DOP Submission

MID-SEMESTER REPORT

HARDWARE realization and TESTING of Approximate Adder

PREPARED BY –

YASH - 2021A8PS3049G

ACKNOWLEDGEMENT

I want to express our profound gratitude to Professor Dipankar Pal for his outstanding mentorship during our project. His expertise and commitment played a pivotal role in shaping the project's success. Professor Dipankar Pal provided invaluable insights, constructive feedback, and unwavering support, elevating our work to new heights.

We are deeply thankful for the time and effort Professor Dipankar Pal invested in guiding us through challenges and refining our project. Their passion for the subject matter ignited our own enthusiasm, creating a positive and collaborative atmosphere.

In conclusion, we extend our heartfelt gratitude to Professor Dipankar Pal. His mentorship has been transformative, leaving a lasting impact on my academic journey. Thank you for being an inspirational guide and for your dedication to our success.

TABLE OF CONTENTS

1. ABSTRACT:

The objective of this report is to enhance this approximate adder based on FPGA that was originally designed to have a 12-bit precision but whose aim shall be to enhance image processing for real-time applications. The approximation has been performed very carefully to 8 bits of which are found to be similar in pixel-level specifications such as digital photographs. Building on previously researched adders, such as the carry prediction adders and the NAND-based half adders, the study enhances the original design and minimizes power usage while enhancing the computational speed.

The report illustrates the use of the reduced 8-bit approximation adder in image processing tasks through design and simulation using Verilog. In particular, 8x8 picture segments are used to assess the adder's performance using image addition operations, and real-time simulations are used to verify the gains in accuracy and efficiency. In addition to highlighting the possibilities for power-efficient, high-performance digital circuit designs relevant to real-world systems like image compression and signal processing, this work emphasizes the trade-off between approximation and computing speed in FPGA-based systems.

The project encompasses the following key objectives:

Implementation:

Utilize Hardware Description Language (HDL) Verilog to implement the Approximate Adder design. Employ simulation tools to verify the accuracy and various other metrics of the design.

Application:

Using Image addition techniques to judge and evaluate according to other research papers.

Documentation and Reporting:

Prepare detailed documentation covering the design specifications, implementation details, testing methodologies, and results.

By successfully designing FPGA-based Image processing of Approximate Adder with its hardware realization and testing of adders, this project aims to enhance the computational capabilities for processing various tasks and justifying the real-world system, making it well-suited for a broad range of applications requiring efficient calculations.

2. INTRODUCTION:

In digital computing systems, arithmetic operations are the cornerstone of both simple and intricate computations. The need for more speed, economy, and optimal performance is growing, from basic data processing to complex activities like image, video, and signal processing. Especially in fields like artificial intelligence, machine learning, and embedded systems, where computing demands are growing, new designs that strike a compromise between accuracy and efficiency are becoming crucial. One such method is the use of approximation adders, which provide trade-offs between precision, speed, and power consumption by permitting controlled errors in arithmetic operations.

A margin of error within a specified range is introduced by approximate adders, in contrast to typical digital arithmetic circuits that yield accurate outputs. In applications where perfect precision is not necessary, such as image and video processing, where little errors usually have no effect on the final product's quality, this tolerance is frequently acceptable. Approximate adders can greatly enhance performance and lower power consumption by allowing these little inaccuracies, which makes them extremely useful in high-speed, low-power settings. These circuits are especially useful in applications that call for effective computational trade-offs because of their adaptability in adjusting the degree of approximation.

In a prior effort, Verilog was used to successfully develop an FPGA-based 12-bit approximation adder. This design investigated several architectural strategies to maximize computing performance and reduce energy consumption, such as carry prediction and reduced power-efficiency algorithms. The project showed how approximate adders can improve system performance, especially in digital circuits that need fast arithmetic operations. The foundation for incorporating approximation adders into useful applications was established by this study.

The new study aims to extend the use of the approximation adder to image processing, with a particular focus on hardware realization and testing, building on the accomplishments of the previous project. The initial 12-bit approximation adder was improved to an 8-bit design, bringing it into compliance with common pixel formats because digital photographs generally use 8-bit pixel data. This reduction in bit width ensures the adder's compatibility with image compression and processing tasks, where performance efficiency and power reduction are critical.

An image processing workflow was created in order to evaluate the performance of the 8-bit approximation adder in practical settings. This procedure starts with using MATLAB to convert RGB photos to greyscale. Next, the approximation adder is applied in image addition, which is a crucial step in image blending and improvement. The performance of the adder was simulated using a Verilog test bench, and the results showed that the adder processes picture data efficiently while retaining a reasonable level of accuracy.
Realizing the hardware implementation of an 8-bit approximation adder for real-time image processing applications is the main goal of this project. The research aims to test the adder's performance in real-world, energy-efficient applications including signal processing and image compression by incorporating it into FPGA-based devices.

3. PROPOSED APPROXIMATE ADDER ARCHITECTURE:

3.1 Introduction:

The primary aim is to maximize operating efficiency while minimizing the number of Look-Up Tables (LUTs) in Field-Programmable Gate Arrays (FPGAs), reducing the power requirement, speed while maintaining the error to max at 42%. The error of further bits leaving the LSB oscillates between 25 to 37 %, thus signifying that the error is not propagated beyond 37% on further approximation.

This technique helps us to multiply the algorithm for large number of bits while making the error constant and by carry prediction. It is done by wisely splitting the adder into separate pieces, directing resources to where they are most needed.

The adder's architecture is divided into five parts utilizing the segmentation concept from the research articles, the LSB part is having prediction while the MSB part have the accurate calculation. Each of which is designed to carry out a certain function during computation. The first three segments deal with approximation for the Least Significant Bits (LSBs), and the last two segments deal with accurate results for the Most Significant Bits (MSBs).
In adder design, approximation is essential because it enables a trade-off between accuracy and important performance measures like speed, power consumption, and LUT utilization. The adder's estimated portion spans seven bits, sacrificing precision in favor of increased efficiency. This calculated risk allows for large resource savings without sacrificing the adder's overall functioning.

3.2 Working:

Part 1 LSB:

This part contains the AND-OR-OR based adder. As we progress from the bit 0 to bit 2, the carry propagates and the inaccuracy increases from 28% to 42% showing that if we add more bits with this logic then the error will increase and go beyond 50%.

So other techniques need to be taken into consideration. This part does not have carry propagation as it is the Least of LSB part so neglecting the carry part.

Part 2 LSB:

This part contains NAND-NAND based adder. For the bit 3 and bit 4 the sum term is basically the nand of the input bits. From here onwards carry prediction is done for the upcoming subsequent blocks. Here the carry prediction is not based on the output of the nand gate which is sum, but it also depends on the input bits (both bits). This technique is similar to the CLA technique for carry generation, but it is simpler and takes only one logic equations. The accuracy here is 25 % and 37%. The carry generation here have the accuracy of 84%.

Part 3 LSB:

This part contains two blocks, one when then the carry incoming is 0 and other when it is 1. When the carry is 0, then the adder will be like the previous LSB part, it doesn't have any changes. The error also is same that is 25% and 37%. But when Carry is 1, then the adder contains AND-AND gates. This block also provides the same error as before 25% and 37%.

With the above implementation, we can see that the carry is not propagated and it oscillates between 25% and 37%. The carry from this part is the OR of the carry generated from the previous block and the carry generated in this block (with the same logic). Here the OR term is there to compensate for the inaccuracy caused in the LSBs.

Part 2 MSB:

This part contains only one block of the carry look-ahead adder on which the carry input is given from the carry output coming from part 3 LSB.

Part 1 MSB:

This part contains the 4-bit Carry look Ahead Adder, with the input carry coming from the previous block. This is the part of the segment which is doing accurate calculations.

This is only for 12-bit implementation. To incorporate more bits we just need to multiply the Part 2 LSB, Part 3 LSB, and Part 2 MSB such that error will remain within 37%.

Figure 1 shows the Block Diagram of the proposed adder:

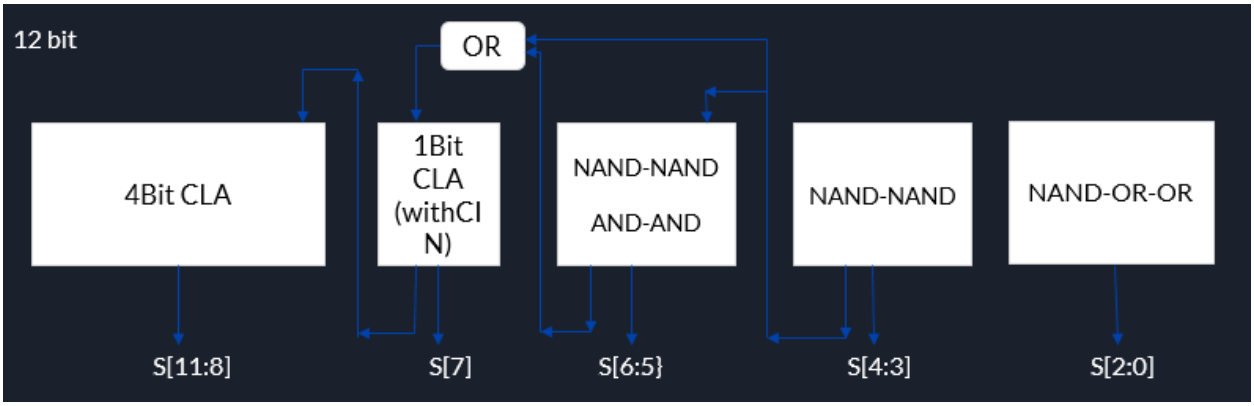


Figure 1

4. IMAGE PROCESSING

Pixel and color depth representations of images are commonly standardized in digital image processing. The tiniest components that go into creating a digital image are called pixels, and they are arranged in a grid to form the overall composition of the image. The binary format in which this information is stored allows each pixel to have a unique color or intensity. The color depth and, thus, the quality of the image are directly impacted by the number of bits utilized to represent each pixel.

Image Pixel Representation

Each pixel in the majority of common digital image formats is represented by a set amount of bits that specify the range of color or intensity values the pixel can accept. For greyscale images, which are frequently utilized in image processing applications, 8 bits per pixel is the most typical bit depth. There are 256 different intensity levels possible with this 8-bit encoding, ranging from 0 for black to 255 for white. 8-bit representation is a commonly used standard for image processing applications such as compression, enhancement, and analysis because it strikes a compromise between image quality and file size.
Three 8-bit values, which stand for the Red, Green, and Blue (RGB) color channels, are commonly used to represent each pixel in color graphics, which together allow for 24-bit color depth or approximately 16.7 million possible colors.

A key concern in image processing is file size, as higher bit depths result in significantly larger files. A typical grayscale image with 512x512 pixels, at 8 bits per pixel, requires a storage size of:

File size = 512 *512 = 262144 bits = 32,768 bytes (approx. 32 KB)

5. UPDATED APPROXIMATE ADDER ARCHITECTURE:

The reduction from 12 bits to 8 bits aligns with the typical standards used in image processing tools and algorithms, ensuring compatibility with widely used software and hardware frameworks, and allowing the results to be easily verified against industry standards. The MATLAB code used in this project to convert images from RGB to grayscale generates 8-bit grayscale images, further reinforcing the necessity of aligning the adder's bit width with the 8-bit pixel format. The decision to use an 8-bit approximate adder in this project is motivated by the standard representation of image pixels, the need for hardware efficiency, and the practical considerations of file size, processing speed, and real-world applicability in image processing tasks.

For most image processing tasks, such as image compression or enhancement, an 8-bit precision is generally sufficient to capture the necessary details without noticeable degradation in quality. Using 12 bits per pixel would provide more precision, allowing for 4,096 intensity levels, but this is often excessive for typical display or processing purposes. The human eye can only differentiate a limited number of shades of gray (around 200-300), making the additional precision of a 12-bit format superfluous for most applications.

So, the designed approximate adder is converted into 8 bits as shown in Figure: 2 truncating the first part of MSB and the first part of LSB from Figure 1.

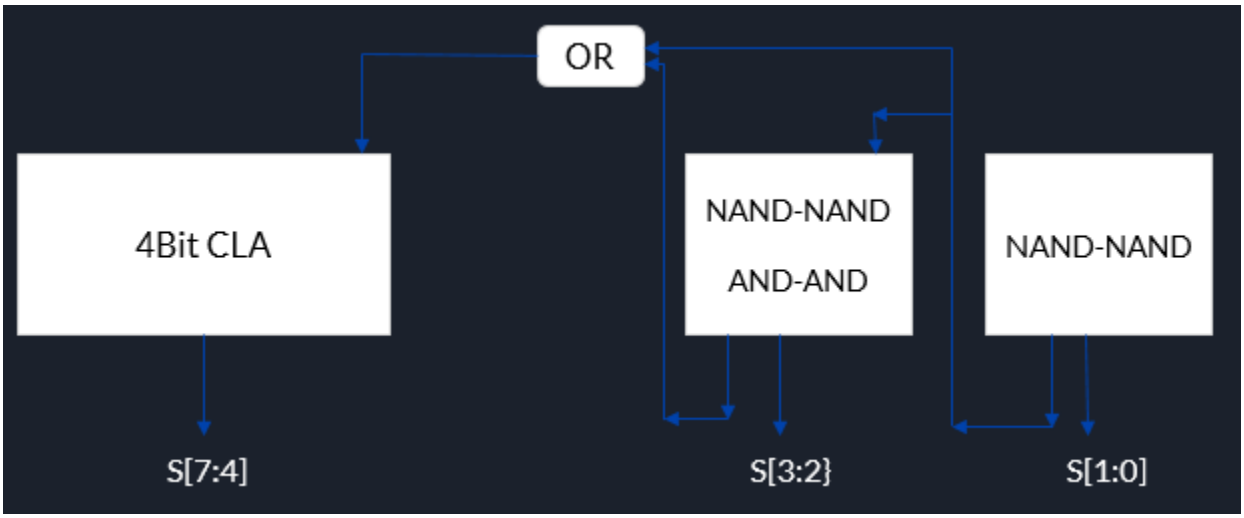
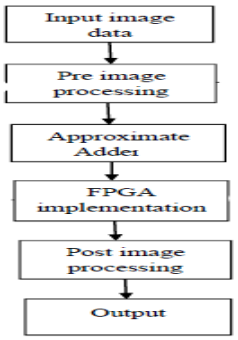


Figure: 2

6. EVALUATION TECHNIQUES:



One of the preliminary steps in the evaluation process of this project is image conversion, from coloured images to Greyscale which is a prerequisite to image processing. The greyscale graphics eliminate the chrominance component which makes it easier to manage the luminance component in order to create emphasis at the correct areas such as edges, textures and gradients. This is why the conversion has been done. The other attribute that is very influential in the processes of image identification and especially in the image processing is Luminance and not colour. For this reason, greyscale image data are more suitable for processing activities such as augmentation, analysis and compression. Greyscale photos make the processing easier but retain all the features that are necessary because they only represent intensity.

The pixel data is transformed into a binary file (.bin format) using MATLAB once the image has been converted to greyscale. The intensity of each pixel in this binary file is represented by an 8-bit integer. This format makes sense because a greyscale image normally only needs 8 bits per pixel, meaning that there are 256 possible intensity levels. The complete image's pixel values are stored in the binary file, readying it for hardware-based processing.

After importing the binary file into Vivado, the hardware design environment is integrated. File-handling techniques in Verilog code manage the picture data. Two greyscale images are processed in this case, and the Verilog test bench calls both of them for further actions. The main arithmetic operation, adding two-pixel values from the source images, is carried out by the approximation adder module, which is positioned at the top of the hardware hierarchy. The relevant pixels from the two photos are added, and the outcome is stored in a new binary file. The Vivado behavioral simulation is used to verify the approximate adder's performance and accuracy in carrying out these tasks.

After the hardware simulation, the binary output is converted back into a visual format using post-image processing. In order to facilitate comparisons between the processed output and the original photos, the binary file that is produced is transformed back into a greyscale image. In this stage, the binary pixel values are translated into appropriate 8-bit unsigned integer fixed-point values that correspond to the final greyscale image's pixel intensities. This guarantees a correct representation of the processed image in a format that can be used for analysis and display.

7. SIMULATIONS AND RESULTS

The Figure 4 below depicts the simulation of the final 8 bit adder code and Table: 1 shows the values.

S.No.	A	B	Sum	Exact sum	Error
1.	11101101	01101101	101010010	101011010	1bit
2.	11011110	11000010	110101101	110100000	3bit
3.	11011110	11011000	110110111	110110110	1bit
4.	11101000	10110011	110100011	110011011	2bit

Table: 1

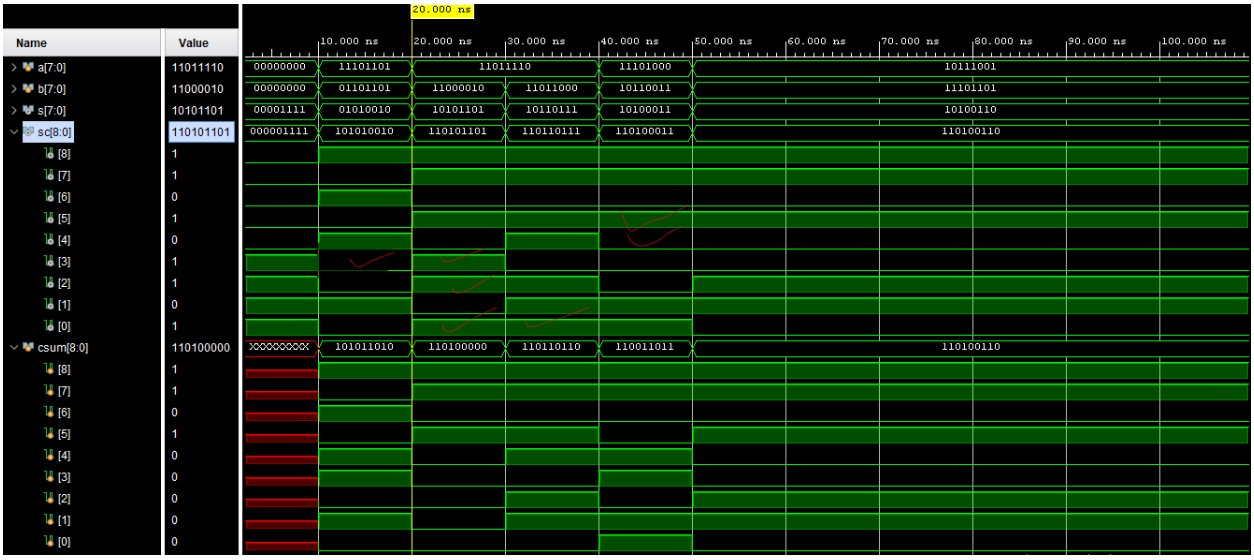


Figure: 4

Figure 5 and 6 is the two input image which is converted into greyscale which is shown by Figure 7 and 8. These images are input in the matlab codes for generating bin file and after running, its bin file is created

Figure: 5

Figure: 6

Figure: 7 Figure: 8

Two image file's bin (Figure: 9 and Figure: 10) details is given in the input in the testbench of the approximate adder and the resultant bin file is generated (Figure: 12) . Figure : 11 shows the bin file of the exact adder.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	21	21	21	21	20	20	20	20	1E	1E	1E	1D	1D	1D	1D	1D
00000010	1C	1C	1C	1C	1C	1C	1C	1C	1C	1C	1C	1D	1D	1D	1D	1D
00000020	1C	1C	1D	1D	1D	1E	1E	1E	1F	1F	1F	1F	1F	1F	1F	1F
00000030	20	20	21	21	21	21	20	20	21	21	21	21	20	20	20	20
00000040	20	20	20	20	20	20	20	20	20	20	20	21	21	22	22	22
00000050	22	22	22	22	22	23	23	23	22	22	22	22	22	21	21	21
00000060	21	21	21	21	21	21	21	21	22	21	20	20	21	22	22	21
00000070	21	21	22	22	22	22	22	22	22	22	21	21	21	21	22	22
00000080	22	22	23	23	23	23	23	23	23	24	24	25	25	26	25	25
00000090	24	24	24	25	25	25	25	25	26	26	27	27	27	27	26	26

Figure: 9

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	26	23	20	22	27	2B	29	24	23	21	2F	6D	75	3F	28	1D
00000010	23	24	29	27	2F	3C	4B	67	7F	84	6E	54	3F	2F	2C	28
00000020	27	30	28	23	25	31	43	42	2D	2F	35	2B	23	24	21	26
00000030	22	26	27	23	20	28	25	61	8A	C9	FC	DB	7F	3A	22	1C
00000040	1D	20	26	2D	28	1D	2B	4B	65	36	23	2A	24	1F	25	25
00000050	1E	24	25	21	2B	3B	5E	9C	85	58	35	20	1C	24	21	1F
00000060	1C	1F	23	24	24	24	24	24	22	24	23	20	23	2C	34	38
00000070	35	39	5A	55	30	2D	34	2A	32	32	34	38	3C	3D	40	43
00000080	3F	38	42	3B	45	3D	52	42	3E	41	43	40	3B	38	3C	42
00000090	3E	3F	40	40	3F	3F	3F	40	43	43	47	47	3E	3A	3A	36

Figure: 10

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	FF	47	44	41	43	47	4B	49	44	41	3F	4D	8A	92	5C	45
00000010	3A	3F	40	45	43	4B	58	67	83	9B	A0	8A	71	5C	4C	49
00000020	45	43	4C	45	40	42	4F	61	60	4C	4E	54	4A	42	43	40
00000030	45	42	46	48	44	41	49	45	81	AB	EA	1D	FC	9F	5A	42
00000040	3C	3D	40	46	4D	48	3D	4B	6B	85	56	43	4B	45	41	47
00000050	47	40	46	47	43	4D	5E	81	BF	A7	7A	57	42	3E	45	42
00000060	40	3D	40	44	45	45	45	45	45	44	45	43	40	44	4E	56
00000070	59	56	5A	7C	77	52	4F	56	4C	54	54	55	59	5D	5E	62
00000080	65	61	5A	65	5E	68	60	75	65	61	65	67	65	60	5E	61
00000090	67	62	63	64	65	64	64	64	65	69	69	6E	6E	65	61	60

Figure: 11

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	FF	4F	52	4F	4F	53	53	4F	4F	41	4F	4D	82	9A	5E	47
00000010	32	43	4B	47	47	4F	53	6B	87	9F	AB	83	7B	5E	4E	43
00000020	47	47	5F	47	42	4A	5F	61	6D	4E	4C	56	48	40	47	42
00000030	45	4F	4F	52	52	4F	4F	4F	8F	AF	EE	2F	02	A3	5F	4F
00000040	4F	4F	4F	4F	5F	4F	4F	53	73	8F	5F	53	4F	4F	41	4F
00000050	4F	4D	4F	4F	4F	51	60	81	C3	AF	7F	5F	4F	4F	4F	4E
00000060	42	4F	42	52	4F	4F	4F	4F	4F	4D	4F	53	4F	52	5F	5F
00000070	5F	5E	5E	7D	7F	5F	5F	5F	4D	5D	5D	5F	5F	6F	6E	6F
00000080	71	61	5F	71	60	72	62	81	71	61	6F	73	6F	62	5F	6B
00000090	6F	6B	67	6F	6F	66	66	66	6F	71	71	74	74	65	61	5D

Figure: 12

Figure: 13

Figure: 14

8. REFERENCES:

1. https://ieeexplore.ieee.org/document/9524629

2. https://www.jstage.jst.go.jp/article/elex/16/6/16_16.20190043/_article

3. https://ieeexplore.ieee.org/document/9521486

4. https://ieeexplore.ieee.org/document/9424329

5. https://www.researchgate.net/publication/224110766_An_enhanced_low-power_high-speed_Adder_For_Error-Tolerant_application

6. https://dl.acm.org/doi/10.1109/TVLSI.2014.2355217

7. https://ieeexplore.ieee.org/document/8419308/

8.https://www.researchgate.net/publication/301700140_High_Level_FPGA_Modeling_for_Image_Processing_Algorithms_Using_Xilinx_System_Generator.

9. https://in.mathworks.com/videos/matlab-to-fpga-in-5-steps-1625206482934.html