# Project Course

CS-399

Supervised By: Prof. Joycee Mekie


## POSITs and Approximate Computing for Neural Networks

Jinay Dagli and Neel Shah

# What is POSIT Number System?

The POSIT number system is composed of a run-time varying exponent component, which is defined by a composition of varying length “regime-bit” and “exponent-bit” (with a maximum size of ES bits, the exponent size). This also makes the fraction part to vary at run-time in size and position.

# Why POSIT?

The IEEE 754 standard has been used in various applications over the years. The POSIT system addresses the flaws or shortcomings of the IEEE 754, and tries to overcome them.

# Limitations of floating-point representation (IEEE 754)

- There is no guarantee of identical results across different computer systems.
- It has the use of NaN(not a number) values, which is not very efficient.
- Invisible Rounding Errors: Individual operands are rounded off in calculations, which sometimes leads to the breaking of arithmetic laws (associativity and distributivity).
- Accuracy is good only across a certain range.
- IEEE 754 makes use of overflow and underflow.

Very large numbers -> rounded off to infinity

Very small numbers -> rounded off to zero

# Advantages of POSIT

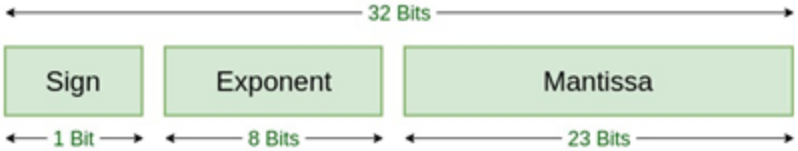
- It doesn't overflow or underflow.
- There is no NaN(Not a Number) case in posits.
- Provides more accurate answers as compared to IEEE floating point numbers with smaller number of bits.
- Certain properties of posits suggests that it may be possible to perform deep learning tasks using posits.
- Deep learning algorithms have limited memory bandwidth, so posits help in cut down of memory bandwidth as compared to IEEE floating point.

# Inefficiencies of POSIT

The floating-point format has a fixed bit format for exponent and fraction (or significand). Due to this reason, for extracting the values of exponent and fraction, we can decode it parallely. But since the bits for exponent and fraction are not fixed in a posit representation, we can only decode it serially.

## IEEE 754 Standard

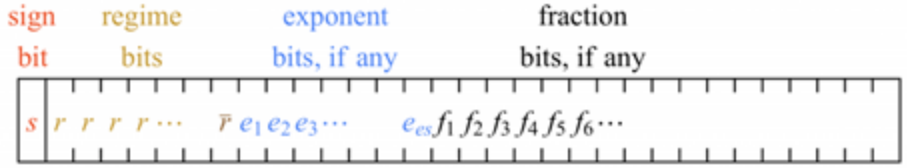
**Single Precision:**




**Double Precision:**


## Posit representation

**Posit<n,es> representation:**




n represents total number of bits.
Es is the size of exponent.

## IEEE 754 Standard

**Single Precision:**

Representation: (-1)^s x (2^(e-127)) x 1.m

Where s: Sign bit ; e:Exponent and m:Mantissa

**Double Precision:**

Representation: (-1)^s x (2^(e-1023)) x 1.m

where s: Sign bit ; e:Exponent and m:Mantissa

## Posit representation

**Posit<n,es> representation:**

Regime consists of a series of all 0s or all 1s.
If a bit is represented as r, then regime ends when the bit becomes r'.
Let m be this number of identical bits.

If leading 0s: let k = -m
If leading 1s: let k = m-1
Let a new term useed be 2^(2^(es)).

Representation: (-1)^s x (useed^k) x 2^e x (1+f)

where s:Sign bit ; e:Exponent ; f:Fraction

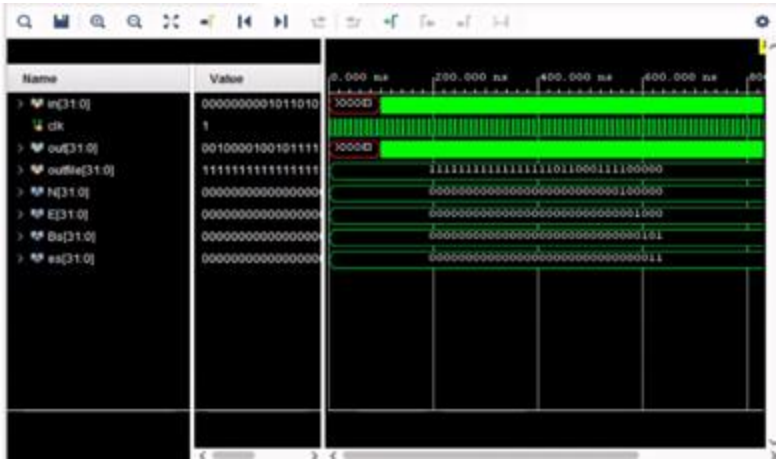
# 1. Posit-to-FP

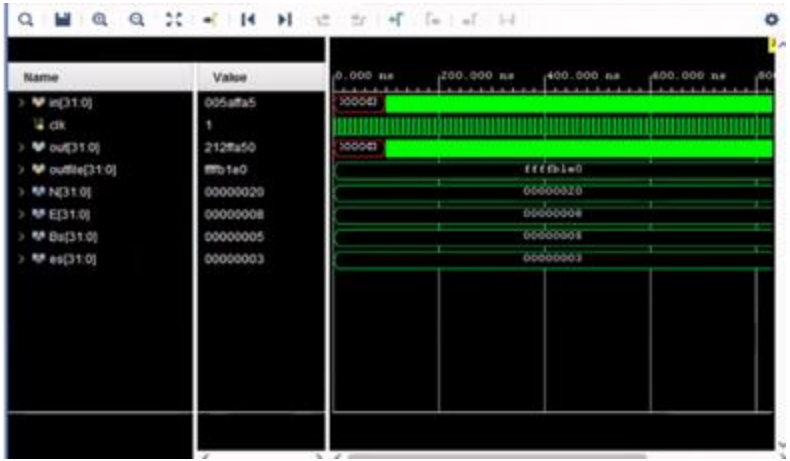
The posit to floating-point convertor takes a 32-bit number in posit representation as input and gives the number in floating point representation as the output.

```
Parameters: N = size of the representation
            E = Exponent in FP representation
            es = Exponent size in the posit representation
            in = 32-bit input FP input
            out = 32-bit output in posit representation
```

Code reference: Posit-HDL-Arithmetic/Posit_to_FP.v at master · manish-kj/Posit-HDL-Arithmetic · GitHub

# Simulation Results:



# Implementation Results:

## *Utilization*

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| **LuT** | 150 | 133800 | 0.11 |
| **IO** | 32 | 400 | 8 |

## *Power*

| | |
|---|---|
| **Total On-Chip Power** | 9.27 W |
| **Junction Temperature** | 42.3℃ |
| **Thermal Margin ∗** | 42.7℃ (22.5 W) |


***Thermal Margin:*** Thermal margin indicates how far the current operating temperature is below the maximum operating temperature of the processor.

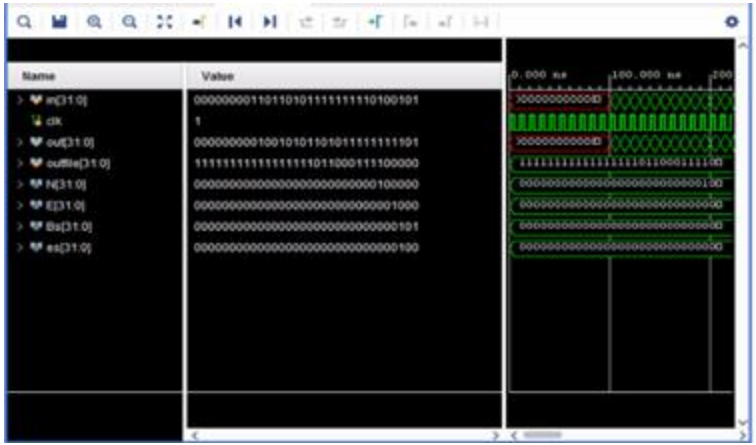
# 2. FP-to-Posit

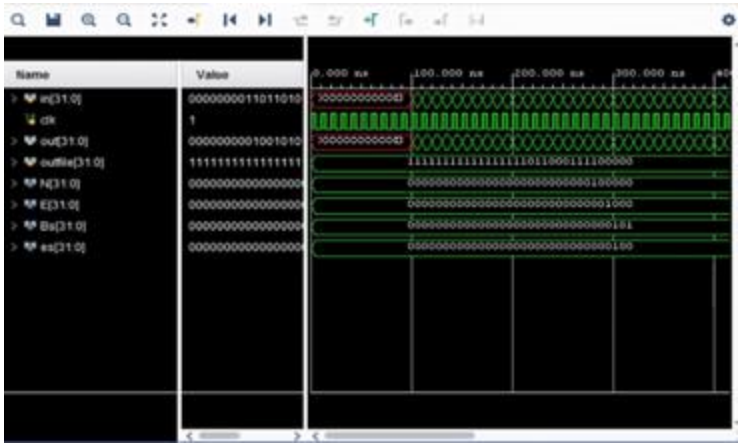
The floating point to posit convertor takes a 32-bit number in floating point representation as input and gives the number in posit representation as the output.

```
Parameters: N = size of the representation
            E = Exponent in FP representation
            es = Exponent size in the posit representation
            in = 32-bit input in posit representation
            out = 32-bit FP output
```

Code reference: Posit-HDL-Arithmetic/Floating-Point_to_Posit_Convertor at master · manish-kj/Posit-HDL-Arithmetic · GitHub

# Simulation Results:



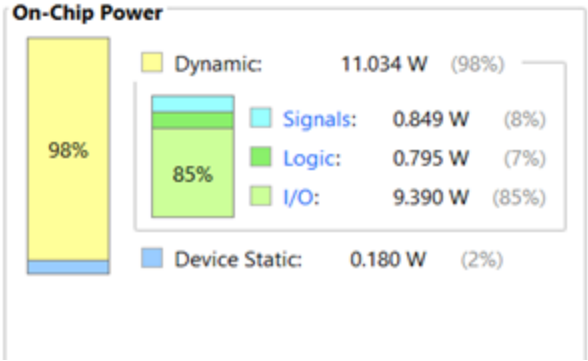
# Implementation Results:

## *Utilization*

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| **LuT** | 107 | 133800 | 0.08 |
| **IO** | 32 | 400 | 8 |

## *Power*

| | |
|---|---|
| **Total On-Chip Power** | 11.214 W |
| **Junction Temperature** | 46°C |
| **Thermal Margin *** | 39°C (20.6 W) |




***Thermal Margin:*** Thermal margin indicates how far the current operating temperature is below the maximum operating temperature of the processor.

# Some terms used henceforth (for reference)

**Neural Network**: A neural network is analogous to the working of the human neural system. The neural network is made up of 'neurons', which takes some input, processes it using some mathematical operations (eg :convolution), and produces the output. The neurons are connected to form the neural network.

**LeNet**: LeNet is a convolutional neural network (CNN) structure which was used in detecting handwritten cheques by banks based on MNIST dataset. The model has 5 layers with learnable parameters and hence it is named LeNet-5.

**ResNet:** It means residual network and it is also a convolutional neural network (CNN) architecture which helped to overcome the "vanishing gradient" problem, making possible the construction of networks with up to thousands of convolutional layers which performs better than shallow networks.

# Some terms used henceforth (for reference)

**MNIST**: The MNIST database is a large database of handwritten digits that is commonly used for training and testing various image processing systems in the field of machine learning. It contains 60,000 small square 28x28 pixels grayscale images of handwritten single digits between 0 to 9.

**Cifar-10**: The CIFAR-10 dataset is a collection of images that are commonly used to train machine learning and computer vision algorithms. It contains 60,000 32x32 pixels color images in 10 different classes. It is one of the most widely used datasets.

**ImageNet:** ImageNet is a database of annotated photographs which can be used for research purposes in computer vision. It can contain around 20000 classes. Convolutional neural networks find patterns at the level of pixels (transfer learning).

**Network: LeNet5  Dataset: MNIST  Base Accuracy: 98.49%**

| Fixed Posit Parameters (N, es, regime) | Accuracy (with Posit) (%) |
|---|---|
| 6, 2, 2 | 98.14 |
| 6, 3, 2 | 97.42 |
| 8, 2, 2 | 98.62 |
| 8, 3, 2 | 98.44 |
| 8, 3, 3 | 98.14 |
| 9, 3, 2 | 98.62 |
| 10, 2, 2 | 98.53 |
| 10, 3, 2 | 98.54 |
| 10, 6, 2 | 98.14 |

**Network: ResNet18 Dataset: CIFAR10 Base Accuracy:82.21%**

| Fixed Posit Parameters (N, es, regime) | Accuracy (with Posit) (%) |
|---|---|
| 6, 2, 2 | 60.11 |
| 8, 3, 2 | 76.37 |
| 8, 4, 2 | 60.11 |
| 9, 4, 2 | 76.37 |
| 9, 3, 2 | 80.73 |
| 10, 4, 2 | 80.73 |
| 10, 3, 2 | 81.82 |
| 10, 6, 2 | 60.11 |

# Some Observations from the data:

1. For the same length of fraction, that is, for the same value of (N - es - regime_length), accuracy remains almost the same.

   **Why?** - A Posit Multiplier just multiplies the fractions of the two operands, and just adds the exponent bits!

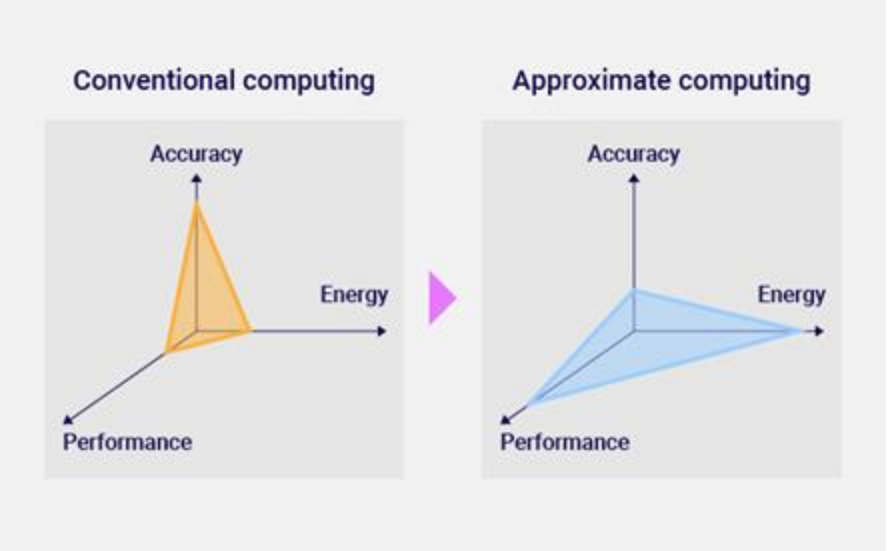
1. Generally, with the increase in the length of fraction bits, accuracy increases as well!

# Approximate Computing for Neural Networks Using 4-2 approximate compressors designed for integer multiplication

# What does Approximate Computing mean?

Approximate Computing has become a well-known computing technique in recent times.

Some approximation would not change the final output, but might lead to large gains in energy, circuit area, and performance.



## Why Approximate Computing?

Approximate Computing has the following pros when compared to the conventional/exact computing:

- Error is an essential part of any computing process, and approximate computing takes benefit of this error generated in computing process.
- Approximate computing has an important application in image processing.

## Why Approximate Computing?

Approximate Computing has the following pros when compared to the conventional/exact computing:

- AC (Approximate Computing) exploits the feature that many systems and applications can tolerate some loss of accuracy in the computation result.
- Computing becomes increasingly heavy with multimedia processing, machine learning, data mining, recognition, etc.

# State of the Art

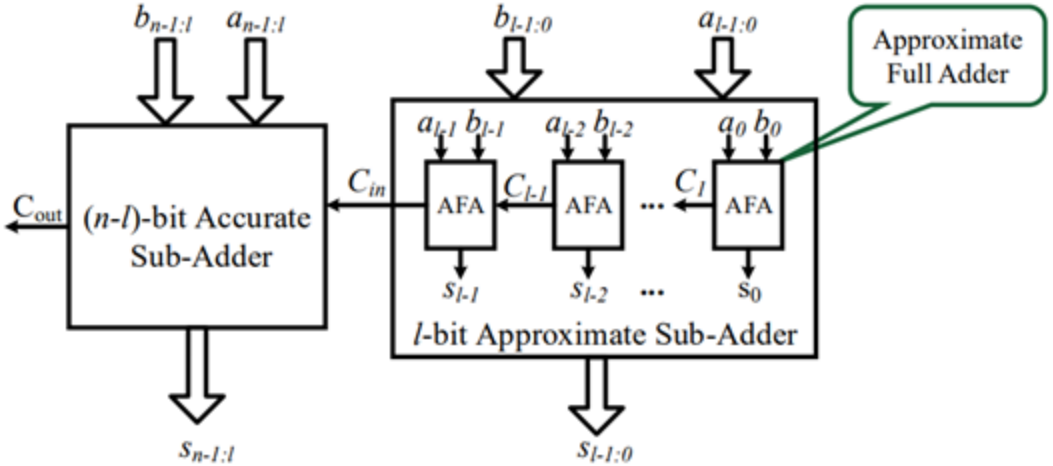
**(Transition from Posits to Approximate Computing)**

Since fixed-posit designs might lead to a greater hardware utilization, the focus of the second part of the project was to incorporate approximation techniques, first for integer multiplication, in order to reduce the hardware utilization and the area utilization in neural networks, without affecting the accuracy by a large amount!

Approximate fixed-posit multipliers is what we finally want to achieve!

# Approximate Full Adders

## A general schematic:




The $n$-bit approximate adder using approximate full adders

- The approximate full adder has a moderate MRED.
- The approximate full adder is slow, but it consumes a low power and area.

1. MRED (mean relative error distance) is used to evaluate the mean relative difference between an approximate result and the accurate result.

# Compressor Designs

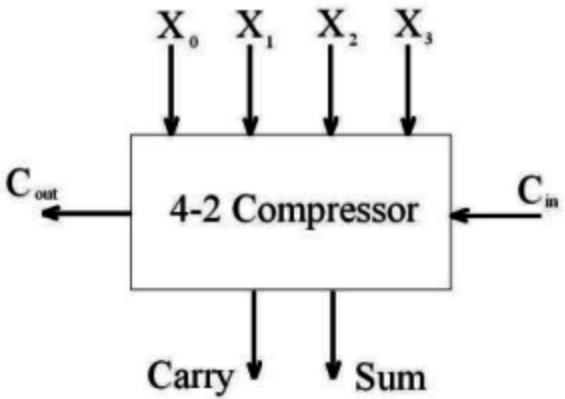



*Figure: Exact 4:2 Compressor*

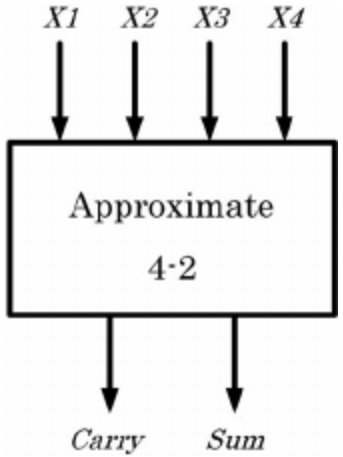



*Figure: Approximate 4:2 Compressor*

# Examples of some compressor Designs

There are several types of proposed compressor designs, out of which we will be focusing on the following 7 designs:

- Yang2
- Lin
- Strollo1
- Strollo2
- Momeni
- Venka
- Sabetz

# Yang-2

- The high-accuracy yang compressors proposed in [1] only introduce at most four errors.
- “Yang-2” introduces two errors as shown in the truth table below.
- The circuit design for “Yang-2” compressor is shown in figure on the right side.

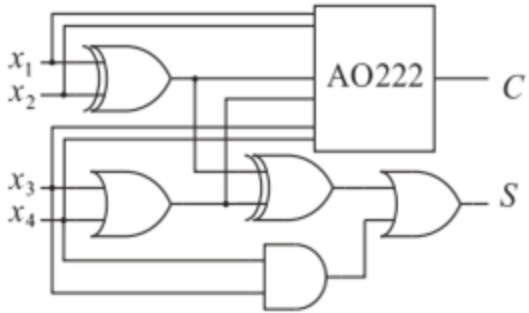



*Figure: yang-2 Compressor*

| $x_4 \ldots x_1$ | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS | 00 | 01 | 01 | 10 | 01 | 10 | 10 | 11 | 01 | 10 | 10 | 11 | 11 | 11 | 11 | 11 |
| E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | +1 | 0 | 0 | -1 |

*Table: Truth Table for yang-2 Compressor*

# Lin

- The lin compressors proposed in [2] have an XOR, an inverter, and a MUX-2 on the critical path.
- The circuit design for “Lin” compressor is shown in figure on the right side.
- “Lin” introduces only one error as shown in the truth table below.

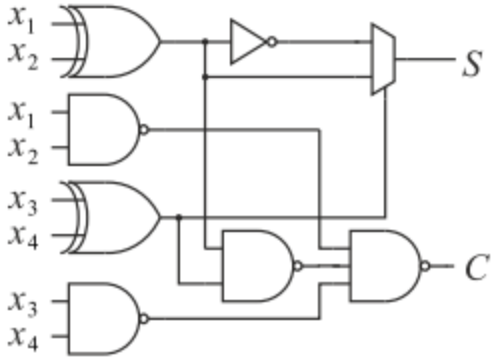



*Figure: lin Compressor*

| $x_4 \ldots x_1$ | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS | 00 | 01 | 01 | 10 | 01 | 10 | 10 | 11 | 01 | 10 | 10 | 11 | 10 | 11 | 11 | 10 |
| E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -2 |

*Table: Truth Table for lin Compressor*

# Strollo-1

- The strollo compressors proposed in [3] uses the “stacking circuit” to propose architectures of compressors.
- “Strollo-1” introduces four errors as shown in the truth table below.
- The circuit design for “Strollo-1” compressors is shown in figure on right side.

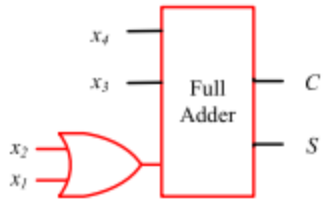



*Figure: strollo1 Compressor*

| $x_4 \ldots x_1$ | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS | 00 | 01 | 01 | 10 | 01 | 10 | 10 | 11 | 01 | 10 | 10 | 11 | 01 | 10 | 10 | 11 |
| E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | -1 | -1 | -1 |

*Table: Truth Table for strollo-1 Compressor*

# Strollo-2

- The strollo-2 compressors proposed in [3] introduces two errors as shown in the truth table below.
- The circuit design for “Strollo-2” compressors is shown in figure on the right side.

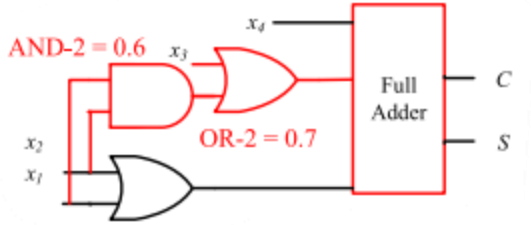



*Figure: Strollo2 Compressor*

| $x_4 \ldots x_1$ | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS | 00 | 01 | 01 | 10 | 01 | 10 | 10 | 10 | 01 | 10 | 10 | 11 | 10 | 11 | 11 | 11 |
| E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 |

*Table: Truth Table for stollo2 Compressor*

# Momeni

- The low-accuracy “Momeni” compressor introduces four errors as shown in the truth table below.
- The circuit design for “Momeni” compressors is shown in figure on the right side.

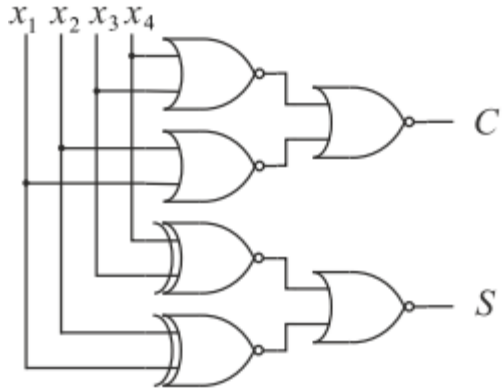



*Figure: momeni Compressor*

| $x_4 \ldots x_1$ | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS | 01 | 01 | 01 | 01 | 01 | 10 | 10 | 11 | 01 | 10 | 10 | 11 | 01 | 11 | 11 | 11 |
| E | +1 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | -1 |

*Table: Truth Table for momeni Compressor*

# Venka

- The venka compressors proposed in [4] has significant performance advantages.
- “Venka” introduces five errors as shown in the truth table below.
- The circuit design for “Venka” compressors is shown in figure on the right side.

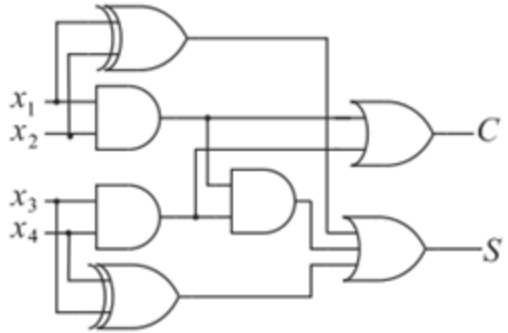



*Figure: venka Compressor*

| $x_4 \ldots x_1$ | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS | 00 | 01 | 01 | 10 | 01 | 01 | 01 | 11 | 01 | 01 | 01 | 11 | 10 | 11 | 11 | 11 |
| E | 0 | 0 | 0 | 0 | 0 | -1 | -1 | 0 | 0 | -1 | -1 | 0 | 0 | 0 | 0 | -1 |

*Table: Truth Table for venka Compressor*

# Sabetz

- The sabetz compressors proposed in [5] introduces as many as eight error as shown in the truth table below.

- The circuit design for “Sabetz” compressors is shown in figure on the right side.


*Figure: sabetz Compressor*

| $x_4 \ldots x_1$ | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS | 01 | 01 | 01 | 01 | 01 | 11 | 01 | 11 | 01 | 11 | 01 | 11 | 11 | 11 | 11 | 11 |
| E | -1 | 0 | 0 | -1 | 0 | +1 | -1 | 0 | 0 | +1 | -1 | 0 | -1 | 0 | 0 | -1 |

*Table: Truth Table for sabetz Compressor*

# Our Proposal

Change the multiplication operation in the convolution function of CNNs with approximate multiplication.

↓

Use the compressor designs for approximate multiplication. We first try approximation for integer multiplication. This can be later extended to floating-point multiplication.

↓

The neural networks are then simulated under these approximate conditions.

# Methodology

Coding the different compressors: The scheme uses half-adders, full adders, exact compressors and approximate compressors

Normalising the inputs of the 8x8 multiplier, since the weights of the neural networks are very small (nearly equal to zero).

Handling the sign separately (as our multiplier is unsigned).

# RESULTS

**Network: LeNet5  Dataset: MNIST**

**Base Accuracy (for floating-point multiplication): 98.49%**

**Base Accuracy (for exact integer multiplication): 98.55%**

| | Yang2 | Lin | Strollo1 | Momeni | Venka | Sabetz | Strollo2 |
|---|---|---|---|---|---|---|---|
| (a x b) multiplication | 98.48 | **98.56** | 98.36 | 13.84 | 98.47 | 9.61 | **98.58** |
| (b x a) multiplication | 98.46 | **98.59** | 98.32 | 14.04 | 98.27 | 9.82 | **98.57** |

# RESULTS

**Network: ResNet18  Dataset: Cifar10**

**Base Accuracy (for floating-point multiplication): 82.21%**

**Base Accuracy (for exact integer multiplication): 53.18%**

| | Yang2 | Lin | Strollo1 | Momeni | Venka | Sabetz | Strollo2 |
|---|---|---|---|---|---|---|---|
| (a x b) multiplication | **58.07** | **58.65** | **55.18** | 10 | 48.23 | 10 | **59.37** |
| (b x a) multiplication | **69.4** | **58.68** | 38.08 | 10 | 49.74 | 10 | **59.74** |

# Observations

- Designs like Yang2, Lin, Strollo1, Strollo2, and Venka showed accuracies very similar to(or in some cases, higher than) the base accuracy(exact multiplication).
- On the other hand, some specific designs, such as Momeni and Sabetz show very low accuracies, owing to their designs.
- Similarly, one interesting thing to observe is that the results for (a x b) ≠ (b x a)!
- This variation is more sharply visible in larger networks. For example, variation in Resnet18 is more clearly visible than in Lenet5.

# Conclusions

- Some of the designs such as Momeni and Sabetz were made in a very ad hoc manner, and this can be observed from the results that are being obtained as well!

- Approximation incorporated in most of the designs do not lead to a much drop in accuracy (<1% in many cases), while it reduces the hardware requirement by a much larger percentage. This shows the potentials of approximation.

- $(a \times b) \neq (b \times a)$ for our approximate multiplier as there is violation of the commutative property as proposed in [Should we code differently when using approximate circuits.]

# Future Work

- Approximate fixed-posit multipliers.
- Incorporation into other neural networks.
- axb or bxa?

# Acknowledgements

We would like to acknowledge Kailash Prasad(PhD Scholar, IIT Gandhinagar), Arpita Kabra(Senior Undergrad., IITGn), and Mallikarjuna P.(MTech, IITGn) for providing us the guidance and resources whenever required.

We would also like to thank Prof. Joycee Mekie for her continuous support, encouragement, and motivation during the project.

# Thank you for listening!

*Hope you liked the presentation!*

## References:


1. Antonio Giuseppe Maria Strollo, Ettore Napoli, Davide De Caro, Nicola Petra, and Gennaro Di Meo, "Comparison and Extension of Approximate 4-2 Compressors for Low-Power Approximate Multipliers".
2. V. Gohil, S. Walia, J. Mekie, and M. Awasthi, “Fixed-posit: A floating-point representation for error-resilient applications,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 68, no. 10, pp. 3341–3345, 2021.
3. Sumit Walia, Bachu Varun Tej, Arpita Kabra, Joydeep Devnath, and Joycee Mekie, "Fast and Low-Power Quantized Fixed Posit High-Accuracy DNN Implementation".
4. Tianqi Kong and Shuguo Li, "Design and Analysis of Approximate 4-2 Compressors for High-Accuracy Multipliers".
5. J. K. Devnath, N. Surana, and J. Mekie, “A mathematical approach towards quantization of floating point weights in low power neural networks,” in 2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID),2020, pp. 177–182.
6. J. Lu, S. Lu, Z. Wang, C. Fang, J. Lin, Z. Wang, and L. Du, “Training deep neural networks using posit number system,” in 2019 32nd IEEE International System-on-Chip Conference (SOCC), 2019.
7. Ankita Nandi, Chandan Kumar Jha and Joycee Mekie, “Should We Code Differently When Using Approximate Circuits?
8. John L. Gustafson, "Posit Arithmetic", 10 October 2017.

# References:

Link to GitHub Repository: https://github.com/arpitakabra/Approximate-Computing-for-Neural-Networks

Link to simulation results sheet for reference: https://docs.google.com/spreadsheets/d/1kAO4w9fM9XEo5n2gxIWQdGn60jpu7Sf4ovAzCXjycgQ/edit?usp=sharing