Chair for Integrated Systems
TUM School of Computation, Information and Technology
Technical University of Munich

Investigation and Implementation of Approximate Dividers for FPGA

Kiran Krishnadas Bhandarkar

Professor:
Prof. Dr.-Ing. Walter Stechele

Supervisor:
Arne Kreddig, M.Sc.

Submitted:
Munich, 22.02.2023

I hereby declare that this research internship (forschungspraktikum) report is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, 22.02.2023 Kiran Krishnadas Bhandarkar

Abstract

Approximate computing is an emerging design paradigm for realizing high-performance, resource optimized, and energy-efficient circuits and systems. However, approximate computing design methods involve a trade-off between certain inaccuracies of the calculations and reducing the overall resource consumption of the implemented design. Recent research focuses on approximating arithmetic units, such as adders, subtractors, multipliers, and dividers. While adders and multipliers are extensively used and investigated, divider designs are only partially explored.

This report investigates approximate dividers suitable for implementation on FPGA. This work involved a study of various approximate divider designs, comparing performance, and selection of the most promising technique. The selected design is a fixed point divider based on restoring array architecture that delivers substantial hardware savings while maintaining high accuracy. Compared to an exact design, the proposed dividers have lowered resource utilization and latency by **7.7%** and **17%**, respectively, while delivering an average accuracy of $\approx$ **80%**. Furthermore, the trade-off between accuracy and performance is studied for various configurations to determine the best compromise.

Contents

Abstract **iii**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Applications of Approximate Dividers . . . 1
1.3 Evaluation Metrics . . . 2
1.3.1 Error Characteristics . . . 2
1.3.2 Circuit Characteristics . . . 2
1.4 Organisation . . . 2

2 Literature Survey **3**
2.1 Approximate Divider Design Approaches . . . 3
2.1.1 Logarithmic Exponent Approximate Divider (LEAD) . . . 3
2.1.2 Divider Based on Piecewise Constant Approximation . . . 3
2.1.3 Approximate Restoring Dividers Using Inexact Cells . . . 4
2.1.4 Scalable Accuracy Approximate Divider with Error Compensation (SAADI-EC) . . . 4
2.1.5 A Multistage Approximation Floating Point Dividers (FPAD) . . . 4
2.1.6 Truncation-based Approximate Divider (TruncApp) . . . 4
2.1.7 High-Speed Rounding based Approximate Divider (SEERAD) . . . 4
2.1.8 Approximate Divider based on piecewise linear approximation . . . 5
2.1.9 Configurable Approximate Divider for Energy Efficiency (CADE) . . . 5
2.2 Approximate Divider Designs Normalised Summary . . . 5

3 Design Evaluation **6**
3.1 Approximate Restoring Dividers Using Inexact Cells . . . 6
3.2 Approximate Divider based on piecewise linear approximation . . . 7
3.3 Performance Comparison . . . 9

4 Experimental Results **11**
4.1 HDL Simulation Results . . . 11
4.2 HDL Synthesis Results . . . 15

5 Conclusion and Outlook **17**

List of Figures **18**

List of Tables **19**

Bibliography **20**

1 Introduction

Advancements in research in artificial intelligence and big data processing have resulted in applications that need complex data computations. The processors and ASICs that perform these massive computations require high energy and many resources. These requirements pose a challenge for already resource and power-constrained designs. Although the computations are complex, they do not need a very high amount of accuracy. Minimal errors in computations do not significantly impact the results. Approximate computing allows leveraging the current CMOS technology to design energy and resource-efficient designs with high performance, albeit with minimal errors. This strategy extends the current CMOS technology life to many more years and, at the same time, conserves the hardware metrics, including power, delay, and area with a compromise in accuracy [1].

1.1 Motivation

Moore's law states that the number of transistors on a chip doubles every 18-24 months. This governing principle of CMOS chip development in the past few decades has resulted in continuous improvements in performance, energy efficiency, area, and resource utilization. However, further reduction in operating voltage in chips manufactured at the nanometer scale is highly challenging due to increased power density. Hence investigating new approaches to chip design, such as parallel computing (multi-core) architectures and approximate computing is essential [2].

The approximate computing paradigm's basic idea is that most common applications, such as multimedia processing (video streaming and image processing) or machine learning programs, are tolerant of some errors. For example, video streaming errors may degrade the video's quality. However, this degradation is not perceived by humans due to physical limitations. Furthermore, the input data is usually noisy and quantized in machine learning applications. Hence, most algorithms are probability-based, which inherently can tolerate some errors. Thus, it is beneficial to use approximate computing circuits for such applications.

Hardware realization of the common arithmetic operations suits the approximate computing paradigm. Approximate adders and multipliers are deeply researched as these are the most common operations. However, division operation is rarely studied since it is a resource-intensive computing operation. Thus, only a few techniques are available for approximate dividers. This work explores approximate dividers, which can serve as reliable replacements for exact dividers.

1.2 Applications of Approximate Dividers

Approximate dividers can be considered for basic image processing applications such as smoothing, sharpening, and compression. In machine learning applications, approximate dividers can be used for division operations to reduce delay and save energy. Designs with various error and circuit characteristics can be deployed in re-configurable systems to enhance reconfiguration flexibility. In digital processing applications, approximate dividers can achieve higher speeds and power efficiency. Finally, approximate arithmetic circuits, including dividers, can be efficiently utilized in deep neural network (DNN) based applications such as face detection and alignment.

1.3 Evaluation Metrics

For the evaluation of various approaches, circuit characteristics and error characteristics are considered.

1.3.1 Error Characteristics

The following error metrics [2] have been used to assess the error characteristics of the design:

- **Error Rate (ER):** Error rate indicates the probability that an incorrect result is generated.
- **Error Distance (ED):** There are two ways to calculate error distance. First, ED is the arithmetic difference between an exact output (M) and an inexact output (M') as given in eq. 1.1.

$$ED = M' - M \tag{1.1}$$

  Second, ED can be the hamming distance between exact output and an inexact output. In this work, the first method is used to calculate ED.

- **Mean Relative Error Distance (MRED):** Relative error distance (RED) gives the relative difference to correct output. The mean of RED is given by MRED, which is calculated using eq. 1.2.

$$MRED = Mean\left(\frac{M' - M}{max(M)}\right) \tag{1.2}$$

- **Normalized Mean Error Distance (NMED):** Mean of the error distances is called Mean Error Distance (MED). NMED as given in eq. 1.3 is the normalization of MED using the largest exact output. This metric helps compare designs with different sizes.

$$NMED = Mean\left(\frac{M' - M}{max(M' - M)}\right) \tag{1.3}$$

1.3.2 Circuit Characteristics

In this work, the primary circuit metrics such as **critical path delay**, **power dissipation**, and **area** are considered. Some other compound metrics, such as the power-delay product (PDP), area-delay product (ADP), and energy-delay product (EDP), also exist. However, for simplicity, these metrics are not used in this study.

1.4 Organisation

The organization of this report is as follows: Chapter 2 gives an overview of different approximate divider designs and a comparison between them based on normalized evaluation metrics. Next, in chapter 3, two selected designs are realized and evaluated using software (MATLAB). Next, chapter 4 includes HDL simulation and synthesis results of the selected design. Finally, chapter 5 gives the conclusion and outlook of this work, followed by references.

2 Literature Survey

Approximate circuits can be implemented using various techniques. This chapter overviews different approaches to implementing fixed- and floating-point approximate dividers. A comparison of these approaches based on standard evaluation metrics is also covered.

2.1 Approximate Divider Design Approaches

2.1.1 Logarithmic Exponent Approximate Divider (LEAD)

A logarithmic divider approximates the division operation into add and multiply operations. LEAD approach proposed in [3] is an improvement over the traditional logarithmic divider. It improves the accuracy by using exponent expansion series for approximation. In traditional logarithmic divider, for two numbers A = $2^{e_a}(1+a)$ and B = $2^{e_b}(1+b)$ the quotient is given by

$$Q = \begin{cases} 2^{(e_a - e_b)} \times (1 + a - b), & \text{if } a - b > 0 \\ 2^{(e_a - e_b - 1)} \times (2 + a - b), & \text{otherwise} \end{cases}$$

Where a and b are mantissa that lies in the range [0,1] and e_a and e_b are the positions of leading ones. In LEAD, the quotient (Q) is given by:

$$2^x = 1 \pm (xln2) + (xln2)^2 \times 0.5 \pm (xln2)^3 \times 0.1667 + ...$$

$$Q = 2^{(e_a - e_b)} \times (1 \pm (0.693(a-b)) + .5 \times (0.693(a-b))^2) + ...$$

The proposed divider significantly reduces error rates over the traditional logarithmic divider.

2.1.2 Divider Based on Piecewise Constant Approximation

This approach presented in [4] is an improvement over Mitchell's Approximation Logarithmic Divider. The basic idea is to transform division into several addition operations. For numbers A and B, the quotient is given by

$$Q = \begin{cases} (1 + A_M)/(1 + B_M), & \text{if } A_M \geq B_M \\ 2(1 + A_M)/(1 + B_M), & \text{otherwise} \end{cases}$$

As per Mitchell's approximation $lg(1+x) = x$. This results in large errors. However, in the piecewise constant approximation method

$$lg(1+x) = \begin{cases} 2x, & \text{if } x \in [-0.5, 0) \\ x, & \text{if } x \in [0, 1] \end{cases}$$

Hence, this approach significantly improved over Mitchell's approach with better error metrics with simple hardware implementation using Look-Up-Table (LUT), multipliers and adders.

2.1.3 Approximate Restoring Dividers Using Inexact Cells

[5] Restoring division involves division of $2k$-bit dividend z by a k-bit divisor d to produce a k-bit quotient q and a k-bit reminder r. In this approach, division operation is converted into shift and subtract operations. Hardware implementation of this logic is achieved using an array divider structure. A $2k/k$ division requires k rows, each containing k restoring divider cells and a single OR-gate. An exact restoring divider cell (EXRDC) uses a full subtractor and a multiplexer. In this approach, this exact cell is replaced by an inexact cell (AXRDC) which utilizes fewer resources albeit introduces inaccuracies. It reduces path delay and requires less area than an exact divider. More details of this approach are presented in section 3.1 of chapter 3.

2.1.4 Scalable Accuracy Approximate Divider with Error Compensation (SAADI-EC)

The approach in [6] is a design improvement over multiplicative adders where the divisor is approximated using the Taylor series, and the error is compensated using a pre-computed factor based on the order of the quotient coefficient. For two numbers A = $2^{e_a} \times a$ and B = $2^{e_b} \times b$ the quotient Q is given by:

$$Q = 2^{e_a - e_b} \times \frac{a}{b}$$

$$Q = 2^{e_a - e_b} \times a \times R(b)$$

where $R(b)$ = $1/b$. In this approach, $R(b)$ is determined using Taylor Series approximation given by

$$\sum_{i=0}^{t} |x|^i = 1 + |x| + |x|^2 + |x|^3 + ... + |x|^i$$

$$R_t(b) = 1 + |x| + |x|^2 + |x|^3 + ... + |x|^t$$

The error compensation is tuned based on the order of quotient coefficient (t). This approach converts division to shift and multiplication operations with high accuracy.

2.1.5 A Multistage Approximation Floating Point Dividers (FPAD)

The approach presented in [7] is based on Newton Raphson Method (Fast Division). Here multistage mathematical approximations are used to improve power consumption and critical path delay. There are two stages of approximations involved in this design. First, the number of bits in the mantissa is truncated to reduce complexity. Second, division operation is approximated to the sum of powers of two. Hence, division operation is now converted to addition and shift operations.

2.1.6 Truncation-based Approximate Divider (TruncApp)

The approach proposed in [8] converts the division operation into inverse and multiply operations. Also, in this approach, the inputs are truncated and used as inputs to the inversion and multiplication blocks. This approach reduces area requirements and power consumption.

2.1.7 High-Speed Rounding based Approximate Divider (SEERAD)

The technique presented in [9] is similar to the TruncApp approach. However, the divisor is rounded off to 2^n form in this design. Thereby, the division operation is approximated to shift and multiply operations. As a result, the SEERAD design operates at high speed with reduced area requirements and power consumption. Furthermore, the divider's accuracy can be tuned based on the rounding-off accuracy.

2.1.8 Approximate Divider based on piecewise linear approximation

The design presented in [10] is an improvement over the SEERAD design. Accuracy is improved by using a rounding function based on linear piecewise approximation. For two unsigned integers A = $(1 + f_a) \times 2^k_a$ and B = $(1 + f_b) \times 2^k_b$ the approximate quotient Q is given by

$$Q = (1 + f_a) \times (\alpha \times f_b + \beta) \times 2^{k_a - k_b}$$

where α and β are pre-computed values using linear piece wise approximation. This approach is covered in detail in section 3.2 of chapter 3.

2.1.9 Configurable Approximate Divider for Energy Efficiency (CADE)

The approach presented in [11] proposes a run-time tunable floating-point approximate divider. This design converts the division operation into a subtraction of mantissa and exponent of the dividend and the divisor. Significant errors are offset using a set of pre-computed compensation values. The hardware implementation requires a LUT, basic gates, and subtractors.

2.2 Approximate Divider Designs Normalised Summary

Table 2.1 summarizes all the approximate divider designs considered for this study. While all designs have pros and cons, this work aims to investigate and implement fixed-point (integer) designs. Hence, design [5] and [10] are considered. While design [5] is area and power-efficient, design [10] has the least path delay.

Table 2.1 Approximate Dividers Designs Normalized Summary

Approximate Divider	**Ref.**	**Type**	**Area** (μm^2)	**Power** (mW)	**Path Delay** (ns)
Logarithmic Exponent Approximate Divider (LEAD)	[3]	Fixed	5482.04	0.73	9.91
Divider Based on Piecewise Constant Approximation	[4]	Float	212.04	0.32	3.82
Approximate Restoring Dividers Using Inexact Cells	[5]	Fixed	304.35	0.19	4.05
Scalable Accuracy Approx. Divider with Error Compensation (SAADI-EC)	[6]	Fixed	1973	0.36	2.12
A Multistage Approximation Floating Point Dividers (FPAD)	[7]	Float	749.07	0.03	0.25
Truncation-based Approximate Divider (TruncApp)	[8]	Fixed	1483	0.80	1.05
High Speed Rounding based Approximate Divider (SEERAD)	[9]	Fixed	1343	0.64	0.61
Approximate Divider based on piecewise linear approximation	[10]	Fixed	1829	0.86	1.25
Configurable Approximate Divider for Energy Efficiency (CADE)	[11]	Float	NA	NA	0.51

3 Design Evaluation

This chapter contains the design details and software-based performance analysis of array-based and linear piecewise approximation-based dividers. MATLAB software is used to realize and evaluate various designs.

3.1 Approximate Restoring Dividers Using Inexact Cells

Restoring division is one of the most commonly used techniques to realize division operation in hardware. It involves repeated subtraction of divisor from shifted partial reminder where a negative partial reminder results in restoration of the previous value. Based on the outcome of the subtraction of the divisor from the partial reminder, the quotient bits are set or cleared [5].

Usually, the restoring algorithm is realized using an array-based structure. Fig.3.2 shows design of a $8/4$ exact restoring array divider. For example, a $2k/k$ division requires an array of k rows, with k restoring divider cells and an $OR-$gate in each row. An exact divider cell consists of a full subtractor and a multiplexer. Fig.3.1 (left) depicts an exact divider cell. The outputs of the exact divider cell are given by:

$$r = q_s d + \bar{q_s} x, b_{out} = \overline{x \oplus y} \cdot b_{in} + \bar{x} y$$

The approximate design proposed in [5] requires the replacement of some of the exact divider cells with a logically simplified inexact cell. Fig.3.1(right) shows an inexact divider cell. The output of the inexact cell is given by:

$$r = q_s \oplus x, b_{out} = \bar{x}$$

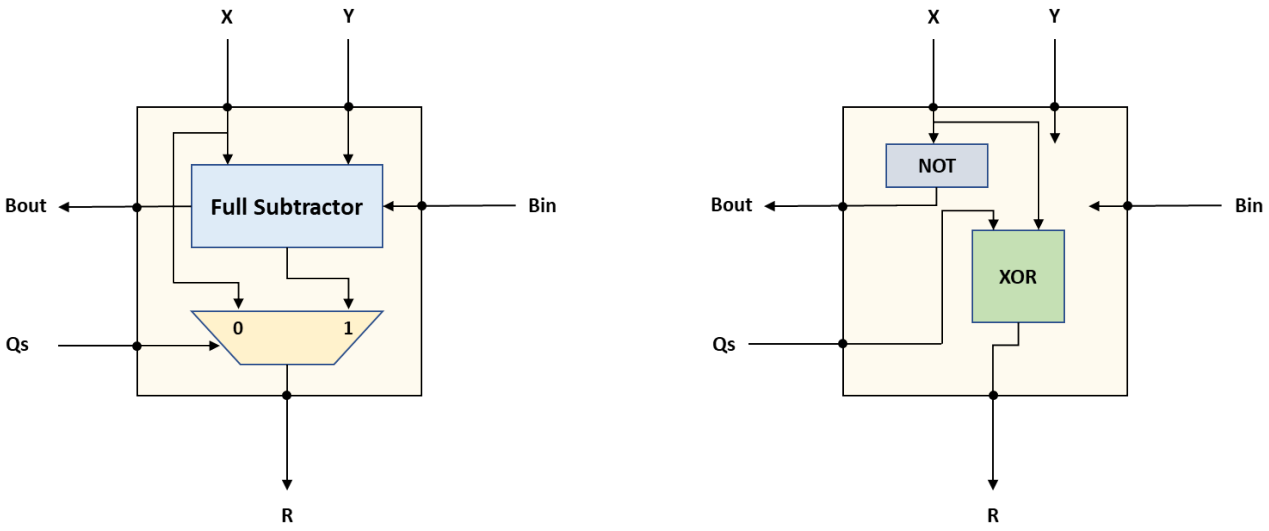


Figure 3.1 Exact restoring divider cell (EXRDC) (left) and approximate divider cell (AXRDC)(right)

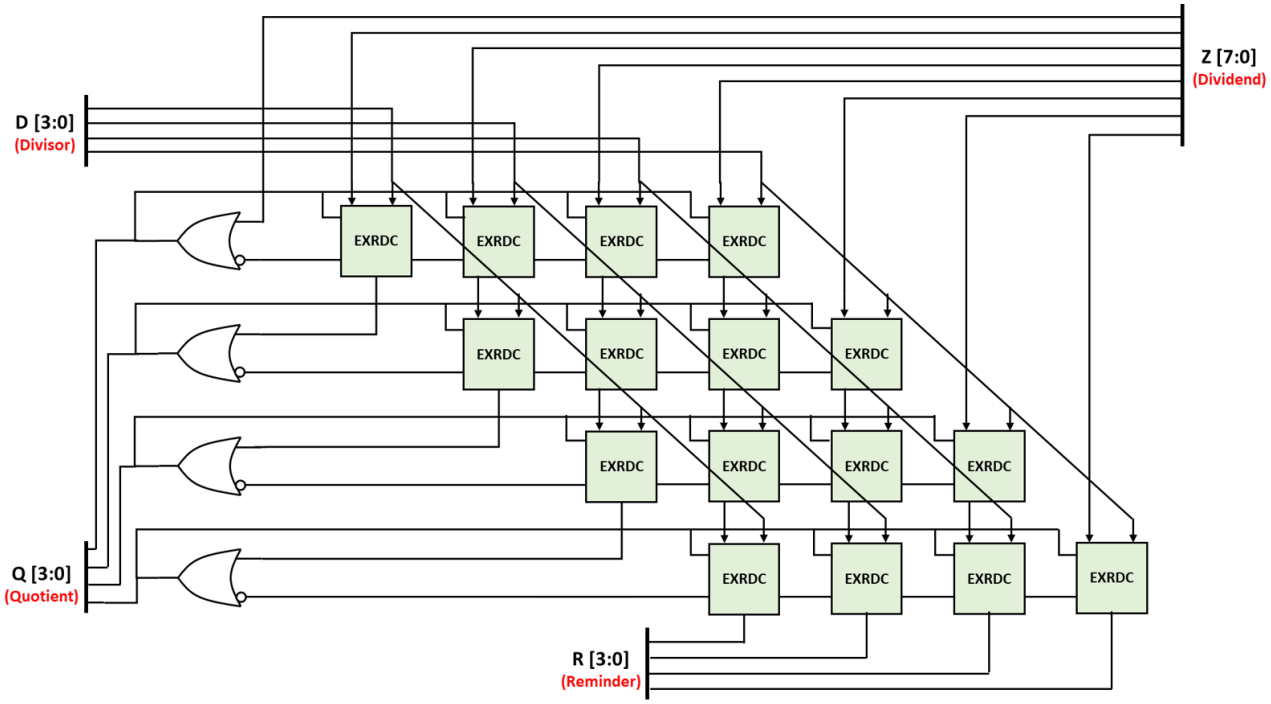


Figure 3.2 Architecture of $8/4$ array divider

The increase in hardware savings and decrease in accuracy is directly proportional to the number of exact and inexact cells in the array. The approximation/tuning parameter p refers to the magnitude of the approximation in the design. The number of the columns (starting from the LSB) of the array whose cells are replaced with inexact cells is expressed by p. For a $2k/k$ division with tuning factor p, the number of approximated cells N is given by:

$$N = \begin{cases} \frac{p(p+1)}{2}, & \text{if } p \leq k \\ k(2p-k+1) - \frac{p(p+1)}{2}, & \text{if } p > k \end{cases}$$

3.2 Approximate Divider based on piecewise linear approximation

Curve fitting-based techniques have recently been one of the most widely researched approximate computing methods. The linear approximation method mentioned in [10] is also based on curve fitting. Contrary to other curve-fitting-based designs, this design reduces the need to store many curve-fitting coefficients. The division operation complexity is further reduced by using simple multiplication and shift blocks along with an inverse block (to calculate the reciprocal of the divisor). An n-bit unsigned integer can be represented as:

$$N = F \times 2^k = (1+f) \times 2^k$$

where f is the fractional part of F and k denotes the position of the leading one in N. For two unsigned numbers, A and B, the exact division quotient (Q) is given by:

$$Q = \frac{F_A \times 2_A^k}{F_B \times 2_B^k} = F_A \times \left(\frac{1}{F_B}\right) \times 2^{k_A - k_B} = F_A \times X_B \times 2^{k_A - k_B}$$

The X_B term is approximated using linear piecewise approximation technique. Therefore, approximate Q is given by:

$$Q = (1 + f_A) \times (\alpha \times f_b + \beta) \times 2^{k_A - k_B}$$

The α and β values are to be computed and stored in a LUT.

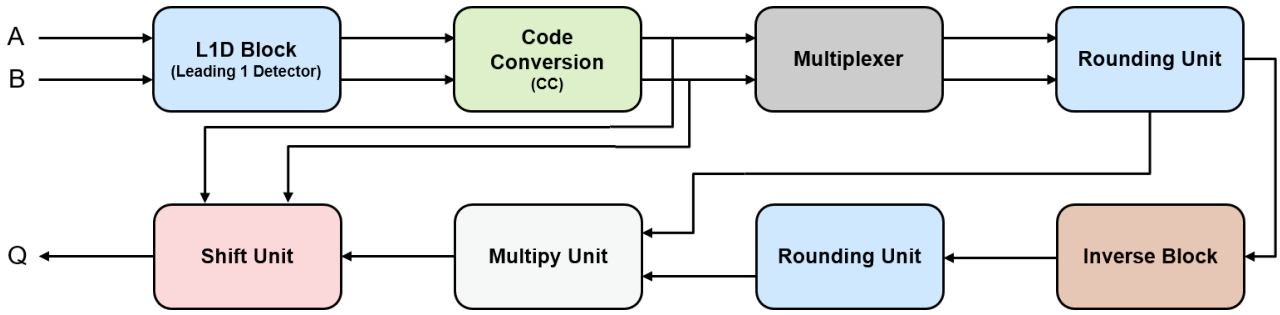


Figure 3.3 Block Diagram of the piecewise linear approximation divider [10]

The block diagram redrawn from [10] for the hardware implementation of the method is shown in fig. 3.3. The Leading one block takes an n-bit dividend and divisor and outputs the position of the leading one. The output size of L1D detector is same as the input. For example, if the input for the L1D detector is (11001100), its output will be (10000000). Next, the code conversion block converts this number to the $log_2(n)$ form i.e., (111) (7^{th} bit position). Next, the CC output is fed to the multiplexer to get the fractional part of the input i.e., (10011000). The rounding blocks help consider only the most significant parts of the inputs. Hence they are used to reduce the complexity of multiplication operation. Next, the rounded divisor is fed into the the inverse block. Inverse block consists of a look-up table containing the pre-computed α andβ values. Next, the output from the inverse block is further rounded and fed to the multiplier block. The multiplier used can be an exact or an inexact design. Finally, the output from the multiplier is shifted based on the position of the leading ones of the dividend and the divisor. The output of the shift unit is the required approximate quotient.

Pre-computing α and β is vital for this technique. The $1/1 + f_b$ curve is divided into linear curves. The values of α, β are the slope and the intercept of the linear curve respectively. The accuracy of the divider increases with the number of linear sub-intervals. Thus, number of sub-intervals r is the tuning parameter for this design. Fig. 3.4 depicts the curve fitting done in MATLAB for different tuning values (r) and the corresponding values of α and β are presented in table 3.1.

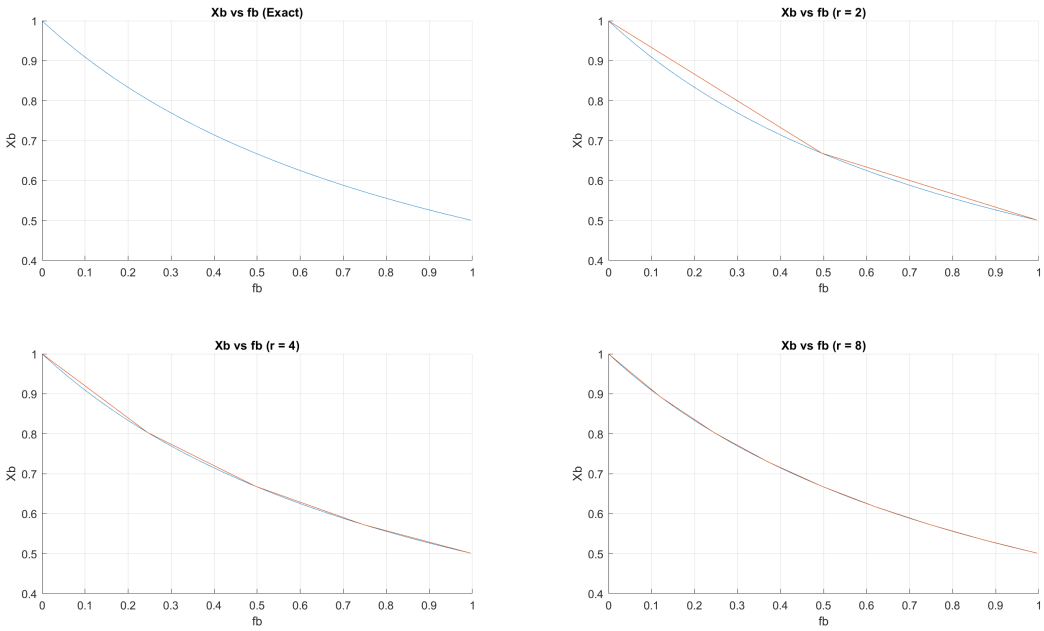


Figure 3.4 X_b vs f_b curves for various tuning factors (r)

Table 3.1 The α and β values calculated using MATLAB for different rounding factors (r)

r = 2			r = 4			r = 8		
f_b	α	β	f_b	α	β	f_b	α	β
0.000, 0.496	-0.668	1	0.000, 0.246	-0.802	1	0.000, 0.121	-0.892	1
0.496, 0.996	-0.334	0.834	0.246, 0.496	-0.536	0.934	0.121, 0.246	-0.715	0.978
			0.496, 0.746	-0.382	0.858	0.246, 0.371	-0.585	0.946
			0.746, 0.996	-0.286	0.786	0.371, 0.496	-0.487	0.910
						0.496, 0.621	-0.412	0.873
						0.621, 0.746	-0.353	0.836
						0.746, 0.871	-0.306	0.801
						0.871, 0.996	-0.267	0.767

3.3 Performance Comparison

This section summarizes the performance of the two selected approximate divider approaches. The performance evaluation of an approximate design is performed against an exact divider. The logic verification metrics mentioned in section 1.3 characterize the designs. It is worth noting that this is pure logic evaluation using software (MATLAB). Fig. 3.5 gives a block diagram view of the logic verification strategy.

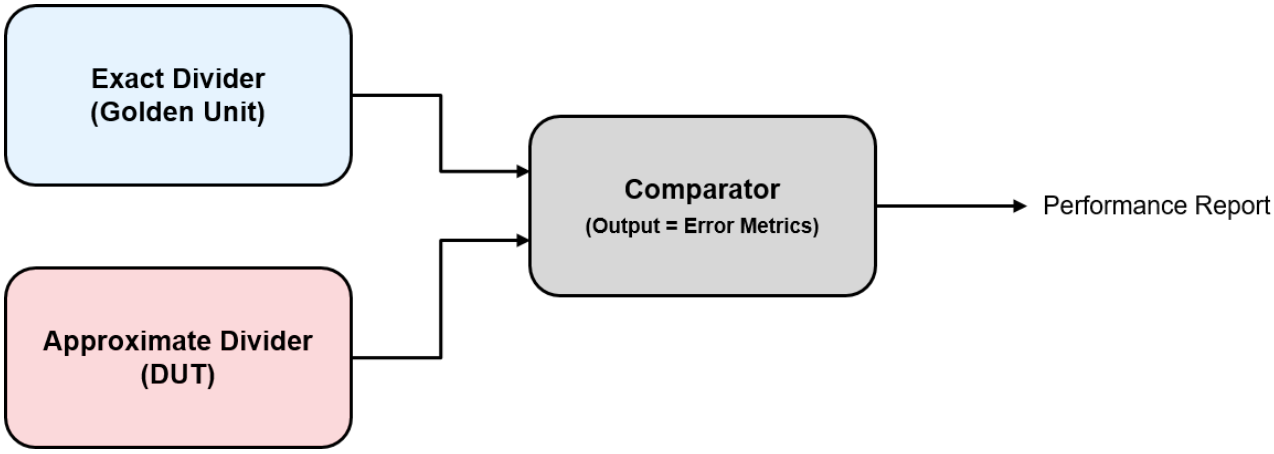


Figure 3.5 Design Verification Strategy Block Diagram

Table 3.2 provides the performance summary of the restoring array-based approximate divider. The evaluation is done for 16/8 dividers with tuning parameter (p) = 4, 6, and 8, respectively. The circuit characteristics are leveraged from [5]. It is conclusive that the error rate (ER) magnitude is proportional to the value of p. Therefore, a designer can select a value for p based on the end application tolerance. For example, p = 6 is a reasonable configuration for most image-processing applications.

Table 3.2 MATLAB Simulation Summary - Array Based Inexact Divider

(p)	**ER (%)**	**MRED**	**NMED**	**ED Max**	**Area** (μm^2)	**Power** (mW)	**Path Delay** (ns)
4 (EX:54, IEX:10)	5.07	$0.02e^{-02}$	0.0073	7	304.35	0.19	4.05
6 (EX:41, IEX:21)	23.40	$0.10e^{-02}$	0.0083	31			
8 (EX:28, IEX:36)	66.24	$0.45e^{-02}$	0.0091	127			

The performance of the linear piecewise approximate divider is summarized in Table 3.3. As proposed in [10], the error rate reduces as the tuning factor (r) increases from 2 to 8. The area characteristics in the table are leveraged from [10].

Table 3.3 MATLAB Simulation Summary - Linear Piecewise Approximate Divider

r	**ER (%)**	**MRED**	**NMED**	**ED Max**	**Area** (μm^2)	**Power** (mW)	**Path Delay** (ns)
2	77.80	0.0309	0.0468	85	1829	0.86	1.25
4	70.50	0.0230	0.0349	85			
8	68.10	0.0228	0.0345	85			

The array divider is preferred among the two selected integer dividers since it has superior circuit characteristics (area and power) and better error characteristics. Hence, array divider HDL implementation is done as a part of this project. The HDL implementation details are covered in Chapter 4.

4 Experimental Results

This chapter covers the HDL Simulation and synthesis results of the restoring array-based approximate divider.

4.1 HDL Simulation Results

The divider simulation is done in ModelSim software. A MATLAB script generates the simulation inputs and reference outputs, and the HDL model results are captured. Finally, the generated and reference outputs are compared, and the design is functionally verified.

Two versions of the divider are realized as a part of this project. The first version is a nonpipelined divider, and the second is a pipelined version. Fig. 4.1 and Fig. 4.3 show the design of the 16/8 nonpipelined array dividers with tuning factors p = 6 and 8, followed by their output waveforms in Fig. 4.2 and Fig. 4.4, respectively. Similarly, Fig. 4.5 and Fig. 4.7 show the design of the 16/8 pipelined array dividers with tuning factor p = 6 and 8 followed by their output waveforms in Fig. 4.6 and Fig. 4.8.

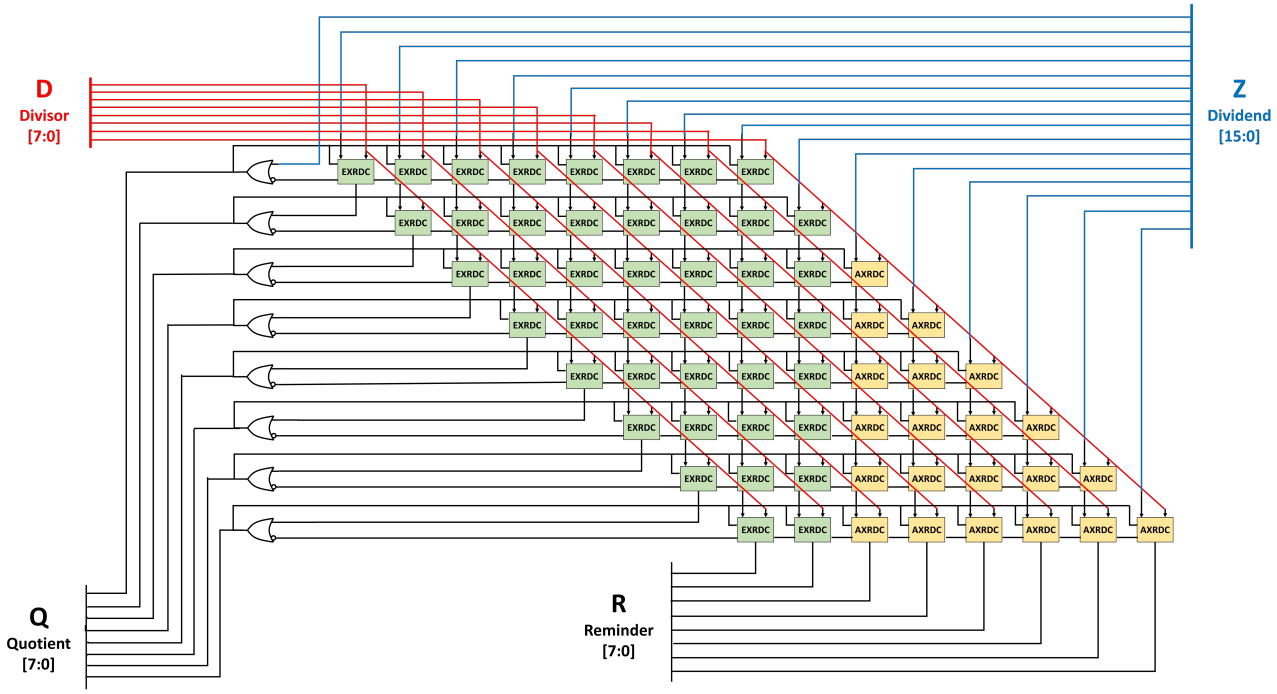


Figure 4.1 Architecture of 16/8 approximate non-pipelined array divider with p=6

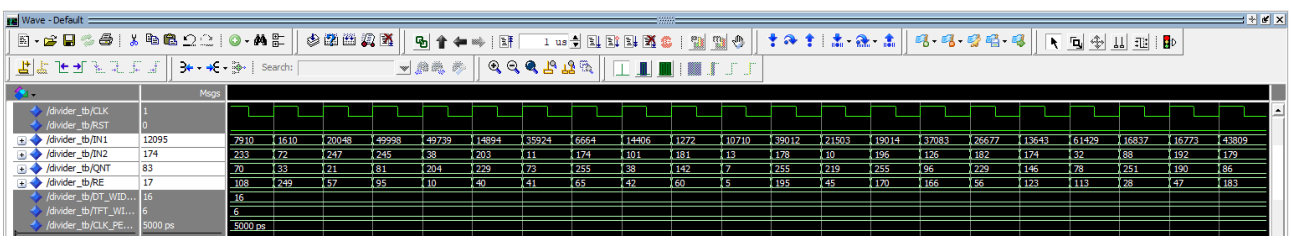


Figure 4.2 Simulation result of 16/8 approximate non-pipelined array divider with p=6

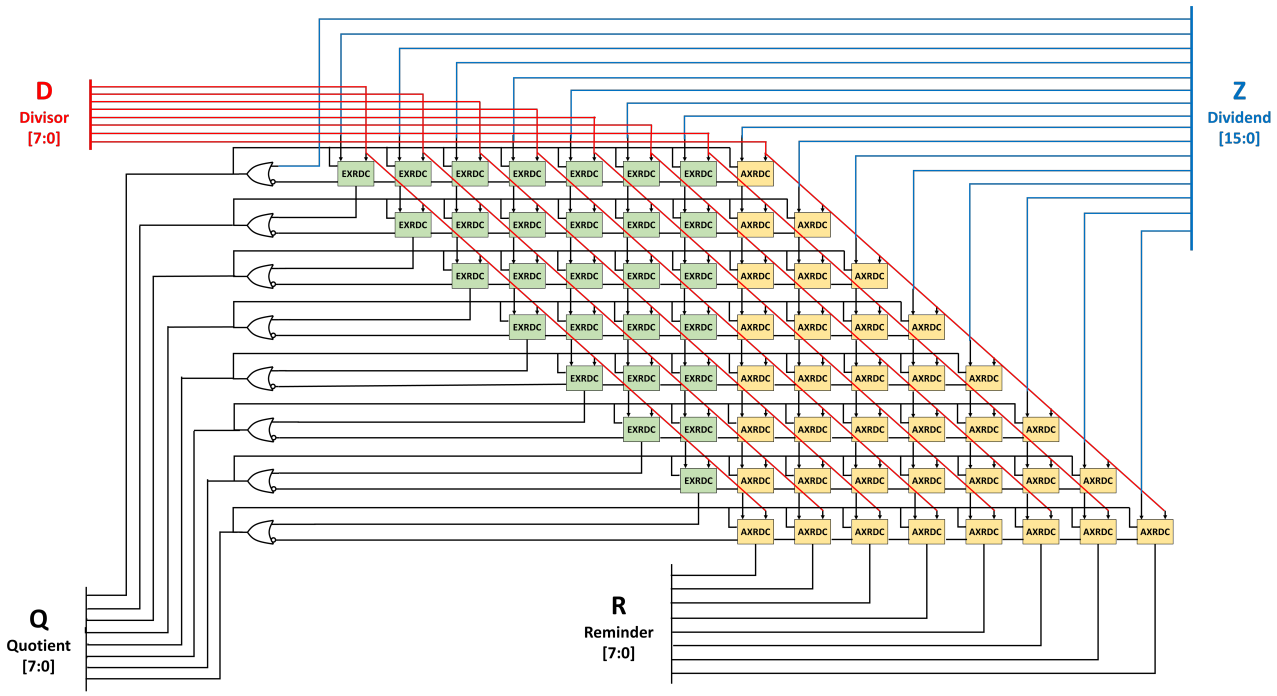


Figure 4.3 Architecture of 16/8 approximate non-pipelined array divider with p=8

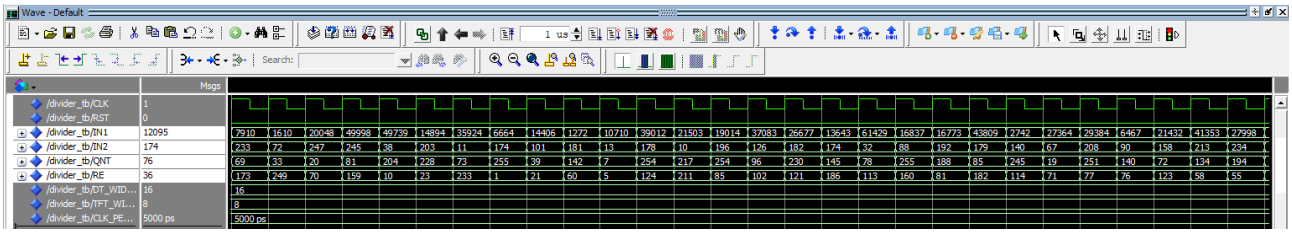


Figure 4.4 Simulation result of 16/8 approximate non-pipelined array divider with p=8

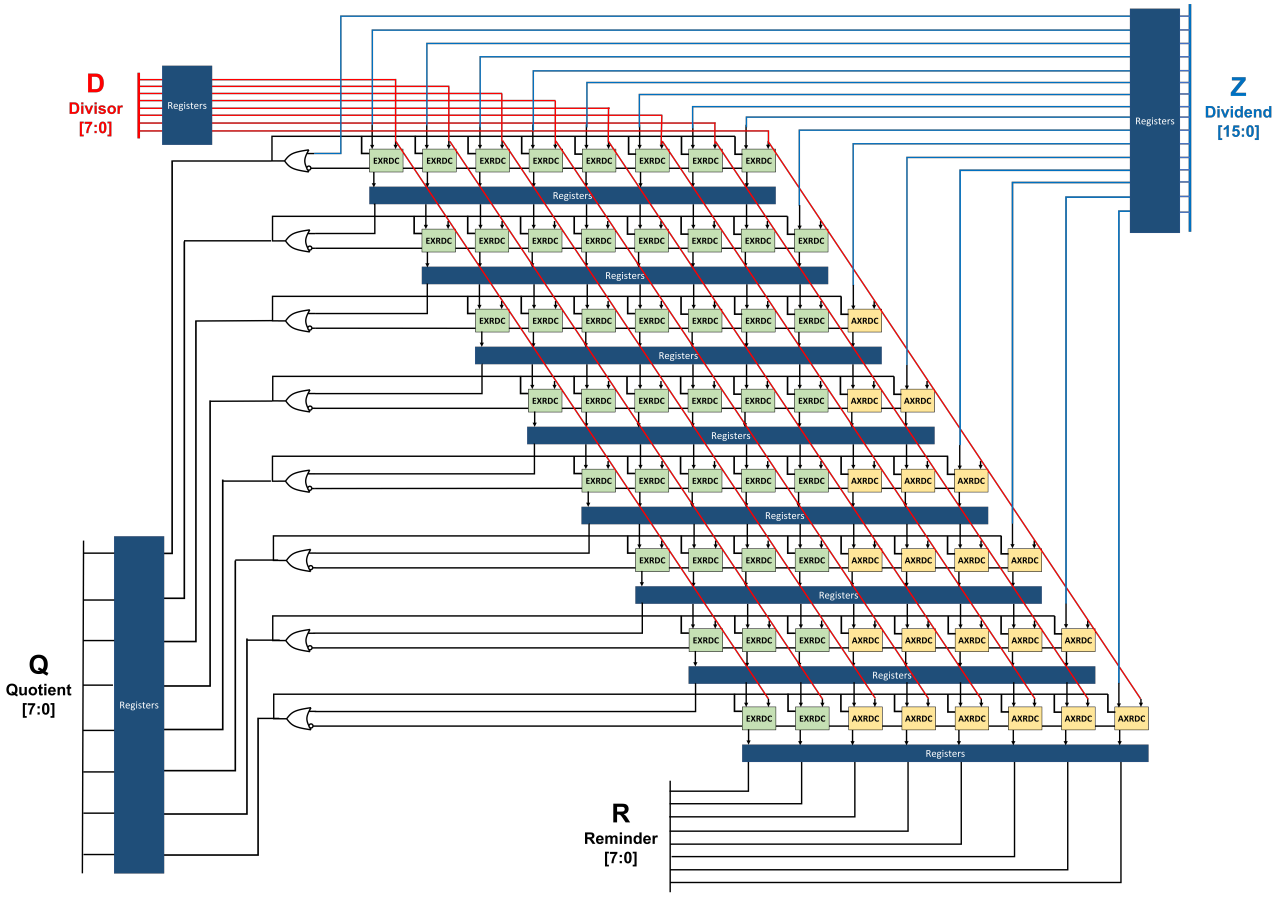


Figure 4.5 Architecture of 16/8 approximate pipelined array divider with p=6

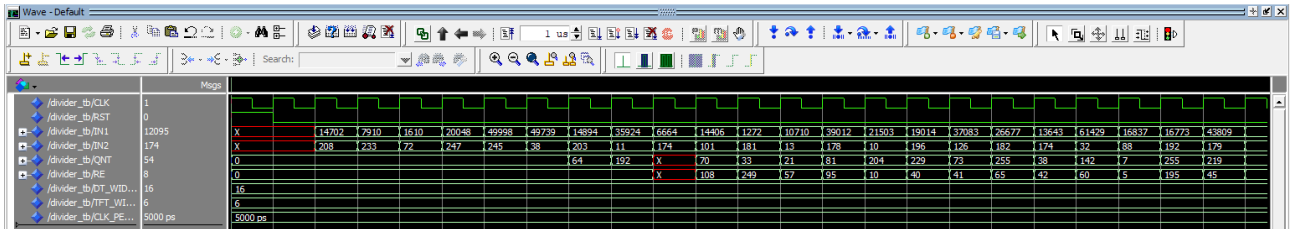


Figure 4.6 Simulation result of 16/8 approximate pipelined array divider with p=6

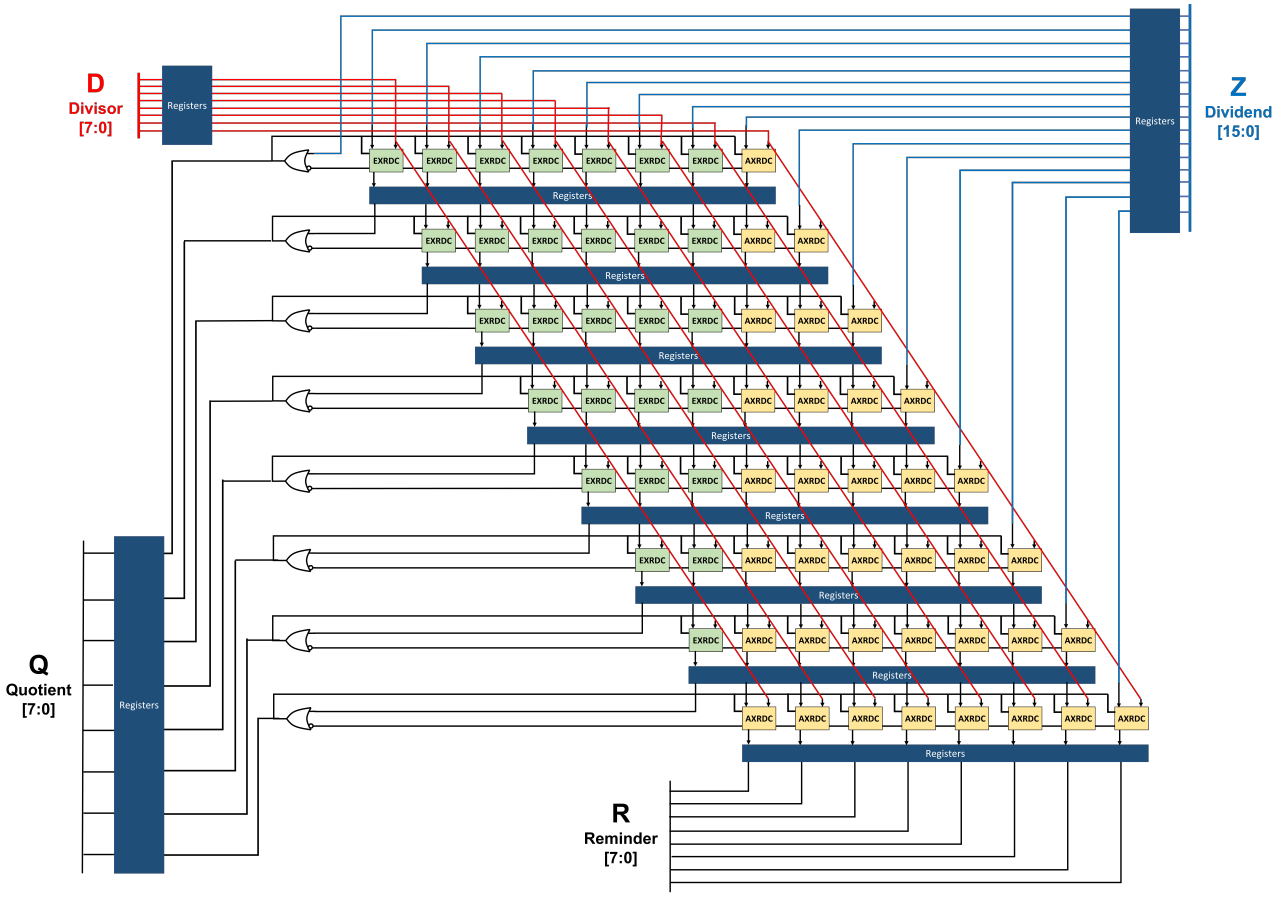


Figure 4.7 Architecture of 16/8 approximate pipelined array divider with p=8

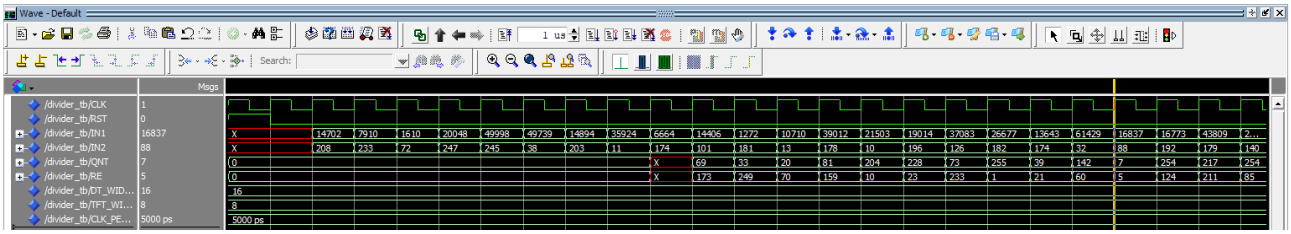


Figure 4.8 Simulation result of 16/8 approximate pipelined array divider with p=8

4.2 HDL Synthesis Results

Design is synthesized after functional simulation. The design is synthesized for Cyclone V FPGA in Intel Quartus Prime Software. The post-synthesis RTL netlist view for the Top level, Exact cells, and the Inexact cells are shown in Fig. 4.9, Fig.4.10 and Fig. 4.11 respectively.

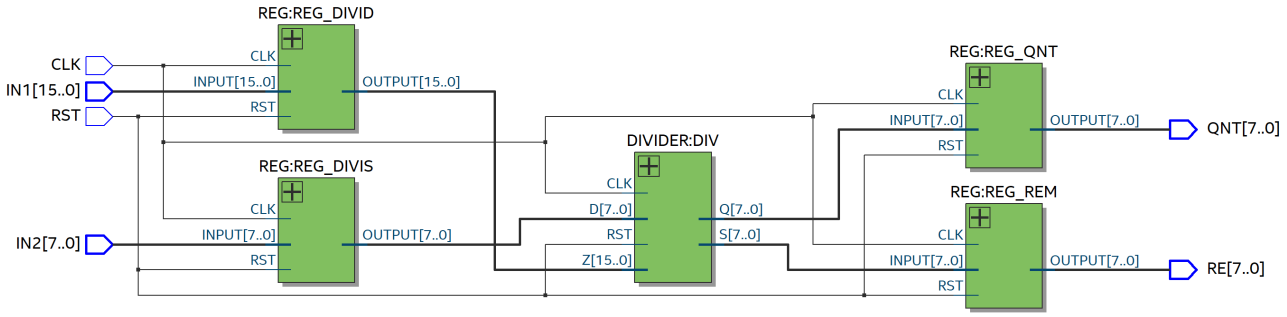


Figure 4.9 RTL View of synthesised design (16/8 divider) (Top View)

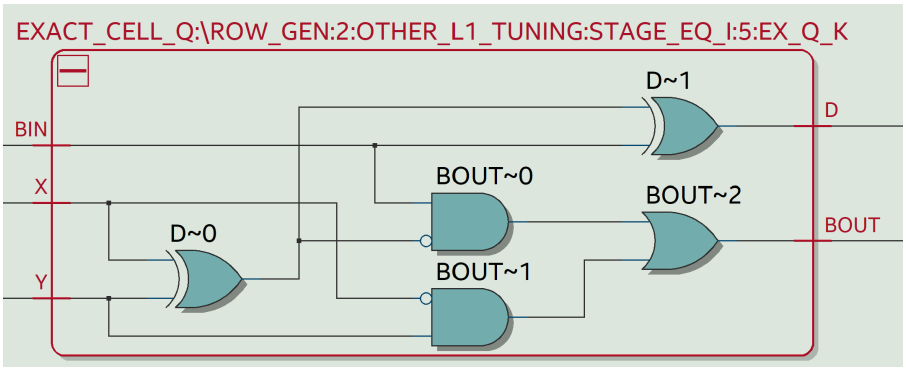
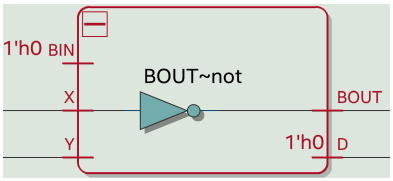
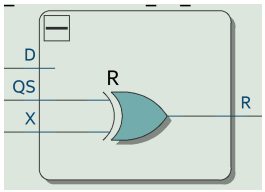


Figure 4.10 RTL View of Exact Divider Cell (EXRDC)



(a) Inexact Quotient Cell (Q)



(b) Inexact Reminder Cell (R)

Figure 4.11 RTL View of Inexact Divider Cell (AXRDC)

Table 4.1 summarizes the resource utilization and maximum clock of the different 16/8 divider designs. Compared to the exact design, the nonpipelined inexact designs save resources ranging from 10.83 to 44.34 percent. Additionally, there is a reduction of the critical path between 19.12 to 82.15 percent. However, the maximum frequency achieved for the nonpipelined divider is 58.15 MHz. Therefore, nonpipelined versions are best suited for low-speed designs.

On the contrary, the pipelined version achieves a maximum frequency of 193.05 MHz, but this speed gain comes at the cost of an increase in the number of registers. Compared to an exact design (p=0), the pipelined inexact versions save resource savings in percent, ranging from 7.74 to 15.18. Similarly, the path delay reduction is from 16.94 to 41.30 percent. On the other hand, although p=4 is an inexact design,

its maximum clock is less than the exact design. This contradiction may be due to ALUT (CLB) location or suboptimal routing. Thus, pipelined versions are most suitable for high-speed designs.

Table 4.1 Summary of Resource Utilisation and Clock Performance of 16/8 Approximate Array Divider

	Tuning Factor (p)		**0 (Exact)**	**4**	**6**	**8**
	Error Rate (%)			5.07	23.40	66.24
Without Pipeline	**Freq. Max (MHz)**		31.92	38.02	42.75	58.15
	Resources	**ALUTs**	163	141	112	73
		Regs	40	40	40	40
	Resource Savings (%)			10.83	25.13	44.34
	Path Delay Reduction (%)			19.12	33.93	82.15
With Pipeline	**Freq. Max (MHz)**		136.63	135.24	159.77	193.05
	Resources	**ALUTs**	112	114	107	97
		Regs	224	214	203	188
	Resource Savings (%)			2.38	7.74	15.18
	Path Delay Reduction (%)			-1.01	16.94	41.30

Table 4.2 summarizes the resource utilization and maximum clock frequency of the different 32/16 divider designs. Compared to the exact design, the nonpipelined inexact designs save resources ranging from 12.28 to 42.04 percent and reduce the critical path by 16.98 to 64.18 percent. The maximum frequency achieved for the nonpipelined divider is 22.05 MHz. Similarly, the pipelined version performs at a maximum frequency of 103.68 MHz. Compared to the exact design, the pipelined inexact versions save resource savings, ranging from 11.15 to 17.03 percent, and reduce the critical path from 5.85 to 11.90 percent. The reduction in maximum clock frequency for the p=18 version maybe due to ALUT (CLB) location or suboptimal routing.

The inference from above data is that as the size of the input increases, the size of the array increases resulting in lower speeds and an increased need for resources.

Table 4.2 Summary of Resource Utilisation and Clock Performance of 32/16 Approximate Array Divider

	Tuning Factor (p)		**0 (Exact)**	**12**	**15**	**18**
Without Pipeline	**Freq. Max (MHz)**		13.43	15.71	19.86	22.05
	Resources	**ALUTs**	498	427	350	255
		Regs	80	80	80	80
	Resource Savings (%)			12.28	25.61	42.04
	Path Delay Reduction (%)			16.98	47.88	64.18
With Pipeline	**Freq. Max (MHz)**		92.66	98.08	103.68	78.87
	Resources	**ALUTs**	460	394	360	332
		Regs	832	754	712	667
	Resource Savings (%)			11.15	17.03	22.68
	Path Delay Reduction (%)			5.85	11.90	-14.48

5 Conclusion and Outlook

The approximate computing paradigm will be vital in leveraging existing manufacturing technologies to realize high-performance, energy-efficient, compact circuits and systems. The state-of-the-art approximate versions of basic arithmetic circuits such as adders, multipliers, and dividers can be reused to reduce development time and effort significantly.

This project aimed to investigate recent approximate divider algorithms and implement the best-suited fixed-point divider in HDL. The restoring array-based approximate divider and linear piecewise approximation dividers were functionally analyzed. The array-based approximate divider methodology was selected based on superior error and circuit characteristics for HDL implementation. Two versions of the array-based divider were realized. While the nonpipelined versions achieve significant resource savings, they are most suited for low-speed applications. For high-speed applications, pipelined versions are best suited.

Additionally, with an increase in the size of the inputs, the array-based divider performance significantly decreases. This decrease can be attributed to the larger critical path of the design. Overcoming this increase in the critical path and achieving higher speeds for larger array-based dividers can be a part of future research.

List of Figures

3.1 Exact restoring divider cell (EXRDC) (left) and approximate divider cell (AXRDC)(right) . . 6
3.2 Architecture of $8/4$ array divider 7
3.3 Block Diagram of the piecewise linear approximation divider [10] 8
3.4 X_b vs f_b curves for various tuning factors (r) 8
3.5 Design Verification Strategy Block Diagram 9

4.1 Architecture of 16/8 approximate non-pipelined array divider with p=6 11
4.2 Simulation result of 16/8 approximate non-pipelined array divider with p=6 11
4.3 Architecture of 16/8 approximate non-pipelined array divider with p=8 12
4.4 Simulation result of 16/8 approximate non-pipelined array divider with p=8 12
4.5 Architecture of 16/8 approximate pipelined array divider with p=6 13
4.6 Simulation result of 16/8 approximate pipelined array divider with p=6 13
4.7 Architecture of 16/8 approximate pipelined array divider with p=8 14
4.8 Simulation result of 16/8 approximate pipelined array divider with p=8 14
4.9 RTL View of synthesised design (16/8 divider) (Top View) 15
4.10 RTL View of Exact Divider Cell (EXRDC) 15
4.11 RTL View of Inexact Divider Cell (AXRDC) 15

List of Tables

2.1 Approximate Dividers Designs Normalized Summary 5

3.1 The α and β values calculated using MATLAB for different rounding factors (r) 9
3.2 MATLAB Simulation Summary - Array Based Inexact Divider 10
3.3 MATLAB Simulation Summary - Linear Piecewise Approximate Divider 10

4.1 Summary of Resource Utilisation and Clock Performance of 16/8 Approximate Array Divider 16
4.2 Summary of Resource Utilisation and Clock Performance of 32/16 Approximate Array Divider 16

Bibliography

[1] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In 2013 18th IEEE European Test Symposium (ETS), pages 1–6, 2013.

[2] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu and J. Han, "Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications," in Proceedings of the IEEE, vol. 108, no. 12, pp. 2108-2135, Dec. 2020.

[3] Omkar G. Ratnaparkhi and Madhav Rao. 2022. LEAD: Logarithmic Exponent Approximate Divider For Image Quantization Application. In Proceedings of the Great Lakes Symposium on VLSI 2022 (GLSVLSI '22). Association for Computing Machinery, New York, NY, USA, 437–442.

[4] Y. Wu et al., "An Energy-Efficient Approximate Divider Based on Logarithmic Conversion and Piecewise Constant Approximation," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 69, no. 7, pp. 2655-2668, July 2022.

[5] E. Adams, S. Venkatachalam and S. -B. Ko, "Approximate Restoring Dividers Using Inexact Cells and Estimation From Partial Remainders," in IEEE Transactions on Computers, vol. 69, no. 4, pp. 468-474, 1 April 2020.

[6] J. Melchert, S. Behroozi, J. Li and Y. Kim, "SAADI-EC: A Quality-Configurable Approximate Divider for Energy Efficiency," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 11, pp. 2680-2692, Nov. 2019.

[7] C. K. Jha, K. Prasad, V. K. Srivastava and J. Mekie, "FPAD: A Multistage Approximation Methodology for Designing Floating Point Approximate Dividers," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 2020.

[8] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram and Z. Navabi, "TruncApp: A truncation-based approximate divider for energy efficient DSP applications," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, Lausanne, Switzerland, 2017, pp. 1635-1638.

[9] R. Zendegani, M. Kamal, A. Fayyazi, A. Afzali-Kusha, S. Safari and M. Pedram, "SEERAD: A high speed yet energy-efficient rounding-based approximate divider," 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 2016, pp. 1481-1484.

[10] Marzieh Vaeztourshizi, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2018. An Energy-Efficient, Yet Highly-Accurate, Approximate Non-Iterative Divider. In Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '18). Association for Computing Machinery, New York, NY, USA, Article 14, 1–6.

[11] M. Imani, R. Garcia, A. Huang and T. Rosing, "CADE: Configurable Approximate Divider for Energy Efficiency," 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 2019, pp. 586-589.