UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Approximate arithmetic units under voltage under-scaling

Master Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Imanol Etxezarreta Martinez

In partial fulfillment
of the requirements for the master in
ELECTRONIC **ENGINEERING**

Advisor: Francesc Moll Echeto
Barcelona, Date 28/01/2021

Contents

List of Figures

List of Tables

Revision history and approval record

Revision	Date	Purpose
0	28/07/2020	Document creation
1	26/01/2021	Document revision
2	28/01/2021	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Imanol Etxezarreta Martinez	imanol.etxezarreta@estudiantat.upc.edu
Dr. Francesc Moll Echeto	francesc.moll@upc.edu

Written by:		Reviewed and approved by:	
Date	27/01/2021	Date	28/01/2021
Name	Imanol Etxezarreta Martinez	Name	Dr. Francesc Moll Echeto
Position	Project Author	Position	Project Supervisor

Abstract

Increasingly power hungry processors and the identification of error tolerant applications have made approximate computing techniques one of the researching scopes of the present. The main objective of this work is to study and assess the benefits of certain approximate arithmetic units for their inclusion in the DRAC project's approximate accelerator.
In order to accomplish this objective, different approximate and accurate adders and multipliers are implemented in *Verilog*, synthesized and simulated using *Cadence* tools and finally compared between them to evaluate whether they offer any benefits or not. Simulations are performed for both *Overclocking* and *Voltage Underscaling* conditions.
The results indicated that some of the approximate adders show very competitive precision values at overclocking simulations for 400 ps, with slightly better power results, and other adders also presented very encouraging results for voltage overscaling simulations, offering higher precision values than the accurate adders for 0.4V. In contrary, the multipliers exhibited worse results than the accurate ones in every aspects for the realistic case dataset provided by the SW team.

Acknowledgements

Firstly, I would like to express my gratitude to the UPC and BSC who offered me the opportunity and the necessary background and knowledge to develop this master thesis. In particular, I would like to thank the Work Package 4 team, conformed by UPC research professors and BSC researchers.

I would also like to extend my gratitude feelings to my master thesis advisor, Francesc Moll, who has been supporting the study the whole time, offering his expertise in the field of research when the progress of the study seemed stuck, and recommending new ambitious research directions.

Finally, special thanks to my family and friends who have always supported and encouraged my work, through any means they could, and have listened and tried to advise me even in themes where they were not experts. I would like to make some special mentions, starting from my close Catalan friend, Pau, for the hours spent writing our theses supporting each other and to my best friend Ion for his big support. And to finish, very special thanks to my parents and brother; Fernando, Beatriz and Josu, and to my girlfriend Mireia for always making me feel loved and supported and for having to deal with me in the tough moments in the development of this work.

”This project is co-financed by the European Union Regional Development Fund within the framework of the ERDF Operational Program of Catalonia 2014-2020 with a grant of 50% of total cost eligible”

Generalitat de Catalunya
Departament d’Empresa i Coneixement
Secretaria d’Universitats i Recerca

Unió Europea
Fons Europeu
de Desenvolupament Regional

1 Introduction

In order to understand the motivation behind the study of the approximate arithmetic and the development of this document, some background is mandatory. The context of this work is the design of accelerators based on the RISC-V processor. It all has its beginning in 2010, when RISC-V is originated at the University of California, Berkeley. Nowadays, it is supported by RISC-V International and it has numerous members all along the globe. The public release of this RISC-V unlocked many many possibilities and facilities to implement a processor, and thus this point on, many processor implementations based on this RISC-V has been developed in both scopes, the industry and the research.

The work developed in this study is made as part of the DRAC (**D**esigning **R**ISC-V-based **A**ccelerators for next generation **C**omputers) project, conformed by the institutions of the BSC, CIC-IPN, IMB-CNM (CSIC) and the UPC.

One those next generation accelerators is intended for applications where the 100% of accuracy is not needed. In other words, there is a project which consist on developing an accelerator whose scope is the boosting of calculations for applications which are error tolerant. For the development of this accelerator, approximate computing is extensively researched in its multiple forms and sources. The accelerator itself will consist on a Deep Neural Network (DNN) which is approximate by definition. But not satisfied with this, other approximation methods are also being researched. From this context is from where the motivation for the approximate arithmetic units is born, and as another goal to achieve with the insertion of approximate computing techniques is the reduction of the power consumption, the study focuses on the behaviour of some approximate arithmetic units under the effects of voltage underscaling.

Voltage underscaling, or voltage reduction, is a technique where the supply voltage of a design is decreased far below the nominal voltage. By the use of this method, the total power consumption of the design can be drastically reduced but this usually comes with the incorrect functioning of the design. This happens because other effect of lowering the supply voltage is the slower saturation of the CMOS transistors, and this means timing errors will appear. This is why approximate arithmetic units can be useful under this circumstances in front of the conventional ones.

This document compiles the work performed in order to accomplish the objectives and requirements established concerning approximate computing and explains the different methods and troubles that have been faced during the elaboration of it so that future similar problems are solved faster. All the objectives, requirements and methods are briefly explained in the following subsections 1.1, 1.2 and 1.3, and more deeply explained in throughout the whole document.

1.1 Statement of purpose (objectives)

The main objectives proposed to be fulfilled during the development of the thesis are listed below:

1. Get familiar with state-of-the-art approximate arithmetic architectures, understand their basics and implement them in verilog.
2. Evaluate whether the studied approximate arithmetic units offer considerable/valuable performance increase under voltage underscaling conditions, and assess if they would be beneficial for the DRAC project.

1.2 Requirements and specifications

As requirements, the correct implementation, synthesis, simulation, error study, power consumption computation and precision degrading under voltage underscaling of the following adders and multipliers must be performed.

- **Adders:**
  - Ripple-Carry accurate adder
  - Kogge-Stone accurate adder
  - Accurate adder using '+' sign
  - ACAI_Gen approximate adder
  - ACAI_RCA approximate adder
  - ACAII_Gen approximate adder
  - ACAII_RCA approximate adder
  - GeAr2_Gen approximate adder
  - GeAr2_RCA approximate adder
  - GeAr4_Gen approximate adder
  - GeAr4_RCA approximate adder
  - GeAr6_Gen approximate adder
  - GeAr6_RCA approximate adder
- **Multipliers:**
  - Mult16 accurate multiplier
  - Accurate multiplier using '*' sign
  - UDM16 approximate multiplier
  - UDM16_SFA approximate multiplier
  - UDM16_SFA_SW approx multiplier

1.3 Methods and procedures

About the methods and procedures used in the development of the thesis, the implementations of the arithmetic units are done using *Verilog*, while the synthesis, simulation, power computation and timing analysis are done using the tools provided by *CADENCE*, such as *Genus*, *Incisive*, *Joules* and *Tempus*.

The arithmetic units under study are synthesized in GlobalFoundries 22FDX technology, a 22nm FDSOI. This is the technology of choice to implement the chips in the DRAC project. There are several libraries for this technology containing hundreds of logic gates characterized in several operating conditions. The main library used for synthesis processes is characterized for the next operating conditions.

- V_{DD} = 0.80V
- V_{SS} = 0V
- V_{BS} = 0V
- T = 25ºC
- Typical transistor

While, throughout the study, another library is used to make extrapolations of different values, which is characterized for the following operating conditions.

- V_{DD} = 0.65V
- V_{SS} = 0V
- V_{BS} = 0V
- T = 25ºC
- Typical transistor

2 State of the art

2.1 Approximate Computing

As mentioned before, Approximate computing is known as the set of techniques that reside in the error tolerance of certain applications in order to further improve the performance of the system. For example, on image processing the quality of the image is evaluated through human perspective, which is subjective, and typical error-resilient image processing algorithms can accept certain quantity of errors without the users noticing it.

Approximation can be achieved from multiple sources and at different levels in the system. Three different levels can be distinguished at which approximation can be applied and inside each level different techniques can be used to introduce the approximation, having this way multiple approximation sources [1]. For example, in the following, the most used ad discussed approximation methods are listed.

1. **Software**
    - Function Skipping
    - Memoization
    - Loop-Perforation
    - Functional Approximation
    - Read/Write Memory Approximation
    - Data Precision Reduction
2. **Architectural**
    - Use of Neural Networks
    - Memory Access Skipping
3. **Circuit and Hardware**
    - Voltage Scaling
    - Inexact or faulty hardware

Even if the previous methods are classified into three different levels, some of them can be implemented in more than one. For example, functional approximation, read/write memory approximation and data precision reduction could also be applied at architectural levels. In order to have an overall idea, some of these techniques are going to be briefly explained.

Loop-perforation, for example, is an approximation technique that resides on stopping a loop before all executions are done because an acceptable result is already obtained. The impact of this technique at software level mainly affects the execution time of the application, while the implementation on a FPGA could also affect the energy and area consumption.

Data Precision Reduction consist, as its name says, reducing the data precision so that the memory footprint is reduced and thus power consumption is reduced in exchange of accuracy.

Neural Networks are capable learning how a standard function implementation behaves in relation to different inputs. Traditional approximate codes can be transformed into neural network that may have worse accuracy but better performance.

These are some of the approximate computing most common techniques, but in this work, the hardware level techniques are going to be studied. Both *inexact/faulty hardware*, *Voltage Scaling (VS)* and the combination of those two are going to be tested and simulated. Being VS just a reduction of the voltage, in the next subsection 2.2 the state of the art of the studied approximate arithmetic is explained.

2.2 Approximate Arithmetic

Approximate arithmetic as inexact hardware logic can be implemented at the architectural level to introduce the approximation. Majority of the approximations on arithmetic units consist on reducing the critical path by just truncating it or even eliminating some of the internal logic in this longest path.

As far as approximate arithmetic is concerned, almost every arithmetic operations have been studied and tried to be approximated. For example, addressing the addition, plenty of approximate adder proposals can be found in the literature until this date. But this may not be the same for multiplication, subtraction, division or even comparison. As mentioned already, this study focuses mainly on 16-bit wide approximate adders and multipliers. With this in mind, the following subsubsections 2.2.1 and 2.2.2 present the state of the art for approximate adders and multipliers respectively.

2.2.1 Adders

Approximate adders are the most studied by the approximate arithmetic research community, and plenty of different methods have been used in order to insert the approximation to the cell. There are different approximation sources such as eliminating transistors at transistor level or layout level of the basic Full Adder cell or at architectural level by truncating the carry propagation (which also happens to be the critical path in adders) by splitting the addition into smaller sub-additions and not propagating the carry.

All these methods offer different benefits and also counterparts, for example the last method may speed up the computation, resulting in a larger slack, but it also may require bigger area and worse accuracy. The methodology followed for the study of these benefits/counterparts is explained in Section 3 and the results for each adder is presented in Section 4.

The approximate adders that will be studied are the following:

1. Almost Correct Adder (ACA - I)
2. Accuracy Configurable Adder (ACA - II)
3. Generic Accuracy Configurable Adder (GeAr)

Almost Correct Adder (ACA-I) This type of adder resides in splitting the sum in smaller sub-adders whose width is determined by the longest sequence of propagate signals. It is obvious that the longest sequence is the propagation of all the bits, but it is very unlikely to happen, and the study made in [2] demonstrates that on average, the length of the longest propagate sequence is approximately *log n*, where *n* is the bit width of the integers. In addition, as the carry output is dependent to the previous bit to the start of the propagate sequence, 2 more bits must be also considered.

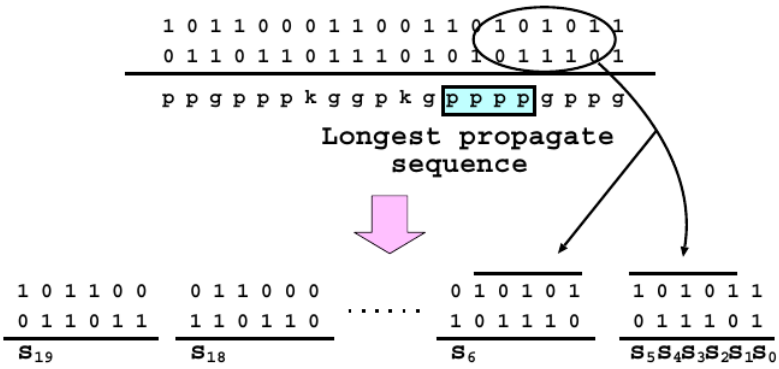

Figure 1: Example: 20-bit integers, longest propagation sequence is 4 and sum split into 6-bit sub-adders [2]

Accuracy Configurable Adder (ACA-II) Once again, this approximate adder cuts the carry chain to reduce the critical path delay and some sub-adders generate results for partial summations. The main idea behind this adder is to insert one additional sub-adder than the needed, in order to increase the accuracy but still have low power consumption [3]. So, for example, when adding two 16-bit integers the summation could be split into two sums of 8-bit sub-adders, but this means that when the eighth carry bit of the first sub-adder is high, an error occurs. But, if one additional 8-bit sub-adder is inserted in the middle and only the 4 MSB of the results are taken, then the error only occurs when one of the carries have to be propagated and the 4 LSB additions of the next sub-adder results in 1111.

Figure 2 shows an example of a 16-bit ACA-II adder. It is also worth noting that the LSB sub-adder is an accurate sub-adder and this is why all the 8 bits of the addition are considered in the final output.

Generic Accuracy Configurable Adder (GeAr) Finally, the Generic Accuracy Configurable adder, born from the necessity for a unified configurable adder, which can provide a wide range of configurability and approximation modes along with error detection and correction capabilities, as stated by the researchers that developed this last adder [4]. The adder has different parameters: N, length of operands to be added; L, length of the sub-adders in parallel to perform the approximate addition ($L \leq N$); R, represents the number of resultant bits contributing to the final sum; and P, represents the number of previous bits used for carry prediction. Each sub-adder produces R-bit results except the first one that produces L-bit result being $L = R + P$.

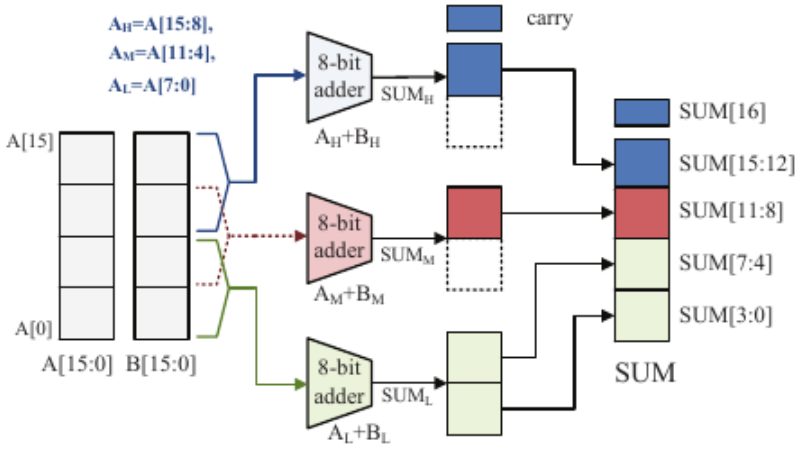

Figure 2: Example of 16-bit ACA-II adder [3]

The approximate adder is fully defined with the three parameters N, R, and P. With these parameters every aspect of the adder can be defined, as for example how many sub-adders k will be needed.

$$k = \left(\frac{N-L}{R}\right) + 1 \tag{1}$$

And also, with those parameters a generic implementation of N-bit GeAr adder can be described in two simple equations shown in Equations (2) and (3). The first one, Equation (2), represents the first sub-adder of the whole approximate adder, which is an exact sum and thus all the resulting bits are contributing to the final result. The second one, Equation (3); however, describes the resulting bits that contribute to the final result, R, adding $L = R + P$ bits, which means that previous P-bits are used to predict the carry.

$$Sum[L-1:0] = A[L-1:0] + B[L-1:0] \tag{2}$$

$$Sum[(R\times i+P-1:R\times(i-1)+P)] = A[(R\times i)+P-1:R\times(i-1)] + B[(R\times i)+P-1:R\times(i-1)] \tag{3}$$

where $1 < i \leq k$.

As this adder was developed from the necessity of having a generic and configurable approximate adder, with different configurations of the previously explained parameters (N, R, P), the State-of-the-Art approximate adders can be implemented. For example, the already explained adders can be obtained configuring the parameters in some particular way. The ACA-I [2] can be made setting $R = 1$ and $P = L - 1$, while the ACA-II [3] is achieved with $R = L/2$ and $P = L/2$.

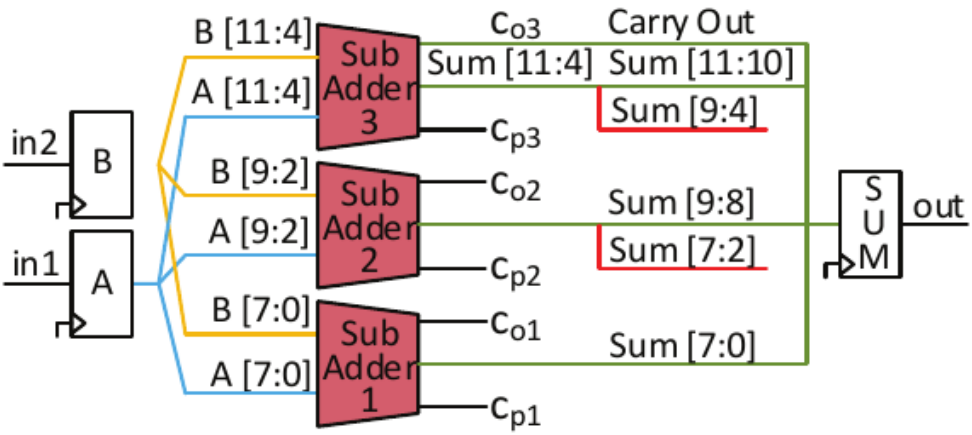

Figure 3: Example of N=12, R=2 and P=6 GeAr adder [4]

Simplified Full Adder (SFA) In addition to the previous adders, there is still another one that will be used in the partial summations of the multipliers and is worth explaining it. The adder consist on a re-design of the Ripple-Carry Adder (RCA) made in [5]. The RCA has two critical paths that start from any of the two primary inputs, and replacing any logic gate by logic 1 or 0 can truncate the critical path and increase the slack.

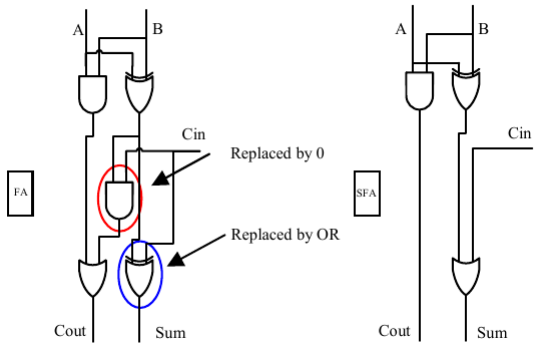

Figure 4: Approximation in full adder cell to obtain SFA [5]

A B C_{in}	Correctness
0 0 0	No error
0 0 1	No error
0 1 0	No error
0 1 1	Error
1 0 0	No error
1 0 1	Error
1 1 0	No error
1 1 1	No error

Figure 5: SFA erroneous operation [5]

The main change is the replacement of the AND gate for a logical 0 to break the critical path. Having the output of this AND gate to 0 makes that the output of the next OR gate will depend only on the other input, so this OR gate can also be removed. In addition, the XOR gate is replaced by an OR gate in order to reduce the Error Significance (ES) while maintaining the Error Rate (ER) constant.

As the SFA is a simplification or a modification of a full adder (FA) cell, the idea is to reduce the critical path in an addition of two numbers with bigger bit widths than just 1. So, the idea is to introduce the SFA cell in between other FA cells to truncate the critical path and make the addition faster in exchange of some accuracy.

2.2.2 Multipliers

Approximate multipliers, unlike approximate adders, have not been so extensively researched although this does not mean that they have not been studied. Even if there are not so much articles concerning inexact multiplication, there are some researches that have proved to be very effective, and in addition, having the possibility of introducing the approximation through a faulty adder in the partial product addition, also extends the possibilities for these arithmetic units. In this paragraph, the approximate multipliers used by the DRAC Workpackage 4 Software team are presented.

Under Designed Multiplier (UDM) One of the best known research concerning approximate multipliers is Kulkarni's [6] Under Designed Multiplier (UDM). This approximation resides in the modification of the Karnaugh Map of a 2x2 multiplier. The trick is to represent the outcome of the multiplication using only three bits instead of the normal four. As the fourth bit is only used in one possibility $(3*3)$, the output of this operation is set to be 111 in contrast to the exact 1001. With this change, the multiplication is correct for fifteen out of sixteen possible inputs and the error occur with a magnitude of $(9-7)=2$ and $1/16$ probability.

A_1A_0 \ B_1B_0	00	01	11	10
00	000	000	000	000
01	000	001	011	010
11	000	011	111	110
10	000	010	110	100

Figure 6: Modified Karnaugh Map from [6]

In addition, even larger multipliers can be built using the 2x2 block explained before to produce the partial products and then adding the shifted partial products. This, can also be expanded to obtain even larger bit width multipliers. Figure 7 shows an example of a 4x4 multiplier which is built out of four 2x2 blocks.

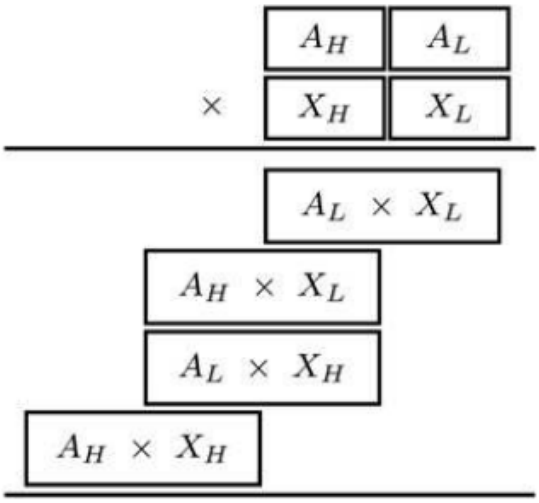

Figure 7: Example of 4x4 multiplier using 2x2 blocks from [6]

Finally, about the error rates of this multiplier, the computation of the simple 2x2 block is very easy as it only has 1 erroneous output from 16 possibilities so the error probability is equal to $1/16$ with a maximum error magnitude of 22.22%. But the computation of these factors for larger bit width multipliers gets more complicated, so from [6] is obtained that the error probability increases with increasing bit-width but the mean-error increases steadily to finally saturate between 3.3% and 3.5%.

Error Tolerant Multiplier (ETM) This last approximate multiplier [7] consist on splitting the multiplication process in two, and treating differently the MSB and the LSB. The MSB are part of the multiplication part while the LSB are part of the non-multiplication part. The length of each part is not need to be equal.

The multiplication part, is just a normal conventional multiplication process from right to left, and this is an accurate multiplication due to the fact that the higher bits have greater weight than the lower ones.

The non-multiplication part; however, starts from leftmost bits of the lower bits and goes all the way to the rightmost ones. Then, the algorithm states that both input bits will be checked and if at any point at least one of the checked input bits happen to be equal to 1, then the algorithm is stopped there and the resulting bits to le right of this position are set all to 1. Figure 8 show both multiplication and non-multiplication parts, and how this last algorithm works.

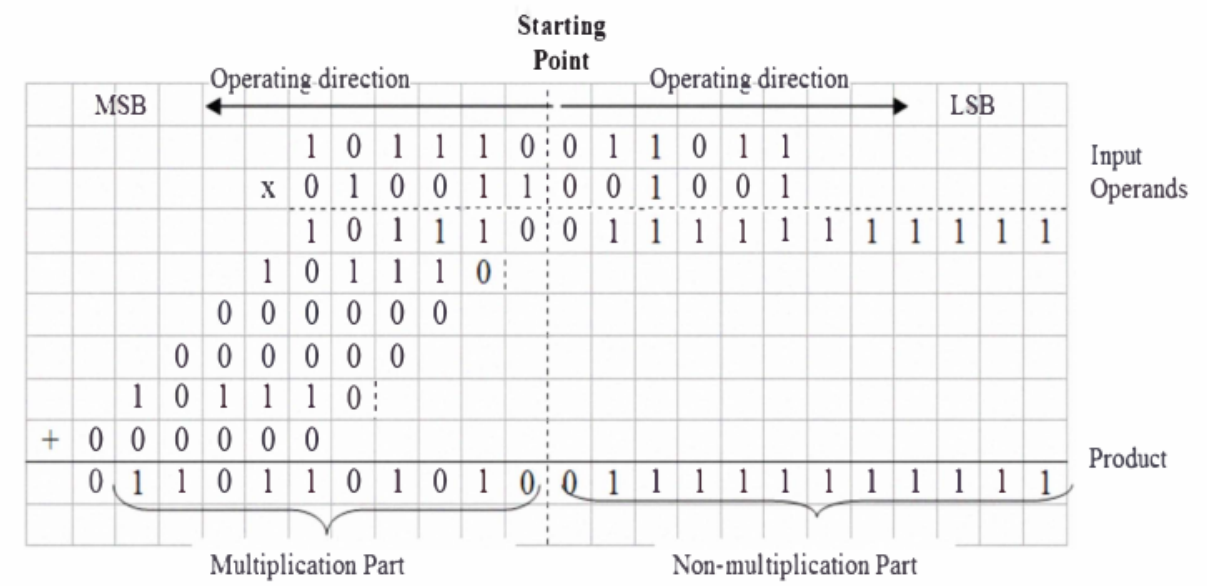

Figure 8: Example of 12-bit input operand multiplication with ETM from [7]

3 Methodology / project development

In this section of the work, the methodology applied for the development of the study is explained. First, the general flow followed for every adder and multiplier is going to be presented and then the steps of this flow are going to be expanded.

3.1 General flow

The general flow is a main scheme that has been established in order to have some guidance when starting this particular research. Normally every studied adder and multiplier follows the flow until the final results are obtained, and it is concluded that they are coherent. There may be some cases were the flow changes at some point due to any unexpected result or any error. But, in general, the flow employed was as follows:

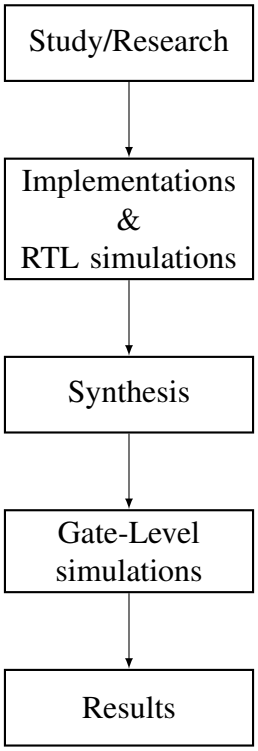

Figure 9: General Flow block diagram

As explained above, the main work flow that has been followed is illustrated in Figure 9, but there are other reasons for which the work flow was modified for certain architectures, simulations, errors or results. If these modifications have strong significance in the specific scenario, they are going to be explained in the corresponding flow step were the change was made.

Apart from the exceptions, the main and most important flow parts are going to be explained with more detail in the next Subsections 3.2, 3.3 and 3.4 of this Section. With the exception

of the first flow step, which is *Study/Research*, that can be considered already explained in the State of the Art, Section 2; what is done in every step, why and how it is done is going to be presented.

3.2 Implementations

Once the research about the desired adder is done, and the basic structure and functionality is understood, the next step is the RTL implementation. Being this a study aimed for the DRAC project, the implementation language must be equal to the one used in the project, therefore the used Hardware Description Language (HDL) is constrained to **Verilog**.

Before diving into the insides of each adder and multiplier's implementation, the different approximate designs and the chosen configurations for them are listed and explained below. The adders are presented first and finally the multiplier configurations are shown.

3.2.1 Adder implementations

First of all, the adders are **16-bit integer additions** and they have registers in both inputs and the outputs. The chosen approximate adders and their configurations are listed below.

- ACA-I (same as GeAr(N=16, R=1, P=7))
- ACA-II (same as GeAr(16, 4, 4))
- GeAr(16, 2, 8) (named as GeAr2)
- GeAr(16, 4, 8) (named as GeAr4)
- GeAr(16, 6, 8) (named as GeAr6)

Both ACA-I and ACA-II are implemented following their specifications, which as explained in Section 2, can be defined also as GeAr(N, R, P) adder and it would result in the same implementation. Concerning the GeAr adders, their configuration is kept equal but only the *R* (sub-adders output bits that are used actually in the final output) parameter is changed. This way the impact of varying this parameter can be assessed.

In addition, internal operations of the whole addition in these approximate adders is divided in sub-adders. As the sub-adders can be defined also in different ways, every approximate adder configuration has two versions of it but with a unique difference in the description of the sub-adders. The first version lets the synthesis tool the optimisation of the sub-adders by describing these with the + addition sign. In contrary, the second version uses an accurate adder for the sub-adders which is a Ripple-Carry Adder (RCA), and this fixes the architecture that will be synthesized. So in the end, concerning the approximate adders, the following versions are going to be studied:

- **Final approximate adder versions:**
  - ACAI_Gen
  - ACAII_Gen
  - GeAr2_Gen
  - GeAr4_Gen
  - GeAr6_Gen
  - ACAI_RCA
  - ACAII_RCA
  - GeAr2_RCA
  - GeAr4_RCA
  - GeAr6_RCA

It is worth explaining that the scope of the study is to show whether the approximation of the arithmetic units offer advantages or not in front of the conventional fully accurate adders. Having this in mind, some **traditional accurate adders** must be implemented too for the sake of comparison. Three different accurate adder structures have been chosen for this purpose:

- Ripple-Carry Adder (RCA)
- Kogge-Stone Adder (KSA)
- Adder structure optimized by the synthesizer using + sign as addition (GenusOpt)

As a brief explanation about the accurate adders, the RCA is the traditional adder were the carry is propagated from the *Cout* of the full-adder module to the next FA's *Cin*, and it is implemented with 16 FA. The KSA is a parallel prefix form of carry look-ahead adder, which has a tree shaped structure. The Kooge-Stone concept was first developed in 1973 by Peter M. Kogge and Harold S. Stone [8]. It is one of the most quicker adder structures and it is widely used nowadays in the industry. Finally, the last adder is simply described as $sum = A + B$ and the structure, optimization and every other aspects are not constrained for the synthesizer program to develop and find the best solution.

Once the implementations were finished, an exhaustive sequential RTL simulation was performed to compute every possible addition. This was made in the simple form of 2 *for* loops, one inside of the other, that went from 0 to 65535 and thus performing a simulation of 65535×65535. With these simulations, the proper functionality of the accurate adders was ensured and it was also obtained the inherent errors that the approximate adders offered. These results will also be presented in the Section 4.

3.2.2 Multiplier implementations

When multiplier implementations are concerned, there are not so many implementations and there are some constraints about their implementations. As already stated, this study is for the DRAC project, more specifically for the Work Package 4 about approximate computing. With this in mind, the software (SW) team tested the Neural Network (NN) they are developing using some specific multipliers. The multiplication in this NN are 16 bit Floating Point (FP) or half-precision FP, which means that the mantissa part of those 16 bits is what is going to be multiplied.

At first the multiplication should be of 11 bits, but due to some data handling on the approximate library used by SW team (SoftFloat), the multiplication is done for 16 bits. This is because the 11 bits of the mantissa are shifted to the left in such a way that the first operand ends with four additional 0 as LSB and the second operand with five 0s as LSB. Assuming the first operand has one additional 0 as the MSB, then both of them have 16 bits and the multiplication is done for this new values.

Thus, the multiplications that are going to be studied in this document correspond to **integer multiplications of 16 bits** in order to recreate the results obtained by the SW team. But, as in principle the performed shifting is not necessary in the hardware domain, in the future the multiplier is going to be simplified to 11-bit multiplication that later should be used in a higher level FP multiplier with its corresponding exception handling as NaN (Not a Number) or infinities.

The implemented multipliers are listed in the following scheme:

- *Under Designed Multiplier* (UDM).
- *Under Designed Multiplier using SFA* to perform partial summations (UDM_SFA).
- *Under Designed Multiplier using SFA* to perform partial summations but *following same structure as in SW* without RTL optimization (UDM_SFA_SW).
- Accurate multiplier using the same structure as in the UDM but using accurate 2x2 multiplier blocks. (Mult16)
- Accurate multiplier using the '$*$' sign and letting the synthesizer tool to optimize the design. (GenusMult16)

Apart from the UDM multiplier, another one was also studied by the SW team known as the Error Tolerant Multiplier (ETM), but due to the results they obtained were not so promising and the lack of time this multiplier structure is not going to be studied in this document, but could be left for future examination.

The UDM multiplier, as explained in the Section 2, it is a 2x2 block that returns a 3-bit answer instead of 4-bits. This block can be used to build larger multipliers and it is exactly what is been done for this study. The SW team provided a general diagram of the multiplier's structure, Figure 10, and this is what it has been followed as a base to implement the different multiplier variations.

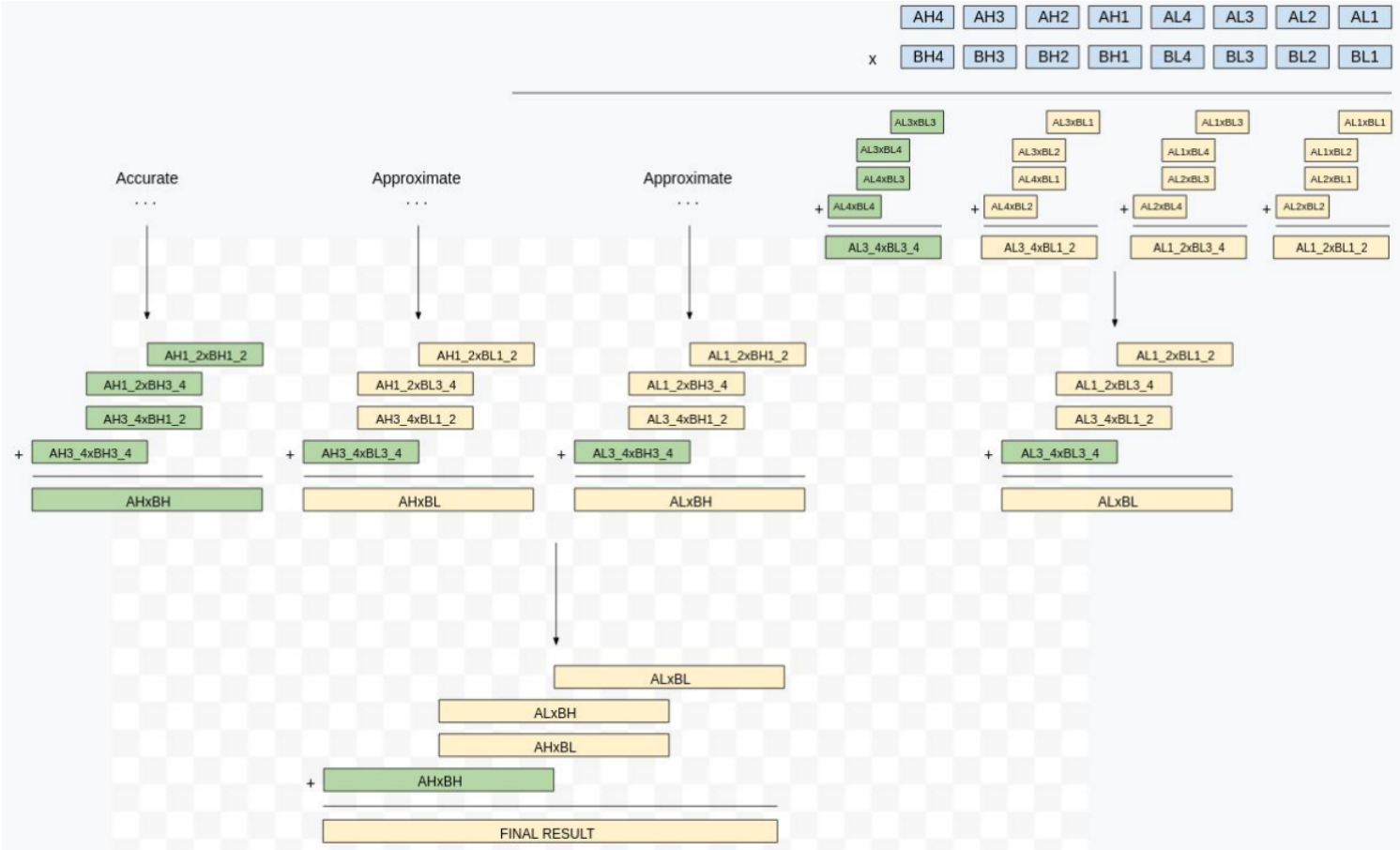

Figure 10: 16-bit UDM structure provided by SW team (Martí Caro, Jaume Abella and Hamid Tabani)

As it can be observed in the structure, the two 16-bit integer numbers are split into little blocks of 2-bits, and then these blocks are multiplied with each others using the 2x2 UDM block. The implementation of the multiplier is done from the final result that can be seen down in the diagram to the top 2x2 multiplications. The multiplications can be differentiated into 4 major blocks which are: AHxBH, AHxBL, ALxBH and ALxBL. Adding these four 16-bit blocks, with their respective shifting observed in the diaggram, the final result is achieved. As AHxBH contains the MSB in the multiplying numbers, the whole process is set to be accurate, which means that the additions and even the 2x2 multiplications are performed using accurate logic. Concerning the rest three blocks, one is going to be explained in more detail because they contain additional approximate logic, but the general structure of the four blocks is the same (partial addition structures, shifting...).

Continuing up in the diagram following the ALxBL path, in order to obtain this result, four 8-bit second level partial additions need to be added in the same way as before. Again, the 8-bit block containing the MSB is computed using solely accurate logic in order to preserve precision. Again, in order to obtain these 8-bit blocks, the results of the partial multiplications are added with their respective shifting. The partial products are obtained performing the 2x2 multiplication and for the MSB block, accurate multiplication is used while for the other three

the UDM block is employed.

This internal structure, even if the diagram does not show it due to redundancy and for the sake of visibility, is used in every major blocks, regardless of being accurate or approximate, the only difference is the usage of approximate logic.

Finally, the described structure is used in every multiplier implemented with the exception of the *GenusMult16* multiplier that is completely free for the synthesizer to optimize. So, the diagram represents the implementation for the *UDM_SFA_SW* multiplier, where both approximation sources (UDM 2x2 block and the SFA cell) are employed. The UDM are used to generate partial products and the SFA cells are located in the middle bit of the corresponding intermediate addition. The case of the *UDM_SFA* multiplier is similar, but the intermediate additions are simplified. For example, in every addition, the LSB directly go to the result instead of adding them with 0s. And the adders used are smaller, thus smaller area and smaller carry length. The *UDM* multiplier only inserts the approximate multiplying blocks maintaining the intermediate additions with accurate adders, and the *Mult16* multiplier uses accurate logic in the whole structure.

Once again, before starting any synthesis process and to ensure the correct implementation of the multipliers, an RTL simulation was performed for each of them. In the accurate multipliers case, a Python program generated $2^{11} - 1 \times 2^{11} - 1$ random pair values without repeating any pair in order to do an exhaustive checking on them. About the approximate multipliers; in contrast, this would not be enough to ensure whether the implementations were correct or not, because they have some fixed errors on them and it would be impossible to determine if the resulting errors from the simulations were due to the approximate logic or due to some implementation error. To solve this, the SW team provided a file with 10 million pair of operands and their respective results, and obtaining 0 errors simulating these inputs and results means that the approximate adder is correctly implemented. To be exact, two different files were provided one for *UDM* multiplier with only 1 source of approximation on the multiplication and another file for *UDM_SFA* and *UDM_SFA_SW* multipliers with two sources of approximation on the multiplications and the partial product tree additions.

3.3 Synthesis

Once the RTL implementations are done and their correct operation is ensured by the simulations, the next step in the flow is to synthesize these designs to an actual gate level netlist with technology libraries and existing cells. As the synthesis tool, *Genus Synthesis* tool from *Cadence* has been used throughout the whole study to synthesize every design implemented.

The SW team specified that the accuracy achieved in the DNN using half-precision (FP16) data types was good enough and the designs were modified to be 16-bit width operations. The technology to be used is Globalfonudries 22FDX, as mentioned above. The synthesized general design is depicted in the Figure 11, where the combinational logic in our case is the adder or multiplier under study.

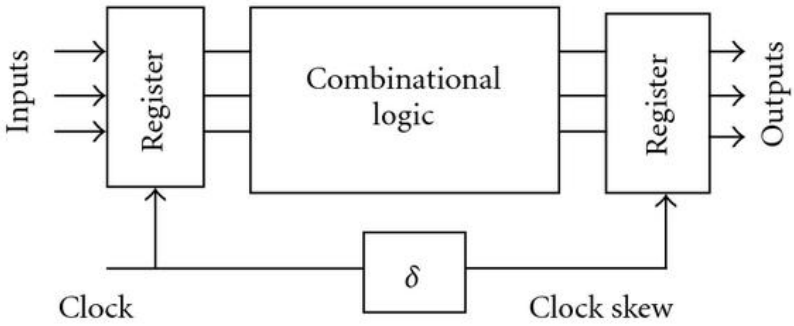

Figure 11: Synthesized general design [9]

As it can be seen, the arithmetic unit to be studied is placed between two registers (input and output registers). This is necessary for the synthesis tool to be able to determine the slack value of the design, and the reason to implement the designs in such a way.

The results obtained form the synthesis process are presented and discussed in Section 4 Results, and in this section a little explanation about the followed synthesis flow and commands is performed in order to have a general idea of how the gate level netlists are generated and what configurations or options have been chosen for these processes.

First of all, in general, all the designs are synthesised for a period of 10 ns but for the voltage underscaling simulations, the adders are synthesized with a period equal to 1 ns and the multipliers for a period of 2 ns. This is done because as it is going to be presented in the Section 4, the obtained slacks for 10 ns are very large.

The procedure to synthesize any design starts from a *Makefile* where the synthesize type is chosen. This *Makefile* then launches a Shell script that calls the synthesis tool, *Genus*, and passes as an argument the corresponding *Tool Command Language*, also named as *Tcl*, (.tcl) file with all the synthesis instructions. Genus works with Tcl scripts, and it has its own commands that can be found in the corresponding Cadence documentation. The first thing that is done in the Tcl script is the variable setup, for example, for convenience the desired adder from the 13 possibilities is asked to the user and the variables are set to synthesize the selected one. Thank to this, a simple script can be used used to synthesize all the 13 possibilities in a row. Apart from this, for general file management, output directories are created and in order to further extend the possibilities of automatizing the synthesis process, the chosen period is read from the Constraints files and used for the output paths.

3.4 Simulations

Next step in the flow are the Gate-Level Simulations (GLS), which are done using the *Cadence* tool *Xcellium*. This step is very important because there are a lot of different simulation types and the results may vary depending on the simulation configuration and it have a lot of beneficial effects [10]. Among the reasons that show why the GLS are important, three are specially important in this study:

1. Checking final performance to ensure that the design works at the desired frequency, with actual delays in place. Which includes structural elements not checked by Static Timing Analysing or formal engines, and buffers or other elements added during synthesis and place and route.
2. Capturing switching factors for power estimation.
3. Analyzing X state pessimism or an optimistic view, in RTL or GLS.

In addition, three different types of GLS can be performed, depending on how the input files and the commands are configured:

1. **Zero Delay Simulation:**

   Zero delay simulation means simulating the netlist without annotating any timing data. It is mainly focused for checking and validating the functionality of the design once it has been translated into a gate-level netlist. This is done once for every design synthesized to ensure that there is no any kind of functionality loss when synthesizing the design.

   These simulations are much faster than the simulations with timing and can be launched even before the netlist being completely frozen, or when the timing information is not available.

2. **Unit Delay Simulation:**

   The unit delay simulation operates under the assumption that all elements in a circuit have an identical delay time. Thus, providing any random delay value is similar to adding a unit delay, which has the advantage of being simpler to implement than SDF simulation. As using zero delay simulations can induce to false race conditions and zero-delay loops, running unit delay simulations can be used to detect and resolve genuine race conditions and timing-sensitive loops.

3. **SDF Simulation (Simulation Delay File specifies timing):**

   SDF simulation is the actual delay simulation where the delay specified in the SDF file is annotated into the design and and all the timing checks are computed and tested. This kind of simulations are slower than the previous two types, and it takes more time to finish. But, due to being the closest to the real conditions, this is the type of simulation that have been performed in the studies for this document.

   An SDF file is an output from the synthesis tool (Genus), the place & route tool (Innovus) or the static timing analysis tool (Tempus). The SDF files are delivered to the verification team and the simulations are performed using these delays and the synthesized gate-

level netlist. Then the delays contained in the SDF are annotated to the corresponding instances/arcs in the netlits, which is also known as *back-annotation*.

During these type of simulations, many annotation warnings are probable to show up and depending on the type of these warnings they may be analysed or not. The mos critical ones are the ones that make reference to non-existent paths, where the delay is specified in the SDF but there is no path for it in the netlist.

Once every necessary file from synthesis is available (SDF and netlist), it is time to set up the configuration files in order to specify *Xcellium* simulation tool how the simulation is desired to be run. The basic input files for a GLS run are,

- A gate-level netlist
- Library cell information
- Delay information in SDF timing file
- Testbench to exercise the netlist

and a typical GLS flow is shown in the Figure 12.

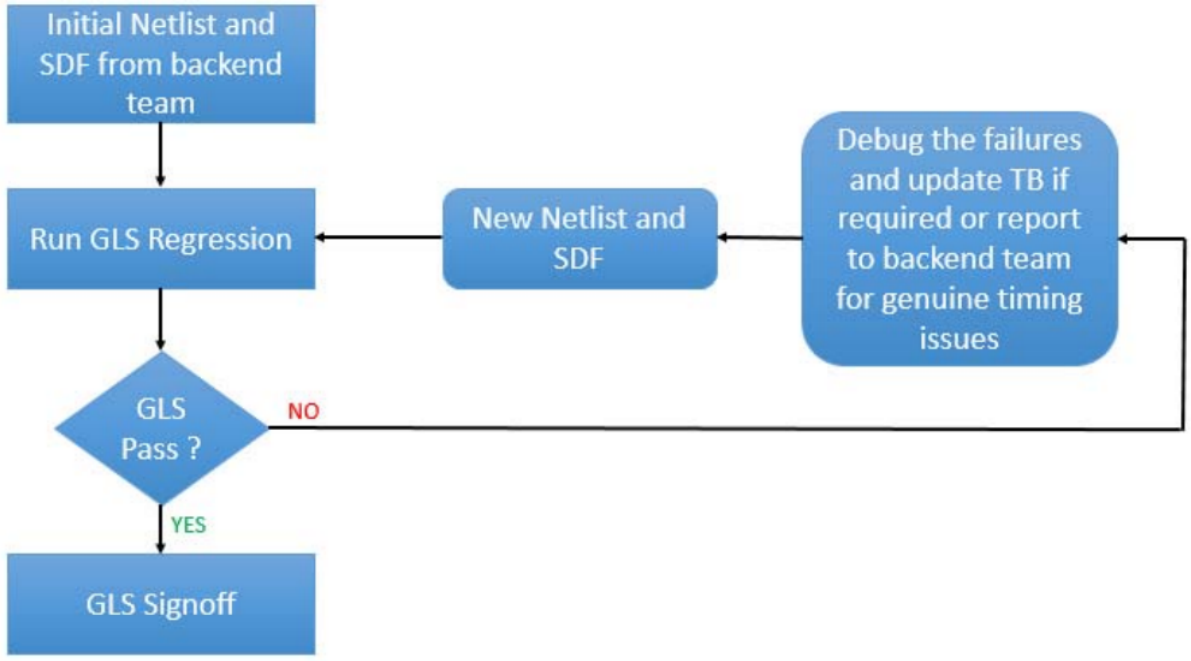

Figure 12: Typical GLS flow from [10]

The testbenches are the same ones used for the RTL simulations but may be changing little things as for example the path to where the simulation results are written and so on. Thus, the main structure and the tests are the same ones. So, for the approximate adders the already mentioned sequential 65535×65535 simulation was done to ensure no functionality loss and, in addition, for a more realistic situation a Python script generated 10 million of random pair numbers between 0 and 65535, and those were set as inputs in the simulation. In the case of

the multipliers simulations, the same tests has been used both the randomly generated operand pairs and the operations provided by the SW team. The only difference is that the testbenches for multipliers have been developed to be more configurable than the adders' ones, because they were developed a while after and the knowledge about both Verilog and Xcellium was much bigger than in the beginning. But again, it is just a matter of handiness and easy configuring when simulating, the basic processes are the same in both adder and multiplier simulations.

Finally, the configuration files are simple configuration scripts (.f) and a command file that specify the options to the simulator. In this configuration file, the timescale, the cell libraries, the synthesized netlist to be used and the testbench are specified for the simulator, while in the command file (with nomenclature: "CF_{Design name}") the commands referring to the SDF file annotation are described (SDF file, log file containing the annotation process and the scope module for the annotation). Also concerning some timing information, there is a timing file (.tfile) that specifies the delays of the input and output ports set by the user. In this study, these delays are always set to the 10% of the clock period. Lastly, another very simple *Tcl* file is passed to the simulator in order to record the activity during the simulations in order to compute more accurate power reports once the simulations are finished. As an example of the commands used to launch the simulations, below there are the two commands used to launch the SDF simulation for the KSA16 adder.

```
#Build the simulation and compile the SDF file
xrun \
        -l nc.log \
        -access \
        +rwc \
        -f simulation_files/run_ksa16.f \
        -sdf_cmd_file sdf/CF_ksa16 \
        -sdf_verbose \
        +maxdelays \
        -tfile simulation_files/run_ksa16.tfile
#Run the previously built simulation
xrun \
        -R \
        -nolog \
        -sdf_cmd_file sdf/CF_ksa16 \
        -sdf_verbose \
        +maxdelays \
        -tfile run_ksa16.tfile \
        -input tcf_ksa16.tcl
```

3.4.1 Performed simulations

Until this point, the simulation options have been explained between different kind of simulations and the different available options for the simulator, in order to understand the basic structure. But now, the two different studies that have been conducted are going to be described and how these are translated to the simulations. The two different studies are named **Overclocking** and **Voltage Underscaling**, and in some way they are similar studies but not. This means

that at first, the idea was to synthesize a design, get a SPICE electrical model of it and test it diminishing its voltage, but due to the lack of the necessary electrical models of the gates at transistor level and the big amount of simulation time it would require, other options were taken.

The first option would be **Overclocking**, where the clock period is reduced having similar effects as to diminishing the voltage. This resides in the fact that when downscaling he voltage, the transistors need more time to enter in saturation mode and thus every delay will increase having higher combinational delay and less slack. This behaviour of having smaller time to compute the necessary operations, can be somehow emulated decreasing the period, which means that the available time for the same delays will be smaller.

The set up of the simulations is not very complex, because the only things that needs to be changed are the period value specified in the testbench and the input and output delays that are specified in the timing file. These two things are easily made using a simple shell script that executes the changes automatically. Using this script, once all the adders and multipliers are simulated and the options can be automatically changed, so it is possible to simulate every design for every period studied with just one command. The obtained results for this simulations are presented in the following Section 4.

The second kind of study, **Voltage Underscaling**, refers to the delays applied during the simulations, in other words, it only concerns the SDF files and the delays that they contain. The idea is to take a netlist that has been synthesised for a certain voltage and use the *Cadence Tempus Static Timing Analysis* tool in order to generate new SDF files for that specific netlist but for different voltage values. This can be done using other existing libraries or even creating fictitious operating conditions and extrapolate the delays from the existing technology libraries. This way the simulation is run at the same period as for the synthesis, but the delays correspond to a lower voltage and they are much bigger, emulating a voltage underscaling condition.

Another *Tcl* file was developed for the *Tempus* tool, so that the netlist is read in and the different voltages and libraries are configured to obtain every SDF file. Finally, the simulations are launched recursively only changing the SDF file name and leaving the period equal. The results are also going to be presented in the next Section 4.

In addition, a special block has been described in the testbenches that detects any possible X value in the output, and changes it to either 1 or 0. This means that, for every design every simulation is computed twice but once changing the X values to 1 and the second time changing those to 0. These X values do not show up when operating under normal conditions, but when having timing errors due to the increase of the frequency or the reduction of the voltage, these errors are translated as these X values. In order to post-process the timing errors, if an X value is obtained as a result, the error distance cannot be computed, and that is why this change is done. Although it may not be realistic, it may cover worst case situations and also some best case options.

3.5 Power Computation

In order to understand where have the obtained power values their origin, the power computation process have to be explained. A power report can be obtained from the synthesis tool, but it would not be as realistic as getting the switching activity during the simulations, and finally use a specialized CADENCE tool to compute the power for that design and the obtained activity file.

So, the first step is the recording of the switching activity of every net in the design when simulating. Different ways or archive types are available to obtain the activity of the design. Depending on the file type chosen, the activity will be recorded in one manner or another. The used simulator tool, *Xcellium*, supports the dumping of 4 different formats concerning this switching activity:

1. **TCF** - Average-based stimulus

   Being an average-based stimulus, the output file is very compact. For each signal, only two pieces of information are stored:

   - The toggle count, that indicates how often the net switches between 1 and 0 states.
   - The probability of the net to be in the logic 1 state.

2. **SAIF** - Average-based with state-dependent arcs

   SAIF format is similar to TCF format and extracts the same type of rate and state probability data. One addition is that the user can take advantage of the state-dependent and path-dependent arcs that are available in the activity file. For example, memory technology cells have power arcs where for instance it is different to receive a clock edge in write mode than in read mode. To take advantage of this, a forward-SAIF file needs to be generated.

3. **VCD** - Event-based stimulus stored in text format

   This file is a Verilog standard event-based activity format, where every single transition of a net is stored in the file. As it is an ASCII file, depending on how much activity is occurring in the stimulus and the simulation duration, it can be extremely large. But, the advantage of this is that a full power profile overtime can be generated from these event-based formats.

4. **SHM** - Event-based stimulus stored in binary format

   This time, SHM is the Cadence standard event-based *binary* activity format. It stores the same things as the VCD but it is highly optimized and it greatly reduces the file size compared to a text-based format like VCD.

The main file type used in the studies is the TCF. The VCD file type was considered in the beginning but it was quickly discarded due to the large size of the output files (between 25 and 50 GB). In addition, a power profile overtime is not so important in this power study, because the objective is the comparison of the power consumption between the different designs and with the computation of the total power consumption.

Once this switching activity is obtained from the simulation, the next step is the power computation using the *Joules* power analysis tool from Cadence. A simple Tcl script was developed which using the netlist, the final design's database and the obtained switching activity file, the power consumption for that activity file and netlist is computed. If the simulation changes, the activity file will also change and thus, the final power consumption will be different. So, having this in mind, the more closer the simulations are to real operations, the more accurate the power results will be.

The Equation 4 illustrates how the dynamic power can be calculated for a general case.

$$P_D = \left[\left(C_{pd} \times f_I \times N_{SW} \right) + \sum \left({C_L}^n \times {f_O}^n \right) \right] \times {V_{CC}}^2 \tag{4}$$

Where:

C_{pd} = power-consumption capacitance (F)
f_I = input frequency (Hz)
${f_O}^n$ = all different output frequencies at each output, numbered 1 through n (Hz)
N_{SW} = total number of outputs switching
V_{CC} = supply voltage (V)
${C_L}^n$ = all different load capacitances at each output, numbered 1 thorugh n

From Equation 4, it can be seen that the dynamic power is dependent on quite many parameters, but the two that most interesting are for this study are the frequency and the supply voltage. The case under study is the behaviour of the approximate arithmetic units under reduced voltage conditions. This means, based on Equation 4, that the dynamic power should decrease with the decrease of the supply voltage. In contrast to reducing the voltage, increasing the frequency increases the dynamic power.

4 Results

This Section the results obtained for the designs and simulations presented previously are presented and discussed. The Section is divided in two Subsections: Adders and Multipliers. This is because apart from being different arithmetic operations, even if the simulations conducted for them are the same, some little things, as the used period values and so, can not be equal. Both subsections are divided in other 5 parts: Slack, Area, Errors, Power and Voltage Underscaling results. From these 5 parts, the Slack and Area parts are quite general as they do not depend on any simulation or SDF file for their results, but both Errors and Power results are obtained from the already explained *Overclocking* simulations, while in the last part the results for *Voltage Underscaling* simulations are presented.

4.1 Adders

Results for the studied adder designs are presented in this Subsection. The results cover from the very first reports obtained in the synthesis that concern the timing information (slack) and the area information. A power report could be also obtained from this step but it would not be as realistic as getting the activity from the simulations and then computing the power based on that activity and thus, the power and the errors are obtained once the simulations are finished.

4.1.1 Setup slack

First of all, The setup slack is defined as the difference between the actual or achieved delay for a timing path (t_d) and the avaliable time for arrival to a flip-flop (FF), which depends on clock period (T_{CLK}), clock-to-FF time ($t_{clk\to q}$) FF setup time (t_{su}), and, generally, clock skew (S).

$$SetupSlack = T_{CLK} - \left(t_{clk\to q} + t_d + t_{su} - S\right) \tag{5}$$

The slack is computed for every path in the design. Thus, each path will have a slack value assigned to it, and the path with the smallest slack will be considered as the longest or critical path. Usually, this is the slack that is taken into account, because if this path fulfils the design constrains or specifications, then this assures that the rest of the paths also fulfil the requirements. So, in the end, in the final timing report that is obtained from *Genus Synthesis Tool* the slack that appears in the report is the same as the critical path's slack, and thus it can be considered to be the whole design's slack.

This slack can either be positive or negative. If the slack turns out to be positive, it means that the design is working properly for the desired period, and this can be understood as the design working fine for the specified frequency. On the contrary, if the slack is negative it means that the longest path takes more time to complete than the provided period and timing errors are expected to occur. The intention is always to have a positive slack in order to avoid any timing error. If during the synthesis the first steps show a negative slack, the synthesis tool tries to optimize the design to the fullest in order to achieve a positive slack, and ensure the correct functionality of the resulting netlist. Sometimes it may also happen that the slack results as

zero, which means that in principle there should not be any timing errors but there is no time margin and the slightest delay addition could make the design's wrong operation.

In the case of this study, as already mentioned, the syntheses of the adders were made using the *Globalfoundries* technology libraries for 22 nm processes and more specifically the library used was characterized for 0.80V and 25ºC with typical transistors, and using a period of 10 ns.

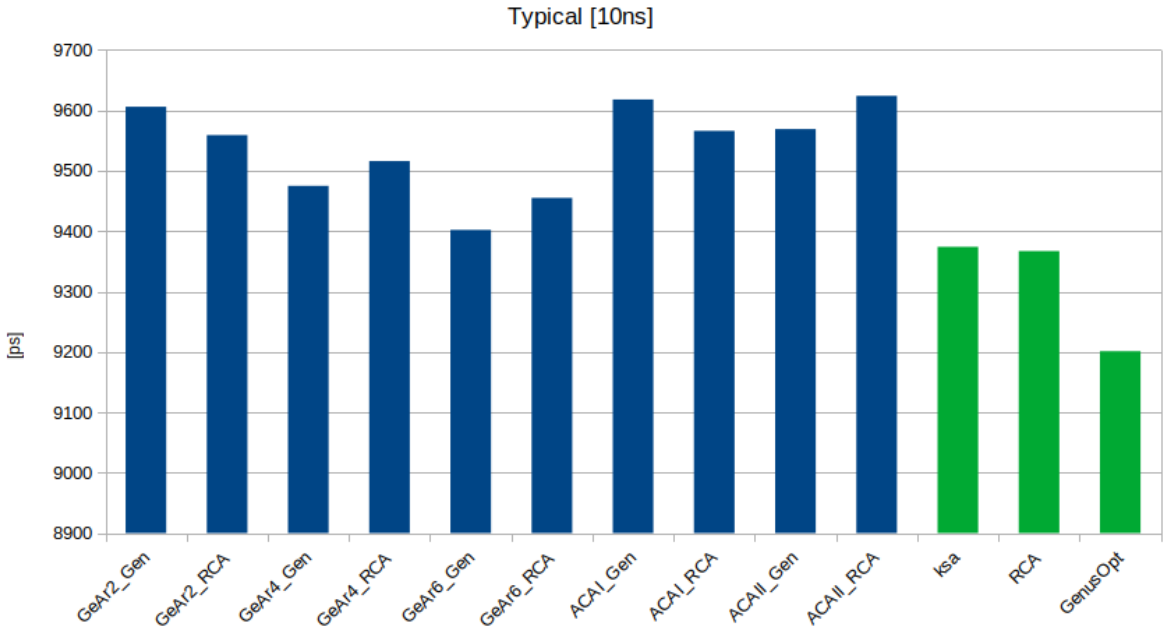

Figure 13: Slack values for different adder implementations

The obtained slack values for the different implemented adders are presented in the bar plot of Figure 13. In this bar plot, the approximate adders are displayed in blue colour while the accurate adders are displayed in green. As it can be clearly observed, all approximate adders offer bigger slacks than the accurate ones, as a consequence of being faster in computing the operation. The maximum difference in slack between approximate and accurate adders is of 423 ps while the minimum one is equal to 28 ps. The best slack values are offered by the ACAI_Gen, ACAII_RCA and GeAr2_Gen designs, while the worst slack is offered by the accurate adders and specially the GenusOpt16 adder. It can be also noticed that all the slack are very large values between 9200-9600 ps. This is because the implemented designs are not very complex, being only an adder with registers in the inputs and outputs (Figure 11), so setting a period of 10 ns is a very soft restriction.

In addition, for the sake of comparison between the approximate adders, the same slack values of the approximate adders shown in Figure 13 are specifically plotted in Figure 14 having each adder type with their two versions next to each other. From this plot it can be seen that depending on the design the usage of the + sign or the RCA as sub-adders can be more beneficial than the other option. For example, in the case of GeAr2 adder it seems that the employment of the + sign leads to a better slack, while in ACAI adder it is the contrary. It is also worth commenting

that the variations in slack for its two different versions is not big enough to notice the benefits in practice.

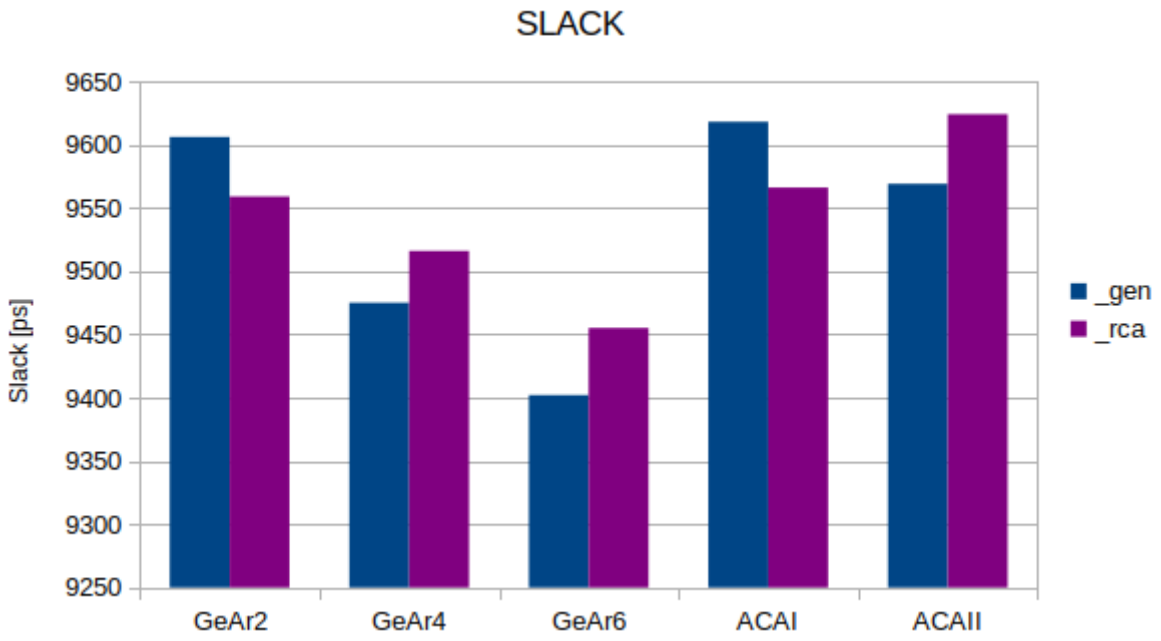

Figure 14: Slack values of approximate adders

4.1.2 Area

About the area, again, it is a value that is obtained right after the synthesis process and it refers to the area that occupies the design. The area reports offer information about how many cells are used (*cell count*), the area in μm^2 that these cells and other macros occupy (*cell area*) and the *net area* which exhibits the area that the routing, interconnections and metal layers occupy. The *total area* is the summation of these last two values, and indicates the whole area of the design. Figure 15 plots the obtained cell areas, while Figure 16 shows the achieved total area values for the different adder topologies. The area values showed also contain the corresponding input and output registers for each design, as depicted in Figure 11

As it can be seen, the total areas are quite similar with values around 165-135 μm, and also with similar cell area values. First of all, it can be clearly observed that the best area proposal is the one given by the GenusOpt16 adder solution. This is due to the freedom that the synthesis tool, Genus, has over the adder topology and also the very relaxed synthesis frequency constrain. This means that as it has plenty of time margin, it can focus on shrinking the design's area in exchange of resulting in worse timing slack. This happens for all the adders, but the rest of the adders have a fixed structure that Genus has to follow, which means that they are more constrained than the GenusOpt16 precise adder.

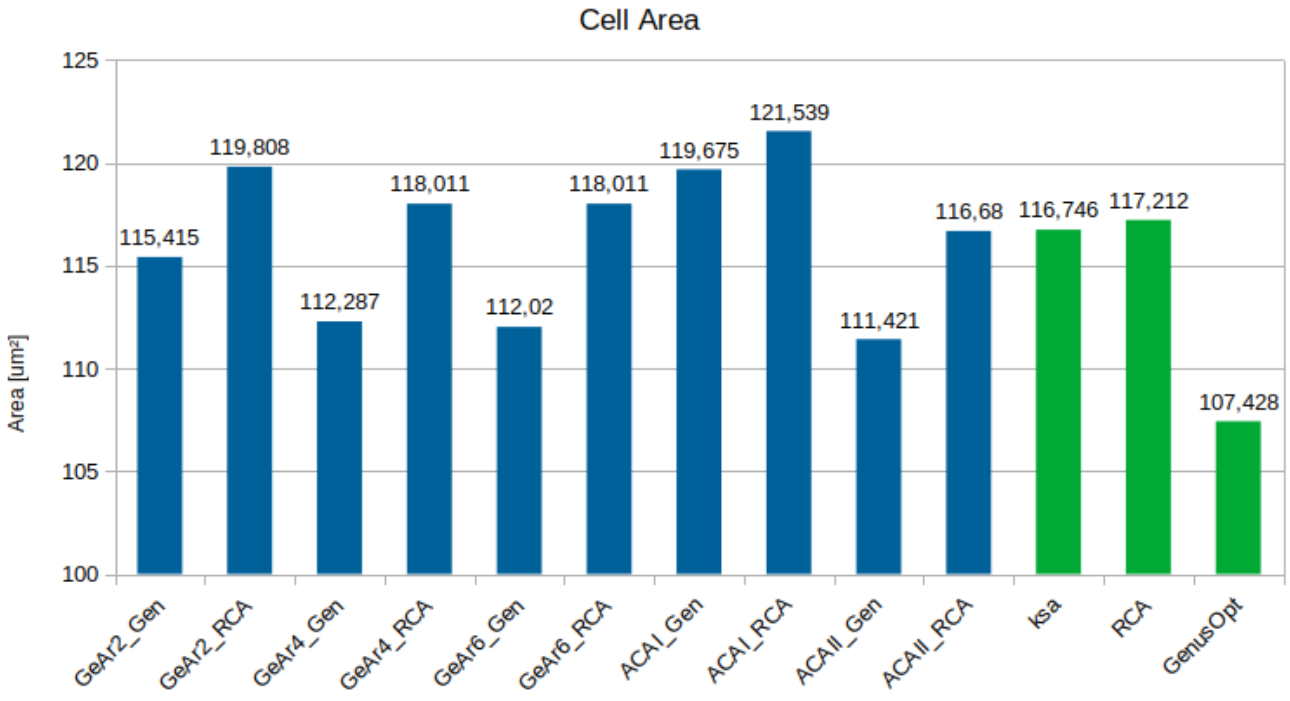

Figure 15: Cell area results for different adder implementations

Apart from this, the rest of adders show very similar total areas. Although some of them show promising cell areas, like GeAr2_Gen adder, due to larger net areas than the other adders they finally end up with similar total areas. In general, and excluding GenusOpt16 adder, GeAr4_Gen, GeAr6_Gen, both versions of ACAII and the precise adders offer good area values around $150 \pm 2\ \mu m$. Worst area values are offered by ACAI adder type, which has sense because it is the adder structure that has the biggest amount of sub-adders.

Finally, looking at Figure 15, it can be noticed that the approximate adders that use the $+$ sign (with suffix _Gen) offer a better cell area than their competitor using the RCA sub-adders. But in contrary, looking at Figure 17, it can be seen that in some cases this last advantage is nullified by a bigger net area, resulting in bigger total area, as it is the case in ACAI adder. Figure 17 compares the same total areas of the approximate adders shown in Figure 16 between their two versions.

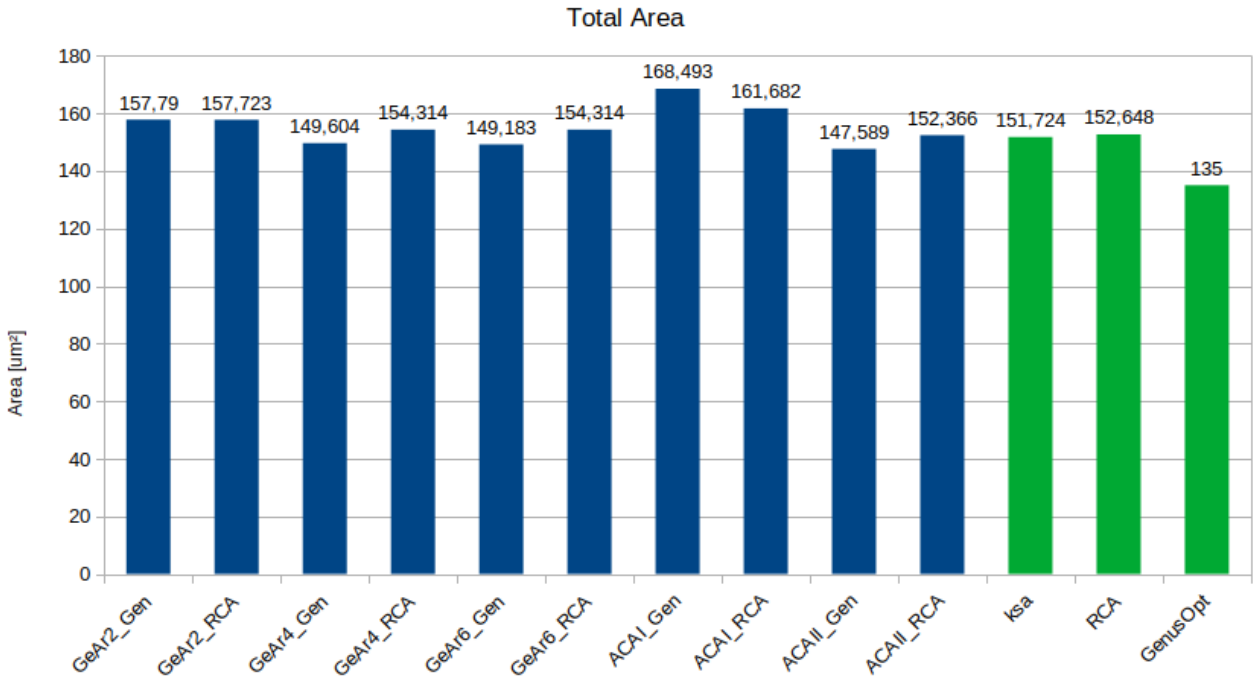

Figure 16: Total area results for different adder implementations

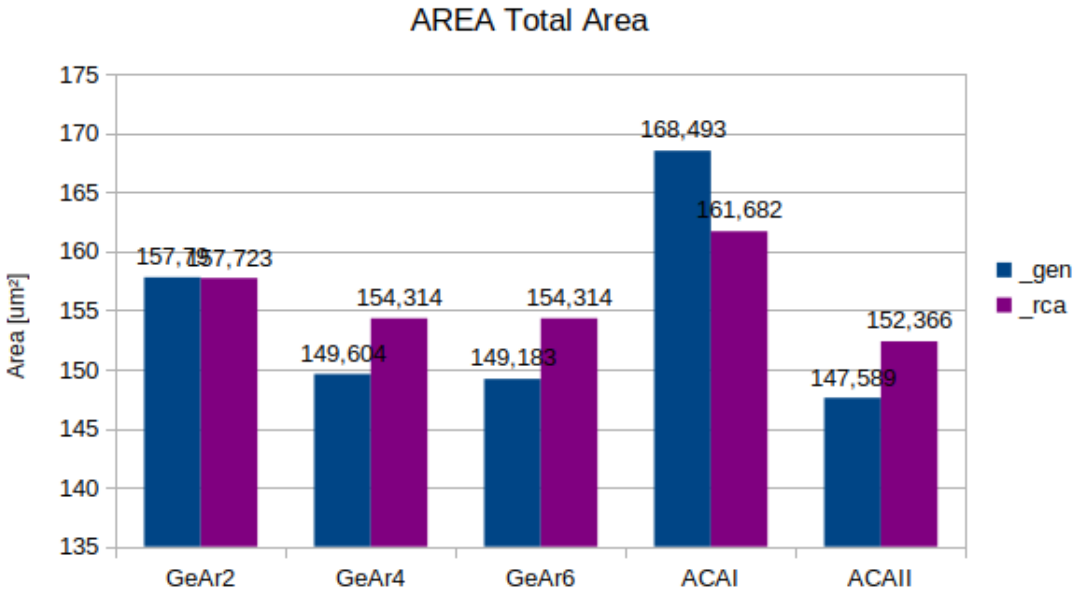

Figure 17: Total area values of approximate adders

4.1.3 Errors and precision

As already mentioned, the results concerning the errors and precision are obtained from Over-clocking simulations, which means that the period is diminished in order to emulate the slower behaviour of the CMOS technology in a voltage underscaling environment, and timing errors appear. Three different periods have been chosen: the first one is the same period used for synthesis (10 ns), while the other two were decided according to the slack results that have been exposed in Subsubsection 4.1.1. These two periods have been chosen so that the adders were studied under a moderately aggressive period (400 ps) and a very aggressive period (200 ps).

Before starting the error and precision study, it is convenient to briefly explain how the error and the metrics used along this chapter are obtained. The most basic one is the **error** or **error distance** which in this case is calculated according to the following formula:

$$Error = Sum_{accurate} - Sum_{approx} \tag{6}$$

The **precision** is also used as a metric during the explanation. This precision is shown in percentage values and it is calculated as:

$$Precision[\%] = \frac{\text{\# of correct additions}}{\text{\# of total additions}} \times 100 \tag{7}$$

Finally, for some cases the **Mean Relative Error (MRE)** [11] is also computed. This Figure of Merit indicates in a percentage value how much the errors deviate from the correct result in average, and it is computed as:

$$MRE[\%] = \frac{1}{N} \times \sum_{i=1}^{n} \left| \frac{R^i_{accurate} - R^i_{approximate}}{R^i_{accurate}} \right| \times 100 \tag{8}$$

Being,

N = Total number of operations
R = Result
n = Total number of errors given for that case

Results for 10 ns period: First of all, the results concerning the 10 ns period are very interesting to observe the intrinsic errors that the approximate adders have due to their approximation. This is because at this period there will not be any timing errors and the only errors that will appear will be those due to the approximation. Another meaning of these results, is that the precision obtained for the adders will be equal to the *top achievable precision* that they will get.

Results for 10ns period				
Adder Type	**Right**	**Wrong**	**Total**	**Precision [%]**
GeAr2_Gen	9955709	44291	10000000	99,55709
GeAr2_RCA	9955709	44291	10000000	99,55709
GeAr4_Gen	9981546	18454	10000000	99,81546
GeAr4_RCA	9981546	18454	10000000	99,81546
GeAr6_Gen	9980605	19395	10000000	99,80605
GeAr6_RCA	9980605	19395	10000000	99,80605
ACAI_Gen	9843748	156252	10000000	98,43748
ACAI_RCA	9843748	156252	10000000	98,43748
ACAII_Gen	9413961	586039	10000000	94,13961
ACAII_RCA	9413961	586039	10000000	94,13961
Kogge-Stone	10000000	0	10000000	100
Ripple-Carry	10000000	0	10000000	100
GenusOpt('+')	10000000	0	10000000	100

Table 1: Top achievable precision of studied adders

The results concerning the precision and the inherent errors that the approximate adders show are presented in Table 1. Evidently, the three of the accurate adders show a 100% of precision because there are not timing errors present yet. It can also be appreciated that every approximate adder offers precision values above 90%, except for the ACAII adder that, although it has high precision, it is still smaller than the rest of the adders. The errors were studied in more depth to record the magnitude of these errors, which happen to be always a power of 2. This fact is totally coherent, because as the approximate adders are based in the carry truncation, there will be some point in their architecture where the carry should be propagated but it will not.

Magnitude of errors for 10ns period								
	Error magnitude							
Adder type	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
GeAr2_Gen	0	0	14773	0	14754	0	14764	0
GeAr2_RCA	0	0	14773	0	14754	0	14764	0
GeAr4_Gen	0	0	0	0	18454	0	0	0
GeAr4_RCA	0	0	0	0	18454	0	0	0
GeAr6_Gen	0	0	0	0	0	0	19395	0
GeAr6_RCA	0	0	0	0	0	0	19395	0
ACAI_Gen	19661	19619	19603	19419	19439	19452	19405	19654
ACAI_RCA	19661	19619	19603	19419	19439	19452	19405	19654
ACAII_Gen	293147	0	0	0	292892	0	0	0
ACAII_RCA	293147	0	0	0	292892	0	0	0

Table 2: Error magnitudes for 10ns period

As it is described in Table 2, the error magnitudes inserted by the modification of the adder architectures have values between 256 and 32768. It is desirable to have errors of value 256

instead of 32768, because in the event of having an error it is preferred to have a smaller deviation from the correct answer. Among the GeAr type adders, GeAr2 generates error values of 1024, 4096 and 16384, while GeAr4 generates errors of magnitude 4096 and GeAr6 of 16384. These last two offer very similar precision, but GeAr4 is more appealing from the error magnitude point of view because these are 4 times smaller than the ones of GeAr6.

About the ACAI adder, the errors are distributed quite uniformly between the magnitudes of 256 and 32768, and concerning ACAII, the errors are concentrated in the values of 256 and 4096 with a large amount of them.

From this point on, timing errors appear in simulations, and as already mentioned, the outputs can contain certain X values. In order to quantify and make the study, the X values are traced and changed in the testbench before recording them. They are change first to logical 1 and later to logical 0, and thus the next results for periods 400 ps and 200 ps will contain two set of results.

Results for 400 ps period: Table 3 presents the results for the simulations performed for a period of 400 ps and considering the X values as logical 1. It can be seen that as expected, the timing errors start to appear in the precise adders, compromising their accuracy (quite considerably in the case of GenusOpt16). In contrast, some of the approximate adders show that their precision is not affected or that despite being affected, the effect is smaller than in the case of the accurate adders.

Results for 400ps period					
Adder Type	**Right**	**Wrong**	**Total**	**Precision [%]**	**Loss [%]**
GeAr2_Gen	9955709	44291	10000000	99,55709	0
GeAr2_RCA	9940206	59794	10000000	99,40206	0,155
GeAr4_Gen	9774563	225437	10000000	97,74563	2,0698
GeAr4_RCA	9949914	50086	10000000	99,49914	0,31632
GeAr6_Gen	9828065	171935	10000000	98,28065	1,5254
GeAr6_RCA	9954337	45663	10000000	99,54337	0,26268
ACAI_Gen	9843748	156252	10000000	98,43748	0
ACAI_RCA	9803703	196297	10000000	98,03703	0,40045
ACAII_Gen	9355772	644228	10000000	93,55772	0,58189
ACAII_RCA	9413961	586039	10000000	94,13961	0
Kogge-Stone	9951756	48244	10000000	99,51756	0,48244
Ripple-Carry	9962242	37758	10000000	99,62242	0,37758
GenusOpt('+')	9628539	371461	10000000	96,28539	3,71461

Table 3: Precision achieved with 400ps period, X $\rightarrow$ '1'

Looking at the Table 3, the precision values obtained in general are quite similar and it could be considered that at 400 ps the approximate adders offer the same precision values as accurate adders. The adder with the highest precision is the conventional RCA, followed by the approximates GeAr2_Gen and GeAr6_RCA, and then the accurate KSA and the GeAr4_RCA. From the additional column, that stands for the precision loss compared to the 10 ns period results, it

can be observed that the accurate adders (except GenusOpt16) are quite in the middle when the precision loss is concerned, which means that there are better and worse options for approximate adders.

One more time, the error magnitudes are recorded to deeply study the impact of the accuracy loss. This time, as timing errors will generate errors with magnitudes different to power of 2 values, the exact error values have been recorded and plotted in histograms. As a result, 6 histograms are obtained where in five of them both versions of each approximate adder is plotted for comparison, and in the last one the three accurate adders are plotted.

Figures 18, 19, 20, 21, 22 and 23 show the previously mentioned histograms where the error magnitudes are shown. In these histograms, the inherent errors from Table 2 are visible (quite clearly in Figures 18, 21 and 22). The rest of the errors are caused by timing violations which, in contrast to the inherent errors, happen to be also negative errors, thus all the negative errors that appear in the histograms are due to timing errors. According to the Equation 6, the meaning of the negative errors is that the obtained approximate addition is bigger than the accurate addition.

In some cases, these new errors seems to be concentrated in lower magnitudes around 0 (Figures 20 and 23), while in others even if having timing errors in lower magnitudes, there are also some big spikes in bigger magnitudes as in Figures 18, 19 and 21.

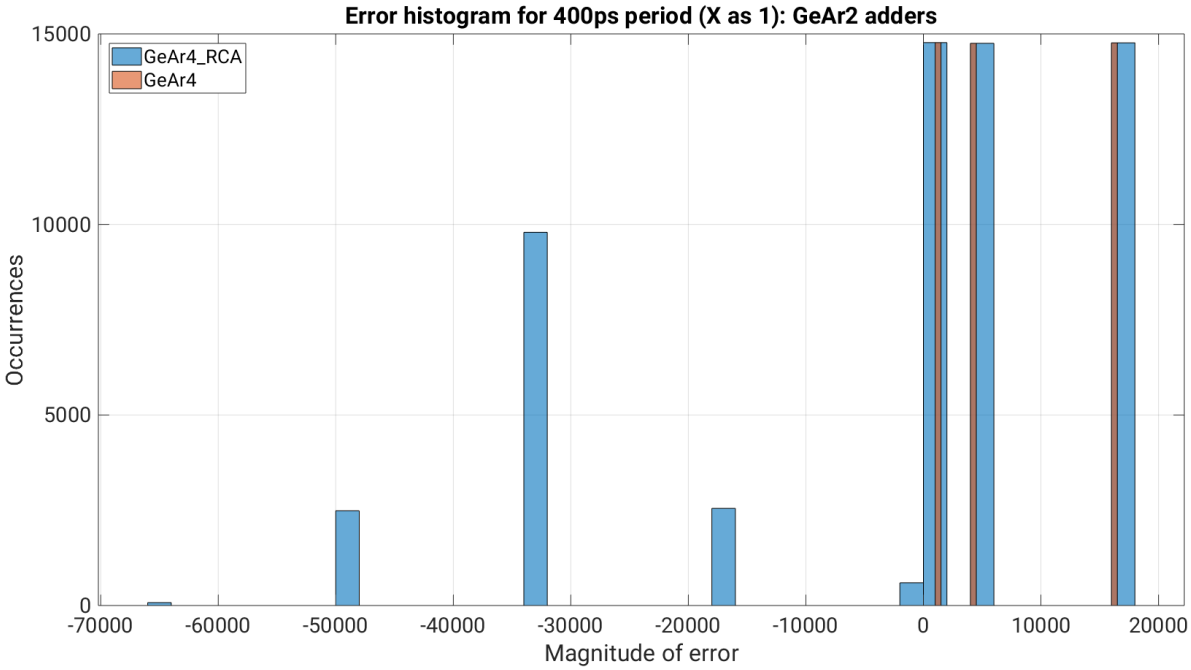

Figure 18: Error magnitude histogram for GeAr2 adder (400ps X as 1)

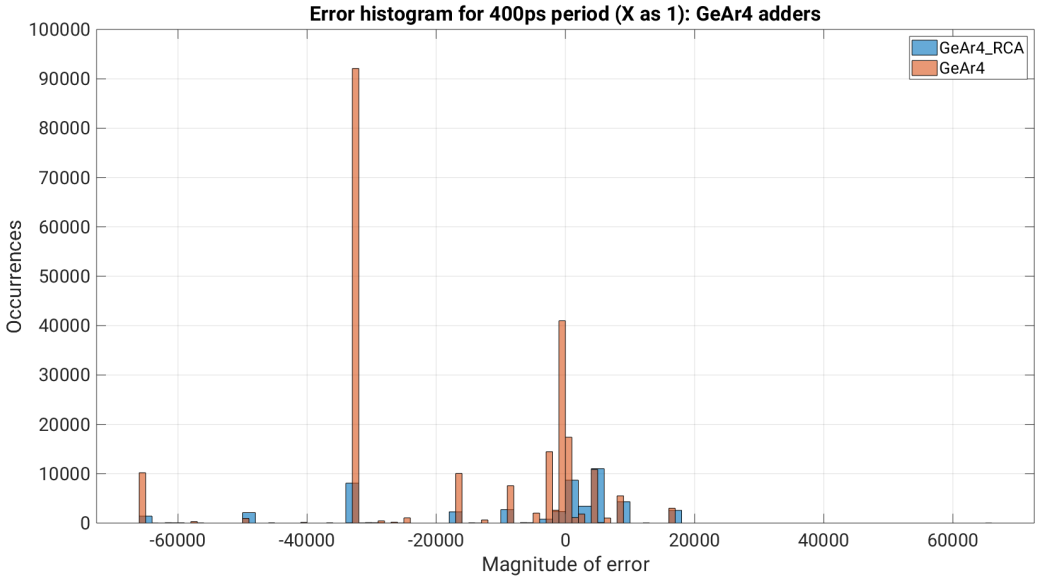

Figure 19: Error magnitude histogram for GeAr4 adder (400ps X as 1)

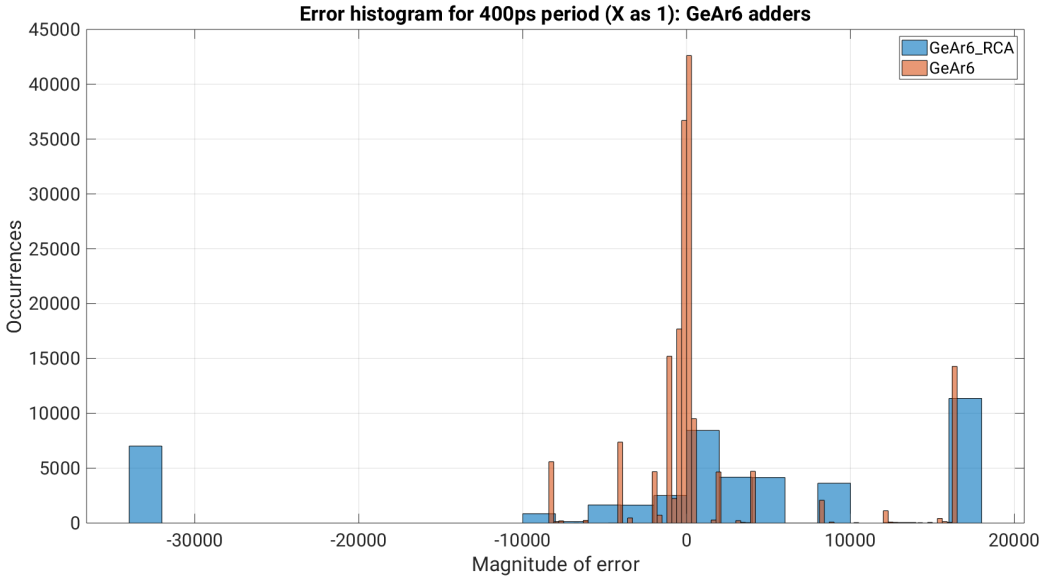

Figure 20: Error magnitude histogram for GeAr6 adder (400ps X as 1)

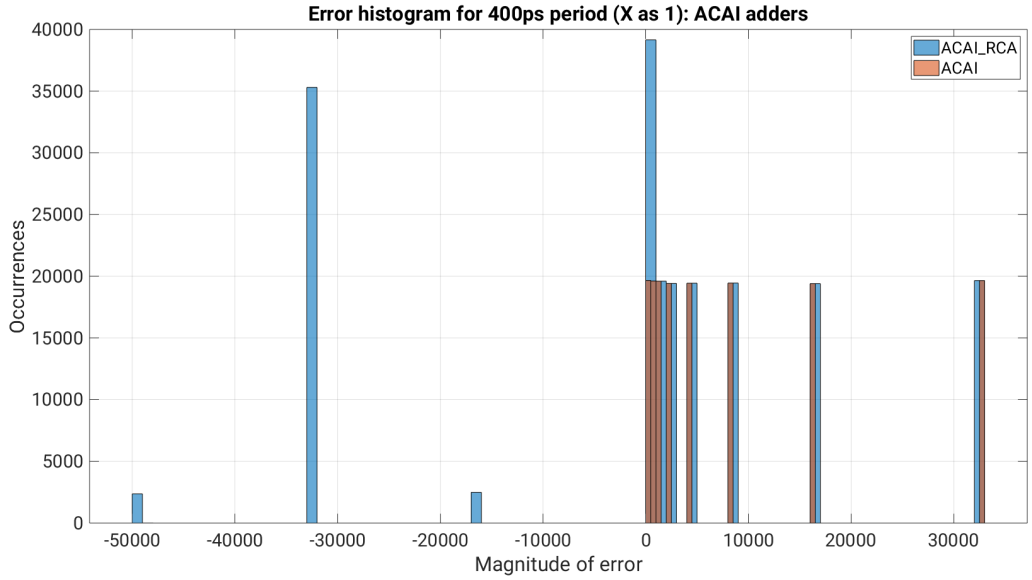

Figure 21: Error magnitude histogram for ACA-I adder (400ps X as 1)

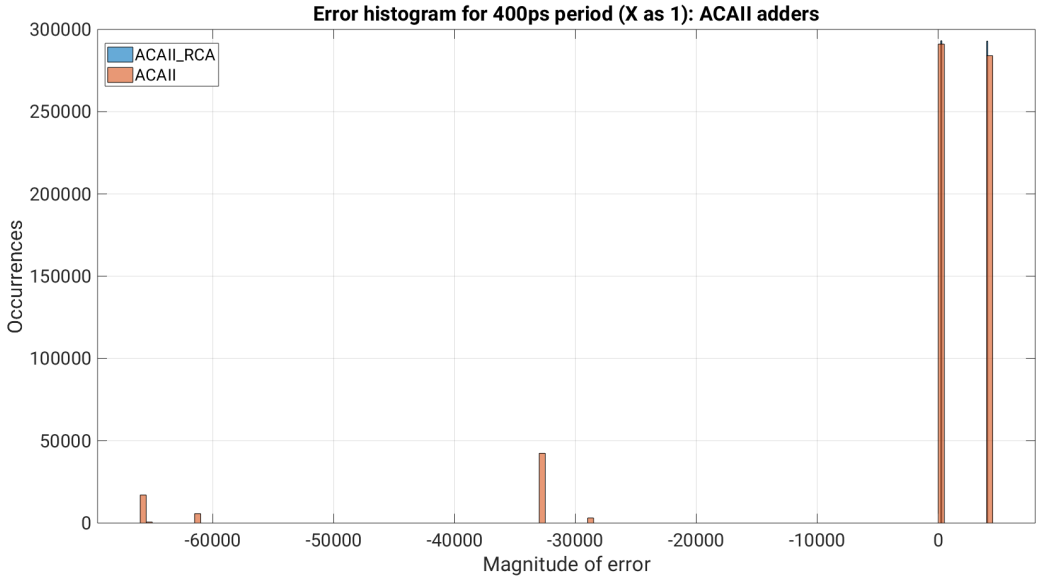

Figure 22: Error magnitude histogram for ACA-II adder (400ps X as 1)

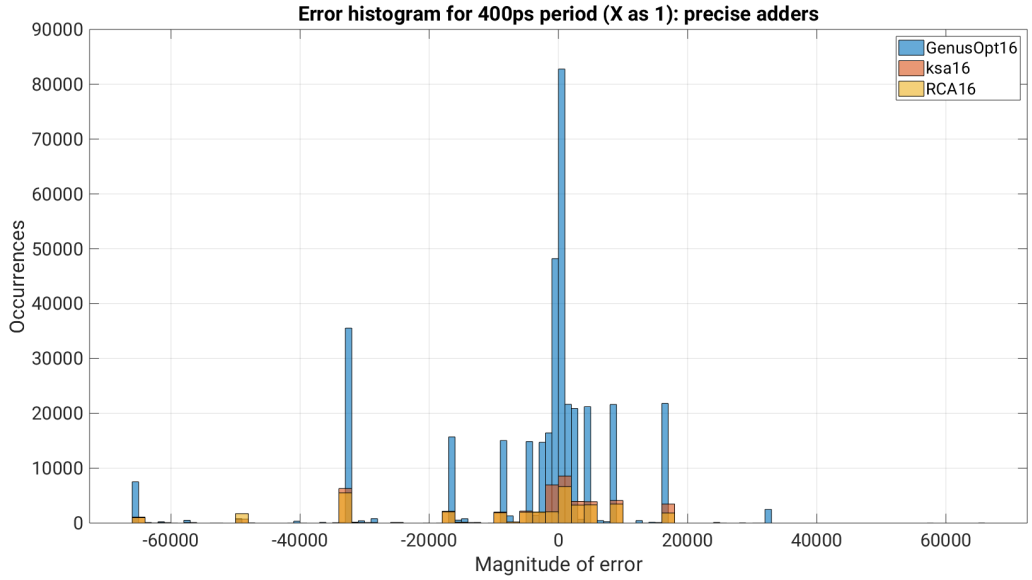

Figure 23: Error magnitude histogram for precise adders (400ps X as 1)

To continue, the results for the same simulations with the same period of 400 ps but now considering the X values as logical 0 are presented.

Results for 400ps period						
Adder Type	**Right**	**Wrong**	**Total**	**Precision [%]**	**Loss [%]**	**Loss for X → '1'**
GeAr2_Gen	9955709	44291	10000000	99,55709	0	0
GeAr2_RCA	9938515	61485	10000000	99,38515	0,17194	0,155
GeAr4_Gen	9799481	200519	10000000	97,99481	1,82065	2,0698
GeAr4_RCA	9950871	49129	10000000	99,50871	0,30675	0,31632
GeAr6_Gen	9799896	200104	10000000	97,99896	1,80709	1,5254
GeAr6_RCA	9957692	42308	10000000	99,57692	0,22913	0,26268
ACAI_Gen	156252	156252	10000000	98,43748	0	0
ACAI_RCA	9826749	173251	10000000	98,26749	0,16999	0,40045
ACAII_Gen	9369564	630436	10000000	93,69564	0,44397	0,58189
ACAII_RCA	9413961	586039	10000000	94,13961	0	0
Kogge-Stone	9950014	49986	10000000	99,50014	0,49986	0,48244
Ripple-Carry	9961786	38214	10000000	99,61786	0,38214	0,37758
GenusOpt('+')	9522318	477682	10000000	95,22318	4,77682	3,71461

Table 4: Precision achieved with 400ps period, X → '0'

Table 4 again shows the different precision values obtained for the studied adders following Equation 7, and it also contains the concerning precision loss compared to the 10 ns precision for this simulation and, for the sake of comparison, the same loss indicative obtained in the

previous case when X values were considered as 1. Looking to these two columns, it is visible that the differences in precision loss between the two cases are not so big, as the precision loss differs normally in some decimals or even less. It is worth commenting that the precision of the GenusOpt16 adder is 1% worse in this case than in the previous one, and that the adders that did not show any precision loss in the previous case, remain without any precision loss in this case too.

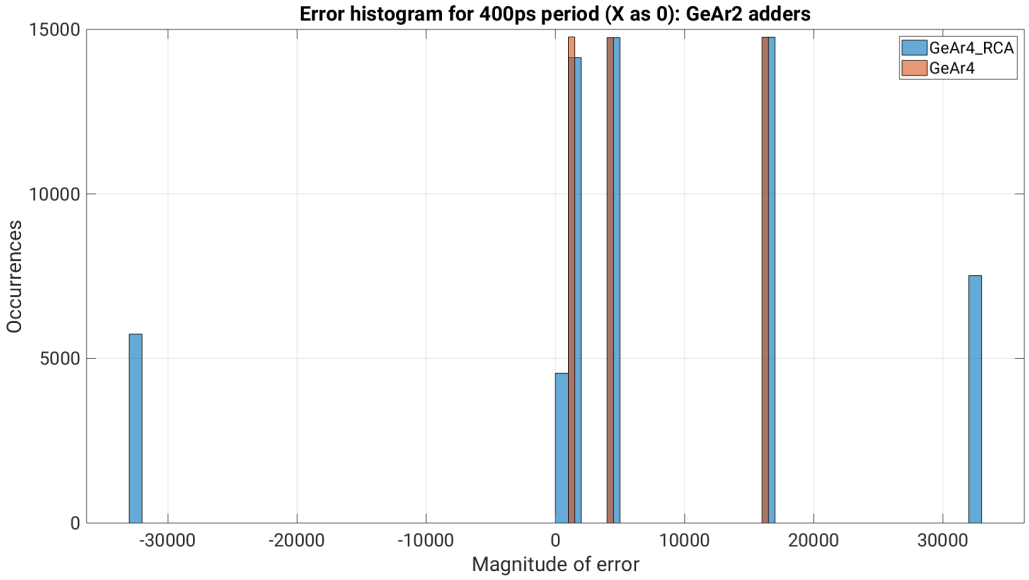

Figure 24: Error magnitude histogram for GeAr2 adder (400ps X as 0)

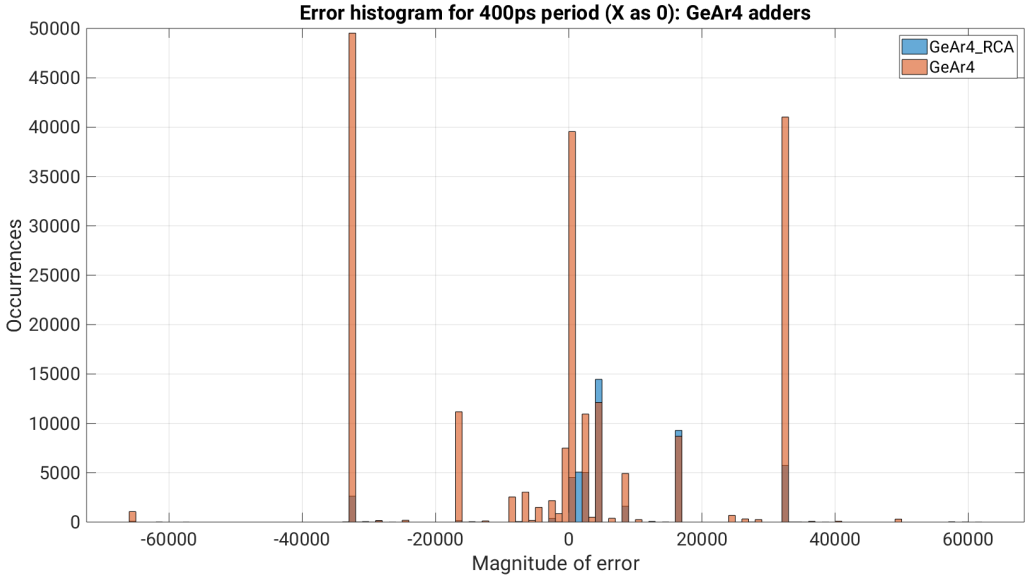

Figure 25: Error magnitude histogram for GeAr4 adder (400ps X as 0)

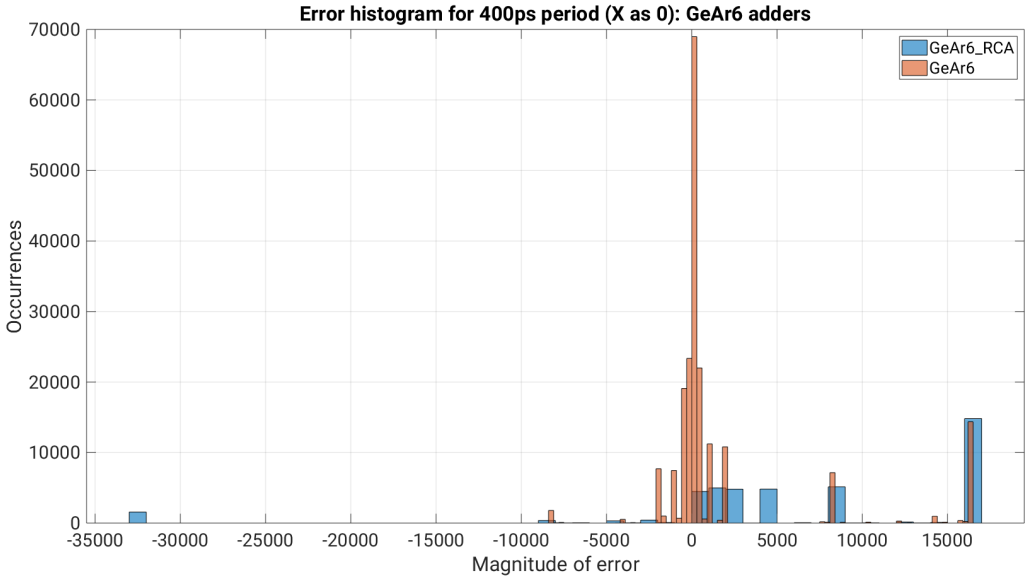

Figure 26: Error magnitude histogram for GeAr6 adder (400ps X as 0)

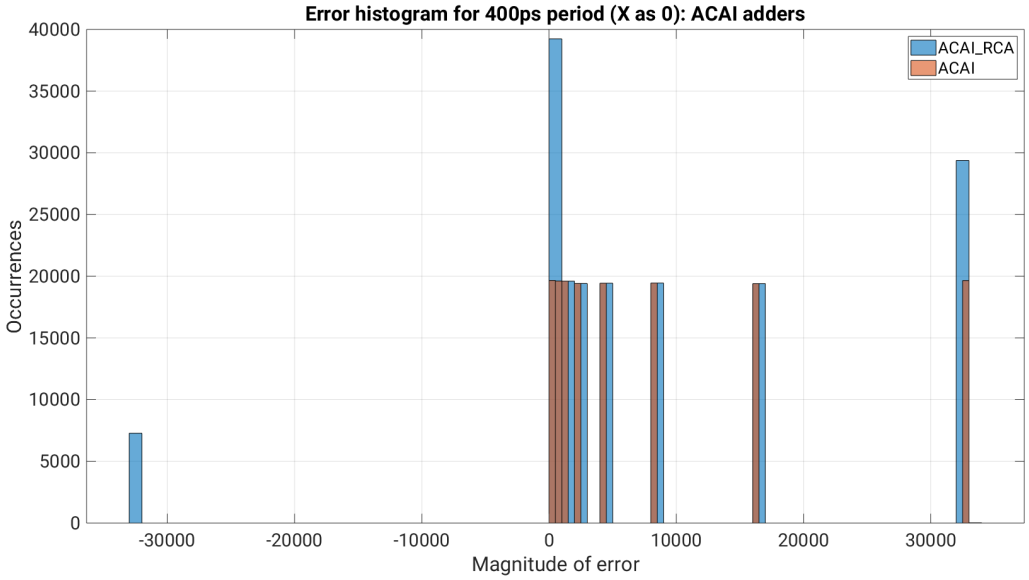

Figure 27: Error magnitude histogram for ACA-I adder (400ps X as 0)

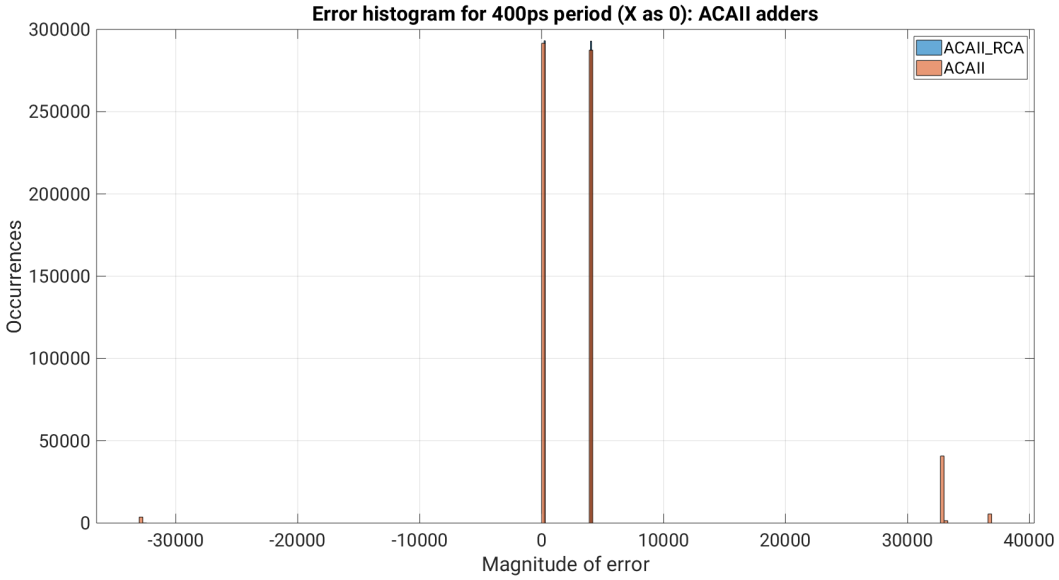

Figure 28: Error magnitude histogram for ACA-II adder (400ps X as 0)

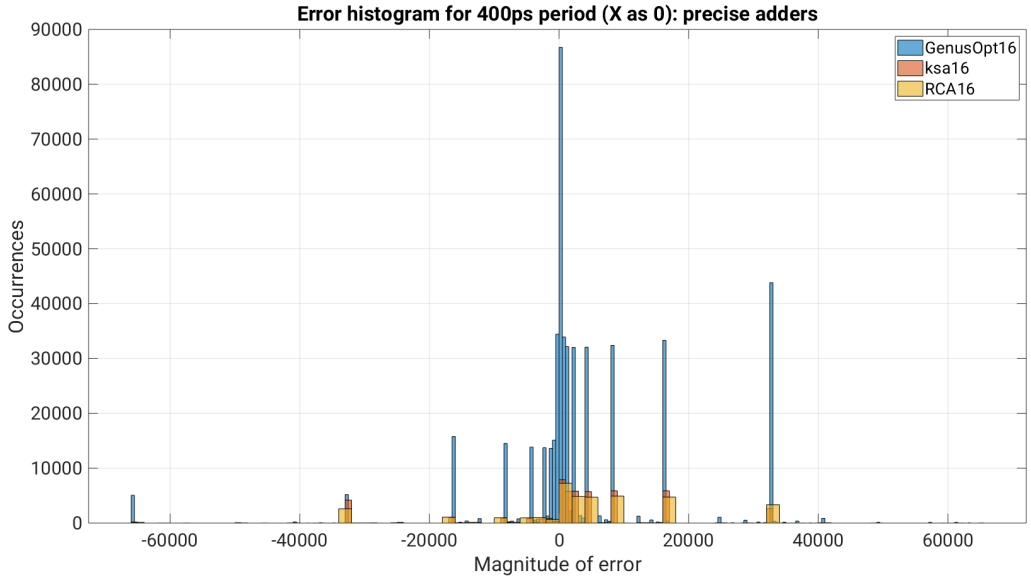

Figure 29: Error magnitude histogram for precise adders (400ps X as 0)

The same idea as in the previous case is employed, and the histograms of Figures 24, 25, 26, 27, 28 and 29 are obtained. In general, the results are very similar, but now the errors show a trend to concentrate in the positive side (27 and 28), in contrary to the previous case where the timing errors were located in the negative side. Finally, it is slightly noticeable that the majority of the errors concentrate in the center, where the near-0 magnitudes are located.

Results for 200 ps period: As already explained before, the same simulations were conducted but now for the very aggressive period value of 200 ps. Again, two set of results are obtained from the simulations, the first one considering the X values as logical 1 and the second one with X values as 0. These aggressive simulations are done with the intention of forcing large amounts of timing violations, and this way seeing if the majority of errors due to this timing violations are distributed uniformly, or they follow any other distribution type.

Same table format have been used for the representation of the simulation results of this part, and in Table 5 the first results for a period of 200 ps with X values as 1 are presented. It is clear that the obtained precisions with this period are not even close to be acceptable at all. Even the two accurate adders RCA and KSA show a precision below the 35%, which is unacceptable, and they are the adders with the highest precisions. Addressing the approximate adders under study, in general the precision offered is even worse than a third of the 35% offered by the accurate ones mentioned before. But, it is worth commenting that GeAr4_RCA, GeAr6_RCA and ACAII_RCA adders offer a precision near 30%.

Results for 200ps period				
Adder Type	**Right**	**Wrong**	**Total**	**Precision [%]**
GeAr2_Gen	226814	9773186	10000000	2,26814
GeAr2_RCA	1645853	8354147	10000000	16,45853
GeAr4_Gen	395222	9604778	10000000	3,95222
GeAr4_RCA	2861719	7138281	10000000	28,61719
GeAr6_Gen	459513	9540487	10000000	4,59513
GeAr6_RCA	2866432	7133568	10000000	28,66432
ACAI_Gen	79676	9920324	10000000	0,796759
ACAI_RCA	761476	9238524	10000000	7,61476
ACAII_Gen	743529	9256471	10000000	7,43529
ACAII_RCA	2727772	7272228	10000000	27,27772
Kogge-Stone	3499523	6500477	10000000	34,99523
Ripple-Carry	3425228	6574772	10000000	34,25228
GenusOpt('+')	1085145	8914855	10000000	10,85145

Table 5: Precision achieved with 200ps period, X $\rightarrow$ '1'

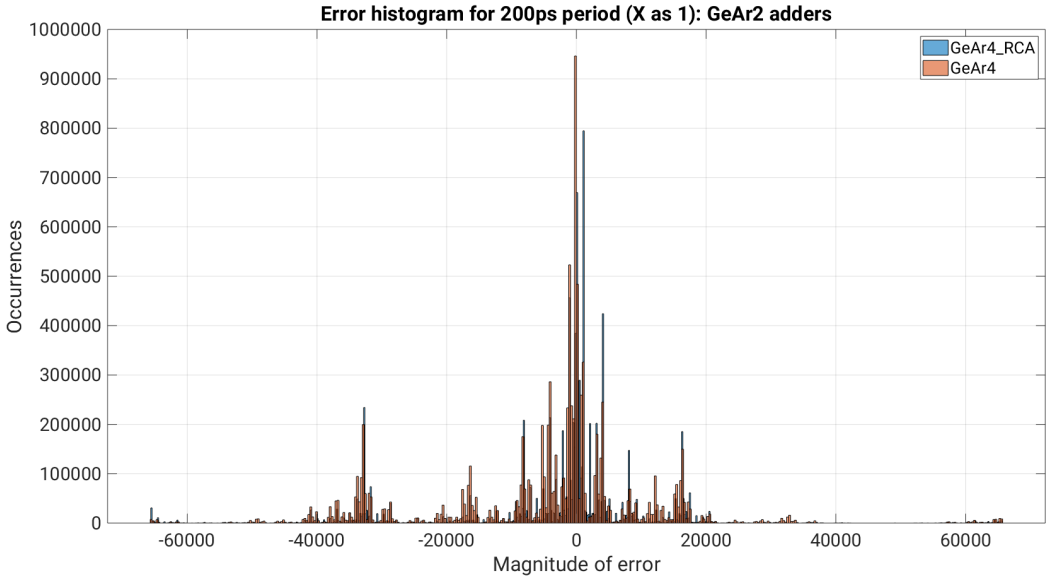

Figure 30: Error magnitude histogram for GeAr2 adder (200ps X as 1)

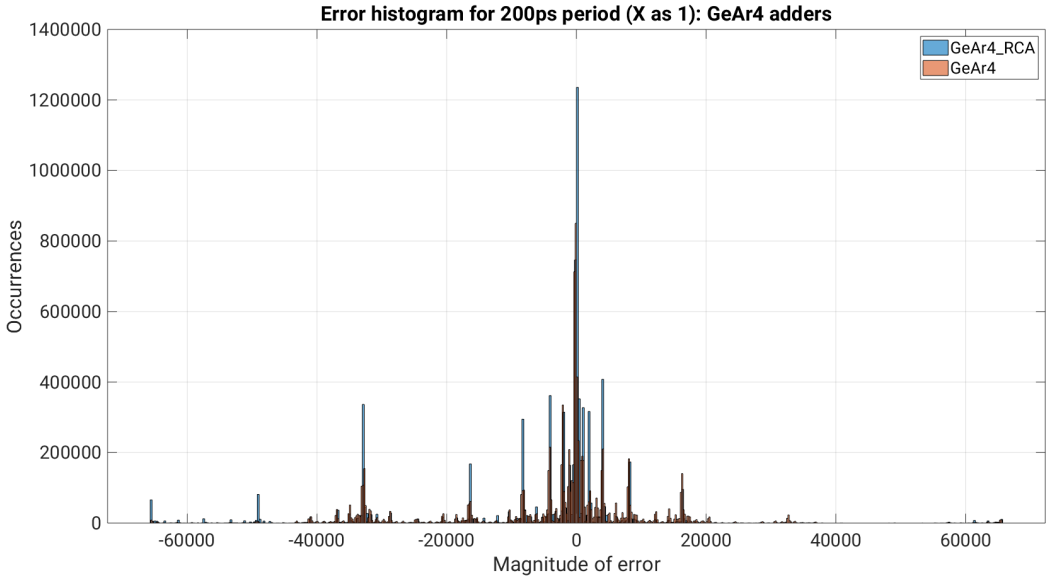

Figure 31: Error magnitude histogram for GeAr4 adder (200ps X as 1)

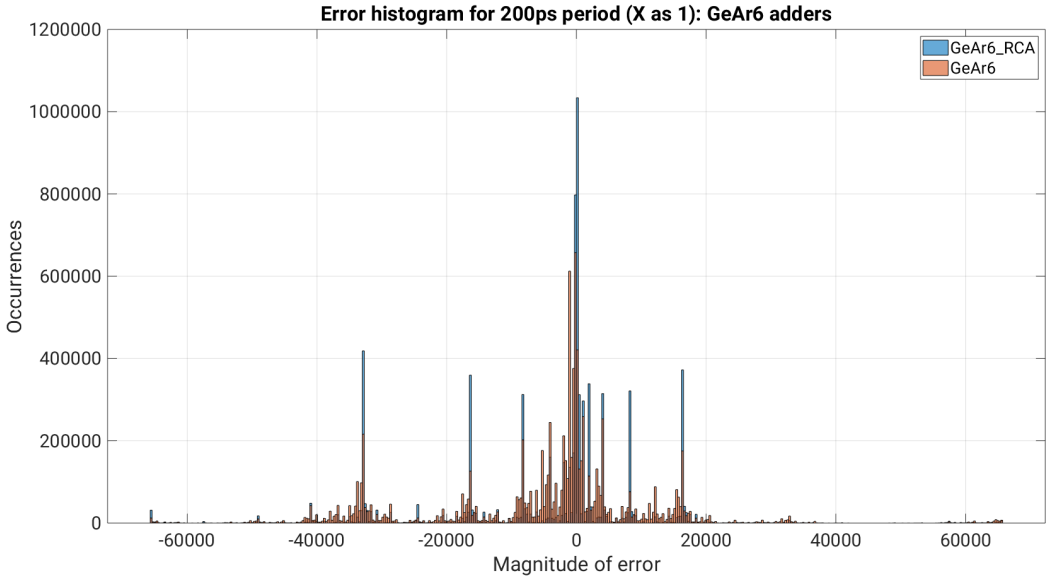

Figure 32: Error magnitude histogram for GeAr6 adder (200ps X as 1)

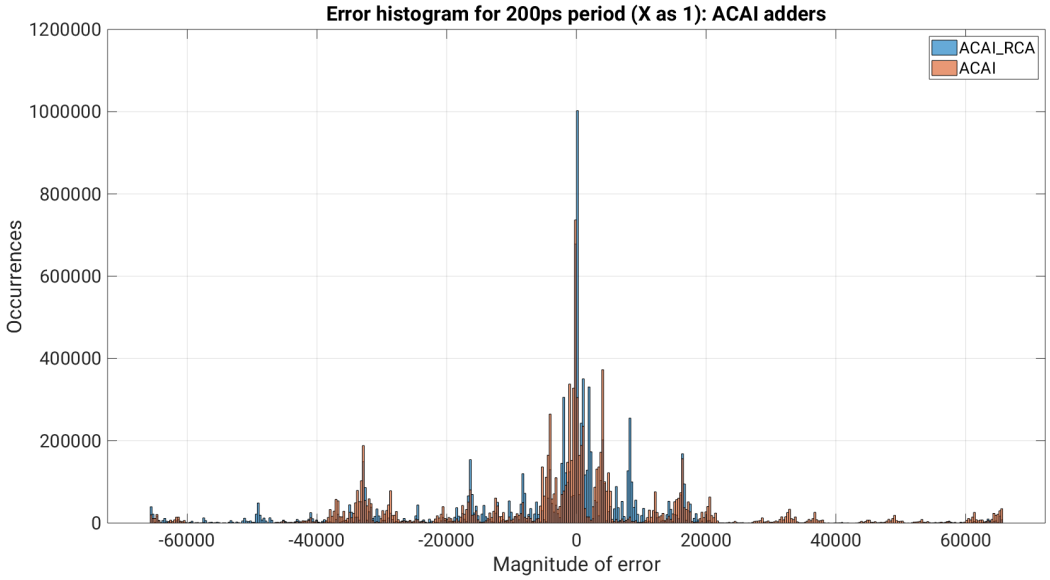

Figure 33: Error magnitude histogram for ACA-I adder (200ps X as 1)

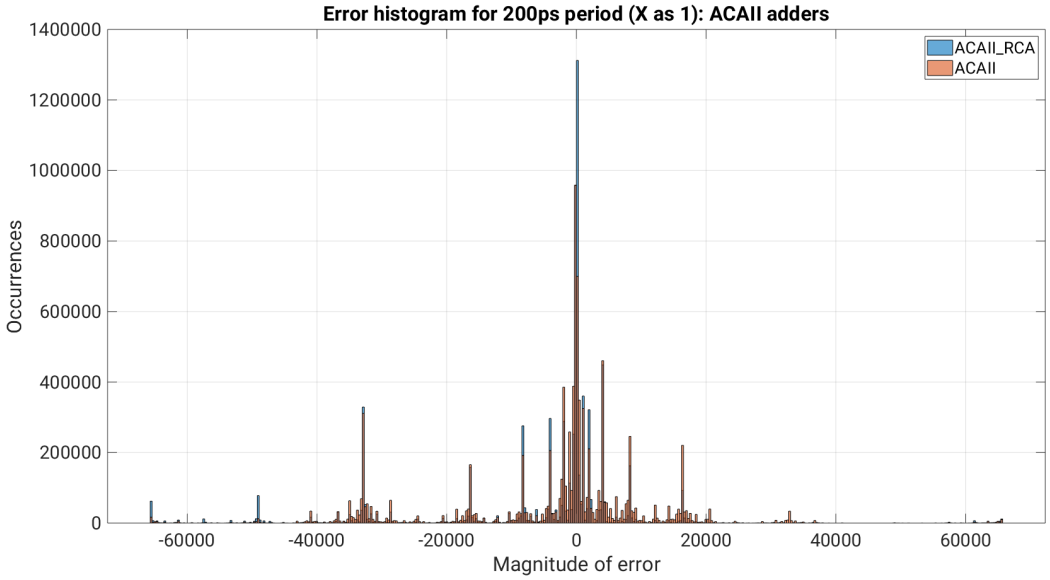

Figure 34: Error magnitude histogram for ACA-II adder (200ps X as 1)

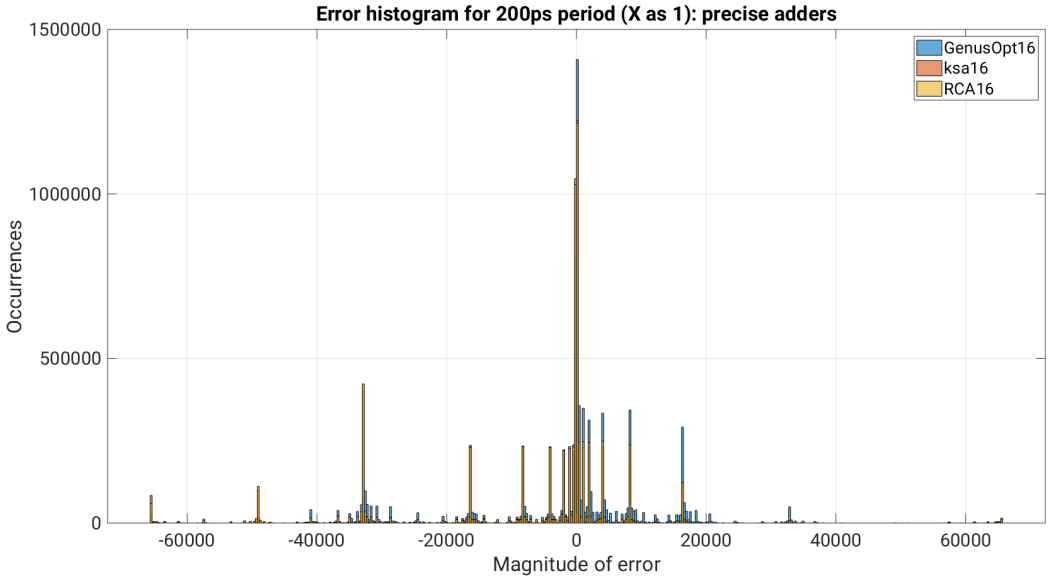

Figure 35: Error magnitude histogram for precise adders (200ps X as 1)

As mentioned in the beginning, the interesting point of these simulation results is not the final precision values but the distribution of the error magnitudes. The histograms of Figures 30, 31, 32, 33, 34 and 35 demonstrates quite clearly that the majority of the timing errors are given with low magnitudes near 0. The biggest spike is centred in 0, and smaller spikes appear when looking at higher error magnitudes. It could be said that these timing errors follow some kind of a Gaussian distribution with mean 0. It can also be appreciated that there are other bigger spikes

that represent the addition of the inherent error of the approximate adders and the timing errors. Finally, concerning the accurate adders, it can be seen that there are much less errors and that the distribution of them is not so much as a normal distribution despite of having the biggest spike in the near 0 values.

To finish with the Overclocking simulation results, the last set of results concerning the 200 ps period and X values as 0 are presented in Table 6. This table also has an additional column with the precision values achieved in the previous case, that also are shown in Table 5. Looking at the two rightmost columns, some adders offer better precision in this case but other see theirs diminished. In general, the adders the increment that these adders experience compared to the previous case is not that large in order to consider it significant, being an increase of some decimals or sometimes of 1-2%. In contrast, the majority of the adders that exhibit a decrease, experiences it in quite hard way with decreases around 6-7% of the precision compared to the other case.

Results for 200ps period					
Adder Type	**Right**	**Wrong**	**Total**	**Precision [%]**	**Precision for X → '1'**
GeAr2_Gen	378540	9621460	10000000	3,7854	2,26814
GeAr2_RCA	1044604	8955396	10000000	10,44604	16,45853
GeAr4_Gen	631712	9368288	10000000	6,31712	3,95222
GeAr4_RCA	2195689	7804311	10000000	21,95689	28,61719
GeAr6_Gen	592131	9407869	10000000	5,92131	4,59513
GeAr6_RCA	2188873	7811127	10000000	21,88873	28,66432
ACAI_Gen	118814	9881186	10000000	1,18814	0,796759
ACAI_RCA	567125	9432875	10000000	5,67125	7,61476
ACAII_Gen	765649	9234351	10000000	7,65649	7,43529
ACAII_RCA	2191603	7808397	10000000	21,91603	27,27772
Kogge-Stone	3443879	6556121	10000000	34,43879	34,99523
Ripple-Carry	2785735	7214265	10000000	27,85735	34,25228
GenusOpt('+')	1116732	8883268	10000000	11,16732	10,85145

Table 6: Precision achieved with 200ps period, X → '0'

Once again, the histogram plots are done and shown in Figures 36, 37, 38, 39, 40 and 41. Overall, there is not much difference between these histograms and the ones shown for the previous case. But, if the trend mentioned in the simulations done for 400 ps is recalled, in these histograms the errors seem to be more concentrated in the positive magnitudes, while for the case where the X values are considered as 1 these errors appear in the negative magnitudes. The histograms referring to ACAI and GeAr6 adders are quite representatives of this last behaviour, where in Figures 33 and 32 the "secondary" error spikes are clearly on the left of the main spike located at 0, while in Figures 39 and 38 these secondary error spikes are on the right side of the "fundamental" spike.

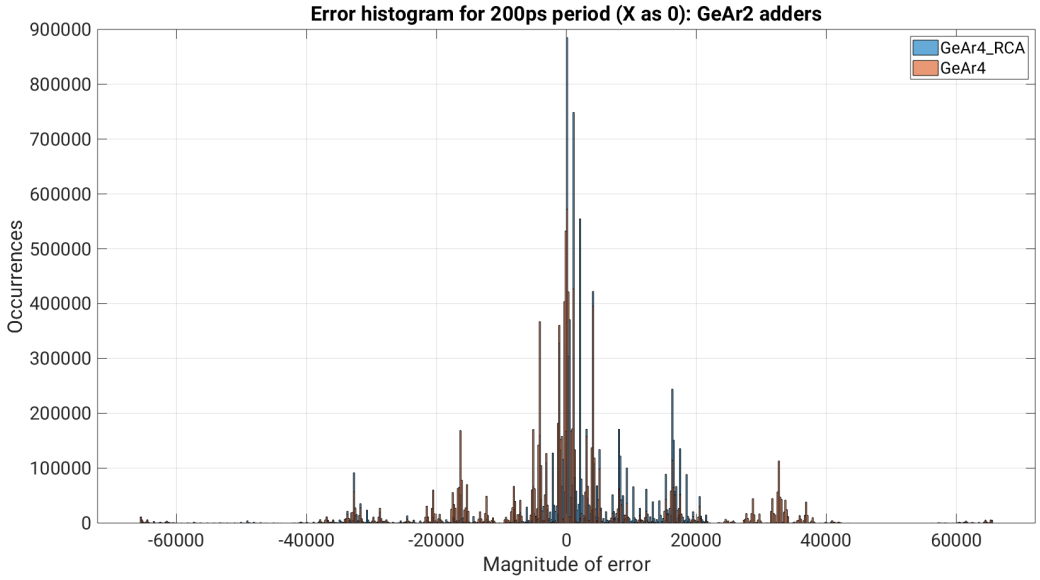

Figure 36: Error magnitude histogram for GeAr2 adder (200ps X as 0)

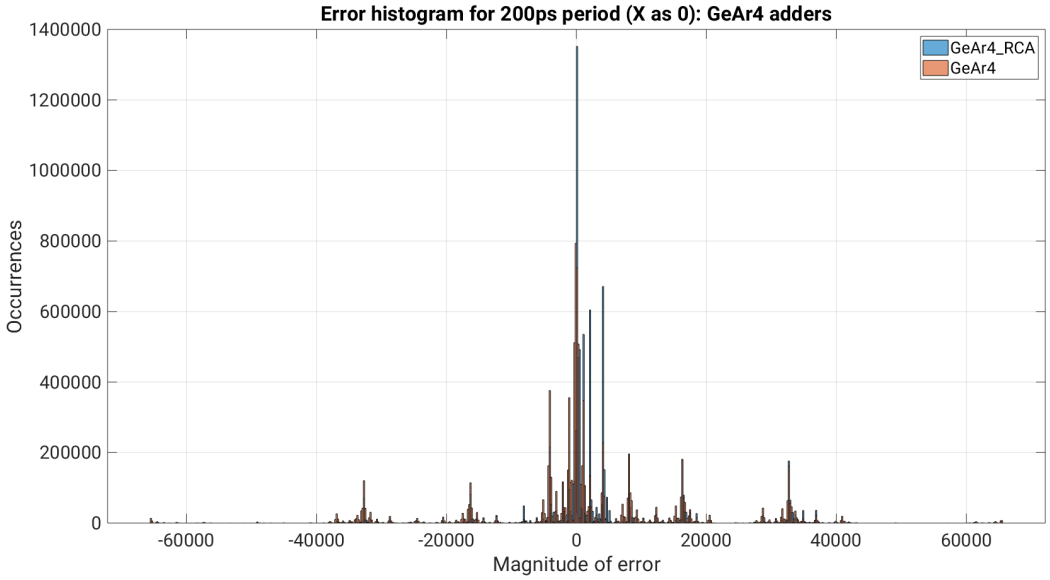

Figure 37: Error magnitude histogram for GeAr4 adder (200ps X as 0)

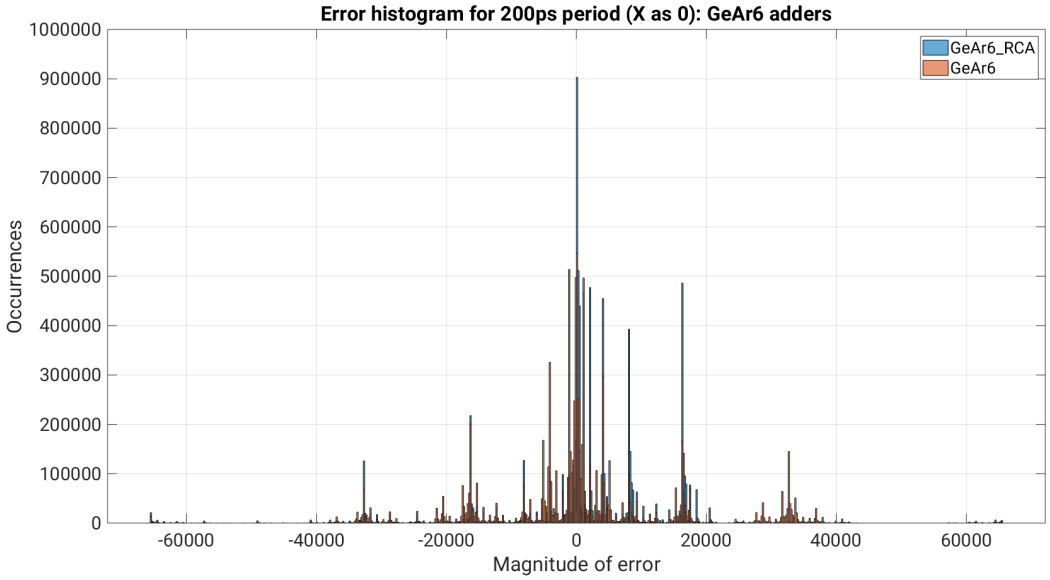

Figure 38: Error magnitude histogram for GeAr6 adder (200ps X as 0)

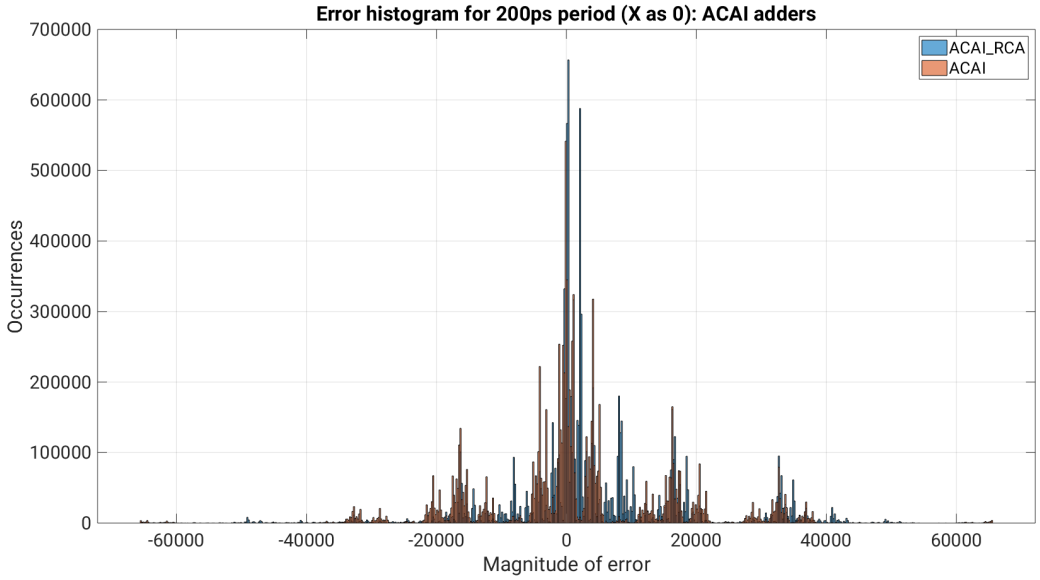

Figure 39: Error magnitude histogram for ACA-I adder (200ps X as 0)

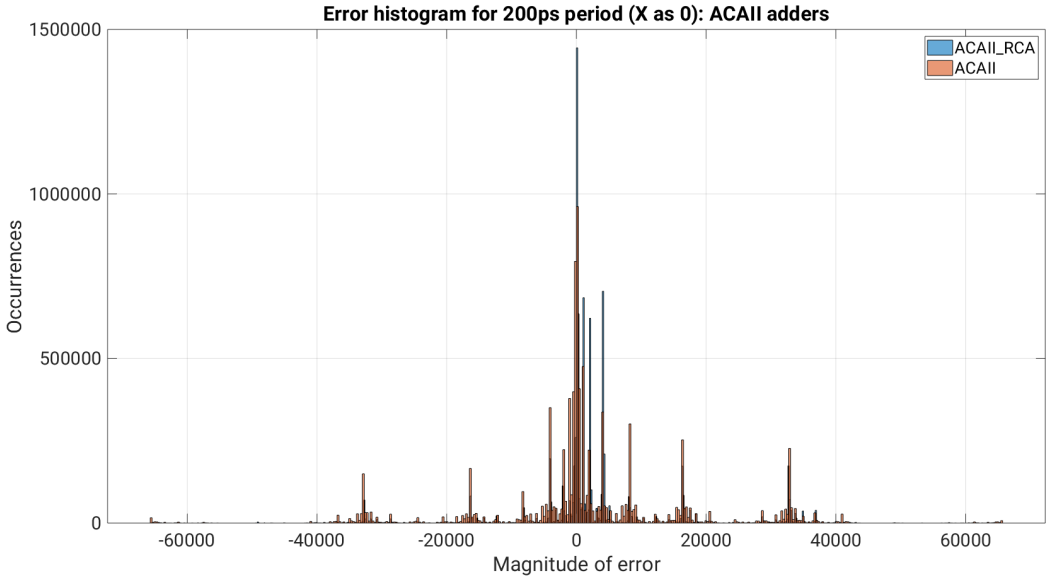

Figure 40: Error magnitude histogram for ACA-II adder (200ps X as 0)

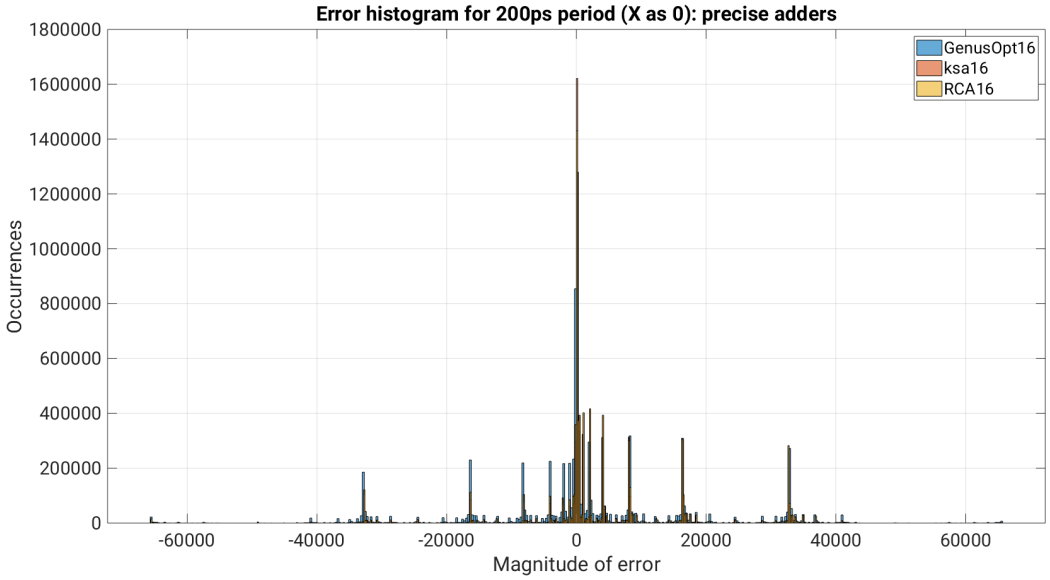

Figure 41: Error magnitude histogram for precise adders (200ps X as 0)

4.1.4 Power

To continue with the study of the approximate adders, the next results are concerning to the power consumption of the different adders. These results are obtained as it was explained earlier

in the methodology and it is a computation based on the switching activity of the nodes in the netlist.

As in previous part, the following power results are referred to the simulations performed for Overclocking and with the period values mentioned before. It also has been done for the two cases where the X values are considered either 1 or 0. Concerning the results, as the dynamic power formula is the one shown on Equation 4, increasing the frequency will have an increasing impact on the dynamic power, and thus the total power will be increased.

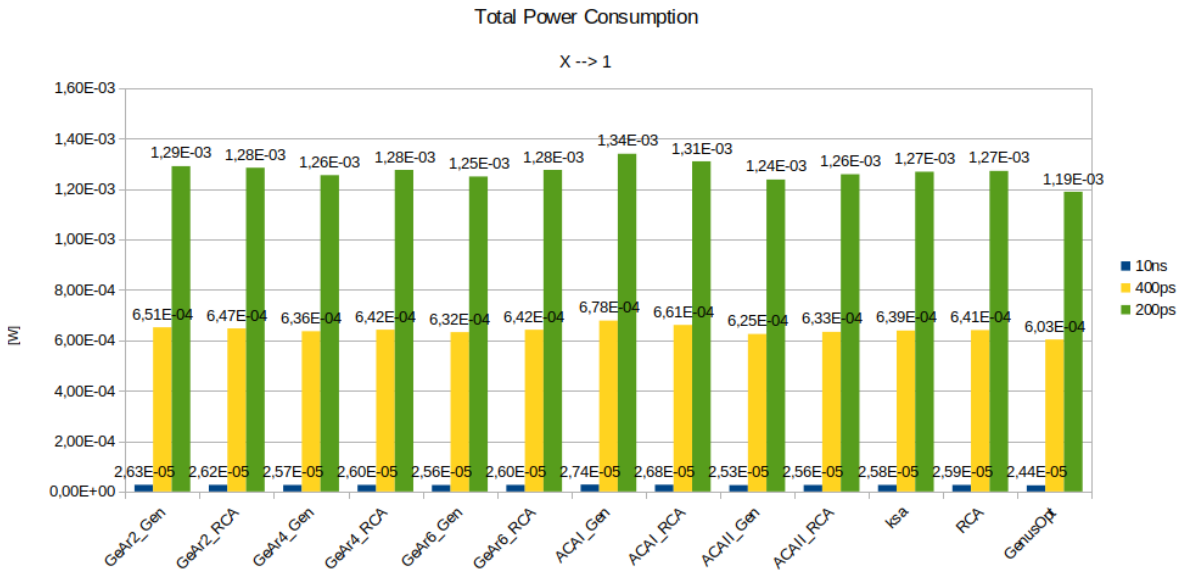

Figure 42: Total power results for approximate adders having X values as 1

As it can be clearly seen from Figure 42, the power drastically increases from period values of 10 ns to 200 ps. This result is what it was expected, based on what it was deduced from the Equation 4. It is also noticeable that the power increase experienced by the adders seem to affect all of them in a similar way, having similar shapes for every periods. Finally, as both power results for simulations done substituting the X values for 1 or 0 are exactly the same, only one result (X $\rightarrow$ 1) is represented.

Lastly, concerning the logic power consumption Figure 43 presents the obtained results. Only the results for 10 ns are presented because the other periods show very similar profiles but only changing the magnitude of the consumed power. These results are really interesting because they only refer to the power consumption of the adder structure without considering the clock and registers consumption, and it can be seen if the approximate adders offer any power advantage compared to the accurate ones.

In general, the power values are quite similar for all adders, being the results between 5 and 9 μW. GenusOpt16 is the adder with the lowest power consumption, while ACAI_Gen is the most power hungry adder. Apart from these two, GeAr4_Gen, GeAr6_Gen and both versions of ACAII offer very similar power consumptions as the other two accurate adders.

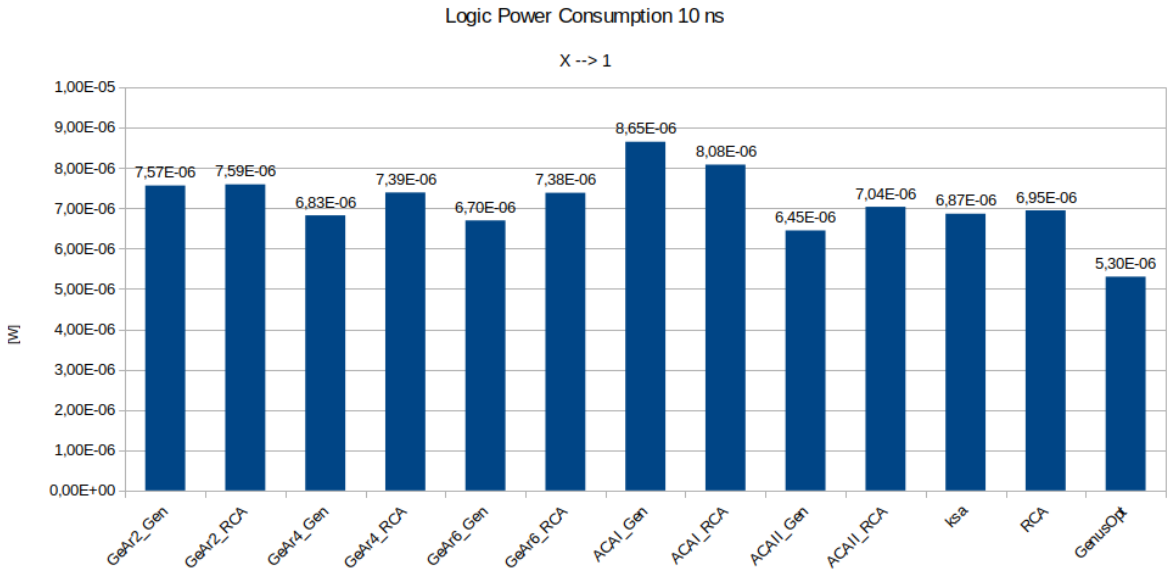

Figure 43: Logic power consumption results for approximate adders having X values as 1

4.1.5 Voltage Underscaling results

To end with the approximate adders study, the results obtained for the Voltage Underscaling simulations will be presented in this last part. But first, the basic ideas behind this simulation type are going to be recalled from the Subsection 3.4. This simulations consist on obtaining different SDF files with delays calculated for a same netlist but different voltages. The effect of calculating these new SDF files for lower voltages is that the delays corresponding to the paths are bigger, which is somehow what would happen in these adders were exposed to a voltage underscaling condition.

In order to calculate this new SDF files, two methods are employed. In order to make a proper voltage underscaling study, both temperature (T) and process parameters must be kept constant and then change the voltage value. As the synthesized netlist uses the library characterized for 0.8V and 25°C, the only library that presents equal temperature and process is the one characterized for 0.65V. In order to extend the study, fictitious voltage values were extrapolated using the Tempus timing analysis tool. For this purpose, three libraries with the same process and temperature values but different voltages (0.9V, 0.8V and 0.65V) were used. Using this method, SDF files for the fictitious voltages of 0.6V, 0.55V, 0.5V, 0.45V and 0.4V.

The next plot in Figure 44 show the precision for each adder at every voltage mentioned above. Note that this precision value is calculated using the same formula as in Equation 7, but this time it is not display as a percentage, and the precision is constrained between 0 and 1. It is also worth mentioning that these simulations are performed for X values considered as 1 as there is not much difference between the two cases.

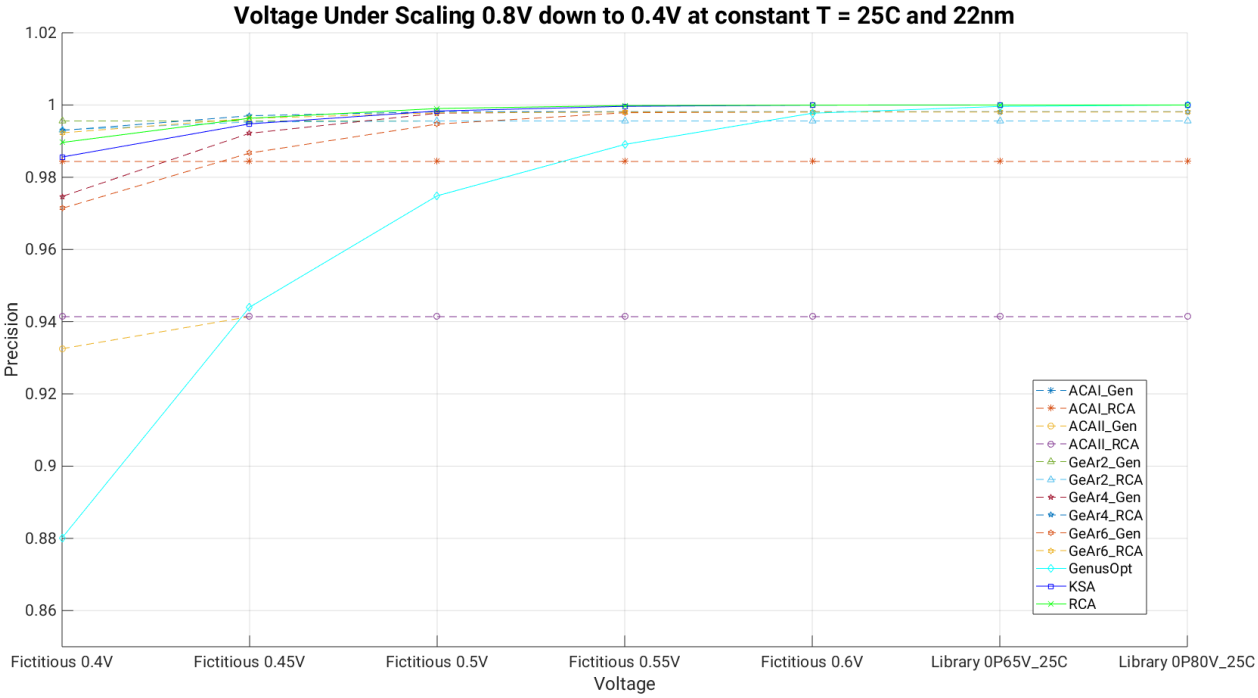

Figure 44: Voltage underscaling results for studied adders

In Figure 44, the continuous lines represent the accurate adder's precision evolution, while the dashed lines represent the approximate ones. As it can be observed, apart from the GenusOpt16 adder, the voltage has to be decreased quite a lot to start observing precision loss. In general, from 0.5V on, every adder starts to have their top precision compromised. Some of them decrease more abruptly than others, for example GeAr4_Gen and GeAr6_Gen. The accurate adders RCA and KSA also see their precision compromised and they start decreasing with bigger slope than some other adders which offer higher (not very much) precision values at 0.45V and 0.4V. This are GeAr2_Gen with the highest precision and GeAr4_RCA and GeAr6-_RCA with a very similar precision. Lastly, it is worth noticing that even at 0.4V there are still some approximate adders that do not experience any decrease in their precision, as they are ACAII_RCA and ACAI_RCA. This last adder, offers almost the same precision value as the accurate adders at 0.4V without having its own precision affected. This means that this adder has the potential of offering a better precision value than the accurate ones at lower voltages.

4.2 Multipliers

To continue, the results obtained for the approximate multipliers are presented in this subsection. The structure of the subsection is exactly the same as for the adders explained before: first the results obtained from the synthesis process are presented (Slack and area), then the errors and precision and the power obtained from the overclocking simulations are provided to finish with the resulting plots for the voltage underscaling simulations. Due to lack of time, the amount of

studied multipliers is not as big as the number of adders and the results may not be as complete as these last ones, but the results obtained are still very interesting and revealing in terms of deciding their inclusion or not.

4.2.1 Setup slack

Same as in the adders, the syntheses of the multipliers are performed using the same technology library files provided by Globalfoundries for 22 nm processes. The developed designs of the multipliers are synthesized also for a period value of 10 ns.

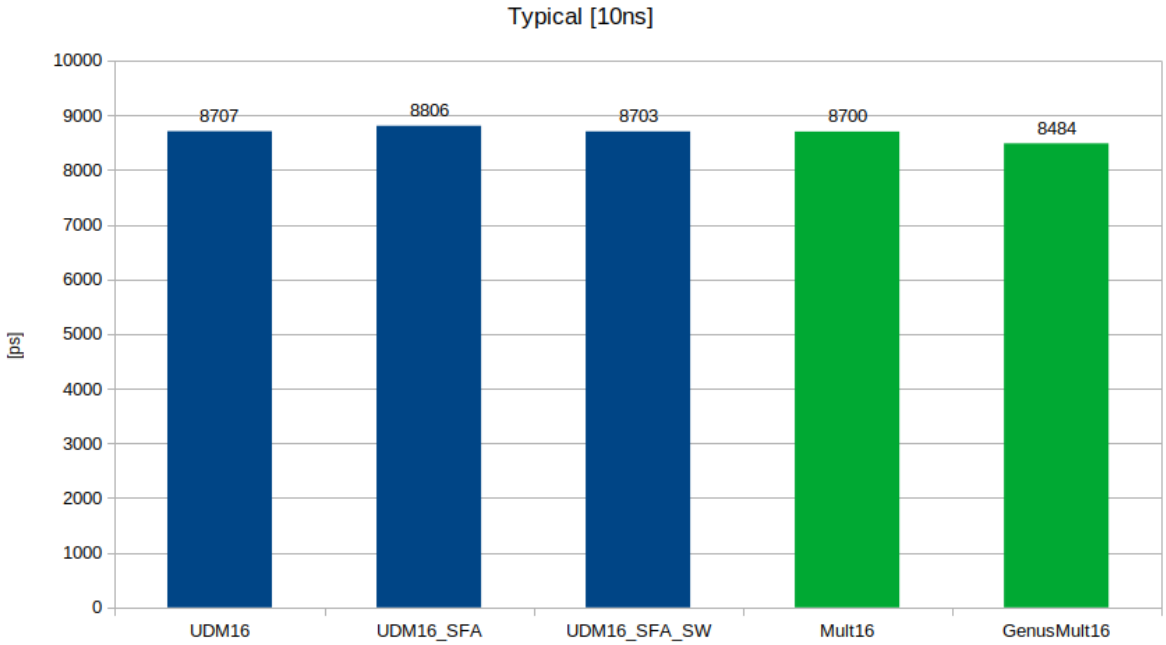

Figure 45: Slack results for different multiplier implementations

The slack values obtained, depicted in Figure 45, are quite similar for the majority of the multipliers, having a maximum variation of 3.65%. Apart from GenusMult multiplier that offers the worst value (around 200 ps smaller than the majority), the multipliers slacks are between 8700 and 8800 ps. Both the UDM16 (only approximation source is in the 2x2 multiplying block) and the UDM16_SFA_SW (approximation source also in partial sums but done like in SW) offer a very similar slack as the Mult16 accurate multiplier. It is the UDM16_SFA approximate multiplier that makes some difference between the slack, where just optimizing the length of the partial sums it can achieve a gain of 100 ps in the slack. Even if it is not much, it could make some difference when it comes to the voltage overscaling conditions.

4.2.2 Area

About the area, there is not much to speak about. Once again, as in the case of the approximate adders, the total areas (Figure 46) are quite similar in all the multipliers and the results are

as expected: the biggest area is offered Mult16 multiplier that uses the same structure than the approximate ones but with accurate logic, then there are the approximate adders that show quite similar area values being the most optimized one the smaller one, and finally there is the GenusOpt16 multiplier which has been implemented using the "$*$" sign and letting the synthesizer optimize the design, which focuses optimizing the area instead of the timing.

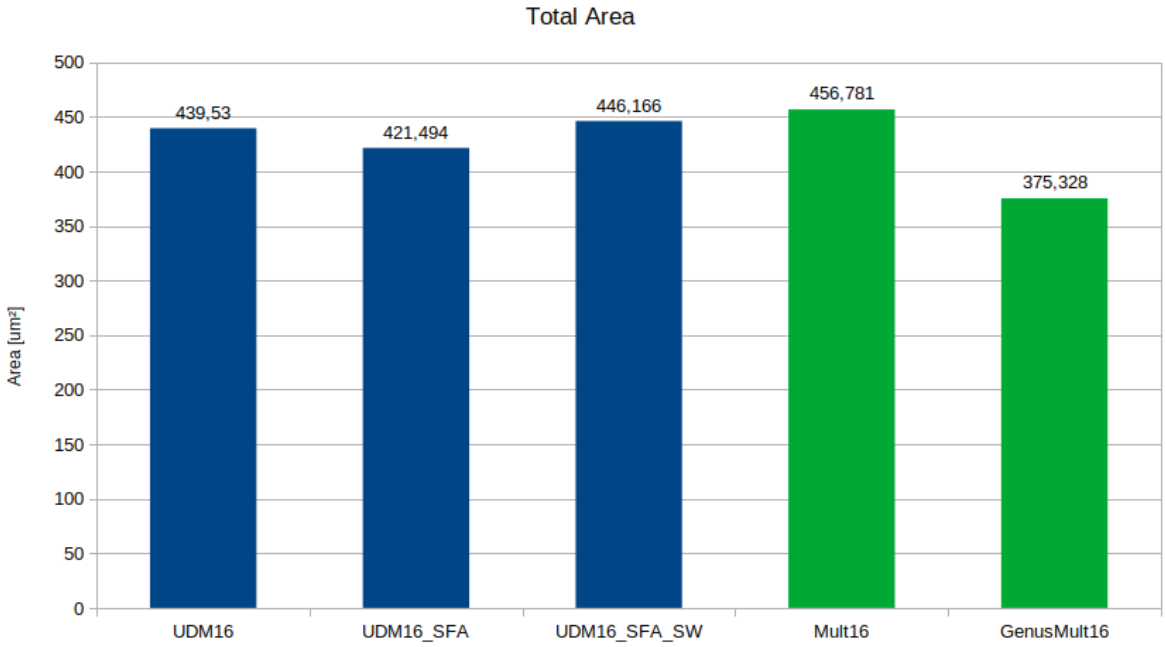

Figure 46: Total area results for multiplier implementations studied

As far as the cell areas are concerned (Figure 47), there is not much difference with the total areas, being the behaviour the same. Which means that the net areas for the multipliers are very similar around 115-126 μm^2.

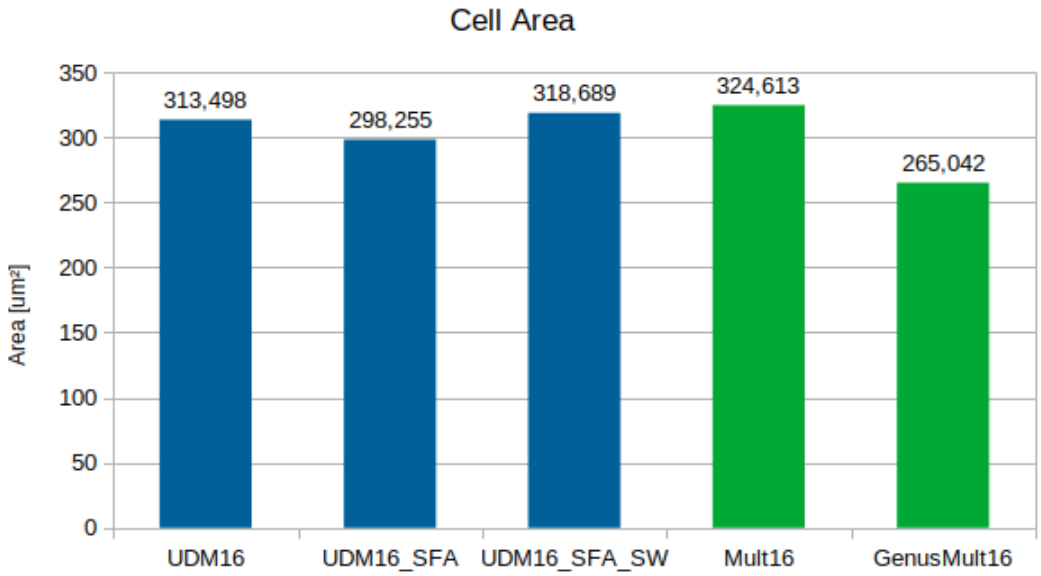

Figure 47: Cell area results for multiplier implementations studied

4.2.3 Errors and precision

As in the case of the adders, the simulations are done in overclocking conditions and using different, more aggressive, period values.

Again, a random set of operands was generated to test the multipliers, but in contrary to the adders, this time the fact of precision was not so important because the Software team already assessed the impact of using the studied multipliers in their own DNN at SW level and the results, at least from the accuracy point of view, were good enough. This was advantageous because the SW team provided some data sets were the inputs and the obtained output for each multiplier was indicated. This data sets are much more realistic than the random ones and thus will have quite more weight when considering these multiplier options for the final project. For each dataset, there are 2 simulation cases where for the timing errors, and in order to quantify the error distance, the X values are considered either 1 or 0.

This time, for the multiplier overclocking simulations, the following periods have been chosen: 1200 ps, 1100 ps, 900 ps, and 800 ps. In contrary to the adders case, where choosing the periods was done looking to the slack values, this time this is not enough. Simulations for periods of 1400 and 1300 ps were also conducted which, in principle, at least 1300 ps period should trigger some timing errors in the GenusMult16 case. But, this was not the case. The explanation to this could be that the slack metric is a very pessimistic metric, and it could happen that the obtained slack value only happens for very rare and specific transition cases that may not be contemplated in the simulation testbenches.

Random Inputs: The total precision values achieved for the different periods and the random input dataset are depicted in Figure 48. As for both cases, X as 1 and 0, the obtained results are equal, they are both represented in one unique figure.

It is worth noting that there are no timing errors appearing at period 1200 ps, and thus those values would refer to the same precision as for a period value of 10 ns, used during syntheses. The most affected multiplier is the GenusMult16, that starting from a perfect precision for 1200 ps, this value is heavily affected for the different period values, offering the worst precision option for 800 ps. Secondly, it is the Mult16 multiplier that same as the previous one starts from the top precision, and it is affected by the period changes. The decrease in precision is not that much as in the previous case, still maintaining the best precision solution for 800 ps, but the decrease in the precision is the second biggest among the multipliers. Concerning the approximate multipliers, the three of them manage to keep their precisions almost equal for every period value, but their precisions alone are not competitive enough to consider their use in top of the Mult16 multiplier. Lastly, between the three approximate multipliers, both UDM16_SFA and UDM16_SFA_SW offer same base precision value, which makes sense because their structure and basic functionality, as well as their approximation sources, are the same. UDM16 multiplier shows better precision than these last two, having only one source of approximation.

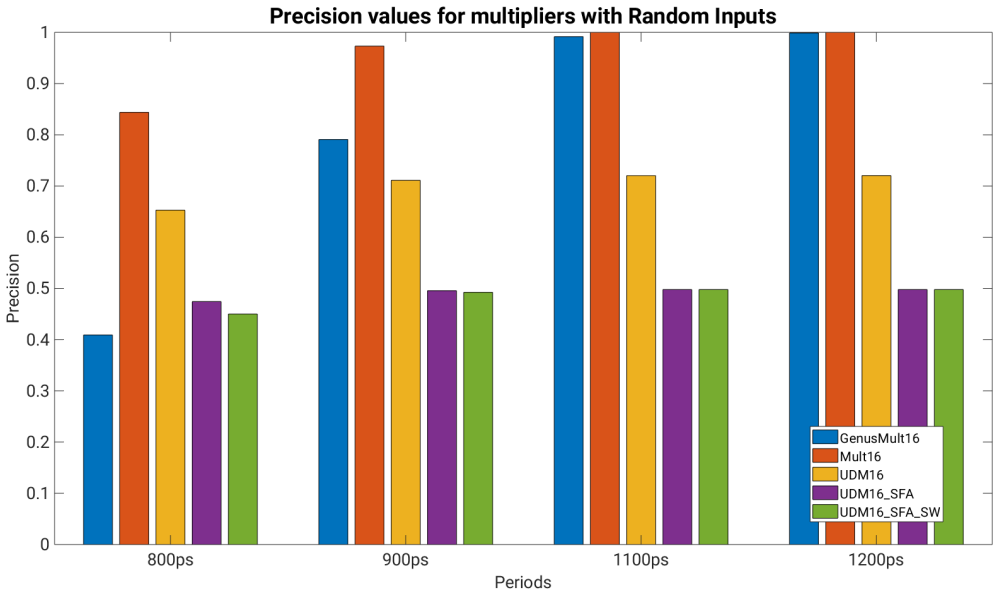

Figure 48: Precision of studied multipliers for the different periods simulated using random inputs

Same as in adders, the study of the errors was intended to be extended to the analysis of the error distance but, the amount of errors and the quantity of these was so large that the management of them was not practical.

SW inputs: Now that the results concerning the precision of the multipliers for the random input dataset have been explained, it is the turn for the dataset provided by the SW team. As mentioned, this dataset is much closer to a realistic operation of the final implementation of these multipliers specifically, and thus these results will have greater impact in the conclusions.

The same concept of total precision as in the previous case is presented now in Figure 49. This time it is a little bit different, because the outputs of the approximate multipliers are not compared with the real multiplication results but with the values that the SW team obtained from the approximate multipliers. So, the precision values of the approximate units for this multiplication set makes reference to the ability of replicating the SW team results.

This time again, the results for both cases of X values are identical and thus they are presented in the same bar plot. Every multiplier maintains the ability to perform correctly at 1200 and 1100 ps, and it is from 900 ps on that timing errors start appearing. This time the precision reduction is not as big as for the previous case, being the reduction bounded between the values of 1 and 0.8. The general behaviour is similar to the previous one, GenusMult16 having the worst precision, Mult16 the best one and the approximate ones being near Mult16. About the approximate multipliers, the UDM16_SFA option is the best one to maintain the ability to reproduce the results but still being smaller than the Mult16 multiplier.

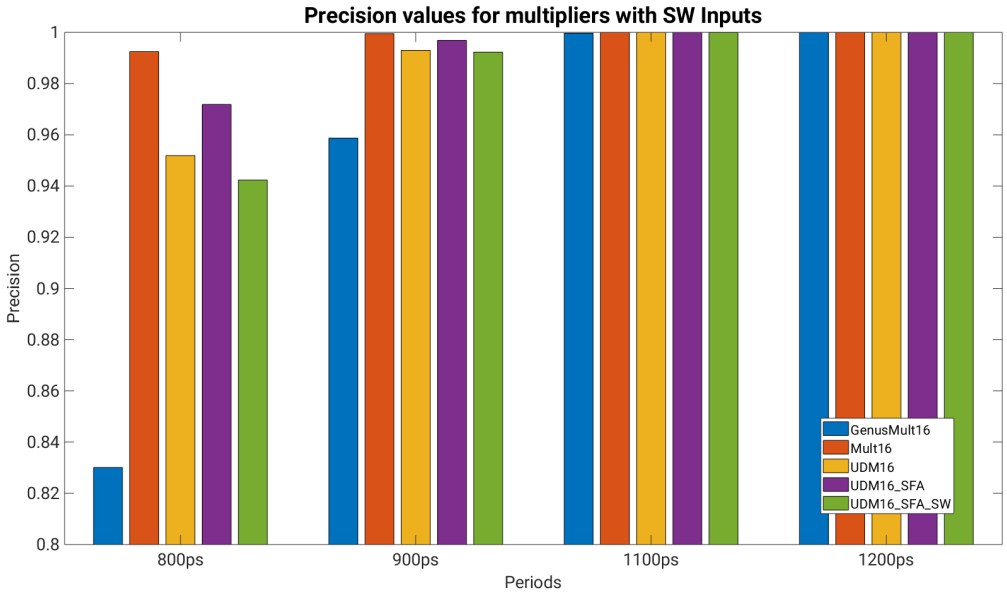

Figure 49: Precision of studied multipliers for the different periods simulated using SW inputs

Unlike in the random input dataset case, this time some error distances could be processed and plotted. More precisely, the multipliers that showed timing errors for periods of 1100 ps and 900 ps are depicted. Looking at the two bar plots above, one would think that for 1100 ps there are no errors, but analysing the results, it can be seen that for some multipliers there are indeed some errors, quite few that it seems the precision is not compromised.

At this point is where the differences between the simulations considering X values as 1 or 0 start to appear. It is very interesting to observe the two histograms in Figures 50 and 51, where the exact error magnitudes and their occurrences are displayed. It is very interesting, indeed, the fact that in the exact errors for X values as 0 appears errors for the multiplier UDM_SFA_SW but for the errors considering the X as logical 1 there are no errors. This does not mean that

there are no errors, in reality the quantity of total errors for both cases is the same because the comparison is done for the result with the X values in it. But in order to compute the error distances, the X are changed to 1 or 0. It is observed that changing the X values to 1 ”solves” each of the errors for the case of this multiplier, although they are still timing errors.

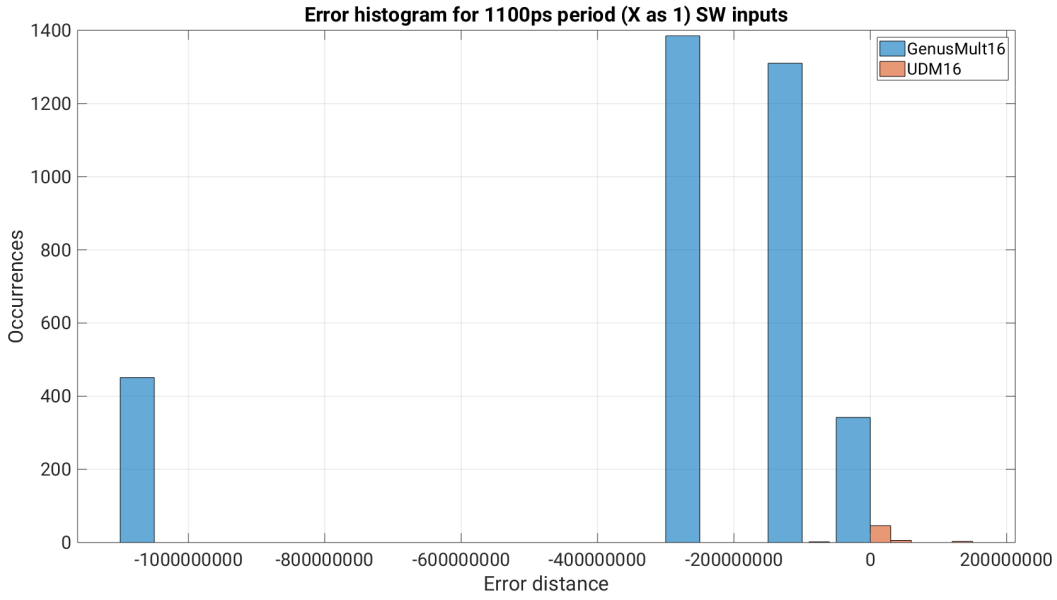

Figure 50: Histogram plot of error distance occurrences for 1100 ps (X as 1)

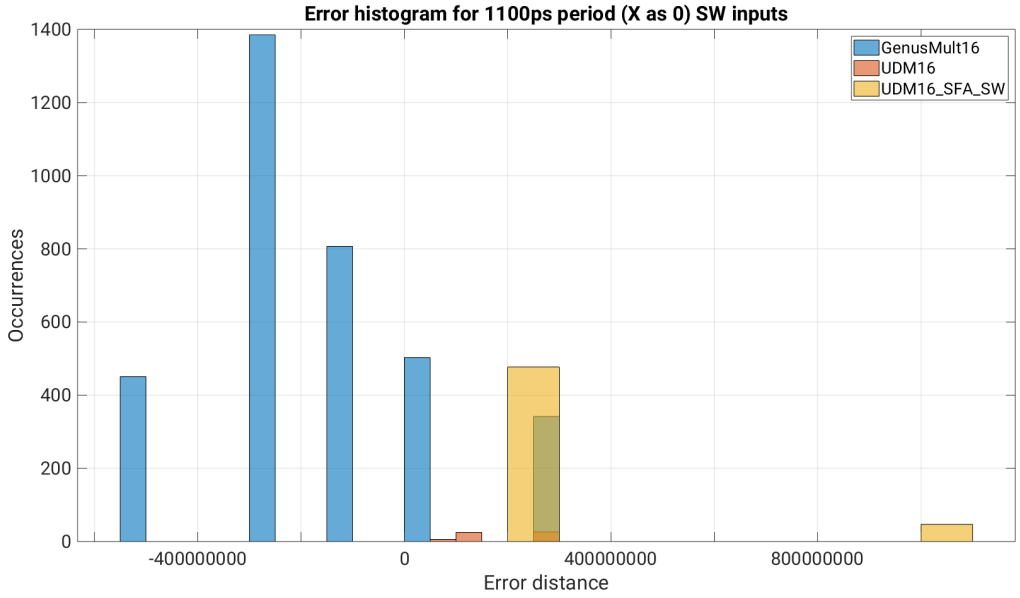

Figure 51: Histogram plot of error distance occurrences for 1100 ps (X as 0)

Apart from this, it can be seen that the errors for UDM16 multiplier are not very significant and they are not very much. Also that the errors that appear for UDM16_SFA_SW multiplier are not more than 600 occurrences for X as 0 and some of them can have quite a big error distance. And finally the GenusMult16 multiplier presents the biggest quantity of errors and tend to have considerably big and negative error distances.

As an addition, in this part the MRE is also computed to compare these multipliers as it is a good error metric. The Table 7 presents the obtained results for these values. As it can be seen, the MREs for X values considered as 1 are slightly better than their counterparts and between the non-zero MRE values, it is clear that the MREs for the approximate multipliers are much better than the one obtained for the accurate one.

Mean Relative Error for 1100 ps period		
Multiplier Type	**X as 1**	**X as 0**
GenusMult16	0.0161%	0.0156%
Mult16	0%	0%
UDM16	5.27×10^{-6}%	1×10^{-4}%
UDM16_SFA	0%	0%
UDM16_SFA_SW	0%	0.0011%

Table 7: Mean Relative Errors for multipliers with errors at 1100 ps periods

Once again the same simulations were conducted for the 900 ps period, and this time every multiplier showed some timing errors. The following plots show the obtained histograms for these simulations and for X values as 1, which are represented in Figures 52, 53 and 54. This time, for sake of visibility, the plots are represented in different plots. It can be appreciated that the two multipliers with the smallest quantity of errors are the Mult16 and UDM16_SFA, while it could be considered that the biggest amount of errors is given by the GenusMult and the rest of approximate multipliers are in between.

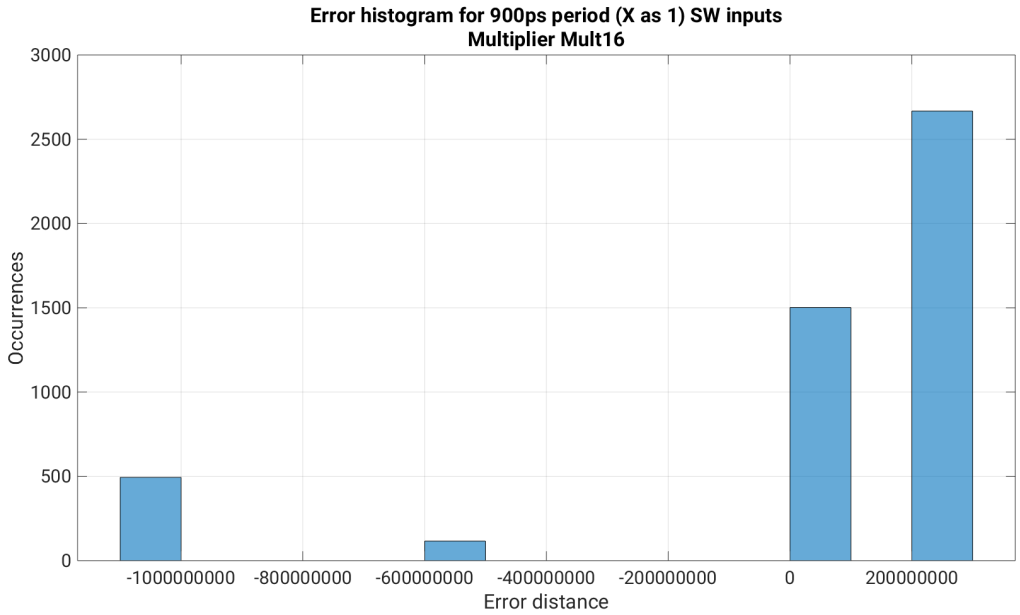

Figure 52: Histogram plot of error distance occurrences for 900 ps (X as 1): Mult16

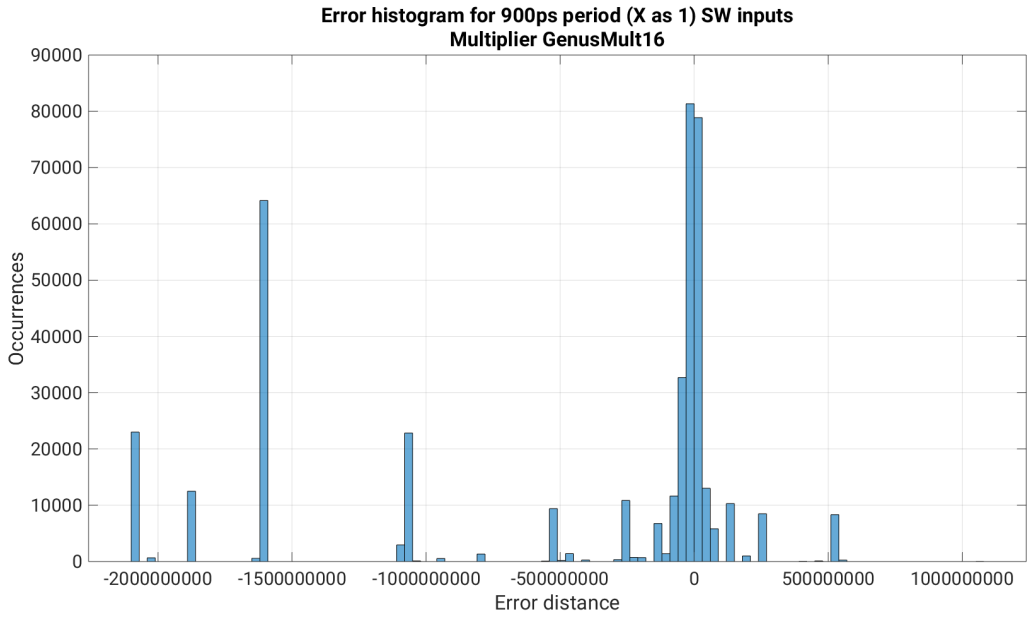

Figure 53: Histogram plot of error distance occurrences for 900 ps (X as 1): GenusMult

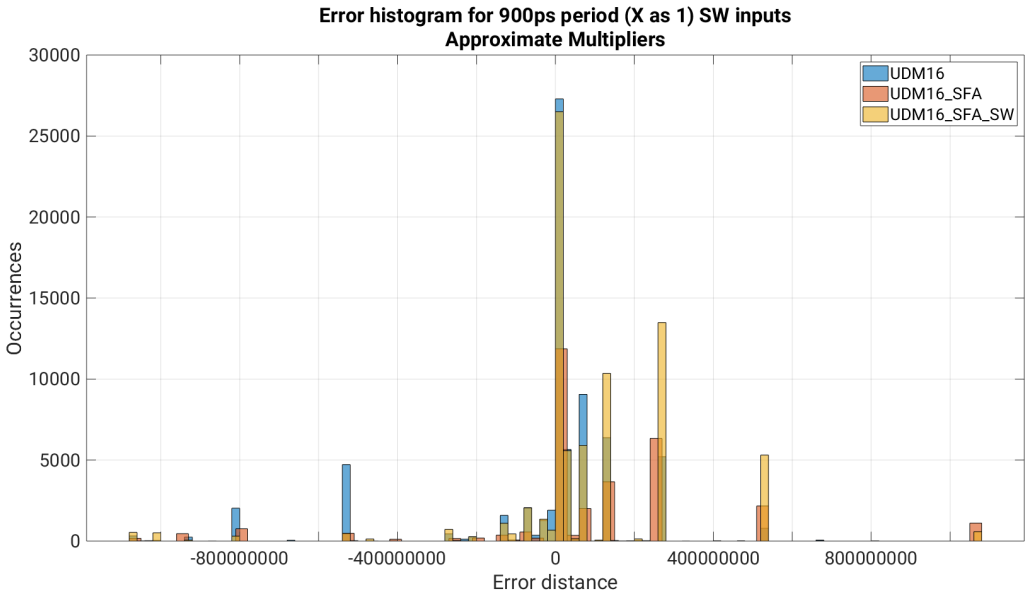

Figure 54: Histogram plot of error distance occurrences for 900 ps (X as 1): Approximate multipliers

Same histograms but for the case where the X values are considered as 0 are shown in Figures 55, 56 and 57. Comparing this results with the ones from the previous case, it is clear that for the approximate multipliers suffer more dispersion. In the previous case, there was a big spike in error magnitudes with values near 0, while in this case it is not so clear, having more spikes in higher values. It is also visible a trend for this multipliers to have positive error distances when the X values are considered as 0. Apart from this, GenusOpt multiplier seems to increase its spike centred around 0 and reduces its quantity of errors magnitudes around -1500000000. Finally, about Mult16, it seems that there is not much change but a new considerably big spike appears at a relatively high error distance.

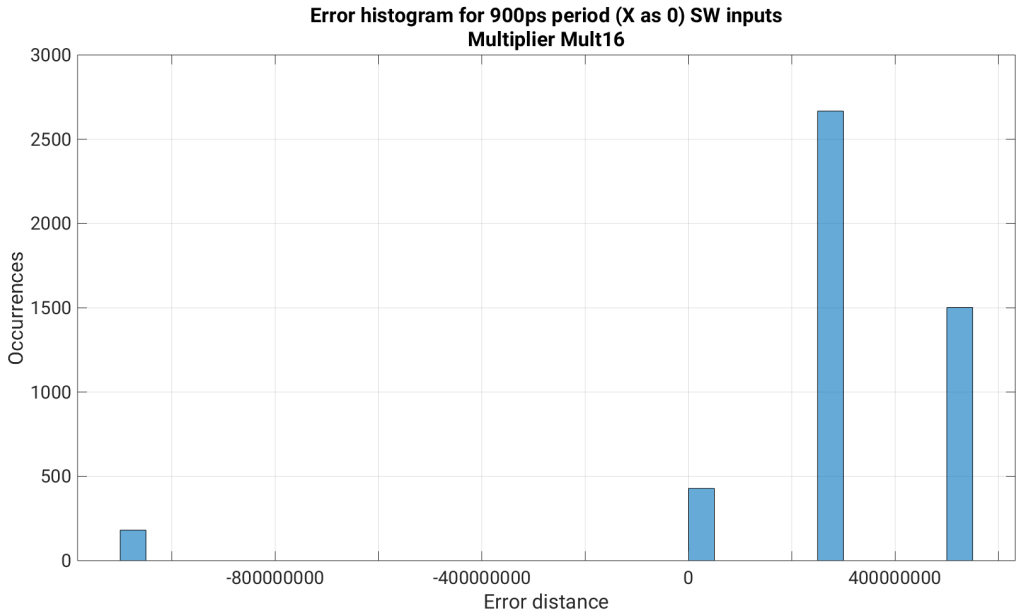

Figure 55: Histogram plot of error distance occurrences for 900 ps (X as 0): Mult16

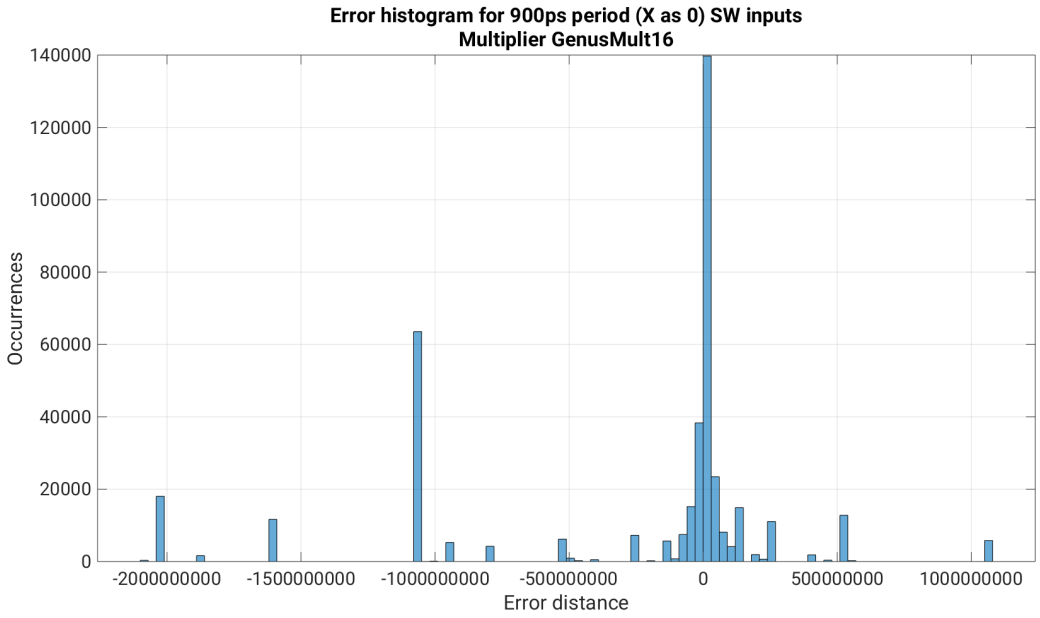

Figure 56: Histogram plot of error distance occurrences for 900 ps (X as 0): GenusMult

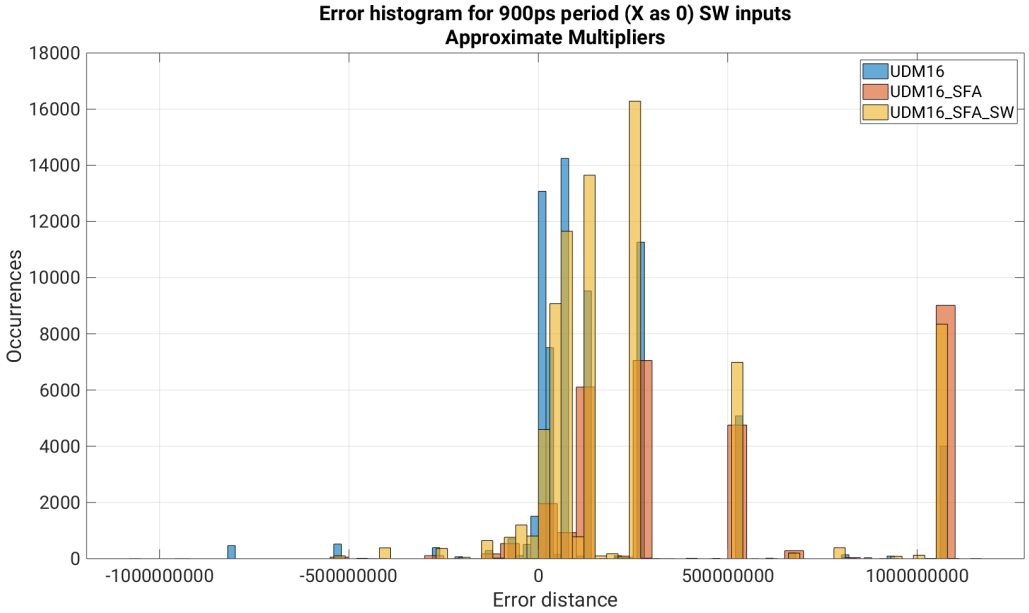

Figure 57: Histogram plot of error distance occurrences for 900 ps (X as 0): Approximate multipliers

To finally end with the error study, the same calculus as before for the MRE is performed for the simulations at 900 ps period. The corresponding MREs for each multiplier are presented in the Table 8 below. As it can be seen, worst MRE by far is offered by GenusMult16 multiplier. Apart from this, cases where X values are considered as 0 seem to be worse having other two approximate multipliers with MRE values of 13 and 18%.

Mean Relative Errors for 900 ps period		
Multiplier Type	**X as 1**	**X as 0**
GenusMult16	86.8914%	65.8079%
Mult16	0.0221%	0.0327%
UDM16	0.0817%	0.1249%
UDM16_SFA	0.0582%	13.7280%
UDM16_SFA_SW	0.1025%	18.8296%

Table 8: Mean Relative Errors for multipliers with errors at 900 ps periods

4.2.4 Power

When power consumption is concerned, same procedure was followed in order to obtain the results for the multipliers. This time, two TCF files were obtained for each multiplier, one for the simulations done using the random input dataset and another one using the SW team delivered input dataset. As the power results do not change for the cases where the X values are considered either 1 or 0, they are done for the case where X is treated as 1. The results obtained

are shown in the graphs below (Figures 58, 59, 60 and 61), having first the total power results for both datasets and then the logic power consumption.

It is really interesting to have these two results face to face. From Figures 58 and 59, the differences can be clearly appreciated. The bars in purple represent the total power consumption of the approximate multipliers, while the green ones are the total power consumption for the accurate ones. In the case of the random input dataset, the total power consumption of the approximates is smaller than the accurate ones, which is quite logical having in mind that they have less logic cells. But, in the case where the input dataset is the one provided by the SW team (which is more realistic), the total power consumption is bigger for the approximate multipliers. This means that for the specific operations that are intended to be done in the DNN, the approximate multipliers are more power hungry than the accurate ones. This could be due to the structure of the approximate multipliers, for some specific transitions the overall activity is higher than the ones for the accurate ones.

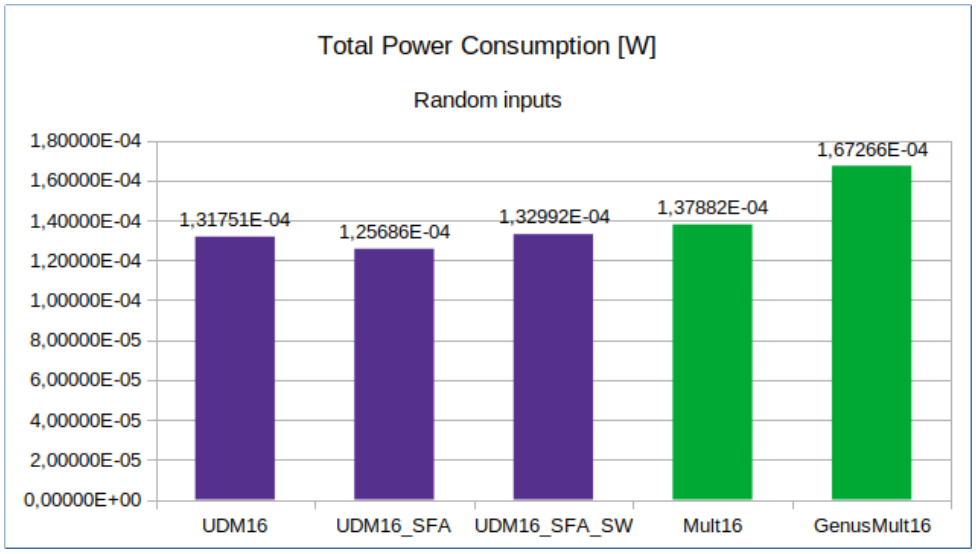

Figure 58: Total power consumption for multipliers using random input dataset

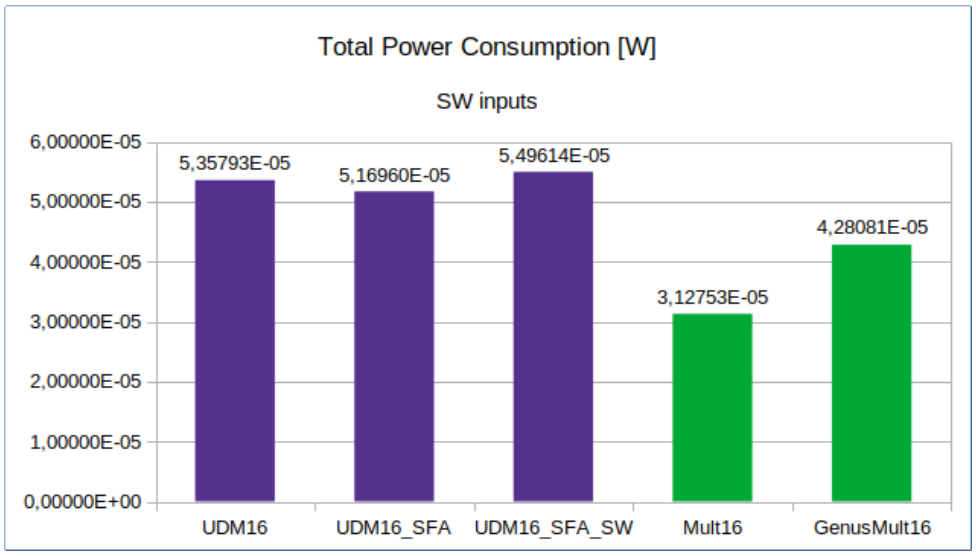

Figure 59: Total power consumption for multipliers using SW input realistic dataset

Overall, the total power consumption is 1 order of magnitude bigger for the random inputs. This is because there are fewer number of operations than in the SW inputs, but each of those operation is different. In the dataset provided by the SW team, being a more realistic scenario, there are quite a lot of consecutive operations that do not change their inputs, so the activity of the nodes is much smaller.

Finally, addressing the logic power consumption, there is not much difference about what has already been said about the total power consumption. The general behaviour of the multipliers is the same in the total and the logic power consumptions.

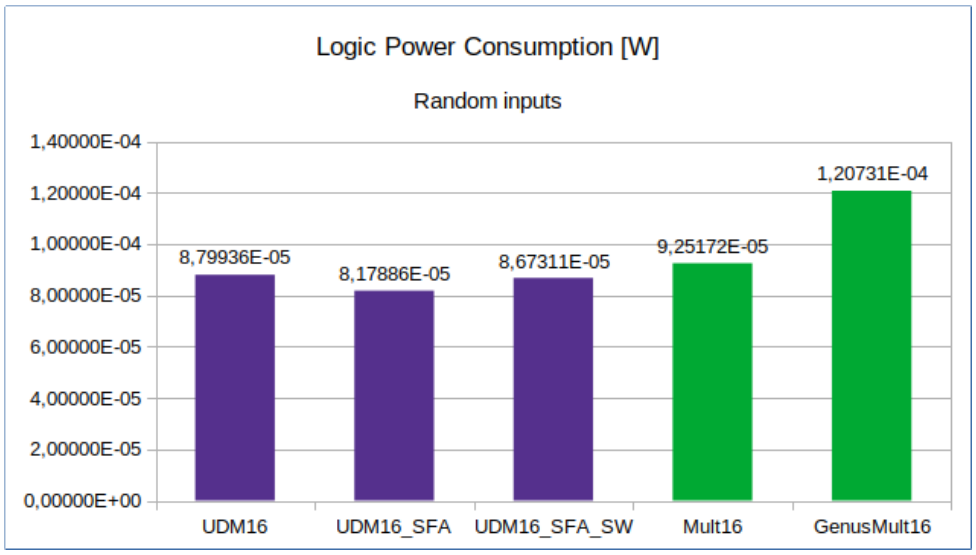

Figure 60: Logic power consumption for multipliers using random input dataset

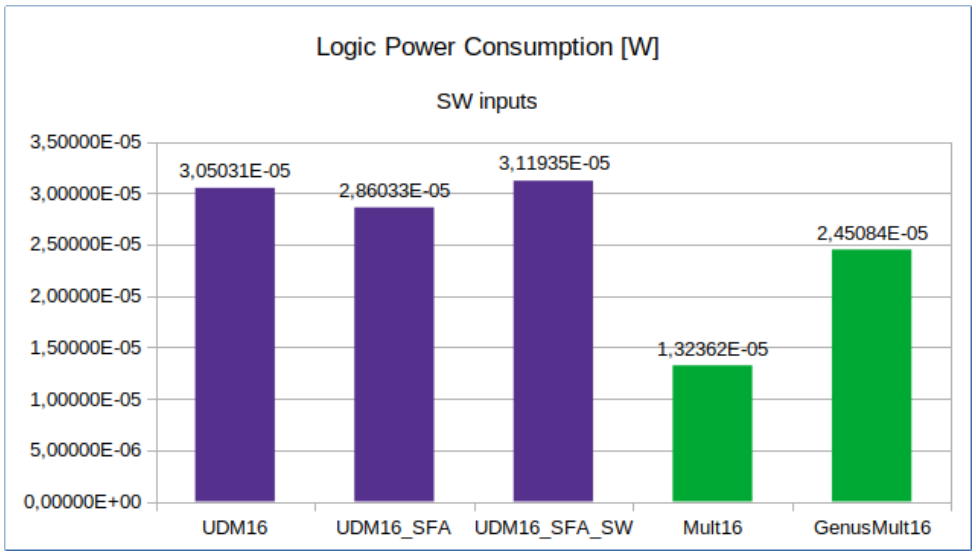

Figure 61: Logic power consumption for multipliers using SW input realistic dataset

4.2.5 Voltage Underscaling results

To end with the whole results chapter, the results for the other simulation conducted are presented. As it has been done in every multiplier's simulations, two different voltage underscaling simulations have been launched. For this simulations, the same multiplier designs have been synthesized again at a 2 ns period using the same exact script as for the synthesis at 10 ns (same constraints, library files, etc.). Similar to the adder's case, the new SDF files are obtained for the same temperature and process while the voltage is underscaled. The only existing library that matches this requirements is the same one used for the adders which is characterised for 0.65V and 25ºC. In addition to this library, other SDF files for voltages 0.6, 0.55, 0.5, 0.45 and 0.4V are calculated extrapolating the values from the other libraries. This way, the delay values for the SDF files at lower voltages will be bigger and they will simulate a voltage underscaling condition. The results for this type of simulation are presented in the Figures 62 and 63.

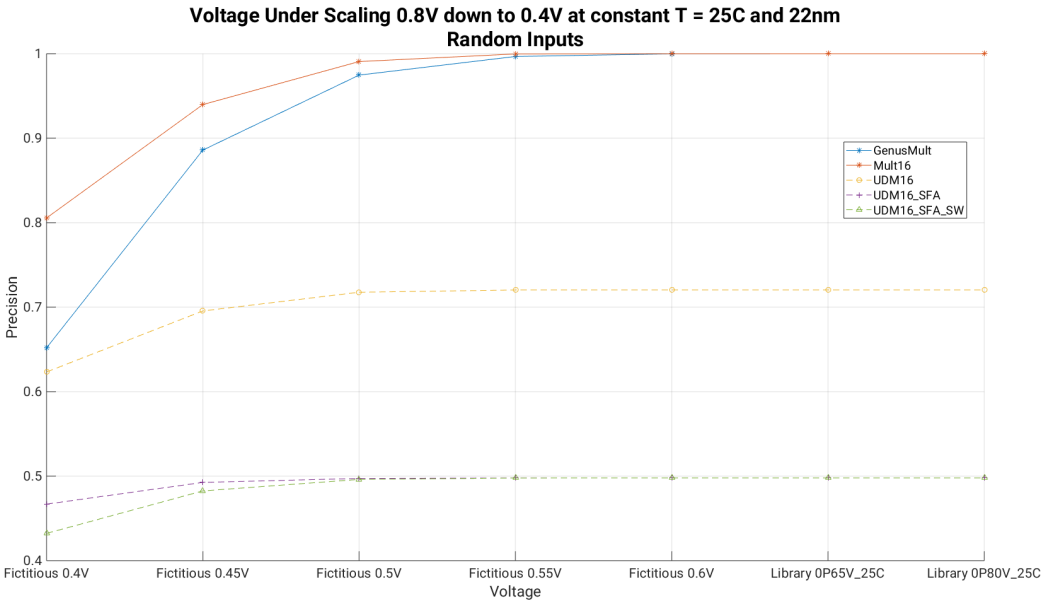

Figure 62: Precision degradation due to voltage underscaling effect for Random Inputs

In the case of the general precision values, even underscaling very aggressively the voltage, still the accurate multipliers show better precision than the approximate ones. At 0.4V, the precision of the GenusMult is quite close to the precision of UDM16, but the precision of the accurate Mult16 is far above the rest. In consideration of the approximate multipliers, it can be seen that their precision degradation starts at lower voltages than the rest of the multipliers, as for example the Mult16 starts suffering precision decrease at voltages of 0.5V and the approximate multipliers with two approximation sources at 0.45V.

Concerning the results obtained for the delivered dataset from the SW team, the same type of plot is presented in Figure 63. This time, the ability of maintaining the expected precision is tested in a more realistic scenario. Once again, Mult16 multiplier shows the best precision and best degradation rate. Second multiplier is UDM16_SFA and very close to it GenusMult16. Both of them show a considerable degradation from 0.45 to 0.4V compared to Mult16. Finally, the other two multipliers show the worst ability to preserve the intended behaviour.

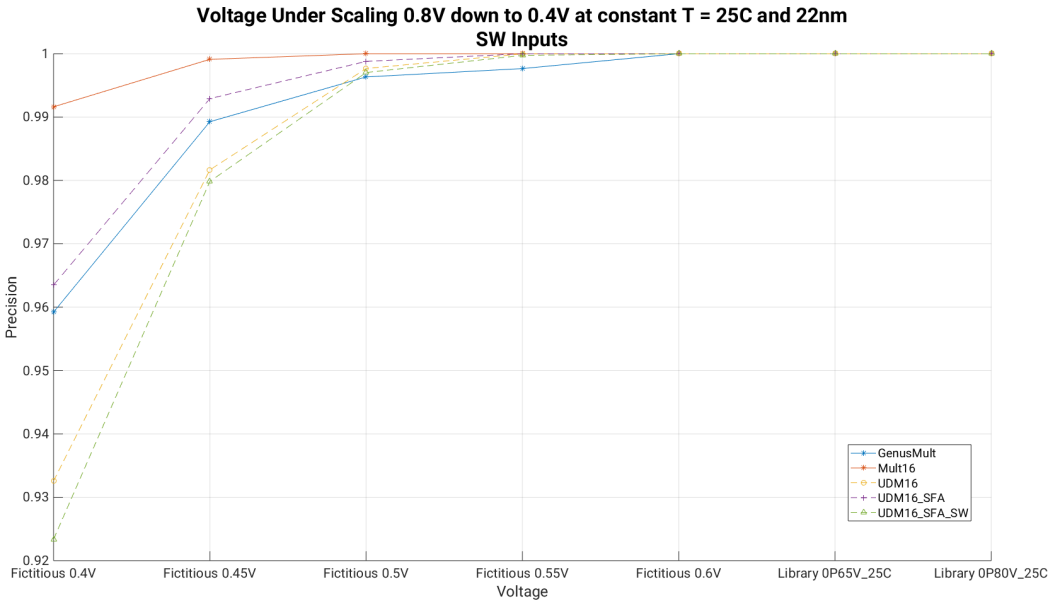

Figure 63: Precision degradation due to voltage underscaling effect for SW Inputs

5 Conclusions

To end with the study, some conclusions about the results are explained, with the objective of assessing whether the use of these arithmetic units are beneficial for the DRAC project or not. First, the conclusions for the adders are going to be explained and then the conclusions about the multipliers are exposed.

First of all, looking at the synthesis and precision results presented for the studied adders, it can be seen a clear trade-off between speed, area and precision. From the speed point of view, every approximate adder is faster than any accurate adder. When area is addressed, in general, every adder show similar total area occupancy. A relationship can be made between the speed and the area required, being normally the fastest adders the ones that require the highest area occupancy. This behaviour seems to be logical, because the structure of the studied approximate adders are based on the truncation of the carry chain, so as for a faster adder a smaller carry chain is needed, these approximate adders use more sub-adders. When precision is concerned, the intrinsic errors of the approximate adders seem to appear in magnitudes where the carry chain has been truncated. For example, for the GeAr2 adder there are three truncations and as a result, three different error magnitudes are observable, while ACAII has two cuts on the carry chain which leads to two different error magnitudes.

When the simulation results for overclocking are concerned, two period values were simulated: 400 ps and 200 ps. This last value proved to be too aggressive, forcing too many timing errors to consider these approximate architectures feasible for competing against accurate adders. But at this period value, the precision obtained even for the accurate adders was not worth, so under this overly aggressive period neither the approximate nor accurate adders would be an option. For period values equal to 400 ps, however, the obtained precisions were very competitive between them, and these approximate adders could be beneficial for applications intended for low power consumption.

Timing errors resulted as X were considered as 1 or 0 in order to evaluate the error distance or magnitude. The main difference between the two sets of simulations is that for the case where logical 1 are considered, the error distances seem to be more allocated in negative values while for the other case these error magnitudes are somehow mirrored to the positive values. In principle, this behaviour is coherent with the definition of error distance made in Equation 6. In the case where the X are considered as 1, if any of those X values would have the 0 value in the correct answer, considering it as a 1 will make the approximate result bigger than the accurate one, and thus the error distance will be negative. The same idea is applied for the other case, but the other way round.

Speaking of the power consumption, there is not any difference between simulations done for the X values considered as 1 or 0, the results are the same in both cases. In addition, the power drastically increases when increasing the frequency, which is coherent with the Equation 4. As mentioned, the power increase in consumption experienced by all the adders seem to be similar for all of them. Thus, it can be concluded that the power consumption of these adders are very similar between them. As far as the approximate adders are concerned, there are some that are below the accurate adders and others that are above their consumption, but the difference in power is almost negligible between them to the point where there would not be much difference

in implementing one or another.

As far as voltage underscaling is concerned, this kind of simulation is where the real potential of the power reduction would be observed, because reducing the voltage would decrease the energy consumption in a big manner. From this simulation, it can be concluded that decreasing the voltage for the simulated adders force some timing errors, and that at 0.45-0.4V the precision offered by this approximate adders is bigger than the ones offered by the accurate ones. These approximate adders are GeAr2_Gen, GeAr4_RCA and GeAr6_RCA. Thus, in this case the usage of these approximate adders that exhibit bigger precision at lower voltages than the accurate adders is totally recommended.

To sum up with the adder's conclusions, depending on the application and the specifications of it, some approximate adders could be beneficial. There are still a lot of approximate adder structures that may be are better than the ones studied in this document. But as far as this study is concerned, the adders showed promising behaviour at moderately aggressive periods under overclocking conditions, and on top of that some of them showed better behaviour at lower voltages, which is one of the goals of this study.

When approximate multipliers are addressed, looking at the synthesis results, the difference in slack is not as big as in the adders case, having three of the multipliers almost the same slack value. This means that the implemented approximate multipliers (except the optimized one that it only show an improvement of 100 ps which is not large enough), are not faster than the accurate ones. When the area occupancy is brought to discussion, they all occupy similar areas, but the approximate multipliers with same structure than the Mult16 show smaller areas. This is expected because the approximate multipliers with this structure use smaller sub-adders and smaller 2x2 multiplication blocks in their structures.

Talking about precision of the implemented multipliers, the study could not have been as completed as it was wanted to be, but still some interesting results have been obtained. First, concerning the random input dataset, the precision values obtained for the approximate multipliers were not even near the accurate ones. Their resiliency against the period reduction proved to be better but not good enough to beat the accurate ones. In addition, for the dataset provided by the SW input, which simulates a much more realistic scenario, again the approximate multipliers resulted in worse performance than the Mult16 multiplier. A conclusion can be deduced from these two results: the resiliency of the approximate multipliers is better when the inputs are randomly generated, but for the specific case of the DNN that will be implemented in the DRAC project they prove to have worse resiliency than the Mult16 multiplier in overclocking conditions.

Furthermore, for this last dataset, the study of some error distances for periods 1100 ps and 900 ps was possible. From these studies, the metric of MRE in computed. For 1100 ps, the approximate multipliers that show any error also exhibit a MRE considerably better than the accurate multiplier GenusMult16. For the 900 ps result, this also maintains but this time the GenusMult multiplier shows a totally unacceptable MRE value, while the MRE for the approximate multipliers only grow 0.05-0.1%. It is worth noting that the MRE values for some approximate multipliers increase considerably for the cases where the X are considered as 0.

About the power consumption of the studied multipliers, if the random input dataset is addressed

it can be concluded that the approximate ones are little less power consuming than the accurate multipliers. But, if the realistic inputs given by SW are concerned, the power consumption of the approximate multipliers is bigger than the accurate ones.

Finally, when voltage underscaling simulations results are taken to examination, in both cases the accurate multipliers show better performance at lower voltages. Even if for the random inputs the approximate multipliers show smaller or latter precision degradation, the gap is too big to be surpassed. For the SW inputs, the approximate multiplier UDM16_SFA show a slightly better performance than the GenusMult16 multiplier, but still worse than the Mult16.

To sum up with the multiplier's conclusions, and end with the study and the document, from the results obtained for the multipliers, the studied approximate multipliers do not show to be promising in almost any aspect that has been able to study. When random inputs are taken to discussion, the approximate multipliers seem to be more competitive in power consumption. These attributes could be beneficial for any other application, but in our particular case where the inputs provided by the DNN of the DRAC project will be the dataset provided by the SW team, these multipliers proved to have worse performance than the accurate ones.

References

[1] Bosio A. Rodrigues G, Lima Kastensmidt F. Survey on approximate computing and its intrinsic fault tolerance. *Electronics*, 9(4):557, 2020.

[2] A. K. Verma, P. Brisk, and P. Ienne. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *2008 Design, Automation and Test in Europe*, pages 1250–1255, March 2008.

[3] A. B. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *DAC Design Automation Conference 2012*, pages 820–825, June 2012.

[4] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. A low latency generic accuracy configurable adder. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[5] D. Shin and S. K. Gupta. A re-design technique for datapath modules in error tolerant applications. In *2008 17th Asian Test Symposium*, pages 431–437, 2008.

[6] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *2011 24th Internatioal Conference on VLSI Design*, pages 346–351, 2011.

[7] Khaing Yin Kyaw, Wang Ling Goh, and Kiat Seng Yeo. Low-power high-speed multiplier for error-tolerant application. In *2010 IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC)*, pages 1–4, 2010.

[8] P. M. Kogge and H. S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Transactions on Computers*, C-22(8):786–793, 1973.

[9] Gopalakrishnan Seetharaman, Bala Venkataramani, and G. Lakshminarayanan. Vlsi implementation of hybrid wave-pipelined 2d dwt using lifting scheme. *VLSI Design*, 2008, 06 2008.

[10] CADENCE. *Gate-level Simulation (GLS): A Quick Guide for Beginners*.

[11] Martí Caro Roca. Unitats funcionals aproximades per a processadors de baix consum. Bachelor Thesis, 2019-2020. Universitat Politècnica de Catalunya (UPC) - Facultat d'informàtica de Barcelona (FIB).