Labor SoC Design Report - Accuracy evaluation for approximate circuits

Alhasan Bondok

Alhussein Bondok

Sebastian Schuster

Stefan Bajtela

Supervisor: Nahla El-Araby

Department of Information and Communication Technology

Vienna University of Technology

March 2023

: `https://github.com/1997Al/SoC_Lab`

Contents

1 Organisation

The team consists of four members with the following division of tasks:

Name	E-Mail	Task
Alhasan Bondok	e11770995@student.tuwien.ac.at	software
Alhussein Bondok	e51867866@student.tuwien.ac.at	hardware
Sebastian Schuster	16087@student.tuwien.ac.at	hardware
Stefan Bajtela	e1525643@student.tuwien.ac.at	software

2 Introduction

Approximate computation involves using algorithms that produce approximate answers to computationally difficult problems. These algorithms trade accuracy for performance/energy efficiency/area, allowing for computations to be completed in much less time than exact algorithms. Approximate computation is often used in areas such as data mining and natural language processing, where a precise answer is not necessary and a fast computation is more important.

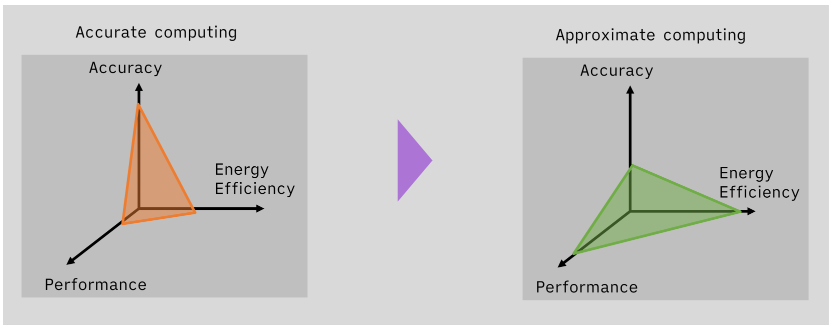


Figure 1: Comparison between approximate and accurate computing

To determine whether the approximation is suitable for the respective application (tolerance), we primarily use the difference between the exact result and the approximated calculation. The verification of this metric can typically be achieved using two different approaches:

- **Simulation:** Simulation in the context of hardware is the process of using a computer to emulate the behavior of a real, physical system. This is done by creating a model of the system and feeding it with different inputs while analyzing its output behavior. The problem, however, with this approach is that to be able to verify the system as a whole, every input combination has to be tested. For most systems, this is not feasible, as the simulation would not be possible from a temporal point of view.
- **Formal Verification:** Formal verification is a process used to prove the correctness of a program or system with respect to a given specification. It involves mathematically proving that the implementation meets the specification. This approach gives us better coverage of the cases and

thus allows us to achieve a better overall view of the shortcomings of the system. Formal verification is often used in safety-critical systems, such as medical devices, automotive systems, and aerospace systems.

Our approach is more experimental, as we use a combination of both methodologies. This is achieved by a limited scope simulation and a curve-fitting algorithm that provides us with mathematical descriptions of the error metrics.

3 Project Structure

The project consists of software and hardware parts. The approximated adder circuits in figures 2 and 3 were used for the demonstration. The Least Significant Bit (LSB) Truncation Adder simply replaces the approximated bits of the result with zeros, while the Lower Or Adder (LOA) uses or gates for the addition of the approximated bits.

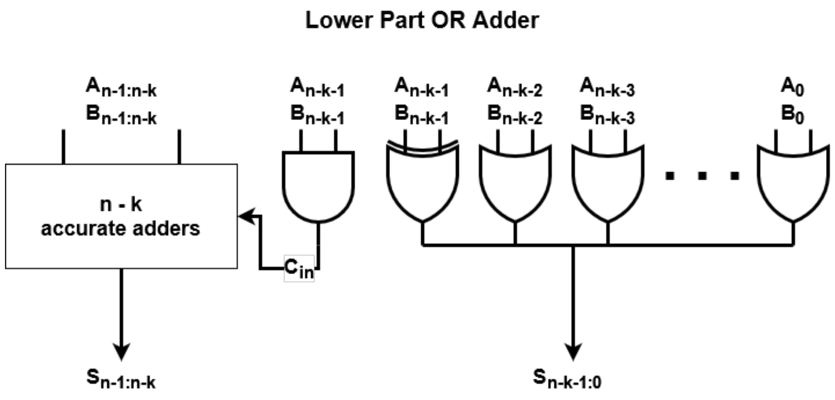


Figure 2: lower part OR adder used

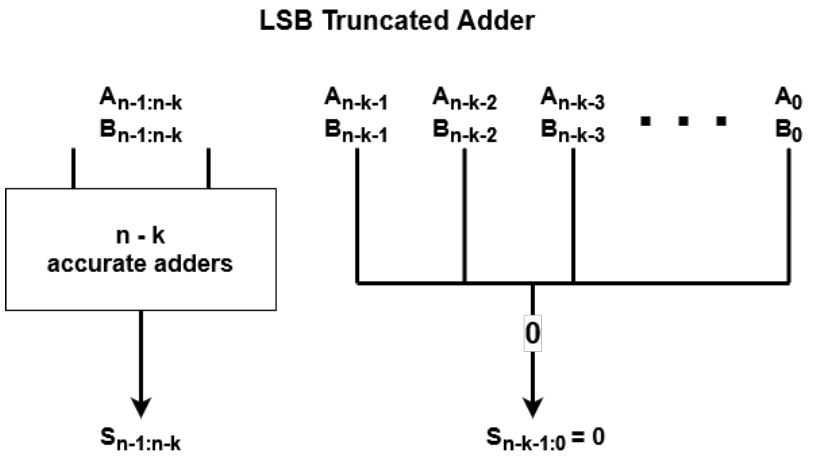


Figure 3: least significant bits truncated adder

3.1 Software Structure

The software part was implemented in python and consists mainly of two files:

- **approximateAdders.py:** This file contains the software implementation of the approximate adder circuits that are used for proof of concept. These algorithms emulate the behavior of the real adder circuits while excluding the temporal component. In the next steps, they are replaced by their in-hardware synthesized counterparts on the Field Programmable Gate Array (FPGA).
- **topModule.py:** This file contains the most important components of the system: the computational and the curve fitting modules. These modules perform the limited scope simulation (time efficient), whose results are then utilized to calculate the mathematical descriptions of the error metrics through a curve-fitting algorithm using significantly less information and time than an only simulation approach.

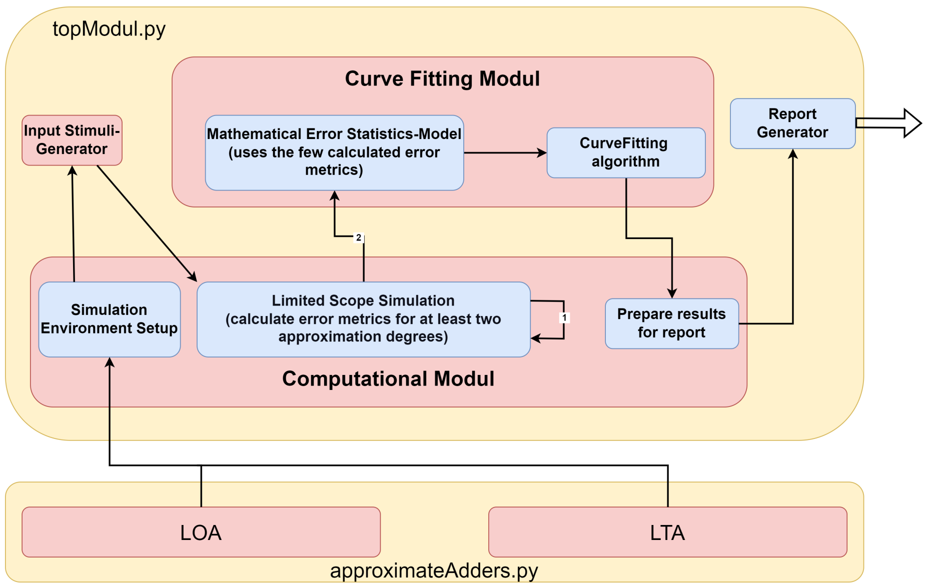

topModul.py
Curve Fitting Modul
Input Stimuli-Generator
Mathematical Error Statistics-Model (uses the few calculated error metrics)
CurveFitting algorithm
Report Generator
2
1
Simulation Environment Setup
Limited Scope Simulation (calculate error metrics for at least two approximation degrees)
Prepare results for report
Computational Modul
LOA
LTA
approximateAdders.py

Figure 4: software structure for proof of concept

3.1.1 Software Algorithm

The following flow chart illustrates the software process that was used for proof of concept. For more details please refer to the documentation provided in the source code.

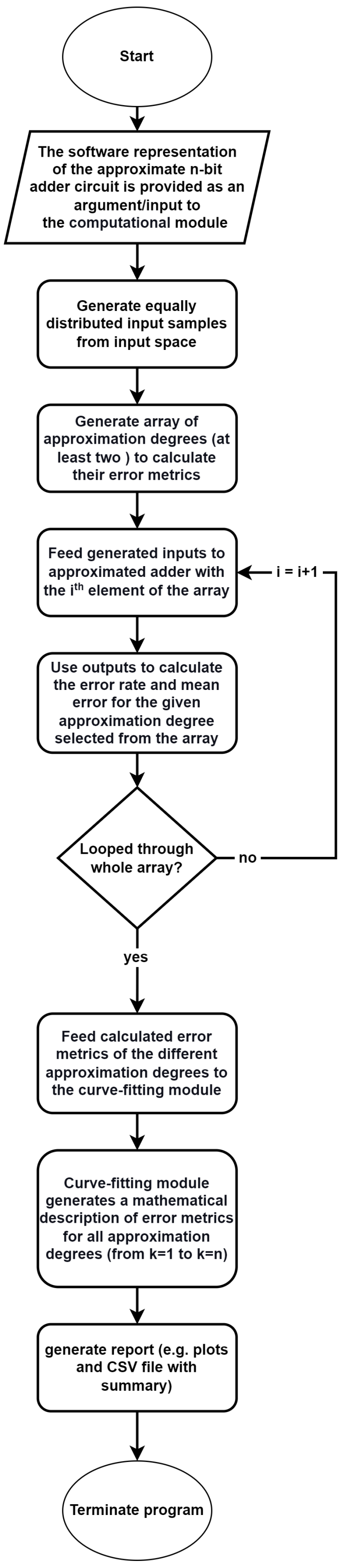

Start
The software representation of the approximate n-bit adder circuit is provided as an argument/input to the computational module
Generate equally distributed input samples from input space
Generate array of approximation degrees (at least two) to calculate their error metrics
Feed generated inputs to approximated adder with the i^th element of the array
i = i+1
Use outputs to calculate the error rate and mean error for the given approximation degree selected from the array
Looped through whole array?
no
yes
Feed calculated error metrics of the different approximation degrees to the curve-fitting module
Curve-fitting module generates a mathematical description of error metrics for all approximation degrees (from k=1 to k=n)
generate report (e.g. plots and CSV file with summary)
Terminate program

Figure 5: flow chart of software process

3.2 Hardware

The hardware was developed in Verilog and the functionality was verified by self-checking testbenches, which compare their results with the reference results from the golden model generated by the SoC_Lab/scripts/generate_testvectors.py script. For more details please refer to the documentation provided in the source code.

To obtain the error statistics the chosen adder architecture is implemented with varying degrees of approximation on the ZedBoard FPGA-Development board which then generates datapoints for the curve fitting module in the host software. On the FPGA itself the Adders interface with the onboard Central Processing Unit (CPU) via General Purpose Input/Output (GPIO) blocks, through which they receive input and sends output values.

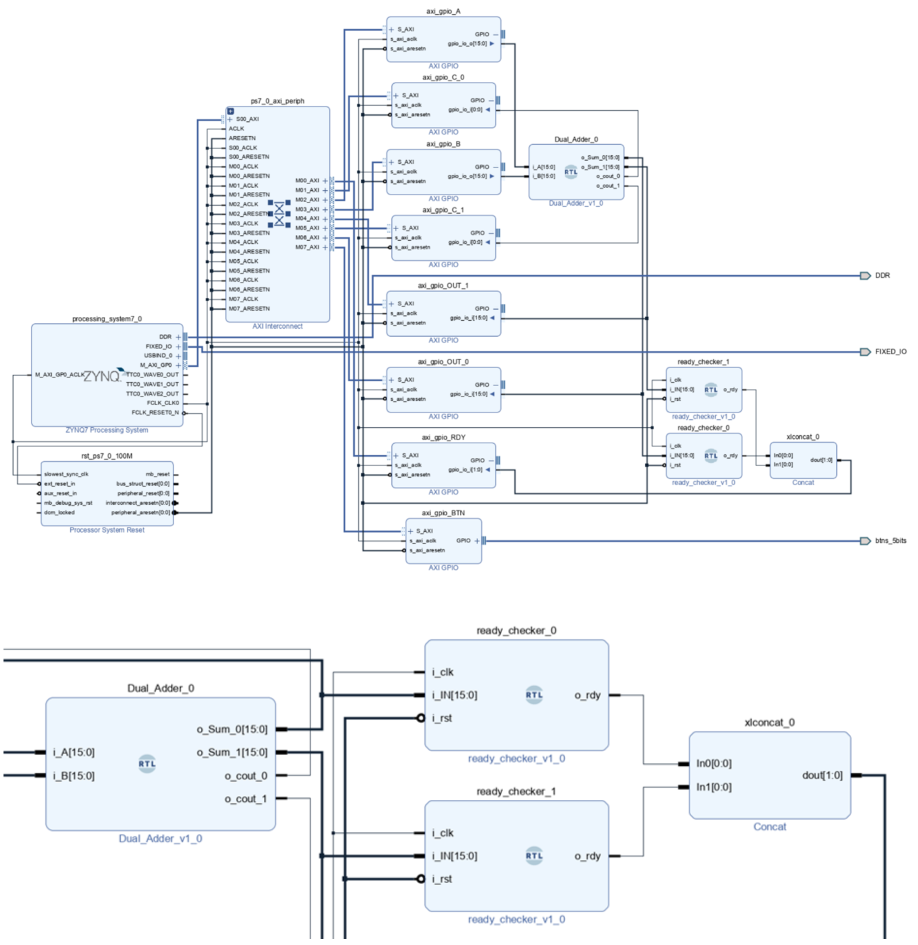
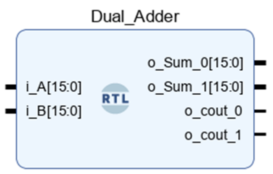


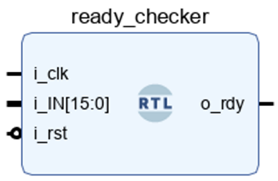
Figure 6: FPGA block design and closeup of custom modules

Dual_Adder



- **Parameters:** N, K_0, K_1
- **Inputs:** A, B
- **Outputs:** S_0, S_1, C_0, C_1
- **Description:** Module that implements 2 N-bit adders of the chosen design with different degrees of approximation K_0 and K_1. The N-bit inputs A and B are passed onto both adders with S and C as their corresponding output. This allows the user to easily change the implemented adder model by simply changing the chosen adder design.

Ready_checker



- **Parameters:** N
- **Inputs:** clk, IN, rst
- **Outputs:** rdy
- **Description:** The ready checker module is used to synchronize the adders and give a ready signal "0" to the CPU once the adders outputs are stable for N clock cycles.

3.3 Embedded Software

The C-program running on the ZedBoard CPU remains idle until one of the 5 buttons on the board is pressed. Once a press is registered random input values are generated and passed on to the Dual_Adder module. The CPU then waits for the ready signal to indicate the arrival of the according output values. The received values are compared against the exact result and error statistics to generate error rate and mean error are collected. This procedure is repeated 106 times. Once the data has been gathered the mean error is calculated and the results are passed onto the host software via serial communication.

```
// INIT & WAIT FOR BUTTONPRESS

while(count > 1){

        // READ DATA FROM GPIO

        if(rdy == 0){
                exact = a+b;
                a=rand()%(MAX+1);
                b=rand()%(MAX+1);
                rdy = 1;

                if(out_0 != exact){
                        err_0++;
                        AE0 += abs(exact - out_0);
                }
```

```
                    if(out_1 != exact){
                            err_1++;
                            AE1 += abs(exact - out_1);
                    }

            count--;
            }
}

// SEND DATA VIA SERIAL
```

3.4 Host Software

The host software has a Graphical User Interface (GUI) that lets you choose the according serial port and then waits for the data gathered by the FPGA. Once data is received, the curve fitting module uses the gathered datapoints to calculate error statistics for all degrees of approximation, i.e. 1-16 approximated bits for a 16-bit adder, for the given adder design. The calculated data is the presented as graphs and can be saved to a .csv file.

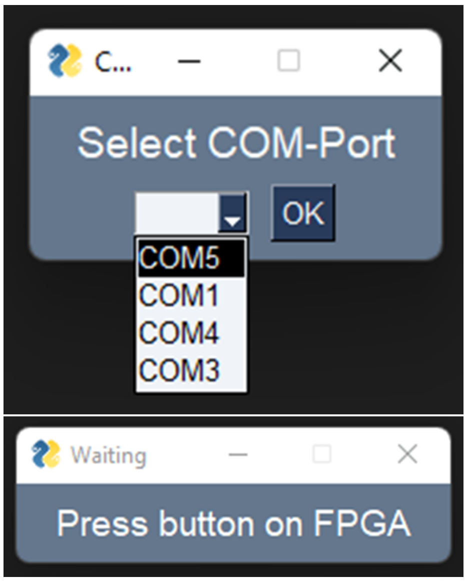


Figure 7: COM-Port Selection / Program waiting for data

4 Results and Conclusion

Equation 4.1 [1] shows how to calculate the mean error of an approximated adder.

$$MED = \frac{1}{n}\sum_{i=1}^{n} ED \tag{4.1}$$

Using the values calculated on the FPGA to fit a curve for the error rate and the mean error, the error statistics for all $k < n$ degrees of approximation are calculated, as seen in figure 8. The blue data points in the figure signal the data points used for the generation of the function, and the black points show the values, calculated from the generated function.

Tables 1 and 2 show the comparison between the calculated values and the simulated values. The estimated values fit the simulated values to a close degree.

Comparing the statistics to the results of a paper [1], as shown in in figure 10, the error rate coincides, however, the mean error differs substantially for the LOA, as we used the absolute error, whereas the paper likely uses the signed values and the error averages out.

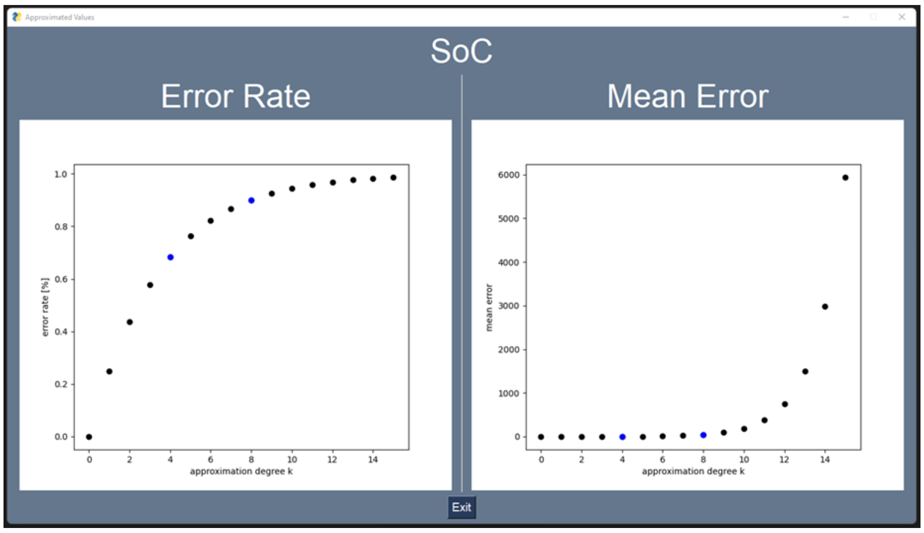


Figure 8: Presentation of results for LOA adder

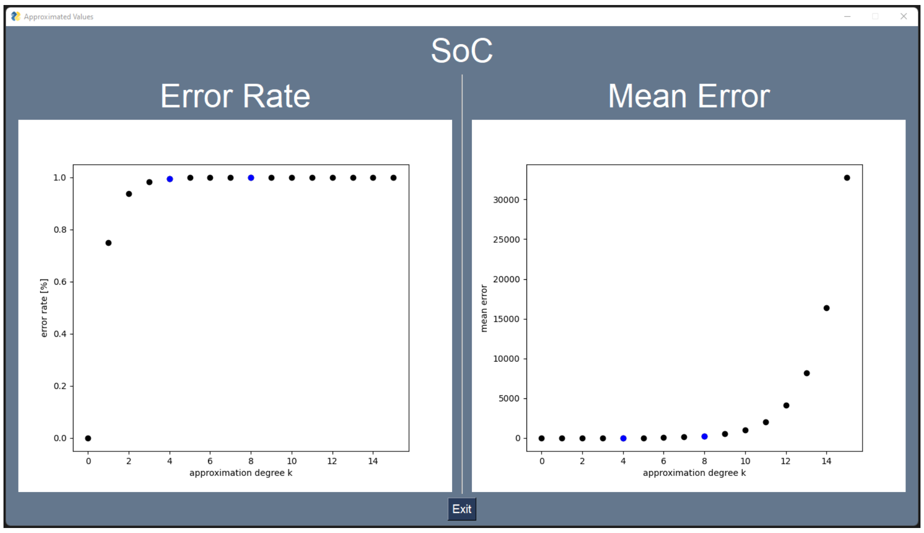


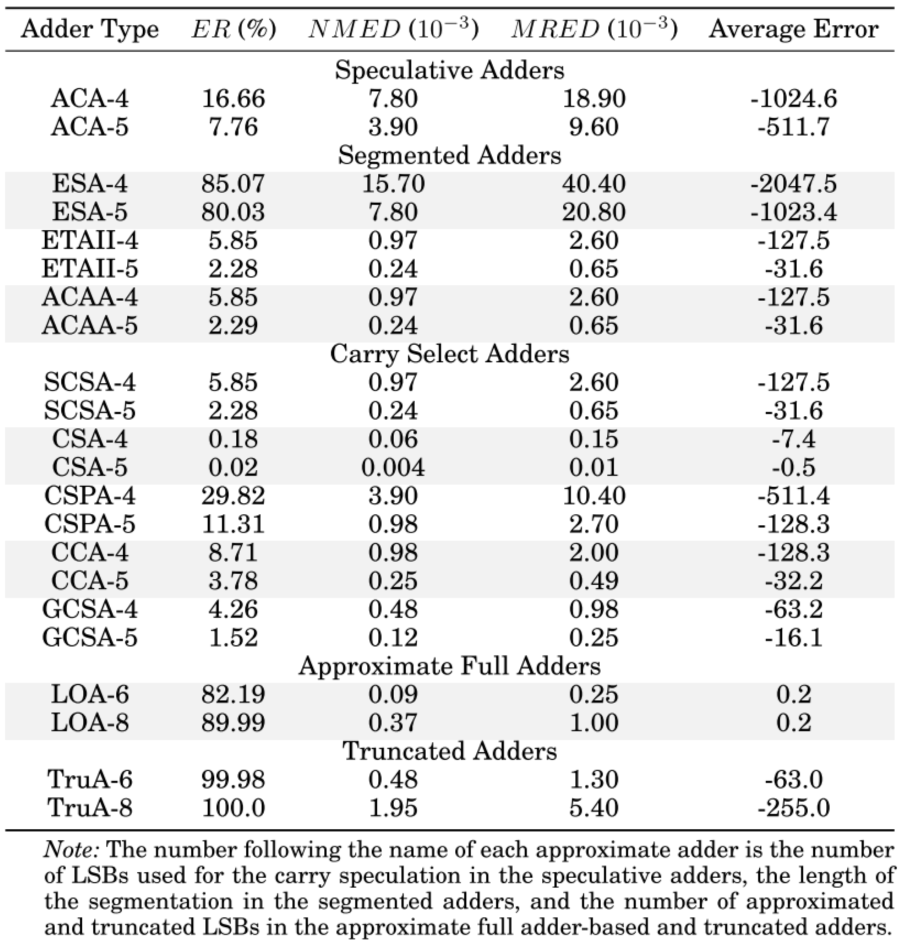
Figure 9: Presentation of results for truncated adder

approx. degree	approximated			simulated	
	error rate	mean error		error rate	mean error
0	0.0000	0.0000		0.0000	0.0000
1	0.2502	0.1942		0.2528	0.2528
2	0.4378	0.5804		0.4369	0.6245
3	0.5785	1.3485		0.5749	1.3641
4	0.6839	2.8764		0.6841	2.8739
5	0.7630	5.9153		0.7611	5.8855
6	0.8223	11.9597		0.8209	11.8387
7	0.8668	23.9819		0.8671	23.9125
8	0.9001	47.8940		0.9006	47.8775
9	0.9251	95.4550		0.9260	95.3677
10	0.9438	190.0535		0.9430	191.4998
11	0.9579	378.2093		0.9581	386.0042
12	0.9684	752.4498		0.9687	767.5191
13	0.9763	1496.8116		0.9753	1535.1259
14	0.9822	2977.3420		0.9826	3092.5267
15	0.9867	5922.1067		0.9867	6135.8519

Table 1: Error statistics for LOA-Adder

approx. degree	approximated			simulated	
	error rate	mean error		error rate	mean error
0	0.0000	0.0000		0.0000	0.0000
1	0.7501	1.0006		0.7504	1.0010
2	0.9376	3.0017		0.9380	2.9912
3	0.9844	7.0034		0.9848	6.9947
4	0.9961	15.0062		0.9964	15.0082
5	0.9990	31.0104		0.9991	31.0050
6	0.9998	63.0162		0.9999	63.0011
7	0.9999	127.0221		1.0000	126.7755
8	1.0000	255.0231		1.0000	255.4312
9	1.0000	511.0031		1.0000	512.2067
10	1.0000	1022.9192		1.0000	1023.2862
11	1.0000	2046.6637		1.0000	2043.6286
12	1.0000	4093.9772		1.0000	4092.6331
13	1.0000	8188.2532		1.0000	8165.8342
14	1.0000	16376.1034		1.0000	16388.4137
15	1.0000	32750.4004		1.0000	32759.5008

Table 2: Error statistics for truncated adder



Adder Type	ER (%)	$NMED$ (10^{-3})	$MRED$ (10^{-3})	Average Error
Speculative Adders				
ACA-4	16.66	7.80	18.90	-1024.6
ACA-5	7.76	3.90	9.60	-511.7
Segmented Adders				
ESA-4	85.07	15.70	40.40	-2047.5
ESA-5	80.03	7.80	20.80	-1023.4
ETAII-4	5.85	0.97	2.60	-127.5
ETAII-5	2.28	0.24	0.65	-31.6
ACAA-4	5.85	0.97	2.60	-127.5
ACAA-5	2.29	0.24	0.65	-31.6
Carry Select Adders				
SCSA-4	5.85	0.97	2.60	-127.5
SCSA-5	2.28	0.24	0.65	-31.6
CSA-4	0.18	0.06	0.15	-7.4
CSA-5	0.02	0.004	0.01	-0.5
CSPA-4	29.82	3.90	10.40	-511.4
CSPA-5	11.31	0.98	2.70	-128.3
CCA-4	8.71	0.98	2.00	-128.3
CCA-5	3.78	0.25	0.49	-32.2
GCSA-4	4.26	0.48	0.98	-63.2
GCSA-5	1.52	0.12	0.25	-16.1
Approximate Full Adders				
LOA-6	82.19	0.09	0.25	0.2
LOA-8	89.99	0.37	1.00	0.2
Truncated Adders				
TruA-6	99.98	0.48	1.30	-63.0
TruA-8	100.0	1.95	5.40	-255.0

Note: The number following the name of each approximate adder is the number of LSBs used for the carry speculation in the speculative adders, the length of the segmentation in the segmented adders, and the number of approximated and truncated LSBs in the approximate full adder-based and truncated adders.

Figure 10: Simulation Results of the Error Characteristics for the Approximate Adders

Acronyms

CPU	Central Processing Unit
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
LOA	Lower Or Adder
LSB	Least Significant Bit

References

[1] Honglan Jiang, Jie Han, and Fabrizio Lombardi. A comparative review and evaluation of approximate adders. pages 343–348, 05 2015.