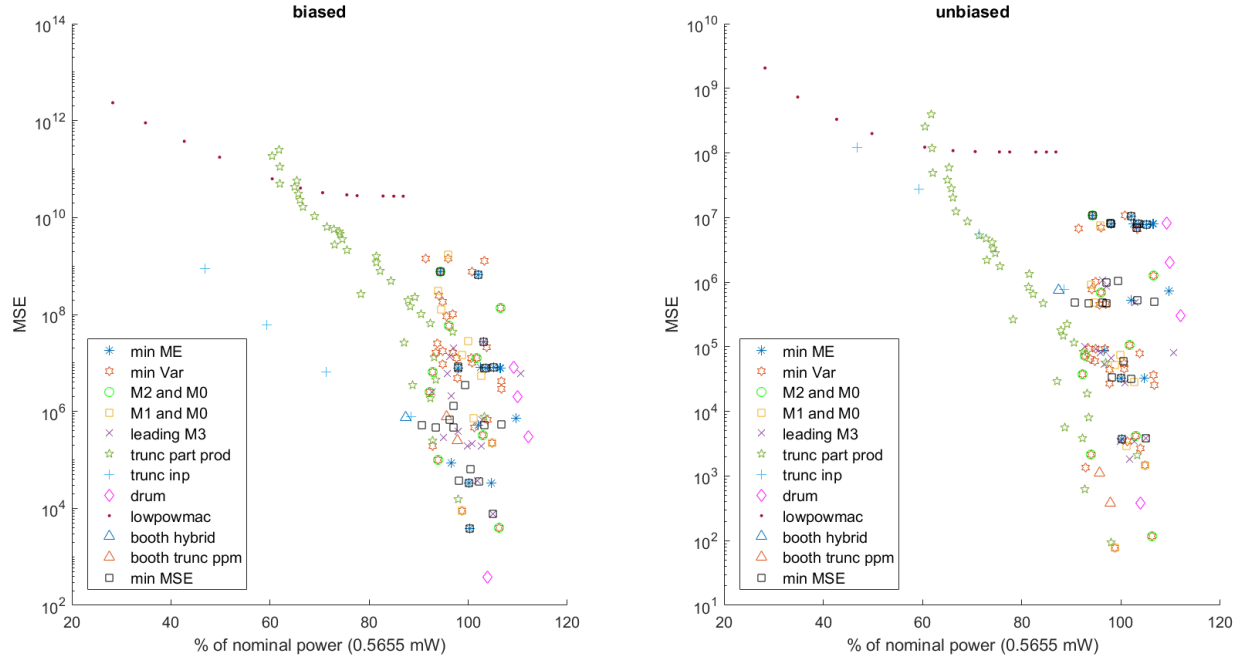

biased
unbiased
MSE
% of nominal power (0.9778 mW)
min ME
min Var
M2 and M0
M1 and M0
leading M3
trunc ppm
trunc inp
drum
lowpowmac
min MSE

c) Stefcal_ET

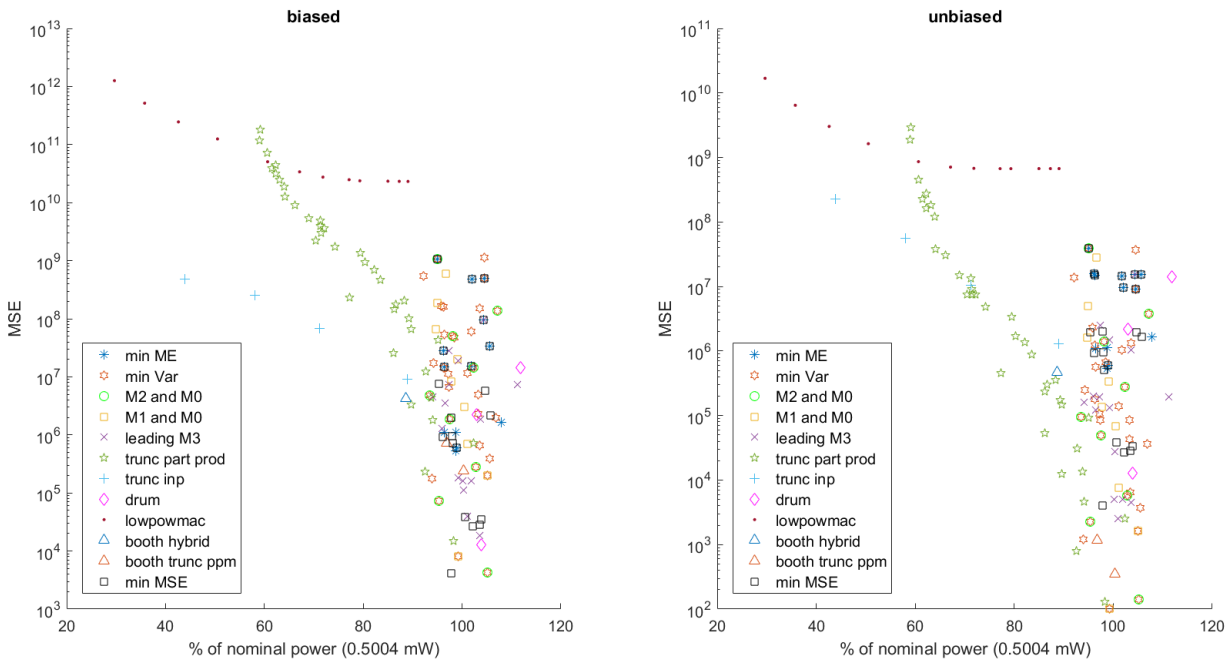
8x8. 2's complement.
Synthesized at 500 MHz.
compile -map_effort high; compile_ultra



a) uniform

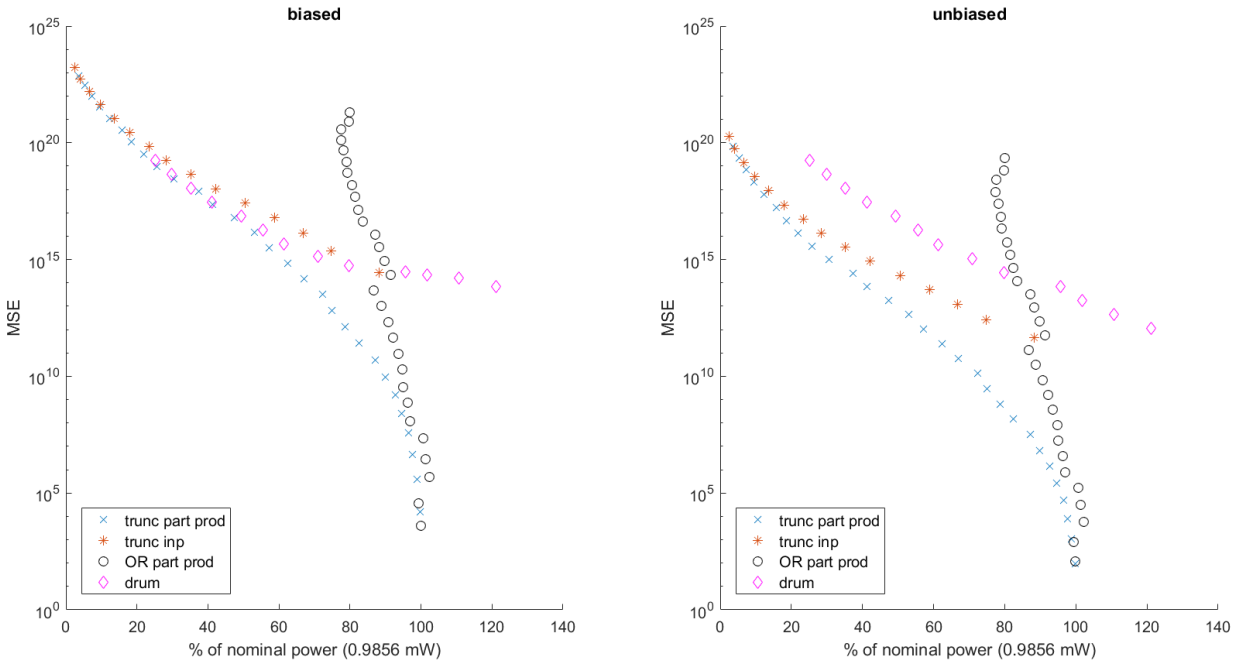


b) normal with mean=0 and std=40

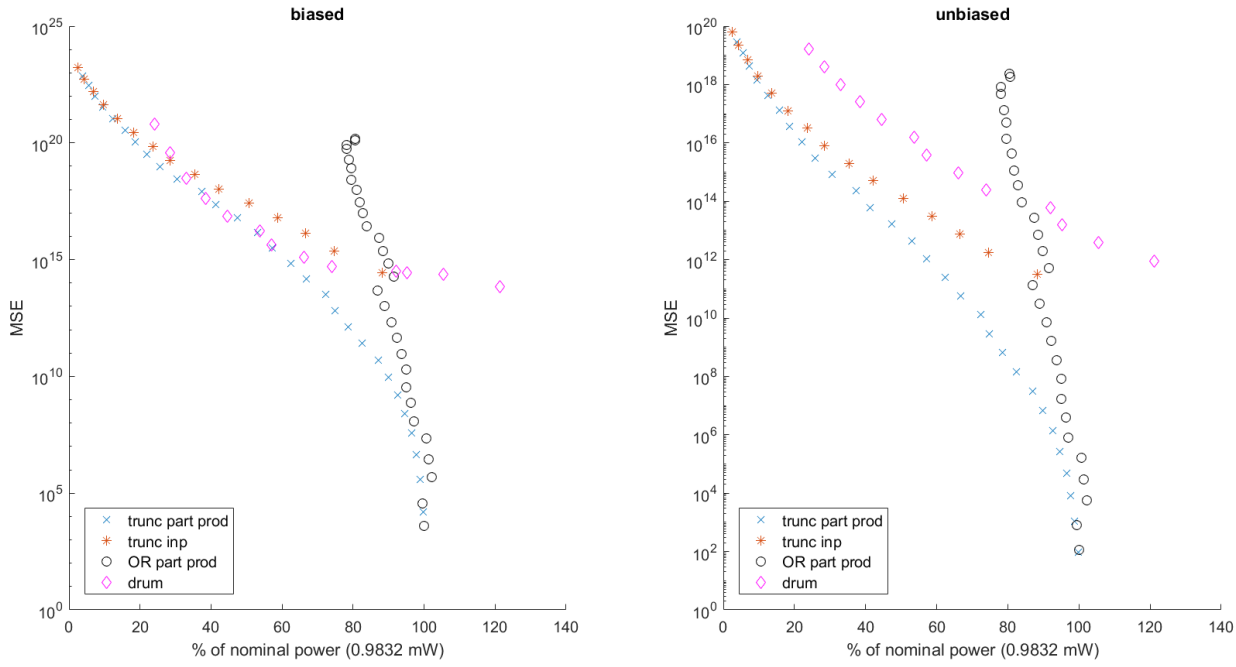

biased
unbiased
MSE
% of nominal power (0.5004 mW)
min ME
min Var
M2 and M0
M1 and M0
leading M3
trunc part prod
trunc inp
drum
lowpowmac
booth hybrid
booth trunc ppm
min MSE

c) Stefcal_ET

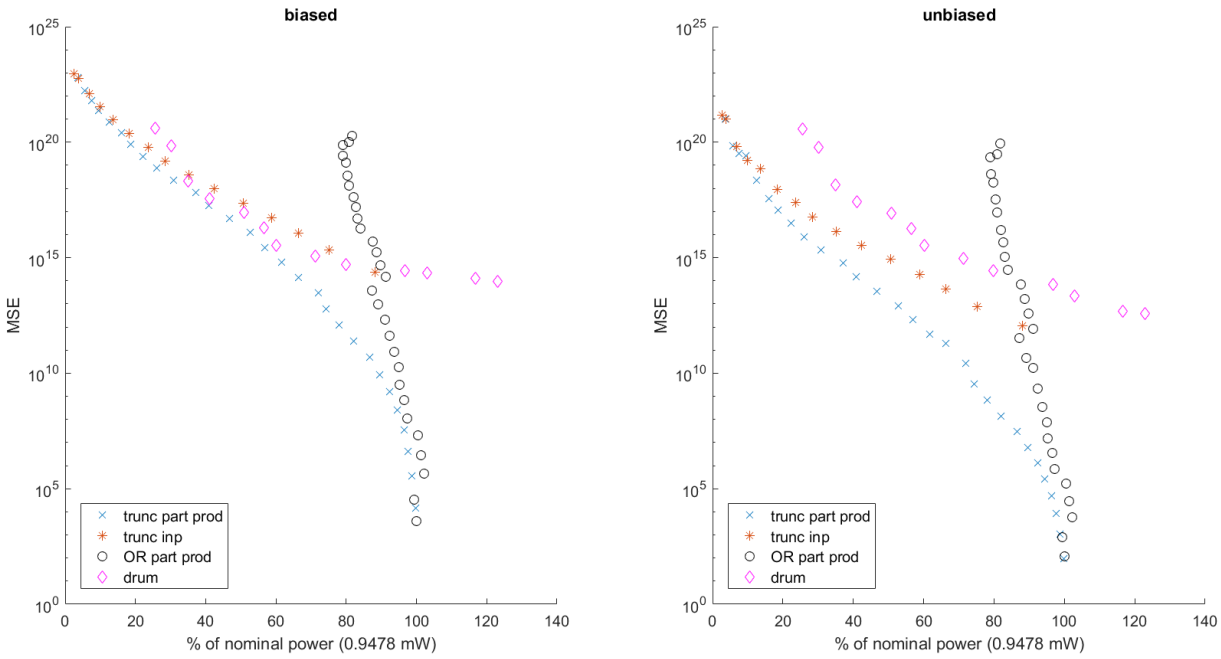
16x16. Unsigned.
Synthesized at 250 MHz.
compile -map_effort high; compile_ultra



a) uniform

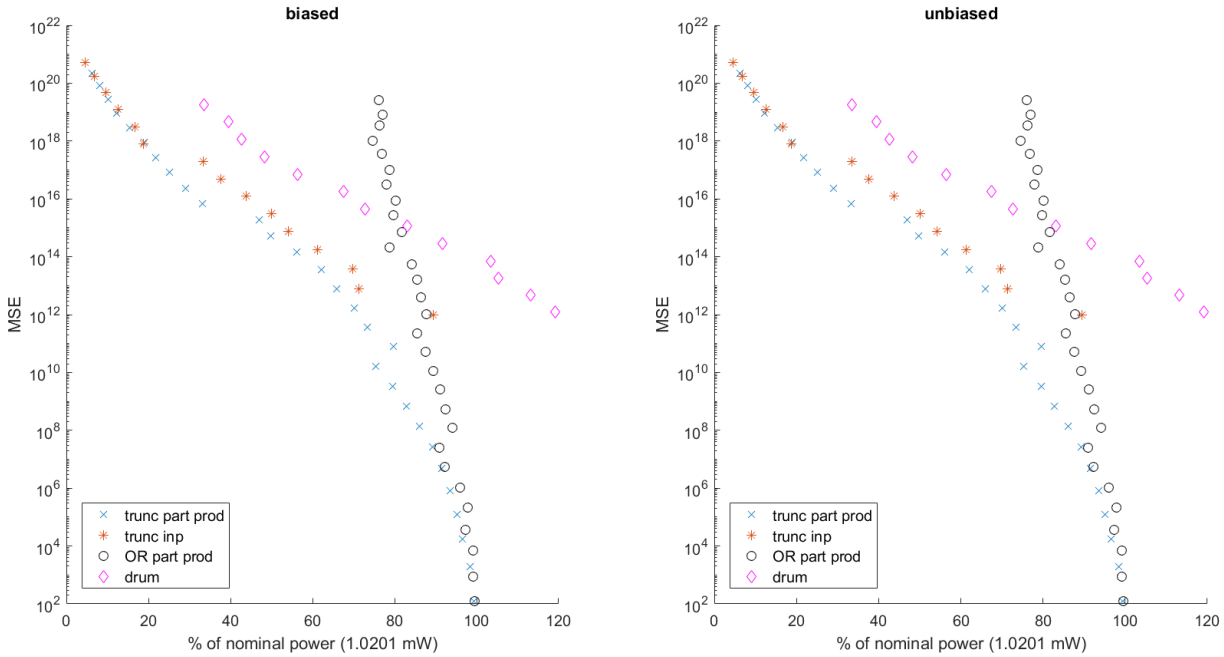


b) normal with mean=32768 and std=10240

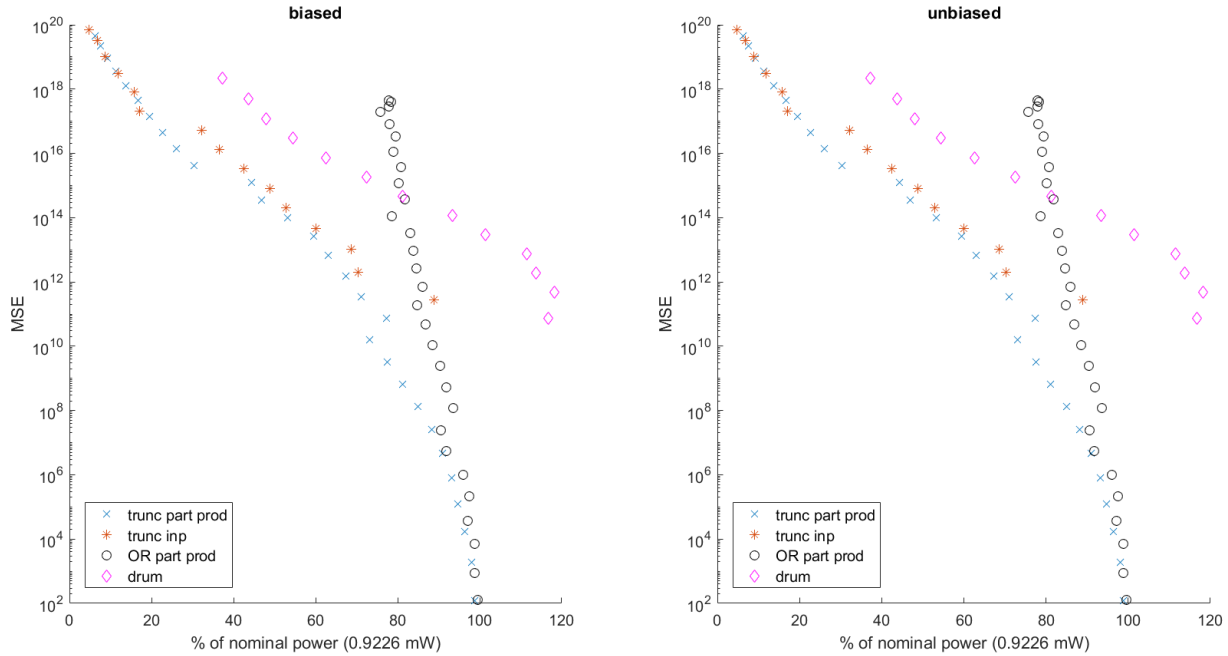

biased
unbiased
MSE
% of nominal power (0.9478 mW)
trunc part prod
trunc inp
OR part prod
drum

c) Stefcal_CE

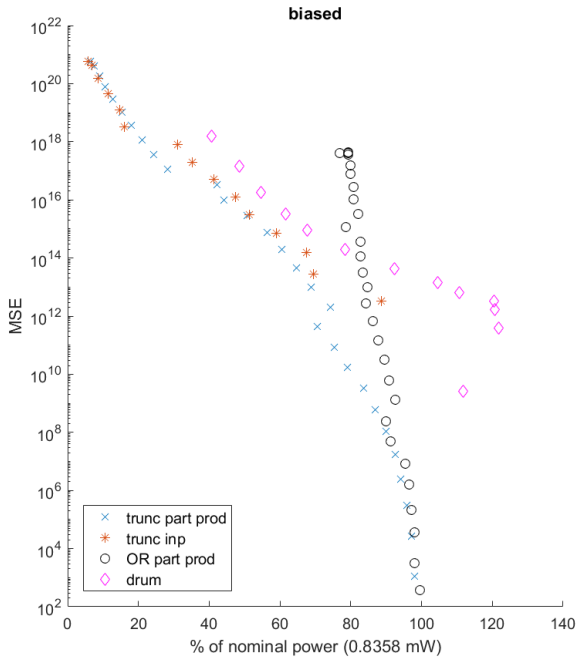
16x16. Signed magnitude.
Synthesized at 250 MHz.
compile -map_effort high; compile_ultra

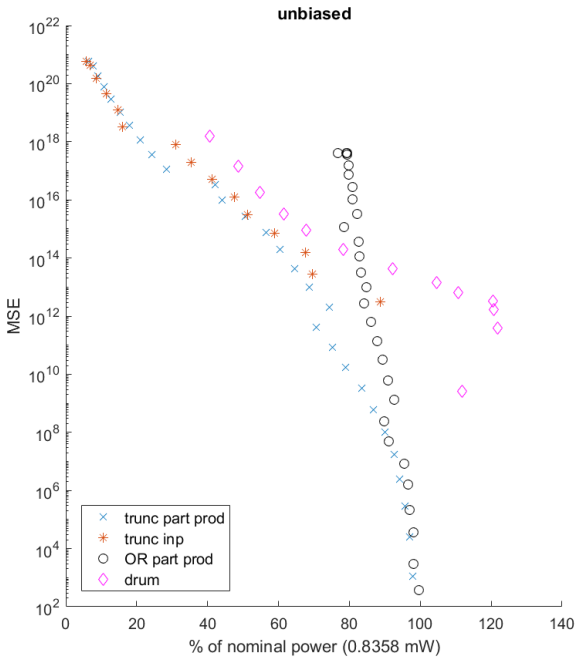


a) uniform



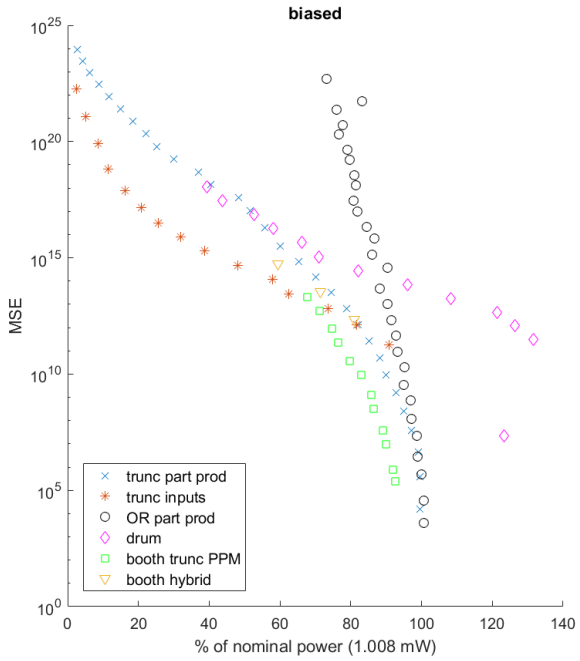
b) normal with mean=0 and std=20480

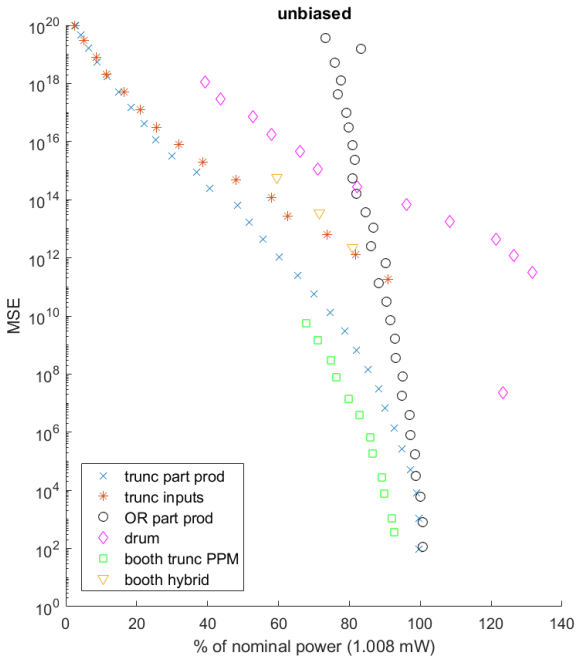

biased
MSE
% of nominal power (0.8358 mW)
trunc part prod
trunc inp
OR part prod
drum


unbiased
MSE
% of nominal power (0.8358 mW)
trunc part prod
trunc inp
OR part prod
drum

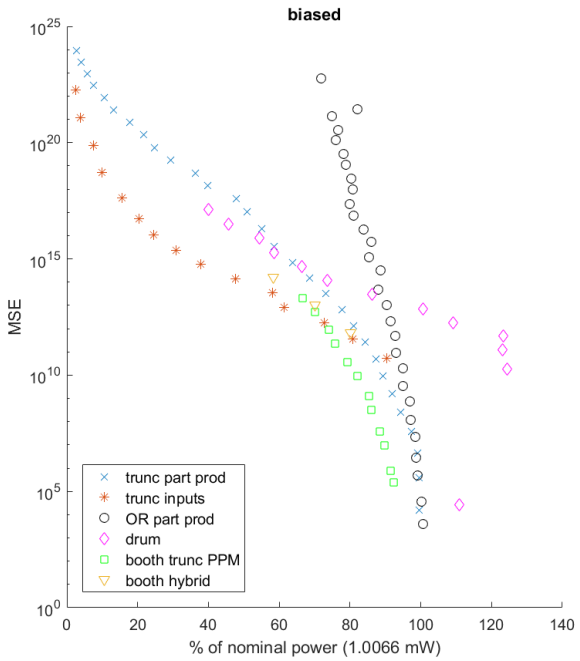
c) Stefcal_ET

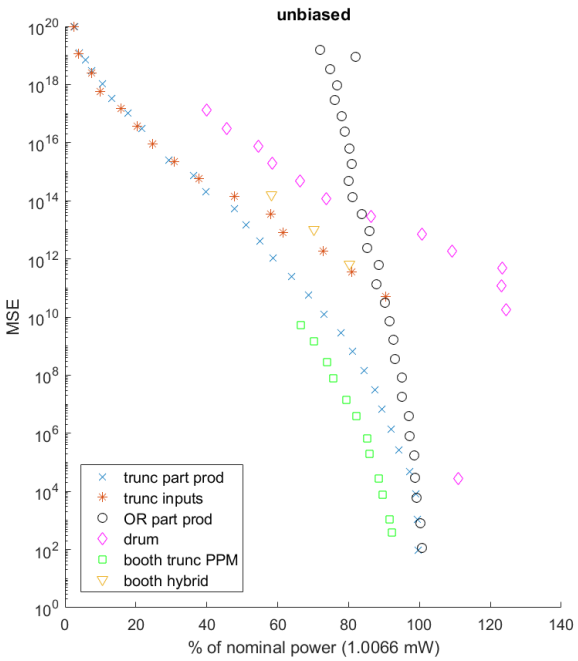
16x16. 2's complement.
Synthesized at 250 MHz.
compile -map_effort high; compile_ultra



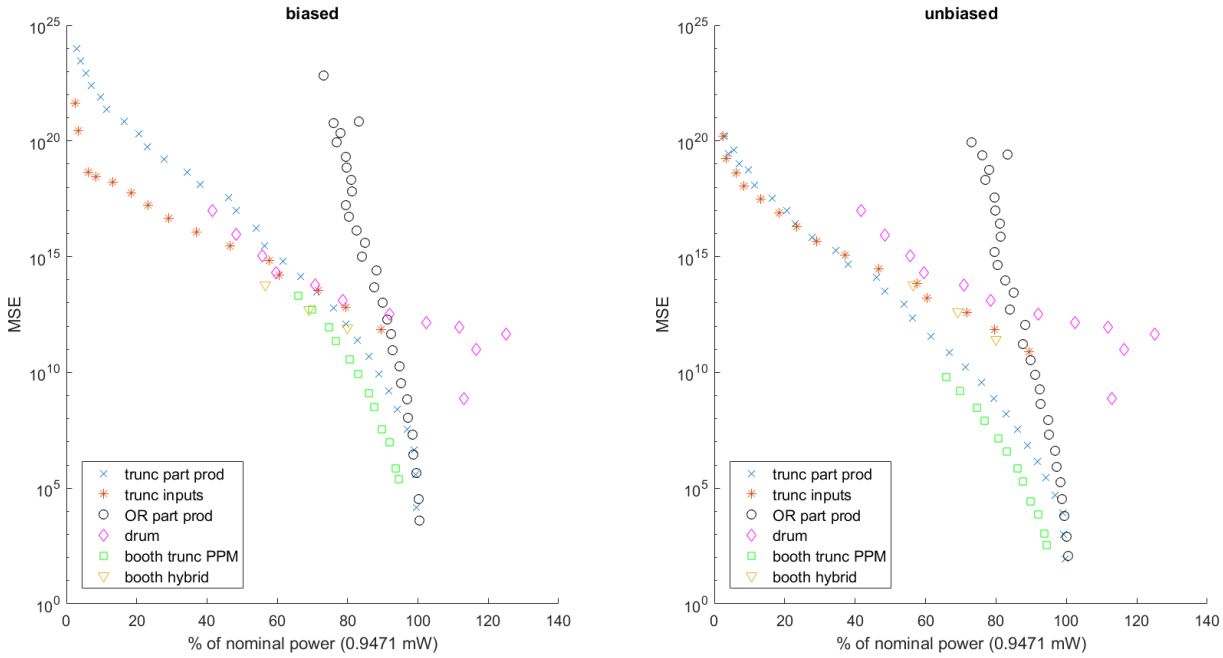


a) uniform





b) normal with mean=0 and std=10240


biased
unbiased
MSE
% of nominal power (0.9471 mW)
trunc part prod
trunc inputs
OR part prod
drum
booth trunc PPM
booth hybrid

c) Stefcal_ET