AxPPA: Approximate Parallel Prefix Adders

Final Project-Paper Presentation

Department of Electronics and Communication Engineering
International Institute of Information Technology
Bangalore - 560100

Outline of presentation

- Introduction
- Different PPA architectures & their operations
- Verilog modelling
- Error Calculation
- FPGA Results
- ASIC Results
- Graphical Analysis
- Application: FIR Filter
- Conclusion
- Future Work
- References

Introduction

- Addition units are widely used in many computational kernels of several error-tolerant applications such as machine learning and signal, image, and video processing.
- Additions are essential building blocks for other math operations such as subtraction, comparison, multiplication, squaring, and division.
- The PPAs, in particular, are among the fastest adders because they optimize the parallelization of the carry generation (G) and propagation (P).
- To evaluate our proposal for approximate POs (AxPOs), we generate the following AxPPAs, consisting of a set of four PPAs: approximate Brent–Kung (AxPPA-BK), approximate Kogge–Stone (AxPPAKS), Ladner-Fischer (AxPPA-LF), and Sklansky (AxPPA-SK).

Flow of Project

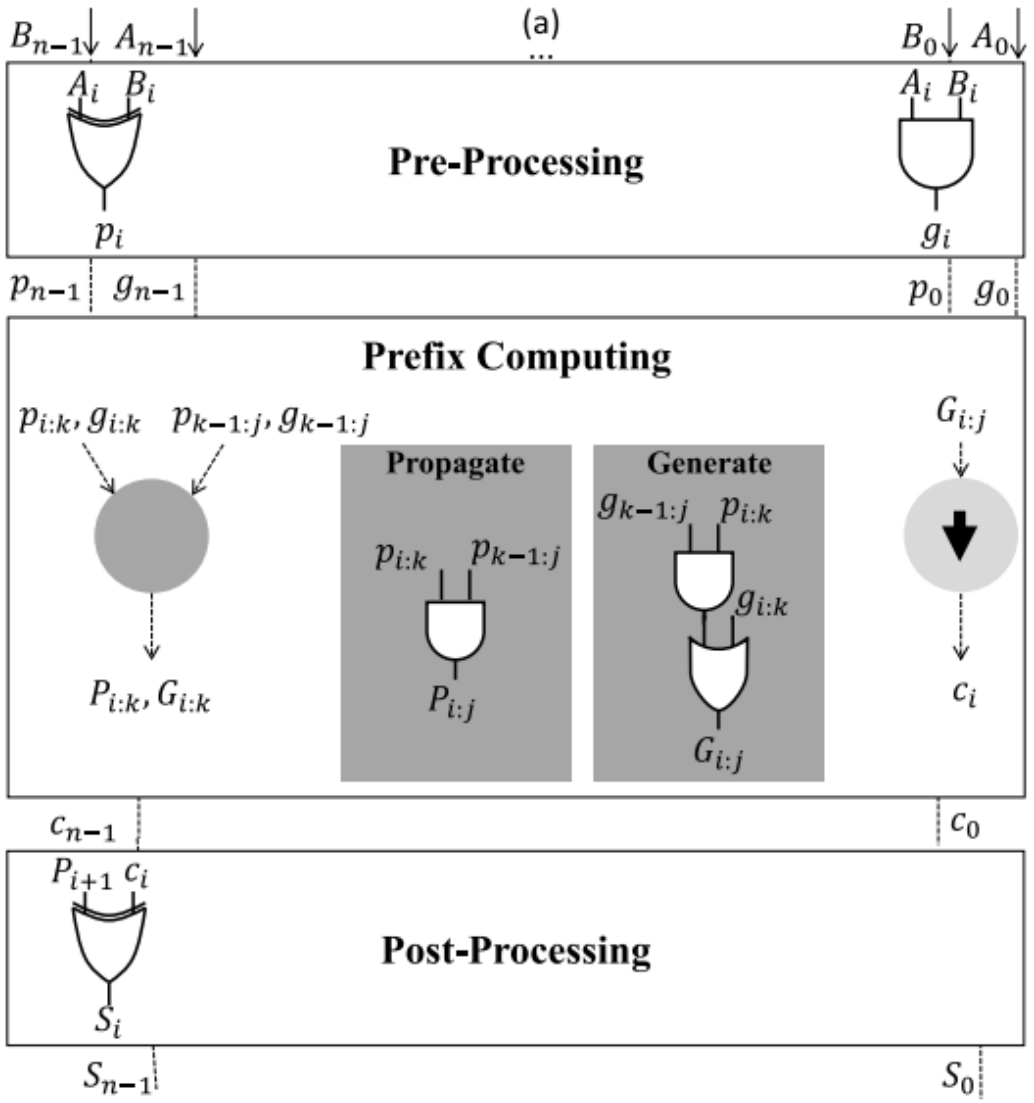
- Our proposal's main goal is to approximate the logic of prefix operators' carry propagation (P) and generation (G) phases.
- We develop and evaluate four AxPPAs using the architectures of Brent Kung (AxPPA-BK), Kogge Stone (AxPPA-KS), Ladner-Fischer (AxPPA-LF), and Sklansky (AxPPA-SK) in order to illustrate the proposal of approximate POs (AxPOs).
- We test the proposed AxPPAs in the following hardware accelerator, which includes many adders, as a complete case study in order to demonstrate their applicability: a finite impulse response (FIR) filter.

Parallel Prefix Adders

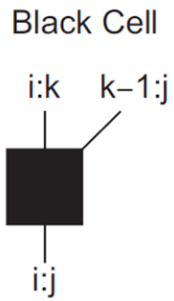
Parallel prefix adders (PPAs) are among the fastest yet most area-efficient addition units because they use logarithmic reduction of carry propagation paths to optimize the delay of the critical paths.

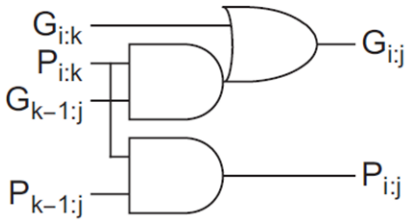
- Prefix: The outcome of the operation depends on the initial inputs.
- Parallel: Involves the execution of an operation in parallel. This is done by segmentation into smaller pieces that are computed in parallel.
- Operation: Any arbitrary primitive operator “ ° ” that is associative is parallelizable. It is fast because the processing is accomplished in a parallel fashion.

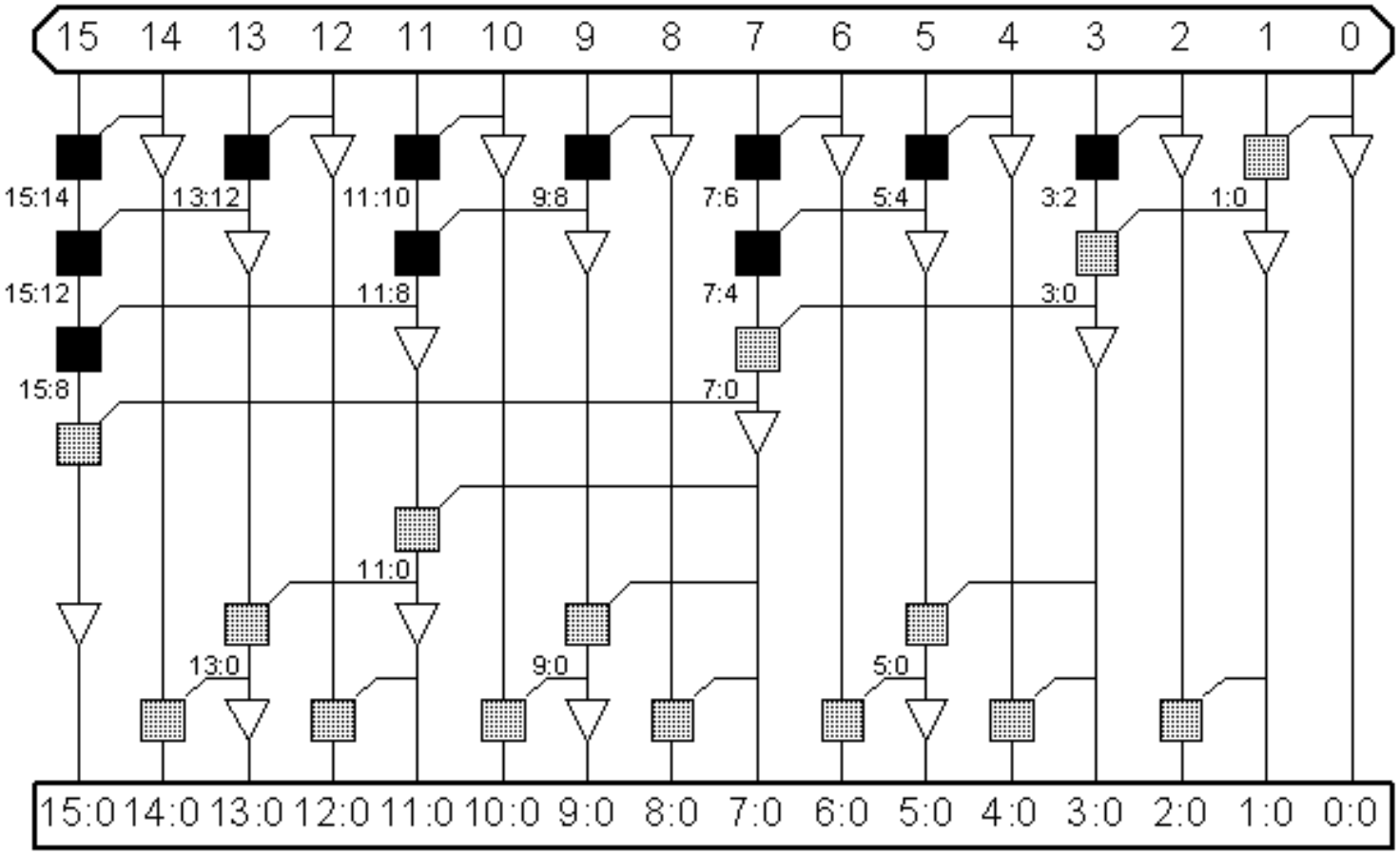
Architecture & Operation

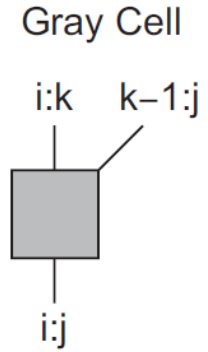


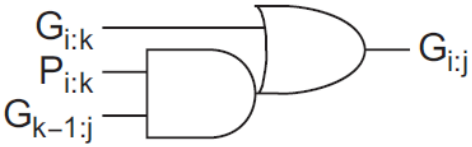
Brent-Kung Adder

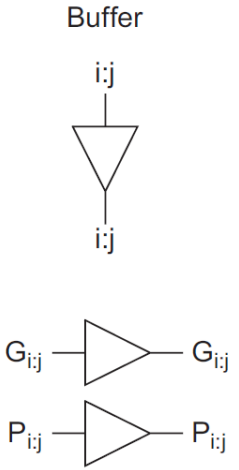










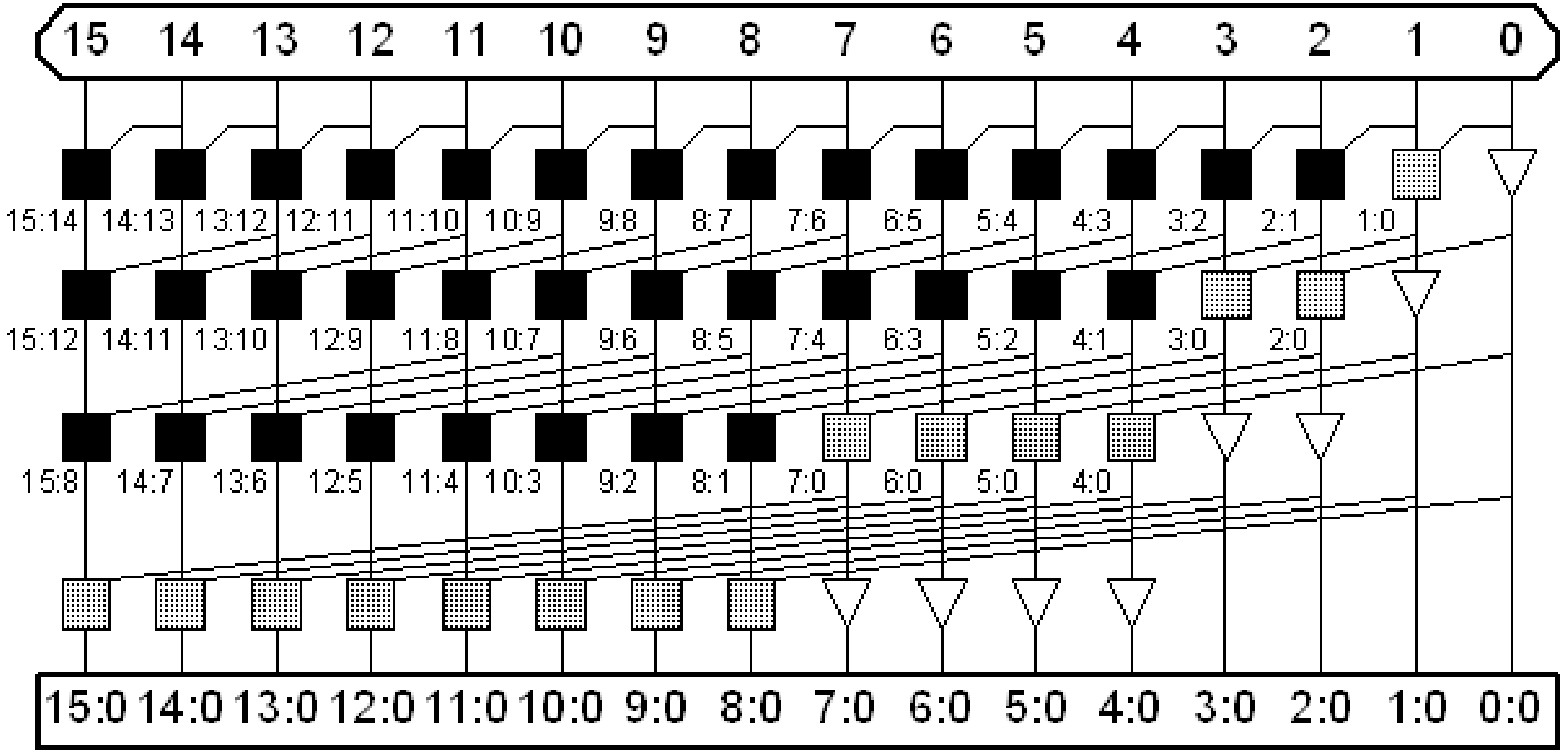


Brent-Kung Adder

The Brent–Kung strategy groups the prefix computation for 2-bit groups and uses it to find prefixes for 4-bit groups and uses it to find prefixes for 8-bit groups. It continues until the sum tree reaches the desired number of bits. Two cells per logical level limit the fan-out. The approach of this adder allows a regular layout to reduce design and implementation costs, one of the most important criteria for the design of VLSI.

The Brent-Kung adder is the extreme boundary case of:
1. Maximum logic depth in PP adders (implies longer calculation time).
2. Minimum number of nodes (implies minimum area)

Kogge-Stone Adder



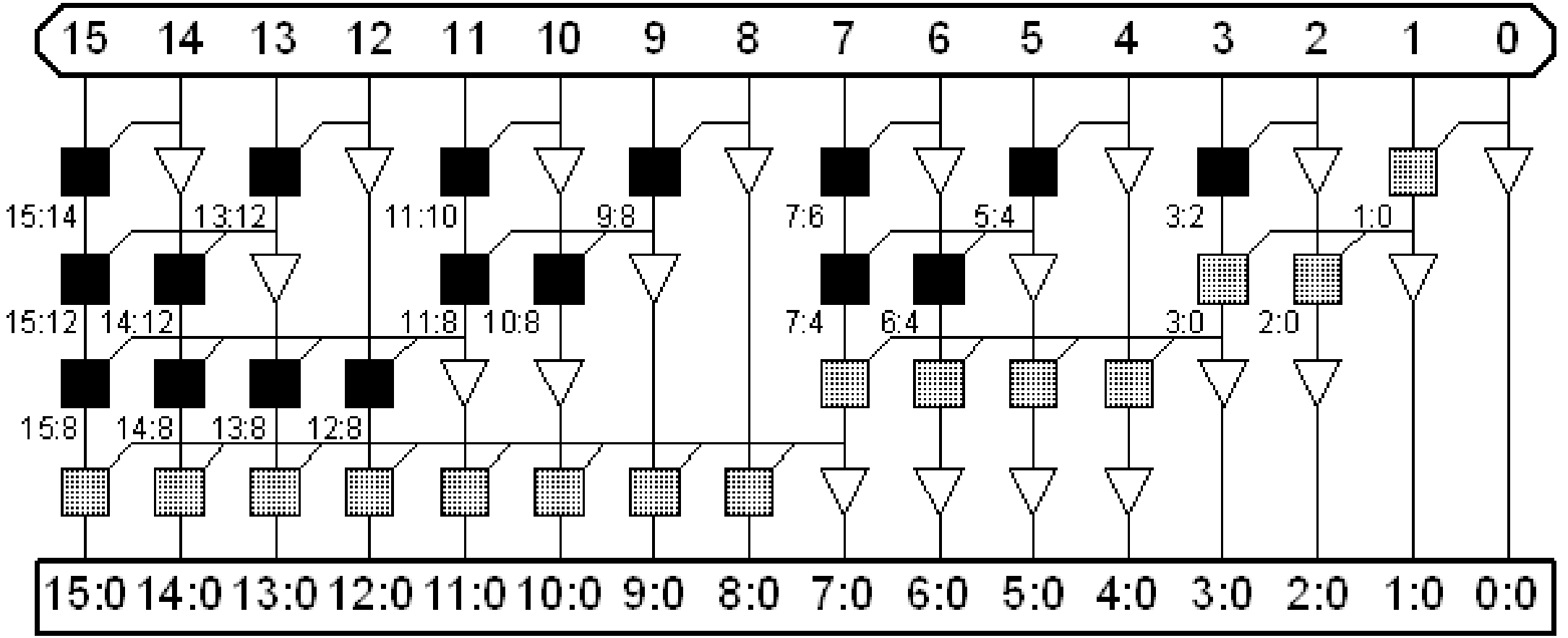
Kogge-Stone Adder

The Kogge–Stone leads to a combination of efficiency and fan-out reduction. The tree contains parallel propagation and generation cells. However, when arranging the adder layout in a regular grid, the circuit area increases due to the routing possibilities.

The Kogge-Stone adder has:

1. Low depth
2. High node count (implies more area).
3. Minimal fan-out of 1 at each node (implies faster performance).

Sklansky Adder



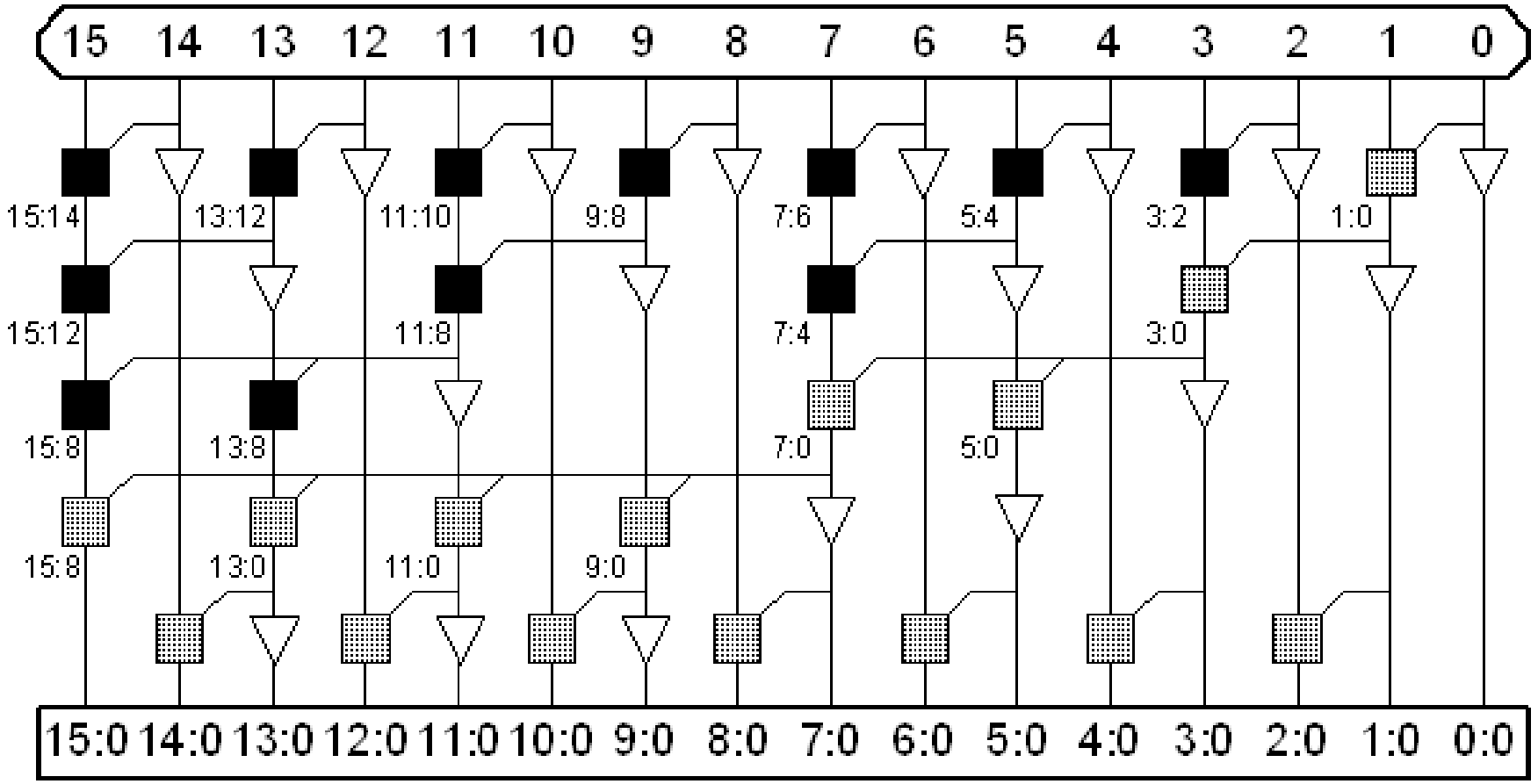
Sklansky Adder

The Sklansky adder operates by breaking down the addition of two n-bit binary numbers into a series of partial additions, each of which can be performed in parallel. The partial additions are arranged in a tree structure, with each node of the tree performing a carry lookahead operation on the partial sum and carry generated by its two child nodes.

The Sklansky adder has:

1. Minimal depth
2. High fan-out nodes

Ladner-Fischer Adder

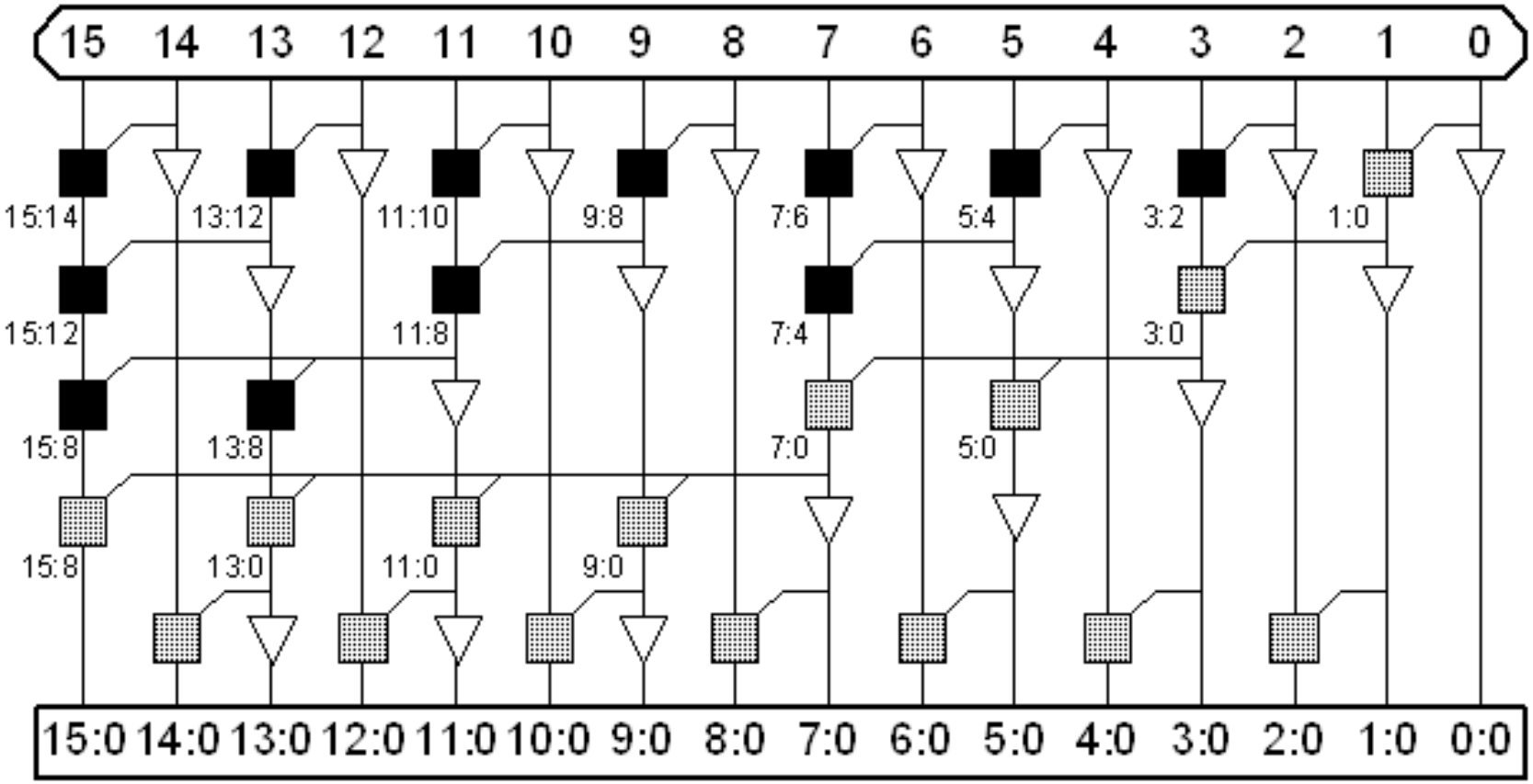


Ladner-Fischer Adder

The Ladner-Fischer adder operates by breaking down the addition of two n-bit binary numbers into a series of partial additions, each of which can be performed in parallel. Unlike the Sklansky adder, which uses a tree structure to perform carry lookahead, the Ladner-Fischer adder uses a linear chain of carry lookahead blocks. The Ladner-Fischer trees are a family of networks between Sklansky and Brent-Kung. It is similar to Sklansky, but computes prefixes for the odd numbered bits and again uses one more stage to ripple into the even positions.

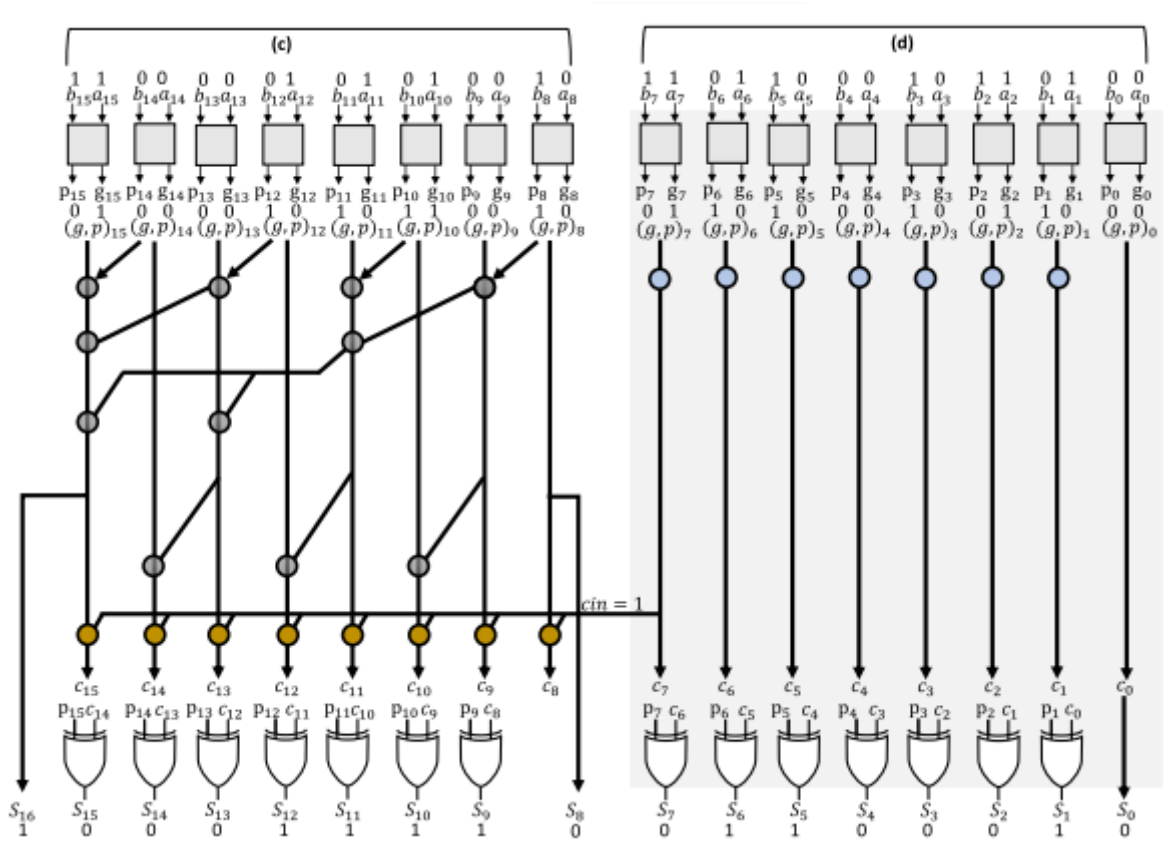
The Ladner-Fischer adder has:
1. Low depth
2. High fan-out nodes

Verilog Modelling



Exact Ladner-Fischer Adder

Verilog Modelling



Approximate Ladner-Fischer (K=8) Adder

Error Calculation / Analysis

1. Mean Absolute Error (MAE) :

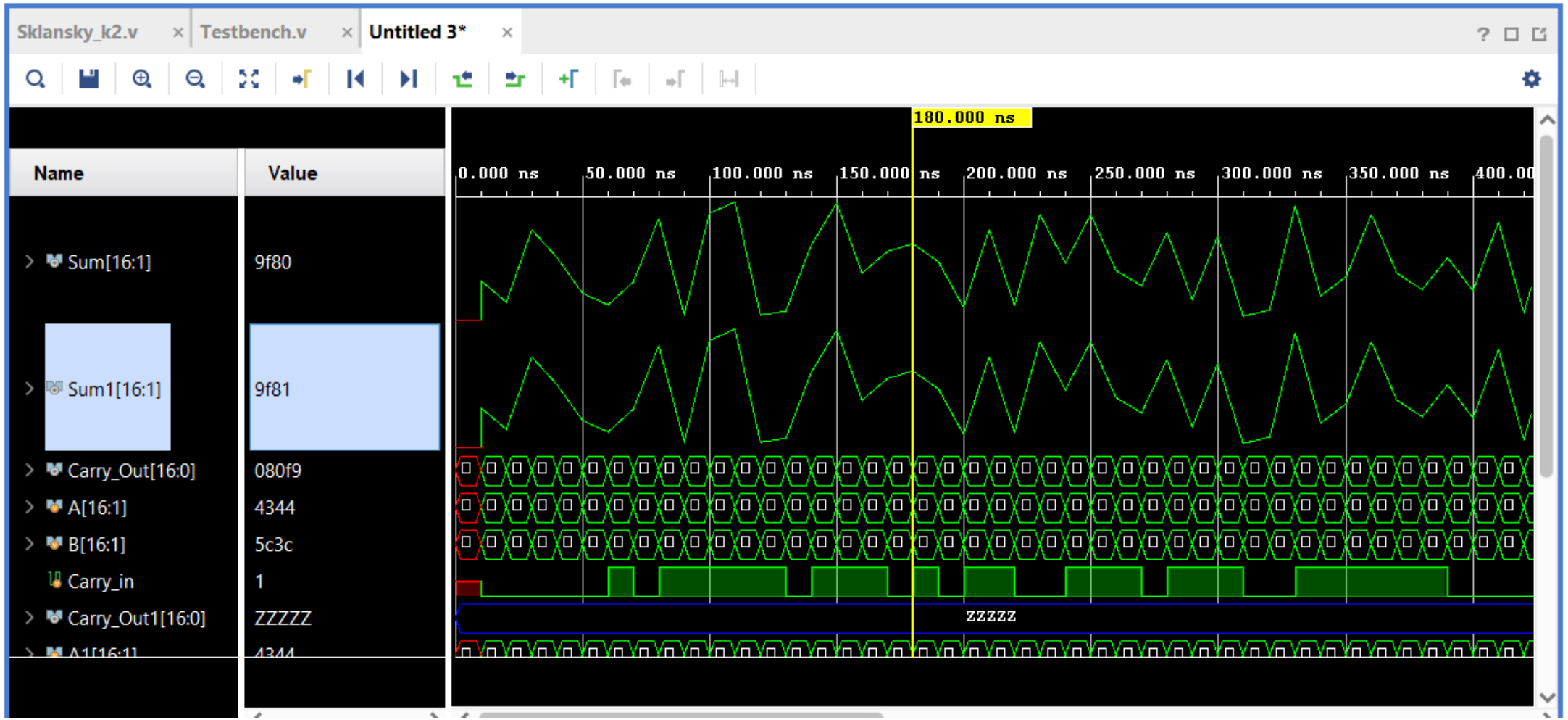
The MAE is the average of all absolute errors (n)

$$\text{MAE} = (1/n)\sum_{i=1}^{n} |E|$$

2. Mean Relative Error Distance (MRED).:

The MRED defines the error distance divided by the accurate sum.

Approximate(K2) comparison with Exact



FPGA Results

Adder	Total Delay (ns)	LUT	Total On-Chip Power (W)
BK Exact	8.183	51	16.40
BK K=2	7.692	39	16.84
BK K=4	7.466	36	16.32
BK K=6	7.623	34	14.86
BK K=8	6.866	26	15.27
KS Exact	8.340	42	18.06
KS K=2	8.358	40	16.84
KS K=4	7.470	36	16.29
KS K=6	7.436	27	15.97
KS K=8	6.876	27	15.37

Adder	Total Delay (ns)	LUT	Total On-Chip Power (W)
Sklansky Exact	8.309	55	17.67
Sklansky K=2	7.771	51	17.07
Sklansky K=4	7.314	44	16.32
Sklansky K=6	7.450	29	15.87
Sklansky K=8	7.449	27	15.25
LF Exact	8.110	47	17.96
LF K=2	7.913	38	16.83
LF K=4	7.470	36	16.29
LF K=6	7.436	27	15.97
LF K=8	7.063	25	15.21

ASIC Results

GENUS (45nm)

Brent Kung

	Exact	App K=2	App K=4	App K=6	App K=8
Area (um2)	106.704	83.79	82.422	82.072	81.054
Power (Watt)	2.41e-6	1.44E-06	1.40E-06	1.40e-6	1.55E-06

Sklansky

	Exact	App K=2	App K=4	App K=6	App K=8
Area (um2)	155.952	116.28	89.262	85.158	81.054
Power (Watt)	2.20E-06	1.61E-06	1.50E-06	1.20E-06	1.48E-06

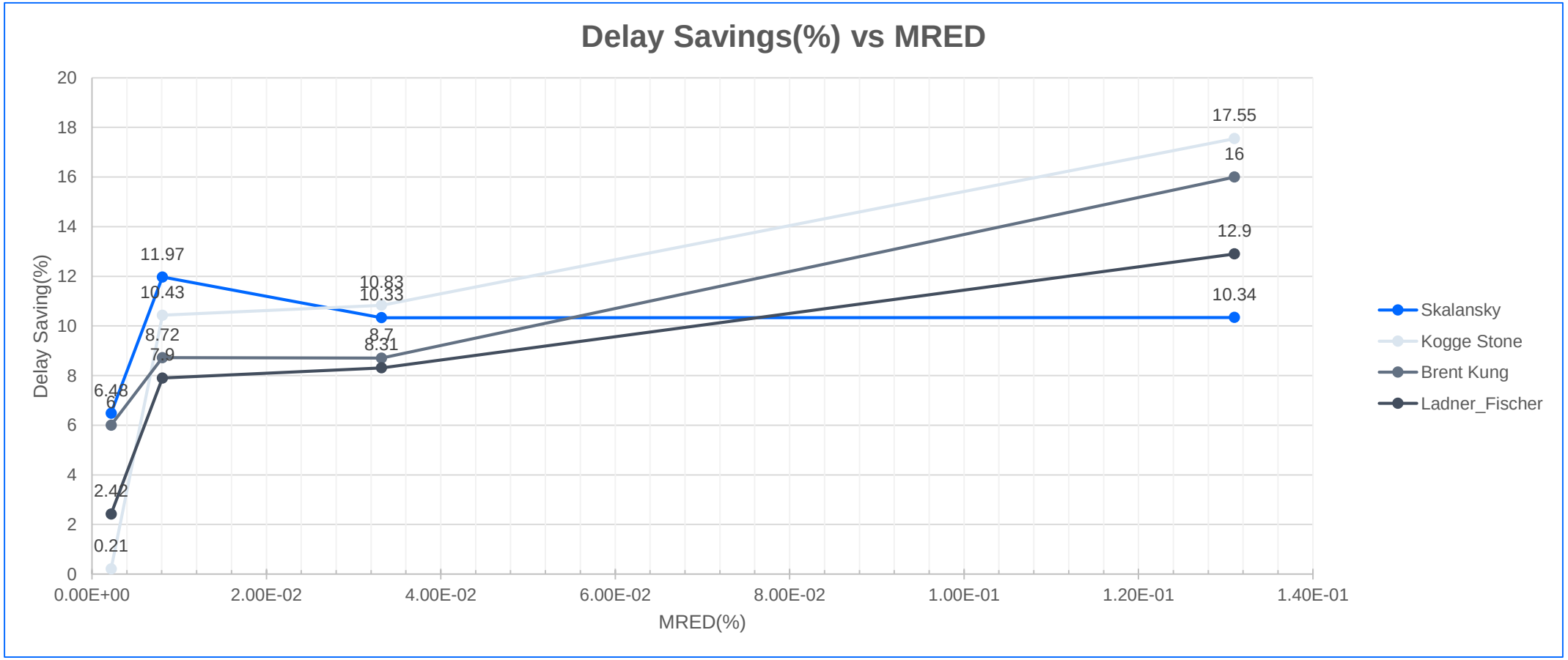
Kogge Stone

	Exact	App K=2	App K=4	App K=6	App K=8
Area (um2)	192.888	149.112	89.262	82.764	84.816
Power (Watt)	2.70E-06	2.20E-06	1.40E-06	1.22E-06	1.17E-06

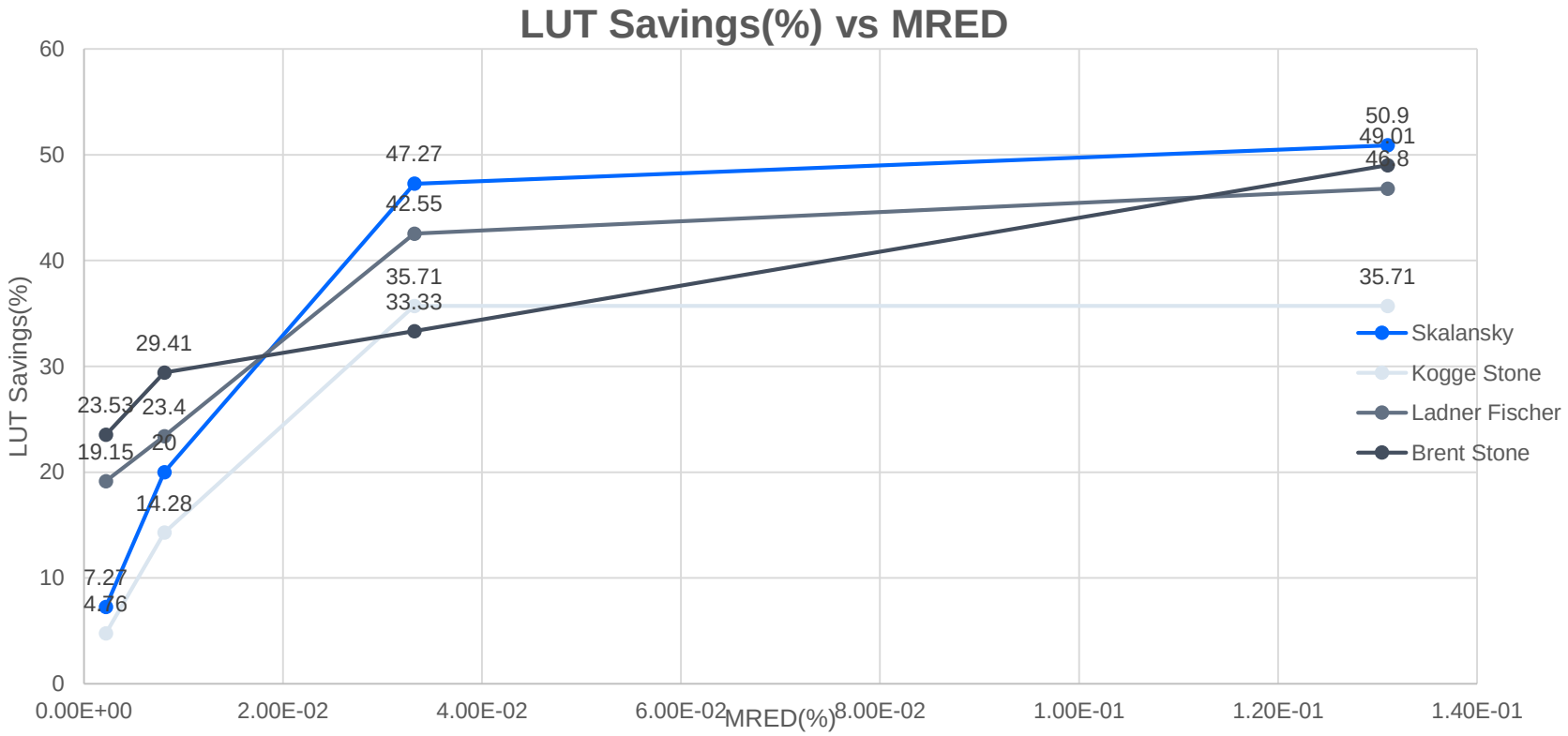
Ladner-Fischer

	Exact	App K=2	App K=4	App K=6	App K=8
Area (um2)	149.112	147.744	113.544	94.05	82.422
Power (Watt)	2.04E-06	2.04E-06	1.47E-06	1.39E-06	1.34E-06

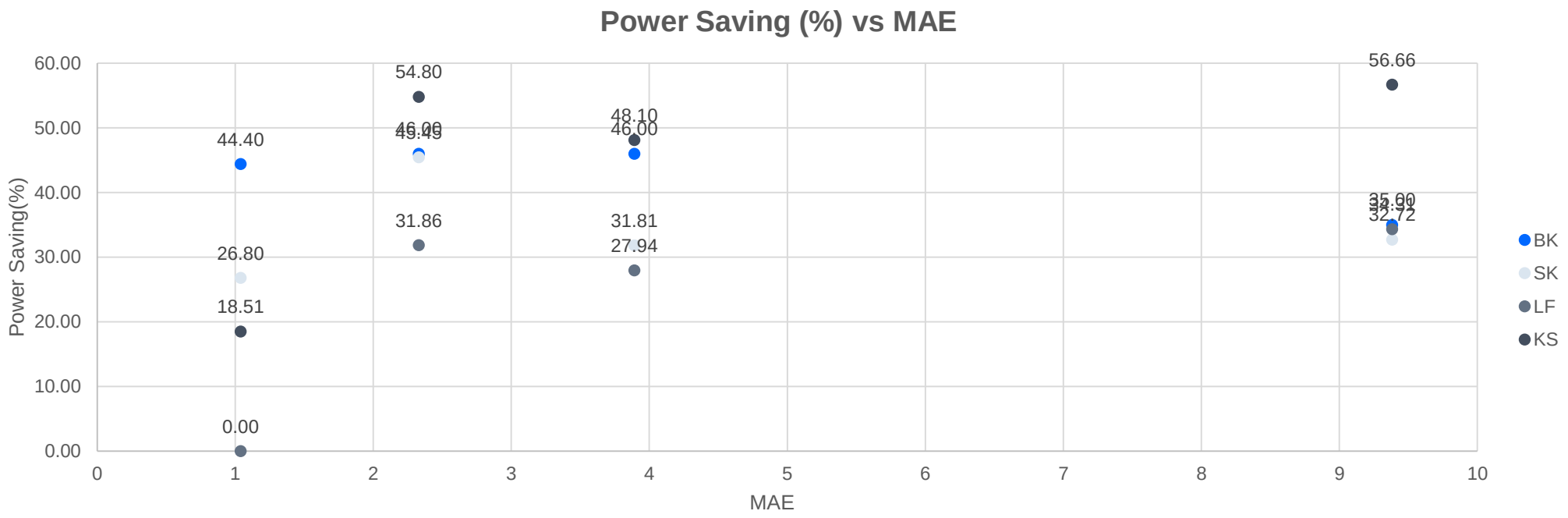
Graphical Analysis



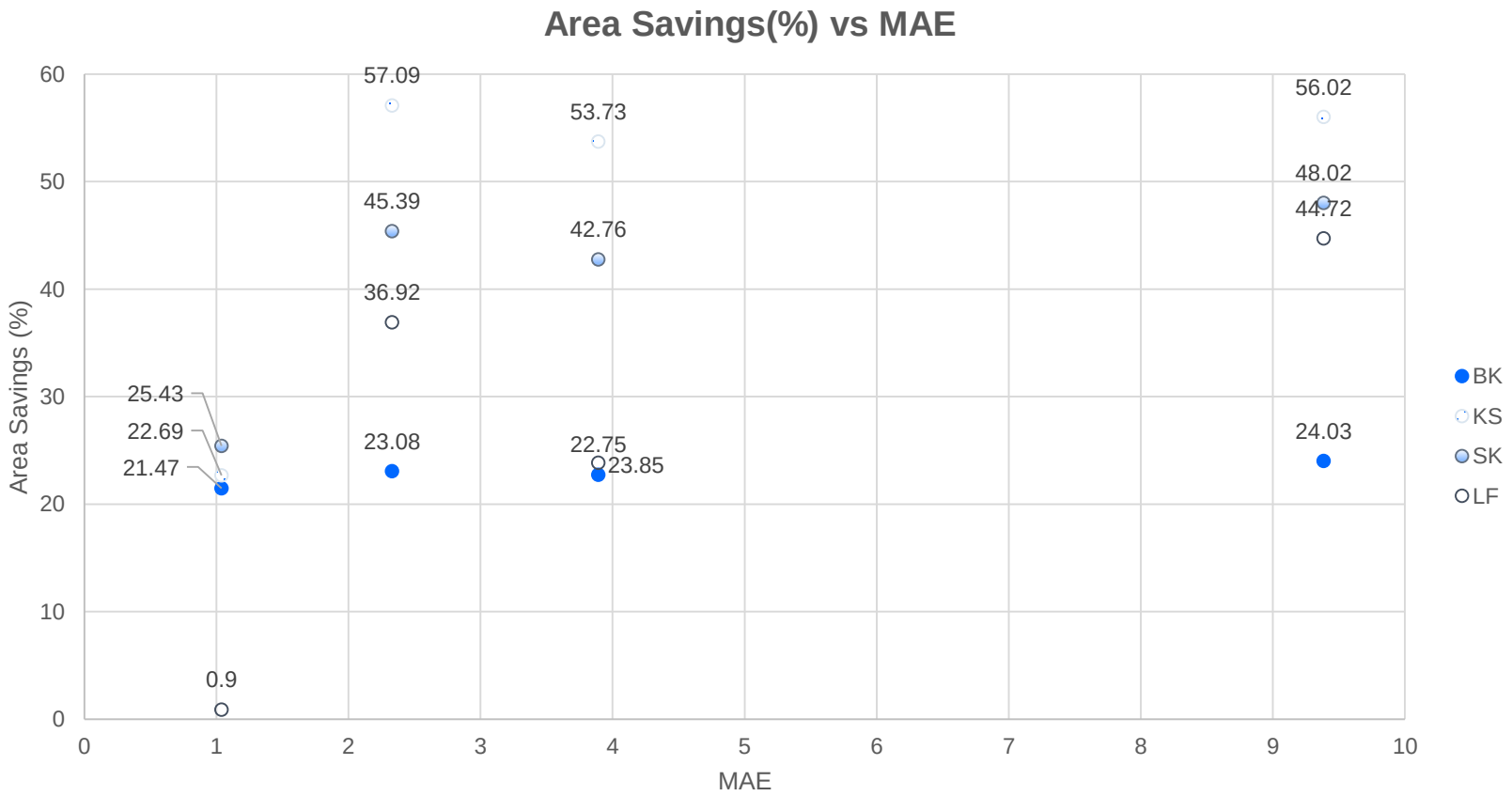
Graphical Analysis



Graphical Analysis



Graphical Analysis

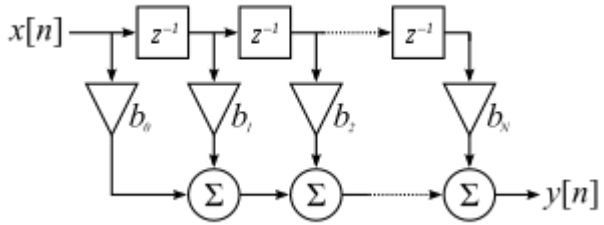


Application in FIR Filter

An FIR (Finite Impulse Response) filter is a type of digital filter used in signal processing. The impulse response of an FIR filter is of finite duration because it settles to zero in a finite amount of time.

For a causal discrete-time FIR filter of order *N*, each value of the output sequence is a weighted sum of the most recent input values:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N]$$
$$= \sum_{i=0}^{N} b_i \cdot x[n-i],$$



Application in FIR Filter

In FIR filter inputs are multiplied with coefficient and constant multiplication are realized in parallel using +/-/>> operations.

So we implemented the verilog code for the FIR filter in FPGA flow and used our proposed approximate PPAs for addition in place of regular adders. And we got improvements in logic speed, power saving & resource utilization with small percentage drawback of accuracy.

FPGA Results

Adder	Total Delay (ns)	LUT	Total On-Chip Power (W)
Sklansky Exact	12.84	93	18.85
Sklansky K=2	12.50	92	18.38
Sklansky K=4	11.70	65	17.44
Sklansky K=6	10.37	67	17.17
Sklansky K=8	10.06	59	16.95
BK Exact	12.111	101	18.37
BK K=2	12.137	87	17.74
BK K=4	11.265	75	17.74
BK K=6	10.685	55	16.76
BK K=8	9.607	59	16.64

Adder	Total Delay (ns)	LUT	Total On-Chip Power (W)
LF Exact	12.58	102	18.83
LF K=2	12.14	87	17.74
LF K=4	11.27	75	17.74
LF K=6	10.69	68	16.84
LF K=8	9.54	55	16.76
KS Exact	12.726	178	21.53
KS K=2	12.244	115	18.94
KS K=4	11.836	106	18.26
KS K=6	10.593	74	17.34
KS K=8	10.008	62	16.83

Conclusion

- This paper proposed several AxPPA architectures. The developed approximation technique computes the POs of the prefix computation step.
- In particular, AxPPA-LF offers the best trade-off between approximation and quality, resulting in significant energy savings for both the application independent (MAE, MRED) and the specific case studies (FIR).

Future Work

- We are planning to implement the most optimum architectures in Machine Learning algorithms like Image Detection, Image Classification etc.

References

- R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” IEEE Trans. Comput., vol. C-31, no. 3, pp. 260–264, Mar. 1982.
- P. M. Kogge and H. S. Stone, “A parallel algorithm for the efficient solution of a general class of recurrence equations,” IEEE Trans. Comput., vol. C-22, no. 8, pp. 786–793, Aug. 1973.
- J. Sklansky, “Conditional-sum addition logic,” IEEE Trans. Electron. Comput., vol. EC-9, no. 2, pp. 226–231, Jun. 1960.
- R. Ladner and M. Fischer, “Parallel prefix computation,” J. ACM, vol. 27, pp. 831–838, Oct. 1980.
- B. Alhazmi and F. Gebali, “Fast large integer modular addition in GF(p) using novel attribute-based representation,” IEEE Access, vol. 7, pp. 58704–58719, 2019.

Thank you

Presented By:

Archan Desai MT2022502

Ishan Desai MT2022506

Nishit Chechani MT2022511

Yash Kothari MT2022525