Summer Internship Report on

“Development Of An Approximate Perceptron For Image Processing & Neural Network”

Submitted by: Tripti Golchha
College Reg no: 2214007
Division of Electronics and Communication Engineering,
National Institute of Technology, Silchar, Assam
Duration: 15 May - 13 July, 2024

Under the supervision of
Prof. Roy Paily Palathinkal
Department of EEE, IIT Guwahati

ACKNOWLEDGEMENT

I express my overwhelming indebtedness in God Almighty for the successful completion of my project. First and foremost, I express my gratitude to Dr. Roy P. Paily, Professor and Head of Electronics and Electrical Engineering, at IIT Guwahati, Assam. I would also like to thank my project guide Anupam Kumari, PhD Scholar (VLSI), Dept. of EEE, IIT Guwahati for guiding me in my project and providing me with adequate facilities in order to complete the task I had undertaken. My sincere gratitude is expressed and extended to lab staff and other faculty members of the Electronics and Electrical Division for their assistance and unwavering support. I proudly thank all my friends and co-interns who helped me in various stages of this endeavour.

Table of Contents

.

1.Abstract

The primary objective of this internship was to develop an approximate perceptron model for image processing and neural network applications. The project focused on designing, implementing, and testing an 8x8 approximate multiplier, which was integrated into a perceptron model to enhance efficiency and performance in image processing tasks. Approximate computing, which leverages the trade-off between accuracy and resource consumption, was utilised to optimise computational efficiency and reduce power consumption.

The project began with an extensive literature review on approximate computing and its applications in neural networks. This was followed by the design of a 4x4 approximate multiplier, which was then extended to an 8x8 architecture. The designed multiplier was verified using Synopsys tools with UMC 65nm technology, and key performance metrics such as area, power, delay, and error rates were calculated.

The approximate multiplier was integrated into a perceptron model, which was then applied to image processing tasks including smoothing and sharpening. The performance of the model was evaluated using standard datasets like CIFAR-10 and MNIST, and metrics such as Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) were used to assess image quality.

The results demonstrated significant improvements in terms of area and power consumption while maintaining acceptable error rates. The approximate perceptron model showed promising results in image processing tasks, with minimal quality degradation. The project successfully highlighted the potential of approximate computing in enhancing the efficiency of neural networks and set the stage for future research in this domain.

2.Introduction

In recent years, the field of approximate computing has gained significant traction, particularly in applications where power efficiency and computational speed are critical. Approximate computing leverages the trade-off between computational accuracy and resource consumption, making it particularly suitable for error-tolerant applications such as image processing and neural networks. The objective of this project was to explore the implementation of an 8x8 approximate multiplier within a perceptron model and evaluate its impact on performance and efficiency in image processing tasks.

The project was conducted under the supervision of Prof. Roy Paily Palathinkal at IIT Guwahati. The internship provided an excellent opportunity to gain practical experience in VLSI design and approximate computing, aligning with the current trends and demands in the field of electronics and electrical engineering.

3.Approximate Computation

Approximate computation is a computing paradigm that trades off computational accuracy for efficiency, aiming to reduce resource usage such as energy, time, and memory while still providing acceptable results. This approach leverages the observation that many applications, including multimedia processing, machine learning, and big data analytics, do not always require exact precision to be effective. Techniques in approximate computation can include reduced precision arithmetic, algorithmic approximations, and hardware-level optimizations. By intentionally introducing controlled inaccuracies, approximate computation can achieve significant improvements in performance and energy efficiency, making it particularly valuable in resource-constrained environments such as mobile devices and embedded systems.

3.1 Characteristics of Approximate Computing:

- Accuracy vs. Efficiency Trade-off
- Reduced Precision Calculations
- Algorithmic Approximations
- Hardware-Level Optimizations
- Context-Aware Adaptation
- Error Tolerance
- Energy Efficiency
- Performance Gains
- Scalability
- Application-Specific Approaches

3.2 Advantages of Approximate Computing:.

- Improved Performance: Faster computations with reduced resource usage.
- Energy Efficiency: Lower power consumption, ideal for portable devices.
- Reduced Computational Load: Handles large data sets more efficiently.
- Cost-Effective: Reduces hardware and operational costs.
- Enhanced Speed: Speeds up processing time for real-time applications.
- Scalability: Easily scales with increasing data complexity.
- Flexibility: Adaptable to various application requirements.
- Resource Optimization: Utilises resources more effectively.

3.3 Disadvantages of Approximate Computing:

- Accuracy Loss: Reduced precision may lead to significant errors in critical applications.
- Error Propagation: Errors can accumulate, affecting overall system reliability.
- Limited Applicability: Not suitable for all tasks, particularly those requiring high accuracy.
- Complexity in Design: Developing and implementing approximate algorithms and hardware can be challenging.

4. Designed 8*8 Approximate Multiplier

Designing 8x8 multipliers based on a recursive method involves using different building blocks to balance electrical performance and accuracy. Each 8-bit unsigned number is divided into two 4-bit parts, and the multiplications aLbL, aHbL, aLbH, and aHbH are performed using 4x4 multipliers. The resulting sub-products are then added together using an exact adder as used in N8-5 and N8-6 or some others using the addition method shown in Fig 1 used in multipliers N8-L1 and N8-L2.

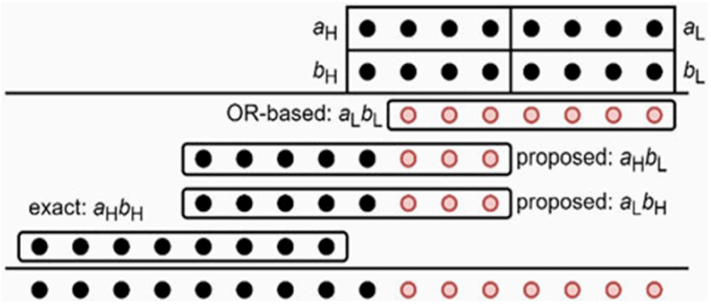
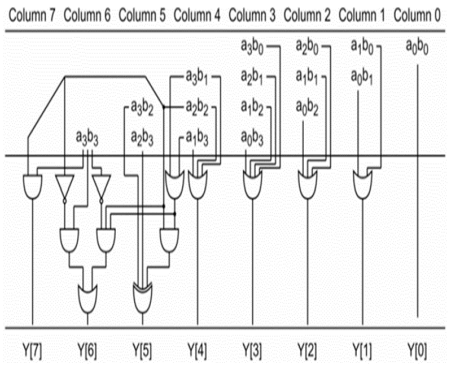


Fig 1: . Proposed 8x8 approximate multiplier architecture. Red bits are added with OR gates, black bits with exact adders.

4.1 Proposed 8*8 Approximate Multiplier Architecture

In the N1 and N2 4*4 multiplication technique, bitwise addition occurs as shown in Fig 2 and Fig 3 Respectively.. The last bits are added simply using OR-gate while the MSB bits are added using the custom gate combination.



:

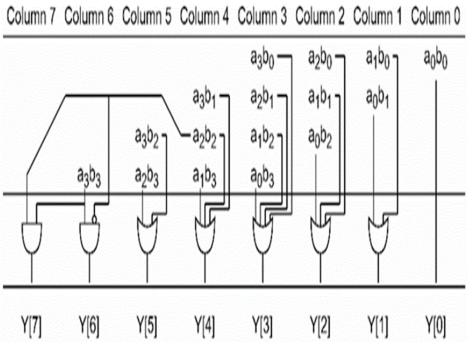


Fig 2: Design of N1 4*4 Multiplier

Fig 3: Design of N2 4*4 Multiplier

Both the designs aim at replacing the heavy and bulky XOR gates with simpler architecture, so as to improve the performance of the multiplier model. Moreover, four different 8*8 Multipliers are designed using different positions of N1 or N2 multipliers as shown in Fig 4.

Design		a_Hb_H	a_Lb_H	a_Hb_L	a_Lb_L
Proposed	N8-L1	Exact	N1	N1	OR-based
	N8-L2	Exact	N2	N2	OR-based
Proposed	N8-5	Exact	Exact	Exact	N1
	N8-6	Exact	Exact	Exact	N2

Fig 4: The Multiplier Design Schemes of N8-L1, N8-L2, N8-5 & N8-6

4.1.1 N8-L1 Architecture:

As mentioned in the section above, this design consists of four 4*4 multiplier , where , albl does OR-based operation, albh & ahbl does N1 operation and ahbh is obtained by exact multiplication. The final addition is done as shown in fig 1.

4.1.1 N8-L2 Architecture:

Similarly, this design also consists of four 4*4 multiplier , where , albl does OR-based operation, albh & ahbl does N2 operation and ahbh is obtained by exact multiplication. The final addition is done as shown in fig 1.

4.1.1 N8-5 Architecture:

This design consists of four 4*4 multiplier , where , albl does N1 operation while, albh, ahbl & ahbh is obtained by exact multiplication. The final addition is done by exact addition.

4.1.1 N8-6 Architecture:

Lastly, this design consists of four 4*4 multiplier , where , albl does N2 operation while, albh, ahbl & ahbh is obtained by exact multiplication. The final addition is done by exact addition.

4.2 Performance Testing

The proposed and reference circuits are synthesised using UMC 65 nm technology with Cadence Genus and appropriate timing constraints. Power dissipation is measured by

simulating the final netlist with random inputs to determine the switching activity of each node. The input vector array is consistent across designs with the same input bit width. "Min delay" indicates the strictest timing constraint where each circuit is synthesised with non-negative slack, reflecting the maximum working speed of the design. Area, power, and delay are compared to those of 8*8 exact multiplier, with exact design described in HDL and optimised by the synthesiser. Error performance is evaluated through exhaustive simulations for 8x8 multipliers.

Error metrics include:

1. **Normalised Mean Error Distance (NMED)**: Average error distance divided by the maximum possible value.
2. **Mean Relative Error Distance (MRED)**: Average relative error distance.
3. **Number of Effective Bits (NoEB)**: Calculated using mean square error.
4. **Error Rate (ER)**: Ratio of erroneous multiplications to total possible inputs.

5. Performance report of 8*8 Approximate Multipliers

The results of the 8*8 approximate multipliers are shown in Table 1. Exact design is reported at the top part of the table, while selected approximate designs following different methodologies are shown at the bottom part. All circuits are synthesised for a 1000ps timing constraint. The total power reported in the table is computed for a clock frequency equal to 1GHz.

5.1 Performances of 8*8 Approximate Multiplier

As clearly seen in the table, we must note that the proposed multiplier design has significantly reduced the area and power of the circuit, with very minimal error data.

Table 1: Performances of 8x8 approximate multipliers

Design	Area (μm^2)	Power (μm^2)	Delay (ns)	Error Rate(%)	MRED	NMED	NoEB
EXACT(8x8)	627.8	27.0609	2.21	-	-	-	16
N8-L1 (Accurate)	420.12	18.9577	1.51	76.858	0.0234	0.00259	8.009
N8-L1 (Reduced)	420.5	18.0169	1.46	76.858	0.0234	0.00259	8.009
N8-L2 (Accurate)	394.92	18.3509	1.57	77.557	0.0289	0.00341	7.64
N8-L2 (Reduced)	396	17.3835	1.46	77.557	0.0289	0.00341	7.64
N8-5	623.88	19.3738	2.22	35.937	0.001173	0.000061	12.83
N8-6	611.64	19.0123	2.21	37.109	0.001585	0.000084	12.42

5.2 Applications

Image processing is one of the most commonly considered error resilient applications and many papers test the proposed circuits in this scenario. In this paper, two image processing applications are considered: image blurring and image edge detection. The applications provide a more in depth understanding of the applicability range of the proposed designs.

5.2.1 Image Smoothing

In image processing, low pass filtering results in image smoothing which effectively removes the high spatial frequency noise from the image. The low-pass filter exploits a moving kernel that processes one pixel at a time and modifies it considering the pixels in proximity. The processing of each pixel requires a number of multiplications that depend on the size of the kernel. In fact, the value of the modified pixel is the weighted average of the neighbouring pixels. Moreover, image blurring is an error tolerant application, as the human eye is not able to detect trivial details. The kernel considered for smoothing is a two dimensional, rotationally symmetric, 3*3 Gaussian low-pass filter, with a standard deviation equal to 1.5.

The same processing has been also performed with exact multipliers to provide an effective comparison for all designs shown in Table 2. The structural similarity index (SSIM) and the peak signal to noise ratio (PSNR) provide a numerical indication of each multiplier's performance in image smoothing. Fig 5 shows the image results of the execution.

Table 2: Performances of 8x8 approximate multipliers in image smoothing

Design	PSNR(dB)	SSIM(%)
N8-L1	41.34	99.977
N8-L2	37.285	99.947
N8-5	59.029	99.999
N8-6	59.029	99.999

Fig 5. Gaussian smoothing of images obtained with different multipliers

5.2.2 Image Edge Detection

Image edge detection using approximate multipliers involves leveraging the trade-offs in accuracy and resource usage to enhance computational efficiency. By utilising approximate multipliers, which offer reduced precision, edge detection algorithms can process images faster and with lower power consumption. This approach is particularly useful in real-time image processing applications where speed and energy efficiency are crucial, and minor inaccuracies are acceptable. The reduced complexity of approximate multipliers can accelerate edge detection while maintaining sufficient quality for practical use.

Sharpening or high pass filtering aims to make fine details more distinct and remove the blurring of a digital image, by enhancing transitions in the spatial intensity of the image. High frequencies are boosted while low frequencies are reduced. It should be noted that over-sharpening might result in unwanted halo artefacts. The image sharpening process is similar to the smoothing process, but it uses a different kernel for the convolution.

SSIM and PSNR with respect to the sharpened image by exact multipliers are reported in Table 3. All proposed circuits have a high similarity ratio with the reference image as proved in Fig 6. Even though there are better performing multipliers for this application, the proposed circuits exhibit reasonable behaviour for such low power designs.

Table 3: Performances of 8x8 approximate multipliers in image edge detection

Design	PSNR(dB)	SSIM(%)
N8-L1	35.039	99.749
N8-L2	30.662	99.044
N8-5	41.176	99.94
N8-6	41.176	99.94

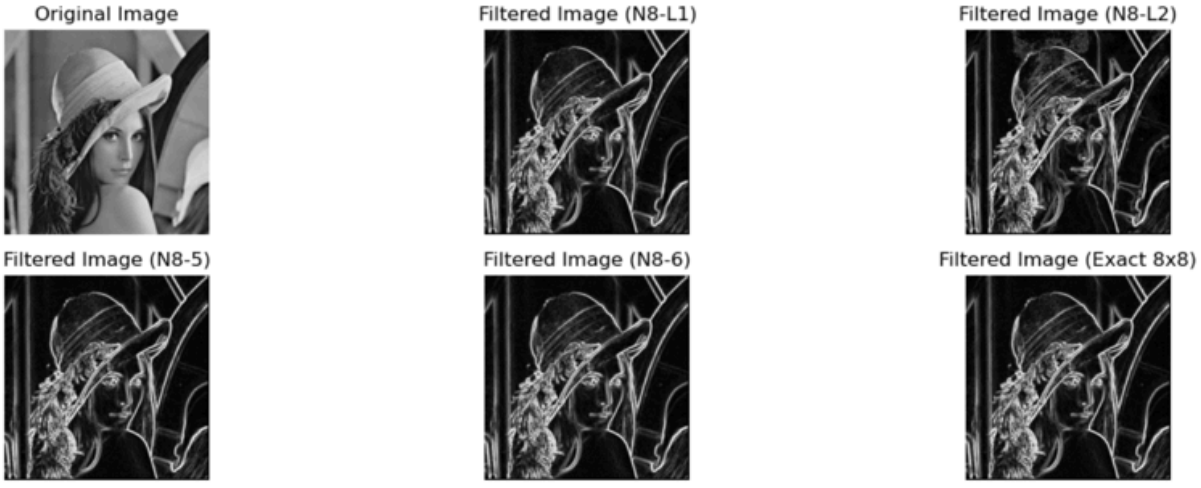


Fig 6. Sobel Edge Detection of images obtained with different multipliers.

5.2.3. Convolutional Neural Network

5.3 Inference from the Performance Report

The implementation of approximate multipliers has shown substantial improvements in both area and power efficiency in digital circuit designs. By adopting lower precision arithmetic, these multipliers effectively reduce the hardware footprint and power consumption compared to their exact counterparts. This reduction in complexity translates to a smaller silicon area and decreased energy usage, which is particularly beneficial for resource-constrained

environments and real-time applications. The trade-off in precision is strategically managed to ensure that the errors introduced remain minimal, thereby achieving significant gains in computational efficiency without substantially compromising overall performance.

In the context of image processing, such as edge detection, approximate multipliers have proven to deliver nearly accurate results while benefiting from reduced computation time. The minimal errors introduced by these multipliers do not notably degrade the quality of image processing outcomes. Consequently, approximate multipliers provide a practical solution for high-speed and energy-efficient image processing applications, maintaining a balance between resource optimization and result accuracy. This demonstrates their effectiveness in scenarios where both performance and power efficiency are critical.

6. Perceptron

A perceptron is a type of artificial neuron and the simplest form of a neural network used for binary classification tasks. It consists of one or more input features, each associated with a weight, and a bias term. The perceptron computes a weighted sum of the inputs, applies an activation function (typically a step function), and outputs a binary result based on whether the weighted sum exceeds a threshold. It is used to classify input data into one of two classes and can be trained using the perceptron learning algorithm, which adjusts the weights and bias based on the classification errors during training.

6.1 Working of a Perceptron

A perceptron works by taking multiple input features, each associated with a weight that signifies its importance. It computes a weighted sum of these inputs, adds a bias term, and then passes the result through an activation function. In a typical perceptron, the activation function is a step function that outputs one class label if the weighted sum exceeds a certain threshold, and another class label if it does not. During training, the perceptron adjusts the weights and bias based on the errors made in predictions to minimize misclassification. This iterative process helps the perceptron learn to classify input data correctly into one of the two possible categories.

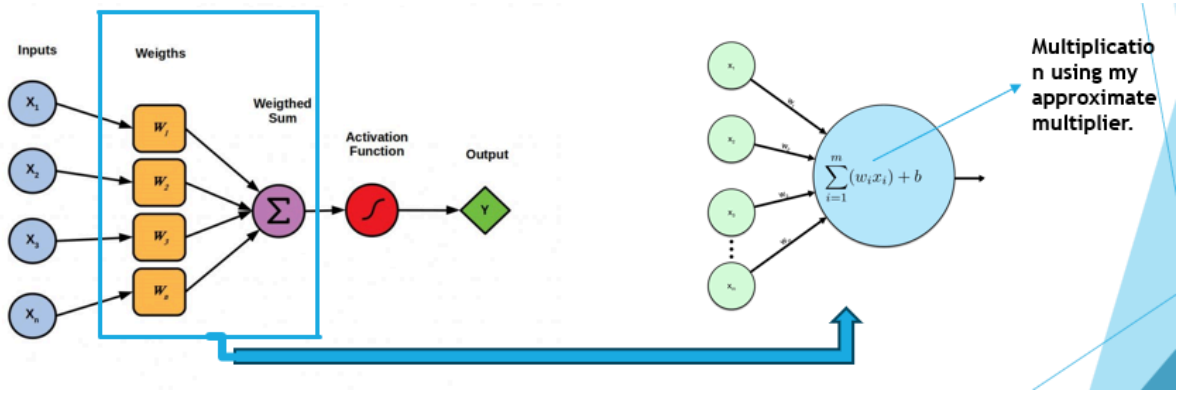


Fig 7. Single Layer Perceptron Model

6.2 Advantages of Perceptron

- Simplicity: Easy to understand and implement.
- Binary Classification: Effective for separating two classes.
- Computational Efficiency: Low resource requirements.
- Foundation for Advanced Models: Basis for more complex neural networks.

- Online Learning: Can update incrementally with new data.
- Convergence: Guaranteed to find a solution for linearly separable data.

6.3 Disadvantages of Perceptron

- Linear Boundaries: Only handles linearly separable data.
- Single-Layer Limitation: Lacks depth for complex patterns.
- No Probabilities: Provides binary outcomes without probability estimates.
- Sensitive to Scaling: Performance can be affected by feature scaling.
- Limited Training: Struggles with noisy or large datasets.

6.4 Exact Perceptron Architecture

The exact architecture of a perceptron consists of the following components:

1. **Input Layer**: This layer includes input nodes, each representing a feature of the data. Each input xix_ixi is associated with a corresponding weight wiw_iwi and a bias term b.
2. **Weighted Sum**: The perceptron computes a weighted sum of the inputs, given by the equation z=∑i wixi + bz = \sum_{i} w_i x_i + bz=∑iwixi+b, where xi are the input features, wi are the weights, and b is the bias.
3. **Activation Function**: The weighted sum z is passed through an activation function. For a simple perceptron, this is typically a step function that outputs one class if z is greater than or equal to a threshold (often zero), and another class otherwise.
4. **Output**: The perceptron produces a binary output based on the activation function's result, classifying the input into one of two categories.

This architecture forms the basic building block of more complex neural networks by providing a foundation for learning and classification.

7. Approximate Perceptron Design Architecture

Approximate multipliers are designed to trade off exact precision for reduced hardware complexity, area, and power consumption. This is particularly useful in resource-constrained environments or applications where slight inaccuracies are acceptable in exchange for efficiency. In the context of perceptron design, using approximate multipliers can enhance the hardware efficiency of neural networks.

7.1 Perceptron Architecture with Approximate Multipliers

7.1.1 Components:

1. **Input Layer**:
    - Consists of input nodes, each corresponding to a feature in the data.
2. **Weights**:
    - Each input xix_ixi is associated with a weight wiw_iwi. The weights are adjusted during the training process to minimise classification errors.
3. **Approximate Multipliers**:
    - These multipliers perform the product wi·xiw_i \cdot x_iwi·xi with reduced precision to save on area and power. Various approximate multipliers like N8-L1, N8-L2, N8-5, or N8-6 offer different trade-offs between accuracy and hardware efficiency.
4. **Weighted Sum**:
    - The perceptron computes the weighted sum zzz as z=∑iwi·xi+bz = \sum_{i} w_i \cdot x_i + bz=∑iwi·xi+b. Here, ·\cdot· represents multiplication using approximate multipliers.
5. **Bias Term**:
    - A bias bb is added to the weighted sum to shift the decision boundary. This bias can be constant or learned during training.
6. **Activation Function**:
    - The activation function, typically a step function, determines the output of the perceptron based on the weighted sum zzz. If z≥0z \geq 0z≥0, the output is usually 1 (or a positive class); otherwise, it is 0 (or a negative class).
7. **Output**:
    - The final output of the perceptron is determined by the activation function and represents the classification result.

7.1.2 Approximate Multiplier Types

1. **N8-L1 and N8-L2**:
    - **N8-L1**: An approximate multiplier with lower precision and reduced complexity, designed to optimize area and power. Suitable for applications where slight errors are acceptable.

    - **N8-L2**: An improved version offering a better trade-off between accuracy and hardware efficiency compared to N8-L1. It may use fewer bits or simplified arithmetic operations.
2. **N8-5 and N8-6**:
    - **N8-5**: An approximate multiplier that reduces complexity by simplifying the multiplication process. It is typically designed for applications with less stringent accuracy requirements.
    - **N8-6**: An enhanced multiplier with slightly better accuracy than N8-5 but still focusing on efficiency. It balances hardware savings with acceptable error margins.

7.1.3 Advantages of Using Approximate Multipliers in Perceptrons

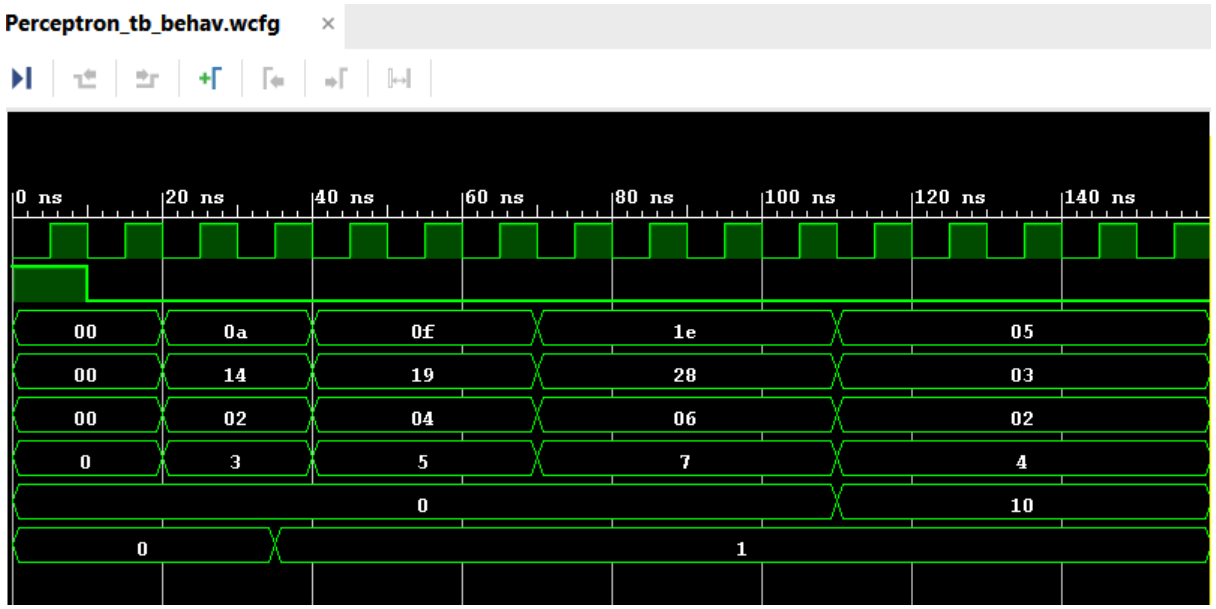
1. **Reduced Hardware Complexity**:
    - Approximate multipliers require fewer gates and less area compared to exact multipliers, leading to smaller and more cost-effective hardware designs.
2. **Lower Power Consumption**:
    - By reducing the number of operations and simplifying arithmetic, approximate multipliers consume less power, making the design more energy-efficient.
3. **Faster Computation**:
    - Simplified multipliers can perform operations more quickly, improving the overall processing speed of the perceptron.
4. **Scalability**:
    - The use of approximate multipliers allows for the scaling of perceptron architectures to larger networks with better hardware efficiency.

7.1.4 Considerations

1. **Error Tolerance**:
    - The effectiveness of approximate multipliers depends on the application's tolerance for inaccuracies. For tasks requiring high precision, the trade-offs may not be suitable.
2. **Training and Calibration**:
    - The perceptron’s training process may need adjustments to account for the inaccuracies introduced by approximate multipliers, ensuring acceptable performance levels.

In summary, integrating approximate multipliers such as N8-L1, N8-L2, N8-5, or N8-6 into the perceptron design architecture can significantly enhance hardware efficiency by reducing area and power consumption. This approach leverages the trade-off between accuracy and resource usage to optimise neural network implementations for specific applications.

7.2 Performance of Approximate Perceptron in Simulation & Waveforms:



7.3 Performance Report of Different Approximate Perceptron

Table 4 shows the area, power and delay reports of the approximate models of perceptron and how significantly it has reduced all the three factors.The perceptron using approximate multipliers were designed first in Verilog. The Verilog files were then run on synopsis @ UMC 65nm technology.

Table 4. Approximate Perceptron Performance WRT Exact Perceptron

Design	Area	Power	Delay
Exact	1421.640007	59.4891	2.78
N8-L1	1012.680003	42.1594	2.57
N8-L2	975.960002	41.1462	2.57
N8-5	1336.680003	51.2959	2.79
N8-6	1314.360003	50.9103	2.79

8. Perceptron design @ UMC 65nm

Synopsys is an Electronic Design Automation (EDA) tool used for design runs.

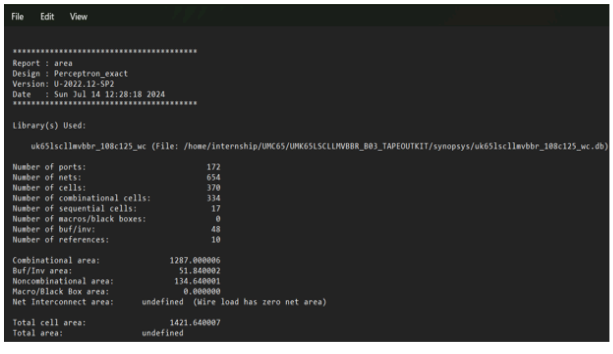
8.1 Methodology

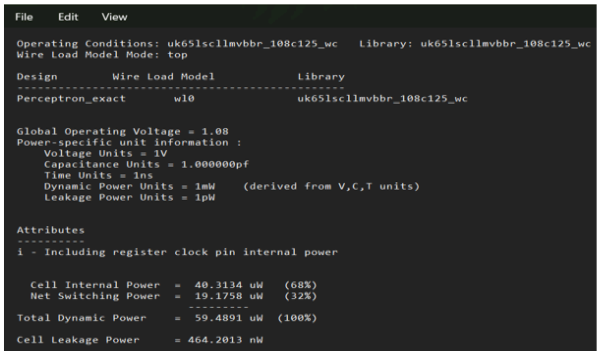
Implementing an approximate perceptron using 65 nm technology involves leveraging reduced-precision multipliers and simplified arithmetic operations to optimise area and power consumption. Techniques such as quantization, rounding, and custom logic design for approximate adders and multipliers are employed to minimise hardware complexity. Power and area are further optimised through clock and power gating, along with dynamic voltage and frequency scaling. Error tolerance is integrated into the design to manage inaccuracies, and training algorithms are adjusted to accommodate the approximations, ensuring that the perceptron maintains effective classification performance despite the reduced precision.

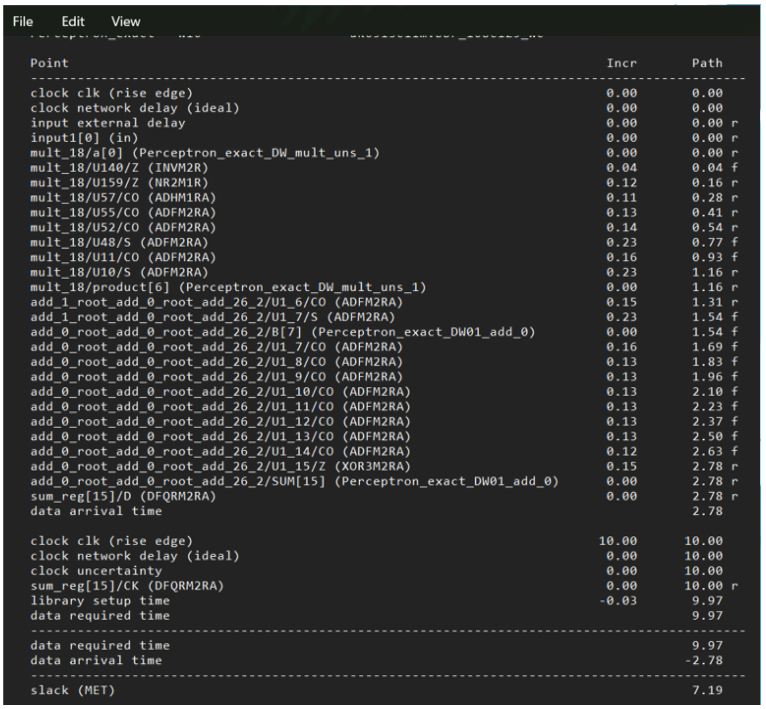
8.2 Area, Power & Delay Report files

The following screenshots are obtained for the various approximate perceptrons that were designed

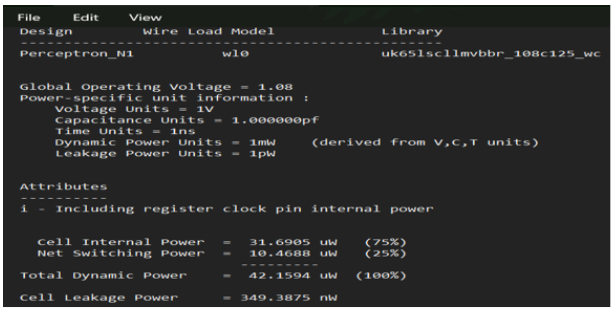
8.2.1 Area, Power & Delay of Perceptron with Exact Multiplier



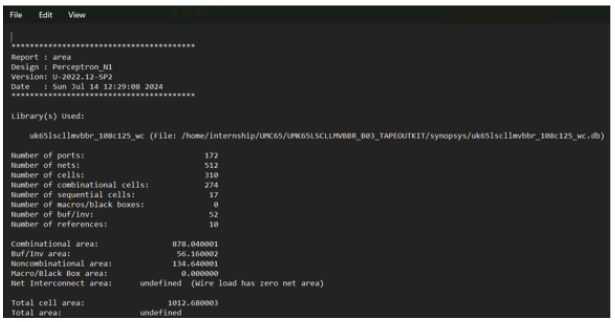




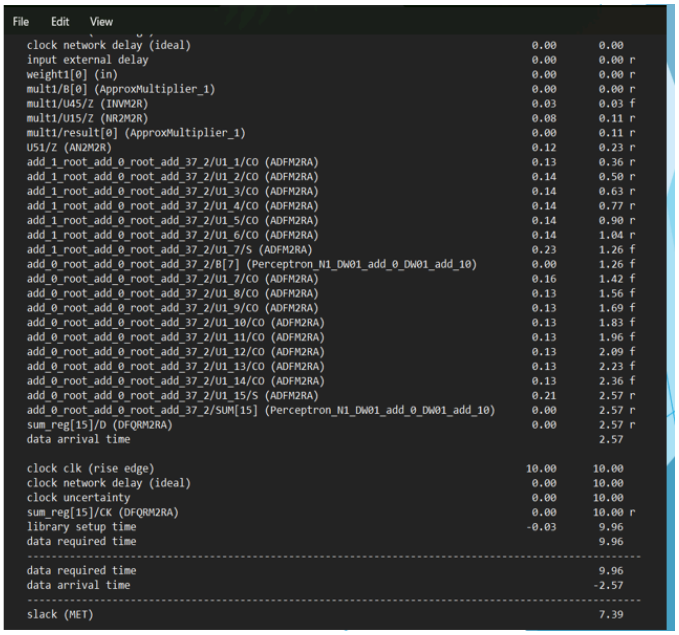
8.2.2 Area, Power & Delay of Approximate Perceptron with N8-L1



```
File    Edit    View
Design          Wire Load Model             Library
------------------------------------------------
Perceptron_N1           wl0                 uk65lscllmvbbr_108c125_wc

Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW

Attributes
----------
i - Including register clock pin internal power

   Cell Internal Power  =  31.6905 uW   (75%)
   Net Switching Power  =  10.4688 uW   (25%)
                           ---------
Total Dynamic Power     =  42.1594 uW   (100%)

Cell Leakage Power      = 349.3875 nW
```

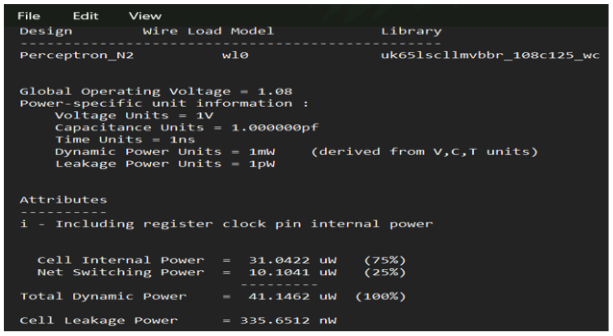


```
File    Edit    View

****************************************
Report : area
Design : Perceptron_N1
Version: U-2022.12-SP2
Date   : Sun Jul 14 12:29:08 2024
****************************************

Library(s) Used:

    uk65lscllmvbbr_108c125_wc (File: /home/internship/UMC65/UMK65LSCLLMVBBR_803_TAPEOUTKIT/synopsys/uk65lscllmvbbr_108c125_wc.db)

Number of ports:                  172
Number of nets:                   512
Number of cells:                  310
Number of combinational cells:    274
Number of sequential cells:        17
Number of macros/black boxes:       0
Number of buf/inv:                 52
Number of references:              10

Combinational area:          878.040001
Buf/Inv area:                 56.160002
Noncombinational area:       134.640001
Macro/Black Box area:          0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:            1012.680003
Total area:                 undefined
```

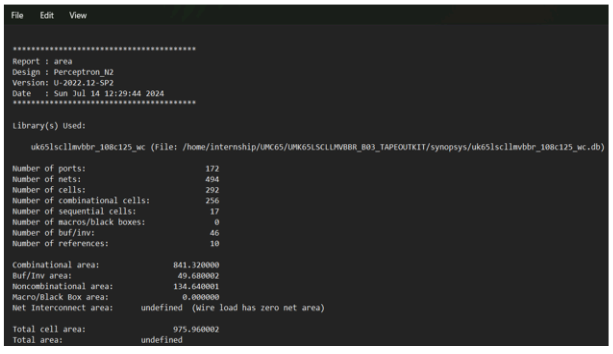


```
File    Edit    View

clock network delay (ideal)                                                    0.00       0.00
input external delay                                                           0.00       0.00 r
weight1[0] (in)                                                                0.00       0.00 r
mult1/B[0] (ApproxMultiplier_1)                                                0.00       0.00 r
mult1/U45/Z (INVM2R)                                                           0.03       0.03 f
mult1/U15/Z (NR2M2R)                                                           0.08       0.11 r
mult1/result[0] (ApproxMultiplier_1)                                           0.00       0.11 r
U51/Z (AN2M2R)                                                                 0.12       0.23 r
add_1_root_add_0_root_add_37_2/U1_1/CO (ADFM2RA)                               0.13       0.36 r
add_1_root_add_0_root_add_37_2/U1_2/CO (ADFM2RA)                               0.14       0.50 r
add_1_root_add_0_root_add_37_2/U1_3/CO (ADFM2RA)                               0.14       0.63 r
add_1_root_add_0_root_add_37_2/U1_4/CO (ADFM2RA)                               0.14       0.77 r
add_1_root_add_0_root_add_37_2/U1_5/CO (ADFM2RA)                               0.14       0.90 r
add_1_root_add_0_root_add_37_2/U1_6/CO (ADFM2RA)                               0.14       1.04 r
add_1_root_add_0_root_add_37_2/U1_7/S (ADFM2RA)                                0.23       1.26 f
add_0_root_add_0_root_add_37_2/B[7] (Perceptron_N1_DW01_add_0_DW01_add_10)     0.00       1.26 f
add_0_root_add_0_root_add_37_2/U1_7/CO (ADFM2RA)                               0.16       1.42 f
add_0_root_add_0_root_add_37_2/U1_8/CO (ADFM2RA)                               0.13       1.56 f
add_0_root_add_0_root_add_37_2/U1_9/CO (ADFM2RA)                               0.13       1.69 f
add_0_root_add_0_root_add_37_2/U1_10/CO (ADFM2RA)                              0.13       1.83 f
add_0_root_add_0_root_add_37_2/U1_11/CO (ADFM2RA)                              0.13       1.96 f
add_0_root_add_0_root_add_37_2/U1_12/CO (ADFM2RA)                              0.13       2.09 f
add_0_root_add_0_root_add_37_2/U1_13/CO (ADFM2RA)                              0.13       2.23 f
add_0_root_add_0_root_add_37_2/U1_14/CO (ADFM2RA)                              0.13       2.36 f
add_0_root_add_0_root_add_37_2/U1_15/S (ADFM2RA)                               0.21       2.57 r
add_0_root_add_0_root_add_37_2/SUM[15] (Perceptron_N1_DW01_add_0_DW01_add_10)  0.00       2.57 r
sum_reg[15]/D (DFQRM2RA)                                                       0.00       2.57 r
data arrival time                                                                         2.57

clock clk (rise edge)                                                         10.00      10.00
clock network delay (ideal)                                                    0.00      10.00
clock uncertainty                                                              0.00      10.00
sum_reg[15]/CK (DFQRM2RA)                                                      0.00      10.00 r
library setup time                                                            -0.03       9.96
data required time                                                                        9.96
-------------------------------------------------------------------------------------------------
data required time                                                                        9.96
data arrival time                                                                        -2.57
-------------------------------------------------------------------------------------------------
slack (MET)                                                                               7.39
```

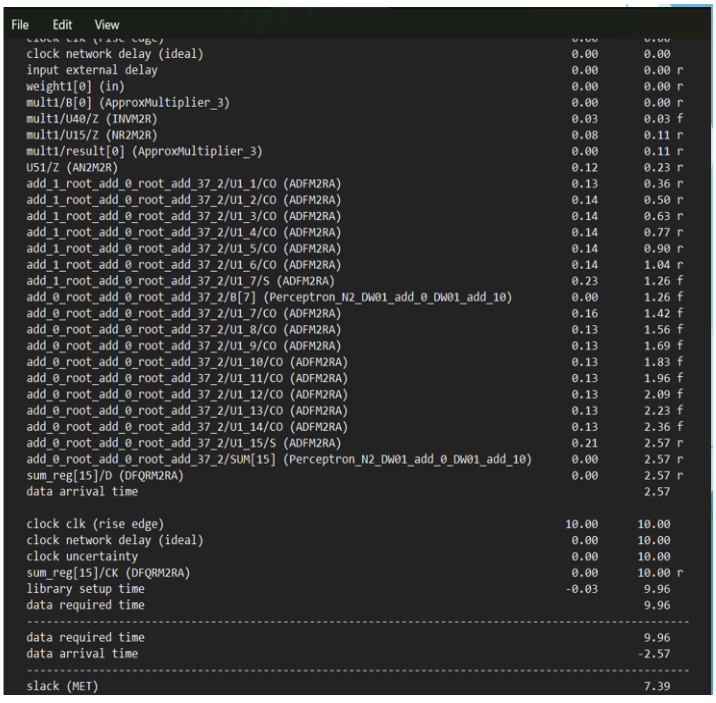
8.2.3 Area, Power & Delay of Approximate Perceptron with N8-L2



```
File    Edit    View
Design          Wire Load Model             Library
------------------------------------------------
Perceptron_N2           wl0                 uk65lscllmvbbr_108c125_wc

Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW

Attributes
----------
i - Including register clock pin internal power

   Cell Internal Power  =  31.0422 uW   (75%)
   Net Switching Power  =  10.1041 uW   (25%)
                           ---------
Total Dynamic Power     =  41.1462 uW   (100%)

Cell Leakage Power      = 335.6512 nW
```

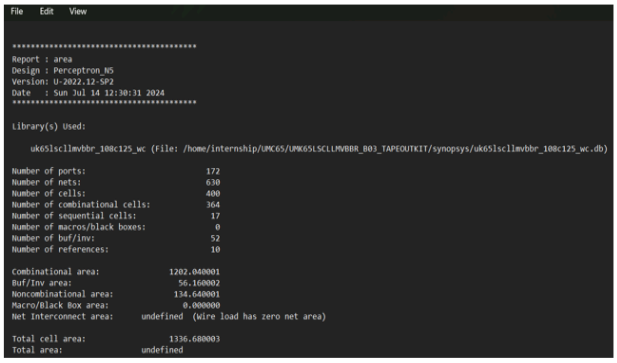


```
File    Edit    View

****************************************
Report : area
Design : Perceptron_N2
Version: U-2022.12-SP2
Date   : Sun Jul 14 12:29:44 2024
****************************************

Library(s) Used:

    uk65lscllmvbbr_108c125_wc (File: /home/internship/UMC65/UMK65LSCLLMVBBR_803_TAPEOUTKIT/synopsys/uk65lscllmvbbr_108c125_wc.db)

Number of ports:                  172
Number of nets:                   494
Number of cells:                  292
Number of combinational cells:    256
Number of sequential cells:        17
Number of macros/black boxes:       0
Number of buf/inv:                 46
Number of references:              10

Combinational area:          841.320000
Buf/Inv area:                 49.680002
Noncombinational area:       134.640001
Macro/Black Box area:          0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:             975.960002
Total area:                 undefined
```

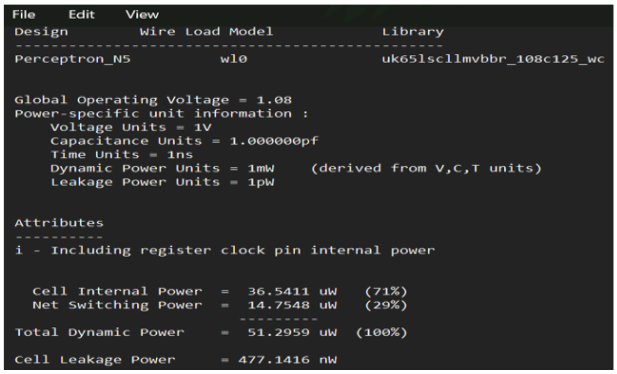


```
File    Edit    View

clock network delay (ideal)                                                    0.00       0.00
input external delay                                                           0.00       0.00 r
weight1[0] (in)                                                                0.00       0.00 r
mult1/B[0] (ApproxMultiplier_3)                                                0.00       0.00 r
mult1/U40/Z (INVM2R)                                                           0.03       0.03 f
mult1/U15/Z (NR2M2R)                                                           0.08       0.11 r
mult1/result[0] (ApproxMultiplier_3)                                           0.00       0.11 r
U51/Z (AN2M2R)                                                                 0.12       0.23 r
add_1_root_add_0_root_add_37_2/U1_1/CO (ADFM2RA)                               0.13       0.36 r
add_1_root_add_0_root_add_37_2/U1_2/CO (ADFM2RA)                               0.14       0.50 r
add_1_root_add_0_root_add_37_2/U1_3/CO (ADFM2RA)                               0.14       0.63 r
add_1_root_add_0_root_add_37_2/U1_4/CO (ADFM2RA)                               0.14       0.77 r
add_1_root_add_0_root_add_37_2/U1_5/CO (ADFM2RA)                               0.14       0.90 r
add_1_root_add_0_root_add_37_2/U1_6/CO (ADFM2RA)                               0.14       1.04 r
add_1_root_add_0_root_add_37_2/U1_7/S (ADFM2RA)                                0.23       1.26 f
add_0_root_add_0_root_add_37_2/B[7] (Perceptron_N2_DW01_add_0_DW01_add_10)     0.00       1.26 f
add_0_root_add_0_root_add_37_2/U1_7/CO (ADFM2RA)                               0.16       1.42 f
add_0_root_add_0_root_add_37_2/U1_8/CO (ADFM2RA)                               0.13       1.56 f
add_0_root_add_0_root_add_37_2/U1_9/CO (ADFM2RA)                               0.13       1.69 f
add_0_root_add_0_root_add_37_2/U1_10/CO (ADFM2RA)                              0.13       1.83 f
add_0_root_add_0_root_add_37_2/U1_11/CO (ADFM2RA)                              0.13       1.96 f
add_0_root_add_0_root_add_37_2/U1_12/CO (ADFM2RA)                              0.13       2.09 f
add_0_root_add_0_root_add_37_2/U1_13/CO (ADFM2RA)                              0.13       2.23 f
add_0_root_add_0_root_add_37_2/U1_14/CO (ADFM2RA)                              0.13       2.36 f
add_0_root_add_0_root_add_37_2/U1_15/S (ADFM2RA)                               0.21       2.57 r
add_0_root_add_0_root_add_37_2/SUM[15] (Perceptron_N2_DW01_add_0_DW01_add_10)  0.00       2.57 r
sum_reg[15]/D (DFQRM2RA)                                                       0.00       2.57 r
data arrival time                                                                         2.57

clock clk (rise edge)                                                         10.00      10.00
clock network delay (ideal)                                                    0.00      10.00
clock uncertainty                                                              0.00      10.00
sum_reg[15]/CK (DFQRM2RA)                                                      0.00      10.00 r
library setup time                                                            -0.03       9.96
data required time                                                                        9.96
-------------------------------------------------------------------------------------------------
data required time                                                                        9.96
data arrival time                                                                        -2.57
-------------------------------------------------------------------------------------------------
slack (MET)                                                                               7.39
```

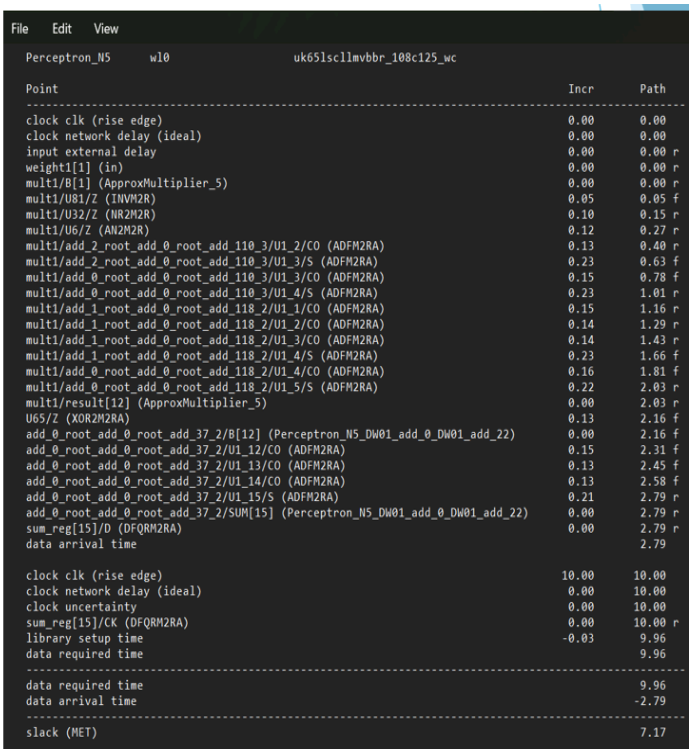
8.2.4 Area, Power & Delay of Approximate Perceptron with N8-5



```
File    Edit    View

****************************************
Report : area
Design : Perceptron_N5
Version: U-2022.12-SP2
Date   : Sun Jul 14 12:30:31 2024
****************************************

Library(s) Used:

    uk65lscllmvbbr_108c125_wc (File: /home/internship/UMC65/UMK65LSCLLMVBBR_B03_TAPEOUTKIT/synopsys/uk65lscllmvbbr_108c125_wc.db)

Number of ports:                  172
Number of nets:                   630
Number of cells:                  400
Number of combinational cells:    364
Number of sequential cells:        17
Number of macros/black boxes:       0
Number of buf/inv:                 52
Number of references:              10

Combinational area:         1202.040001
Buf/Inv area:                 56.160002
Noncombinational area:       134.640001
Macro/Black Box area:          0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:            1336.680003
Total area:                 undefined
```

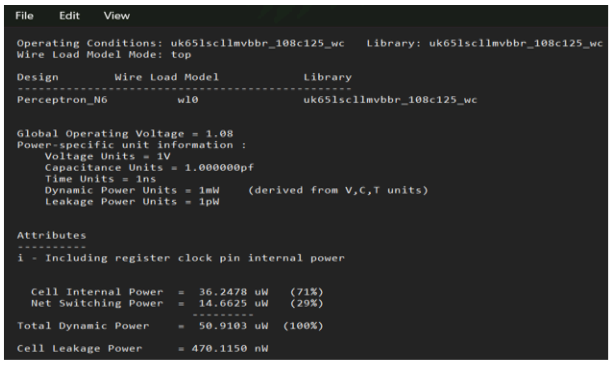


```
File    Edit    View
Design          Wire Load Model             Library
------------------------------------------------
Perceptron_N5           wl0                 uk65lscllmvbbr_108c125_wc


Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


Attributes
----------
i - Including register clock pin internal power


  Cell Internal Power  =  36.5411 uW   (71%)
  Net Switching Power  =  14.7548 uW   (29%)
                          ---------
Total Dynamic Power    =  51.2959 uW  (100%)

Cell Leakage Power     = 477.1416 nW
```

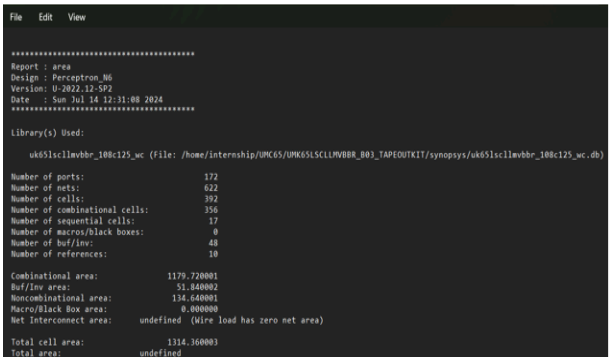


```
File    Edit    View
Perceptron_N5      wl0                 uk65lscllmvbbr_108c125_wc

Point                                                                                                   Incr       Path
--------------------------------------------------------------------------------------------------------------------------
clock clk (rise edge)                                                                                   0.00       0.00
clock network delay (ideal)                                                                             0.00       0.00
input external delay                                                                                    0.00       0.00 r
weight1[1] (in)                                                                                         0.00       0.00 r
mult1/B[1] (ApproxMultiplier_5)                                                                         0.00       0.00 r
mult1/U81/Z (INVM2R)                                                                                    0.05       0.05 f
mult1/U32/Z (NR2M2R)                                                                                    0.10       0.15 r
mult1/U6/Z (AN2M2R)                                                                                     0.12       0.27 r
mult1/add_2_root_add_0_root_add_110_3/U1_2/CO (ADFM2RA)                                                 0.13       0.40 r
mult1/add_2_root_add_0_root_add_110_3/U1_3/S (ADFM2RA)                                                  0.23       0.63 f
mult1/add_0_root_add_0_root_add_110_3/U1_3/CO (ADFM2RA)                                                 0.15       0.78 f
mult1/add_0_root_add_0_root_add_110_3/U1_4/S (ADFM2RA)                                                  0.23       1.01 r
mult1/add_1_root_add_0_root_add_118_2/U1_1/CO (ADFM2RA)                                                 0.15       1.16 r
mult1/add_1_root_add_0_root_add_118_2/U1_2/CO (ADFM2RA)                                                 0.14       1.29 r
mult1/add_1_root_add_0_root_add_118_2/U1_3/CO (ADFM2RA)                                                 0.14       1.43 r
mult1/add_1_root_add_0_root_add_118_2/U1_4/S (ADFM2RA)                                                  0.23       1.66 f
mult1/add_0_root_add_0_root_add_118_2/U1_4/CO (ADFM2RA)                                                 0.16       1.81 f
mult1/add_0_root_add_0_root_add_118_2/U1_5/S (ADFM2RA)                                                  0.22       2.03 r
mult1/result[12] (ApproxMultiplier_5)                                                                   0.00       2.03 r
U65/Z (XOR2M2RA)                                                                                        0.13       2.16 f
add_0_root_add_0_root_add_37_2/B[12] (Perceptron_N5_DW01_add_0_DW01_add_22)                             0.00       2.16 f
add_0_root_add_0_root_add_37_2/U1_12/CO (ADFM2RA)                                                       0.15       2.31 f
add_0_root_add_0_root_add_37_2/U1_13/CO (ADFM2RA)                                                       0.13       2.45 f
add_0_root_add_0_root_add_37_2/U1_14/CO (ADFM2RA)                                                       0.13       2.58 f
add_0_root_add_0_root_add_37_2/U1_15/S (ADFM2RA)                                                        0.21       2.79 r
add_0_root_add_0_root_add_37_2/SUM[15] (Perceptron_N5_DW01_add_0_DW01_add_22)                           0.00       2.79 r
sum_reg[15]/D (DFQRM2RA)                                                                                0.00       2.79 r
data arrival time                                                                                                  2.79

clock clk (rise edge)                                                                                  10.00      10.00
clock network delay (ideal)                                                                             0.00      10.00
clock uncertainty                                                                                       0.00      10.00
sum_reg[15]/CK (DFQRM2RA)                                                                               0.00      10.00 r
library setup time                                                                                     -0.03       9.96
data required time                                                                                                 9.96
--------------------------------------------------------------------------------------------------------------------------
data required time                                                                                                 9.96
data arrival time                                                                                                 -2.79
--------------------------------------------------------------------------------------------------------------------------
slack (MET)                                                                                                        7.17
```

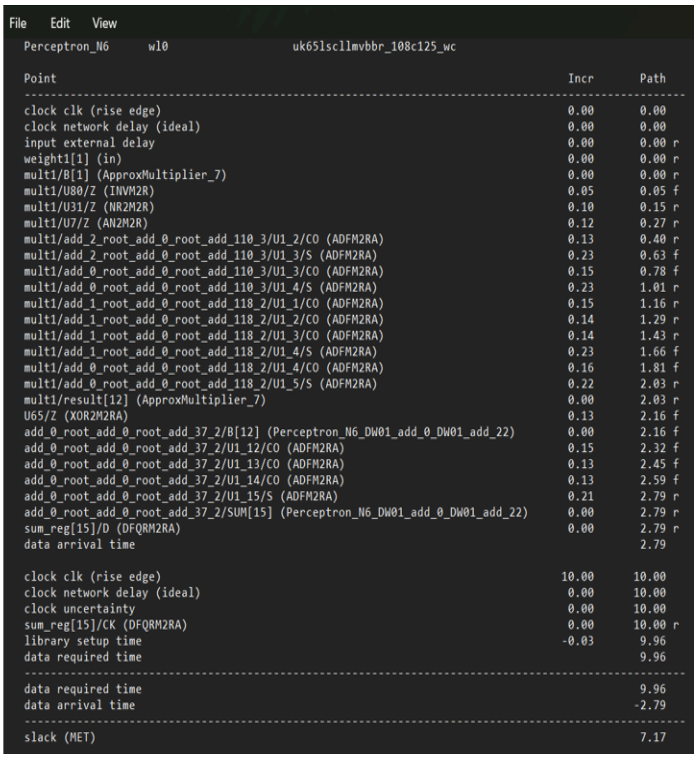
8.2.4 Area, Power & Delay of Approximate Perceptron with N8-6



```
File    Edit    View

Operating Conditions: uk65lscllmvbbr_108c125_wc   Library: uk65lscllmvbbr_108c125_wc
Wire Load Model Mode: top

Design          Wire Load Model             Library
------------------------------------------------
Perceptron_N6           wl0                 uk65lscllmvbbr_108c125_wc


Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


Attributes
----------
i - Including register clock pin internal power


  Cell Internal Power  =  36.2478 uW   (71%)
  Net Switching Power  =  14.6625 uW   (29%)
                          ---------
Total Dynamic Power    =  50.9103 uW  (100%)

Cell Leakage Power     = 470.1150 nW
```



```
File    Edit    View

****************************************
Report : area
Design : Perceptron_N6
Version: U-2022.12-SP2
Date   : Sun Jul 14 12:31:08 2024
****************************************

Library(s) Used:

    uk65lscllmvbbr_108c125_wc (File: /home/internship/UMC65/UMK65LSCLLMVBBR_B03_TAPEOUTKIT/synopsys/uk65lscllmvbbr_108c125_wc.db)

Number of ports:                  172
Number of nets:                   622
Number of cells:                  392
Number of combinational cells:    356
Number of sequential cells:        17
Number of macros/black boxes:       0
Number of buf/inv:                 48
Number of references:              10

Combinational area:         1179.720001
Buf/Inv area:                 51.840002
Noncombinational area:       134.640001
Macro/Black Box area:          0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:            1314.360003
Total area:                 undefined
```



```
File    Edit    View
Perceptron_N6      wl0                 uk65lscllmvbbr_108c125_wc

Point                                                                                                   Incr       Path
--------------------------------------------------------------------------------------------------------------------------
clock clk (rise edge)                                                                                   0.00       0.00
clock network delay (ideal)                                                                             0.00       0.00
input external delay                                                                                    0.00       0.00 r
weight1[1] (in)                                                                                         0.00       0.00 r
mult1/B[1] (ApproxMultiplier_7)                                                                         0.00       0.00 r
mult1/U80/Z (INVM2R)                                                                                    0.05       0.05 f
mult1/U31/Z (NR2M2R)                                                                                    0.10       0.15 r
mult1/U7/Z (AN2M2R)                                                                                     0.12       0.27 r
mult1/add_2_root_add_0_root_add_110_3/U1_2/CO (ADFM2RA)                                                 0.13       0.40 r
mult1/add_2_root_add_0_root_add_110_3/U1_3/S (ADFM2RA)                                                  0.23       0.63 f
mult1/add_0_root_add_0_root_add_110_3/U1_3/CO (ADFM2RA)                                                 0.15       0.78 f
mult1/add_0_root_add_0_root_add_110_3/U1_4/S (ADFM2RA)                                                  0.23       1.01 r
mult1/add_1_root_add_0_root_add_118_2/U1_1/CO (ADFM2RA)                                                 0.15       1.16 r
mult1/add_1_root_add_0_root_add_118_2/U1_2/CO (ADFM2RA)                                                 0.14       1.29 r
mult1/add_1_root_add_0_root_add_118_2/U1_3/CO (ADFM2RA)                                                 0.14       1.43 r
mult1/add_1_root_add_0_root_add_118_2/U1_4/S (ADFM2RA)                                                  0.23       1.66 f
mult1/add_0_root_add_0_root_add_118_2/U1_4/CO (ADFM2RA)                                                 0.16       1.81 f
mult1/add_0_root_add_0_root_add_118_2/U1_5/S (ADFM2RA)                                                  0.22       2.03 r
mult1/result[12] (ApproxMultiplier_7)                                                                   0.00       2.03 r
U65/Z (XOR2M2RA)                                                                                        0.13       2.16 f
add_0_root_add_0_root_add_37_2/B[12] (Perceptron_N6_DW01_add_0_DW01_add_22)                             0.00       2.16 f
add_0_root_add_0_root_add_37_2/U1_12/CO (ADFM2RA)                                                       0.15       2.32 f
add_0_root_add_0_root_add_37_2/U1_13/CO (ADFM2RA)                                                       0.13       2.45 f
add_0_root_add_0_root_add_37_2/U1_14/CO (ADFM2RA)                                                       0.13       2.59 f
add_0_root_add_0_root_add_37_2/U1_15/S (ADFM2RA)                                                        0.21       2.79 r
add_0_root_add_0_root_add_37_2/SUM[15] (Perceptron_N6_DW01_add_0_DW01_add_22)                           0.00       2.79 r
sum_reg[15]/D (DFQRM2RA)                                                                                0.00       2.79 r
data arrival time                                                                                                  2.79

clock clk (rise edge)                                                                                  10.00      10.00
clock network delay (ideal)                                                                             0.00      10.00
clock uncertainty                                                                                       0.00      10.00
sum_reg[15]/CK (DFQRM2RA)                                                                               0.00      10.00 r
library setup time                                                                                     -0.03       9.96
data required time                                                                                                 9.96
--------------------------------------------------------------------------------------------------------------------------
data required time                                                                                                 9.96
data arrival time                                                                                                 -2.79
--------------------------------------------------------------------------------------------------------------------------
slack (MET)                                                                                                        7.17
```

9. Synopsis Results Description

The specified design was synthesised and verified using the Synopsys tool following the steps mentioned in the previous section. The area, timing and power reports obtained after synthesis are given below.

9.1 Area Report

This will report the total area of the design, as well as an area break-down of each of the contained modules. The report lists the total combinational, non-combinational, buffer/inverter, and macro areas separately—it even includes an estimate of the interconnect area. The sum of the first four areas is called the cell area, while the sum of all listed area values is the total area. The figure 5 given below shows the area report generated through synopsis for the integrated design of RISC and AES along with UART. fig 5 : Area report from Synopsys

9.2 Power Report

Similar to the area report, the power report will also break down the power consumption for each module in the design. The report starts by reporting the operating voltage, as well as the voltage, capacitance, time, and power units. Then, for each module, the dynamic and leakage power are listed. These two quantities are expressed with different orders of magnitude (e.g., while the dynamic power is given in μW, the leakage power is in pW).The dynamic power is 26 decomposed into two parts: The internal power, which corresponds to the cells' internal nodes, and the switching power, which happens at the cells' input and output nodes. The figure 6 given below shows the area report generated through synopsis for the integrated design of RISC and AES along with UART fig 6: Power report from Synopsys

9.3 Timing Report

Synopsys DC details each one of the most critical paths, indicating the specific cells and ports through which the path goes, the delay added by each one of these cells, and even if the transition happening at each node in the critical path is rising (r) or falling (f). The difference between the clock signal's arrival time (called data required time in the report) and the data signal's arrival time (data arrival time) is called the slack; i.e., slack =data required time- data arrival time. For the circuit to work at the specified cycle time, the clock signal's rising edge should always arrive later (or at the same time) than the data signal, which means that slack should be non-negative. In case you have a positive slack for the most critical path, this means that data signal could get delayed (i.e., it could slack) for that amount of time, and it would still be sampled correctly at the flip-flop. Then, in principle, you could reduce the clock period by exactly the slack of the most critical path, and your circuit would still work. The figure 7 given below shows the area report generated through synopsis for the integrated design of RISC and AES along with UART. 27 fig 7: Timing report from Synopsys

10. Application & Implementation of Approximate Perceptron

The approximate perceptron design can be effectively applied to the MNIST dataset, a well-known benchmark for handwritten digit classification. By leveraging approximate computing techniques, such as approximate multipliers, the perceptron model can achieve faster inference times and lower power consumption while processing the dataset. This approach is particularly useful for resource-constrained environments, such as embedded systems or mobile devices, where power efficiency and processing speed are critical. The MNIST dataset will allow for evaluating the trade-offs between accuracy and performance, demonstrating how approximate perceptrons can maintain acceptable classification results with reduced hardware overhead. The results will provide valuable insights into the practical benefits of approximate computing in real-world applications of machine learning

10.1 Neural Network Test on MNIST Dataset

Table 5 shows the report and accuracy obtained when approximate perceptron was implemented to test and train MNIST dataset.

Table 5. Accuracy Analysis of Neural Network with Approximate Perceptron Layers

Design	Accuracy
Exact Perceptron	99%
N8-L1 Perceptron	81.7%
N8-L2 Perceptron	81.9%
N8-5 Perceptron	80.4%
N8-6 Perceptron	80.5%

11. Conclusion and future work

The project on designing an approximate perceptron using approximate computing has achieved significant improvements in hardware performance by employing approximate multipliers, leading to reductions in area and power consumption while retaining acceptable accuracy. The design illustrates how approximate computing can optimise resource usage in hardware without drastically compromising performance.

For future work, it is essential to conduct a comprehensive analysis of the accuracy trade-offs associated with various levels of approximation to better understand their impact on overall performance. Advanced optimization algorithms, such as genetic algorithms or reinforcement learning, should be explored to further fine-tune the design parameters for enhanced performance. Hardware validation on FPGA or ASIC platforms is crucial for assessing real-world performance, providing insights into speed and power consumption beyond simulation environments.

Comparative studies with traditional perceptron designs and other approximate computing methods will help benchmark and validate the effectiveness of the proposed approach. Additionally, exploring applications beyond perceptrons, such as neural networks or signal processing, could uncover new opportunities for approximate computing. Investigating fault tolerance and long-term reliability will ensure that the design remains robust under various conditions. These steps will advance the practical application of approximate computing in hardware design.

12. Reference

1. IEEE Paper on Approximate Recursive Multipliers Using Low Power Building Blocks by EFSTRATIOS ZACHARELOS , ITALO NUNZIATA , GERARDO SAGGESE , ANTONIO G.M. STROLLO , (Senior Member, IEEE), AND ETTORE NAPOLI, (Senior Member, IEEE)

2. Github Repository: victorfei/perceptronApproxMultiplier: Perceptron (simple machine learning model) + approximate multiplier, implemented with Python simulated light-weight processor (github.com)

3. Github Repository: approximate-multipliers · GitHub Topics

4. Other IEEE papers on various different types of approximate multipliers