Digital image processing based on the residue number system

By

Azadeh Safari

A thesis submitted to Macquarie University
for the degree of
Doctor of Philosophy
Department of Engineering, Faculty of science
October 2014

MACQUARIE
UNIVERSITY
FACULTY OF SCIENCE

Dedicated to

My beloved husband, Peyman, whose endless love and support made PhD a significant journey,

and

My parents, whose unconditional love and encouragement allowed me to start and complete this journey.

Acknowledgements

I would like to express my sincere gratitude and heartfelt appreciation to my supervisor, Dr Yinan Kong for his excellent guidance and continuous encouragement throughout my research. I particularly appreciate his patience and motivation that helped me to learn a lot, and challenge myself to do the best job that I could do.

I also would like to extend my thanks to A/Prof. Sam Reisenfeld, my co-supervisor, for his excellent support. I am thankful to Niras C.V., James Nugent, Fujimi Bentley, Fateme Ghasemi, and Davar Kheirandish for being great collaborators, and for helpful ideas.

I would like to thank Mr Adrian Ng and Synopsys Customer Support for providing outstanding support and frequent meetings to help us solve technical problems with Synopsys tools.

I also thank Professor Danijela Cabric of the Electrical Engineering Department of the University of California, Los Angeles, and Professor Philip Leong of the University of Sydney for inviting me to visit their research laboratories to collaborate and meet with their graduate students and faculty members.

I am grateful to Dr Keith Imrie for valuable advice and useful comments that improved the quality of this thesis. I am also thankful to the admirable staff in the Department of Engineering for their wonderful support.

I wish to acknowledge Macquarie University for awarding me the International Macquarie University Research Excellence Scholarship (iMQRES), and providing financial support to attend national and international conferences during this project.

List of Publications

Publications related to the field of this thesis for which the author is the primary author are as follows.

- Azadeh Safari and Yinan Kong. *Simple, Fast and Synchronous Hybrid Scaling Scheme for the 8-bit Moduli Set*, Journal of Emerging Trends in Computing and Information Sciences, Vol. 3, No. 6, June 2012.
- Azadeh Safari and Yinan Kong. *Four Tap Daubechies filter banks based on RNS*, International Symposium on Communications and Information Technologies (ISCIT) 2012, pp. 957-960, 2-5 Oct., Gold Coast, Australia.
- Azadeh Safari and Yinan Kong. *The application of lifting in Digital Image processing*, Advances in Mechanical and Electronic Engineering, Lecture Notes in Electrical Engineering, Volume 178, 2013, pp. 449-453, Springer, 2013.
- Azadeh Safari, Niras C V, and Yinan Kong. *VLSI architecture of multiplierless DWT image processor*, IEEE TENCON Spring 2013 Conference in Sydney Australia, pp. 280-284, 17-19 April 2013.
- Azadeh Safari, James Nugent, and Yinan Kong. *Novel Implementation of Full Adder Based Scaling in Residue Number Systems*, IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS2013), The Ohio Union at the Ohio State University, Columbus Ohio, August 4-7, 2013.
- Azadeh Safari and Yinan Kong. *Performance comparison of orthogonal and biorthogonal wavelets using technology libraries*, The 13th International Symposium on Communications and Information Technologies, IEEE, Samui Island, Thailand, September 4-6, 2013.
- Azadeh Safari, Fujimi Bentley, and Yinan Kong. *Operational Capability and Suitability of Image Compression Methods for Different Applications*, CCECE

2014, Ontario, Canada 4-7 May, pp.875-880, 2014.

- Azadeh Safari, and Yinan Kong. *Power-Performance Enhancement of RNS-Based DWT Image Processor Using Multiple Voltage Domains*, In preparation.

Publications related to the field of this thesis for which the author is the second author are as follows.

- Davar Kheirandish, Azadeh Safari, and Yinan Kong. *Using one hot residue number system (OHRNS) for digital image processing*, The 16^{th} international symposium on artificial intelligence and signal processing (AISP 2012), 2-3 May 2012, Shiraz University, Shiraz, Iran.

- Niras C V, Azadeh Safari, and Yinan Kong. *Overlapped block processing VLSI architecture for separable 2D filters*, National Conference on Emerging Trends in VLSI and ES, January 2013.

- Davar Kheirandish, Azadeh Safari, and Yinan Kong. *A Novel Approach for Improving Error Detection and Correction in WSN*, CCECE 2014, Ontario, Canada, 4-7 May, pp. 370-373, 2014.

- Yinan Kong, Azadeh Safari, Niras C.V. *A low-cost architecture for DWT filter banks in RNS applications*, Abstract accepted in International Symposium on Integrated Circuits (ISIC 2014), Singapore, 10-12 December, 2014.

- Fatemeh Ghasemi, Azadeh Safari, Saied Sorouri, Amir Sabbagh Molahosseini, Yinan Kong, *An Efficient RNS Scaler for the Four-Moduli Set* $\{2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1\}$, Submitted to Very Large Scale Integration (VLSI) Systems, IEEE Transactions on.

Abstract

This thesis presents the design, optimisation and physical implementation of a two-dimensional (2D) discrete wavelet transform (DWT) image processor using the residue number system (RNS), and examines it against an initial processor designed based on existing binary modules. The original contributions of the proposed design include a low-complexity hardware architecture of the RNS-based filter banks, optimised transposition units, and exploitation of the multi-voltage scheme to reduce the power consumption. Modular adders and multipliers of the RNS-based filter banks are simplified to save on hardware complexity, while modular arithmetic and 6-bit dyadic-fraction filter coefficients are applied to improve the system performance. The proposed design is synthesised with the Synopsys 90 *nm* Generic Library ($SAED90nmEDK$) using the Synopsys synthesis and implementation tools. The synthesis results show that the proposed RNS-based processor is 23% faster than the initial processor. Another noteworthy result is that the total area of the RNS-based processor is less than the total area in the initial binary processor. It confirms that using the proposed architecture for RNS-based filter banks has saved on the hardware complexity and the system area requirement. The proposed RNS-based processor is implemented using the multi-voltage (MV) low power design (LPD) scheme to improve the power performance of the proposed processor. The power synthesis results show that using the multi-voltage scheme reduces the total power of the proposed RNS-based design by up to 50%. The proposed residue arithmetic units are explained in details to illustrate the novelty of the proposed design.

Contents

List of Figures

List of Tables

1 Introduction

As technology has advanced in past decades, people have seen the potential of computing devices and have dreamed of the faster and more efficient products and services that can be provided only by digital hardware. This has created a demand for continual advancement in the speed and efficiency of existing systems.

Advances in science and technology have also required the use of digital image compression in demanding applications such as commercial photography, industrial imagery, geophysics, machine vision, medical imaging, control and automation, telemetry, satellite imagery, military and security sciences, agriculture applications, graphic arts and multimedia, network, Internet, or storage media. All these applications require a large volume of computational operations for processing images. Hence, exploring a way to increase the processing speed and reduce the power consumption is essential.

Various methods have been developed and introduced to optimise existing image processing systems. Among proposed schemes, optimising the arithmetic level of image processors and replacing conventional number systems by the Residue Number System (RNS), have drawn more attention, since arithmetic system plays an important role in satisfying the requirement for a large volume of computational operations in image processing [10, 11].

Conventional number systems use a base approach, where each digit is multiplied by a base value to attain its real value. All the weighted values are then added up to attain the actual number. The three most commonly used examples of conventional number systems are binary (base 2), decimal (base 10) and hexadecimal (base 16).

Replacing conventional binary number systems with the Residue Number System (RNS) has been steadily rising in favour over the last 50-60 years. The RNS-based architecture allows the processing of modular channels simultaneously, and saves significant delays in arithmetic operations. Using small integers in independent channels also reduces the carry propagation and the number of partial products in adders and multipliers, respectively [12, 13].

Despite many publications regarding the application of RNS for digital image processing [3, 14–19], it is still at an early stage. Due to RNS's complex operations (sign detection, division and magnitude comparison) most of these studies are based on ultimately unjustified assumptions, and are inexact with many errors in the results.

This thesis will provide a design and optimisation for a RNS-based digital image processor in detail and examine it against a binary processor. The study will also implement the proposed processor with static voltage scaling to achieve the best power-performance trade-off in the proposed image processor.

1.1 Existing Literature

The literature regarding RNS-based image processing is sparse. An extensive literature search on the "RNS-based image processor" results in only a few tens of hits and many of them are not directly relevant to the interests of this thesis. The literature which has been deemed to be directly relevant to our topic is discussed below. The first FIR filter implementation using RNS has been reported in 1977, by Jenkins [20]. They implemented a dual-bandpass FIR filter using the moduli set (7, 9, 11, 13, 16), and compared it with a 2's-complement contender. The authors concluded that the RNS-based FIR filters perform with higher throughput than the 2's-complement contender. In 2000, a RNS application in implementing orthogonal wavelet filter banks was presented [21, 22]. The filter banks were designed using LUTs, and they processed 8-bit inputs with 10-bit filter coefficients. The resulting system ran 23.45% and 96.58% faster than a 2's-complement design for one and two octaves, respectively. The weakness of their design was using LUTs for modular multiplications. They provided a comparison between FPL implementation of RNS and binary 1D DWT architectures. No details of the synthesis results were provided.

In 2001, Ammar et al. [23] used RNS and the Chinese Remainder Theorem (CRT) for encoding and encryption of image pixels. The result was an encrypted image that needed the moduli set (key) for decryption and reading. They used look-up tables (LUT) for conversion from RNS to the binary number system; however there was no compression process in the proposed technique.

The design and implementation of a RNS wavelet processor using custom IC technologies has been presented by in 2003 [14]. They used RNS for implementing wavelet filter banks using an enhanced index transformation over a Galois field. Their system was compared with 2's-complement designs, and showed up to 100% performance improvement. The authors only contributed to designing the filter banks rather than implementing the whole processor. The main shortcoming of the Ramirez design is that it does not address any other RNS modules in the RNS wavelet processor apart from the filter banks. Hence, they neglected the effect of the multi-dimensional requirement of processing an image, which is addressed in this thesis.

In contrast with [14], the authors in [24] used RNS to implement low-power and low-leakage FIR filters. This idea was contrary to the Ramirez design, that concluded that RNS increased the speed of operations but also increased the hardware complexity and power consumption. They applied RNS in order to reduce the static and dynamic power consumption of FIR filters. [24] compared 16-, 32- and 64-tap FIR filter implementations in 2's-complement and RNS, and showed that the RNS filters offer a reduction of 50% in static power dissipation and a total power reduction of 40%. The effect of RNS on power consumption of a system is provided in Chapter 7.

In 2004, the authors in [25] investigated the application of RNS for digital image processing. They presented a VLSI implementation of an image coding scheme using the RNS and modified CRT. The proposed scheme encrypted the entire image and did not require any additional component other than a standard RNS system. Again, no compression algorithm was used for image compression. The details of the modular units, nor the synthesis results of their processor were not provided.

Another RNS-based design in 2004 was the design and implementation of an RNS-based image processor using DWT filter banks and RNS arithmetic in [3]. The proposed design used 27 look-up tables (LUT) for modular multiplication, each with 8-bit width and 256 entries. The downside of their processor was that the LUTs and RAMs are the main sources of leakage power, which is the major concern of standalone applications like mobile phones and cameras. In addition, the authors in [14] have reported that the best RNS designs have a hardware cost about the same as or more than the binary designs. In other words, while using RNS can help to enhance the performance of an image processor, the architecture of using modular arithmetic might increase the hardware complexity and power consumption of the system [26]. Therefore, one question that needs to be answered is what aspects of RNS-based designs are superior to binary designs.

The authors in [27] have discussed a modular implementation of image convolution steps based on the Generalised Fermat number transform (GFNT), and implemented the proposed circuit using VHDL. The proposed system used the Fermat Number Transform (FNT) and modulus $q = 2M + 1$, where M is an integer power of two. The resultant system has been suggested for image convolution applications; however, there was a limitation on the appropriate word length for the moduli set. Two solutions were proposed to overcome this limitation. The Generalised FNT (GFNT) is proposed as the first solution, in which M was not restricted to be an integer power of two but may be an arbitrary integer. This solution was suitable for block-based image and video filtering applications, but it reduced the transform length to 16 or 32 points for $M < 32$. A second solution was to use number-theoretic transforms based on the RNS and enlarge the effective modulus with moduli $q(1) = 2 \times 16 + 1$ and $q(2) = 2 \times 8 + 1$. It provided a transform length as large as 256 points, and a 24-bit dynamic range.

The authors in [28] suggested Number-Theory-based Image Compression Encryption (NTICE) using a number-theoretic paradigm and CRT for compression and encryption of colour images. The results showed that this method was comparable to more complicated methods. The algorithm was applied for image multiplexing to achieve a high level of security with compression. This algorithm was tested on various test images using Matlab, and no further studies on hardware complexity and power efficiency of their algorithm were conducted.

The authors in [29] decided to combine binary and RNS architectures. They proposed an adaptive FIR filter using a mixed binary-RNS architecture. The RNS has been adopted for its great savings in time and power dissipation, while integrating the design with a conventional binary system was to avoid the scaling step.

Finally, digital image filtering of spatial and frequency domains based on the RNS has been studied in [18] to achieve high speed, high security and low-power-consumption digital image processing. The transmission security and filtering issues in the frequency domain are discussed, and the authors propose a method to compute new pixel values using the selected mask in the RNS. The study concluded that the RNS permit efficient digital image filtering.

As can be seen from the above literature review, the design and physical implementation of a RNS-based image processor has never been reported in scholarly papers.

In this thesis, details of a RNS-based image processor, from designing modules to verification will be presented. The resultant system is a chip that is ready for tape-out.

1.2 Motivation for This Research

In 1943 Thomas Watson, chairman of IBM, said: "I think there is a world market for maybe five computers". This is an indication of how the electronics industry has developed, and has been a surprise even to this leader of industry. His prediction became true in 1946 when early electronic computers used valves (*vacuum tubes*).

One of the world's first computers, called *Eniac*, had 18000 valves which used 10 *hp*, and it weighed 30 tonnes. In 1947, the first transistor was developed by Walter Houser Brattain, William Shockley, and John Bardeen in the Bell Laboratories. Early transistors in the mid 1940s cost between 5 and 45 $ each. In contrast, today each transistor costs 15 cents or less, and a tiny fraction of a cent in an integrated circuit. It shows that things get cheaper as we advance them.

In 1949, there was a clue in Popular Mechanics magazine where they forecast that "Computers in the future may weight no more than 1.5 ton". In 1954, IBM produced the first fully transistorised computer, that had 2000 separate transistors. In 1958, the first integrated circuit (IC) was invented at Texas Instruments by Jack Kilby. In 1971, the first microprocessor was produced by Intel, using 2300 integrated transistors. It was a huge number at that date, which is tiny today.

In 1975, chip complexity was predicted to double every one and a half years *(Moore's law)* by Gordon Moore, who was one of the founders of Intel. His original prediction in 1965 was: "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year". In 1975, ten years later, he looked back and refined his law to "one and half years". So, Moore's law says that, every one and half years, chips achieve double complexity and do twice as much. Up to date, in 2014, Moore's law has continued to rule the industry.

In 1977, Ken Olsen, president, chairman and founder of Digital Equipment Corporation, gave a disturbing quote: "There is no reason any one would want a computer in their home". In 1983, Apple introduced the first user-friendly computer. She was named "Lisa", weighed 52 pounds and cost 10,000 dollars. This was the entry to the portable computer age.

In 1998, the Intel Pentium II processor was produced with 7.3 million transistors. So, in a short time, we have gone from 2000 transistors up to 7 million transistors. In the same year, Aart J. De Geus, the CEO of Synopsys, said "Customers need to be 10 times more productive every 6 years”, which is in line with and in support of Moore’s law. In 2011, Inteligon produced a ten-core Westmere Xeon FX processor. This is 2600 million transistors in the size of a finger.

Today, the need for faster digital circuitry continues. Various methods have been developed and introduced to optimise existing systems. RNS arithmetic was first used in Computer Science. However, due to problems associated with the inefficient hardware of that time, it was not as well appreciated as it deserved. Advances in VLSI technology and an ability to design more efficient hardware has encouraged researchers to use this arithmetic system more than ever. In this thesis, we will focus on designing an image processor based on the RNS with less hardware complexity, a low area requirement and less power consumption.

1.3 Research Objectives

The main issues of image processing are increasing the transmission speed, minimising the power consumption and decreasing the storage requirement. This thesis seeks to remedy these problems by designing and implementing a high-speed, low complexity, power-efficient digital image processor. The main focus of this thesis is to investigate the effect of using the RNS on the speed of calculations, hardware complexity and power consumption of the proposed image processor. The resultant system is capable of compression of images using low power and less area, at high speed. The aim of this thesis is to serve as a gateway to enhance image processing features in the future.

1.4 Thesis Outline

The thesis is organised as follows:

- **Chapter 2:** A Review of Image Compression Algorithms and Schemes

  In this chapter the necessary background and preliminary study of the most common image compression methods and ways to improve and expand the algorithms are provided. It aims to provide an insight into the characteristics of commonly used image compression algorithms. Each method is tested with controlled test images to determine the compression method’s operational capabilities for the different test cases. Their different performance under certain circumstances is simulated and compared with other methods, and discussions on how algorithms will behave in other circumstances are given in detail. The desired outcome of this chapter is a clear comparison between each method simulated, thereby determining its suitability for different applications. At the end of this chapter, we will select the compression algorithm and filtering scheme for the proposed digital image processor.

- **Chapter 3:** Discrete Wavelet Transform for Image Processing Applications

  This chapter begins with a brief introduction to the discrete wavelet transform (DWT). It will then move on to the description of orthogonal and bi-orthogonal wavelets. The most common orthogonal wavelets compete with the bi-orthogonal wavelets in the Xilinx synthesis tools. To establish the properties of each wavelet family, further synthesis will be performed using Synopsys Application Specific Integration Circuit (ASIC) technology libraries. This unique study of wavelets with standard cell libraries using high-quality methodology and industry-leading technical software will serve as a convenient reference for wavelet users and reviews. At the end of this chapter, we will choose the right wavelet family for image compression in this thesis.

- **Chapter 4:** The Residue Number System

  This chapter provides background information and a preliminary study of the residue number system (RNS) that are directly relevant and useful for designing and implementing the proposed processor. The most useful tools in number theory, including the Chinese remainder theorem (CRT) and a modified CRT, are presented. Furthermore, efficient moduli set selection and bit efficiency improvement are investigated. We use this study to choose the appropriate moduli set and bit width of the RNS operational blocks in the proposed image processor.

- **Chapter 5:** Scaling in Residue Number System

  In this chapter, we will propose a scaler based on modular reducers, as a modification to Chang's full-adder-based scaler in [1]. Subsequently, a hybrid scaling scheme for the three-moduli set $(2^n - 1, 2^n, 2^n + 1)$, scaling factor 2^n $(n = 8)$ will be presented. Both LUTs and modular adders are employed efficiently to generate the accurate scaled residues. A reverse conversion from residues to the original binary number is generated as by-product of scaling. For the first time, we propose a scaling scheme for a four-moduli set. To date, no scaling scheme is reported for any form of four-moduli set, yet.

- **Chapter 6:** Logic Design and FPGA Implementation of Initial and the Proposed 2D RNS-based DWT Image Processors

In this chapter, we will present the logic design of an initial and the proposed 2D RNS-based DWT image processor. We will start with an initial image processor designed based on binary arithmetic and optimise it to achieve the thesis goals. Development of RNS-based architectures for the optimal performance and coordination with VLSI will be carried out using Matlab, Maple and Xilinx FPGA tools.

- **Chapter 7:** RTL to Gate Synthesis

  We have considered timing and delay modification as the highest priority in our design goals, followed by the power and area of the proposed processor. In this chapter, we will synthesise the proposed logic design using Synopsys tools, and look for the best trade-off between timing, area and power.

- **Chapter 8:** Physical Implementation Using Design Compiler (DC) Topographical Technology in ASIC Methodology

  As the last step of the project, the proposed image processor will be implemented using DC topographical technology in ASIC methodology. The DC topographical technology is selected because it eliminates design iterations and reduces the overall design cycle. The synthesised design will be ready for tape-out and final testing and verification. Milkyway technology and the IC compiler are used for this purpose.

- **Chapter 9:** Thesis Conclusion and Recommendations for Future Work

  Finally, a conclusion of the thesis and an indication of future research directions will be provided in this last chapter.

1.5 Tools

Tools used in this thesis:

- Matlab R2013b
- Maple 17
- Xilinx ISE project navigator 14.5
- PlanAheadTM
- Mentor Graphics Modelsim

- Synopsys Asia Pac FrontEnd University Bundle (Design Compiler, Design Vision, Custom Wave View, Prime Time PT-PX, VCSMX)
- Synopsys Asia Pac BackEnd University Bundle (IC Compiler, Milkyway Environment)
- Red Hat Enterprise Linux WS release 4
- LaTeX Basic MiKTex 2.9.5105
- Workstation: Dual Intel(R) Xeon(R) X5650 2.66 GHz, 12M cache, 6.4 GT/s QPI, Turbo, HT, 6C 72GB (9 $\times$ 8BG) DDR3 RDIMM Memory, 1333MHz, ECC 500GB 7200 RPM 3.5” SATA hard drive Dell Precision T5500 Chassis with TPM

2 Review of Image-Compression Algorithms and Schemes

Image compression is one of the most widely used groups of signal processing schemes in real-time applications. Compression of digital images is essential in image storage and transmission. The main objective of image compression is to decrease the memory space, increase the transmission speed and minimise bandwidth utilisation. There are various algorithms for compressing images, each with pros and cons. Each method of data compression behaves differently for various types of data, thus a review of the common image algorithms seems necessary to compare the compression ratio, compression and decompression time, and generated file sizes [30].

In this chapter the necessary background and preliminary studies of the most common image-compression methods, and systematic approaches to improve and expand the algorithms, are provided. It aims to provide an insight into the characteristics of commonly used image-compression algorithms. Each algorithm is tested with controlled test images to determine the operational capabilities of each algorithm for different test cases. Subsequently, the performance of each algorithm is simulated and compared with other methods. Discussions on how algorithms behave with various test images are given in detail. The desired outcome of this chapter is a clear description of compression algorithms, and determining the suitability of each algorithm for different applications.

2.1 Image-Compression Schemes

Minimising redundant information in digital images is the main concern in image-compression algorithms. They can be classified, based on their integrity to the original image, to be lossy or lossless (noisy or noiseless). Lossless image compression is the exact reconstruction of the original image, while in lossy image compression some of the information cannot be achieved in the receiver.

Each classification has merits to fulfil the requirements of particular applications. Common lossless compression standards are PNG, JPEG-LS, GIF, JBIG and Photo CD. They are generally based on Shannon and entropy coding theories. The Shannon-Hartley theorem states that an infinite-bandwidth and noise-free analogue channel can transmit error-free data/unit of time unlimitedly [31].

JPEG is an ISO/ITU standard for compression of continuous-tone images which can be lossy or lossless. In 1991, a joint group of image specialists in the International Organisation for Standardisation (ISO) and the International Electro-technical Commission (IEC), called the "Joint Photographic Experts Group", proposed the JPEG compression scheme [32].

The JPEG scheme accepts an input image in 8×8 blocks, and provides 8×8 frequency-space components. In each block, the pixel located at row 0 and column 0 is called a "DC term", and the remaining 63 pixels are called "AC components". The DC term represents the average frequency rate of the block, while the AC terms are the spatial frequencies of the cosine terms within the series [31].

Two extensions of JPEG are JPEG-LS and JPEG 2000. The JPEG-LS compression standard is based on the LOCO-I (LOw Complexity LOssless COmpression Image) algorithm [33] which is known as an efficient algorithm. JPEG 2000 is the latest version of JPEG, designed to provide efficient compression for existing compression ratios [34]. The main feature of JPEG 2000 over prediction-based lossless JPEG is the use of reversible DWT [35]. JPEG 2000 also has reversible and irreversible schemes, depending on the filter bank coefficients. Reversible JPEG 2000 uses a rounded version of the bi-orthogonal CDF 5/3 wavelet transform with no quantisation [36]. CDF 5/3 uses integer coefficients, and the quantisation step size is equal to one (no quantisation performed). For irreversible coding, the CDF 9/7 wavelet transform is used. This scheme is not reversible because it introduces quantisation noise (rounding) depending on the precision of the decoder [37].

Other popular coding techniques are Vector Quantisation (VQ), Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT), which can be either lossy or lossless.

The VQ algorithm is a block coding scheme that yields a better image quality only when the bit rate is small. This algorithm needs $K^2 \times N$ computations for coding an image of size $N \times N$, where K is the number of code words [38].

The DCT is the basis of the well-known JPEG compression standard. It is a derivative of the Fourier Transform. By eliminating the sine components in a Fourier Transform, the DCT is derived as a series of cosine expressions [39]. In theory, a Fourier transform represents an input signal with a series of sine and cosine expressions. However, in DCT the sine components are eliminated [40]. In DCT, $N^2 log_2 N$ operations are required for coding an image of size $(N \times N)$. In block-coding DCT schemes, block size affects the image quality and compression efficiency. A large block size causes quality degradation and ringing in the image.

Image-compression algorithms can also be classified based on the selected compression steps. Image compression has two main steps. The first step is decorrelation, which is removing inter-pixel or spatial redundancy. The second step is entropy coding which is removing coding redundancy.

Some of the decorrelation techniques are Differential Pulse Code Modulation (DPCM), the Walsh-Hadamard Transform (WHT), and multi-resolution techniques. The multi-resolution algorithms such as hierarchical interpolation (HINT), Laplacian pyramid, and S-transform perform transmission of data in progressive levels of increasing resolution. The DPCM algorithm usually requires a high bit rate compared to other coding algorithms for the same image quality, executes with a high capability of reducing redundancies, and is very susceptible to channel errors [38].

Subsequent to decorrelation, entropy coding such as Huffman coding or arithmetic coding are usually employed to achieve higher compression ratios [41]. Huffman coding is a lossless compression technique which is complex when there are many colour levels. In Huffman coding, the fundamental concept is to use long code words for infrequent symbols and short code words for the most frequent symbols [41].

2.2 Common Image-Compression Algorithms

Among image-compression algorithms, the four most common algorithms are discussed in detail in the following. Selected algorithms are simulated, and the results are compared to provide a clear picture of the advantages and disadvantages of each algorithm.

2.2.1 Run-Length Encoding

The Run-Length Encoding (RLE) algorithm refers to a form of lossless data compression used for consecutive data elements. It is one of the simplest methods of data compression and, depending on the variant and techniques used with it, can yield very varied results in different circumstances. However, the only suitable application for the general form of the RLE is when data is repeated in large amounts; otherwise, this method can easily expand and create an output larger than the original data. The expanding effect occurs due to the singular control element in the RLE. As a result, many adaptations of RLE have been developed. Two main approaches are inclusion of a table of elements, and a predictive algorithm. The inclusion table of elements is a similar technique to Huffman coding, that uses a binary tree as a dictionary of elements. Predictive algorithms discard changes that are too small to change the control element. The RLE algorithm provides lossy data compression, which can be acceptable in particular applications. Using a predictive algorithm is generally a small modification that is commonly used as a minor improvement but yields a greater compression ratio at the cost of processing time [42]. The RLE algorithm performs linear decoding, traversing through the encoded elements. There is no effective way to perform partial decompression due to the linearly encoded nature of the RLE compression. Figure 2.1 shows the flowchart of the RLE algorithm.

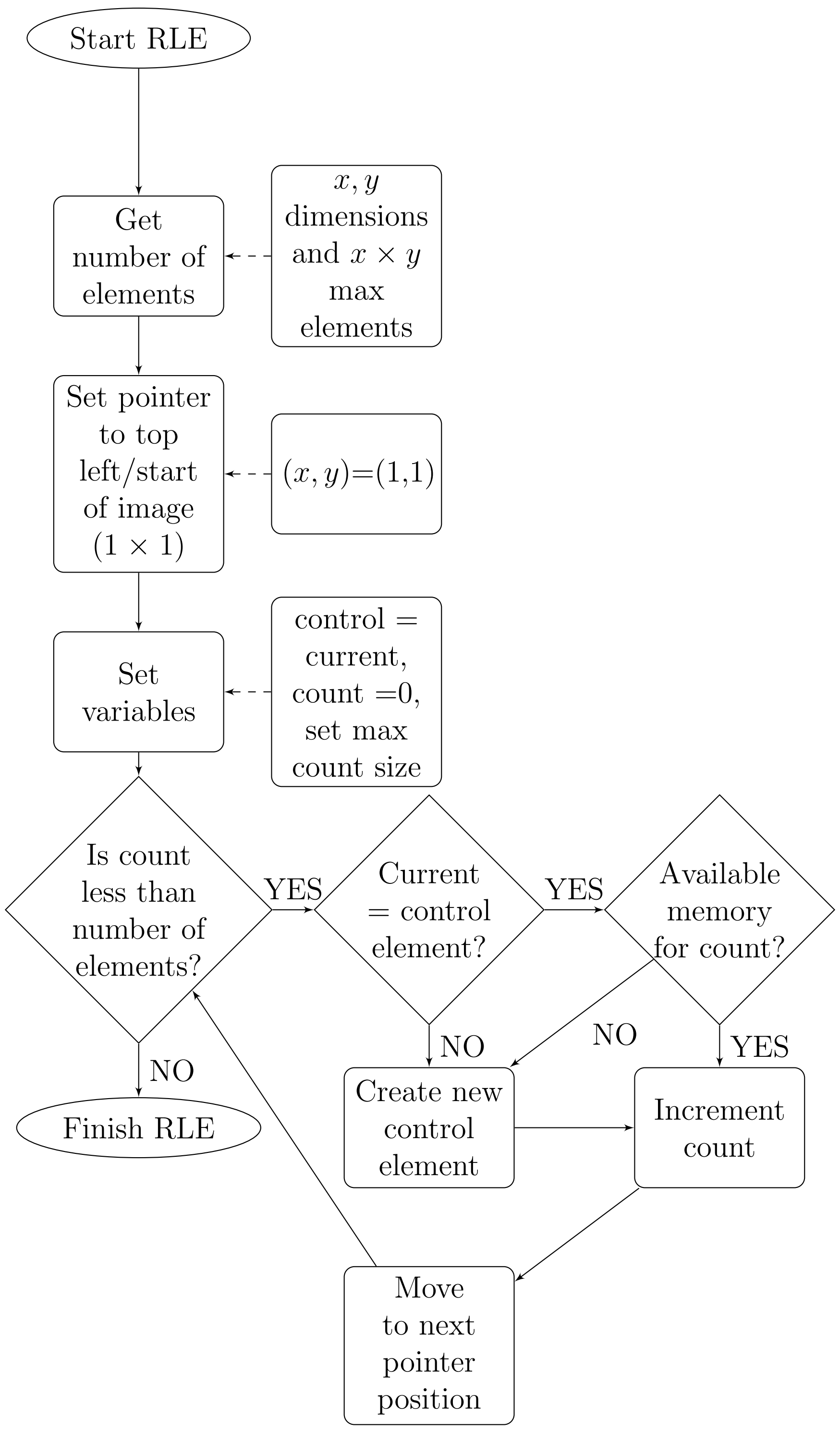

Start RLE
Get number of elements
x, y dimensions and $x \times y$ max elements
Set pointer to top left/start of image (1×1)
(x, y)=(1,1)
Set variables
control = current, count =0, set max count size
Is count less than number of elements?
YES
Current = control element?
YES
Available memory for count?
NO
NO
YES
NO
Finish RLE
Create new control element
Increment count
Move to next pointer position

Figure 2.1: RLE flowchart

2.2.2 Entropy Coding

Entropy coding is usually implemented using either Huffman or Arithmetic coding. Huffman coding is, in effect, Arithmetic coding that has shortcuts at key points. Arithmetic coding generates the most optimised values, while Huffman stops the process at a certain point to maintain a balance between processing time and achieving acceptable levels of entropy encoding. Entropy coding is a lossless algorithm, unless it has been modified to include a prediction/merging algorithm, which is not common.

Although Arithmetic coding increases overall compression by 5% to 10% compared to Huffman coding, it is not as widely used as Huffman. One reason is that the highly computational requirement of Arithmetic coding makes it a less attractive algorithm for image-compression applications. Dedicated hardware implementations exist which greatly mitigate this factor, but in most cases Arithmetic coding is unsuitable for real-time applications [43].

2.2.3 Discrete Cosine Transform

The Discrete Cosine Transform (DCT) is a very popular algorithm for image compression due to its energy-compaction capability. The main driving force behind the popularity of DCT is its use of real numbers and its potential for reaching the maximum theoretical efficiency.

An image to be processed by DCT is broken down into matrices, and is then further broken down into sections of size N, where N is typically 8, however fairly common variants of DCT exist where N = 4, 8 or 16. The effect of increasing the value of N has very well documented properties, where $N = 8$ is regarded as balanced as it yields an acceptable compression delay / (compression ratio + quality), a lower value has a high compression delay / (lower compression ratio and decreased quality) and higher values of N yields the inverse.

Due to the energy compaction nature of the DCT algorithm, the precision required to accurately recreate an image decreases the further the pixel is from the top left. The actual method of determining how much quantisation should be applied is another reason as to why there are so many different variants of DCT. Quantisation reduces the number of bits needed to store an integer value. The Joint Photographic Experts Group created a quantisation matrix by finding the error between input and output and determining what an acceptable level of error is by trial and error, and as such there are many different variants of this matrix.

In this thesis, three different variants of DCT is considered. The standard version of DCT needs $O(N^4)$ clock cycles for computations. It has been optimised to $O(N^2)$ clock cycles. which is very close to the optimum $O(NlogN)$, through the inclusion of the Fast Fourier Transform (FFT). DCT benefits from energy compaction and focuses most of the required data (described as energy) in a small area. Thus, the rest of the image requires less information to be reproduced [44].

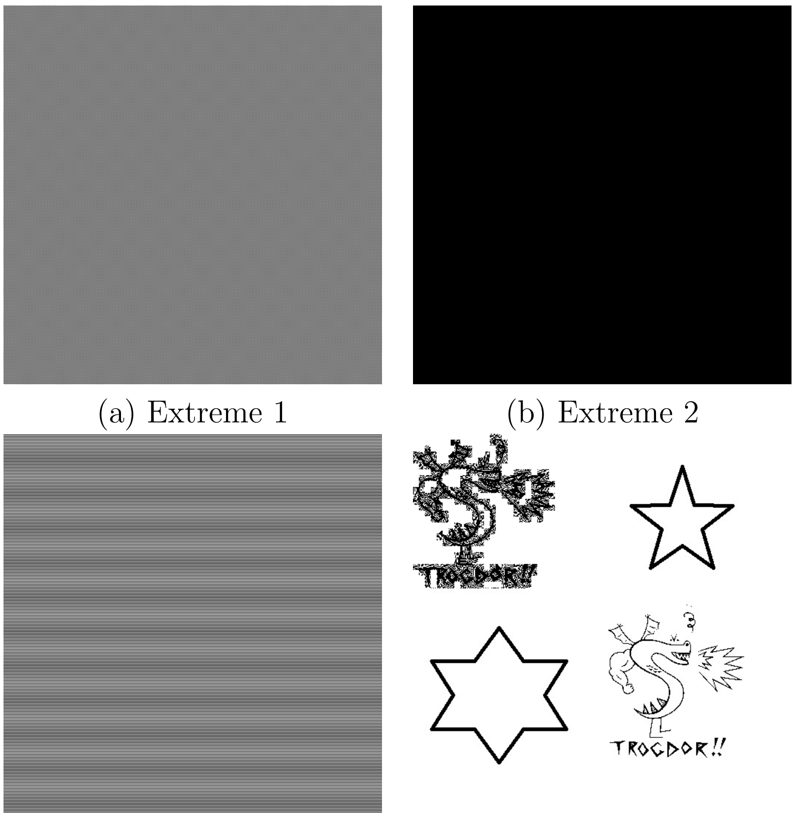
It is important to note that DCT itself does not create a compressed or lossy compression (unless the accuracy of the trigonometric function are approximate), the quantisation of the coefficients and rouding combined with some form of simple compression such as RLE and commonly combined with entropy encoding result in compression.

2.2.4 Discrete Wavelet Transform

The DWT works the same as the human eye and in the same way that the brain processes an image. In other words, luminance (brightness) or low-frequency components are more important than chrominance (colour difference) or high-frequency components to provide a fine-quality image. Taking advantage of this feature, many applications use the multi-resolution capability of DWT to decompose an image into high- and low-frequency sub-bands or “Chroma sub-sampling”. DWT also has the ability to localise finite signals such as images in both frequency and time domains simultaneously. It transforms a discrete signal from the time domain to the time-frequency domain. The transformation product is a set of coefficients organised in a way that enables not only a spectrum analysis of the signal, but also the spectral behaviour of the signal in time. Compared to traditional transforms such as the Fourier transform, the DWT yields a higher compression ratio and better visual quality [27, 45].

2.3 Performance Comparison and Applications of Common Compression Algorithms

The performance of selected compression algorithms is compared for four monochrome $N \times N$ test images shown in Figure 2.2 using Matlab simulations. All images are 65536 bytes in raw form. The peak-signal-to-noise-ratio (PSNR), which is the difference (error) between original and compressed image, is used to evaluate quality of compressed images. Matlab benchmark and workstation specifications are provided in Appendix A.



(a) Extreme 1 (b) Extreme 2

(c) Extreme 3 (d) Normal

Figure 2.2: Test images: (a) a solid 0101 extreme, (b) a solid 1111 extreme, (c) one row 1111 and one row 0000 alternately, (d) a normal image.

2.3.1 Performance of RLE Algorithm on Test Images

The RLE encoding traverses through the test images linearly, therefore the processing delay is $O(N)$ clock cycles. The decoding process is also performed linearly through the RLE encoded data, which populates an empty matrix. This clearly shows the effect of the RLE and its performance on different test images. Table 2.1 shows the results produced for encoding and decoding the test images using RLE. The Peak Signal-to-Noise Ratio (PSNR) is an infinite value as the noise is zero.

2.3.2 Performance of Entropy Coding on Test Images

Canonical Huffman coding is normally selected over Arithmetic coding due to its popularity and efficiency. The PSNR is omitted from the results because this method is lossless and no predictive algorithms are used. Table 2.2 shows the results of encoding

Table 2.1: The RLE Algorithm: results for encoding and decoding the test images

Test image	Encoded size (*bytes*)	CR	Encoding delay (*s*)	Decoding delay (*s*)
Extreme 1	13557	0.2069	0.1032	4.266
Extreme 2	199	0.003	0.0028434	0.0021296
Extreme 3	212	0.0032	0.0028434	0.0021296
Normal	7959	0.1214	0.0071	0.1976

Table 2.2: The Canonical Huffman algorithm: results for encoding and decoding the test images

Test image	Encoded size (*bytes*)	CR	Encoding delay (*s*)	Decoding delay (*s*)
Extreme 1	410	0.0063	1.259	6.8752
Extreme 2	179	0.0027	4.6039	6.8689
Extreme 3	311	0.0047	1.293	6.8693
Normal	6744	0.1029	1.2389	6.7013

and decoding the test images using the Huffman algorithm. The compression delay for Extreme 2 is due to Matlab's own functions. Another reason is that the operation have caused it to perform extra functions with every iteration.

2.3.3 Performance of DCT Algorithm on Test Images

Three versions of the DCT algorithm are simulated on test images: the standard DCT that requires $O(N^4)$ clock cycles, the FFT-based DCT, and the AAN direct-matrix-manipulation DCT.

The direct matrix manipulation method uses the precalculated matrix solution created by Arai, Agui and Nakajima's (AAN) matrix. The advantage of the AAN-based DCT is that it uses the least number of multiplications to produce the DCT encoded image. The decoding operation for both FFT-based and AAN-based DCTs can be performed with the inverse of the DCT algorithm, which is inefficient, or by reversing the encoding operations. Quantisation of test images is performed with a simple quantisation matrix approved by JPEG. The resulting PSNR is 356.4725, which is very high for a lossy compression algorithm. This result is due to the use of the same quantisation matrix for all test images. Tables 2.3 and 2.4 show the results of encoding and decoding the test images using DCT algorithms.

TABLE 2.3: The DCT variant's encoding delay (s) over the test images

Test image	Standard DCT	FFT-based DCT	AAN-based DCT
Extreme 1	4.6382	0.0265	13.8028
Extreme 2	4.4732	0.0238	13.6482
Extreme 3	4.4589	0.0242	13.6528
Normal	4.6151	0.0258	14.3034

TABLE 2.4: The DCT variant's decoding delay (s) over the test images

Test image	Standard DCT	FFT-based DCT	AAN-based DCT
Extreme 1	6.8651	0.32267	13.6348
Extreme 2	6.3974	0.3059	13.1631
Extreme 3	6.4022	0.3021	13.1629
Normal	6.7578	0.3249	14.2998

2.3.4 Performance of DWT Algorithm on Test Images

The objective of the DWT algorithm is to decompose the input signal into approximation and detail coefficients, and analyse them separately. Compression occurs because DWT distributes the majority of the coefficients into a few large coefficients. Quantising and Entropy coding (Huffman or Arithmetic coding) of the resulting data provide the actual compression.

Lossy types of DWT are created by quantising negligible values and smoothing the wavelet, which results in faster decompression and a greater overall compression ratio with Entropy encoding but with less detail [44, 46]. A threshold of 15 is used in quantisation while increasing the threshold to 20 has often been used in removing noise. A PSNR value of 67.4630 is received for both transforms. Table 2.5 shows the

TABLE 2.5: The DWT encoding and decoding delay for the test images

Test images	Db4		Haar	
	Encoding	Decoding	Encoding	Decoding
Extreme 1	2.4293	3.7559	2.3467	3.6495
Extreme 2	0.8322	1.7434	2.2848	3.6271
Extreme 3	0.8438	1.734	2.3486	3.7054
Normal	1.0459	2.1477	2.3509	3.6352

Table 2.6: Performance of RLE over standard DCT coefficients

Test image	Encoded size (*bytes*)	Compression ratio	Compression delay (*s*)	Decompression delay (*s*)
Extreme 1	203	0.0031	4.6626	7.4238
Extreme 2	199	0.0030	4.5032	7.0362
Extreme 3	221	0.0034	4.9643	7.0409
Normal	3425	0.0523	4.9935	7.3592

Table 2.7: Performance of RLE over FFT-based DCT coefficients

Test image	Encoded size (*bytes*)	Compression ratio	Compression delay (*s*)	Decompression delay (*s*)
Extreme 1	203	0.0031	0.0499	0.9887
Extreme 2	199	0.0030	0.0468	0.9775
Extreme 3	221	0.0034	0.0472	0.9739
Normal	3425	0.0523	0.0483	0.9872

DWT encoding and decoding delay for Daubechies (Db4) and Haar wavelets.

It is noteworthy that Daubechies and Haar show the same encoding/decoding delay, and that the Daubechies wavelet is faster than the Haar for all the test images (approximately 50% faster encoding/decoding delay) except for Extreme 1.

2.4 Overall Performance of Selected Algorithms

In this section, the performance of the selected image-compression algorithms from the overall system viewpoint is investigated.

2.4.1 Performance of RLE Over DCT Coefficients

The RLE algorithm is performed over the DCT coefficients generated in Section 2.3.3. Simulation results show that RLE is very consistent and performed well with DCT as expected. As expected, the RLE algorithm operated very fast. Tables 2.6 to 2.8 show the simulation results.

Table 2.8: Performance of RLE over AAN-based DCT coefficients

Test image	Encoded size (*bytes*)	Compression ratio	Compression delay (s)	Decompression delay (s)
Extreme 1	219	0.0033	13.84012	14.3623
Extreme 2	231	0.0035	13.6825	13.8362
Extreme 3	238	0.0036	13.6893	13.8459
Normal	3318	0.0506	14.3410	14.9684

Table 2.9: Performance of Huffman coding over Db4 coefficients

Test image	Encoded size (*bytes*)	Compression ratio	Compression delay (s)	Decompression delay (s)
Extreme 1	5530	0.0844	3.6229	10.5125
Extreme 2	81	0.0012	2.0174	8.5923
Extreme 3	79	0.0012	2.0496	8.3264
Normal	3004	0.0458	2.2034	8.7484

2.4.2 Performance of Huffman Coding Over DWT Coefficients

Performing Huffman coding over the wavelet coefficients of Section 2.3.4 produced the simulation results of Tables 2.9 and 2.10.

2.5 Discussion of Simulation Results

Comparing the simulation results on the test images shows that some results are consistent with the existing literature, while some results are slightly unexpected. Tables

Table 2.10: Performance of Huffman coding over Haar coefficients

Test image	Encoded size (*bytes*)	Compression ratio	Compression delay (s)	Decompression delay (s)
Extreme 1	5524	0.0843	3.5928	10.4962
Extreme 2	5531	0.0843	3.5374	10.4527
Extreme 3	5527	0.0843	3.5587	10.4697
Normal	4868	0.0743	3.5622	10.4684

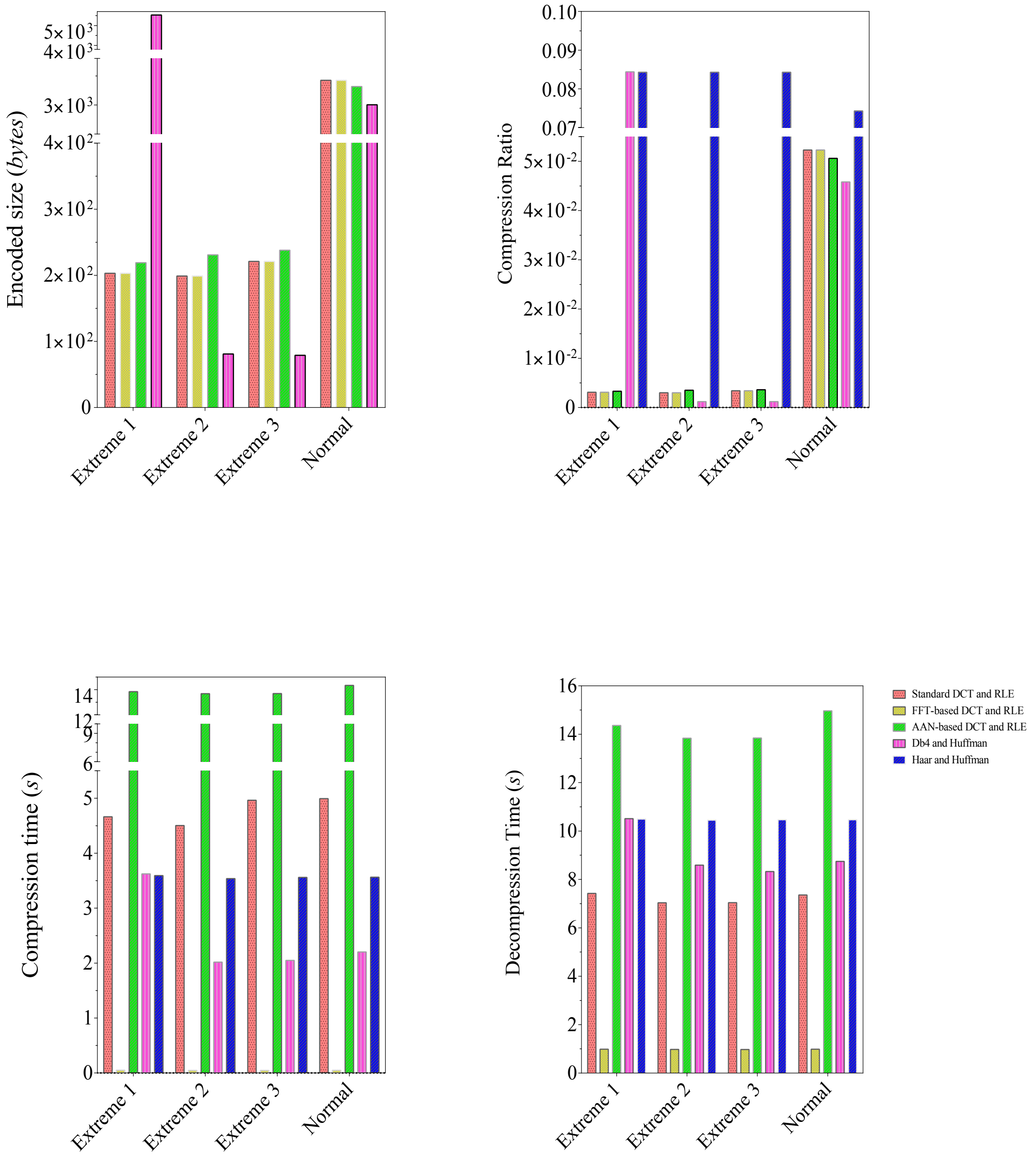


Figure 2.3: Overall performance of selected algorithms

2.6 to 2.10 are summarised in charts in Figure 2.3.

All test images except Extreme 2 were very consistent when coded with Huffman. It is believed that it is the dictionary/probability table creation process that created the unexpected compression delay.

The AAN-based DCT produced slow results in Matlab. It was not expected to produce results slower than the standard DCT. One reason is the in-built optimised Matlab sine and cosine functions in the standard form of DCT.

It was expected that the Haar and Daubechies wavelets would have slightly different file sizes when compressed with entropy. However, Extreme 1 (where the data is always different) generated an almost identical file size. The DWT algorithm performed slower than the FFT-based DCT, which is due to Matlab optimisation of the FFT and matrix specific operations.

Comparing RLE and Huffman coding shows that the simplicity of RLE allows better performance in optimum situations, while Huffman coding is more consistent with different test images as expected.

Performing RLE on AAN-based DCT created a different level of compression compared to other DCT variants. It performed marginally better with random data than with sorted data. Although the difference is minor, it is interesting that it performed almost equally with Extreme 2 and Extreme 3, where the FFT-based DCT and standard DCT performed almost identically to RLE compression.

At the end of this section, based on the simulation results, DWT is selected as the transformation to design the proposed digital image processor. It outperformed standard and AAN-based DCT on encoding delay. The performance of Huffman coding over Db4 DWT coefficients shows low compression and decompression delay and small encoded size. Another reason for choosing DWT over DCT is that the DCT algorithm uses complex cosine operations, but DWT can be implemented using simple operations. The Discrete Wavelet Transform (DWT) has already proven to be the most widely used group of the wavelet family, and has been extensively used for image compression due to its multi-resolution features and its ability to localise finite signals. Examples of successful applications of DWT in digital image compression and noise reduction applications are the FBI wavelet scalar quantisation and the JPEG2000 still-image compression standard [41, 47, 48].

2.6 Convolution vs. Lifting

A greyscale image is a two-dimensional array (matrix) $M \times N$, where each index represents the brightness level of that point (pixel) in the image.

To process an image with a compression algorithm, it should be convolved or lifted by filter coefficients. Convolution-based filtering is the most common filtering scheme, that performs convolution of the sequence to be transformed (image) and the transform (filter or mask). Convolution-based image processing can be applied either directly like the Fourier transform or indirectly such as the Hartley transform.

The transforms directly possessing the convolution property can be defined generically as [27]:

$$X_k = \sum_{n=0}^{N-1} X_n W^{kn} \tag{2.1}$$

where $k = 0, ..., N-1$, N is the transform length, X_n is the sequence to be transformed, X_k is the transformed sequence, and W is the transform kernel. Convolving digital images with a filter or mask results in sharpening, noise removing or edge detection in many applications [35, 49].

Lifting-based filtering splits odd and even samples followed by prediction and updating steps. The split step divides the odd and even samples into two sets of samples. Then consecutive samples are predicted. Prediction is done by taking advantage of the correlation of neighbouring samples. In the last step, new samples are updated by previous samples [50].

Figure 2.4 shows the lifting scheme. The differences between the actual samples and the prediction of input data X is calculated $d = (b - a)$, and stored (updated) in b. An average, s, is then computed using a and newly computed difference, as $s = (a + d)/2$. Therefore, the lifting scheme requires only integer add and shift [51]. The inverse transform can be performed by inverting the signs of polynomials and reversing the algorithm stages [52].

The lifting scheme helps to optimise the speed and area requirement of image-compression algorithms [53–56]. It uses fixed-point arithmetic instead of more costly floating-point mathematical operations of convolution-based processing. In hardware perspective, using fixed-point arithmetic saves on design complexity.

Lifting-based image processing is preferred in some studies. Authors in [53] presented techniques for improving the performance and lowering the power consumption of image processors by employing lifting image filtering. In [54] a lifting scheme has

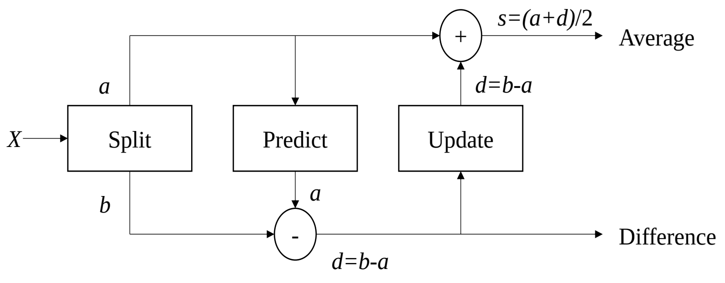


FIGURE 2.4: Lifting scheme with three steps: split, predict and update

TABLE 2.11: Forward filter coefficients of lifting-based LeGall53

x_{2i}	s_i
x_{2i+1}	d_i
$d_i - (s_i + s_{i+1})/2$	d_i
$s_i + (d_{i-1} + d_i)/4$	s_i

improved the performance of a 2D DWT image processor. A scalable architecture of different DWT families using a lifting scheme has been proposed in [55]. The performance of the convolution- and lifting-based DWT has been compared in [56], which showed that using a lifting DWT for digital image processing reduces the complexity of operations and improves the simplicity of design by saving on computation steps. Despite the above mentioned studies, the authors in [52] doubted the lifting advantages in VLSI implementations, and argued that there is no indication that lifting reduces the hardware complexity.

2.6.1 Lifting-Based DWT Image Compression

Convolution-based image processing is a traditional method of image-compression implementation. For further clarification about lifting features, the wavelet transform LeGall53 is implemented using a lifting scheme.

The original highpass and lowpass LeGall53 analysis filters have $5 + 3 = 8$ coefficients, whereas an implementation with the lifting scheme has only $2 + 2 = 4$ filter coefficients [57]. Tables 2.11 and 2.12 show forward and reverse filter coefficients, where x is the input data, s is the average or low-frequency coefficient, and d is the difference or high-frequency coefficient.

Table 2.12: Reverse filter coefficients of lifting-based LeGall53

$s_i - d_i/2$	s_i
$d_i + s_i$	d_i
s_i	x_{2i}
d_i	x_{2i+1}

Wavelet-based image compression is inherently a multi-resolution signal analyser, and variable levels of compression can be easily achieved. The number of filtering levels results in different numbers of sub-bands. If there are $2k$ elements in the row or column, then k lowpass and k highpass coefficients will be produced after each level of filtering. The analysis filter equations are shown in (2.2) and (2.3):

Highpass coefficients:

$$g(k) = 2.x(2k+1) - x(2k) - x(2k+2) \tag{2.2}$$

Lowpass coefficients:

$$h(k) = x(2k) + g(k-1) + g(k)/8 \tag{2.3}$$

where $g(k)$ is the k^{th} highpass coefficient, $h(k)$ is the k^{th} lowpass coefficient, and $x(k)$ is the input data at the k^{th} position. The synthesis filter equations are shown in (2.4) and (2.5).

Even samples:

$$x(2k) = h(k) - (g(k-1) + g(k+1)/8) \tag{2.4}$$

Odd samples:

$$x(2k+1) = g(k) + h(k) + h(k+1)/2 \tag{2.5}$$

A pseudocode for the lifting scheme is developed in Table 2.13.

The lifting pseudocode is implemented using a Maple® worksheet for a 256×256 phone.jpg image. Package *ImageTools* performs various functions of image processing. By invoking this package, both greyscale and colour images are presented as arrays of 64-bit-hardware floating-point data.

```
with(ImageTools):
A1 := Matrix(ToGrayscale(Read(cat(kernelopts(datadir), phone)))):
```

The procedure to perform the transform:

```
T:=proc(x,n, st, off1):
```

Table 2.13: Pseudocode of the lifting scheme

```
Start
Get input parameters: image, decomp level
Set $i := 0$;
Set $mid = (n/2) - 1$;
while (the end condition is not met)
Split the image pixels:
$s(i+1) := x(off1 + (2*i*st) + 1)$;
$d(i+1) := x(off1 + (2*i*st) + st + 1)$;
Predict the highpass and lowpass coefficients:
$d(i+1) := d(i+1) + d(i+1) - s(i+1) - s(i+1+1)$;
$s(i+1) := s(i+1) + ((d(i-1+1) + d(i+1))/8)$;
Update the approximation signal:
$x(off1 + (i*st) + 1) := s(i+1)$;
$x(off1 + ((i+mid+1)*st) + 1) := d(i+1)$;
i := i + 1;
end
```

Using pseudo code in Table 2.13, the transform is as below:

```
for i from 0 by 256 to (nt-1)*256 do:

imgT:= T(A1,nt, ROW, i):

end do:

for i from 0 by 1 to nt-1 do:

imgT:= T(A1, nt, COL, i):

end do:

View(Create(imgT));
```

Figure 2.5 shows the original and decomposed image after one level of DWT using the lifting scheme in Maple®.

In the lifting scheme, the analysis and synthesis filters work in the reverse order, hence two different VLSI realisations are needed,a disadvantage in the perspective of this hardware project. Another problem with lifting is that it runs sequentially and fails to increase the speed of operation. The throughput of the lifting scheme is less than half that of parallel working convolution-based methods.

Convolution-based filtering, however, has a number of attractive features, such as no image blurring or spatial dislocation in multi-resolution analysis. This thesis will use convolution-based filtering for the proposed image processor.

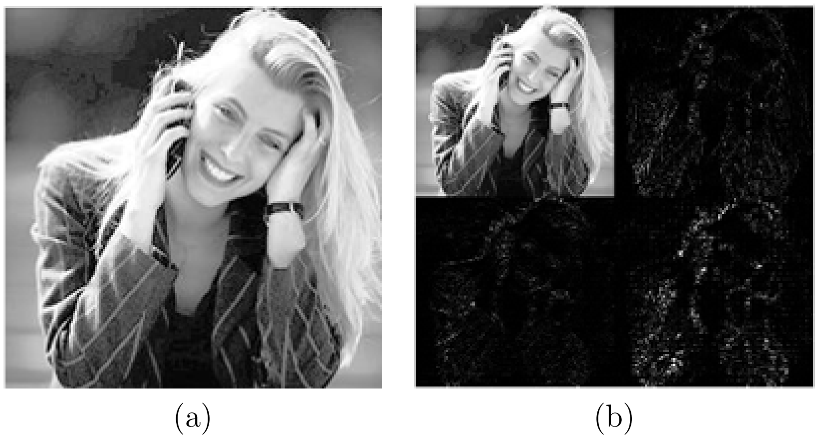


Figure 2.5: Lifting-based image compression: (a) Original image "phone.jpg" 256 × 256, (b) One level of lifting decomposition

2.7 Chapter Summary

In this chapter, image-compression algorithms have been studied, and common algorithms are compared and simulated using Matlab. Subsequently, simulation results are discussed, and potential improvements and suitable applications for each algorithm are provided.

RLE performed lossless data compression, and is suggested for palette-based icons. The wavelet transform outperformed DCT. Huffman coding is suggested as a "back-end" to other compression methods.

Convolution and lifting filtering schemes are suggested as the main methods to implement image-compression algorithms.

Finally, convolution-based DWT is selected because it overcomes the convolution-based sub-band filtering problem that increases the coefficient length and causes data expansion and boundary artifacts. CDF97 is very well-known for it's compatibility of symmetry and exact reconstruction. Furthermore, its symmetrical features and four vanishing moments makes it suitable to distinguish a smooth signal in gray scale images.

Publications pertaining to this chapter:

- Azadeh Safari and Yinan Kong. *The application of lifting in Digital Image processing*, Advances in Mechanical and Electronic Engineering, Lecture Notes in Electrical Engineering, Volume 178, 2013, pp. 449-453, Springer, 2013.

- Azadeh Safari, Fujimi Bentley, and Yinan Kong. *Operational Capability and Suitability of Image Compression Methods for Different Applications*, CCECE 2014, Ontario, Canada 4-7 May, pp.875-880, 2014.

3

Discrete Wavelet Transform for Image-Processing Applications

Various image-compression algorithms and schemes have been discussed in Chapter 2. There have been methods used to divide an input signal into various components in the frequency domain. Fourier analysis is among the most common approaches. The Fourier transform converts a signal from the time/spatial domain to the frequency domain. This is done by breaking down the signal into sinusoids of different frequencies but of infinite duration. Using the Fourier transform, time/spatial information is lost after transforming to the frequency domain, i.e. it can be said that there was a special event but not when, or in the case of images, where it occurred. One approach to overcome this shortcoming is the Short-Time Fourier Transform (STFT); another one is the wavelet transform [58].

The STFT approach applies the Fourier transform, not to the whole signal at once but to small sections of the signal. Thereby, it provides some information about the frequency and the time. The sections are created by a window function. The size of the window is fixed and determines the resolution. A wide window allows good frequency, but poor time, resolution. A narrow window has good time, but poor frequency, resolution.

There is another approach that can be used to divide signal data in the frequency domain, which is wavelets. Wavelets transform the signal to the frequency domain using small waveforms (wavelets) as base functions instead of sinusoids. Wavelets are transient waveforms of finite duration defined by mother wavelet function $\psi(t)$ and scaling function $\phi(t)$. The mother wavelet is scaled and shifted during the transformation. The wavelet and scaling functions are shown in (3.1) and (3.2).

$$\psi_{j,k}[t] = 2^{1/2} \sum_{k} d_{j,k} \psi[2^j t - k] \tag{3.1}$$

$$\phi_{j,k}[t] = 2^{1/2} \sum_{k} c_{j,k} \phi[2^j t - k] \tag{3.2}$$

where d_j and c_j are the wavelet and scaling coefficient at scale j [59]. The wavelet and scaling functions associate with highpass (g) and lowpass (h) filter coefficients in filter banks, respectively.

The wavelet transform is called a *multi-scale* or *multi-resolution* approach. It analyses the signal at different scales and allows a good time resolution for high frequencies (=low scale) and good frequency resolution for low frequencies (=high scale) [60].

As mentioned before, wavelets are a multi-scale approach. Hence, for a wavelet a pseudo-frequency F_a could be calculated, and a purely periodic signal of frequency F_c is associated with a given wavelet:

$$F_a = \frac{F_c}{a \times \Delta} \tag{3.3}$$

where a is the scale, Δ is the sampling period and F_c is the centre frequency. Figure 3.1 shows wavelet db7 (blue) and a centre-frequency-based approximation.

The order of wavelet transforms are usually determined by the number of vanishing moments of the analysis wavelet. In Daubechies db$A$ wavelets, A refers to the number of vanishing moments. Number of vanishing moments of scaling and wavelet functions is significant in smoothness of wavelets [61]. A more smooth function has more vanishing moments [62].

3.1 Discrete Wavelet Transform

In DWT, an image is decomposed by passing through an analysis filter bank. The analysis filter bank decomposes the image to lowpass and highpass filter coefficients

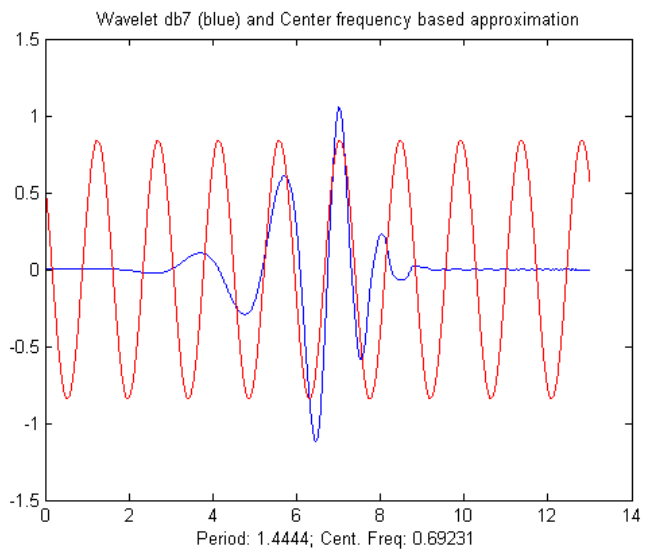


Figure 3.1: Wavelet db7 (blue) and a centre-frequency-based approximations

and "decimate them by two". The decimated outputs constitute the approximation (L) and the detail (H) signals. Figure 3.2 shows a one-dimensional DWT splitting an input signal $F(x, y)$ into L and H.

In a two-dimensional (2D) DWT, a one-level DWT is performed on the rows and columns of a 2D input signal (image), and generates an approximation signal (LL) and detail signals (LH, HL and HH). If further decomposition is required, the previous level's approximation signal (LL) becomes the next-level input [36, 60, 63]. Figure 3.3 shows how 2D DWT decomposes an input signal to LL, LH, HL and HH data sets. Figure 3.4 shows the arrangement of the data sets in the compressed image. The effect of one and two levels of 2D DWT on a binary image are shown in Figure 3.5.

DWT can be implemented using the pyramid algorithm (an octave-band filter bank with j levels) in the multi-resolution analysis framework. The level j specifies the scales of the wavelets $a = 2^j$. For a signal sequence $x(n)$, the approximation a_n^i and detail signals d_n^i at level j are defined using (3.4) and (3.5), respectively [22, 64]:

$$a_n^i = \sum_{k=0}^{n-1} h_k a_{2n-k}^{(i-1)} \tag{3.4}$$

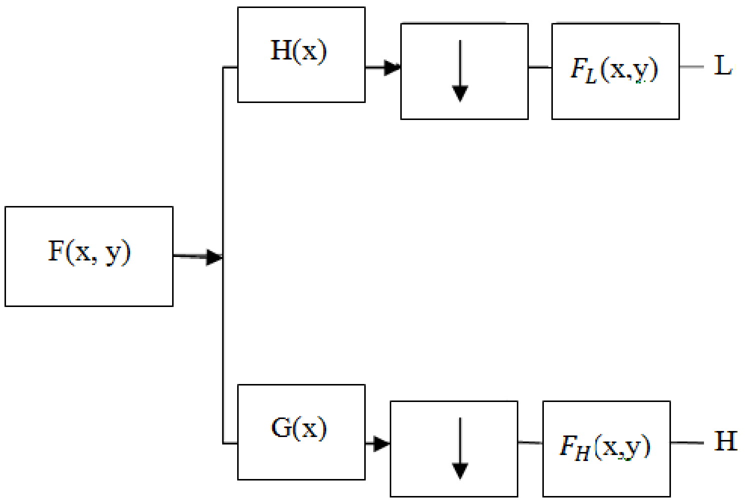


Figure 3.2: One-dimensional DWT

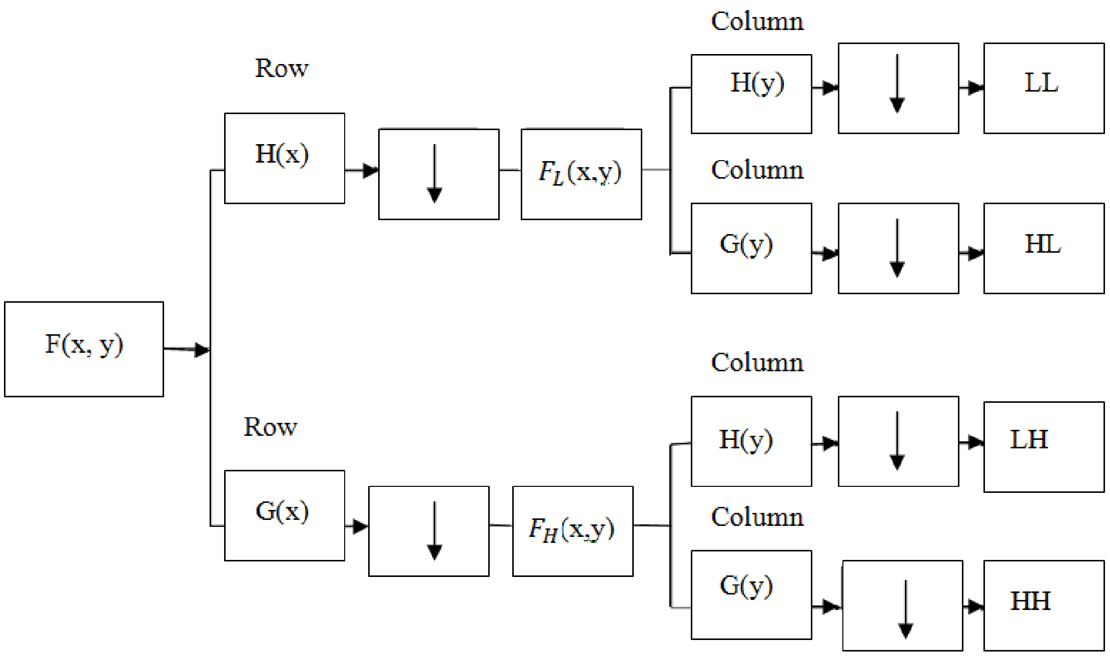


Figure 3.3: Two-dimensional DWT

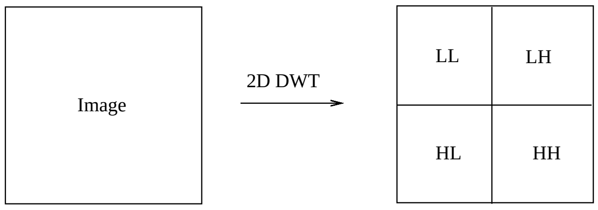


Figure 3.4: Decomposition of an image using 2D DWT to LL, LH, HL and HH data sets

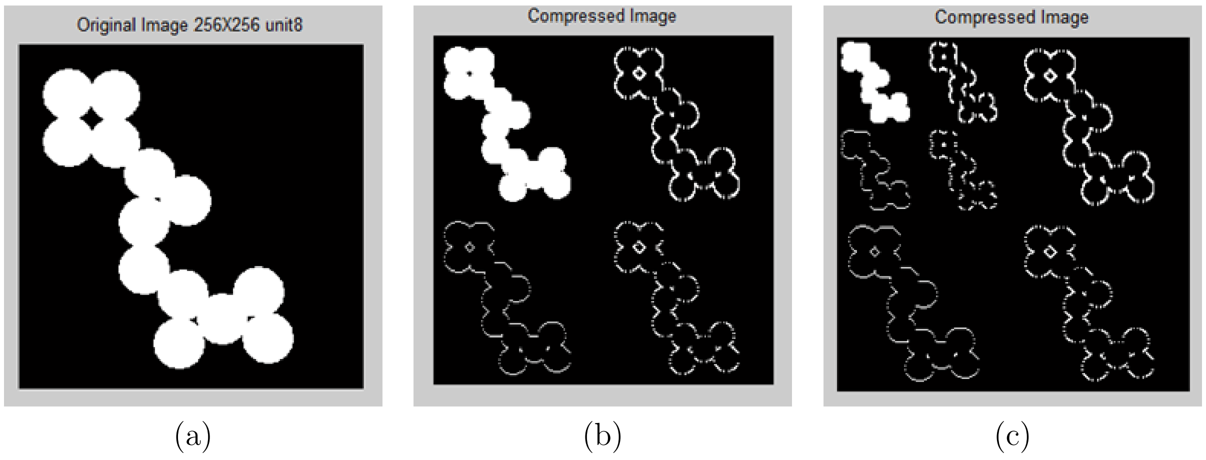


Figure 3.5: Decomposition of a binary image using 2D DWT: (a) Original image "circles.png" (256 × 256), (b) Image at level one, (c) Image at level two

$$d_n^i = \sum_{k=0}^{n-1} g_k a_{2n-k}^{(i-1)} \tag{3.5}$$

where $i = 1, 2, ..., j$, and h_k and g_k are lowpass and highpass coefficients selected based on the chosen wavelet family, n is the filter length, k is the coefficient length.

3.2 Orthogonal and Bi-Orthogonal Wavelets

The DWT can be classified according to the wavelet properties. One such classification is orthogonal or bi-orthogonal, depending on the relation between the analysis and synthesis filter banks. When the analysis and synthesis filters are transposes as well as inverses of each other, the filter bank is orthogonal, e.g. Haar and Daubechise. When they are inverses, but not necessarily transposes, the filter bank is bi-orthogonal, e.g. CDF97 and LeGall [65].

There is a vast use of orthogonal and bi-orthogonal wavelets in various fields and applications. Some applications fit well to orthogonal wavelets while others prefer bi-orthogonal wavelets [3, 66, 67]. However, fewer studies have surveyed the properties of each family in real-time processing applications. This is done in the subsequent sections.

3.3 Orthogonal DWT

The wavelet transform decomposes a signal in terms of a set of basis functions, which are localised both spatially and spectrally. In the case of orthogonal wavelets, this decomposition can be performed by quadrature mirror filters (QMF). Usually, in QMF schemes the forward transform is computed by a hierarchical arrangement of g, h filter pairs and decimators, and the inverse transform via $\bar{g}$, $\bar{h}$ filter pairs and adders. Orthogonality of filter banks halves the number of multipliers and adders at each level of decomposition and reconstruction; hence they require less memory space. Equations (3.6) and (3.7) show the filter relations in orthogonal wavelets [22]:

$$h_k = (-1)^{k+1} g_{n-k-1} \tag{3.6}$$

$$\bar{h}_k = (-1)^{k+1} \bar{g}_{n-k-1} \tag{3.7}$$

where $k = 0, 1, ..., N-1$, g_k and $\bar{g}_k$ are highpass, and h_k and $\bar{h}_k$ are lowpass, decomposition and reconstruction filter coefficients, respectively.

Orthogonal wavelets have non-linear and balanced frequency responses. They are useful in noise removal from audio signals and in data compression due to their use of overlapped windows and a high-frequency coefficient spectrum, which reflects all high-frequency changes [7].

In [67], the authors employed the orthogonal wavelet for image compression because of the excellent spatial and spectral locality features of orthogonal families. Implementing and comparing the Haar and Daubechies wavelets using FPGA technology has been described in [7]. It showed that the Daubechies wavelet is more efficient for audio applications than the Haar wavelet based on the results of FPGA implementation.

3.3.1 Haar Wavelet Transform

The Haar transform is the oldest and simplest possible wavelet, proposed by Alfred Haar in 1909. Like all wavelet transforms, the Haar transform decomposes a discrete signal into two sub-signals of half the length. It actually serves as a prototype for all other wavelet transforms. The Haar transform is one of the fast and simple family of wavelets which can be used in memory-efficient designs. It is the only orthogonal wavelet family that is exactly reversible without edge effects, a problem with other orthogonal wavelet transforms [7].

The Haar transform also has limitations, which can be a problem with some applications [68]. The Haar window is only two elements wide. If a big change takes place between two successive elements, the change will not be reflected in the high-frequency coefficients. So, the Haar transform is not useful in compression and noise removal of audio signal processing. The Haar wavelet is a square wave, which is equivalent to a "sum and difference" transformation. The Haar lowpass and highpass filter coefficients are shown in Table 3.1 [69].

TABLE 3.1: Haar lowpass and highpass filter coefficients [7]

h_0	h_1	g_0	g_1
0.5	1	1	0.5
0.5	-1	1	-0.5

The Haar scaling function and mother function are presented in (3.8), (3.9) and (3.10). The mother wavelet and scaling functions of the Haar transform are shown in Figure 3.6.

$$\phi(x) = \begin{cases} 1 & 0 \leq x < 1 \\ 0 & otherwise \end{cases} \tag{3.8}$$

$$\Psi(x) = \phi(2x) - \phi(2x-1) \tag{3.9}$$

$$\Psi(x) = \begin{cases} 1 & 0 \leq x < 1/2 \\ -1 & 1/2 \leq x < 1 \\ 0 & otherwise \end{cases} \tag{3.10}$$

3.3.2 Daubechies Wavelet Transform

The Daubechies wavelets are a family of orthogonal wavelets developed by Ingrid Daubechies [64]. The Daubechies wavelet extends the Haar wavelets by using longer

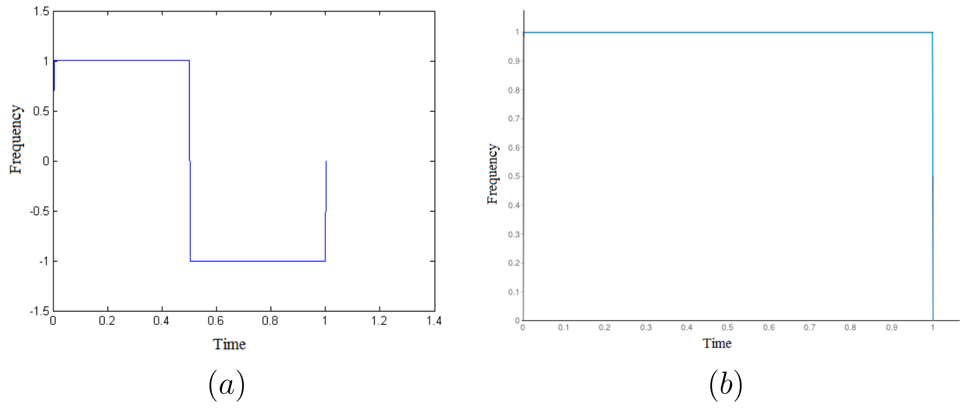


FIGURE 3.6: Haar transform: (a) Mother wavelet function, (b) Scaling function (Figure is generated using Matlab)

filters to produce smoother scaling functions. It has a higher number of vanishing moments, which allows better compression of a signal. It is important to note that the Haar wavelet is significantly simpler to encode but has approximately the same complexity as the Daubechies wavelet [46].

In the db(n) family, n is the number of vanishing moments, and the resulting filter has $2n$ filter taps. Figures 3.7 and 3.8 show the wavelet and scaling functions of db4 and db8, respectively.

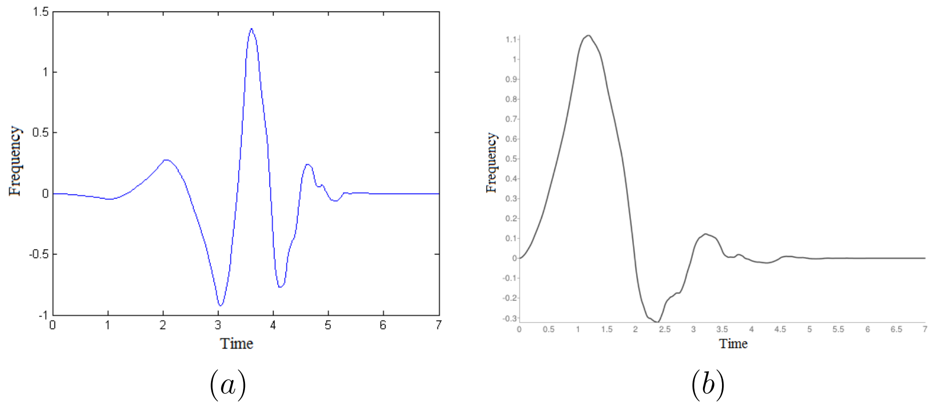


FIGURE 3.7: db4: (a) Wavelet function, (b) Scaling function (Figure is generated using Matlab)

The coefficient relations of orthogonal wavelets in (3.6) and (3.7) can be substituted in approximation and detail sequences in (3.4) and (3.5). Hence, the detail sequence

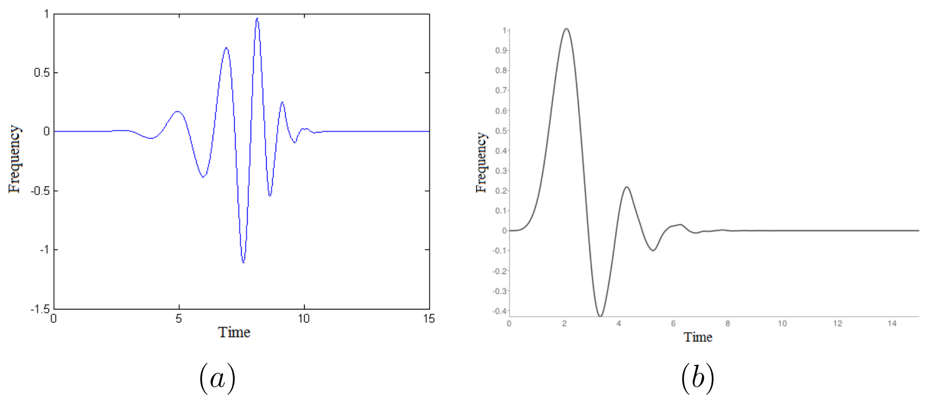


Figure 3.8: db8: (a) Wavelet function, (b) Scaling function (Figure is generated using Matlab)

will be:

$$\begin{aligned} d_n^{(i)} &= \sum_{k=0}^{N-1} (-1)^{k+1} g_{N-k-1} a_{2n-k}^{(i-1)} \\ &= \sum_{k=0}^{\frac{N}{2}-1} g_{N-2k-2} a_{2n-2k-1}^{(i-1)} - \sum_{k=0}^{\frac{N}{2}-1} g_{N-2k-2} a_{2n-2k}^{(i-1)} \end{aligned} \tag{3.11}$$

Equation (3.11) indicates that lowpass and highpass coefficients share N multipliers in alternative cycles to compute approximation and detail coefficients. The relation between filters allows the forward and inverse transforms to be calculated using shared hardware. Therefore, the Daubechies-based encoders are hardware efficient.

3.4 Bi-Orthogonal DWT

One common well-known problem in sub-band filtering is that linear convolution or filtering increases the coefficient length and causes boundary artifacts. One simple method to overcome this problem is symmetric extension and linear-phase wavelet filters. In orthogonal wavelets the same FIR filters are used for decomposition and reconstruction, so symmetry and exact reconstruction are incompatible (except in the Haar wavelet). In addition, other than Haar, none of the orthogonal filters are linear-phase. Thus, bi-orthogonal wavelets are the de-facto standard for applications that avoid data expansion. The viability of symmetric extension with bi-orthogonal wavelets is the primary reason cited for their superior performance. The bi-orthogonal DWT uses a linear-phase filter, which solves the problem of coefficient expansion, border artifacts, image blurring and spatial dislocations in multi-resolution analysis [66, 70].

In bi-orthogonal filters, two wavelets are introduced instead of just one; one wavelet is used for analysis, and the other wavelet is used for synthesis [37, 54].

3.4.1 Cohen-Daubechies-Feauveau (CDF97) Wavelet

The analysis filter bank in Cohen-Daubechies-Feauveau (CDF97) uses nine-tap lowpass filters, and seven-tap highpass filters. CDF97 has four vanishing moments for both lowpass and highpass filters, which makes it suitable for distinguishing smooth greyscale input data from rough noise. The CDF97 wavelet has been used in the JPEG2000 image-compression standard for lossy compression, and the FBI national fingerprint-storage database. Table 3.2 shows the FIR filter coefficients for the CDF97 wavelet [71].

TABLE 3.2: CDF97 analysis filter coefficients [8]

k	Lowpass Filter (h_k)	Highpass Filter (g_k)
0	0.6029490182363579	1.115087052456994
±1	0.2668641184428723	-0.5912717631142470
±2	-0.07822326652898785	-0.05754352622849957
±3	-0.01686411844287495	0.09127176311424948
±4	0.02674875741080976	

3.4.2 LeGall (b, c) Wavelet

The LeGall (b, c) is a family of bi-orthogonal wavelets in which b and c can be any positive integers whose sum is even. b and c are the number of vanishing moments of the analysis and synthesis filters, respectively. LeGall53 has been used in the JPEG2000 image-compression standard for lossless compression. The LeGall53 analysis filters coefficients are shown in Table 3.3.

TABLE 3.3: LeGall53 analysis filter coefficients

k	Lowpass Filter (h_k)	Highpass Filter (g_k)
0	6/8	1
±1	2/8	1/2
±2	-1/8	0

Table 3.4: Area, speed and power consumption of selected wavelet families

	Coefficient Bit width	Hardware utilisation (FPGA slices)	Power (mW)	Delay (ns)
CDF97	6	622	1297.84	5.551
LeGall53	3	241	1298.09	4.043
Haar	1	103	1297.25	2.048
db4	5	496	1298.03	7.732
db8	4	459	1298.29	6.189

3.5 Performance Comparison of Wavelets

We have designed and implemented selected orthogonal and bi-orthogonal DWT families using VHDL in Xilinx ISE Design Suite. For consistency, the same structure of filter banks is used for all wavelets. Details of the filter designs are provided in Section 6.2.1.2.

The wavelets are designed, and simulated using Modelsim on Virtex6 part xc6vcx75t over speed grade -2. Table 3.4 shows the area, speed and power consumption of selected wavelet families.

Table 3.4 shows that Haar is the best choice in applications with limited area concerns, followed by other orthogonal families. This is due to the strong link between the orthogonality of filter banks and hardware utilisation. The orthogonal families use the same analysis and synthesis filter banks, while bi-orthogonal filters use two separate filters. One unanticipated result is that LeGall53 bettered the orthogonal family in hardware utilisation which is due to fewer coefficient bits.

Comparing the power consumption results, Haar used the least power followed by CDF97, closely followed by db4 and LeGall53. db8 has the most power dissipation among the wavelets.

In the delay comparison, Haar is the fastest. The bi-orthogonal families are slower and other orthogonal families are slowest. The delay of LeGall53 is twice that of Harr, which makes selection easier in applications with fast processing requirements.

3.6 Synthesising Wavelets Using Design Compiler

In this section, the area, speed and power consumption of the selected orthogonal and bi-orthogonal wavelets are compared, based on Design Compiler (DC) synthesis results.

Table 3.5: Library $TCB015GHDWC$ report

Library type	Technology $TSMC150nm$
Process	General Purpose Nominal V_t
Time Unit	1 ns
Capacitive Load Unit	1.000000 pf
Pulling Resistance Unit	1 $kilo - ohm$
Voltage Unit Leakage	1 V
Current Unit	1 mA
Dynamic Energy Unit	1.000000 pJ
Power Unit	1 nW
Operating Conditions:	
Operating Condition Name	WCCOM
Library	$Tcb015ghdwc$
Process	1.00
Temperature	125.00
Voltage	1.35
Interconnect Model	Balanced_tree

The aim of this section is to serve as a convenient reference for selecting appropriate wavelets for specific applications.

To synthesise a wavelet using Design Compiler (DC), it is analysed and elaborated with a technology-independent library (GTECH).

The $TSMC150nm$ standard-cell library is used to compile wavelets and refine the timing and environmental restrictions. Table 3.5 reports the library specifications and worst-case operating conditions used to define the design environment.

For the purpose of area measurements, the top-level designs of wavelets are compiled with a “medium map effort” to execute area optimisations. Table 3.6 reports combinational, non-combinational, and total area for each wavelet family. Comparing Tables 3.4 and 3.6, the total cell areas of the designs are similar to those of the FPGA synthesis results. The results also confirm the theoretical equations in that the relations within filters reduce memory space requirements in orthogonal families.

The power consumptions of the designs are evaluated using the same libraries and operating conditions. The *segmented wire load model* mode is selected. Hence, the wire-load model of each segment of a net is determined by the design encompassing the

Table 3.6: Area comparison of the selected wavelet families (μm^2)

	Combinational	Buf/Inv	Non-combinational	Total cell
CDF97	9333.273	675.993	5016.038	14349.312
LeGall	3668.198	499.737	3440.102	7108.300
Haar	1600.819	460.339	2258.150	3858.969
db4	8941.363	564.019	3489.868	12431.232
db8	7755.264	651.110	4622.054	12377.318

Table 3.7: Top-level design power analysis

	Internal Power (mW)	Switching Power (mW)	Total Dynamic Power (μW)	Leakage Power (μW)	Total Power (mW)
CDF97	0.5823 (54%)	0.4923 (46%)	1.0745 (100%)	7.2498	1.0818
LeGall	0.3546 (48%)	0.3319 (52%)	686.4899 (100%)	3.3317	0.6898
Haar	0.2008 (57%)	0.1513 (43%)	352.0773 (100%)	1.6756	0.3538
db4	0.3369 (59%)	0.2298 (41%)	566.7487 (100%)	6.7170	0.5735
db8	0.5046 (56%)	0.4044 (44%)	908.9482 (100%)	6.2136	0.9152

segment. The total power dissipated falls into two broad categories which are dynamic power (including cell internal power and net switching power) and leakage power. The results are provided in Table 3.7 for Wire-load model: `TSMC8K_Conservative`. These results are also consistent with the results in Table 3.4, where Haar has the least, and CDF97 and db8 have the most power consumption.

The design is first mapped to gates without setting constraints. This helps determine the initial design speed. Then, the speed of the initial design is used to determine a starting-point value for constraints.

To check for timing violations and whether the designs have met the timing constraints, the data arrival time is generated, which is relative to the rising edge of the clock at time zero when the data is launched from the starting point. None of the designs have negative time slack, hence the path with the least positive timing slack has the critical path delay. The data required time, data arrival time and the slack for the critical path delay are provided in Table 3.8.

The endpoint slack histograms of selected wavelets are also shown in Figures 3.9 to 3.13 as high-level graphical descriptions of how the designs have met the timing constraints. The overall timing performance of the designs shows that CDF97 has the most critical delay paths, followed by db8. The distribution of the timing-slack values

Table 3.8: Critical path delay (*ns*)

	Data required time	Data arrival time	Slack	Timing constraints
CDF97	9.39	9.10	0.29	MET
LeGall	9.43	5.05	4.38	MET
Haar	9.43	3.48	5.96	MET
db4	9.31	7.33	1.98	MET
db8	9.43	6.84	2.60	MET

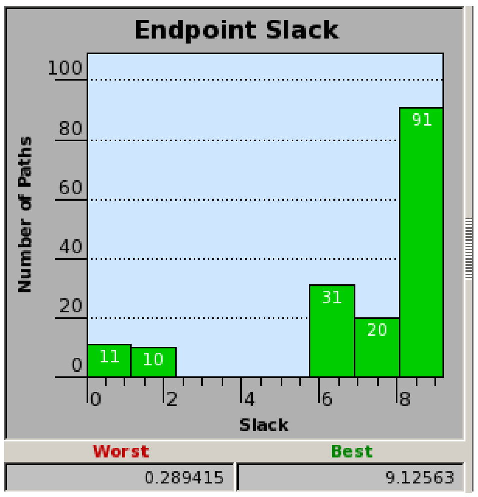


Figure 3.9: Endpoint slack histogram of CDF97 with no timing violations

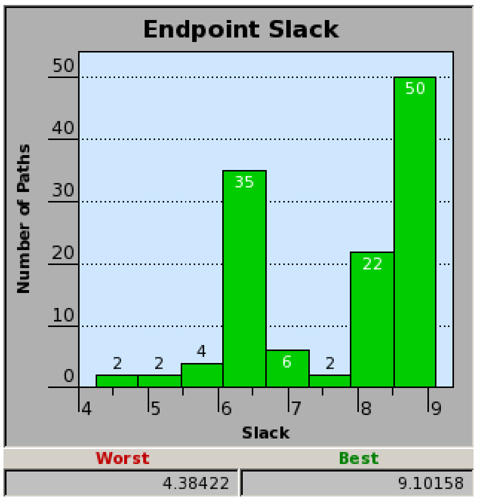


Figure 3.10: Endpoint slack histogram of LeGall with no timing violations

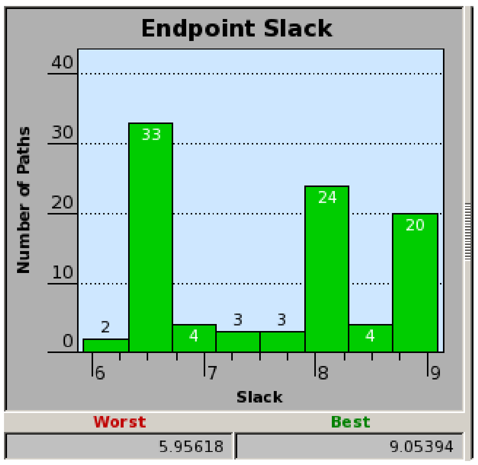


Figure 3.11: Endpoint slack histogram of Haar with no timing violations

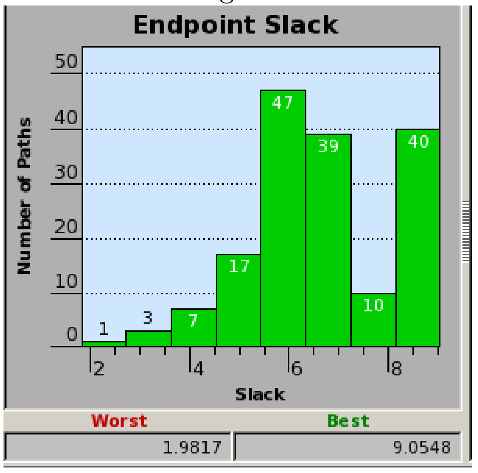


Figure 3.12: Endpoint slack histogram of db4 with no timing violations

for Haar shows that most paths have delay close to the defined timing constraint, so

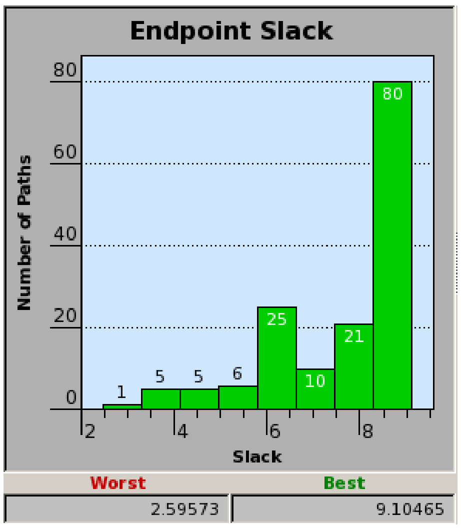


Figure 3.13: Endpoint slack histogram of db8 with no timing violations

Haar is the fastest design. These results support the results provided in Table 3.4.

3.7 Chapter Summary

Different types of predefined wavelets exist and new wavelets can be created by users. It is important to choose a correct wavelet for an application. Hence, an important question is: "When is the wavelet transform a good choice?"

The wavelet transform achieves good results if the signal is transient; for smooth signals other transforms (e.g. the cosine transform) could be better choices.

The results of both FPGA and Synopsys Design Compiler give an account of, and the reason for, the widespread use of different wavelets in various applications. Selected families from orthogonal and bi-orthogonal wavelets are provided, and properties of each family based on a comparison of the area, speed and power consumption are investigated. The results are consistent with those of VHDL implementations in literature.

The mother and scaling functions of wavelets are window-like functions in spatial and spectral domains. Hence, they are unique in localising finite signals like images. The bi-orthogonal DWT is chosen for image compression using simple, short FIR filters. CDF97 is chosen because it is symmetric and has four vanishing moments, which makes it suitable to distinguish a smooth signal in greyscale images.

Publication pertaining to this chapter:

- Azadeh Safari and Yinan Kong. *Performance comparison of orthogonal and biorthogonal wavelets using technology libraries*, The 13th International Symposium on Communications and Information Technologies, IEEE, Samui Island, Thailand, September 4-6, 2013.

4 The Residue Number System

Digital image processing involves many algebraic operations. These operations introduce delay and power-dissipation overhead to the system. Hence, we should seek methods to increase speed and decrease power by decreasing the number of algebraic operations on the input image. One way to increase the speed of operations is to use the residue number system (RNS).

Splitting large integers to small residues in RNS can significantly enhance the performance of the image processor by speeding up the operations and reducing the processing time, switching activity, storage space and dynamic power consumption. Implementing modular operations in RNS can be accomplished by simple, parallel, independent blocks. Furthermore, applying RNS provides easy error correction and detection. Since RNS is a non-positional system and modular channels have no weight information, an error in one channel does not propagate to other channels. Therefore, isolation of the faulty residues allows fault tolerance and facilitates error detection and correction.

The first documented occurrence of multiple-residue representation of a number dates back to more than 1500 years ago. Sun Tsu was a Chinese scholar who proposed using a set of smaller numbers instead of a large number [72].

The RNS is a non-weighted number system. Each number (X) in the RNS is represented as a set of the least positive remainders (residue set) when it is divided by a set of moduli (moduli set). The residue set is commonly shown as $(r_1, r_2, \ldots, r_i)$, where r_i is the i^{th} residue. The moduli set is shown as $(m_1, m_2, \ldots, m_i)$, where m_i is the i^{th} modulus, and m_i and m_j should be relatively prime to each other.

Each number (X) in RNS can be expressed as:

$$|X|_{m_i} = r_i \tag{4.1}$$

where r_i is calculated using:

$$r_i = \begin{cases} X \bmod m_i, & X \geq 0 \\ (m_i - X) \bmod m_i, & X < 0 \end{cases} \tag{4.2}$$

The RNS has a *dynamic range* which is the total number of different values that can be represented using the moduli set [72]:

$$M = \prod_{i=1}^{n} m_i. \tag{4.3}$$

In the RNS any number in the range $[0, M-1]$ has a unique set of residues. If the integer (X) is greater than $(M-1)$ or negative, then the residue set will be repeated. Hence more than one integer can have the same residue representation. It shows the importance of the pair-wise relatively prime integers, and the moduli set selection which results in the size of the dynamic range [73].

4.1 Algebraic Operations in the RNS

The RNS algebraic operations are classified into two main groups:

- The simple operations (addition, subtraction and multiplication)
- The complex operations (sign detection, division and magnitude comparison).

For the residue set of $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$, addition, subtraction and multiplication are performed independently on each digit, i.e only the i^{th} residue of the operands participates in a simple operation:

$$|XoY|_M = (|x_1 o y_1|_{m_1}, |x_2 o y_2|_{m_2}, \ldots.., |x_n o y_n|_{m_n}) \tag{4.4}$$

where (o) represents addition or multiplication. Subtraction is performed as addition of the addition inverse:

$$|X - Y|_M = (|x_1 - \bar{y}_1|_{m_1}, |x_2 - \bar{y}_2|_{m_2}, \ldots.., |x_n - \bar{y}_n|_{m_n}) \tag{4.5}$$

where $\bar{r}_i$ is the m_i complement of r_i, and should satisfy the relation $|r_i + \bar{r}_i|_{m_i} = 0$. It can be calculated using

$$\bar{r}_i = |m_i - r_i|_{m_i}. \tag{4.6}$$

Simple operations are the primary advantage of using the RNS. Modulo-dependent operations also solve the carry-propagation problem by limiting it within a single residue [74].

Division, sign detection and magnitude comparison in the RNS are considered as complex operations. Unfortunately, (4.4) does not hold for division. Dividing an integer by a divisor to give a quotient, is a complicated process. If there was an accurate way to represent the inverse of the divisor, it would be easy to multiply the inverted divisor and the dividend. However, unlike 2's-complement systems, there is no easy way to accurately represent the multiplicative inverse of a divisor in RNS, and no easy way to perform division. Division would be even more complex by a random number.

The common solution for performing complex operations is to convert residues to a weighted number system and perform the complex operations, then convert back the results to the RNS. Complex operations are the reasons that RNS has not been extensively applied [75, 76].

In spite of the great advantages of RNS-based architectures, research on this topic is in the early stages. In addition, complex operations are far more difficult and costly to implement. Also, conversion circuits from residue back to binary are complex and offset the high speed of the processor. Therefore, the RNS is mostly suggested for applications with predominantly addition and multiplication computations within a certain range of results.

4.2 Forward and Reverse Conversions

To design a system in the RNS, the first step is to convert the weighted numbers in 2's-complement into the RNS. The binary-to-residue (B/R) converters convert binary integers into the residues in the RNS. If (X) is a decimal integer and $X_{10} = (b_{n-1}, ..., b_1, b_0)_2$

is the binary representation of X, then X modulo-m_i can be calculated using:

$$|X|_{m_i} = |\sum_{i=0}^{n-1} b_i |2^i|_{m_i}|_{m_i} \tag{4.7}$$

where b_i is a bit value of either 0 or 1 in the binary number system.

After performing modular operations, the results can be converted back to the binary system using residue-to-binary (R/B) converters. Converting binary numbers to the RNS is straightforward; however, converting residues back to binary numbers is a very complicated and costly operation. The most popular method in R/B conversion is the Chinese Remainder Theorem (CRT).

4.2.1 The Chinese Remainder Theorem

The most useful tool for R/B conversion is the "Chinese Remainder Theorem (CRT)".

Theorem [10, 72, 74] Given the residue set $(x_1, x_2, ..., x_i)$ corresponding to the co-prime moduli set $(m_1, m_2, ..., m_i)$, the system has one and only one common solution modulo-M:

$$|X|_M = |\sum_{i=0}^{n-1} \hat{M}_i |\alpha_i x_i|_{m_i}|_M \tag{4.8}$$

where:

$$\hat{M}_i = \frac{M}{m_i} \tag{4.9}$$

and

$$\alpha_i = |\hat{M}_i^{-1}|_{m_i} \tag{4.10}$$

where α_i is the multiplicative inverse of $\hat{M}_i$ and should satisfy the condition $|\hat{M}_i|\alpha_i|_{m_i}|_{m_i} = 1$.

4.2.2 Multiplicative Inverse for $(2^n - 1, 2^n, 2^n + 1)$

As (4.8) shows, the multiplicative inverse is required for reverse conversion. To calculate the multiplicative inverse of moduli set $(2^n - 1, 2^n, 2^n + 1)$, (4.9) is used for $i = 1, 2, 3$:

$$\hat{M}_1 = \frac{M}{m_1} = \frac{(2^n - 1)(2^n)(2^n + 1)}{2^n - 1} = 2^n(2^n + 1) \tag{4.11}$$

$$\hat{M}_2 = \frac{M}{m_2} = \frac{(2^n-1)(2^n)(2^n+1)}{2^n} = (2^n-1)(2^n+1) = (2^{2n}-1) \tag{4.12}$$

$$\hat{M}_3 = \frac{M}{m_3} = \frac{(2^n-1)(2^n)(2^n+1)}{2^n+1} = (2^n-1)2^n \tag{4.13}$$

The condition $|\hat{M}_i|\alpha_i|_{m_i}|_{m_i} = 1$ should be satisfied. Therefore, the multiplicative inverse of moduli set $(2^n-1, 2^n, 2^n+1)$ using (4.10) is as follows:

Theorem [1, 77] Given the moduli set $\{m_1, m_2, m_3\} = \{2^n-1, 2^n, 2^n+1\}$, the following holds true.

$$\alpha_1 = |\hat{M}_1^{-1}|_{m_1} = 2^{n-1} \tag{4.14}$$

$$\alpha_2 = |\hat{M}_2^{-1}|_{m_2} = 2^n - 1 \tag{4.15}$$

$$\alpha_3 = |\hat{M}_3^{-1}|_{m_3} = 2^{n-1} + 1 \tag{4.16}$$

4.2.3 Modified CRT

The CRT limits the implementation efficiency of large dynamic ranges. In the modified CRT a set of reduced modules is used to reduce the dynamic limitation of CRT, and increase speed and efficiency [78].

$$X = x_1 + p_1|\sum_{i=1}^{m} w_i x_i^{'}|_{p_2..p_m} \tag{4.17}$$

where

$$w_1 = \frac{N_1|N_1^{-1}|_{p_1} - 1}{p_1} \tag{4.18}$$

$$x_1' = x_1 \tag{4.19}$$

and for $i = 2, 3, ..., m$:

$$w_i = \frac{N_i}{P_1} \tag{4.20}$$

$$x_i' = |N_i^{-1} x_i|_{p_i} \tag{4.21}$$

4.2.4 Mixed-Radix Conversion

Another method for R/B conversion is mixed-radix conversion (MRC).

Theorem Given the moduli set $(m_1, m_2, \ldots, m_k)$ and residue set $(x_1, x_2, \ldots, x_k)$, number (X) can be calculated as:

$$X = a_k \prod_{i=1}^{k-1} p_i + \ldots + a_3 p_1 p_2 + a_2 p_1 + a_1 \tag{4.22}$$

where (a_i) is the Mixed-Radix coefficient:
$a_1 = |X|_{p_1} = x_1$, $a_2 = |\frac{X}{P_1}|_{p_2}, \ldots,$
and $a_i = |\frac{X}{\prod_{j=1}^{i-1}}|_{m_i}$.

4.3 Moduli Set Selection and Bit-Efficiency Improvement

4.3.1 Moduli Set Selection

The general moduli set is usually shown as $(m_1, m_2, \ldots, m_i)$, where m_i is the i^{th} modulus, where for $(i \neq j)$ congruence (4.23) should hold true [73]:

$$GCD(m_i, m_j) = 1. \tag{4.23}$$

Selecting an efficient moduli set is very important in performance of a RNS-based system since an efficient moduli set for a specific dynamic range improves the bit efficiency of the system [4]. In addition, a moduli set should be selected appropriate to the application [79]. There are different moduli sets available in the literature [80–82] for various applications. Some of the most common moduli sets are [18]:

- $(2^n - 1, 2^n, 2^n + 1)$
- $(2^n, 2^n - 1, 2^{n-1} - 1)$
- $(2^n - 3, 2^n + 1, 2^n - 1, 2^n + 3)$
- $(r^a, r^b - 1, r^c + 1)$
- $(r^n - 1, r^n, r^n - 2)$
- $(r^n - 1, r^n, r^n + 1)$

- $(2^n - 1, 2^n + 1, 2^{n+1} - 1)$
- $(2^n + 1, 2^n, 2^n - 1, 2^{n+1} - 1, 2^{n-1} - 1)$

Finding the best moduli set used to be done heuristically. Yet there are some guidelines and theoretical analysis that guarantee the best selection. The following steps are the "rules" of successful moduli set selection $S = m_1, m_2, ..., m_i$ reported in [83].

1. Each pair of moduli should be relatively prime.
2. The largest modulus determines the speed of the arithmetic operations and should be selected as small as possible. Another consideration is the form of modulus, i.e modulus 16 is a better choice than the smaller modulus 13 because it is of the form 2^k and saves significantly on arithmetic operations. The best moduli set can be chosen based on the following:
   (a) There should be only one modulus of the form 2^k.
   (b) Any number of moduli of the form $2^k + 1$ is allowed unless they are from the same s_d, where s_d={all numbers of the form $F_{k,d} = 2^{2^d k + 2^{d-1}} + 1$ with $k = 0, 1, 2, 3, ...$ and $d = 1, 2, 3, ...$}.
   (c) Any number of moduli of the R_q is allowed as long as rule (2b) is met, where R_q = {all numbers of the form $M_{k,q} = 2^{2k+1}q - 1$ with $k = 0, 1, 2, 3, ...$ and q an odd prime}.
   (d) One modulus of the form $2^m - 1$ (m even) is allowed. This rule follows rule (1), that all moduli set should be pairwise prime to each other. $2^{m_1} - 1$ and $2^{m_2} - 1$ are not prime to each other (Section 5.2, Theorem 2).
   (e) Any number of moduli of the form $2^r - 1$ (r odd) is allowed. $2^{r_1} - 1$ and $2^{r_2} - 1$ are prime to each other.
   (f) Where possible decompose the selected moduli into smaller "sub-moduli", with one of the sub-moduli being of the form $2^k \pm 1$. The sub-modulus satisfies all the rules of moduli set selection.
3. Efficient conversion and inter-moduli operations should be possible. The moduli set $(2^{k_1} - 1, 2^{k_2}, 2^{k_3} - 1)$ has already solved the problem of conversions, and has simple mathematical operations.
4. The dynamic range of the moduli set should cover all the arithmetic operations.

5. The moduli set should keep the balance of the number of bits. In other words, the moduli set should avoid large gaps between the numbers of bits of different moduli.

6. Keep the number of moduli in the moduli set as small as possible.

4.3.2 Moduli Set for Video and Image Processing

The most recommended moduli set in greyscale video and image processing is $(2^n - 1, 2^n, 2^n + 1)$, called the "low cost moduli set"[23, 84, 85]. This moduli set is very popular due to its simplicity and the design efficiency of functional and conversion units [1, 80, 86].

The authors in [87] have recommended the moduli set $(2^n - 1, 2^n, 2^n + 1)$ as "the most standard and widely used" moduli set. Also, the authors in [88] have compared the low-cost moduli set with "general moduli sets" in term of speed and hardware complexity. They compared the area and speed of four general moduli sets for 8-, 16-, 32- and 64-bit ranges and concluded that the R/B converter of the low-cost moduli set "is the fastest and requires the least amount of data".

The B/R conversion of this moduli set is straightforward and the R/B conversion problem is well resolved [89]. Another reason for the popularity of this moduli set is the ease of implementing modular addition, subtraction, multiplication, and shift circuits [73, 90].

The authors in [91] argued that, for the medium dynamic range (21 bits or less), the most efficient moduli set is the low-cost moduli set. They suggested that, for large dynamic ranges (22 bits or more), the low-cost moduli set cannot be used any more, and the set should be of the form $(2^{n_1}, 2^{n_1} + 1, 2^{n_1} - 1, 2^{n_2} \pm 1, ..., 2^{n_i} \pm 1)$. Later on, they referred to [92] and said: "the upper bound of the dynamic range for using three-moduli sets is around 24 bits".

In this thesis, the low-cost moduli set is selected for $n = 8$. The dynamic range (M) has 16776960 different values or 24 bits, which is in the range suggested by [91]. The total number of bits required for different arithmetic blocks is shown in (4.24). It shows that, for the selected moduli set, each operating block should have 25 bits to avoid overflow.

$$\lceil log_2 m_1 \rceil + \lceil log_2 m_2 \rceil + \lceil log_2 m_3 \rceil = 8 + 8 + 9 = 25 \tag{4.24}$$

4.4 Scaling in the RNS

Scaling in the RNS corresponds to division of an integer (X) by a constant (k). It can be shown as

$$Y = \left\lfloor \frac{X}{k} \right\rfloor \tag{4.25}$$

where k is called the scaling factor, Y is the result of scaling X by k and $\lfloor x \rfloor$ is the floor function (the greatest integer function) [93].

There are many scaling schemes available in the literature. Some studies have chosen a constant as the scaling factor [94], while other schemes have suggested using a scaling factor co-prime to the moduli set [93]. The authors in [1] have suggested one of the moduli as the scaling factor, saying that using one of the moduli as the scaling factor reduces the complexity of inter-modulo operation. In other studies [95–97] the scaling factor has been the product of a subset of the moduli set, such as $k = \prod_{i=1}^{s} m_i$, where $(s < n)$ and n is the number of the moduli.

4.5 Residue Number System Merged With Other Number Systems

The RNS can be merged with other number systems to form new number systems such as: the logarithmic residue number system (LRNS), the polynomial residue number system (PRNS), and the one-hot residue number system (OHRNS). Each number system is suitable for a specific application, with pros and cons.

4.5.1 Polynomial Residue Number System

The polynomial Residue Number System (PRNS) is very similar to the Residue Number System (RNS). It was first introduced on $GF(2^m)$ in [98]. Generally, PRNS is employed to achieve a better performance in parallel digital signal processing [99–101]. In RNS-based systems each modular channel is a representation of a residue, however in PRNS each channel is a polynomial of residues. The Chinese Remainder Theorem (CRT) is still valid in the PRNS [102–104]. The PRNS can be used on $GF(2^m)$ for error detection. In addition to these operations, addition, subtraction and multiplication of each channel is independent and can be performed in parallel [105]. The PRNS architecture is also supported for encryption systems [106]. A set of polynomials in the binary number system are selected for each polynomial channel in PRNS. In the moduli set= $(m_1, m_2, \ldots, m_n)$, n is the number of selected channels for PRNS and d_i is the degree of m_i. In order to demonstrate the value of $GF(2^m)$ as a unique measure of the degree d_i, a product of polynomials is used which should be less than m [107]:

$$\sum_{i=1}^{n} d_i \geq m \tag{4.26}$$

If (4.26) holds true, the $P(x)$ element of the polynomial can be formatted using PRNS and can be shown as a list of the remaining:

$$\bar{P} = (p_1, p_2, ..., p_n) \tag{4.27}$$

where $\bar{P} = p(x) \bmod m_i(x)$ for $i = 1, 2, ..., n$

Equations (4.28) and (4.29) show that, in PRNS based on $GF(2^m)$, addition, subtraction and multiplication can be done in parallel.

$$A \pm B = (< a_1 \oplus b_1 > m_1, ..., < a_n \oplus b_n > m_n) \tag{4.28}$$

$$A \times B = (< a_1 \times b_1 > m_1, ..., < a_n \times b_n > m_n) \tag{4.29}$$

Addition and subtraction are performed by XOR-bit binary systems. So there is no overflow problem and the need to reduce modulo operation is eliminated. However, modulo reduction is necessary for multiplication [103, 104]. Converting from PRNS format to weighted polynomial can be performed based on the extended CRT to polynomial conversion. Mixed-Radix Conversion (MRC) is also valid for a conversion algorithm [98]. Equation (4.30) is used to convert the residue number system to the polynomial weighted number system [98, 107]:

$$p(x) = \sum_{i=1}^{n} (p_i l_i \bmod m_i) M_i(x) \tag{4.30}$$

where $M_i(x) = \frac{M(x)}{m_i(x)} = m_1 ... m_{i-1} m_{i+1} ... m_n$ and $l_i = |M_i^{-1}|_{m_i}$.

4.5.2 One-Hot Residue Number System

The OHRNS has been suggested for applications with intense algebraic operations like image processing [108]. In [109] new circuits for the OHRNS addition and subtraction using one barrel-shifter structure have been proposed. The proposed circuits have a reduced amount of hardware, and can generate the addition and subtraction results simultaneously. The authors in [110] proposed a modulo-$(r^n - 1)$ adder by combining OHRNS and Multi-Valued Logic to reduce the required number of transistors and the power consumption.

Implementing OHRNS-based systems is simple and has regular structure. Addition in OHRNS can be done by shifts and rotates [23, 111, 112]. To represent the numbers in OHRNS, (m) signals are used for (m) moduli, and the residues modulo-(m_i) are between zero and $(m_i - 1)$. At each clock cycle only one signal is high (active) and other signals are low (inactivated), where each active signal is the respective residue in that moduli set [25]. Table 4.1 shows decimal, binary and the OHRNS moduli (m_i).

Table 4.1: Decimal, binary and one-hot residue modulo-m_i

Decimal	Binary	OHRNS
0	$000\ldots00$	$1000\ldots0$
1	$000\ldots01$	$0100\ldots0$
2	$000\ldots10$	$0010\ldots0$
$\vdots$	$\vdots$	$\vdots$
$m_i - 1$	$000\ldots01$	$0000\ldots1$

The delay of operations in OHRNS is equal to delay of a transistor. However, they are not usually recommended for large moduli sets since the number of transistors increases exponentially with number of moduli set [113].

4.6 Chapter Summary

The residue number system is a non-weighted number system that enhances system performance by high speed parallel and fault tolerant operations. In a RNS-based system, moduli set and dynamic range selection have direct effect on the speed of the process and the implementation of efficient VLSI circuits for B/R and R/B converters. In this thesis, the low-cost moduli set $(2^n - 1, 2^n, 2^n + 1)$ is selected for designing a digital image processor. The greyscale image pixels are between [0,255], hence $n = 8$ will cover all possible inputs. Each operating block should have 25 bits to avoid overflow. Numbers in this system can be efficiently scaled by any of the moduli; however, scaling by 2^n is simpler than scaling by $(2^n - 1)$ or $(2^n + 1)$.

Publications pertaining to this chapter:

- Davar Kheirandish, Azadeh Safari, and Yinan Kong. *Using one hot residue number system (OHRNS) for digital image processing*, The 16th international symposium on artificial intelligence and signal processing (AISP 2012), 2-3 May 2012, Shiraz University, Shiraz, Iran.
- Davar Kheirandish, Azadeh Safari, and Yinan Kong. *A Novel Approach for Improving Error Detection and Correction in WSN*, CCECE 2014, Ontario, Canada, 4-7 May, pp. 370-373, 2014.

5 Scaling in the Residue Number System

RNS has always been challenging in its complex and costly operations for sign detection, division, magnitude comparison, base extension, scaling and reverse conversion. Nevertheless, these operations are very common when applying the RNS in computation-intensive applications such as digital signal and image processing [10, 72, 74].

Among all the complex operations, reverse conversion and scaling have been the main concern since they are the gateway for RNS-based designs to communicate with binary systems. Conversion from residue back to binary is a complex operation and would offset speed of the processor [88–90, 114]. Scaling is necessary to avoid dynamic-range overflow. Hence, simplifying the reverse conversion and scaling operations will be a breakthrough for using the RNS more than ever [76, 115–117].

In this Chapter, we will propose a modification to the full-adder-based scaler in [1] and the results will be compared with the full-adder-based scaling scheme in [1]. Subsequently, a hybrid scaling scheme for the three-moduli set $(2^n - 1, 2^n, 2^n + 1)$ will be presented. In the proposed designs, the scaling factor is one of the moduli (2^n) and $n = 8$. For the first time, a scaling scheme for the four-moduli set $(2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1)$ with dynamic range $6n$ and scaling factor $m_2 = 2^{2n}$ is proposed. Scaling for four-moduli sets has never been reported in the literature previously. Hence, this is the first scaler for a four-moduli set. The proposed scaling scheme is designed based on

CRT algorithm since it can be applied in parallel. Furthermore, a new modulo-(2^n+1) adder which can also be used for the modulo-$(2^{2n}+1)$ adder is proposed.

5.1 Previous Work

The first scaling scheme has been proposed by Szabo and Tanaka in 1967. They proposed a scaler that needed n clock cycles for a n-bit moduli set. Although scaled residues had errors, and the scheme did not provide correct scaled residues, it was a significant stage in the development of RNS-based systems [74]. In another major study in 1973, Okeefe and Wright designed a faster and more efficient scaler than the Szabo scaler. Again the results were not error-free but their approach provided results closer to the correct scaled integers [118]. In 1987, Jullien was successful in designing an algorithm that needed fewer clock cycles, but provided faulty results [119].

In 1981, Taylor and Huang proposed a design based on the MRC [120]. It was the first time a scaler based on the MRC was proposed. Until then, all designs were based on CRT or base-extension. The CRT-based algorithms generally generated fractional errors due to coarse assumptions, while the latter approach was error-free but computationally intensive. One year later, Taylor and Huang presented a scaler that used a special moduli set and LUTs. However, their design required n clock cycles to generate the scaled residues [121]. In 1984, Polky and Miller proposed a design that needed $(n+1)$ clock cycles but the scaled residues were closer to the correct results [122]. In other words, their design provided more accurate scaled residues at the cost of more clock cycles.

Two scaling algorithms for the 8-bit moduli set $(2^n-1, 2^n, 2^n+1)$ using CRT were proposed in 1988 [95]. The first algorithm was based on the L-CRT algorithm for L-tuple RNS. The second algorithm was based on 2^{2n-q}-CRT using an approximation of $M = 2^{3n}$ and $M^{'} = 2^q$, $q < 2n$. Although both algorithms came with assumptions and errors, the most important consequence of the L-CRT algorithm appeared in [96], where the authors suggested dividing the errors into two distinct bands: a band of small errors, and a band of errors on the order of the reduced-moduli system.

Five years later, Shenoy and Kumarson proposed two scaling techniques (approximate and exact), where residues were scaled by the product of a subset of the moduli set [76]. The approximate technique used a redundant residue to eliminate modulo-(M) operation, while the exact technique used a modified version of CRT. The modified CRT states that (5.1) never exceeds the dynamic range more than once (M), i.e. the sum is either (M) or $(X + M)$:

$$X = \left| \sum M_i \left| \hat{\alpha_i x_i} \right|_{m_i} \right|_M \tag{5.1}$$

The scaling error in the approximate technique was bounded by $[i/2]$, where i is the number of moduli in the moduli set. Their design saved a considerable amount of delay and generated results in only $\log n$ clock cycles. The exact technique, however, generated an error of at most unity, and used a redundant channel to keep track of odd or even residues. Ulman published a modified version of the Szabo scaler in 1993 [123], and since then, the results of all scalers have errors less than 1.5.

Another CRT-based scaling scheme was presented in 1995 [124]. It used LUTs and $\log_2 n$ clock cycles to generate scaled residues. The aim of the design was to achieve a precise result without using any redundant representation of numbers. The disadvantage was its worst case delay is n clock cycles. In 1999, two stages of look-up-cycle scaling, namely *look-up calculation* and *look-up generation*, was presented in [125]. The design was recommended for 5-bit input and three moduli sets. It was cascadable to other algorithms for larger sets of moduli, and reduced the bulk memory requirement for small moduli sets. In 2003, an alternative CRT-based scaler for up to 16-bit dynamic range was proposed [94]. The proposed scheme used only RNS operations within small-word-length channels. It was suitable for small-word-length applications and performed scaling directly on the residue digits rather than relying on residue-to-binary conversion.

From the implementation point of view, scaling algorithms are implemented either in LUT-based approaches [76, 93, 95, 123] or adder-based approaches [1, 73, 126, 127]. Generally, all the LUT-based designs in the literature are subject to poor pipeline-ability and high hardware complexity when the number of moduli increases. Adder-based designs are faster and provide huge savings in storage space. There are also other scaling circuits that benefit from both LUTs and full adders [128]. Most scaling schemes reported in the literature are based on LUTs, and none of the papers discussed the order of generation of scaled residues until 2007 [29]. Almost all publications have agreed that LUTs are more efficient for small inputs, as in image processing applications, while a FA-based structure is well suited for long inputs [129].

Extensive measurements in area, delay and hardware utilisation for full-adder-based designs have been provided in [126]. Subsequently, the authors in [130] have proposed full-adder-based RNS scaling schemes for VLSI implementation. They used CRT and a redundant modulus to perform the base extension and obtain the n least-significant bits of residues. In 2011, Chang [1] offered a new perspective in scaling circuits by modular full adders which resulted in speed increase and less power consumption [131]. Chang [1] scaler is the fastest and the most efficient full-adder based scaler to date. Downside of Chang scaler is that it is designed for three-moduli set with a dynamic range of $3n$ bits, which does not suffice for todays computer needs. Chang [1] proposed an error-free design and achieved improvements over previous implementations. It used both MRC and CRT in parallel. Channel one and three were designed based on MRC, while channel two used only CRT to generate the scaled residue.

Other contributions such as scaling negative integers by introducing a corrective factor have been reported in [132] and [130].

Reverse-conversion schemes have been also changing along with the scaling schemes. The first designs required many consecutive addition cycles for reverse conversions [127]. However, recent schemes have embedded reverse conversion as a by-product of the scaling procedure and saved a significant hardware cost [1, 123].

5.2 Mathematical Basis for Designing Scalers

Some useful lemmas, axioms and a theorem to simplify scaling equations are presented in the following.

Scaling in the RNS corresponds to division of an integer (X) by a constant (k). It can be shown as

$$Y = \left\lfloor \frac{X}{k} \right\rfloor \tag{5.2}$$

where k is called the scaling factor, Y is the result of scaling X by k and $\lfloor x \rfloor$ is the floor function (the greatest integer function) [93].

Axiom 1:

$$a|X|_b = |aX|_{ab} \tag{5.3}$$

Axiom 2:

$$|X \pm Y|_m = ||X|_m \pm |Y|_m|_m \tag{5.4}$$

Axiom 3:

$$|X.Y|_m = ||X|_m.|Y|_m|_m \tag{5.5}$$

Axiom 4:

If X is an n-bit number, then:

$$|-X|_{2^n+1} = \overline{X} + 2 \tag{5.6}$$

Lemma (a) [72, 74]

Given: $|aX|_m = b$ and $(a, m) = 1$, where $a, m \geqslant 0$, what is $|X|_m =?$

Ans: $|a^{-1}a|_m = 1$ and $|X|_m = |X|_m$, therefore:

$$|a^{-1}aX|_m = |X|_m. \tag{5.7}$$

If $|aX|_m = b$, then (5.7) can be written as:

$$|a^{-1}aX|_m = |a^{-1}b|_m \tag{5.8}$$

The left sides of (5.7) and (5.8) are equal. Hence, the right sides are equal too:

$$|X|_m = |a^{-1}b|_m \tag{5.9}$$

Lemma (b) [72, 74]

Given: $|X|_m = a$, $k|x$ and $(k, m) = 1$, what is $|\frac{X}{k}|_m =?$

Ans: Let $\frac{X}{k} = b$, hence $X = kb$

Given $|X|_m = a$, which is $|kb|_m = a$

Using lemma (a):

$|b|_m = |k^{-1}a|_m$, hence we have

$$|\frac{X}{k}|_m = |k^{-1}a|_m \tag{5.10}$$

Lemma (c)

Given: $1 \leq a \leq m - 1$ and $(a, m) = 1$, evaluate the multiplicative inverse of a with regard to m, i.e. evaluate a^{-1} such that $|a^{-1}a|_m = 1$.

Method 1:

$$a^{-1} = \begin{cases} 1, & a = 1 \\ \frac{mk+1}{a}, & 1 \leq k \leq a - 1 \end{cases} \tag{5.11}$$

where k is the value that can make a^{-1} an integer.

Method 2:

$$a^{-1} = |a^{\varphi(m-1)}|_m \tag{5.12}$$

where $\varphi(m) = m\prod_{p|m}(1 - \frac{1}{p})$ is the number of values in $(1, 2, ..., m)$ that are prime to m, and p is a prime factor of m.

Theorem 1

Given $p = kq$, where k is an integer:

$$\left| |X|_p \right|_q = |X|_q \tag{5.13}$$

Proof:

$$|X|_p = X - \varepsilon p \tag{5.14}$$

where $\varepsilon = 0, 1, 2, \ldots$. Using (5.4) and (5.5), it can be expanded as:

$$\left| |X|_p \right|_q = |X - \varepsilon p|_q = \left| |X|_q - |\varepsilon|_q . |p|_q \right|_q . \tag{5.15}$$

Since $p = kq$ (p is divisible by q) and $|p|_q = 0$, then:

$$\left| |X|_p \right|_q = |X|_q \tag{5.16}$$

Theorem 2 The following holds true [133]:

The positive integers $(2^a - 1)$ and $(2^b - 1)$ are relatively prime if and only if a and b are relatively prime.

5.3 Scaling Scheme Based on Modular Reducers

The full-adder-based scaling scheme has been presented in [1]. It proposed an error free design and achieved improvements over previous implementations. It used both the MRC and CRT in parallel. Channel one and three were designed based on MRC, while channel two used only CRT to generate the scaled residue. It has been shown that a full-adder-based scaler yields savings on total power consumption and leakage power up to 44.3% and 41.4%, respectively. Figure 5.1 shows the full-adder-based scaler in [1].

In this section, a modified version of the scaling scheme in [1] is proposed. The proposed scheme is performed using full adders as opposed to Read-Only Memory (ROM) and LUTs. One particular advantage of the proposed scheme over [1] is that the proposed design can be easily pipelined, which is one of the main ways of increasing throughput. It also reduces the complex lower-level logic of Chang scaler [1].Various contributions and substitutions are made to the logic blocks at the various stages, as shown by its structure in Figure 5.2.

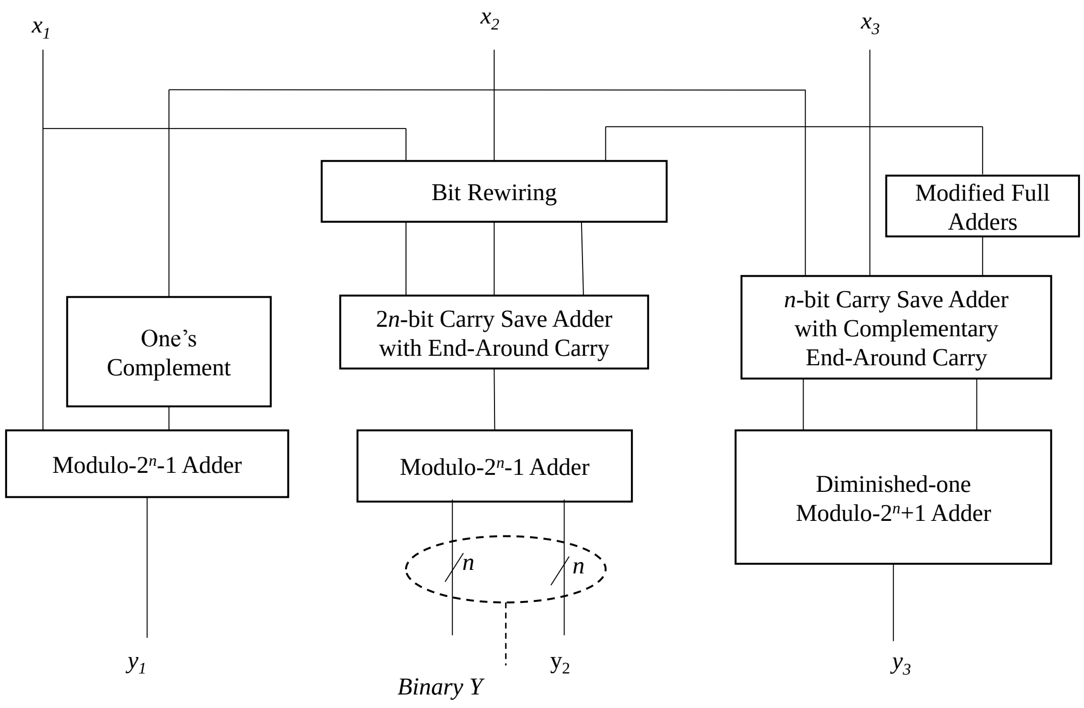

x_1
x_2
x_3
Bit Rewiring
Modified Full Adders
One's Complement
$2n$-bit Carry Save Adder with End-Around Carry
n-bit Carry Save Adder with Complementary End-Around Carry
Modulo-2^n-1 Adder
Modulo-2^n-1 Adder
Diminished-one Modulo-2^n+1 Adder
n
n
y_1
Binary Y
y_2
y_3

FIGURE 5.1: Full-adder-based scaler [1]

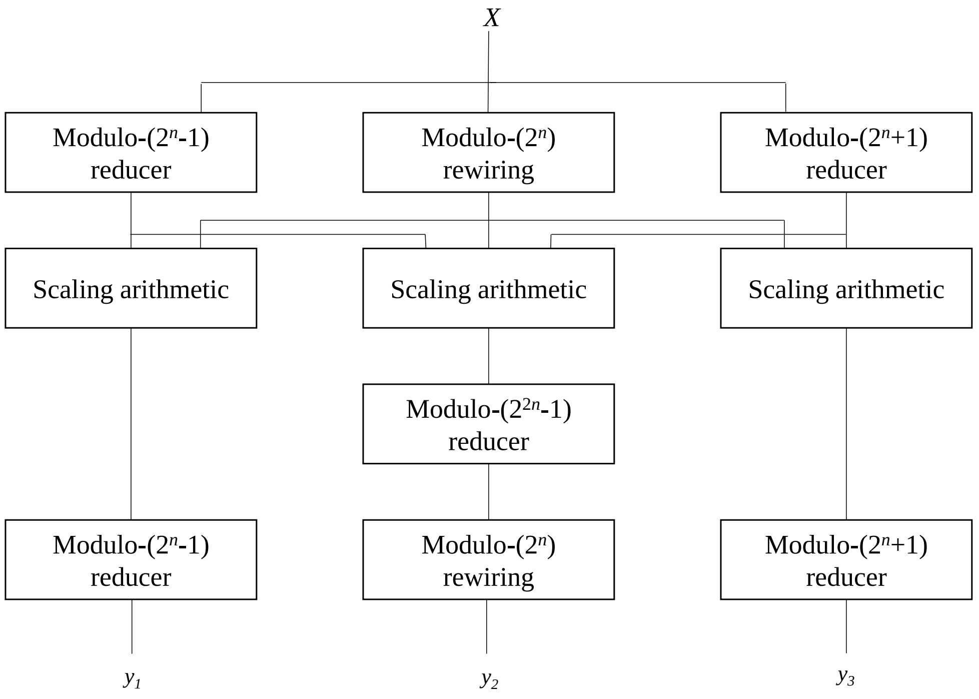


FIGURE 5.2: Proposed scaling scheme using full-adder-based modular reducers

There are two key aspects of the proposed scheme that make it different from the majority of scaling circuits in literature. First, it is designed using full-adder-based modular reducers, while most scaling circuits use LUTs to implement RNS scalers [95, 123]. Second, the use of forward converters is eliminated.

The proposed scaler assumes that the given inputs are not in residue form, i.e. the binary inputs need to be converted to residues. Conversion from binary to residues increases the overall time, complexity and power consumption of the system. Hence, we have modified the proposed scaler and eliminated the forward-conversion overhead.

5.3.1 Modulo-$(2^n - 1)$ Reduction

With any n-bit RNS-based system, the dynamic range of that system is $3n$-bit [134],[73]. This means that the system needs a reduction that can handle $3n$-bit inputs. Luckily, the reduction of a $3n$-bit number can be broken down in to three smaller reductions. We choose to represent the $3n$-bit input X as A, B, and C, where:

$$A = X_{3n-1}X_{3n-2}...X_{3n-n} \tag{5.17}$$

$$B = X_{2n-1}X_{2n-2}...X_{2n-n} \tag{5.18}$$

$$C = X_{n-1}X_{n-2}...X_0 \tag{5.19}$$

According to the numeric properties of residues and moduli sets, there is a relation between A, B, and C of

$$|X|_{2^n-1} = |A + B + C|_{2^n-1} \tag{5.20}$$

While (5.20) may seem to increase the complexity of reduction, it significantly helps reduce the problem, as this modular reducer can be implemented as an addition with an End-Around Carry (EAC). To implement the modulo-$(2^n - 1)$ reducer, a modulo adder is used based on half-adders and *Brent-Kung (BK)* adders [135].

5.3.2 Modulo-$(2^n + 1)$ Reduction

Implementing the modulo-$(2^n + 1)$ reducer is done similar to the modulo-$(2^n - 1)$ reducer. We represent (X) as three separate n-bit numbers A, B, and C, and reduce the modular operation into the reduction of the addition and subtraction of A, B and C.

$$|X|_{2^n+1} = |A - B + C|_{2^n+1} \tag{5.21}$$

To implement the $(2^n + 1)$ reduction using (5.21), an End-Around-Carry (EAC) adder is used. The design uses BK and half-adders to implement the modulo-$(2^n + 1)$ reducer.

As the signal goes through different stages the size of the signal changes. The expected size for the second channel before the modulo-$(2^{2n} - 1)$ reducer is up to 24 bits. Hence, the resize functions are necessary to keep the outputs constant. Essentially the outputs are padded with zeros on the left side of the most significant bit (MSB).

5.3.3 Optimised Modulo-$(2^n + 1)$ Reducer

The modulo-$(2^n + 1)$ reducer is limited to only two inputs. This initially is not a problem, but for a system with a $3n$-bit dynamic range, each reducer would be expected to receive three inputs: A, B, and C. The solution is to use two stages of adders for adding three numbers; however, when joining the output of one adder to the input of another, the $(2^n + 1)$ reducer requires $(n + 1)$ bits (9 bits in our case) to fully represent all the possible solutions. To handle this issue, a correction bit is subtracted from the input after the second stage of modular addition. During testing it is found that, when the 9^{th} bit is one, the result would be (+2) off. To correct this error, a subtraction is required. The easiest way to implement a subtraction is *type casting*, taking advantage of the unsigned integer arithmetic library of Xilinx tools.

$$typeCast_3 <= unsigned(in3) - 2 * unsigned(correctionBit); \tag{5.22}$$

where $typeCast_i$ corresponds to x_i. The design requires another type casting into and back from the unsigned integer signal, when it undergoes its final $(2^n + 1)$ reduction, to allow for the correction bit. The ideal solution would be implementing full-adders or three-two compressors on the top level of the modulo reducer as opposed to half-adders.

5.3.4 Numerical Example

A numerical example of the proposed modular-reducer-based scaler is considered. Let $X = 10079317$ which is equivalent to 24 bits (100110011100110001010101) in the binary number system. The proposed algorithm splits (X) to three eight-bit numbers: $C =$

10011001, B = 11001100 and A = 01010101.

For scaling factor $k = 256$, the scaled (X) is equal to $\frac{X}{k} = 39372$, and the algorithm yields the scaled residues $s_1 = 102$, $s_2 = 204$, $s_3 = 51$ which are same as the results using (11), (12) and (13) in [1].

$$s_1 = |x_1 - x_2|_{m_1} = |187 - 85|_{255} = 102 \tag{5.23}$$

$$\begin{aligned} s_2 &= ||(2^{2n-1} + 2^{n-1})x_1 - 2^n x_2 + (2^{2n-1} + 2^{n-1} - 1)x_3|_{m_1 m_3}|_{m_2} \\ &= ||32896 \times 187 - 256 \times 85 + 32895 \times 34|_{65535}|_{256} = 204 \end{aligned} \tag{5.24}$$

$$s_3 = |x_2 - 2^n x_3|_{m_3} = |85 - 256 \times 34|_{257} = 51 \tag{5.25}$$

5.3.5 Synthesising the Proposed Modular-Reducer-Based Scaler

The proposed scaler is designed using Xilinx tools in such a way that the entire code is reusable if the design is to be scaled or fitted as sub-modules of a larger design. The VLSI architecture of modulo-$(2^n - 1)$ and modulo-$(2^n + 1)$ reducers are shown in Figures 5.3 and 5.4, respectively.

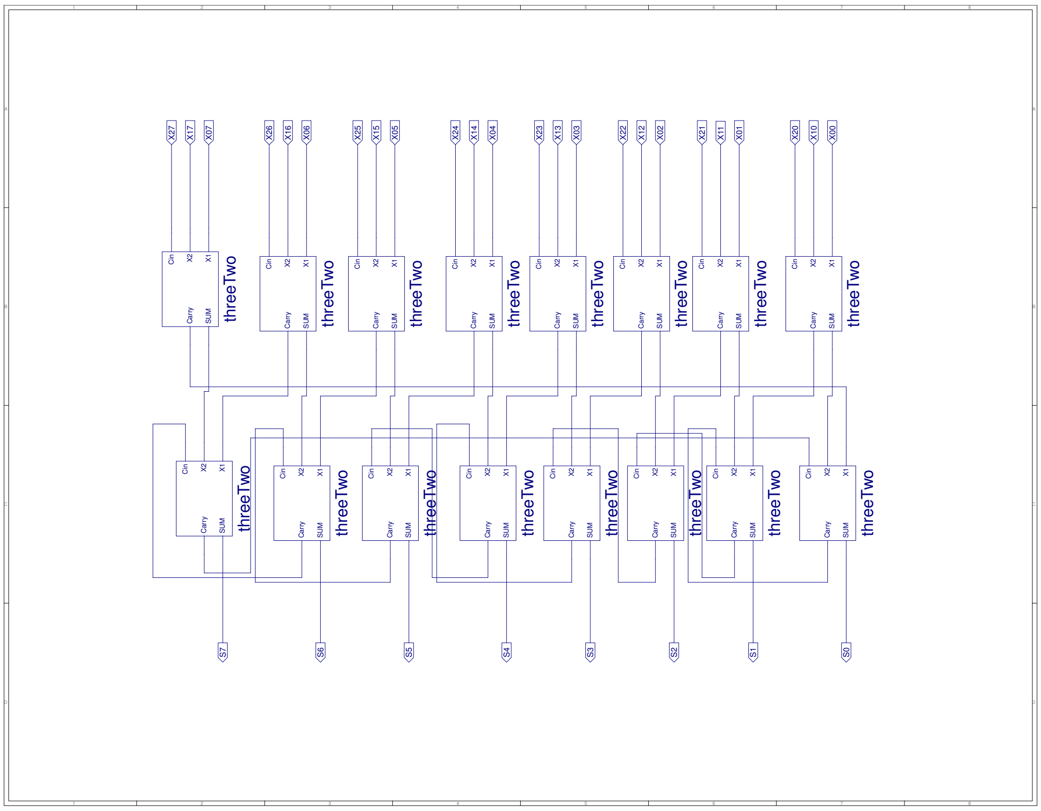


Figure 5.3: VLSI architecture of modulo-$(2^n - 1)$ reducer

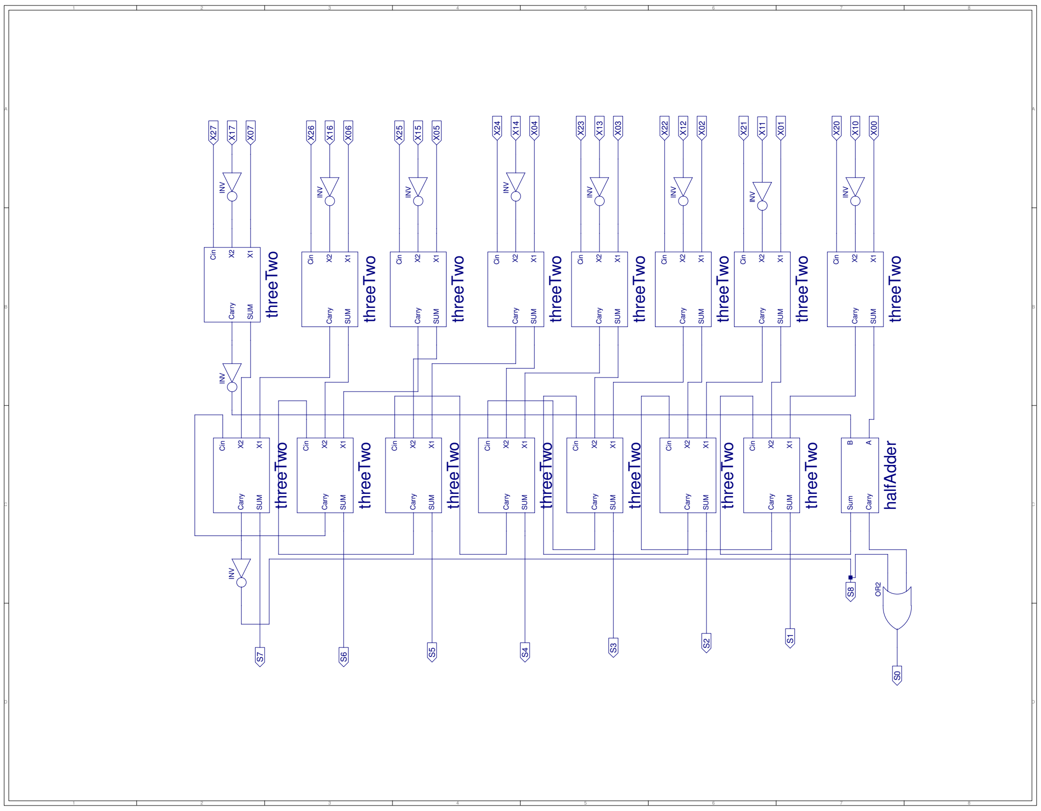


Figure 5.4: VLSI architecture of modulo-$(2^n + 1)$ reducer

In designing the modulo-$(2^n - 1)$ reducer, the 3:2 compressors (full-adders) and half-adder. The VLSI architectures of these modules are shown in Figures 5.5 and 5.6, respectively.

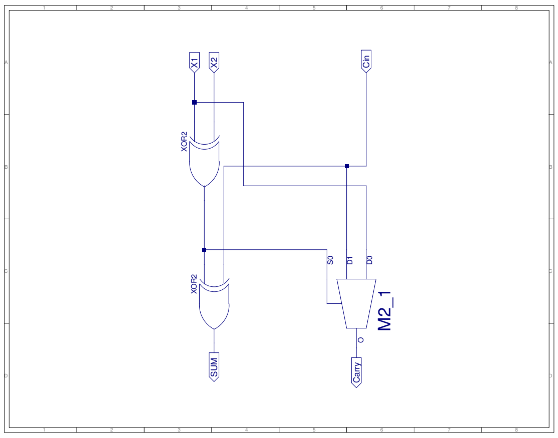


FIGURE 5.5: VLSI architecture of the 3:2 compressor (full adder) used in modular reducers

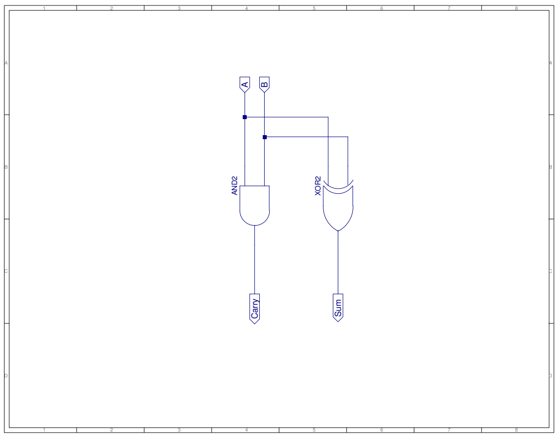


FIGURE 5.6: VLSI architecture of the half adder used in designing the modulo-$(2^n - 1)$ reducer

The design is synthesised using Xilinx tools with a clock period of 20 ns (clock cycle 50%) on target device: $xc6vlx75t - 3 - ff484$. The synthesis results show that the total timing of the circuit is 33.11 ns, which is a combination of both routing and logic delays. The hardware utilisation is provided in Table 5.1.

Table 5.2 provides a power-supply summary. The estimated power including the resource utilisations of factors such as LUTs, I/Os, registers and clock frequency in

Table 5.1: Hardware utilisation of modular-reducer-based scaler

Slice Logic Utilisation	
Number of slice registers	403 out of 93120
Number of Slice LUTs	518 out of 46560
Number used as Logic	518 out of 46560
Slice Logic Distribution	
Number of LUT Flip Flop pairs used	529
Number with an unused Flip Flop	126 out of 529
Number with an unused LUT	11 out of 529
Number of fully used LUT-FF pairs	392 out of 529
Number of unique control sets	1
IO Utilisation	
Number of IOs	66
Number of bonded IOBs	58 out of 240
Specific Feature Utilisation	
Number of BUFG/BUFGCTRLs	1 out of 32

the design are presented in Table 5.3. The Target device specifications and logic slice utilisation are also provided. The total memory usage is 251944 kilobytes.

Table 5.2: Power supply summary of modular-reducer-based scaler

	Total	Dynamic	Quiescent
Supply Power (mW)	1347.01	50.59	1296.42

Table 5.3: On-chip power summary of modular-reducer-based scaler

On-Chip	Power (mW)	Used	Available	Utilisation (%)
Clocks	3.11	1	-	-
Logic	2.61	350	46560	0
Signals	2.44	580	-	-
I/Os	42.43	58	240	24
Leakage	1296.42	-	-	-
Total	1347.01	-	-	-

5.3.6 Comparison of the Synthesis Results of the Proposed Modular-Reducer-Based Scaler and a Full-Adder-Based Scaler

For comparison purposes, the-full-adder based scaler in [1] is designed and synthesised as well. For consistency reasons, the same clock period and target device are selected. Tables 5.4 to 5.6 show the synthesis results. Details of the synthesis results of full-adder-based scaling scheme (Chang scaler) in [1] are provided in Appendix B.

TABLE 5.4: Hardware utilisation of full-adder-based scaler

Slice Logic Utilisation	
Number of slice registers	207 out of 93120
Number of Slice LUTs	232 out of 46560
Number used as Logic	168 out of 46560
Number used as memory	64 out of 16720
Slice Logic Distribution	
Number of LUT Flip Flop pairs used	289
Number with an unused Flip Flop	82 out of 289
Number with an unused LUT	57 out of 289
Number of fully used LUT-FF pairs	150 out of 289
Number of unique control sets	2
IO Utilisation	
Number of IOs	51
Number of bonded IOBs	51 out of 240
Specific Feature Utilisation	
Number of BUFG/BUFGCTRLs	1 out of 32

TABLE 5.5: Supply power of full-adder-based scaler

	Total	Dynamic	Quiescent
Supply Power (mW)	1326.76	30.82	1295.94

A comparison of the modular-reducer-based scaler synthesis results in Table 5.1 and the full-adder-based scaler in Table 5.4 reveals that there is no decrease in hardware utilisation. In other words, any benefit of using modular reducers in decreasing the

Table 5.6: On-chip power summary of full-adder-based scaler

On-Chip	Power (mW)	Used	Available	Utilisation (%)
Clocks	2.83	1	-	-
Logic	0.71	221	46560	0
Signals	0.70	334	-	-
I/Os	26.58	51	240	21
Leakage	1295.94	-	-	-
Total	1326.76	-	-	-

hardware utilisation could not be identified in the synthesis results. The comparison of the supply power and the on-chip power summary in Tables 5.2 and 5.3 with Tables 5.5 and 5.6 does not show much increase in performance either.

Although no performance improvements are reported in the proposed modular-reducer-based scaler, it is able to generate accurate scaled residues which was not achieved in the primary scalers. We propose several suggestions to reduce the hardware complexity of the modular-reducer-based scaler. One suggestion is that the multiplexer would be switched by a $2n$- or n-bit EAC. When the multiplexer select input is one, it switches to the n-bit EAC. When it is zero, it switches to the $2n$-bit EAC. Hardware reduction would generally occur in a larger system that would be able to monitor the stage of the algorithm to determine the appropriate value of the determining bit. When the adder is in n-mode, it assumes that all other values are outside of the n-bit range, hence they are assumed to be zero. In the proposed scheme, this solution would be inappropriate as there is no governing circuit monitoring the stages of the algorithm, but in a bigger system where scaling is useful, such as DSP, there may be a use for such a reduction. In the following section, we will present a scaler based on full-adders and LUTs to achieve the desired results.

5.4 Simple, Fast, Hybrid Scaling Scheme Using Full-Adders and LUTs

This section presents a hybrid scaling scheme for RNS-based systems using a three-moduli set $(2^n - 1, 2^n, 2^n + 1)$. We will use this scaler in an initial RNS-based image processor in Chapter 6. It is based on a combination of LUT-based and adder-based approaches, and provides a scaled integer in the normal binary representation. Hence the tedious reverse conversion is totally eliminated. The main contributions of the proposed scaler are as follows:

1. It is a hybrid implementation and benefits from combining both high-speed LUTs and area-efficient modular adders

2. There is the same delay in generating each of the scaled residues in the modular channels

3. Scaling in the channel modulo-$(2^n + 1)$ is simplified

4. This scheme is specifically efficient for RNS-based image processing applications where the image pixels are in limited bound and the LUTs are small.

5.4.1 Proposed Hybrid Scaler

Since scaling is an inter-moduli operation, it is necessary to have a knowledge of all the residues for scaling and generating the original scaled integer [96, 124]. For a three-moduli set, the CRT algorithm is expressed as:

$$\begin{aligned} X &= \left| \sum_{i=1}^{N} M_i \left| M_i^{-1} \right|_{mi} x_i \right|_M \\ &= \left| \sum_{i=1}^{3} M_i \left| M_i^{-1} \right| x_i \right|_M \\ &= \left| m_2 m_3 \left| M_1^{-1} \right|_{m_1} x_1 + m_1 m_3 \left| M_2^{-1} \right|_{m_2} x_2 + m_1 m_2 \left| M_3^{-1} \right|_{m_3} x_3 \right|_M \end{aligned} \tag{5.26}$$

For scaling (X) by (k), both sides of (5.26) should be divided by k followed by the floor operation:

$$\left\lfloor \frac{X}{k} \right\rfloor = \left\lfloor \left| \frac{(m_2 m_3)}{k} |\hat{M}_1^{-1}|_{m_1} x_1 + \frac{(m_1 m_3)}{k} |\hat{M}_2^{-1}|_{m_2} x_2 + \frac{(m_1 m_2)}{k} |\hat{M}_3^{-1}|_{m_3} x_3 \right|_{\frac{M}{k}} \right\rfloor \tag{5.27}$$

For the scaling factor $k = m_2 = 2^n$, (5.27) can be simplified to calculate the scaled integer directly using the residues of (X).

$$\left\lfloor \frac{X}{k} \right\rfloor = \left\lfloor \left| m_3 |\hat{M}_1^{-1}|_{m_1} x_1 + \frac{(m_1 m_3)}{m_2} |\hat{M}_2^{-1}|_{m_2} x_2 + m_1 |\hat{M}_3^{-1}|_{m_3} x_3 \right|_{m_1 m_3} \right\rfloor \tag{5.28}$$

In channel one, where $m_1 = 2^n - 1$, (5.28) is simplified, and needs no multiplication:

$$\left| \left\lfloor \frac{X}{k} \right\rfloor \right|_{m_1} = |x_1 - x_2|_{m_1} \tag{5.29}$$

If the last modulo-m_2 step is neglected, the original binary number is achieved with no additional logic or delay cost.

$$\left| \left\lfloor \frac{X}{k} \right\rfloor \right|_{m_2} = ||(2^{2n-1} + 2^{n-1})x_1 - 2^n x_2 + (2^{2n-1} + 2^{n-1} - 1)x_3|_{m_1 m_3}|_{m_2} \tag{5.30}$$

In channel three, this simplification leads to

$$\left| \left\lfloor \frac{X}{k} \right\rfloor \right|_{m_3} = |x_2 + 2^n x_3|_{m_3}. \tag{5.31}$$

However, the multiplication $(2^n x_3)$ modulo-$(2^n + 1)$ can be further simplified to $(-x_3)$ modulo-$(2^n + 1)$. Thus, instead of evaluating (5.31) containing a multiplication, it can be directly calculated using [73]:

$$\left| \left\lfloor \frac{X}{k} \right\rfloor \right|_{m_3} = |x_2 - x_3|_{m_3} \tag{5.32}$$

5.4.2 Error Analysis of Approximation

The error analysis of the approximation made in (5.29) - (5.32) is reconsidered. The exact equation in (5.30) is

$$\begin{aligned} \left\lfloor \frac{X}{k} \right\rfloor = \Big\lfloor &|(2^{2n-1} + 2^{n-1})x_1 + (2^{2n} - 2^{n+1} + 2^n - 1 + \frac{1}{2^n})x_2 \\ &+ (2^{2n-1} + 2^n - 2^{n-1} - 1)x_3|_{m_3 m_1} \Big\rfloor \end{aligned} \tag{5.33}$$

The relation between the approximate equation (5.30) and the exact equation (5.33) is:

$$\left\lfloor \frac{X}{k} \right\rfloor = \left\lfloor \frac{X}{k} \right\rfloor' + \left\lfloor \frac{x_2}{2^n} \right\rfloor \tag{5.34}$$

where $\left\lfloor \frac{X}{k} \right\rfloor'$ is the approximate scaling. Since $0 \leq x_2 < 2^n$, then $0 \leq \frac{x_2}{2^n} < 1$ and as a property of the floor function, therefore $\left\lfloor \frac{x_2}{2^n} \right\rfloor = 0$. The proposed algorithm is accurate and error free.

TABLE 5.7: Numerical example of proposed hybrid scaler for residue set (123, 55, 250) and scaling factor 256

	Channel 1	Channel 2	Channel 3
m_i	255	256	257
x_i	123	55	250
M_i	65792	65535	65280
M_i^{-1}	128	255	129
$x_i.M_i.M_i^{-1}$	1035829248	919128375	2105280000
y_i	68	65	62

Example Table 5.7 shows a numerical example the of proposed algorithm. Consider the residues $x_1 = 123$, $x_2 = 55$ and $x_3 = 250$ corresponding to the moduli set (255, 256, 257). The dynamic range of the system is M =16776960, and $M_1 = 65792$, $M_2 = 65535$ and $M_3 = 65280$. Using (4.11) to (4.13), the multiplicative inverses are $M_1^{-1} = 128$, $M_2^{-1} = 255$ and $M_3^{-1} = 129$. The scaling factor is $k = 2^n$ for $n = 8$. The scaling is performed for each channel using (5.29), (5.30) and (5.32) as:

$$y_1 = |x_1 - x_2|_{m_1} = |123 - 55|_{255} = 68 \tag{5.35}$$

$$y_2 = ||32896{\times}123 - 256{\times}55 + 32895{\times}250|_{65535}|_{256} = 65 \tag{5.36}$$

$$y_3 = |x_2 - x_3|_{m_3} = |55 - 250|_{257} = 62 \tag{5.37}$$

Note that channel two generates the 16-bit original binary number (X) and y_2 is the eight least-significant bits (LSB) of the result. Channel two generates a binary number (X = 000001101000001), equivalent to 833 in the decimal number system. This is the scaled original binary number, and this number modulo-256 is the scaled residue y_2.

5.4.3 Synthesis Results

We use n-bit two-operand modulo-$(2^n - 1)$ and modulo-$(2^n + 1)$ subtractors to implement the scaled residues y_1 and y_3, respectively. To generate the scaled residue y_2, all three residues have to be present for computations. We use three (256×16) LUTs for this purpose. The size of the LUTs is selected based on the 256 levels of luminance in greyscale images, and the 16-bit width is the result of using modulo-65535 operations. LUT1, LUT2 and LUT3 store the following modular-operation results:

LUT1:

$$|(2^{2n-1} + 2^{n-1})x_1|_{m_1.m_3} = |32896 \times x_1|_{65535} \tag{5.38}$$

LUT2:

$$|2^n x_2|_{m_1.m_3} = |256{\times}x_2|_{65535} \tag{5.39}$$

LUT3:

$$|(2^{2n-1} + 2^{n-1} - 1)x_3|_{m_1.m_3} = |32895{\times}x_3|_{65535} \tag{5.40}$$

Outputs from the LUTs go to a $2n$-bit modular subtractor and a $2n$-bit modular adder. The result of the $2n$-bit modular adder is the scaled original number, and the eight least-significant bits of the result represent the scaled residue y_2. Figure 5.7 shows the block diagram of the proposed hybrid scaling scheme. It consists of three 256-word by 16-bit LUTs, two n-bit modular subtractors, one $2n$-bit modular subtractor and one $2n$-bit modular adder. Although the arithmetic operations are performed independently in each channel, the three residues from each channel should be present at the same time for scaling.

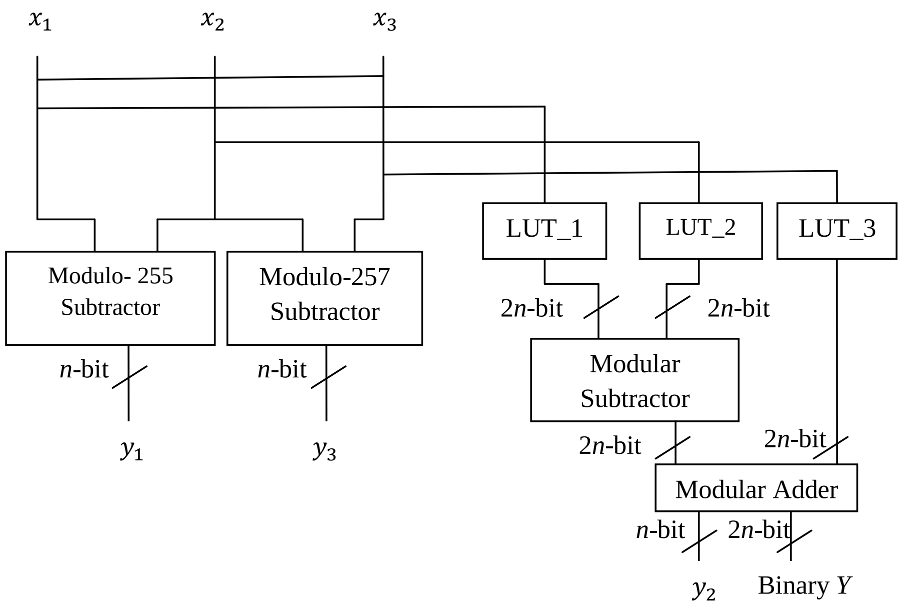


Figure 5.7: Proposed hybrid scaling scheme

The proposed hybrid scaler is implemented using Xilinx tools. The logic synthesis and gate-level Modelsim simulations are performed to verify the functionality and logic of the design. The placement and routing is performed, and the RTL and technology layouts are generated. Figure 5.8 shows the schematic of the proposed hybrid scaler using Xilinx tools.

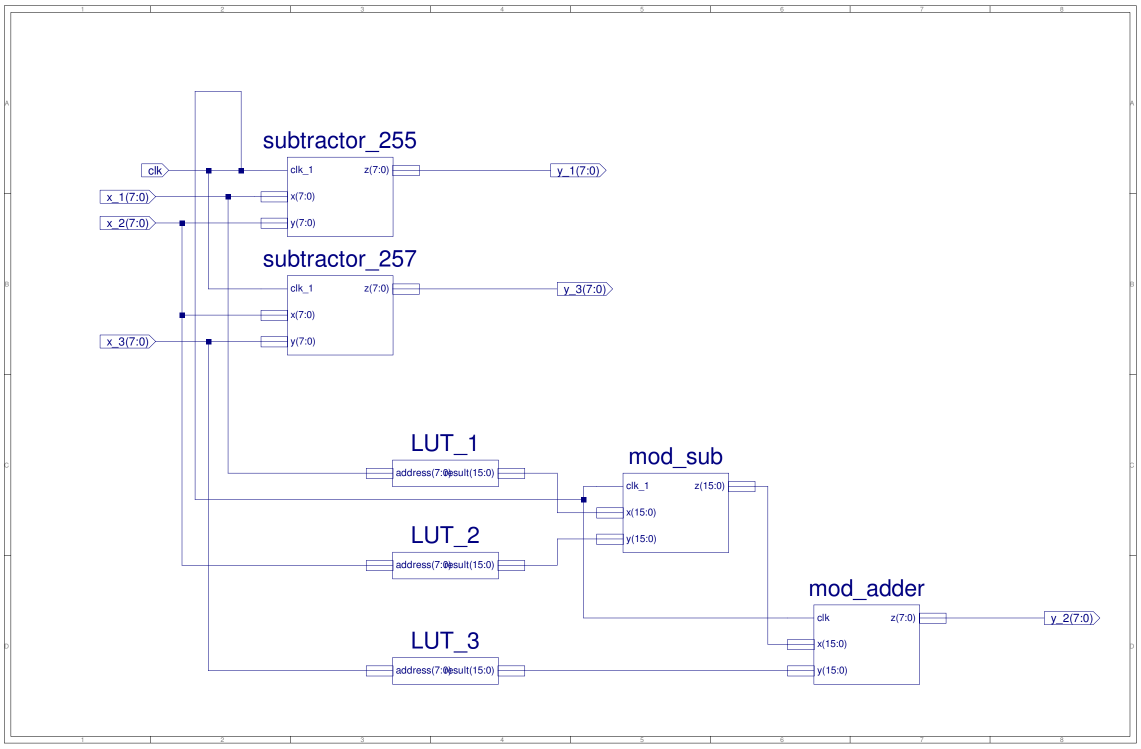


FIGURE 5.8: Schematic of the proposed hybrid scaling scheme

Table 5.8 shows the summary of the estimated values of device utilisation. The synthesis results show that the minimum period is 8.295 ns (Maximum Frequency:120.550 MHz), while the minimum input arrival time before the clock is 7.401 ns, and the maximum output required time after the clock is 3.597 ns.

Table 5.8: Hardware utilisation of hybrid scaling scheme

Slice Logic Utilisation	
Number of slice registers	24 out of 93120
Number of Slice LUTs	156 out of 46560
Number used as Logic	156 out of 46560
Slice Logic Distribution	
Number of LUT Flip Flop pairs used	156
Number with an unused Flip Flop	132 out of 156
Number with an unused LUT	0 out of 156
Number of fully used LUT-FF pairs	24 out of 156
Number of unique control sets	1
IO Utilisation	
Number of IOs	49
Number of bonded IOBs	49 out of 240
Specific Feature Utilisation	
Number of BUFG/BUFGCTRLs	1 out of 32

The on-chip power summary and supply power of the proposed scaler are provided in Tables 5.9 and 5.10, respectively. It is based on resource utilisations of the LUTs, I/Os, registers, and the clock frequency of the design. The total on-chip power consumption is 1310.41 mW, where 14.85 mW is dynamic and 1295.55 mW is the quiescent power consumption.

Table 5.9: On-chip power summary of hybrid scaling scheme

On-Chip	Power (mW)	Used	Available	Utilisation (%)
Clocks	1.73	1	-	-
Logic	0.33	133	46560	0
Signals	0.35	202	-	-
I/Os	12.44	49	240	20
Leakage	1295.55	-	-	-
Total	1310.41	-	-	-

Table 5.10: Supply power of hybrid scaling scheme

	Total	Dynamic	Quiescent
Supply Power (mW)	1310.41	14.85	1295.55

5.4.4 Evaluation and Comparison

In this section, the area and delay of the proposed hybrid scaler are evaluated and compared with other scaling schemes surveyed in [1] and [126] including themselves in terms of hardware implementation, scaling error, area and unit gate delay. For this purpose, the transistor count, which is the most common measure of integrated-circuit complexity, is used to evaluate the hardware cost.

The unit-gate model presumes that each two-input monotonic gate (except XOR) performs one elementary gate function for both area and delay, i.e. one unit of area and one unit of delay. There is an exception for the XOR gate that counts for two elementary gates for both area and delay, i.e. two units of area and two units of delay [136]. The area and delay of an inverter is a negligible fraction of a unit, hence it is assumed to have a zero unit area and delay [1].

The area and delay for each modular channel is evaluated independently by studying the area and time complexity of the logic-gate implementation of scaling in each channel. Based on the proposed architecture in Figure 5.7, both channels one and three have one n-bit modulo-255 and one modulo-257 subtractor, respectively. The authors in [137] have estimated the area and delay for a modulo-(2^n-1) adder to be $3n\left\lceil log_2 n-1\right\rceil+12n$ and $2\left\lceil log_2 n-1\right\rceil+3$ units, respectively. We need n inverters for implementing the one's-complement module. In modular channel two, the proposed design has three LUTs of size $2^n \times n$. Therefore, the total number of transistors of the proposed design is about $3 \times A_{ROM(2^n\times n)}$. The area, A_{ROM}, and the unit gate delay, T_{ROM}, for a LUT of size $2^n \times n$ are [138]:

$$A_{ROM} = 2^{\left\lceil \frac{n}{2}\right\rceil}(\left\lceil \frac{n}{2}\right\rceil + 1) + 2^n n + 2^{\left\lfloor \frac{n}{2}\right\rfloor} n(\left\lfloor \frac{n}{2}\right\rfloor + 2) + n(2^{\left\lfloor \frac{n}{2}\right\rfloor} + 1) \tag{5.41}$$

$$T_{ROM} = (1 + \lceil log_2 n\rceil + \left\lceil \frac{n}{2}\right\rceil).t_{NAND} \tag{5.42}$$

where n is the bit width of the moduli set, and t_{NAND} is the delay of a two-input NAND gate with the minimum drive. As discussed before, a NAND gate is presumed to have one unit of delay and one unit of area. Table 5.11 shows the estimation of area for each modular channel.

The speed of the proposed design is evaluated in terms of gate delay as well. Generally, the execution speed of the arithmetic operations of a design depends on two factors: the technology of the circuit, and the selected algorithm. In this section, we are interested only in the algorithm and not in the technology; therefore, the speed of

Table 5.11: Estimation of unit gate area of the proposed hybrid scaler

Modulus	A_{adder}	A_{LUT}	Total
m_1	$3n\lceil log_2 n - 1\rceil + 12n$	-	$3n\lceil log_2 n - 1\rceil + 12n$
m_2	$6n\lceil log_2 n\rceil + 24n$	$3A_{ROM}$	$6n\lceil log_2 n\rceil + 24n + 3A_{ROM}$
m_3	$3n\lceil log_2 n - 1\rceil + 12n$	-	$3n\lceil log_2 n - 1\rceil + 12n$

the algorithms will be expressed in terms of gate delays. Translating the number of gate delays for a given technology to the actual speed can be simply done by multiplying the number of gate delays by the gate speed of that technology [139]. The estimation of the unit gate delay of the proposed design is presented in Table 5.12.

Table 5.12: Estimation of unit gate delay of the proposed hybrid scaler

Modulus	D_{adder}	D_{LUT}	Total
m_1	$2\lceil log_2 n - 1\rceil + 3$	-	$2\lceil log_2 n - 1\rceil + 3$
m_2	$2\lceil log_2 n\rceil + 3$	$(1 + \lceil log_2 n + \lceil \frac{n}{2}\rceil\rceil).t_{NAND}$	$2\lceil log_2 n\rceil + 3 + (1 + \lceil log_2 n + \lceil \frac{n}{2}\rceil\rceil).t_{NAND}$
m_3	$2\lceil log_2 n - 1\rceil + 3$	-	$2\lceil log_2 n - 1\rceil + 3$

Since the hardware complexity is expressed as a number of transistors, the transistor count of the proposed scaler is estimated from its unit-gate area by reasonably and conservatively assuming that a two-input monotonic gate can be implemented with six transistors using a classical CMOS implementation. We consider an inverter as equivalent to two transistors. The conversion of the unit gate area from Table 5.11 to the equivalent number of transistors of the proposed scaler is shown in Table 5.13.

Table 5.13: Estimated number of transistors of proposed hybrid scaler

Modulus	A_{adder}	A_{LUT}	Total
m_1	$18n\lceil log_2 n - 1\rceil + 72n$	-	$18n\lceil log_2 n - 1\rceil + 72n$
m_2	$18n\lceil log_2 n\rceil + 144n$	$18A_{ROM}$	$18n\lceil log_2 n\rceil + 144n + 18A_{ROM}$
m_3	$18n\lceil log_2 n - 1\rceil + 72n$	-	$18n\lceil log_2 n - 1\rceil + 72n$

The proposed scaler is compared with existing schemes in the literature using the same bit-width ($n = 8$) and the same moduli set. Table 5.14 shows a summary of the hardware implementation strategy (LUT and FA), and the scaling errors from Table V of [1] and for the proposed design. All the schemes except [1] and [126] have used LUTs. In comparison, the proposed design uses mixed FAs and LUTs. Table 5.14 also shows that all designs have scaling errors except the proposed design, [1] and [125].

Table 5.14: Comparison of estimated number of transistors in proposed hybrid scaler and other designs for $n = 8$, $(255, 256, 257)$, and $M = 16776960$

	Hybrid scaler	[1]	[126]	[125]	[123]	[76]	[95]
Implementation	LUT+ FA	FA	FA	LUT	LUT	LUT	LUT
Scaling error	0	0	N/A	0	1	1.5	N
Number of transistors	56160	4696	11940	928152	323438	1216816	89196
Area reduction	-	-	-	93.94%	82.63%	95.38%	37.03%

Comparing Table VII of [1] with the proposed design shows that FA-based designs need the least number of transistors, while LUT-based designs require a large number of transistors. Among these designs, the complexity of the proposed design is ranked just after the fully FA-based design in [1]. The proposed scaler has less complexity than all the LUT-based designs, with 37.03% less area. Moreover, it has merits over the FA-based designs in [1] and [126]. The merit over [126] is that the current design is error free but [126] is not even able to produce the correct scaled residue. The merit over [1] is that modular channel three is simplified in the proposed scaler.

Comparison of the estimated unit gate delay in Table VIII of [1] and for the proposed design is shown in Table 5.15. It shows that the proposed design is faster than all LUT-based designs, with a 19.04% delay reduction over [125]. It is even faster than the FA-based design in [126]. The reason lies in the many iterative steps that [126] has used to calculate the scaled residues. The proposed scaler shows a 87.02% delay reduction over [126], and up to 69.64% savings over LUT-based designs.

Table 5.15: Comparison of estimated unit gate delay of proposed hybrid scaler and other designs for $n = 8$, $(255, 256, 257)$, and $M = 16776960$

	Hybrid scaler	[1]	[126]	[125]	[123]	[76]	[95]
Unit gate delay	17	15	131	21	50	26	56
Delay reduction	-	-	87.02%	19.04%	66%	34.61%	69.64%

5.5 Proposing a Scaler for Four-Moduli Set $(2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1)$

Scaling is one of the most important units and a necessary module to avoid overflow in a RNS-based system. Various scalers for three moduli set $(2^n - 1, 2^n, 2^n + 1)$ have been developed and introduced in the literature. Nowadays with the increasing demand for large dynamic range and parallel computing, four-moduli sets are more attractive. In this paper, for the first time, design and implementation of a four-moduli set RNS scaler is presented. In order to retain high performance while decreasing hardware complexity, three main points are considered: First, the four-moduli set $(2^n - 1, 2^n + 1, 2^{2n}, 2^{2n} + 1)$ is selected due its large dynamic range and well-formed moduli set. Second, the Chinese Remainder Theorem with arithmetic simplifications are used to reduce complexity. Finally, a special design of modulo-$2^n + 1$ adder is used to adapt with scaling formulas of the selected moduli set which resulted in efficient design of the proposed scaler. The proposed scaler is synthesised using Synopsys Design Compiler (DC) with $csm18os120\_typ$ Library for $n = 4, 8, 16, 32, 64, 128$ for all components and for separate modular channels.

5.5.1 Proposed Algorithm

One of the common methods to decrease the high power consumption of ROM matrices in ROM-based scaling schemes is to replace them by modular multiplexers. This method, however, increases the cost of implementation drastically, which would be even worse for large moduli sets. As pointed before, hardware cost of all ROM-based scalers increase by increasing the number of moduli. Hence, they can be manipulated with full, adders for less area requirement [15]. The proposed algorithm is based on full-adders and uses CRT for the four-moduli set $(2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1)$. For this moduli set, the CRT (5.1) becomes:

$$X = \Big| m_2 m_3 m_4 \left|M_1^{-1}\right|_{m_1} x_1 + m_1 m_3 m_4 \left|M_2^{-1}\right|_{m_2} x_2 + m_1 m_2 m_4 \left|M_3^{-1}\right|_{m_3} x_3 + m_1 m_2 m_3 \left|M_4^{-1}\right|_{m_4} x_4 \Big|_M \tag{5.43}$$

where x_1, x_2, x_3, and x_4 are n-bit, $2n$-bit, $(n+1)$-bit, and $(2n+1)$-bit integers. Equation (5.43) needs the dynamic range to calculate $M_i = \frac{M}{m_i}$ for $i = 1, 2, 3, 4$. The dynamic range of the four-moduli set $(2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1)$ is the product of its four moduli:

$$M = \prod_{i=1}^{N} m_i = \prod_{i=1}^{4} m_i = (2^n - 1)(2^{2n})(2^n + 1)(2^{2n} + 1) \tag{5.44}$$

For $i = 1$:

$$M_1 = m_2 m_3 m_4 = \frac{(2^n-1)(2^{2n})(2^n+1)(2^{2n}+1)}{(2^n-1)} = (2^{2n})(2^n+1)(2^{2n}+1) \tag{5.45}$$

For $i = 2$:

$$M_2 = m_1 m_3 m_4 = \frac{(2^n-1)(2^{2n})(2^n+1)(2^{2n}+1)}{(2^{2n})} = (2^n-1)(2^n+1)(2^{2n}+1) \tag{5.46}$$

For $i = 3$:

$$M_3 = m_1 m_2 m_4 = \frac{(2^n-1)(2^{2n})(2^n+1)(2^{2n}+1)}{(2^n+1)} = (2^n-1)(2^{2n})(2^{2n}+1) \tag{5.47}$$

For $i = 4$:

$$M_4 = m_1 m_2 m_3 = \frac{(2^n-1)(2^{2n})(2^n+1)(2^{2n}+1)}{(2^{2n}+1)} = (2^n-1)(2^{2n})(2^n+1) \tag{5.48}$$

To calculate the multiplicative inverse, condition (5.49) should be satisfied:

$$M_i^{-1} = \left|M_i \times M_i^{-1}\right|_{m_i} = 1. \tag{5.49}$$

Equation (5.49) for $i = 1, 2, 3, 4$ gives:

$$i = 1 \Rightarrow M_1^{-1} : \left|M_1 \times M_1^{-1}\right|_{m_1} = 1 \Rightarrow \left|(2^n)(2^n+1)(2^{2n}+1) \times M_1^{-1}\right|_{(2^n-1)} = 1$$

$$\Rightarrow M_1^{-1} = (2^{n-2}) \tag{5.50}$$

$$i = 2 \Rightarrow M_2^{-1} : \left|M_2 \times M_2^{-1}\right|_{m_2} = 1 \Rightarrow \left|(2^n-1)(2^n+1)(2^{2n}+1) \times M_2^{-1}\right|_{(2^{2n})} = 1$$

$$\Rightarrow M_2^{-1} = (2^{2n}-1) \tag{5.51}$$

$$i = 3 \Rightarrow M_3^{-1} : \left|M_3 \times M_3^{-1}\right|_{m_3} = 1 \Rightarrow \left|(2^n-1)(2^{2n})(2^{2n}+1) \times M_3^{-1}\right|_{(2^n+1)} = 1$$

$$\Rightarrow M_3^{-1} = (2^{n-2}) \tag{5.52}$$

$$i = 4 \Rightarrow M_4^{-1} : \left|M_4 \times M_4^{-1}\right|_{m_4} = 1 \Rightarrow \left|(2^n-1)(2^{2n})(2^n+1) \times M_4^{-1}\right|_{(2^{2n}+1)} = 1$$

$$\Rightarrow M_4^{-1} = (2^{2n-1}+1) \tag{5.53}$$

Scaling of X in (5.43) using (5.2) gives:

$$\begin{aligned} Y &= \left\lfloor \frac{X}{k} \right\rfloor \\ &= \left\lfloor \frac{1}{k} \left| m_2m_3m_4 \left|M_1^{-1}\right|_{m_1} x_1 + m_1m_3m_4 \left|M_2^{-1}\right|_{m_2} x_2 \right. \right. \\ &\left. \left. + m_1m_2m_4 \left|M_3^{-1}\right|_{m_3} x_3 + m_1m_2m_3 \left|M_4^{-1}\right|_{m_4} x_4 \right|_M \right\rfloor \\ &= \left\lfloor \left| \frac{m_2m_3m_4}{k} \left|M_1^{-1}\right|_{m_1} x_1 + \left|\frac{m_1m_3m_4}{k}\right|_{m_2} x_2 \right. \right. \\ &\left. \left. + \left|\frac{m_1m_2m_4}{k}\right|_{m_3} x_3 + \left|\frac{m_1m_2m_3}{k}\right|_{m_4} x_4 \right|_{\frac{M}{k}} \right\rfloor \end{aligned} \tag{5.54}$$

For the scaling by the second modulus $k = m_2 = 2^{2n}$, (5.54) can be simplified to (5.55):

$$\begin{aligned} Y &= \left| \left\lfloor \frac{X}{k} \right\rfloor \right|_{m_2} = \left\lfloor \frac{X}{m_2} \right\rfloor \\ &= \left| \left| m_3m_4 \left|M_1^{-1}\right|_{m_1} x_1 + \frac{m_1m_3m_4}{m_2} \left|M_2^{-1}\right|_{m_2} x_2 \right. \right. \\ &\left. \left. + m_1m_4 \left|M_3^{-1}\right|_{m_3} x_3 + m_1m_3 \left|M_4^{-1}\right|_{m_4} x_4 \right|_{m_1m_3m_4} \right| \end{aligned} \tag{5.55}$$

Scaling in each channel using Theorem 1 (5.13) for $i = 1, 2, 3, 4$ gives:

$$i = 1 \Rightarrow y_1 = \left| \left\lfloor \frac{X}{k} \right\rfloor \right|_{m_1} = \left| m_3m_4 \left|M_1^{-1}\right|_{m_1} x_1 + \frac{m_1m_3m_4}{m_2} \left|M_2^{-1}\right|_{m_2} x_2 \right|_{m_1} \tag{5.56}$$

$$\begin{aligned} i = 2 \Rightarrow y_2 = \left| \left\lfloor \frac{X}{k} \right\rfloor \right|_{m_2} = \left\lfloor \frac{X}{m_2} \right\rfloor = \left| \left| m_3m_4 \left|M_1^{-1}\right|_{m_1} x_1 + \frac{m_1m_3m_4}{m_2} \left|M_2^{-1}\right|_{m_2} x_2 \right. \right. \\ \left. \left. + m_1m_4 \left|M_3^{-1}\right|_{m_3} x_3 + m_1m_3 \left|M_4^{-1}\right|_{m_4} x_4 \right|_{m_1m_3m_4} \right|_{m_2} \end{aligned} \tag{5.57}$$

$$i = 3 \Rightarrow y_3 = \left| \left\lfloor \frac{X}{k} \right\rfloor \right|_{m_3} = \left| \frac{m_1m_3m_4}{m_2} \left|M_2^{-1}\right|_{m_2} x_2 + m_1m_4 \left|M_3^{-1}\right|_{m_3} x_3 \right|_{m_3} \tag{5.58}$$

$$i = 4 \Rightarrow y_4 = \left| \left\lfloor \frac{X}{k} \right\rfloor \right|_{m_4} = \left| \frac{m_1 m_3 m_4}{m_2} \left| M_2^{-1} \right|_{m_2} x_2 + m_1 m_3 \left| M_4^{-1} \right|_{m_4} x_4 \right|_{m_4} \tag{5.59}$$

Some terms are repeated in (5.56)-(5.59). They can be combined to give:

$$m_3 m_4 \left| M_1^{-1} \right|_{m_1} = (2^n + 1)(2^{2n} + 1)(2^{n-2}) = (2^{4n-2} + 2^{2n-2} + 2^{3n-2} + 2^{n-2}) \tag{5.60}$$

$$m_1 m_4 \left| M_3^{-1} \right|_{m_3} = (2^n - 1)(2^{2n} + 1)(2^{n-2}) = (2^{4n-2} + 2^{2n-2} - 2^{3n-2} - 2^{n-2}) \tag{5.61}$$

$$m_1 m_3 \left| M_4^{-1} \right|_{m_4} = (2^n - 1)(2^n + 1)(2^{2n-1} + 1) = (2^{4n-1} + 2^{2n} - 2^{2n-1} - 1) \tag{5.62}$$

$$\begin{aligned} \frac{m_1 m_3 m_4}{m_2} \left| M_2^{-1} \right|_{m_2} &= \frac{(2^n - 1)(2^n + 1)(2^{2n} + 1)(2^{2n} - 1)}{(2^{2n})} = \frac{(2^{6n} - 2^{2n} - 2^{4n} + 1)}{2^{2n}} \\ &= \frac{2^{2n}(2^{4n} - 1 - 2^{2n} + \frac{1}{2^{2n}})}{2^{2n}} = 2^{4n} - 1 - 2^{2n} + \frac{1}{2^{2n}} = 2^{4n} - 2^{2n} - 1 \end{aligned} \tag{5.63}$$

The (5.60)- (5.63) are divided by $\frac{1}{2^{2n}}$ to derive new vaiables as $k_1 = 2^{4n-2} + 2^{2n-2} + 2^{3n-2} + 2^{n-2}$, $k_2 = 2^{4n} - 2^{2n} - 1$, $k_3 = 2^{4n-2} + 2^{2n-2} - 2^{3n-2} - 2^{n-2}$, $k_4 = 2^{4n-1} + 2^{2n} - 2^{2n-1} - 1$, and also $p = m_1 m_3 m_4$. Hence:

$$\begin{aligned} \left\lfloor \frac{X}{S} \right\rfloor' &= \\ &= \Big\lfloor \Big| (2^{4n-2} + 2^{2n-2} + 2^{3n-2} + 2^{n-2}) x_1 + (2^{4n} - 2^{2n} - 1) x_2 \\ &\quad + (2^{4n-2} + 2^{2n-2} - 2^{3n-2} - 2^{n-2}) x_3 + (2^{4n-1} + 2^{2n} - 2^{2n-1} - 1) x_4 \Big|_{m_1 m_3 m_4} \Big\rfloor \end{aligned} \tag{5.64}$$

$$\left\lfloor \frac{X}{S} \right\rfloor' = \left\lfloor \left| k_1 x_1 + k_2 x_2 + k_3 x_3 + k_4 x_4 \right|_p \right\rfloor \tag{5.65}$$

Using (5.14), the above equation (5.65) is reconstituted as:

$$\left\lfloor \frac{X}{S} \right\rfloor' = \left\lfloor |k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4 - \varepsilon p|_p \right\rfloor \quad = k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4 - \varepsilon p \tag{5.66}$$

where ε is a nonnegative digit.

Without replacing k_1,k_2,k_3 and k_4, it would be:

$$\begin{aligned}\left\lfloor \frac{X}{S} \right\rfloor &= \\ &= \bigg\lfloor \bigg|(2^{4n-2} + 2^{2n-2} + 2^{3n-2} + 2^{n-2})x_1 + (2^{4n} - 2^{2n} - 1 + \frac{1}{2^{2n}})x_2 \\ &+ (2^{4n-2} + 2^{2n-2} - 2^{3n-2} - 2^{n-2})x_3 + (2^{4n-1} + 2^{2n} - 2^{2n-1} - 1)x_4\bigg|_{m_1m_3m_4}\bigg\rfloor \\ &= \left\lfloor \left| k_1x_1 + (k_2 + \frac{1}{2^{2n}})x_2 + k_3x_3 + k_4x_4 \right|_p \right\rfloor \\ &= \left\lfloor \left| k_1x_1 + k_2x_2 + \frac{x_2}{2^{2n}} + k_3x_3 + k_4x_4 \right|_p \right\rfloor\end{aligned} \tag{5.67}$$

If $c = \frac{x_2}{2^{2n}}$, then (5.67) is simplified as follow:

$$\begin{aligned}\left\lfloor \frac{X}{S} \right\rfloor &= \left\lfloor |k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4 + c|_p \right\rfloor \\ &= \lfloor k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4 + c - \varepsilon p \rfloor\end{aligned} \tag{5.68}$$

$$\left\lfloor \frac{X}{S} \right\rfloor = k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4 - \varepsilon p + \lfloor c \rfloor \tag{5.69}$$

Since $0 \leq x_2 \leq 2^{2n}$, then:

$$c = \frac{x_2}{2^{2n}} \Rightarrow 0 \leq c < 1 \Rightarrow \lfloor c \rfloor = 0 \tag{5.70}$$

which concludes that:

$$\left\lfloor \frac{X}{S} \right\rfloor' = \left\lfloor \frac{X}{S} \right\rfloor \tag{5.71}$$

Using (5.71) and Notes in the following, modular channels y_i of proposed scaler are calculated.

Note 1: If a k bite digit (such as y) is multiplied by a number of the form 2^n, the $|2^n y|_{2^k-1}$ is the n-bit circular left shift of y.

Note 2: If y is negative, then $|-y|_{2^k-1}$ is the two's complement of y.

5.5.2 Generating y_1

Substituting (5.60) and (5.63) in (5.56), the scaled residue y_1 splits into two terms, A and B:

$$y_1 = \left|\left\lfloor \frac{X}{k} \right\rfloor\right|_{m_1} = \left|\overbrace{(2^n+1)(2^{2n}+1)(2^{n-2})}^{A}\, x_1 + \overbrace{(2^{4n}-2^{2n}-1)}^{B}\, x_2\right|_{2^n-1} \tag{5.72}$$

where

$$A = \left|(2^n+1)(2^{2n}+1)(2^{n-2})x_1\right|_{2^n-1} \tag{5.73}$$

Using (5.13), A is given by:

$$\Rightarrow \begin{cases} |(2^n+1)|_{2^n-1} = 2 \\ |(2^{2n}+1)|_{2^n-1} = |2^n \times 2^n + 1|_{2^n-1} = 2 \\ |(2^{n-2})|_{2^n-1} = 2^{n-2} \\ |x_1|_{2^n-1} \end{cases}$$

$$\Rightarrow \left|(2 \times 2 \times 2^{n-2} \times x_1)\right|_{2^n-1} = |x_1|_{2^n-1} \tag{5.74}$$

and similarly

$$B : \left|(2^{4n} - 2^{2n} - 1)x_2\right|_{2^n-1}$$

$$\Rightarrow \left|(2^{4n} - 2^{2n} - 1)x_2\right|_{2^n-1} = |(1-1-1)x_2|_{2^n-1} = |-x_2|_{2^n-1} \tag{5.75}$$

Using (5.74) and (5.75), the scaled residue y_1 is given by

$$y_1 = \left|\left\lfloor \frac{X}{k} \right\rfloor\right|_{m_1} = |x_1 - x_2|_{2^n-1}. \tag{5.76}$$

Since x_2 is a $2n$-bit negative integer, we split (5.76) to x_{21} and x_{22}, and express it as (5.131), where $\bar{x}_2$ is the two's complement of x_2. Figure 5.9 shows the proposed

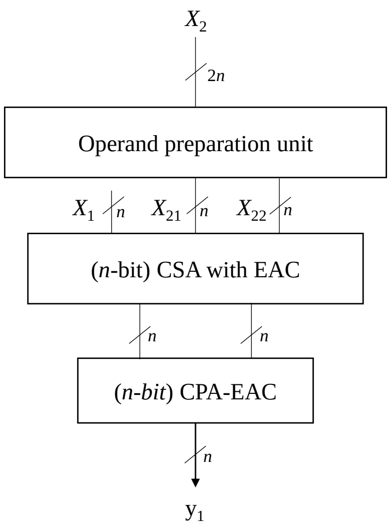


Figure 5.9: Proposed method to generate y_1 in the moduli set $(2^n-1, 2^{2n}, 2^n+1, 2^{2n}+1)$

method to generate y_1.

$$y_1 = (\overbrace{x_{1,n-1}\ldots\ldots x_{1,0}}^{n}) + (\underbrace{\overbrace{\overline{x_{2,2n-1}}\ldots\ldots\overline{x_{2,n}}}^{n}}_{x_{21}}) + (\underbrace{\overbrace{\overline{x_{2,n-1}}\ldots\ldots\overline{x_{2,0}}}^{n}}_{x_{22}}) \tag{5.77}$$

5.5.3 Generating y_2

Substituting (5.60), (5.61) and (5.62) in (5.57) results in the scaled residue y_2 that can be split into four terms: y_{21}, y_{22}, y_{23} and y_{24} .

$$\begin{aligned} y_2 &= \left|\left\lfloor \frac{X}{k} \right\rfloor\right|_{m_2} \\ &= \left|\left|\underbrace{(2^n+1)(2^{2n}+1)(2^{n-2})}_{y_{21}} x_1 + \underbrace{(\frac{(2^n-1)(2^n+1)(2^{2n}+1)(2^{2n}-1)}{(2^{2n})})}_{y_{22}} x_2 \right.\right. \\ &\left.\left. + \underbrace{(2^n-1)(2^{2n}+1)(2^{n-2})}_{y_{23}} x_3 + \underbrace{(2^n-1)(2^n+1)(2^{2n-1}+1)}_{y_{24}} x_4 \right|_{(2^n-1)(2^n+1)(2^{2n}+1)}\right|_{2^{2n}} \end{aligned} \tag{5.78}$$

$$\Rightarrow y_2 = |y_{21} + y_{22} + y_{23} + y_{24}|_{2^{2n}} \tag{5.79}$$

5.5.3.1 Generating y_{21}

The first term in (5.78) splits into four terms A, B, C, and D.

$$y_{21} = \left|(\overbrace{2^{4n-2}}^{A} + \overbrace{2^{2n-2}}^{B} + \overbrace{2^{3n-2}}^{C} + \overbrace{2^{n-2}}^{D})x_1\right|_{2^{4n}-1} \tag{5.80}$$

Each term modulo-$(2^{4n}-1)$ is given by:

$$A = \left|2^{4n-2}x_1\right|_{2^{4n}-1} = \left|2^{4n-2}(\overbrace{00......0}^{3n}, \overbrace{x_{1,n-1}......x_{1,0}}^{n})\right|_{2^{4n}-1} = (\overbrace{x_{1,1}x_{1,0}}^{2}, \overbrace{00......0}^{3n}, \overbrace{x_{1,n-1}......x_{1,2}}^{n-2}) \tag{5.81}$$

$$B = \left|2^{2n-2}x_1\right|_{2^{4n}-1} = \left|2^{2n-2}(\overbrace{00......0}^{3n}, \overbrace{x_{1,n-1}......x_{1,0}}^{n})\right|_{2^{4n}-1} = (\overbrace{00......0}^{(n+2)}, \overbrace{x_{1,n-1}......x_{1,0}}^{n}, \overbrace{00......0}^{(2n-2)} \tag{5.82}$$

$$C = \left|2^{3n-2}x_1\right|_{2^{4n}-1} = \left|2^{3n-2}(\overbrace{00......0}^{3n}, \overbrace{x_{1,n-1}......x_{1,0}}^{n})\right|_{2^{4n}-1} = (\overbrace{\ 00\ }^{2}, \overbrace{x_{1,n-1}......x_{1,0}}^{n}, \overbrace{00......0}^{(3n-2)}) \tag{5.83}$$

$$D = \left|2^{n-2}x_1\right|_{2^{4n}-1} = \left|2^{n-2}(\overbrace{00......0}^{3n}, \overbrace{x_{1,n-1}......x_{1,0}}^{n})\right|_{2^{4n}-1} = (\overbrace{00......0}^{(2n+2)}, \overbrace{x_{1,n-1}......x_{1,0}}^{n}, \overbrace{00......0}^{(n-2)}) \tag{5.84}$$

Replacing terms A, B, C, and D in (5.80) gives

$$y_{21} = |(A + B + C + D)x_1|_{2^{4n}-1}. \tag{5.85}$$

Substituting (5.81) to (5.84) in (5.85) gives

$$y_{21} = (\overbrace{x_{1,1}x_{1,0}}^{2}, \overbrace{x_{1,n-1}......x_{1,0}}^{n}, \overbrace{x_{1,n-1}......x_{1,0}}^{n}, \overbrace{x_{1,n-1}......x_{1,0}}^{n}, \overbrace{x_{1,n-1}......x_{1,2}}^{(n-2)}) \tag{5.86}$$

5.5.3.2 Generating y_{22}

The second term in (5.78) splits into three terms, A, B, and C:

$$y_{22} = \left|\left|(\frac{(2^n-1)(2^n+1)(2^{2n}+1)(2^{2n}-1)}{(2^{2n})})x_2\right|_{(2^n-1)(2^n+1)(2^{2n}+1)}\right|_{2^{2n}} \tag{5.87}$$

$$y_{22} = \left|(\overbrace{2^{4n}}^{A} - \overbrace{2^{2n}}^{B} - \overbrace{1}^{C})x_2\right|_{2^{4n-1}} \tag{5.88}$$

Modulo-$(2^{4n}-1)$, each term is given by:

$$A = \left|2^{4n}x_2\right|_{2^{4n}-1} = \left|2^{4n}(\overbrace{00......0}^{2n}, \overbrace{x_{2,2n-1}......x_{2,0}}^{2n})\right|_{2^{4n}-1} = (\overbrace{00......0}^{2n}, \overbrace{x_{2,2n-1}......x_{2,0}}^{2n}) \tag{5.89}$$

$$B = \left|-2^{2n}x_2\right|_{2^{4n}-1} = \left|-2^{2n}(\overbrace{00......0}^{2n}, \overbrace{x_{2,2n-1}......x_{2,0}}^{2n})\right|_{2^{4n}-1} = (\overbrace{\overline{x_{2,2n-1}}......\overline{x_{2,0}}}^{2n}, \overbrace{11......1}^{2n}) \tag{5.90}$$

$$C = \left|-x_2\right|_{2^{4n}-1} = \left|-(\overbrace{00......0}^{2n}, \overbrace{x_{2,2n-1}......x_{2,0}}^{2n})\right|_{2^{4n}-1} = (\overbrace{11......1}^{2n}, \overbrace{\overline{x_{2,2n-1}}......\overline{x_{2,0}}}^{2n}) \tag{5.91}$$

By swapping the $2n$ MSBs of B and C, they will be:

$$B = (\overbrace{\overline{11......1}}^{2n}, \overbrace{11......1}^{2n}) \tag{5.92}$$

$$C = (\overbrace{\overline{x_{2,2n-1}}\ldots\ldots\overline{x_{2,0}}}^{2n}\,\overbrace{\overline{x_{2,2n-1}}\ldots\ldots\overline{x_{2,0}}}^{2n}) \tag{5.93}$$

Hence $A + C = (\overbrace{\overline{x_{2,2n-1}}\ldots\ldots\overline{x_{2,0}}}^{2n}, \overbrace{11\ldots\ldots1}^{2n})$. Replacing A, B and C in (5.87) gives

$$y_{22} = |A + B + C|_{2^{4n}-1} = (\overbrace{\overline{x_{2,2n-1}}\ldots\ldots\overline{x_{2,0}}}^{2n}, \overbrace{11\ldots\ldots1}^{2n}) \tag{5.94}$$

5.5.3.3 Generating y_{23}

To calculate y_{23}, we use the third term in (5.78) and split it into four terms A, B, C, and D:

$$y_{23} = \left|(2^n - 1)(2^{2n} + 1)(2^{n-2})x_3\right|_{2^{4n}-1} \tag{5.95}$$

$$y_{23} = \left|(\overbrace{2^{4n-2}}^{A} + \overbrace{2^{2n-2}}^{B} - \overbrace{2^{3n-2}}^{C} - \overbrace{2^{n-2}}^{D})x_3\right|_{2^{4n}-1} \tag{5.96}$$

Modulo-$(2^{4n} - 1)$, each term is calculated as:

$$A = \left|(2^{4n-2})x_3\right|_{2^{4n}-1} = \left|2^{4n-2}(\overbrace{00\ldots\ldots0}^{(3n-1)}, \overbrace{x_{3,n}\ldots\ldots x_{3,0}}^{(n+1)})\right|_{2^{4n}-1} = (\overbrace{x_{3,1}x_{3,0}}^{2}, \overbrace{00\ldots\ldots0}^{(3n-1)}, \overbrace{x_{3,n}\ldots\ldots x_{3,2}}^{(n-1)}) \tag{5.97}$$

$$B = \left|(2^{2n-2})x_3\right|_{2^{4n}-1} = \left|2^{2n-2}(\overbrace{00\ldots\ldots0}^{(3n-1)}, \overbrace{x_{3,n}\ldots\ldots x_{3,0}}^{(n+1)})\right|_{2^{4n}-1} = (\overbrace{00\ldots\ldots0}^{(n+1)}, \overbrace{x_{3,n}\ldots\ldots x_{3,0}}^{(n+1)}, \overbrace{00\ldots\ldots0}^{(2n-2)}) \tag{5.98}$$

$$C = \left|(-2^{3n-2})x_3\right|_{2^{4n}-1} = \left|-2^{3n-2}(\overbrace{00\ldots\ldots0}^{(3n-1)}, \overbrace{x_{3,n}\ldots\ldots x_{3,0}}^{(n+1)})\right|_{2^{4n}-1} = (\overbrace{\;1\;}^{(1)}, \overbrace{\overline{x_{3,n}}\ldots\ldots\overline{x_{3,0}}}^{(n+1)}, \overbrace{11\ldots\ldots1}^{(3n-2)}) \tag{5.99}$$

$$D = \left|(-2^{n-2})x_3\right|_{2^{4n}-1} = \left|-2^{n-2}(\overbrace{00......0}^{(3n-1)}, \overbrace{x_{3,n}......x_{3,0}}^{(n+1)})\right|_{2^{4n}-1}$$
$$= (\overbrace{11......1}^{(2n+1)}, \overbrace{\overline{x_{3,n}}......\overline{x_{3,0}}}^{(n+1)}, \overbrace{11......1}^{(n-2)}) \tag{5.100}$$

For simplification the last MSB of A and C are swapped as:

$$A = (\overbrace{1x_{3,0}}^{2}, \overbrace{00......0}^{(3n-1)}, \overbrace{x_{3,n}......x_{3,2}}^{(n-1)}) \tag{5.101}$$

$$C = (\overbrace{x_{3,1}}^{(1)}, \overbrace{\overline{x_{3,n}}......\overline{x_{3,0}}}^{(n+1)}, \overbrace{11......1}^{(3n-2)}) \tag{5.102}$$

Hence:

$$A + B = (\overbrace{1x_{3,0}}^{2}, \overbrace{00......0}^{(n-1)}, \overbrace{x_{3,n}......x_{3,0}}^{(n+1)}, \overbrace{00......0}^{(n-1)}, \overbrace{x_{3,n}......x_{3,2}}^{(n-1)}) \tag{5.103}$$

Similarly by swapping the $n+2$ MSBs of C and D, the result of $C+D$ would be:

$$C + D = (\overbrace{x_{3,1}}^{1}, \overbrace{\overline{x_{3,n}}......\overline{x_{3,0}}}^{(n+1)}, \overbrace{11......1}^{(n-1)}, \overbrace{\overline{x_{3,n}}......\overline{x_{3,0}}}^{(n+1)}, \overbrace{11......1}^{(n-2)}) \tag{5.104}$$

Replacing (5.103) to (5.104) in (5.95) gives

$$y_{23} = |(A+B+C+D)|_{2^{4n}-1}$$
$$= \overbrace{1x_{3,0}}^{2}, \overbrace{00......0}^{(n-1)}, \overbrace{x_{3,n}......x_{3,0}}^{(n+1)}, \overbrace{00......0}^{(n-1)}, \overbrace{x_{3,n}......x_{3,2}}^{(n-1)}$$
$$+ (\overbrace{x_{3,1}}^{1}, \overbrace{\overline{x_{3,n}}......\overline{x_{3,0}}}^{(n+1)}, \overbrace{11......1}^{(n-1)}, \overbrace{\overline{x_{3,n}}......\overline{x_{3,0}}}^{(n+1)}, \overbrace{11......1}^{(n-2)}) \tag{5.105}$$

5.5.3.4 Generating y_{24}

To calculate y_{21}, we use the last term in (5.78) and split it into four terms A, B, C, and D:

$$y_{24} = \left|(2^n-1)(2^n+1)(2^{2n-1}+1)x_4\right|_{2^{4n}-1} \tag{5.106}$$

$$y_{24} = \left|(\overbrace{2^{4n-1}}^{y_{241}} + \overbrace{2^{2n}}^{y_{242}} - \overbrace{2^{2n-1}}^{y_{243}} - \overbrace{1}^{y_{244}})x_4\right|_{2^{4n}-1} \tag{5.107}$$

Each term modulo-$(2^{4n}-1)$ gives

$$y_{241} = \left|(2^{4n-1})x_4\right|_{2^{4n}-1} = \left|2^{4n-1}(\overbrace{00......0}^{(2n-1)}, \overbrace{x_{4,2n}......x_{4,0}}^{(2n+1)})\right|_{2^{4n}-1} = (\overbrace{x_{4,0}}^{1}, \overbrace{00......0}^{(2n-1)}, \overbrace{x_{4,2n}......x_{4,1}}^{(2n)}) \tag{5.108}$$

$$y_{242} = \left|(2^{2n})x_4\right|_{2^{4n}-1} = \left|2^{2n}(\overbrace{00......0}^{(2n-1)}, \overbrace{x_{4,2n}......x_{4,0}}^{(2n+1)})\right|_{2^{4n}-1} = (\overbrace{x_{4,2n-1}......x_{4,0}}^{(2n)}, \overbrace{00......0}^{(2n-1)}, \overbrace{x_{4,2n}}^{1}) \tag{5.109}$$

$$y_{243} = \left|(-2^{2n-1})x_4\right|_{2^{4n}-1} = \left|-2^{2n-1}(\overbrace{00......0}^{(2n-1)}, \overbrace{x_{4,2n}......x_{4,0}}^{(2n+1)})\right|_{2^{4n}-1} = (\overbrace{\overline{x_{4,2n}}......\overline{x_{4,0}}}^{(2n+1)}, \overbrace{11......1}^{(2n-1)}) \tag{5.110}$$

$$y_{244} = \left|-x_4\right|_{2^{4n}-1} = \left|-(\overbrace{00......0}^{(2n-1)}, \overbrace{x_{4,2n}......x_{4,0}}^{(2n+1)})\right|_{2^{4n}-1} = (\overbrace{11......1}^{(2n-1)}, \overbrace{\overline{x_{4,2n}}......\overline{x_{4,0}}}^{(2n+1)}) \tag{5.111}$$

Replacing (5.108) to (5.111) in (5.106) gives

$$\begin{aligned} y_{24} &= \left|(y_{241} + y_{242} + y_{243} + y_{244})\right|_{2^{4n}-1} \\ &= (\overbrace{x_{4,0}}^{1}, \overbrace{00......0}^{(2n-1)}, \overbrace{x_{4,2n}......x_{4,1}}^{(2n)}) + (\overbrace{x_{4,2n-1}......x_{4,0}}^{(2n)}, \overbrace{00......0}^{(2n-1)}, \overbrace{x_{4,2n}}^{1}) \\ &+ (\overbrace{\overline{x_{4,2n}}......\overline{x_{4,0}}}^{(2n+1)}, \overbrace{11......1}^{(2n-1)}) + (\overbrace{11......1}^{(2n-1)}, \overbrace{\overline{x_{4,2n}}......\overline{x_{4,0}}}^{(2n+1)}) \end{aligned} \tag{5.112}$$

Figure 5.10 shows the proposed method to generate y_2.

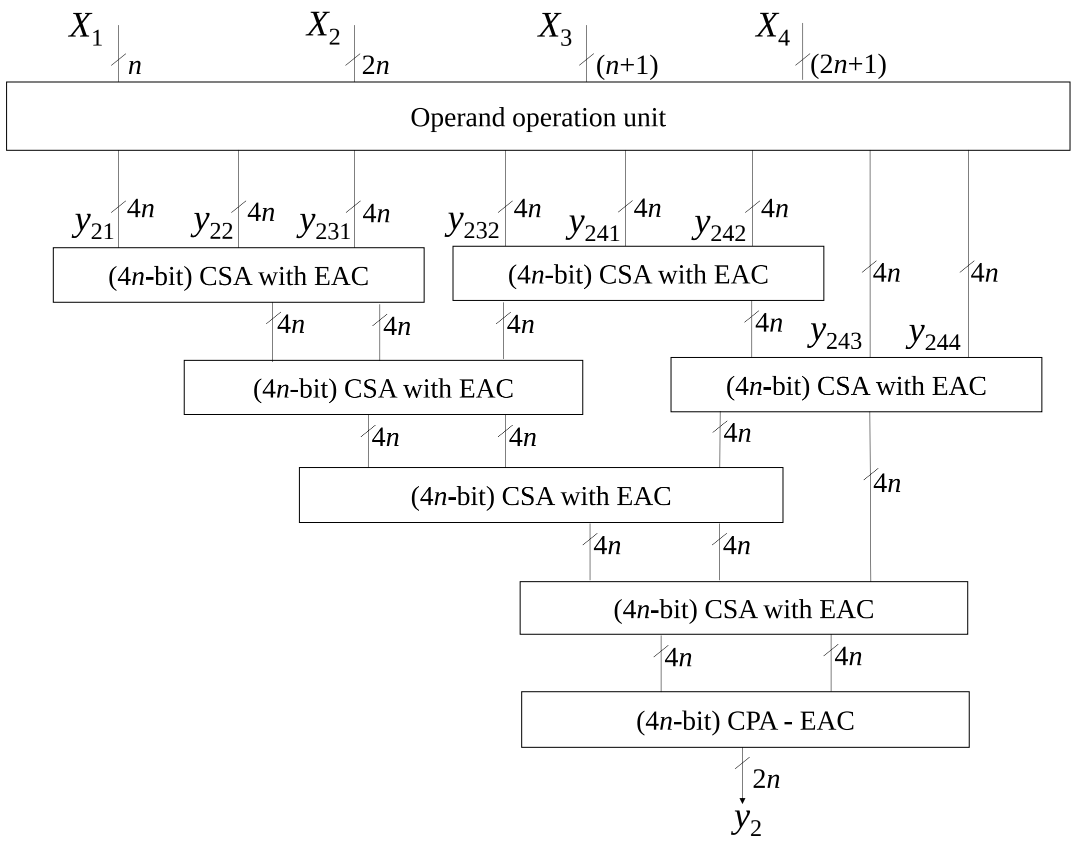


FIGURE 5.10: Proposed method to generate y_2 in moduli set $(2^n-1, 2^{2n}, 2^n+1, 2^{2n}+1)$

5.5.4 Generating y_3

Replacing (5.61) and (5.63) in (5.58), the scaled residue y_3 splits into two terms, A and B:

$$\begin{aligned} y_3 &= \left|\left\lfloor \frac{X}{k} \right\rfloor\right|_{m_3} = \left| \frac{m_1 m_3 m_4}{m_2} \left|M_2^{-1}\right|_{m_2} x_2 + m_1 m_4 \left|M_3^{-1}\right|_{m_3} x_3 \right|_{m_3} \\ &= \left|(2^{4n} - 2^{2n} - 1)x_2 + (2^n - 1)(2^{2n}+1)(2^{n-2})x_3\right|_{2^n+1} \end{aligned} \tag{5.113}$$

where A and B can be simplified to:

$$A : \left|(2^{4n} - 2^{2n} - 1)x_2\right|_{2^n+1} = \left|(+1 - 1 - 1)x_2\right|_{2^n+1} = \left|-x_2\right|_{2^n+1} \tag{5.114}$$

$$B : \begin{cases} |(2^n - 1)|_{2^n+1} = -2 \\ |(2^{2n} + 1)|_{2^n+1} = \left|(2^n \times 2^n + 1)\right|_{2^n+1} = 2 \\ |(2^{n-2})|_{2^n+1} = 2^{n-2} \end{cases}$$

$$\Rightarrow \left|(-4)(2^{n-2})x_3\right|_{2^n+1} = |(-2^n)x_3|_{2^n+1} = |x_3|_{2^n+1} \tag{5.115}$$

Substituting (5.114) and (5.115) in (5.113) gives

$$A, B \Rightarrow y_3 = \left|-x_2 + x_3\right|_{2^n+1} \tag{5.116}$$

To design y_3 we use (5.6):

$$\begin{aligned} y_3 = \left|-x_2 + x_3\right|_{2^n+1} &= \left|(-2^n(\overbrace{x_{2,2n-1}......x_{2,n}}^{n})) - (\overbrace{x_{2,n-1}......x_{2,0}}^{n}) + x_3\right|_{2^n+1} \\ &= \left|(x_{2,2n-1}......x_{2,n}) + (\overbrace{(\overline{x_{2,n-1}}......\overline{x_{2,0}}) + 2}^{(Axiom: |-z|_{2^n+1} = \overline{z}+2)}) + x_3\right|_{2^n+1} \end{aligned} \tag{5.117}$$

$$\begin{aligned} |x_3|_{2^n+1} = \left|x_{3,n} \times 2^n + \overbrace{x_{3,n-1}......x_{3,0}}^{x_3^*}\right|_{2^n+1} &= |-x_{3,n} + x_3^*|_{2^n+1} = |(\overline{x_{3,n}} + 2) + x_3^*|_{2^n+1} \\ = \left|(x_{2,2n-1}......x_{2,n}) + ((\overline{x_{2,n-1}}......\overline{x_{2,0}}) + 2) + (\overline{x_{3,n}} + 2) + x_3^*\right|_{2^n+1} \end{aligned}$$

$$\begin{aligned} \Rightarrow y_3 &= \left|(x_{2,2n-1}......x_{2,n}) + (\overline{x_{2,n-1}}......\overline{x_{2,0}}) + \overline{x_{3,n}} + x_3^* + 4\right|_{2^n+1} \\ &= \left|(x_{2,2n-1}......x_{2,n}) + (\overline{x_{2,n-1}}......\overline{x_{2,0}}) + x_3^* + x_3^{**}\right|_{2^n+1} \end{aligned} \tag{5.118}$$

where

$$\begin{aligned} x_3^{**} = \overline{x_{3,n}} + 4 = \left|(\overbrace{11......1}^{(n-1)bits}, \overbrace{\overline{x_{3,n}}}^{1bit}) + (\overbrace{00......0}^{(n-3)bits}\overbrace{100}^{3bits})\right|_{2^n+1} &= \left|(\overbrace{00......0}^{(n-2)bits}\overbrace{1\overline{x_{3,n}}}^{1bit}) + 2^n\right|_{2^n+1} \\ = \left|(\overbrace{00......0}^{(n-2)bits}\overbrace{1\overline{x_{3,n}}}^{1bit}) - 1\right|_{2^n+1} \end{aligned} \tag{5.119}$$

$x_{3,n}$ can be either zero or one. Hence (5.119) can be simplified to:

$$x_{3,n}=1 \Rightarrow (\overline{x_{3,n}}=0, x_3^*=0) \rightarrow y_3 = \left|(x_{2,2n-1}......x_{2,n}) + (\overline{x_{2,n-1}}......\overline{x_{2,0}}) + 1\right|_{2^n+1} \tag{5.120}$$

$$x_{3,n}=0 \Rightarrow (\overline{x_{3,n}}=1, x_3^*) \rightarrow y_3 = \left|(x_{2,2n-1}......x_{2,n}) + (\overline{x_{2,n-1}}......\overline{x_{2,0}}) + x_3^* + 2\right|_{2^n+1} \tag{5.121}$$

Figure 5.11 shows the proposed method to generate y_3.

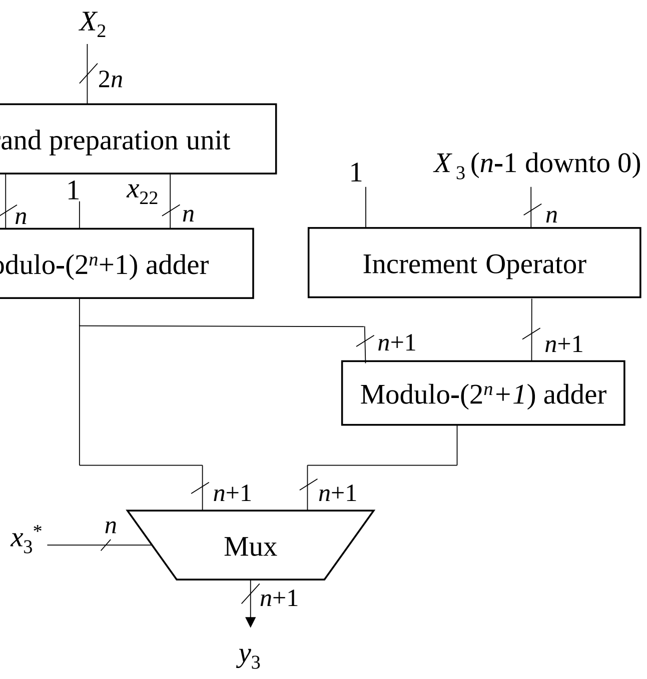


FIGURE 5.11: Proposed method to generate y_3 in moduli set $(2^n-1, 2^{2n}, 2^n+1, 2^{2n}+1)$

5.5.4.1 A New Modulo-(2^n+1) Adder

The proposed method to calculate y_3 in Figure 5.11 uses (x_{21}) and (x_{22}), which are n-bit numbers. However, existing modulo-(2^n+1) adders use $(n+1)$-bit numbers. To solve this problem, we propose a new modulo-(2^n+1) adder which can be used as a modulo-$(2^{2n}+1)$ adder as well. The operation of the proposed adder for two residues

a and b is:

$$|a+b|_{2^n+1} = \begin{cases} a+b+1-(2^n+1) & if(a+b+1) > 2^n+1 \\ a+b+1 & otherwise \end{cases} \tag{5.122}$$

Equation (5.122) can be shortened to

$$|a+b|_{2^n+1} = \begin{cases} a+b-2^n & if(a+b+1) > 2^n \\ a+b+1 & otherwise \end{cases} \tag{5.123}$$

To add two n-bit numbers and subtract 2^n from them in (5.123), all we need to do is to use the n LSB bits of that number. Hence, to implement the proposed adder, we use a multiplexer as shown in Figure 5.12, controlled by the carry out from the (n-bit) CPA adder. Equation (5.121) can be written, based on x_{21} and x_{22}, as

$$x_{3,n} = 0 \Rightarrow (\overline{x_{3,n}} = 1, x_3^*) \rightarrow y_3 = \left| \overbrace{(\overline{x_{2,2n-1}} \ldots\ldots \overline{x_{2,n}})}^{x_{21}} + \overbrace{(\overline{x_{2,n-1}} \ldots\ldots \overline{x_{2,0}})}^{x_{22}} + 1 + x_3^* + 1 \right|_{2^n+1} \tag{5.124}$$

We split (5.124) into two terms y_{31} and y_{32}. A multiplexer will choose between these two possible inputs depending on the carry out from the CPA adder.

$$y_{31} = \left| \overbrace{(\overline{x_{2,2n-1}} \ldots\ldots \overline{x_{2,n}})}^{x_{21}} + \overbrace{(\overline{x_{2,n-1}} \ldots\ldots \overline{x_{2,0}})}^{x_{22}} + 1 \right|_{2^n+1} \tag{5.125}$$

$$y_{32} = |x_3^* + 1|_{2^n+1} \tag{5.126}$$

Equation (5.125) has already been calculated in (5.120). To calculate (5.125), the unit increment is used since x_3^* is n-bit. A final modulo-2^n+1 adder is used for final addition. The selection of one of the (5.125) and (5.126) is done using a multiplexer.

5.5.5 Generating y_4

The scaled integer y_4 is calculated using (5.59), which can be split into two terms, A and B:

$$\begin{aligned} y_4 = \left| \left\lfloor \frac{X}{k} \right\rfloor \right|_{m_4} &= \left| \frac{m_1 m_3 m_4}{m_2} \left| M_2^{-1} \right|_{m_2} x_2 + m_1 m_3 \left| M_4^{-1} \right|_{m_4} x_4 \right|_{m_4} \\ &= \left| (2^{4n} - 2^{2n} - 1) x_2 + (2^n - 1)(2^n + 1)(2^{2n-1} + 1) x_4 \right|_{2^{2n}+1} \end{aligned} \tag{5.127}$$

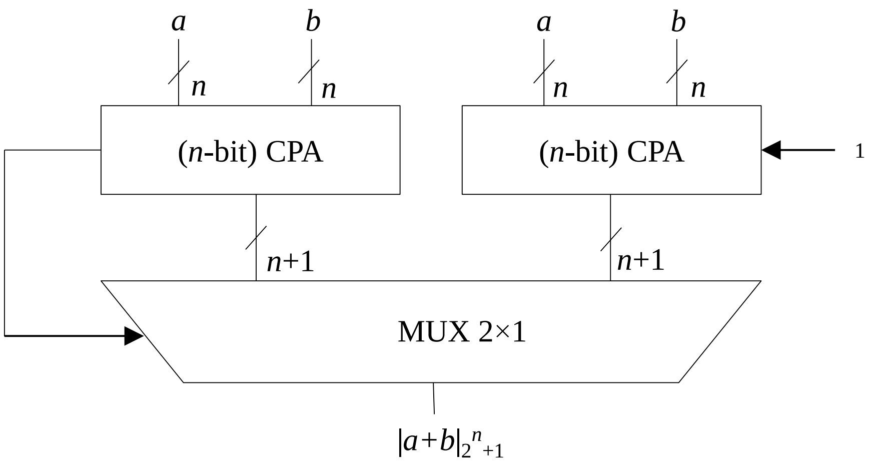


Figure 5.12: Proposed modulo-(2^n+1) adder

Modulo-$2^{2n}-1$, each term gives:

$$A : \left|(2^{4n}-2^{2n}-1)x_2\right|_{2^{2n}+1} = |(1-(-1)-1)x_2|_{2^{2n}+1} = |x_2|_{2^{2n}+1} \tag{5.128}$$

$$\begin{aligned} B : \left|(2^n-1)(2^n+1)(2^{2n-1}+1)x_4\right|_{2^{2n}+1} &= \left|(2^{4n-1}+2^{2n}-2^{2n-1}-1)x_4\right|_{2^{2n}+1} \\ &= \left|((\underbrace{2^{4n}\times\frac{1}{2}}_{\frac{1}{2}}) + (\underbrace{2^{2n}}_{-1}) - (\underbrace{2^{2n}\times\frac{1}{2}}_{-\frac{1}{2}})) - 1)x_4\right|_{2^{2n}+1} \\ &= |-x_4|_{2^{2n}+1} \end{aligned} \tag{5.129}$$

Substituting (5.128) and (5.129) in (5.127) gives:

$$A, B \Rightarrow y_4 = |x_2 - x_4|_{2^{2n}+1} \tag{5.130}$$

$$y_1 = (\overbrace{x_{1,n-1}\ldots\ldots x_{1,0}}^{n}) + (\underbrace{\overbrace{x_{2,2n-1}\ldots\ldots x_{2,n}}^{n}}_{x_{21}}) + (\underbrace{\overbrace{x_{2,n-1}\ldots\ldots x_{2,0}}^{n}}_{x_{22}}) \tag{5.131}$$

$$y_4 = \left| \underbrace{\overbrace{(x_{2,2n-1}......x_{2,0})}^{(2n)}}_{x_2} + \underbrace{\overbrace{\left(\overbrace{00.....0}^{(n-1)} \overbrace{x_{4,2n}}^{1} \right)}^{(2n)}}_{x_{41}} + \underbrace{\overbrace{(\overline{x_{4,2n-1}}......\overline{x_{4,0}})}^{(2n)}}_{x_{42}} + 2 \right|_{2^{2n}+1} \tag{5.132}$$

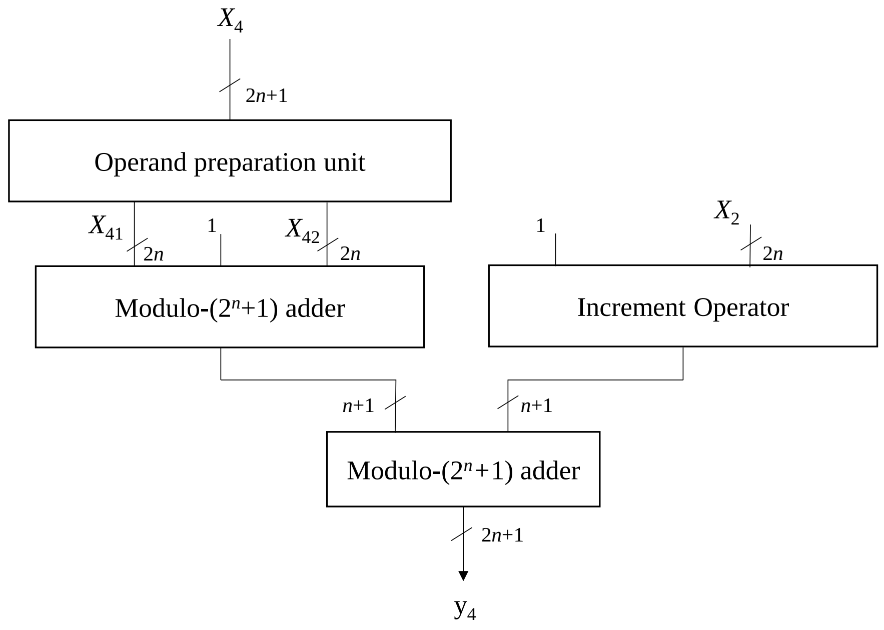


FIGURE 5.13: Proposed method to generate y_4 in moduli set $(2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1)$

Example Table 5.16 shows a numerical example of the proposed algorithm for $n = 2$. Consider the residues $x_1 = 2$, $x_2 = 15$, $x_3 = 2$ and $x_4 = 13$ corresponding to moduli set (3, 16, 5, 17). The dynamic range of the system is M =4080, and $M_1 = 1360$, $M_2 = 255$, $M_3 = 816$ and $M_4 = 240$. The multiplicative inverses are $M_1^{-1} = 1$, $M_2^{-1} = 15$, $M_3^{-1} = 1$, and $M_4^{-1} = 9$. The scaling factor is $k = 2^{2n}$ for $n = 2$. The scaling is performed for each channel using (5.76), (5.79), (5.116) and (5.130).

Table 5.16: Numerical example of proposed scaler for four-moduli set (3, 16, 5, 17) and scaling factor 16

	Channel 1	Channel 2	Channel 3	Channel 4
m_i	3	16	5	17
x_i	2	15	2	13
M_i	1360	255	816	240
M_i^{-1}	1	15	1	9
y_i	2	2	2	2

5.5.6 Performance Evaluation

Synthesis results of proposed scaler for the four moduli set $(2^n-1, 2^{2n}, 2^n+1, 2^{2n}+1)$ is presented in this section. The proposed four-moduli scaler is designed using Xilinx tools in such a way that the modular channels (Scale1, Scale2, Scale3, Scale4) can be synthesised separately. The proposed scaler is designed VHDL code, and performance and accurate functionality of the design are investigated using Modelsim simulation tool.

It is also synthesised using Synopsys Design Compiler (DC) with $csm18os120\_typ$ (WORST case operating condition and 1.62 V) library. For comparison purpose, the Chang scaler [1] is also designed and synthesised using DC. The proposed scaler and Chang scaler are designed using generic variables, hence they are synthesised for $n = 8, 16, 32, 64, 128$.

The proposed scaler is designed for four moduli set $(2^n-1, 2^{2n}, 2^n+1, 2^{2n}+1)$ which has $6n$ bits dynamic range. The synthesis results of proposed scaler are comparable with Change scaler in terms of dynamic range. In other words, since the dynamic range of the selected moduli set is two times the dynamic range of the moduli set in the Chang scaler, we can make a fair comparison by choosing the n in proposed scaler half the n in Chang scaler. For example suppose n in Chang scaler is 8, then with dynamic range $3n$, the dynamic range would be 24 bits. In the proposed scaler the dynamic range is $6n$, so we should consider $n = 4$, therefore comparison of two scalers are performed for the same dynamic ranges.

In addition, since no scaler is designed for four moduli set and the fact that our proposed scaler is the first design for four moduli set, we are obliged to survey another comparison that is related to implementing the proposed scaler.

The proposed scaler is synthesised with and without the proposed modulo-2^n+1 adder. Since the proposed modulo-2^n+1 adder is only used in designing y_3 and y_4 (Scale

3 and Scale 4, respectively), results for these two scales are presented i.e. synthesis results of y_1 and y_2 did not change.

5.5.7 Comparison of Design Compiler Synthesis Results for the Proposed Four-Moduli Scaler and the Chang Scaler

Tables 5.17 to 5.19 are the synthesis results of the four-moduli scaler for each modular channel. Table 5.20 presents the synthesis results of the whole scaler.

TABLE 5.17: Full adder based (Chang) scaler Scale 1 ($2^n - 1$) synthesis results

n	Total Area (μm^2)	Critical Path Delay(ns)	Dynamic Power(mW)
8	141.5671701	7.97	0.6479695
16	382.3178201	7.92	1.3442
32	1223.3960751	8.05	2.8404
64	2088.9032141	8.05	5.4510
128	4328.8594161	8.05	11.1017

TABLE 5.18: Full adder based (Chang) Scaler Scale 2 (2^n) synthesis results

n	Total Area (μm^2)	Critical Path Delay(ns)	Dynamic Power(mW)
8	419.3385001	8.03	0.8705410
16	938.3209391	8.05	1.7581
32	2588.0806131	8.03	1.3466
64	4796.9495391	8.05	7.7639
128	8256.000000	8.05	15.9356

TABLE 5.19: Full adder based (Chang) scaler Scale 3 ($2^n + 1$) synthesis results

n	Total Area (μm^2)	Critical Path Delay(ns)	Dynamic Power(mW)
8	529.2107291	8.03	0.8845161
16	1517.0663771	8.04	1.8815
32	3087.6980271	8.05	3.6661
64	6450.7495011	8.53	7.5083
128	12223.3692681	9.72	14.4292

Table 5.20: Full adder based (Chang) scaler all components synthesis results $(2^n - 1, 2^n, 2^n + 1)$

n	Total Area (μm^2)	Critical Path Delay(ns)	Dynamic Power(mW)
8	2618.9877081	8.05	5.4435
16	5392.0567041	8.01	10.5801
32	7149.3175761	8.05	10.6893
64	13895.1950201	8.73	21.2366
128	26587.5102861	9.90	42.1940

The proposed scaler is designed using proposed modulo-$2^n + 1$ adder. Tables 5.21 to 5.24 are the synthesis results of the four-moduli scaler for each modular channel. Table 5.25 presents the synthesis results of the whole scaler.

Table 5.21: Four-moduli Scale 1 $(2^n - 1)$ synthesis results

n	Total Area (μm^2)	Critical Path Delay (ns)	Dynamic Power (mW)
4	182.386340	4.92	0.5258
8	362.885000	7.97	1.0638
16	814.599744	9.34	2.1900
32	2175.369701	7.77	4.5914
64	4148.985167	9.06	8.9315
128	7939.613555	9.34	17.7401

Table 5.22: Four-moduli Scale 2 (2^{2n}) synthesis results

n	Total Area (μm^2)	Critical Path Delay (ns)	Dynamic Power (mW)
4	1055.451209	9.29	1.6939
8	2220.734298	9.35	3.3027
16	4800.863481	9.36	6.7997
32	9713.336072	9.35	13.6283
64	18970.837927	9.32	26.8167
128	39299.516447	9.34	54.6831

TABLE 5.23: Four-moduli Scale 3 ($2^n + 1$) synthesis results

n	Total Area (μm^2)	Critical Path Delay (ns)	Dynamic Power (mW)
4	256.670060	7.38	0.5924
8	498.080921	9.35	1.1542
16	1341.058380	9.19	2.3970
32	3058.576897	9.34	4.9337
64	6261.598690	9.34	9.8388
128	13115.207755	9.32	19.9361

TABLE 5.24: Four-moduli Scale 4 ($2^{2n} + 1$) synthesis results

n	Total Area (μm^2)	Critical Path Delay (ns)	Dynamic Power (mW)
4	422.655319	9.18	0.8866
8	1218.871812	9.12	1.8186
16	2848.215605	9.34	3.6305
32	5862.153464	9.34	7.2182
64	12170.822912	9.41	14.5167
128	25819.124727	9.33	29.2811

TABLE 5.25: Four-moduli scaler all components ($2^n - 1, 2^n + 1, 2^{2n}, 2^{2n} + 1$) synthesis results

n	Total Area (μm^2)	Critical Path Delay (ns)	Dynamic Power (mW)
4	1718.873714	9.41	3.2619
8	3941.477415	9.42	6.5072
16	9210.649533	9.34	13.4007
32	14365.195717	9.34	18.0601
64	39372.413472	9.34	54.0209
128	79594.399595	9.34	108.1393

The proposed scaler is also designed without proposed modulo-$2^n + 1$ adder. Synthesis results of the proposed scaler for Scale 3, Scale 4 and overall system performance are presented in Tables 5.26 to 5.28.

Table 5.26: Four-moduli Scale 3 $(2^n + 1)$ synthesis results

n	Total Area (μm^2)	Critical Path Delay (ns)	Dynamic Power (mW)
4	322.951070	9.34	0.6727
8	896.565629	9.34	1.3893
16	2525.772698	9.34	3.0362
32	5260.317860	9.34	6.2973
64	11465.669641	9.56	13.3694
128	21108.661270	11.51	25.1989

Table 5.27: Four-moduli Scale 4 $(2^{2n} + 1)$ synthesis results

n	Total Area (μm^2)	Critical Path Delay (ns)	Dynamic Power (mW)
4	880.317850	9.33	1.0558
8	2426.100002	9.33	2.2047
16	5041.457724	9.35	4.4953
32	10729.681824	9.55	9.3871
64	19768.513598	11.31	17.8143
128	38369.412324	13.42	34.9114

Table 5.28: Four-moduli scaler all components $(2^n - 1, 2^n + 1, 2^{2n}, 2^{2n} + 1)$ synthesis results

n	Total Area (μm^2)	Critical Path Delay (ns)	Dynamic Power (mW)
4	2193.937235	9.35	3.4981
8	5602.178795	9.35	7.1300
16	12369.907155	9.33	14.8756
32	25772.426149	9.64	30.1932
64	50123.309240	11.22	59.8731
128	97675.514330	13.46	118.3128

5.5.8 Design Summary

In recent years, RNS became very popular in extended use of special-purpose processors, and increase in hardware capability of complex operations. Optimising performance of complex operations plays a significant role in the RNS-based system performance. In this paper, a new efficient and low-cost scaler for a $6n$ bits dynamic range for four moduli set $(2^n - 1, 2^n + 1, 2^{2n}, 2^{2n} + 1)$ is presented. Proposed scaler is designed based on the CRT algorithm, and is simplified to an efficient VLSI architecture.

Proposing a scaler for larger dynamic ranges will ultimately overcome restrictions of using RNS in applications with large dynamic range requirements.

5.6 Chapter Summary

Complexity of scaling and reverse-conversion overhead are the major challenges of RNS-based systems. In this chapter, we have studied various scaling schemes, and proposed a $(2^n - 1, 2^n, 2^n - 1)$ scaler based on modular reducers as a modification to Chang's FA-based scaler. For the same moduli set, a hybrid scaling scheme is proposed to avoid overflow error in iterative computations in RNS-based systems. The main objective was to simplify the implementation of current designs for greyscale image and video processors. Furthermore, scaling for the four-moduli set $(2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1)$ has been proposed for the first time.

Publications pertaining to this chapter:

- Azadeh Safari and Yinan Kong. *Simple, Fast and Synchronous Hybrid Scaling Scheme for the 8-bit Moduli Set*, Journal of Emerging Trends in Computing and Information Sciences, Vol. 3, No. 6, June 2012.
- Azadeh Safari, James Nugent, and Yinan Kong. *Novel Implementation of Full Adder Based Scaling in Residue Number Systems*, IEEE 56^{th} International Mid-west Symposium on Circuits and Systems (MWSCAS2013), The Ohio Union at the Ohio State University, Columbus Ohio, August 4-7, 2013.
- Fatemeh Ghasemi, Azadeh Safari, Saied Sorouri, Amir Sabbagh Molahosseini, Yinan Kong, *An Efficient RNS Scaler for the Four-Moduli Set* $\{2^n - 1, 2^{2n}, 2^n + 1, 2^{2n} + 1\}$, Submitted to Very Large Scale Integration (VLSI) Systems, IEEE Transactions on.

6 Logic Design and FPGA Implementation of RNS-Based DWT Digital Image Processor

In Chapter 2, the DWT image-compression algorithm and convolution-based filtering have been selected based on simulation results, for the proposed digital image processor. The biorthogonal wavelet family CDF97 was chosen in Chapter 3, because it is very well-known for its compatibility of symmetry and exact reconstruction. Furthermore, its symmetrical features and four vanishing moments make it suitable to distinguish a smooth signal in gray scale images. The RNS has been studied in Chapter 4. It was concluded that, for a dynamic range of no longer than 24 bits, the low-cost moduli set $(2^n - 1, 2^n, 2^n + 1)$ results in the least delay. Subsequently, 25-bit operational blocks was selected based on calculations for the proposed image processor.

In this chapter, logic design and FPGA implementations of the proposed RNS-based DWT digital image processor is provided in detail. The proposed image processor is examined against an binary image processor which is used as a set point to measure the proposed processor's improvements and costs.

The dynamic range of the selected moduli set for the proposed RNS-based image processor is 24 bits, hence the binary processor uses 24-bit computational blocks to be comparable with the proposed design. The proposed residue arithmetic units are explained in details to illustrate the novelty of the proposed design.

6.1 Hierarchy of the Proposed Image Processor

Initially, an image processor is designed which uses binary arithmetics. Figure 6.1 shows the hierarchy of the binary processor. The logic modules of the initial binary processor are optimised to the RNS-based image processor for the goals of this thesis: increased speed of operations, reduced hardware complexity, and decreased power consumption. Figure 6.2 shows the hierarchy of the proposed RNS-based image processor. A modular library is developed for modular calculations in the filter banks.

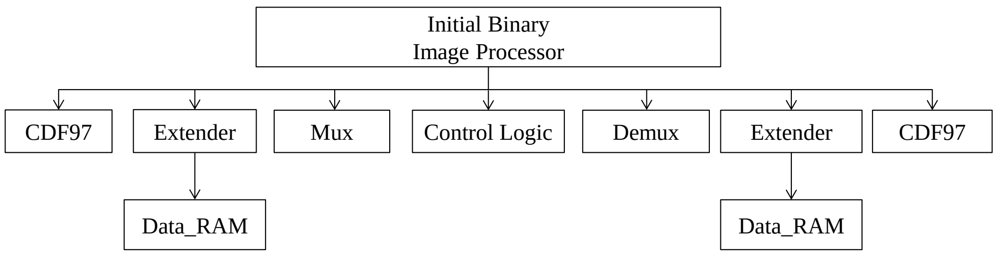


FIGURE 6.1: Hierarchy of initial binary image processor

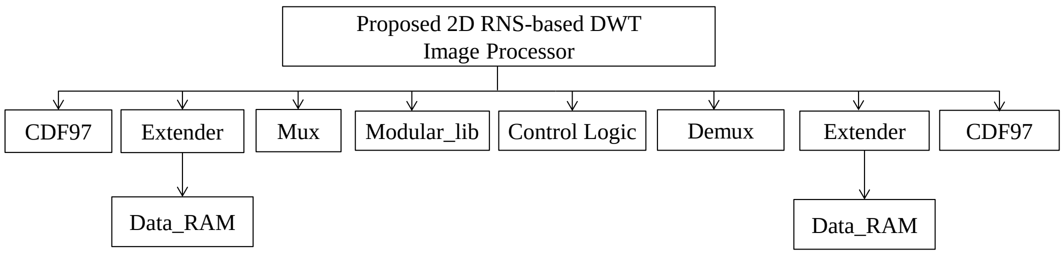


FIGURE 6.2: Hierarchy of proposed RNS-based image processor

6.2 Logic Design of the Proposed Image Processor Using Xilinx Tools

The logic modules of initial and the proposed image processor are designed using Xilinx ISE project navigator. Figure 6.3 shows the schematic of the proposed processor, consisting of the RNS-based filter banks (CDF97) for row-wise and column-wise processing, the transposition units with embedded symmetric extension (Extender), and the main control unit.

Since binary conversion for the low-cost moduli set is resolved in the literature [88], it is assumed that the proposed processor is already in the RNS domain, and that binary conversion overheads are eliminated. In the following, details of the modules of proposed processor are explained.

6.2.1 The RNS-Based Filter Banks

Two-dimensional image processing is performed using two separable one-dimensional filters because using one-dimensional filters reduces computational complexity. A two-dimensional filter is separable [140] if the filter kernel can be expressed as the product of two vectors. Filtering a $M \times N$ image with a $P \times Q$ filter kernel requires $(MNPQ)$ multiplications and additions. With a separable filter kernel, 2D operations can be simplified as a one-dimensional filtering in the horizontal direction and a one-dimensional filtering in the vertical direction. Horizontal filtering requires (MNP) multiplications and additions, and vertical filtering requires (MNQ) multiplications and additions, which is a total of $MN(P+Q)$. Hence, the separation of operations saves on the number of clock cycles required to generate 2D filter coefficients.

6.2.1.1 Existing Design of RNS-Based Filter Banks

Existing RNS-based filter banks are designed using LUTs and modular adder trees. Authors in [3] used LUTs with an 8-bit width and 256 entries for modular multiplications. Figure 6.4 shows a filter bank with a modular adder tree.

In designing the filter banks, floating-point coefficients require large hardware resources to retain the precision and to implement the multipliers. The authors in [63] have multiplied each coefficient by decimal number 256, and scaled by 256 after row-wise filtering in the scaling module. The CDF97 filter coefficients in Table 3.2 are

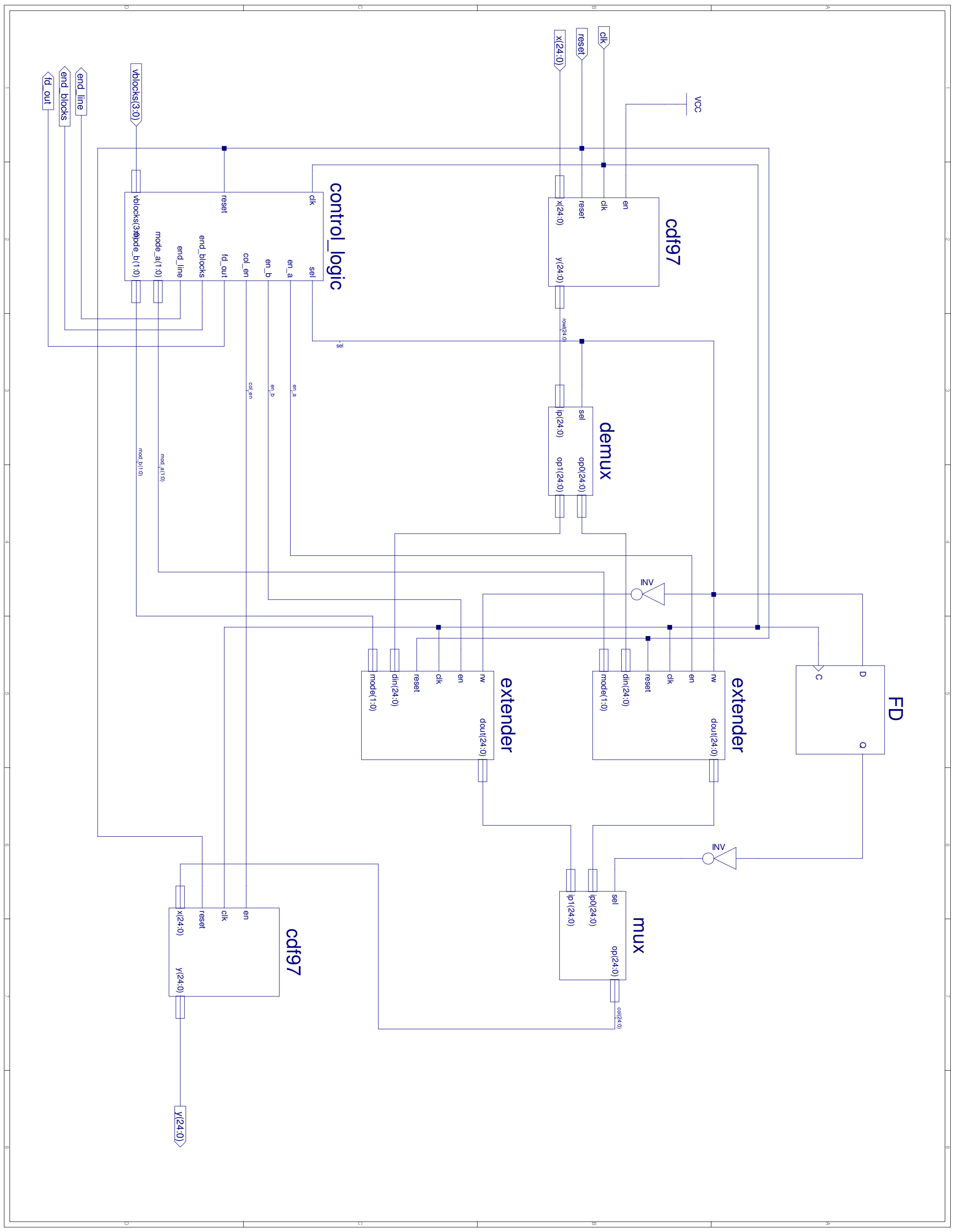


Figure 6.3: Schematic of the proposed 2D RNS-based DWT digital image processor

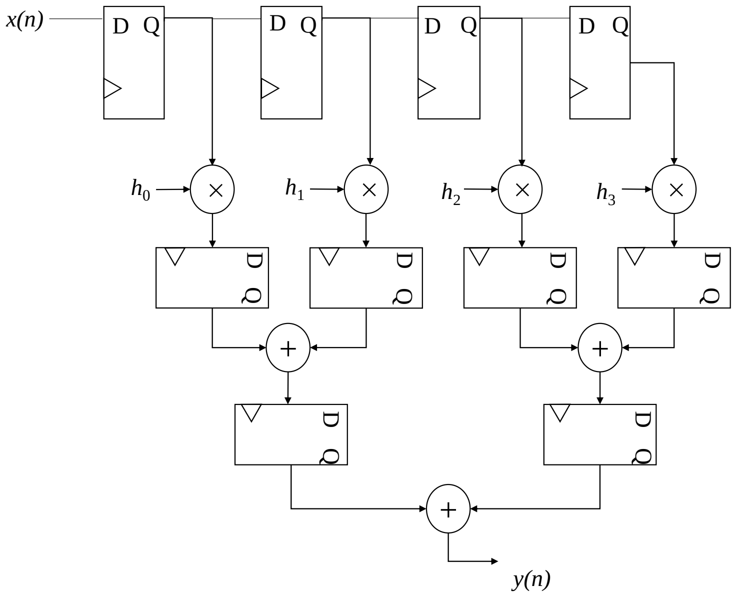


Figure 6.4: A filter bank with a modular adder tree

Table 6.1: CDF97 filter coefficients multiplied by decimal number 256

k	Lowpass Filter (h_k)	Highpass Filter (g_k)
0	154	285
±1	68	-151
±2	-20	-14
±3	4	23
±4	6	

multiplied by 256, and the new filter coefficients are shown in Table 6.1.

Existing modular filter banks use two modular adder trees (shown in Figure 6.4) to generate the highpass and lowpass sequences. The lowpass modular adder tree is composed of modular adders, while the highpass modular adder tree is composed of modular adders and a modular subtractor. The subtraction operation is implemented by augmenting the inputs of the adders with logic, controlled by a special signal, to conditionally negate the second operand, based on a control signal activated. The modular adders are clocked at half the frequency of the main clock.

The image pixels pass through delay registers at each clock cycle, and go through “selecting multiplexers”. The multiplexers split the image pixels based on the odd and even clock cycles. It selects the highpass filter at odd clock cycles, and the lowpass filter at even clock cycles. Two out-of-phase clocks are used for the modular adders. The

TABLE 6.2: Dyadic fractions of CDF97 filter coefficients

k	Lowpass Filter (h_k)	Highpass Filter (g_k)
0	46/64	1
±1	16/64	-9/16
±2	-8/64	0
±3	0	1/16
±4	1/64	

LUTs are used for modular multiplication of the image pixels and filter coefficients.

The downside of existing RNS-based filter banks is that the LUTs are the main sources of power leakage, which is the major concern of standalone applications like mobile phones and cameras. In addition, using LUT increases the hardware complexity of RNS-based designs.

6.2.1.2 Proposed RNS-Based Filter Banks

To optimise the existing modular filter banks, multiplierless RNS-based filter banks are proposed, where the floating-point filter coefficients are dyadic fractions (numbers of the form $k/2^n, k, n \in Z$) [141]. The dyadic fractions of the CDF97 filter coefficients are shown in Table 6.2.

Using the dyadic fractions shown in Table 6.2, the lowpass and highpass filter coefficients are calculated using (6.1) and (6.2). The VLSI architecture of a multiplierless filter bank is shown in Figure 6.5.

$$LPF = (46w_0 + 16w_1 - 8w_2 + 0w_3 + w_4)/64 \tag{6.1}$$

$$HPF = (16w_0 - 9w_1 + w_3)/16 \tag{6.2}$$

It has nine shift registers at the input to load the input data ($x_{i\pm n}$ for $n = 0, 1, ..., 4$). After loading all nine registers, they are added together using $w_0 - w_4$ adders in the specific order of $w_0 = x_i$, and for $n = 1, 2, 3, 4$:

$$w_n = (x_{i-n}) + (x_{i+n}) \tag{6.3}$$

These adders are shared between the lowpass and highpass filters (LPF and HPF),

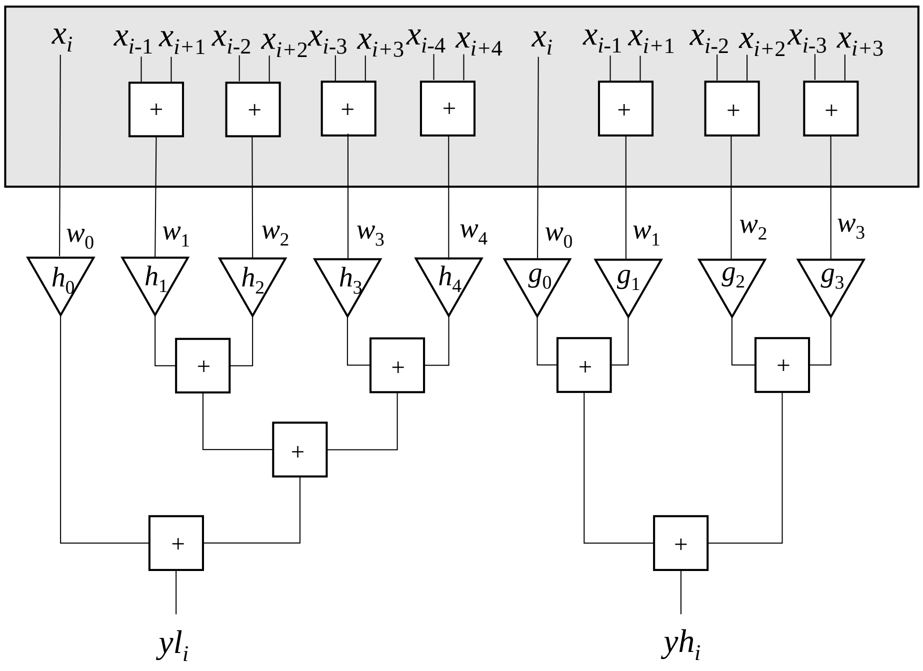


Figure 6.5: Multiplierless implementation of DWT filter banks [2]

which saves on hardware complexity. The results from the register adders are then shifted by 2, 4, 8 or 16 bits.

Multiplication by 2^n is a simple shift, and this covers all the coefficients in Table 6.2 except 46 and 9. The solution is that total 46 is composed of $(1 + 2 + 4 + 16) \times 2$, and total 9 is composed of $(8 + 1)$, shown in Figure 6.6, where the generating steps of the LPF and HPF are provided.

Each operating block of the RNS-based processor is packed in a 25-bit binary-coded number format. The first 9 bits (24 downto 16) are modulo-257 diminished-one data with the MSB as the indication flag, followed by 8 bits (15 downto 8) of modulo-256 data and 8 bits (7 downto 0) of modulo-255 data. Figure 6.7 shows the block diagram of the RNS-based filter banks in [3], and the proposed RNS-based filter banks in this thesis.

To implement the RNS-based filter banks, modular multipliers and adders are used instead of the binary blocks. In the following, details of the modular multipliers and adders for the moduli set (255, 256, 257) are described.

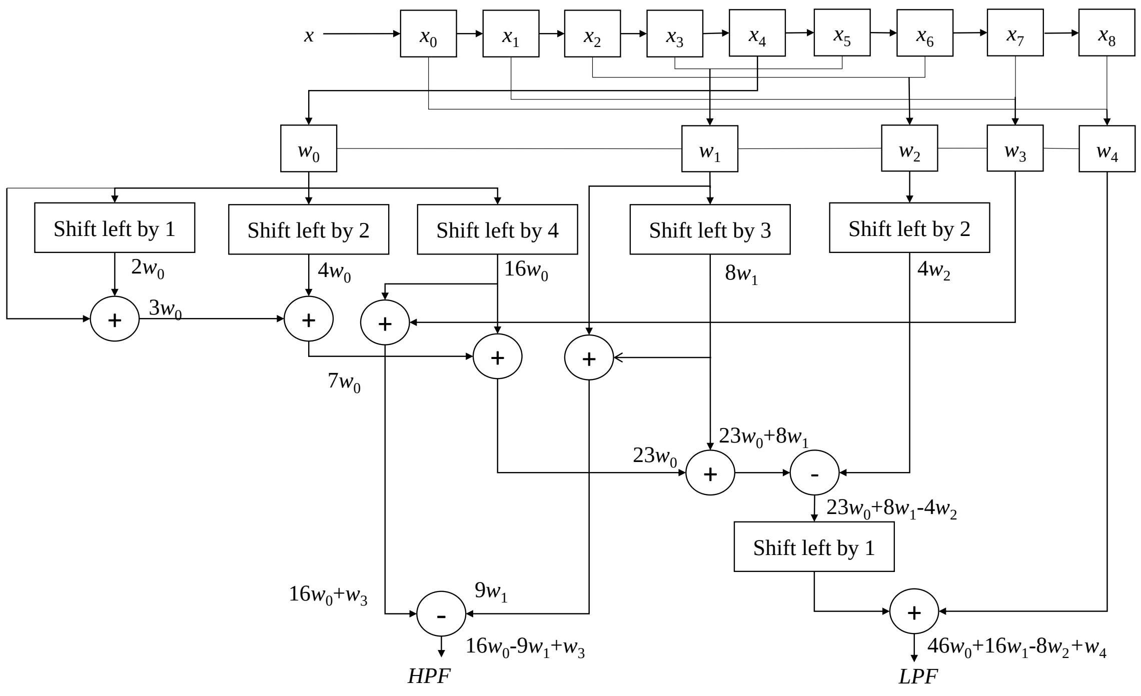


Figure 6.6: Generating lowpass and highpass filter coefficients using shifts and additions

Modulo-m_i multiplier Designing the modulo-256 multiplier is straightforward and can be performed by selecting the least-significant eight bits of the binary data. However, to implement modulo-255 and -256 multipliers, a simple modification of the *product partitioning scheme* is selected [4]. In this scheme, a modulo-m reduction of a normal unsigned multiplication is considered. The mathematical representation of the product partitioning idea is presented in (6.4). Figure 6.8 shows the partitioned-operand modulo-m multiplier [4].

$$\begin{aligned} |AB|_m &= |2^n U + L|_m \\ &= ||2^n U|_m + |L|_m|_m \\ &= |cU + L|_m \end{aligned} \tag{6.4}$$

since $|2^n|_m = c$ and $L < m$.

Optimised modulo-m_i multiplier The modulo-m_i multipliers are optimised by using filter coefficients of the form 2^n and the low-cost moduli set. Hence, in the first multiplier in Figure 6.8, one of the operands is always of the form 2^n. Therefore, the first multiplier is replaced with a "shift left by n" in the optimised modulo-m

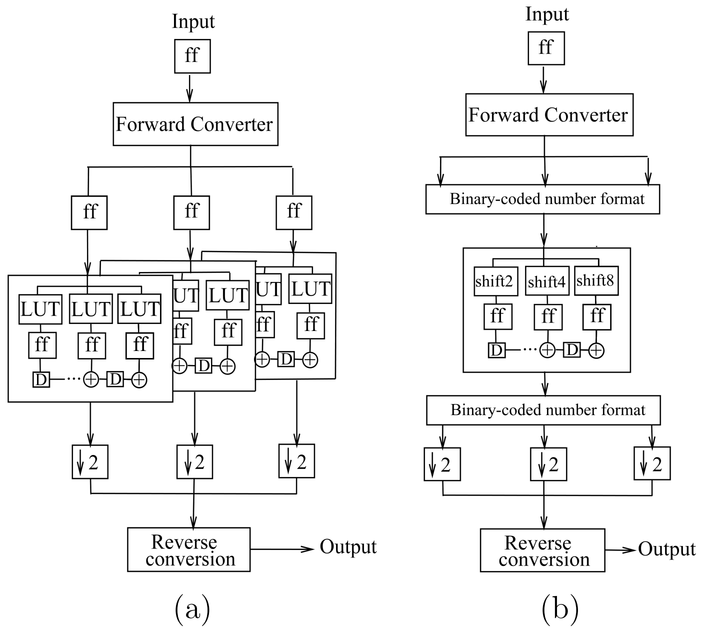


Figure 6.7: Block diagram of (a) RNS-based filter banks based on the design in [3], (b) Proposed RNS-based filter banks

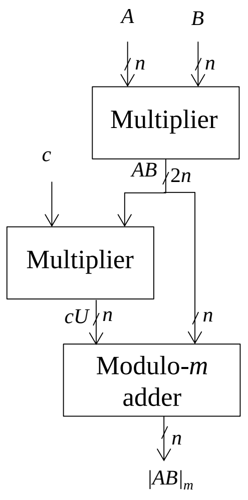


Figure 6.8: Partitioned-operand modulo-m multiplier [4]

multiplier. In Figure 6.8, the middle multiplier is used to find cU. For modulo-255 and -257 multiplication, c becomes ± 1 and U is a few bits of MSB of the input. Therefore, the middle multiplier is not required and can be replaced with bit rewiring which impose no delay to the design. In other words, only a modular adder/subtractor is needed to implement the multiplication. Modulo-257 multiplication requires additional care for the cases when the result is 256 (9 bits). In this thesis, a diminished-one representation of numbers is used for modulo-257 arithmetic. In the diminished-one number system, each number (X) is represented as $\overset{*}{X} = X - 1$, except for zero. $\overset{*}{X}$ is an n-bit vector (the number part). Calculations including zero are done separately. X is represented as $az\overset{*}{X}$, where az is a single bit, i.e. an additional bit (MSB) is used for zero indication,where $MSB = 0$ means the data is not zero, and $MSB = 1$ indicates the data is zero.

Figure 6.9 shows the optimised modulo-m multiplier.

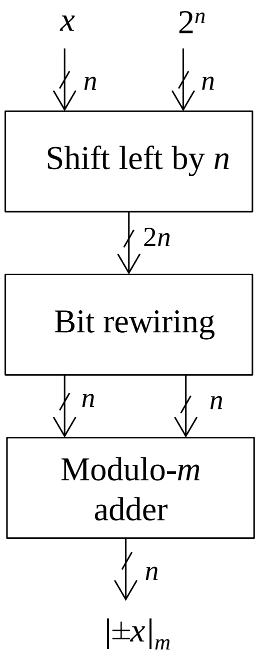


FIGURE 6.9: Optimised modulo-m multiplier

Modulo-m_i adder The modular adders are required to add the results from modular multiplication and generate the filter coefficients. They are designed based on the general function of a modulo-m_i adder:

$$|A + B| \bmod m_i = \begin{cases} A + B & if A + B < m \\ A + B - m & otherwise \end{cases} \tag{6.5}$$

In this thesis, a modulo-255 adder is implemented based on the *basic modulo adder* in [4]. Basic modulo-m adder is shown in Figure 6.10. The proposed architecture is modified to a low-cost high-performance modular adder as shown in Figure 6.11. The modified modulo-255 adder uses inputs A and B concurrently to generate $A + B$ and $A + B + 1$ and makes a decision based on the carry out. If $c_1 = c_2 = 0$, the result is $s = A + B$; otherwise the result is the eight LSBs of $s = A + B + 1$. To implement a modulo-257 adder, the *diminished-one parallel-prefix adder* is employed [142].

Example: Let $A = 236$ and $B = 117$. The result of the first adder is $S = A + B = 353$ with carry out$= 1$. According to the above algorithm, the least-significant eight bits of $s = A + B + 1$ give 98 which is consistent with the theoretical calculation of $|353|_{255} = 98$.

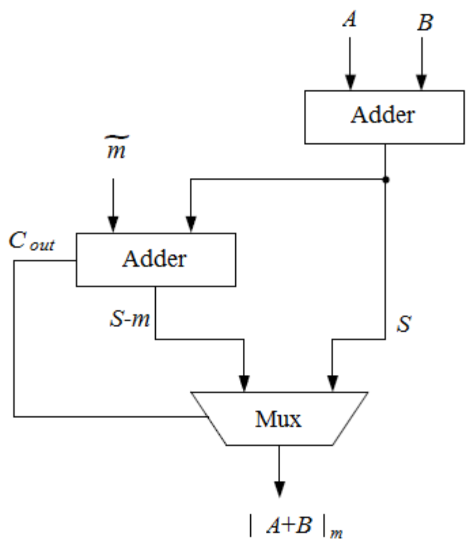


FIGURE 6.10: Basic modulo-m adder where $(\tilde{m})$ denotes 2's-complement of m [4]

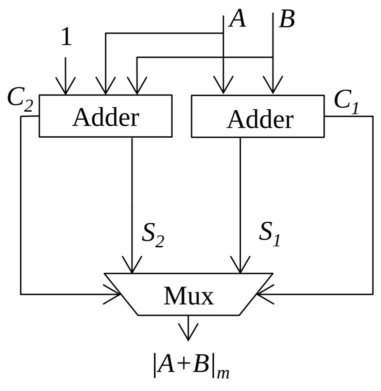


FIGURE 6.11: Proposed modulo-m adder

6.2.2 Transposition Unit

To process a two-dimensional data set like an image, a one-dimensional wavelet transform should be performed on both row and column directions. To this end, the proposed image processor uses two RNS-based filter banks, one for row-wise processing and the other for column-wise processing. The outputs from the row-wise filter have to be transposed before going for column-wise processing; hence a transposition unit is necessary between the two filter banks.

6.2.2.1 VLSI Architecture of a Scalable Matrix Transposer

Various matrix transposition designs have been reported in the literature [143–148]. Generally, the most efficient technique is the memory addressing technique, which stores data in rows and accesses them in columns. Having said that, the authors in [143] presented a modular and cascadable matrix transposer for real-time applications. They chose not to use memory addressing techniques to avoid address-sequence computation and memory-access-time overheads. In this section, scalable transposition unit is implemented, and is evaluated whether it is the best choice for the proposed image processor. Figures 6.12 and 6.13 show the transposition unit using a modular scalable transposer, and the structure of the modular scalable transposer, respectively.

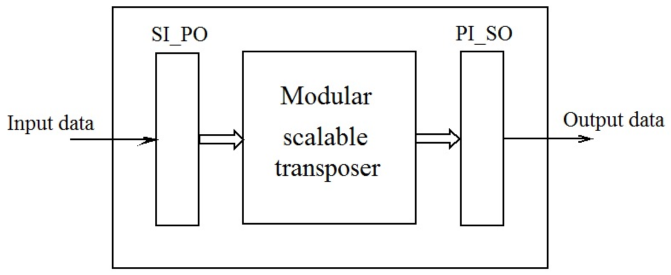


Figure 6.12: Transposition unit

Figure 6.14 shows a serial-in, serial-out transposition unit. It consists of a scalable matrix transposer [143], shift register, serial-in parallel-out (SI-PO) and parallel-in serial-out (PI-SO) modules.

The transposition unit accepts input data serially, likewise provides transposed output data serially. Hence, two SI-PO and PI-SO modules are necessary to connect

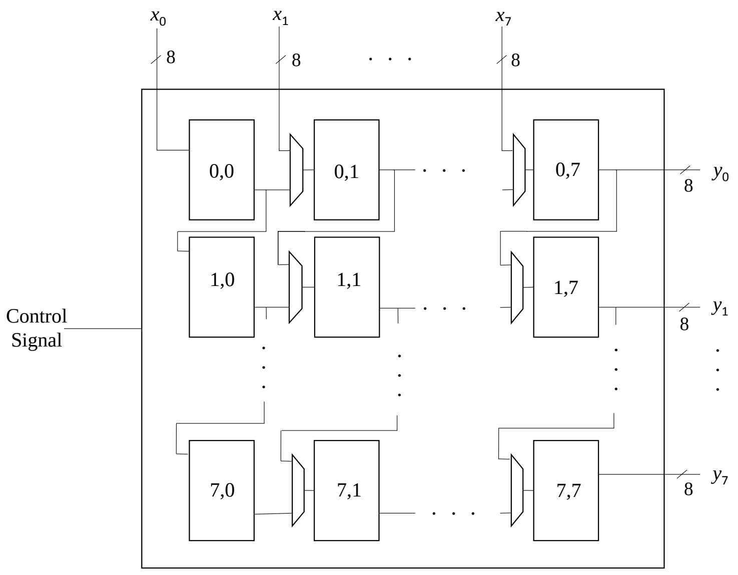


Figure 6.13: Modular scalable transposer

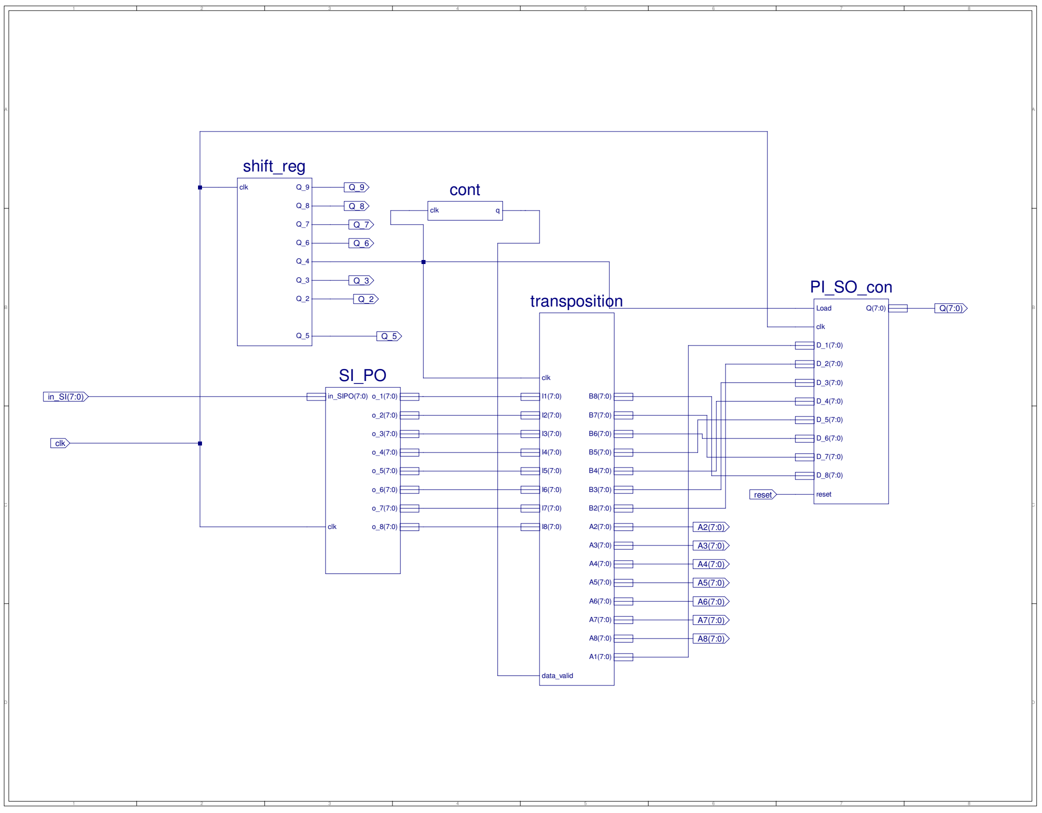


Figure 6.14: Serial-in, serial-out transposition unit

the scalable transposer with other modules of the image processor.

Figure 6.15 shows the transposer consisting of 16 COLTM4 (one column of the Transposer Module (TM) with four) elements for transposing eight image pixels. Each COLTM4 is composed of four DFM (D-type flip-flop with multiplexer) cells. In other words, the transposer consists of 64 DFM cells connected as eight rows by eight columns for transposing an 8×8 matrix of data. The VLSI architectures of the COLTM4 and DFM are shown in Figures 6.16 and 6.17, respectively.

At each clock cycle, a row of eight pixels is loaded into and out of the transposer. Since the 1D DWT is providing the results of row computation in series, there is a need for a parallel-to-series module as an interconnection module to convert the serial data to eight-bit parallel data. It accepts eight image pixels at eight consecutive clock cycles and provides the transposed output at the 9^{th} clock cycle. This module generates outputs at each clock cycle, however only the data that appear every eight clock cycles are valid. For this purpose handshaking signals (*data_valid*) and an eight-bit ring counter are employed.

Two control signals determine the direction of row and column data movements. When $Mode\_A$ is active, register (i, j) is copied from register $(i, j - 1)$, hence data move horizontally from left to right. When $Mode\_B$ is active, register (i, j) is copied from register $(i - 1, j)$ and data move upward.

To generate the $Mode\_A$ and $Mode\_B$ control signals, a counter is used which counts the number of the *Data_valid* signals. Initially the transposer is on $Mode\_A$ to accept the data, and when the counter reaches eight it will switch to $Mode\_B$.

Data_valid indicates that the serial-to-parallel module has valid data on that clock cycle. It is high only when data is valid, and during the rest of the clock cycles *Data_valid* is low. Every eight clock cycles, a set of valid data is available from the serial-to-parallel module. Thus, a “valid” signal is sent to the modular scalable transposer and valid data is loaded to the module. The procedure continues and the module is loaded with eight rows of valid data while shifting them horizontally in $Mode\_A$. As soon as the last row of valid data is loaded into the modular scalable transposer, it switches to $Mode\_B$. In this mode, the data are shifted vertically and the transposed rows are available through the outputs.

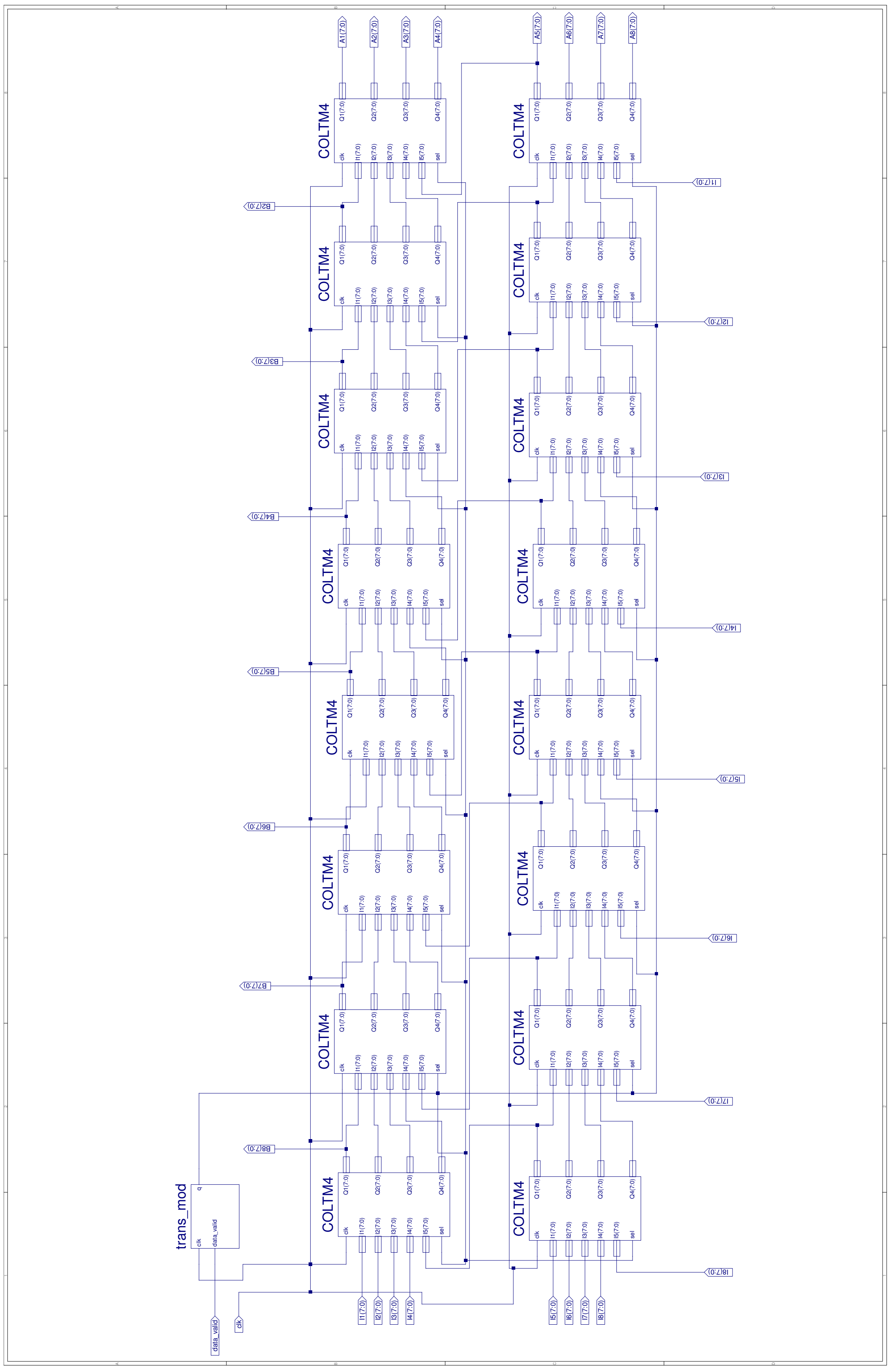


Figure 6.15: VLSI architecture of the “transposition” in Figure 6.14

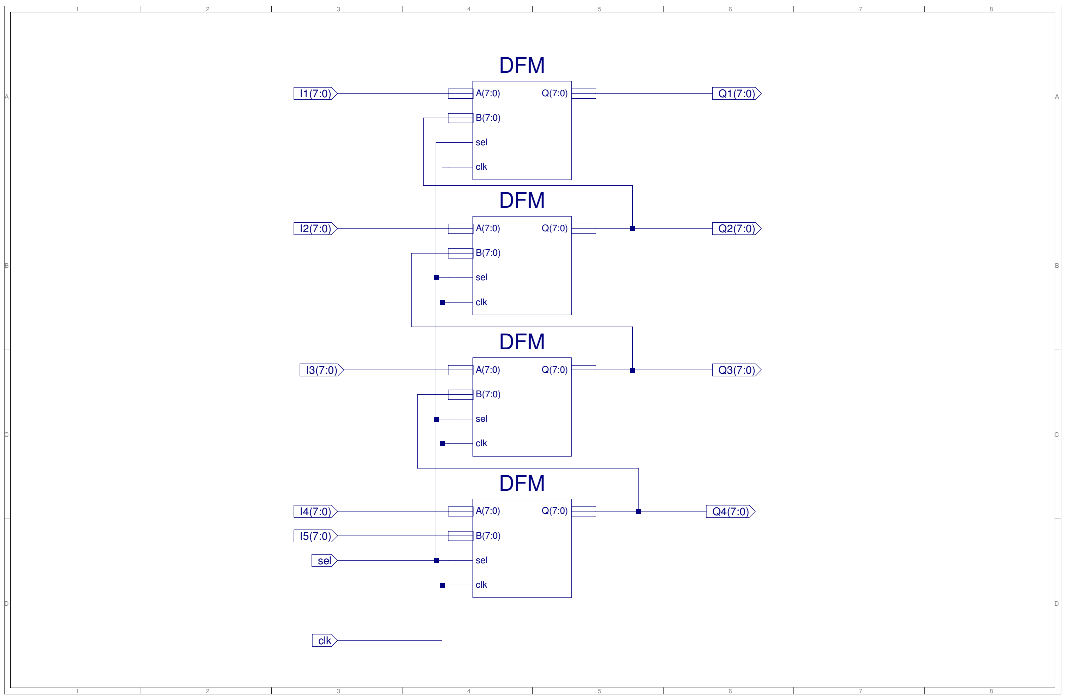


Figure 6.16: VLSI architecture of COLTM4 in Figure 6.15

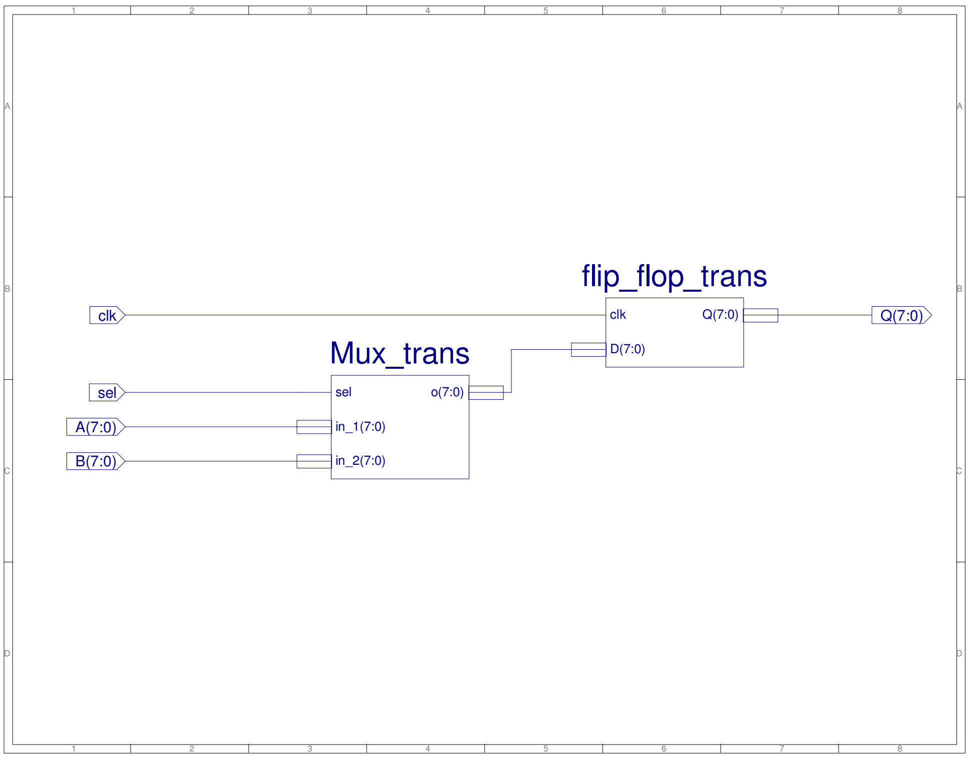


Figure 6.17: VLSI architecture of DFM in Figure 6.16

6.2.2.2 Proposed Transposition Unit with Overlapped Sub-Blocks

The main drawback of the modular scalable transposer is its hardware cost, which is incompatible with area-efficient design. Hence, a memory-based transposition technique is developed. Using this technique to process an image, however, requires huge memories, proportional to the image size. These huge memories generally require external memory access, which fails to compress a high-quality image. One common solution for this problem is to divide the entire image into sub-blocks and perform processing on sub-blocks [149]. This method leads to block artifacts, which is undesirable in many image-processing applications.

The proposed image processor in Figure 6.3 uses two Extenders (transposers embedded with symmetric extension) to transpose lowpass and highpass filter coefficients. They are serial-in serial-out modules, and provide the transposed and overlapped data at the output only after completing write operations. Hence, two such units are required for continuous operation. While one transposition unit is accepting outputs of row processing (writing mode) the other one is reading data and sending them for column-wise filtering. After completing write/read cycles, they switch their modes. While reading the next sub-block and saving it to the other transposition unit, the previous transposition unit feeds transposed data to the column filter, and 2D coefficients are generated in the output of the column CDF97 filter.

Figure 6.18 shows the reading pattern of the transposition units. In this figure, the image is processed by the overlapped sub-blocks to provide intermediate results. This approach saves on the large memory requirements of image processing. In this scheme, sub-blocks of size $B \times B$ are selected and the overlapping data is $(L - 1)$ filter taps, where (L) is the filter length and should be odd and greater than 1. In this thesis design, the sub-block size is $B = 64$ and the filter length is $L = 9$. For further explanation of how the overlapped scheme works, consider the row processing of the image. In row-wise processing, $(L - 1)$ coefficients are overlapped in each row from the previous block. The filter bank loads the $(L - 1)$ outputs of the previous row and discards the first $(L - 1)$ coefficients of the output. Hence, the results are the continuation of the previous sub-block. Another advantage of this approach is that discarding $(L - 1)$ bits of data from each row of a $(B \times B)$ sub-block requires only $B \times (B + L - 1)$ clock cycles and saves on computations. The transposition units are also responsible for symmetric extension. While writing $(B \times B)$ sub-block data into the transposition unit, it preserves the last $B \times (L - 1)$ written data, which serve as the necessary overlapped data to the column-filter inputs. To avoid coefficient

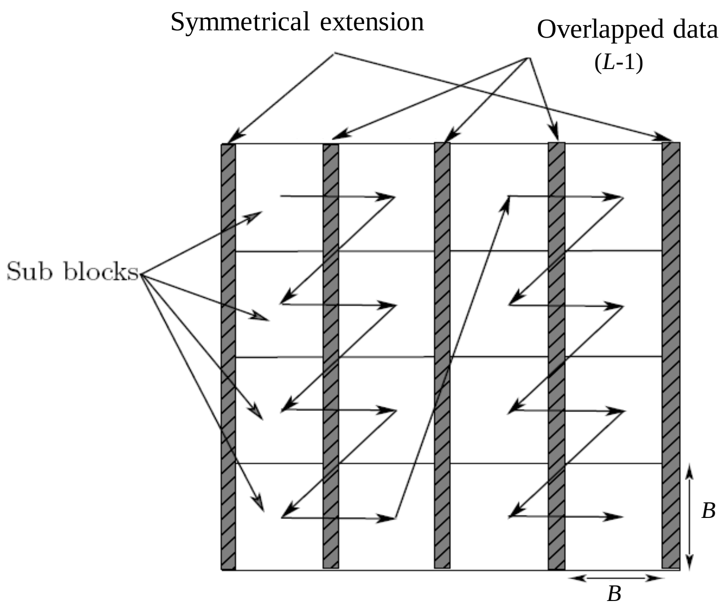


Figure 6.18: Reading pattern of image data in transposition units

expansion, the transposition unit is embedded with a symmetric extension $(1,1)$ of the row-processing outputs. Different operating modes, i.e. with or without symmetrical extension, are selected via *mode* inputs.

6.2.2.3 Modified Transposition Unit with Symmetric Extension (Extender)

Coefficient expansion is a problem in DWT operation as it uses linear convolution. One solution is to use circular convolution by periodic extension (wraparound) of inputs. But this generally introduces a jump, and hence boundary artifacts. Circular convolution of the inputs results in large wavelet coefficients in the highpass filter at the location of a jump or discontinuity. This large number of wavelet coefficients in the highpass filter is undesirable, because it requires a large number of bits to store the image accurately. Symmetric extension does not introduce a jump or boundary artifacts, and eliminates the large wavelet coefficients caused by the border discontinuities. The CDF97 wavelet used in this design involves symmetric bi-orthogonal filters (or linear-phase filters), hence the periodic symmetric inputs give a periodic symmetric output [27, 67, 150] and efficient symmetric extensions are the best choice. Table 6.3 shows various synthesis extensions for a signal x of length N_0. The symmetric extension

$E_s^{(1,1)}$ shown in Figure 6.19 is used, and to prevent the coefficient expansion redundant samples have been removed from the output.

Table 6.3: Symmetric extensions of signal x with filter length of N_0

Signal x	Filter	Sub-band		Shift	Synthesis
Length N_0	h Center	Center	Dimension		extension
Even	$2v$	v	$N_0/2$	v	$E_s^{(1,2)}$
	$2v+1$	$v+1/2$	$N_0/2$	$v+1$	$E_s^{(2,1)}$
Odd	$2v$	v	$(N_0+1)/2$	v	$E_s^{(1,1)}$
	$2v+1$	$v+1/2$	$(N_0+1)/2$	$v+1$	$E_s^{(2,2)}$

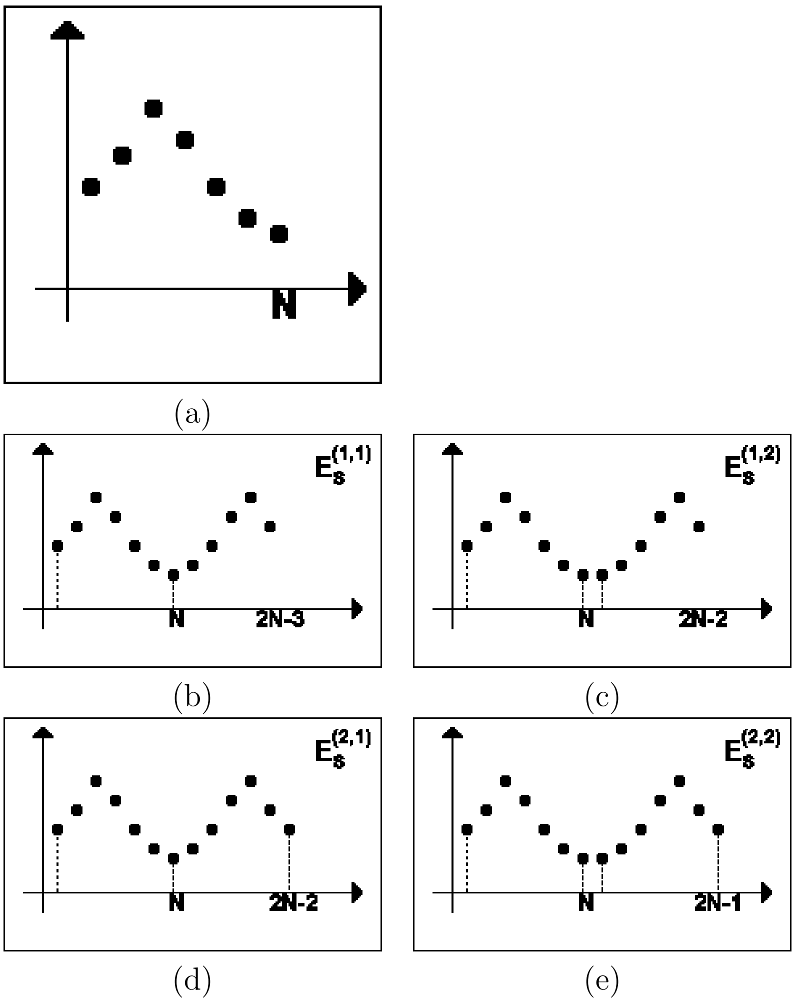


Figure 6.19: Symmetric extension: (a) Original data of length N, (b) Extension(1,1), (c) Extension(1,2), (d) Extension(2,1), (e) Extension(2,2)

The Extenders read the first block of data with symmetrical extension. The row coefficients are saved in one transposition unit, and moved to the right-hand block. In each sub-block of the image, the data is read row-wise. The transposing and overlapping are done with the help of s simple address-generation unit in the transposition unit. In this thesis, the transposition unit with an embedded symmetric extension is called an "Extender".

6.2.3 Control Logic

The control logic is the main controller of the design. It controls the processor operations using input ports *clk* and *reset*, and the number of vertical blocks (*vblocks*) of input image. The number of vertical blocks is determined by dividing the number of rows of the input image by the number of rows of the sub-blocks. In this thesis, the size of the sub-blocks is 64×64. Therefore, the largest image size can be 1024×1024 with 16 vertical blocks, as previously shown in Figure 6.18. The functions of the control unit are to select the output and input of the row (*sel*) and column (*col_en*) RNS-based filter banks, and enable them using (*en_a* and *en_b*). It also controls the operating mode and the read/write mode of each transposition unit (*mode_a* and *mode_b*).

The *mode_a* and *mode_b* signals are responsible for controlling the symmetric extension of the first and second transposition units, respectively. $Mode = 01$ represents the initial symmetric extension where, the writing address starts at zero. In this mode, there is no need for the previous output data, and overwriting is allowed. Hence, data reading can be started with a specific address. This writing address can be determined by checking for column-counter overflow when changing from writing to reading mode. $Mode = 10$ represents the symmetric extension at the end, and the address is decremented after the counter crosses its limit. Hence, to calculate further points, $(L-1)/2$ clock cycles should be provided. The new address can be found by adding $(RAM\_size - Image\_width)$ to the current address.

For the initial symmetric extension, only $(L-1)/2$ samples are extended and the number of clock cycles required is the same as the number of clock cycles required for sub-block processing with no extension. However, in the case of the symmetric extension at the end, $(L-1)$ samples are extended. The remaining samples are redundant, and should be discarded. It also requires $(L-1)/2$ more clock cycles than the initial extension mode. Hence, the controller must keep all other modules disabled while providing additional clock cycles to the transposition units.

The control unit also provides other control signals such as end of line (*end_line*), end of blocks (end_blocks), and enable data outputs (fd_out) to properly arrange the output data. The control logic uses a main counter to enable the first transposition unit and a counter that counts up to $(RAM\_size - 1)$ state for sub-block image processing. When the sub-block counter is equal to $(RAM\_size - 1)$, it resets to zero for the next sub-block image. The size of the required RAM is calculated using

$$\begin{aligned} RAM\_size &= Sub\_image\_size + (L-1) * Image\_width \\ &= 64 \times 64 + 8 \times 64 = 4608 \end{aligned}$$

where L is the number of filter taps (should be even), $Image\_width$ is the sub-block width (or height) and Sub_image_size is the product of the sub-image width and height.

6.3 Simulation and Functionality of the Proposed Processor

The proposed image processor has been designed using Xilinx ISE project navigator. To check whether the proposed image processor has accurate functionality, it is simulated using the ModelSim simulator. Figure 6.20 shows the generated waveforms of the proposed image processor.

The image data has been saved as a text file (*image_input.txt*) and read to the test bench. It has been processed by the proposed image processor and the results of the 2D processing have been written back to the output text file (*image_output.txt*) through the test bench. Matlab is used to reconstruct the decomposed image. Figure 6.21 shows the compressed image after one-level 2D DWT processing using the proposed image processor.

6.4 Top-Level and Hierarchy Synthesis Results

The proposed RNS-based image processor is synthesised using target device xc6vlx75t for synthesis and FPGA implementation. Tables 6.4 and 6.5 show the selected FPGA device and the parameter set of the design, respectively.

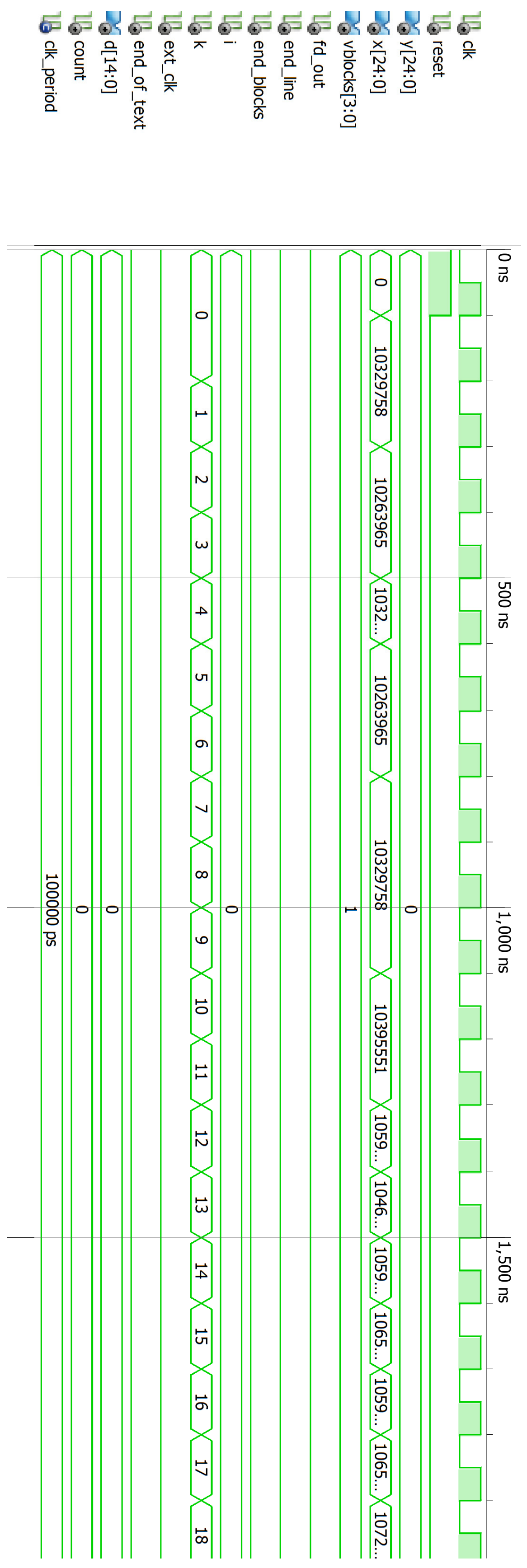


Figure 6.20: ModelSim simulation of proposed RNS-based image processor

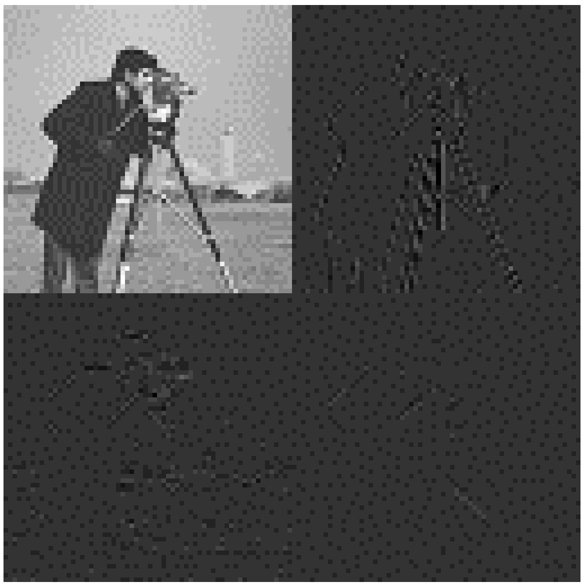


Figure 6.21: Camera-man image after one level of 2D processing using proposed image processor

Table 6.4: Target FPGA device

Family	Virtex6
Part	xc6vlx75t
Package	ff484
Temp Grade	Commercial
Process	Typical
Speed Grade	-3
Characterisation	Production,v1.3,2011-05-04

Table 6.5: Parameter set

Parameter	Variable
Number of bits	$n = 25$ bits
Image width	$IMAGE\_WIDTH = 64$ bits
RAM size	$RAM\_SIZE = 4608$ bits
Address line no of bits for RAM	$a = 13$
Filter length	$L = 9$
Symmetric extension start address	$SYM\_ADDR\_START = 512$
Extended clocks	$EXTENDED\_CLOCKS = 256$

6.4.1 Synthesising the Modular Adders

The modular adders work under a main clock signal of frequency 100 MHz. Timing details of the modular adders are provided in Table 6.6 for timing constraint clock period 10 ns and 50% duty cycle.

Table 6.6: Delay of the modular adder

MET	Constraint	Check	Worst-case slack	Delay
YES	TS_clk "clk" 10 ns HIGH 50%	MINPERIOD	8.750	1.250 ns

The synthesis results show the best case available for modular adders is 1.250 ns.The FPGA resource consumption and power consumption of the modular adders are shown in Table 6.7.

Table 6.7: FPGA resource consumption and power consumption of modular adders

Channels	Total Power (mW)	Logic Power (mW)	Signal Power (mW)	FFs	LUTs	CARRY4s
255	0.28	0.09	0.20	1	18	2
256	0.21	0.04	0.17	-	8	2
257	0.26	0.07	0.19	8	17	2

6.4.2 Synthesising the Modular Multipliers

The modular multipliers are designed to work under a main clock signal of frequency 100 MHz. Timing details for the modular multipliers are provided in Table 6.8 for timing constraint clock period 10 ns and 50% duty cycle. The FPGA resource consumption and power consumption of the modular multipliers are shown in Table 6.9.

The synthesis results show that the best case available of the modular multipliers is the same as for the modular adders. This shows that the modular multipliers are optimised, and the delay of the modular multipliers is reduced to the delay of the modular adders.

Table 6.8: Delay of the modular multiplier

MET	Constraint	Check	Worst-case slack	Delay
YES	TS_clk “clk” 10 ns HIGH 50%	MINPERIOD	8.750	1.250 ns

Table 6.9: FPGA resource consumption and power consumption of modular multipliers

Channels	Total Power	Logic Power	Signal Power	FFs	LUTs
	(mW)	(mW)	(mW)		
255	0.14	0.03	0.11	8	10
256	0.09	0.00	0.09	8	3
257	0.10	0.00	0.10	8	3

6.4.3 Synthesising the Modular Filter Banks

The timing details for the modular channels are provided in Table 6.10 for timing constraint clock period 10 ns and 50% duty cycle. It shows that timing constraints are MET in all three modular channels. Modulo-256 channel is the fastest and requires almost one third of the delay used by other two modular channels. Modulo-255 and -257 channels have similar delays. Modulo-255 channel is slighly faster, though.

Table 6.10: Delay of modular channels

Channels	n	Constraint	SETUP	HOLD	Delay
			(ns)	(ns)	(ns)
255	8	MET	0.182	0.132	9.818
256	8	MET	6.665	0.116	3.335
257	9	MET	0.110	0.143	9.89

The power and FPGA resource consumption of modular channels are shown in Table 6.11. According to the table, similar pattern of timing analysis for modular channels has accured in FPGA resource and power consumption. Modulo-256 channel requires least amount of FPGA resources and dissipates least power than other two modular channels. Modulo-255 and -257 channels require same amount of power, however, the latter channel demands more FPGA resources.

Table 6.11: FPGA resource and power consumption of the modular channels

Channels	Total Power (mW)	Logic Power (mW)	Signal Power (mW)	FFs	LUTs	CARRY4s
255	5.74	3.30	2.44	92	272	22
256	2.69	1.48	1.21	86	84	8
257	5.74	2.90	2.84	105	318	26

6.5 Initial Binary Processor

Initially, an image processor is designed which uses binary arithmetics. The initial binary image processor is designed based on the LUT based RNS filter banks, modular scalable transposers and 24-bit binary arithmetic blocks. Subsequently, the logic modules of the initial binary processor are optimised to the proposed RNS-based image processor.

6.6 Performance Comparison of the Proposed RNS-Based Image Processor with Existing Designs

In this section, the performance of the proposed image processor is compared with the initial binary processor and similar existing implementations available in the literature.

6.6.1 Comparing Synthesis Results of Modular Scalable Transposer and the Proposed Transposer (Extender)

Both the modular scalable transposer in Figure 6.14 and the proposed transposer 6.18 are synthesised. Table 6.12 shows delay of each transposer. The synthesis results show that the proposed overlapped sub-block with symetric extension transposer, which is memory based, is faster than the modular scalable transposer in [143].

Table 6.12: Delay of modular scalable and proposed transposers

Transposer	n	Constraint	SETUP (ns)	HOLD (ns)	Delay (ns)
Modular scalable	8	MET	6.057	0.001	3.943
Proposed	8	MET	6.274	0.061	3.726

Comparisons of the power and FPGA resource consumption of the modular scalable and the proposed transposer are shown in Table 6.13. It shows that the proposed transposer uses only 34% of the total power used by modular scalable transposer. It requires less amount of FPGA resources, too.

Table 6.13: FPGA resource consumption and power consumption of transposers

Transposer	Total Power (mW)	Logic Power (mW)	Signal Power (mW)	FFs	LUTs	BRAMs	CARRY4s
Modular scalable	15.38	8.06	7.32	615	611	-	8
Proposed	5.27	4.45	0.82	41	149	4	12

6.6.2 Comparison of the Initial and Proposed RNS-based CDF97 Filter Banks Designed Using Existing and Proposed Modules

The initial and proposed RNS-based filter banks are designed based on the proposed multiplierless structure of filter banks (shown in Figure 6.6), and dyadic fraction filter coefficients. The RNS-based filters, however, take advantage of optimised modular adders and multipliers, and less bit-width in modular channels. Comparison of two set of filters is in Table 6.14. It shows that the proposed RNS-based filters operate 22% faster than the existing filter banks operated by binary arithmetic.

Table 6.14: Delay of CDF97 filter banks designed by existing and proposed RNS-based modules

Filter Bank	n	Constraint	SETUP (ns)	HOLD (ns)	Delay (ns)
Initial	24	MET	0.082	0.081	9.918
Proposed RNS-based	25	MET	2.272	0.092	7.728

Comparison of the power and FPGA resource consumption of the initial and RNS-based CDF97 filter banks in Table 6.15 shows that proposed RNS-based filter banks use 43% less power and 7% less FPGA resources. This result shows that using RNS arithmetic and optimising modular adders and multipliers have saved on FPGA sources.

Table 6.15: FPGA resource consumption and power consumption of modular filter banks

Filter Bank	Total Power (mW)	Logic Power (mW)	Signal Power (mW)	FFs	LUTs	CARRY4s
Initial	13.99	7.76	6.22	285	654	56
Proposed RNS-based	7.96	5.11	2.85	316	502	104

6.6.3 Comparison of Initial and RNS-based Processors

A comparison of the timing constraints of the initial and proposed RNS-based image processors is shown in Table 6.16. The initial processor uses multiplierless filter banks, modular scalable transposers and 24-bit binary arithmetic blocks. The RNS-based processor, however, uses proposed transposers (Extenders) and optimised filter banks. The table shows that the proposed RNS-based processor requires 2.297ns less delay to generate the 2D filter coefficients. In other words, the worst case available for the proposed RNS-based processor is 23% less than for the initial processor. Based on these results, it is confirmed that using a RNS-based processor increases the speed of operations.

Table 6.16: Delay of binary and RNS-based image processors

Processor	n	Constraint	SETUP (ns)	HOLD (ns)	Delay (ns)
Initial	24	MET	0.103	0.036	9.897
Proposed RNS-Based	25	MET	2.400	0.075	7.600

From the FPGA resource consumption of the initial and proposed RNS-based image processors in Table 6.17, it can be seen that hardware complexity of the proposed RNS-based processor is similar to that of initial binary processor. Usually, it is expected to get higher hardware complexity in RNS-based designs than designs operated on binary arithmetics. This expectation is due to use of LUTs as modular multipliers, scalers and transposers. Current results show that replacing LUTs with multiplierless filter banks, optimising transposers and optimising the design to remove scalers have retained the hardware complexity of proposed RNS-based processor similar to the initial processor.

Table 6.17: FPGA resource consumption and power consumption of processors

Processor	Total Power (mW)	Logic Power (mW)	Signal Power (mW)	FFs	LUTs	BRAMs	CARRY4s
Initial	62.59	45.31	17.28	695	1760	26	142
Proposed RNS-Based	52.29	39.57	12.71	753	1416	24	236

6.7 Comparison of Initial and RNS-based Processors with Existing Designs

Table 8.11 shows the parametric comparison of the proposed RNS-based image processor with similar designs. Comparison of the number of multipliers shows that all the proposed designs use multipliers except the initial binary-based processor, which uses the proposed multiplier-less filter banks. They use simple left shift. Multiplier-less filter banks are a highlight of the initial processor. They are replaced by modular multipliers in the RNS-based filters. Nevertheless, proposed RNS-based processor uses fewer multipliers than previous designs. The authors in [3] have used LUTs for modular multiplications, which is not hardware efficient. In [151], [152] and [153] the number of adders is twice the number of multipliers, which is not an efficient way of designing filters. The filter banks are optimised to use $2\log_2(N) + 1$ adders, which is one adder more than the number of multipliers in the RNS-based filters. The authors in [3] have used seven times as many adders as multipliers, which introduces a big hardware overhead to the design.

The proposed RNS-based processor needs $N \times (N + L - 1)$ clock cycles to process a 64×64 image block (L is the filter length), while [3] needs N^2 clock cycles to compress 4×4 image blocks. In other words, the proposed processor and the processor in [3] need 65535 and 4608 clock cycles to compress a 64×64 image block, respectively. Hence, proposed processor is 90% faster to compress the same image block.

The existing designs have been selected to be either SI-SO or PI-PO, which affects the speed (pixels per clock cycle). Proposed processor uses the SI-SO scheme, thus it takes one image pixel per clock cycle and generates one compressed data per clock cycle.

The filter banks in [3] have been clocked by a 25 MHz clock signal derived from the main clock. The authors used two separate 180-degree out-of-phase clocks in the lowpass and highpass filter banks. In the proposed design, the filter banks are clocked at the same frequency as the main circuit. Thus, faster modular filter banks are developed

Table 6.18: Comparison of the proposed RNS-based image processor with similar designs

Design	Year	No. of multipliers	No. of adders	Transform	Cycles/block	Speed (pixels/cycles)	Speed/ no. of Multiplier	Transposition	I/o operation	Frequency MHz
Gong [153]	2004	N	$2N$	DCT	N^2	1	$1/N$	No	PI-PO	125
Liu[3]	2004	LUT	$7N$	DWT	1	N^2	N/A	Yes	PI-PO	100
Cheng[154]	2008	24	76	DWT	-	$N^2/3$	$N^2/3$	-	-	58.73
Tian[155]	2011	$4\times Throughputrate$	$8\times Throughputrate$	DWT	-	N	1/4	Yes	PI-PO	63.52
Basant[156]	2012	$4.5N$	$8N$	DWT	8N	–	-	Yes	PI-PO	-
Initial processor	2014	0	$2\log_2(N)+1$	DWT	$N(N+L-1)$	1	-	Yes	SI-SO	100
Proposed RNS processor	2014	$2\log_2(N)$	$2\log_2(N)+1$	DWT	$N(N+L-1)$	1	$1/(2\log_2(N)+1)$	Yes	SI-SO	100

which work under the same frequency of 100 MHz as the main circuit.

6.8 FPGA Implementation of Proposed RNS-Based Image Processor

The proposed RNS-based image processor is synthesised using Xilinx front-end tools. To implement the design on the target device, a synthesised netlist and timing constraints are generated in the *Netlist Constraints File* (NCF) and *User Constraints File* (UCF), and placed on back-end tools to finalise the processor design. Additional placement constraints are also placed on the target device. At the end of the implementation process, a bit-stream file is generated. In the following, implementation process including translate, map and place and route (P&R) is presented.

6.8.1 Translate

For checking the timing and logical-design specifications, the design is translated. Translating involves I/O allocations and setting design constraints. The first constraint to place on the design is timing constraints. Timing constraints are created with a clock period of 100 ns and a 50% duty cycle. Rising-edge constraints (external setup time-offset in and data-valid duration) are setup to 100 ns. The System Synchronous Single Data Rate (SDR) Rising Edge interface is selected to capture one word of data per clock cycle using the rising clock edge. The *Constraints Editor* and *PlanAheadTM* softwares are used to create the UCF file, assign I/O locations and set placement constraints.

The clock in the System Synchronous interface is the same for both the transmitting and receiving devices. To properly analyse this interface, rising-clock-edge registers are constrained with an (OFFSET IN) constraint. The timing constraints for outputs (OFFSET OUT) are set on output ports $y(24)$ to $y(0)$ and a timing group is created.

6.8.2 Map

The next step is to assign I/O locations using the *PlanAhead* software. This pin planning is used for post-synthesis purposes. The I/O ports of the processor are listed under scalar ports:

Input ports: $x(24:0)$, clk, $reset$, $vblocks(3:0)$

Output ports: $end\_blocks, end\_line, fd\_out, y(24:0)$

During mapping, I/O signals are also placed to appropriate pin locations, and the file is saved. The design is mapped to CLBs and IOBs. A set of reports is generated containing warning and error messages from the translation process, information about how the target device resources are allocated, references to trimmed logic, and device utilisation.

After mapping the design, timing analysis is used to evaluate block delays. To avoid timing violations, the post-map static timing report is reviewed and checked. The report generates clk, offset in/out and group timing delays. The setup path, hold path, and component switching limits of the clock path are set. The offset-out-path report also reports the slowest and fastest paths in the design. The post-map static timing reports minimum period $6.852ns$, (Maximum frequency: $145.943MHz$) and minimum output required time after clock: $7.781ns$.

6.8.3 Place and Route

There are two methods to Place and Route (P&R) a design: using timing constraints and ignoring all timing constraints. In the first step of implementation (translate), the timing constraints have been created, so the P&R using timing constraints is performed. The P&R generates a set of reports containing device utilisation and delay summary. The asynchronous delay report also lists all nets and the delays of all loads on the net.

After analysing the reports and optimising the timing and I/O assigning, a configuration bit stream of the design is generated for the selected target device. A configuration file is generated for a Xilinx Serial PROM using *iMPACT*. Figure 6.22 shows the generation PROM file using iMPACT.

6.9 Chapter Summary

Image-processing applications demand faster VLSI architectures of image processors with low power dissipation and less storage space. In this chapter, the logic design and FPGA implementation of the proposed 2D RNS-based image processor are presented. The proposed processor processes images of size 1024×1024. For any other size of images a simple modification in parameter settings is required. Comparing the FPGA synthesis results with the initial processor, the proposed processor is best designed for applications with high number of operations on large numbers in short processing time and less storage space concerns. Hence, it minimises the chip overhead and overcomes the performance restrictions of particular applications. The proposed processor,

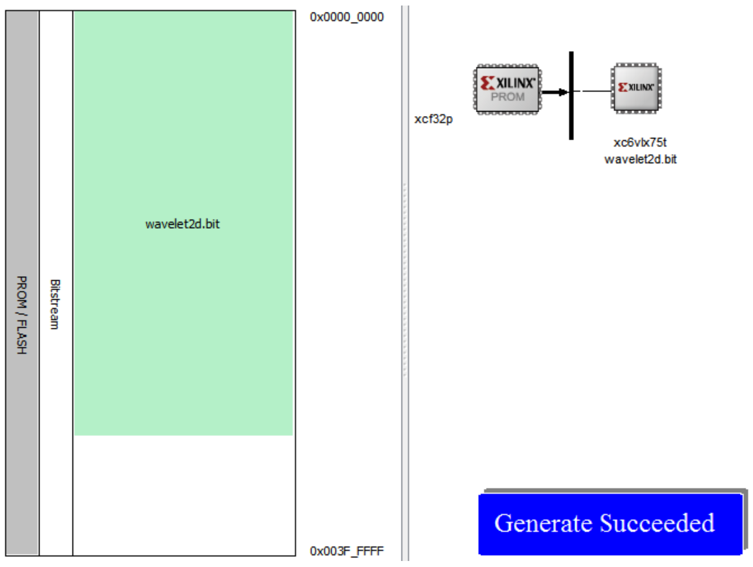


Figure 6.22: Generation of iMPACT PROM file

however, showed slightly more power consumption which is due to the use of modular library. To moderate the power consumption and develop an efficient processor, multi-power domain technique is used to achieve a power-performance enhancement in the next chapter.

Publications pertaining to this chapter:

- Azadeh Safari and Yinan Kong. *Four Tap Daubechies filter banks based on RNS*, International Symposium on Communications and Information Technologies (ISCIT) 2012, pp. 957-960, 2-5 Oct., Gold Coast, Australia.
- Azadeh Safari, Niras C V, and Yinan Kong. *VLSI architecture of multiplier-less DWT image processor*, IEEE TENCON Spring 2013 Conference in Sydney Australia, pp. 280-284, 17-19 April 2013.
- Niras C V, Azadeh Safari, and Yinan Kong. *Overlapped block processing VLSI architecture for separable 2D filters*, National Conference on Emerging Trends in VLSI and ES, January 2013.
- Yinan Kong, CV Niras, Azadeh Safari, *A low-cost architecture for DWT filter banks in RNS applications*, Abstract accepted in International Symposium on Integrated Circuits (ISIC 2014), Singapore, 10-12 December, 2014.

7 RTL-to-Gate Synthesis

In this thesis, three goals are specified in designing the proposed image processor:

Timing - fast operation

Power - low power consumption

Area - small area

The timing is considered as the highest priority in design goals, followed by power consumption and the area requirement. In this chapter, the logic design of the proposed image processor will be synthesised using Synopsys Design Compiler (DC) for the best trade-off between timing, power and area.

7.1 Logic Synthesis

Logic synthesis is the process that creates a logic circuit from a circuit description. Synthesising a design generates the gate-level netlist [5]. To synthesise the proposed image processor, the RTL code, design constraints and environment attributes are defined. The target technology library, symbol library and link library are set. A search path for sources, scripts and libraries is created. Figure 7.1 shows the inputs and outputs of logic synthesis using DC. In the following, input and output files of the

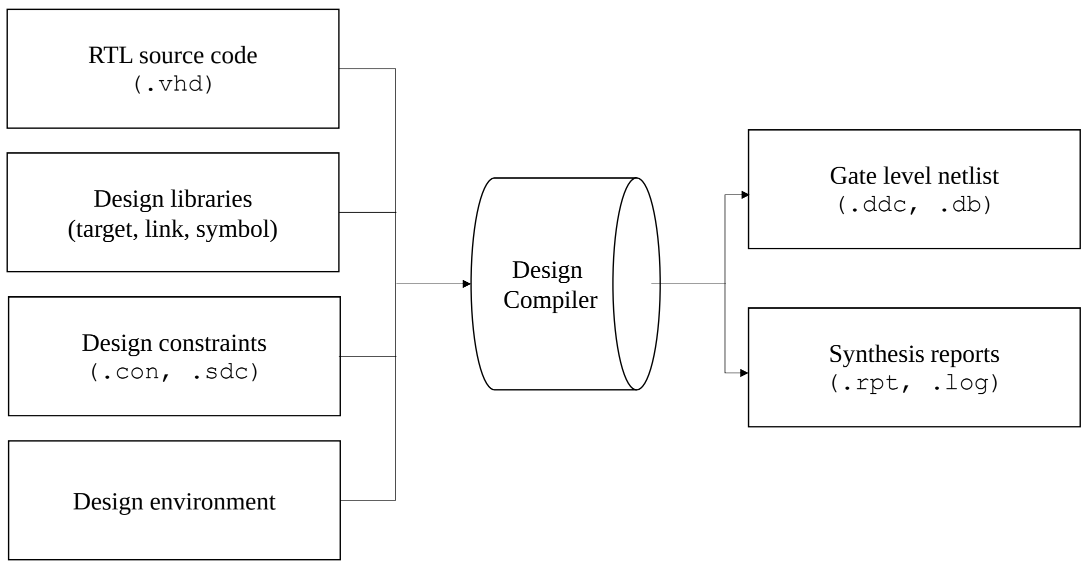


FIGURE 7.1: Input and output files of Design Compiler for logic synthesis

DC are introduced.

7.1.1 The RTL Source

The RTL source-code format can be Verilog, VHDL or System verilog. In Chapter 6, the VHDL source codes have been developed for the proposed image processor. In this chapter, the Tcl (Tool command language) scripts to read the RTL source codes are developed. It is noteworthy to mention that DC checks for validity of the code and the syntax. Correct function of the code is a concern for the designer rather than for DC. DC also checks whether the RTL source code and synthesised netlist are equivalent [5]. Functionality of the source code has been checked in Chapter 6.

7.1.2 Libraries

The target library should be set for optimisation. It contains various logic cells (for example NAND gates) with various areas, drive strengths, power and area requirements. The link library, however, searches the memory to solve the references used in hierarchies. This library will be used after analysing and elaborating the design while linking the references. The DesignWare libraries set advanced architectures. In addition, by setting DesignWare libraries, DC will leverage from more arithmetic and data-path components with more complex architectures [157].

Table 7.1: Setting up the search path, and target, link, symbol and work libraries

Search path	/home/.../ref/models /home/.../work /home/.../src /home/.../db /
Target library	`saed90nm_max_hth.db`
Link library	* `saed90nm_max_hth.db`
Set symbol library	`saed90nm.sdb`
Design Library	WORK

There are two views available from foundaries: the front-end view and the back-end view. The front-end view contains library information for synthesis, and place and route. This view of libraries does not include technology files that are necessary for tape-out. The back-end view contains tap-out information like GDSII, OASIS, LEF and DEF file formats. DC uses the front-end view of libraries [158]. Table 7.1 shows the search path necessary to find the source codes, library files and constraints file. It also shows the and target, link, symbol and work libraries used in this thesis.

7.1.3 Design Constraints

As Figure 7.1 shows, design constraints are also set for logic synthesis. The design constraints are classified as *design-rule constraints* and *optimisation constraints*. The *design-rule constraints* are defined in the technology (target) library (see target library in Table 7.1), hence they are applied to the design when it is mapped to the target library. These constraints must be met for correct function. The *optimisation constraints* are design goals that are followed in the proposed design. Both these constraints are applied to the design; however, design rule constraints have priority over optimisation constraints [5]. Figure 7.2 shows the classification of design constraints.

Table 7.2 shows the design constraints of the proposed processor. The clock period is 100 *ns*. The proposed design has a clock port "clk", hence it creates a real clock; otherwise, it would set a virtual clock. The input delay 1.2 *ns* is set on the input ports. This delay is related to clock port `clk`, and includes all the input ports except the `clk` port itself. The output delay and clock uncertainty are set to 1.5 *ns* and 0.45 *ns*,

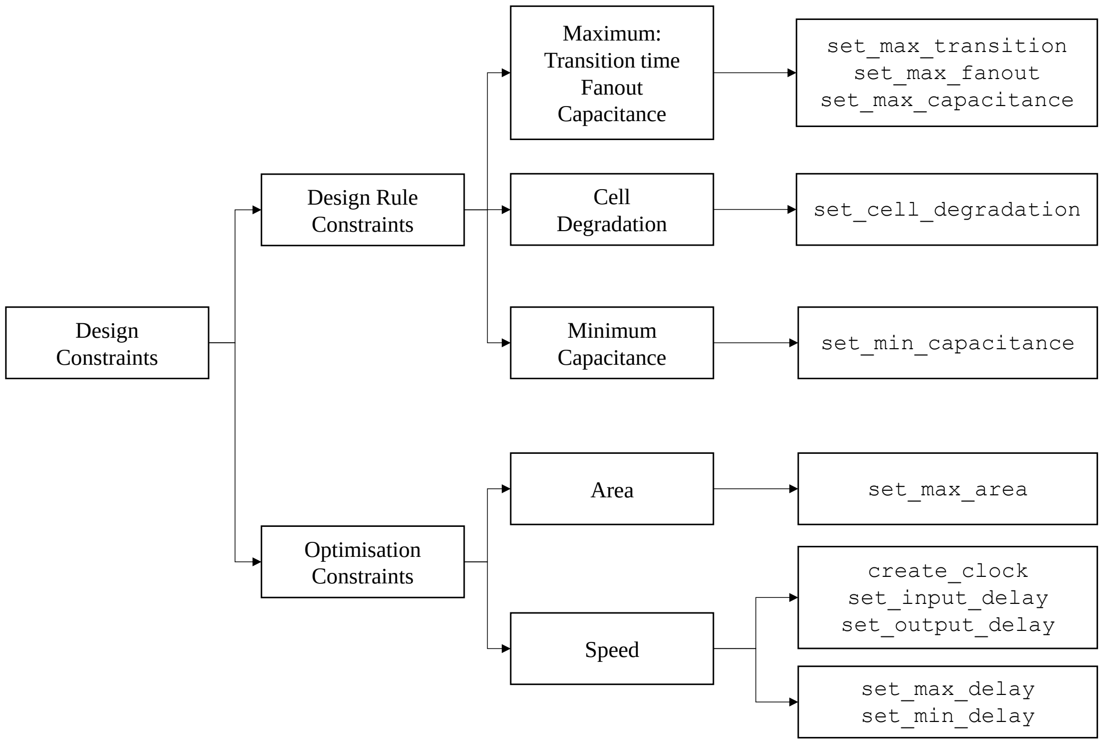


FIGURE 7.2: Design constraints classified as design-rule constraints and design-optimisation constraints [5]

TABLE 7.2: Design constraints

Attribute	Constraint	Port
System clock period	100 *ns*	*clk*
Real clock period	100 *ns*	*clk*
Virtual clock period	100 *ns*	*vclk*
Input delay	1.2 *ns*	vblocks[3] vblocks[2] vblocks[1] vblocks[0] x[24] x[23] x[22] x[21] x[20] x[19] x[18] x[17] x[16] x[15] x[14] x[13] x[12] x[11] x[10] x[9] x[8] x[7] x[6] x[5] x[4] x[3] x[2] x[1] x[0] reset
Output delay	1.5 *ns*	all outputs
Clock uncertainty	0.45 *ns*	*clk*

respectively.

7.1.4 Design Environment

The environmental conditions of the design is specified prior to optimisation as they have direct effect on the synthesis results. Design environment includes operating conditions (temperature, voltage, and process variations), wire-load models, and interface characteristics of design to model the system interface (input drives, input and output loads, and fanout loads) [5]. Table 7.3 shows the design environment of choice. In the following selected design environment attributes are discussed.

Table 7.3: Design environment

Attribute	Constraint	Port
Load on output ports	1.5	all outputs
Driving cell	INVX0	all inputs
Infinite drive strength	0	*clk*
Operating condition	WORST	
Auto wire load selection	True	

7.1.4.1 Operating Conditions

The operating conditions are defined in the selected technology library. They are placed on the design automatically when reading the design from technology libraries. It includes the temperature, voltage, process and interconnect model of the operating environment [5]. Table 7.4 shows the operating conditions of target library `lib90nm_max_hth`. The library has a WORST operating condition.

7.1.4.2 Wire-Load Model and Wire-Load Mode

The technology library provides information about the resistance, capacitance and area of the wire-load model. This information, is needed to calculate the speed and wire delays. There are three methods available to set up the wire-load model: user defined, automatic selection, and technology library specification. A design with no wire-load model selection provides an optimistic timing budget due to a lack of loading and propagate information. There are three wire-load modes for hierarchical designs [5]:

- *Top*: The same wire-load model is used for all nets of the top-level design and all subdesigns. All the wire load model constraints of the subdesigns are overwritten by the top-level constraints.

Table 7.4: Library `saed90nm_max_hth` report

Library type	Technology, PG pin based
Time Unit	1 *ns*
Capacitive Load Unit	1.000000 *ff*
Pulling Resistance Unit	1 $k\Omega$
Voltage Unit	1 *V*
Current Unit	1 μA
Dynamic Energy Unit	0.0010000 *pJ*
Leakage Power Unit	1 *pW*
Operating Conditions:	
Operating Condition Name	WORST
Library	$SAED90nm\_max\_hth$
Process	1.00
Temperature	125.00°*C*
Voltage	1.08
Interconnect Model	Balanced_tree

- *Enclosed*: A more accurate inter-block constraint is to use wire-load model of the smallest design that fully encloses a net. This constraint is recommended for designs with similar logical and physical hierarchies.

- *Segmented*: Nets crossing hierarchical boundaries form divided segments. The segmented wire-load model uses the wire-load model of the design containing that segment.

The auto wire-load selection is default in the selected technology library, and has been set on the design.

7.1.4.3 Interface Characteristics

To accurately model the design's interface with other peripherals, the characteristics of input and output ports are defined.

7.1.4.4 Drive Characteristics of Input Ports and Driving Cell

The drive resistance and capacitance load of an input port determines the transition time delay. Hence, the setting drive characteristics of an input port directly affects the transition delay of that port. The assumption of infinite drive strength is set on input ports by default. This assumption is kept in design environment attributes for heavily loaded driving ports (`clock`) to avoid buffering the net. By setting the drive characteristics on ports that are driven by cells in the technology library, delay calculators can accurately model the drive capability of an external driver. If a driving cell has different arcs (two-input AND gate) with different transition times, the arc with the worst case should be chosen as the driver. The worst-case arc is the slowest in set up violations and the fastest in hold violations. The drive resistance of the top-level design also can be set when the input port drive cannot be characterised with a cell in technology library [5]. The driving cell `INVX0` is chose for all inputs.

7.1.4.5 Load on Input and Output Ports

The typical loads on the input and output ports are selected to obtain the appropriate cell drive strength of the output ports and also models the transition delay on the input ports. The load unit set on the ports are same as the load unit in the target library [5].

7.1.4.6 Fanout_load on Output Ports

Fanout is a capacitive parameter for CMOS. The selected technology library provides information about the maximum fanout limit. The expected fanout load values on the output ports plus the fanout load of cells connected to the output port driver should be less than the maximum-fanout limit of the technology library [5]. The maximum fanout limit of output port `clk` is set to 1000.

7.1.5 Synthesis Reports

The information whether the design meets or violates the constraints is generated in synthesis reports. The most important information is in the following reports [5]:

The constraints report shows the results of checking design rules and optimisation goals.

The library report shows the content of the library including library units, cells, operating conditions (PVT), and the wire-load models.

The timing report is in effect the most important information from design synthesis. A design must meet the timing constraints in order to operate at the intended clock rate. The timing report shows the path start/end point, path delays, data arrival/required time, slack, path-group name and path-timing check type (set up or hold).

The power report provides the leakage power, internal power and multiple power supplies information.

The area report can be generated for the top-level design, or hierarchy levels. The latter one is a detailed area report and provides a hierarchical area distribution.

It gives the combinational (basic logic gates such as AND, OR, ...), non-combinational (registers) and total area of a design. The total area also reports target library, library directory, number of ports, nets, cells, combinational cells, sequential cells, macros, buf/inv, and references. The most common area units are μm^2 and the number of equivalent gates. The library vendor decides on the area unit and provides it in the library data sheet.

Gate density evaluation is either a raw estimation or accurate information. The first approach is based on the two-gate density definition, and the accurate approach is based on the P&R information. It is a design-dependent information.

The TSMC libraries use two gate-density definitions, based on the two-input NAND-gate area, and the ratio of the combinational and sequential cell areas. In a typical design, sequential cells are more common than combinatorial cells (20% to 40% of total cells). The first gate-density definition uses the area of a `ND2D0` cell ($6.2208\mu m^2$) equivalent to one gate for combinatorial cells. In standard-cell libraries, four transistors are equivalent to a gate. The second gate-density definition uses the unit area as the combination of 20 two-input NAND cells and one scan flip-flop ($20*ND2D0 + SDFD1 = 170.0352\mu m^2$). In the TSMC standard cell libraries, the layout of the sequential cells is more compact than the layout of combinatorial cells; hence, typically the first definition (of raw estimation) is greater than the second definition [5]. The selected target library provides the area report in μm^2 units.

Synthesis reports of initial and RNS-based image processors are provided in Section 7.3.

7.2 Synthesis Flow

To synthesise the logic design of image processors, the synthesis flow is followed as shown in Figure 7.3. Initially, the search path, and the link, target, and working libraries are set up. The `saed90nm_max_hth.db` technology library is selected for compiling the design.

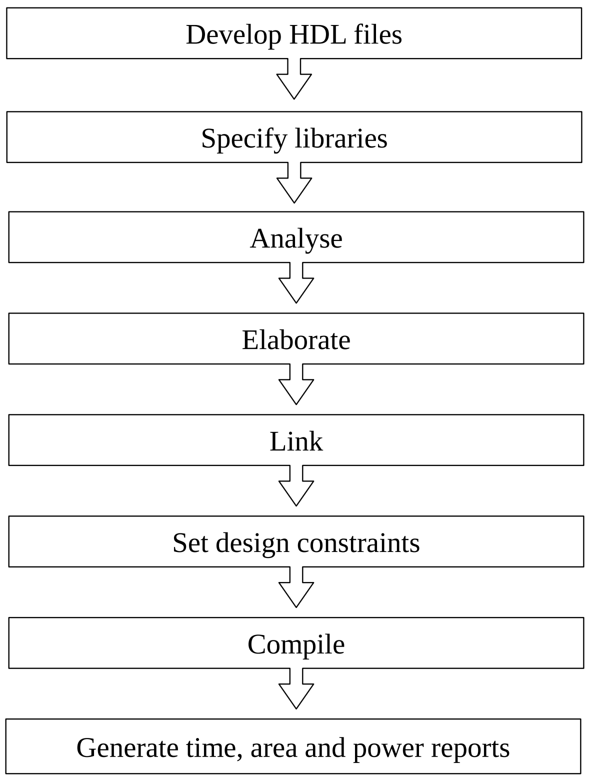


Figure 7.3: DC synthesis flow used for logic synthesis of the image processor

7.2.1 Read Design

As the first step, the source codes are read using DC. The designs are read using "Analyse and elaborate". The first time that the design is analysed, the VHDL source codes are checked for syntax errors. The errors are fixed and the design is analysed again until all the errors are removed. The errors which are about the code format are removed within the Synopsys tools; however, to change or check the functionality of a source code the ISE design suite is used. This is because the Synopsys tools do not check the functionality of the designs. The top-level design and subdesigns can be analysed either one by one or all together. The latter one is fastre, hence it is used. The VHDL source codes of the RNS-based image processor contain a library (*modulo_lib*) and a reference code (*param*) which defines the design parameters, they are analysed first, otherwise analysis was not successful. After analysing the designs, HDL library objects are generated and saved in the working directory.

The VHDL source code of the top-level design (`RNS2D.vhd`) and the hierarchies (subdesigns) of the proposed image processor are analysed.

```
analyze -library WORK -format vhdl {/home/.../src/dwt_param.vhd
/home/.../src/modulo_lib.vhd
.
.
.
/home/.../src/RNS2D.vhd}
```

The top-level design is then elaborated. It translates the top-level design to the technology-independent library `GTECH` using output files from the analysing step. It allocates generic values of the source code, and uses the symbol library to replace the arithmetic operators in the source code. By elaboration out-of-date libraries are re-analysed, and design references are resolved (`link`). Figure 7.4 shows the GTECH schematic of the proposed RNS-based processor including the in/output ports and subdesign modules.

```
elaborate RNS2D -architecture BEHAVIORAL -library WORK -update
link
```

Once the design is analysed and elaborated, they can be skipped, and structural or gate-level source codes are loaded to memory by `read` in next time uses. Reading source codes integrates analyse and elaborate, and does not generate the intermediate files. Hence, linking a design and resolving the references are done separately [5]. Reading can be done individually or for a list of designs. A Tcl file to read the design is used for every time use.

7.2.2 Compile Strategies

Different compile strategies suit different hierarchy levels and components of a design. Three compile strategies can be considered [5]:

-*Top-Down Compile*

The top-down compile strategy is a push-button approach for small designs, that compiles the top-level design and all sub-designs at the same time. The top-down strategy is useful for small designs (less than $100k$ gates) with no memory limitations. Otherwise, it takes too long to compile all levels of hierarchy simultaneously.

-*Bottom-Up Compile*

The Bottom-up compile strategy compiles all the sub-designs of a medium to large design, separately. The compile starts from the bottom of the hierarchy and compiles up through the hierarchy levels to the top-level design. The top-level design is the last design compiled. This strategy requires less memory and takes less running time.

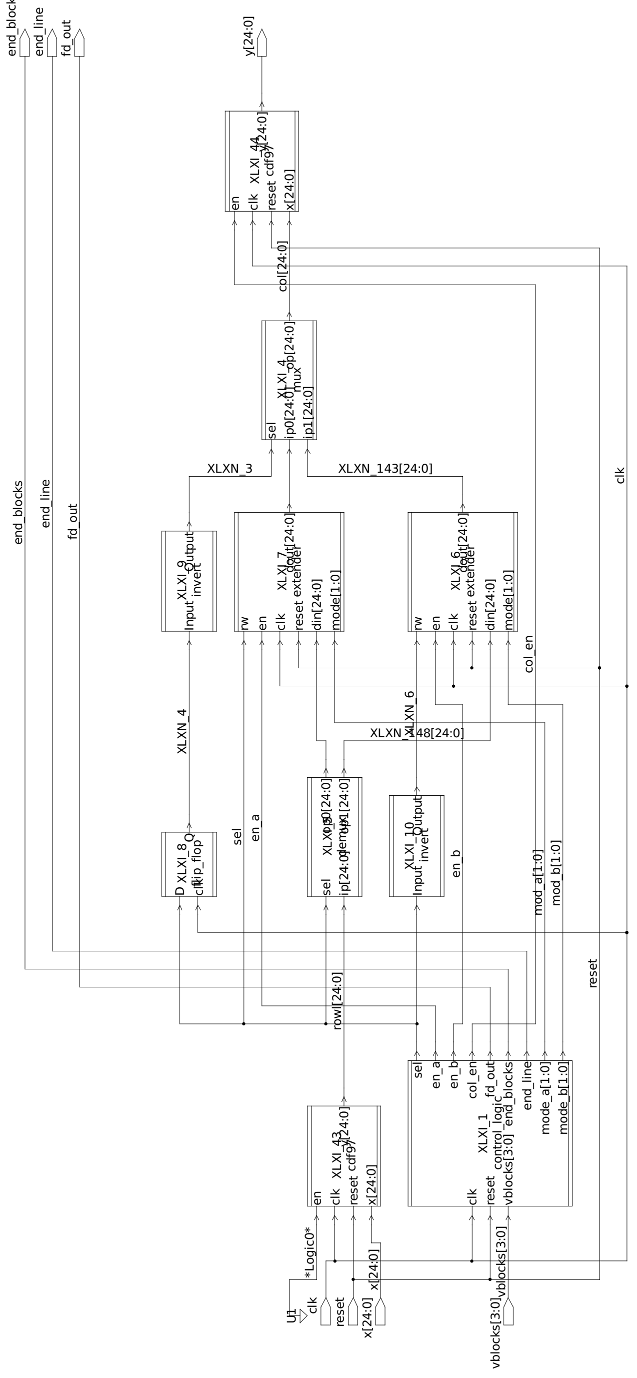


Figure 7.4: Schematic of GTECH (technology independent) of the proposed 2D RNS-based DWT digital image processor

-*Mixed Compile*

The mixed compile strategy takes advantage of both the top-down and bottom-up strategies. The sub-designs use either of the strategies, whichever is most appropriate. The small hierarchies of blocks use the top-down compile strategy. The bottom-up compile strategy is used for larger hierarchies of blocks. If a block has no hierarchy, the time budgets and specifications of the first level of hierarchy will be applied to it.

The top-down compile strategy is used to compile the image processors. This strategy is selected because it is straight forward, and suitable for designs with small hierarchy levels.

The proposed design contains multiple-referenced subdesigns (`CDF97, EXTENDER, invert`). To compile them, there are three compiling methods [5]:

- **The compile-once-don't-touch method:** This method compiles the reference design and preserves it while the remaining designs are compiled. First the subdesign's instance with the worst-case environment is characterised and the referenced subdesign is compiled. Using the set-don't-touch method will preserve all instances referenced to a compiled subdesign. It requires less memory and less running time than the uniquify method below.

- **The ungroup method:** This method goes one step further than the uniquify method by removing the user-defined design hierarchy. It creates unique copies of the subdesign and also removes the hierarchy. This method provides the best synthesis results but cannot be applied to designs with the don't-touch attribute. This method requires less compiling time than the compile-once-don't-touch method. This method is not used in this thesis to retain the hierarchy levels of the design.

- **The uniquify method:** In this method the design compiler copies the original subdesign and creates new instances by renaming them uniquely for multiple uses, and then removes the original design from memory. The new naming is based on the original name and can be used when there are no memory or compile-time restrictions.

The compile-once-don't-touch and uniquify methods are used for multiple-referenced subdesigns while comiling the design. The design is compiled with a global voltage (1.08 V), and saved as `RNS2D_MAPPED.ddc`. Another important file to save is `RNS2D.sdc`, which is the design intent, including the timing, power and area constraints. This file is used in the next chapter for the physical implementation.

7.3 Comparison of Proposed RNS-based Image Processor with Initial Binary Processor

In this section, the synthesis results of the initial binary processor and the proposed RNS-based processor from previous chapter are compared. For this purpose, synthesis reports are generated to evaluate designs' performance in terms of timing, area and power consumption. The timing report is the most important information from design synthesis. It contains the path start/end point, delays for the paths, data arrival/required time, slack, path group name and path timing check type (SETUP or HOLD). The power and area hierarchy of the initial binary processor and the proposed RNS-based processor are generated for full path, max delay and max paths. The power consumption of each cell as well as the area requirement is provided in Tables 7.5 to 7.8. The total area is the sum of the total cell area and the net-interconnect area. The total cell area is the combinational area, buf/inv area and non-combinational area. The net-interconnect area is the area of the wires that connect cells.

In some libraries, the area information for the wire-load models are not included. So the net-interconnect and total area are left undefined. It appears as: "wire load has zero net area" at the end of the area report

Table 7.5: Power hierarchy of all the references of the initial binary image processor

Cell	**Switch power**	**Internal power**	**Leakage power**	**Total power**	**%**
cdf97_row	0.582	40.577	2.32e+08	272.684	0.2
Extender_0	0.197	1.93e+04	6.17e+10	8.10e+04	49.8
Demux	0.116	3.323	4.84e+07	51.860	0.0
CDF97_col	5.301	52.749	2.40e+08	297.874	0.2
Control_logic	0.226	6.719	2.04e+07	27.386	0.0
Mux	4.58e-04	5.87e-03	3.57e+06	3.577	0.0
Extender_1	151.164	1.94e+04	6.15e+10	8.10e+04	49.9
OR_gate	0.000	0.000	1.18e+05	0.118	0.0
Invert_0	6.65e-05	1.38e-05	4.75e+04	4.76e-02	0.0
Invert_1	5.15e-06	5.17e-04	1.65e+06	1.649	0.0
Flip_flop	0.000	0.151	3.47e+05	0.498	0.0
Total	159.338	3.88e+04	1.24e+11	1.63e+05	100.0

Table 7.6: Area hierarchy of all the references of the initial binary image processor

Cell	**Absolute total**	**Combinational**	**Non-combinational**	**%**
cdf97_row	26568.8063	4398.7967	8451.0721	0.2
Extender_0	5945173.3539	3250.4832	1322.4960	49.9
Demux	1225.7280	1225.7280	0.0000	0.0
CDF97_col	26568.8063	4398.7967	8451.0721	0.2
Control_logic	2278.1952	804.5568	1281.9456	0.0
Mux	355.7376	355.7376	0.0000	0.0
Extender_1	5911585.6484	3442.1760	1322.4960	49.6
OR_gate	7.3728	7.3728	0.0000	0.0
Invert_0	5.5296	5.5296	0.0000	0.0
Invert_1	31.3344	31.3344	0.0000	0.0
Flip_flop	31.3344	0.0000	31.3344	0.0
Total	11913867.7894	5922577.5968	5991290.1926	100.0

Table 7.7: Power hierarchy of all the references of the RNS image processor

Cell	**Switch power**	**Internal power**	**Leakage power**	**Total power**	**%**
cdf97_row	0.548	38.870	2.41e+08	280.230	0.2
Extender_0	3.740	1.86e+04	7.35e+10	9.21e+04	49.8
Demux	5.09e-04	2.69e-04	3.00e+06	3.000	0.0
CDF97_col	0.581	38.957	2.45e+08	284.512	0.2
Control_logic	0.226	6.718	2.04e+07	27.336	0.0
Mux	4.57e-04	2.93e-03	3.44e+06	3.441	0.0
Extender_1	0.400	1.86e+04	7.38e+10	9.23e+04	49.9
OR_gate	0.000	4.021	5.51e+06	9.530	0.0
Invert_0	6.62e-06	5.15e-04	1.65e+06	1.649	0.0
Invert_1	6.65e-05	1.37e-05	4.75e+04	4.76e-02	0.0
Flip_flop	0.000	0.154	2.74e+05	0.428	0.0
Total	7.798	3.72e+04	1.48e+11	1.85e+05	100.0

Table 7.8: Area hierarchy of all the references of the RNS image processor

Cell	**Absolute total**	**Combinational**	**Non-combinational**	**%**
cdf97_row	26543.9232	8588.3903	8096.2561	0.2
Extender_0	5919492.9615	30.4128	0.0000	49.8
Demux	442.3680	442.3680	0.0000	0.0
CDF97_col	26848.0512	8892.5183	8096.2561	0.2
Control_logic	2272.6656	799.0272	1281.9456	0.0
Mux	343.7568	343.7568	0.0000	0.0
Extender_1	5919267.1696	31.3344	0.0000	49.8
OR_gate	622.0800	0.0000	622.0800	0.0
Invert_0	31.3344	31.3344	0.0000	0.0
Invert_1	5.5296	5.5296	0.0000	0.0
Flip_flop	32.2560	0.0000	32.2560	0.0
Total	11897814.4158	6129936.6493	5767877.7666	100.0

Table 7.9: Critical path delay (In/out put: 25 bits, Filter coefficients: 6 bits, Operating voltage: 1.08 V)

	Initial binary processor	Proposed RNS-based processor
Timing constraints (ns)	29	27.12
Slack	MET	MET

The performance of the proposed RNS-based processor is compared with the initial binary processor. For consistency and accuracy of performance comparison, similar operating conditions are chosen for compiling designs. Table 7.9 shows the critical path delay of the initial binary and the RNS-based processor for 25-bit in/output and 6-bit filter coefficients when the global operating voltage is 1.08 V. The results indicate that the RNS-based processor is 34.6% faster than the initial binary processor and has met the timing constraints.

The area reports are summarised in Table 7.10. It shows that there is a small improvement in the total cell area of the RNS-based design. The results confirm that the new architecture of the proposed processor has reduced the hardware complexity.

The dynamic power, leakage power and total power consumption of the top-level design are presented in Table 7.11. It shows that there is more leakage power and hence more total power for the RNS-based design. It is interesting to note that, although

Table 7.10: Area comparison of the binary and RNS designs (In/out put: 25 bits, Filter coefficients: 6 bits, Operating voltage: 1.08 V)

Area (μm^2)	Initial binary processor	Proposed RNS-based processor
Combinational area	5922577	6129936
Buff/Inv area	1158010	1411490
Non-combinational area	5991290	5767877
Net interconnect area	3677441	3172405
Total cell area	11913867	11897814
Total area	15591308	15070219

Table 7.11: Top-level design power analysis of initial binary and RNS processors (In/out put: 25 bits, Filter coefficients: 6 bits, operating voltage: 1.08V)

Power (mW)	Initial binary processor	Proposed RNS-based processor
Internal power	38.8487	37.2254
Net switching power	162.9202	6.1124
Total dynamic power	39.0117	37.2315
Leakage power	123.5424	147.70
Total power	1.6248e+02	1.8507e+02

the proposed processor saves on delay and area, it demands more power for the higher speed and superior functionality than the initial binary processor.

Synthesis results of the proposed image processor shows a massive leakage power in the memory modules of the extenders. Hence, reducing the power consumption of the memory modules can contribute to a better design. In other words, circuit-level power optimisation should be applied to the memory modules to achieve the maximum power reduction. Scripts for synthesising initial binary and the proposed RNS-based image processors using Synopsys DC are provided in Appendix C.

7.4 Optimising Power Consumption of Proposed Image Processor

In this section, power management trends and developments, and the challenges of low-power design are studied. The physics of sources of power consumption is discussed and low-power-design (LPD) strategies are considered.

7.4.1 Components of Power Dissipation

Dynamic and static (leakage) power are two sources of power consumption. The dynamic power is generated by current flow when the device is operating and signals are changing. It is composed of the switching current and the short-circuit current. In CMOS transistors, the switching current is used to charge and discharge the output capacitance on a gate. The short-circuit current is consumed when both the NMOS and PMOS transistors are on during switching of the cell [9].

The static (leakage) power is used for powering up the device with no change in signals. The main source of static power consumption in CMOS devices is leakage power. In new technologies, the leakage power tends to be greater than the dynamic power; hence, power-consumption management should receive extra care. Equation 7.1 shows the power consumption components [9]:

$$P_{total} = P_{dynamic} + P_{static} \tag{7.1}$$

The dynamic (7.2) and static (7.3) power consumption in a combinational circuit is determined using leakage power (nW/gate), working frequency (MHz), switching activity (F) and gate count of combinational logic (S_{comb}).

$$P_{dynamic} = P_{comb} \times F \times S_{comb} \times N_{comb} \tag{7.2}$$

$$P_{static} = P_{leakage} \times N_{comb} \tag{7.3}$$

As the proposed image processor suffers from large leakage power in Extenders, sources of the leakage power are considered.

7.4.2 Leakage Power Sources

Three major leakage-current sources are the subthreshold leakage, the gate leakage, and the reverse-biased-junction leakage.

- The subthreshold leakage

The subthreshold leakage occurs when the MOSFET is operating in the subthreshold region. In this region, the CMOS gate is not turned off completely. In this case, the magnitude of the threshold voltage is greater than the magnitude of the gate-to-source voltage. Equation 7.4 shows the subthreshold current (I_{sub}), which is exponentially dependent on (V_{th}):

$$I_{sub} = I_o e^{\frac{V_{gs}-V_{th}}{nV_T}} \left(1 - e^{\frac{-V_{ds}}{V_T}}\right) \tag{7.4}$$

where $I_o = \frac{W\mu_0 C_{ox} V_T^2 e^{1.8}}{L}$, $V_T = \frac{KT}{q}$ is the thermal voltage, V_{gs} is gate-source voltage, V_{ds} is the drain to source voltage, V_{th} is threshold voltage, n is a function of the device fabrication process between [1.0, 2.5], μ is the carrier mobility, C_{ox} is the gate capacitance, and W and L are the dimensions of the transistor [159].

- The gate leakage

The gate leakage occurs when electrons directly flow by tunneling through the gate oxide. The advanced processes with small oxide thickness, oxide field and voltage drop across the oxide are particularly vulnerable for gate leakage.

- The reverse-biased-junction leakage

The reverse-biased (drain-substrate and source-substrate) junction leakage occurs when the high electric field across a reverse-biased p-n junction causes significant current to flow through the junction (band-to-band-tunnelling). This happens due to tunnelling of electrons from the valence band of the p-region to the conduction band of the n-region.

$$I = I_o \left(e^{\frac{-qV}{kT}} - 1\right) \tag{7.5}$$

where I is p-n junction current, I_o is the reverse leakage current, and V is the reverse junction voltage. As (7.5) shows, the junction-tunneling current depends exponentially on the junction doping and the reverse bias across the junction. In a MOSFET, when the drain-substrate and/or the source-substrate junction is reverse biased at a potential higher than that of the substrate, a significant current flows through the junctions. In nano-technology devices, this current may increase the total leakage current due to higher doping at the junctions [9].

As nano-technology library is used to synthesis the proposed processor, all three leakage sources are significant. Hence total leakage power has increased significantly. The importance of each leakage current source changes with technology node. For the 90nm technology library in this thesis, gate leakage current is concerned. For technologies below 90nm, all three leakage sources are important [9].

7.4.3 Low-Power Design Techniques

There are various low-power design (LPD) approaches to reduce total and leakage power, from the early steps of circuit design to the very last steps of system implementation. The most common LPD methods are the clock-gating, power-gating, multi-threshold and multi-voltage approaches [9]. These techniques are introduced in the following, and the proper LPD method is investigated for lowering the leakage power.

- Clock gating:

Each IC or chip has a large number of buffer cells to distribute the clock signal equally to the chip. These chips usually operate at very high frequencies, hence buffer cells are switching very fast and consuming large switching power. Therefore, more than 50% of the dynamic power dissipates in the clock buffers. Besides, ICs have different activity patterns in different modules.

The clock gating technique eliminates clock distribution to regions that are not active (operating) at certain times, and saves a considerable amount of dynamic power. There are different clock gating approaches: general, multi-stage and hierarchical.

In the general clock-gating approach a special circuit called a "clock gating cell" is used with clock signals that drive a large number of buffer cells. It prevents clock distribution through a register bank when it is not necessary and the data is not enabled. The clock gating should be done without changing the logic function of the circuit. The clock gating cell contains an AND gate and a latch. The AND gate disables the lock signal and the latch synchronises the enable and clock signals.

Clock gating can be applied not only near register banks but also in larger parts. In this approach, the clock is gated globally for the blocks that are not operating at a given time. These blocks should not be hierarchically related. Another approach is using clock gating for hierarchical levels. This method is useful for hierarchy levels that share a common enable and fall in the same clock group. It also reduces the number of clock-gating cells automatically.

- Multi-threshold:

In the multi-threshold technique, multi-threshold (3-5) transistors are used in the same design. The low-threshold-voltage transistors are faster than high-threshold-voltage transistors. They have more sub-threshold leakage current, though. The low-threshold voltage (LVT) cells are used in critical paths (delay on these paths should be the smallest, hence they consume more power). The rest of the design uses high-threshold voltage (HVT) cells to minimise leakage power and reduce the delay [160].

- Power gating:

The power-gating technique uses special transistors (switches) called "sleep control transistors" and turns off the power supply of modules that are not operating at a given time. The sleep transistors are selected from high-threshold-voltage transistors to minimise the leakage power. They are placed between the permanent power supply and the main circuit power supply.

- Multi-voltage:

The muti-voltage technique is also based on the different activity profiles of the circuit. In the multi-voltage technique, the internal logic of the chip is partitioned into multiple voltage regions (power domains), each with its own supply. The blocks with a low performance requirement receive less voltage and save power. In contrast, the blocks with a high performance requirement receive high voltage and dissipate some power. Hence, a trade-off between power and performance is necessary for different blocks.

The idea behind the multi-voltage technique is that the instantaneous power is the power drawn from a voltage source supplied through the VDD port (7.6).

$$P(t) = i_{DD}(t)V_{DD} \tag{7.6}$$

If the block's voltage always remains constant, the method is called "static multi-voltage". If the voltage and frequency are scaled based on a block's performance, the method is called "dynamic voltage and frequency scaling". This is due to the scaling of frequency to maintain the block functionality after scaling the voltage. The voltage and frequency of blocks can be controlled based on on-chip conditions and performance requirements using a voltage regulator. This approach is called "adaptive voltage and frequency scaling". The multi-voltage technique can be combined with the power-gating method using sleeping-control transistors, and applied to non-critical functional blocks [9]. Table 7.12 shows a comparison of common LPD techniques.

TABLE 7.12: Comparison of common LPD methods [9]

Technique	Power benefit	Timing penalty	Area penalty	Impact on			
				Architecture	Design	Verification	P&R
Multi V_{th}	Medium	Little	Little	Low	Low	None	Low
Clock gating	Medium	Little	Little	Low	Low	None	Low
Multi voltage	Large	Medium	Little	High	Medium	Low	Medium

According to Table 7.12, the multi-voltage method shows the largest power benefit. It is a simple and efficient technique for designs with different performance objectives. The MV method can be implemented using several strategies as shown in Figure 7.5.

- **Static voltage scaling (SVS):** Different blocks are supplied with a selected fixed supply voltage.

- **Multi-level voltage scaling (MVS):** The voltage of blocks switches between a few (two or more) voltage levels.

- **Dynamic voltage and frequency scaling (DVFS):** The voltage of blocks switches between a larger number of voltage levels than the multi-level voltage scaling approach.

- **Adaptive voltage scaling (AVS):** A control loop adjusts the voltage of DVFS blocks.

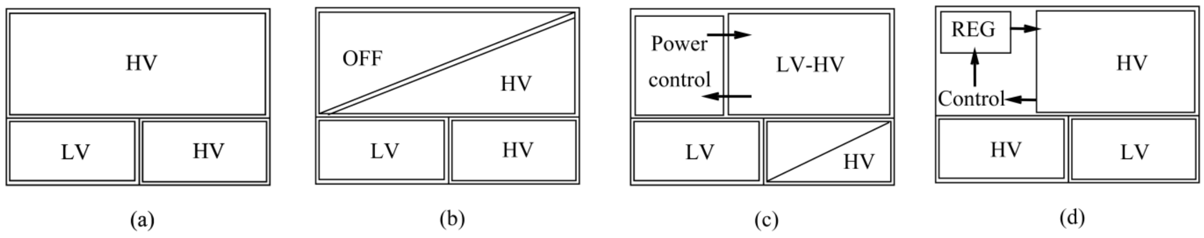


Figure 7.5: The multi-voltage strategies: (a) Static, (b) With power gating, (c) Dynamic voltage/frequency scaling, (d) Adaptive voltage scaling (AVS) [6]

In this thesis, the static voltage scaling (SVS) approach of the multi-voltage technique is used for transposition units to reduce the total power consumption. In this method different blocks are supplied with a selected fixed supply voltage.

7.4.4 Impact of LPD Techniques on Standard Cell Libraries

To implement a design using LPD techniques, different versions of libraries and special versions of cells are required. Furthermore, additional data and attributes are needed for tools to be able to work with power-related data. In addition to the standard characterisation corners used for regular libraries, additional corners are needed to place special cells in non-regular operation modes [158].

The special cell requirements and special versions of libraries for each LPD technique are as follows:

Special cell requirements

- Clock gating technique: Clock gating cell
- Multi-voltage technique: Level shifter
- Power gating technique: Power gating cell (turning blocks on and off)

State retention register (keep state of the register when the block is shut off)

Isolation cell (keep outputs at known state when the block is shut off)

Special versions of libraries

- Multi-threshold technique: Multi-V_{th} libraries (low V_{th}, standard V_{th} and high V_{th})

-Multi-voltage technique: Libraries with level shifters

In this thesis, the SVS approach of the MV technique is used, hence specialised libraries with special cells (level shifters) are necessary to meet the requirements of the design. Specialised cells are required to prevent timing closure problems and excessive short-circuit (crowbar) switching currents. They are used duo to the fast transmission of signals between different power domains.

7.4.5 Impact of LPD Techniques on Synthesis Flow

To implement a chip using LPD techniques, they should be included at the earliest design stages and at almost every step of the design flow. In the early stages of design, the modules with maximum performance and the modules which can be shut down during some periods of time should be known. Furthermore, logical and physical synthesis tools need to be modified to accept the power intent or design strategy as an input [26].

Formal verification tools should be modified to verify the gate-level design and initial RTL. Additional file formats like the Unified Power Format (UPF) are also required to follow a power-aware design. The clock-gating and multi-threshold techniques can be implemented automatically, but the power-gating and multi-voltage techniques need a specification of power intent (UPF). The UPF file can be generated at each step of design (design specification, logic synthesis and physical synthesis) separately, and be modified as required [26].

To synthesise the logic design of the proposed image processor using the multi-voltage technique, the synthesis flow shown in Figure 7.6 is followed.

7.5 Impact of the Multi-Voltage LPD Technique on Quality of Results

The synthesis results of the previous section confirm the expectation that the majority of the power consumption of the system, when implemented with the nano-technology process, is the leakage power, and the extenders (transposition units designed with

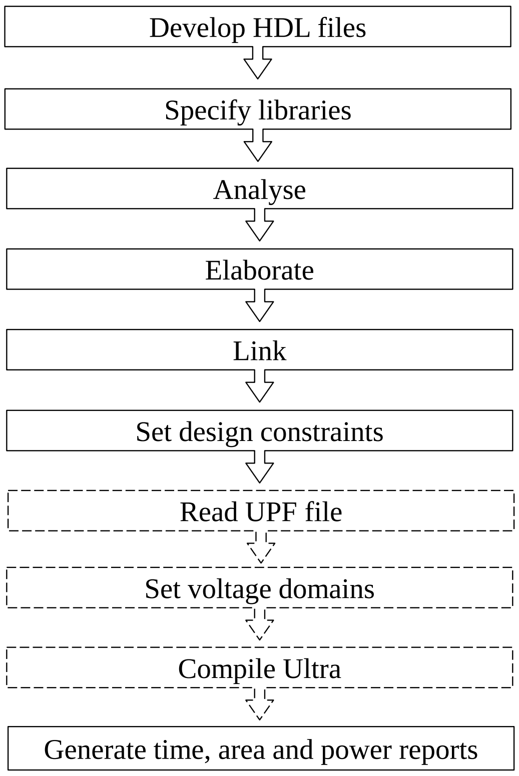


Figure 7.6: Optimised synthesis flow to synthesise the proposed RNS-based image processor using the multi-voltage technique

static RAM) are the main source of leakage power. To moderate the leakage power and decrease the total power, the synthesis methodology is modified based on a quality-of-results (QoR) strategy with unified-power-format (UPF) flow. The flow is designed to optimise the power consumption of the design using the SVS method. For this purpose, the chip is partitioned into two power domains: the extender, and the top-level and cell domains. The voltage level of the extender power domain is set lower (LV=0.7 V) than the voltage of the top-level and cell domains (HV=1.08 V) while they share a common VSS. The technology library `saed90nm_max` with low operating voltage is added. Table 7.13 shows the operating conditions of target library `saed90nm_max`, and Table 7.14 is the Power State Table (PST) showing the supply port, the state name and the corresponding voltage level.

In order to create two different power domains, special library cells (level shifters) are required to convert signals between selected voltage levels. They shift a signal that comes in at one voltage level and goes elsewhere at a different voltage level. Another

TABLE 7.13: Library `saed90nm_max` report used as low-voltage library

Library type	Technology, PG pin based
Time Unit	1 ns
Capacitive Load Unit	1.000000 ff
Pulling Resistance Unit	1 $k\Omega$
Voltage Unit	1 V
Current Unit	1 μA
Dynamic Energy Unit	0.0010000 pJ
Leakage Power Unit	1 pW
Operating Conditions:	
Operating Condition Name	WORST
Library	$SAED90nm\_max$
Process	1.00
Temperature	$125.00°C$
Voltage	0.70
Interconnect Model	Balanced_tree

TABLE 7.14: Power state table (PST) of the multi-voltage processor

Supply Port	State name	Voltage level (V)
VDD	HV	1.08
$VDDXS$	HV	1.08
$VDDGS$	LV	0.7
VSS	GND	0

reason for using level shifters is that a 0.7 V signal, driving a 1.08 V gate, will turn on both the NMOS and PMOS networks, causing crowbar currents. Hence, high-to-low and low-to-high level shifters are employed in the proposed design.

A high-to-low level shifter is composed of two inverters in series with a single power rail from a lower power domain. The delay introduced by a high-to-low level shifter is a buffer delay and can be ignored. However, the low-to-high level shifters have more complicated circuitry. Each is composed of a buffer and inverter for the lower-voltage signal. The inverted voltage is used to drive cross-coupled transistors running at the higher voltage. Figure 7.7 shows the high-to-low and low-to-high level shifters.

Figure 7.8 shows the placement of the level shifters in the power domains. They

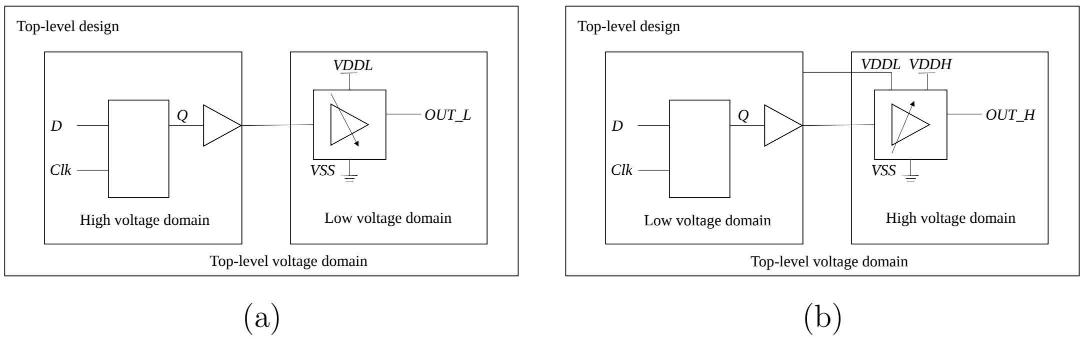


Figure 7.7: Structure of level shifters: (a) High to low, (b) Low to high

are placed in the receiving domain (the lower domain for high-to-low shifters and the higher domain for low-to-high shifters). The extender power domain operates at a lower voltage, hence high-to-low level shifters are inserted in the extender power domain and low-to-high level shifters are placed in the cell power domain.

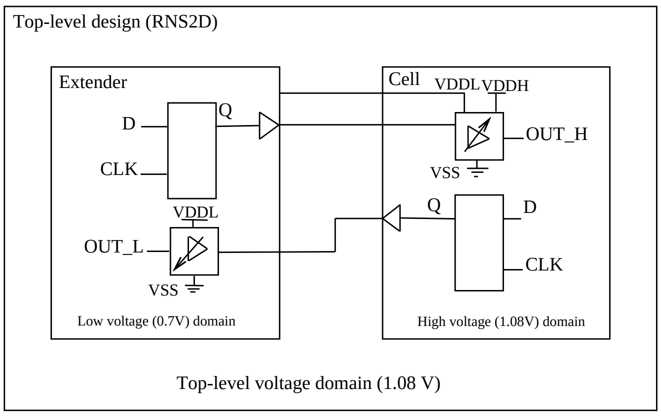

Top-level design (RNS2D)
Extender
D
CLK
Q
VDDL
OUT_L
VSS
Low voltage (0.7V) domain
Cell
VDDL VDDH
OUT_H
VSS
Q
D
CLK
High voltage (1.08V) domain
Top-level voltage domain (1.08 V)

Figure 7.8: Level-shifter placements in power domains

For consistency reasons, when comparing the initial binary and proposed RNS-based designs an identical UPF file and PST are used for both designs. The UPF file has specified power domains, power supply network (supply ports, supply nets, power network connections), power state table, power switches, special dual-supply and cell strategies on a per-power-domain basis (isolation, retention, level shifter). The UPF diagram of the RNS-based processor is shown in Figure 7.9.

The RNS-based and binary designs are then compiled using the SVS method and the critical path delay, area and power consumption reports are presented in Tables

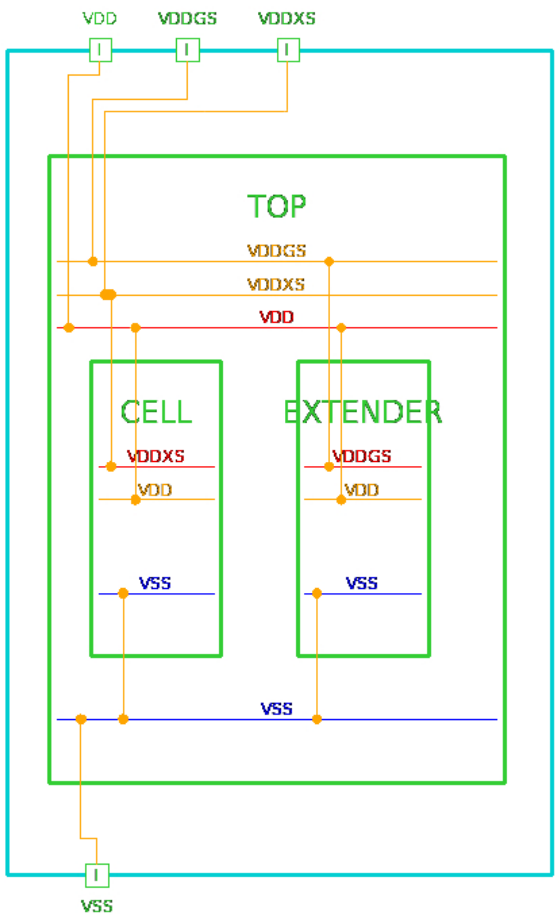


Figure 7.9: Unified Power Format (UPF) diagram of the proposed image processor

7.15 to 7.17. Tcl Scripts for Multi-Voltage Synthesis of Initial Binary and the Proposed RNS-Based Image Processors Using Synopsys DC Topographical Mode are provided in Appendix D.

Table 7.15: Critical path delay of the initial binary and proposed RNS-based designs (In/out put: 25 bits, Filter coefficients: 6 bits, Top-level voltage: 1.08 V, Extender voltage: 0.7 V)

	Initial Binary processor with SVS	Proposed RNS-based processor with SVS
Timing constraints (ns)	47	35.24
Slack	Met	Met

Table 7.16: Area comparison of the initial binary and proposed RNS-based designs (In/out put: 25 bits, Filter coefficients: 6 bits, Top-level voltage: 1.08 V, Extender voltage: 0.7 V)

Area (μm^2)	Initial Binary processor with SVS	Proposed RNS-based processor with SVS
Combinational area	4736294	4910879
Buff/Inv area	126695	151376
Non-combinational area	5984146	5628070
Net interconnect area	6446384	5946379
Total cell area	10720440	10690325
Total area	17166825	16636704

Table 7.17: Top-level-design power analysis of initial binary and proposed RNS-based processors (In/out put: 25 bits, Filter coefficients: 6 bits, Top-level voltage: 1.08 V, Extender voltage: 0.7 V)

Power (mW)	Initial binary processor with SVS	Proposed RNS-based processor with SVS
Internal power	31.6545	29.0756
Net switching power	48.3741	3.9957
Total dynamic power	31.7229	29.7540
Leakage power	48.0803	62.3578
Total power	79.903	91.716

The comparison of the critical path delays in Table 7.9 shows that using the RNS in the arithmetic level of computationally intensive systems reduces the critical path delay, which is consistent with other studies [3] and [14]. A similar pattern occurs in studies of applying the SVS method. Applying the SVS method has reduced the critical path delay in both the RNS and 2's-complement image processors.

The area requirements of each design are also provided in Tables 7.10 and 7.16. It is interesting that there is only a little area penalty (up to 9%) in the total area of the processors after applying the SVS method, which can be due to using specialised library cells such as level shifters. Another noteworthy result over the previous studies is that the total area of the RNS-based processor is less than the total area in the initial binary processor. It confirms that using the proposed architecture for RNS-based filter banks, and the binary-coded number format, has saved on the hardware complexity and the system area requirement. For area analysis, the impact of using the multiplier-less filters on area reduction is neglected since the same filter structure is used for both the RNS-based and initial designs.

The power consumption of the RNS-based and initial binary designs in Table 7.11 showed that, although the RNS-based processor has better performance in area and delay, it dissipates more power than the 2's-complement processor. To overcome this issue, the SVS method has been chosen to improve the power performance of the proposed design. It shows that using the SVS method reduces the total power of the initial binary and the RNS-based designs by up to 50%.

The present results are significant in at least two major aspects of system design: the arithmetic-level optimisation and the low-power-design development. The results support the idea that arithmetic-level optimisation with circuit-design techniques are able to achieve the best trade-off for power-performance of the existing processors.

7.6 Preparing the Design for Power Analysis Using VCS

Further simulations on the proposed RNS-based image processor are performed using VCS-MX. VCS is the Synopsys simulator tool, and DVE (Discovery Visual Environment) is the VCS graphic user interface (GUI). DVE is used for functional verification, and to view waveforms and schematics.

The VHDL source codes of the top-level design and subdesigns, with the test bench are debugged and simulated to generate a VCD file, which is necessary for power analysis. Three general steps to generate the VCD file are [161]:

- Analyse

  The VHDL source codes of all designs are analysed using `vhdlan`. Since VCS is a simulator, the test bench is also analysed using Synopsys 1076 VHDL Analyser G-2012.09 in the same directory. The simulation machine is 64-bit, hence `vhdlan` is run in `-full64` option.

- Elaboration

  After analysing the source files, the test bench is elaborated as the top-level design. Elaboration is the second step, that builds the instance hierarchy and generates a binary executable. The elaboration can be done either in "optimised (batch)" or "debug (interactive)" mode. It is recommended to run the debug mode before optimised mode. Figures 7.10 -7.17 are the schematic of the test bench as the top-level design, and all subdesigns.

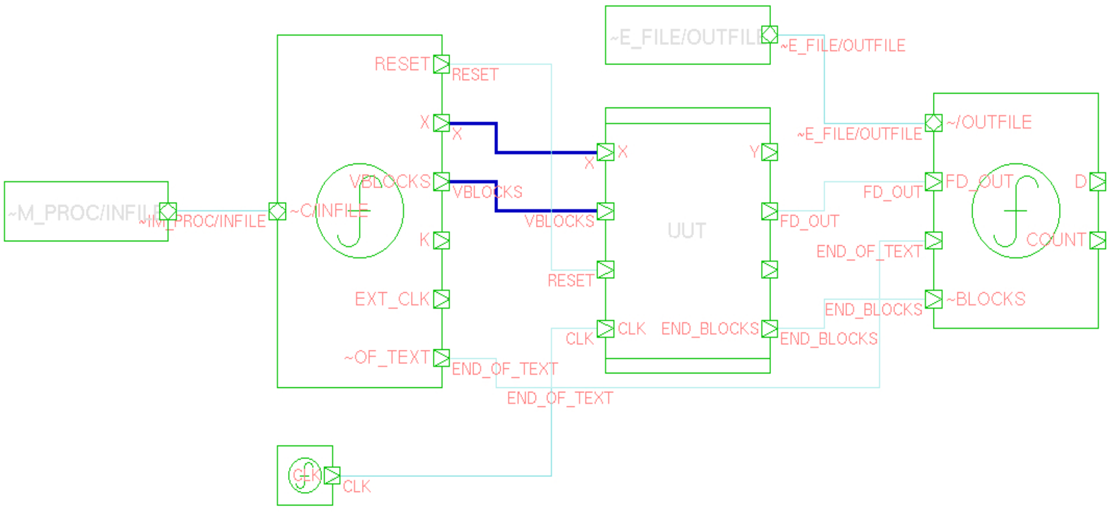


Figure 7.10: VCS schematic of the test bench including top-level, read and write modules

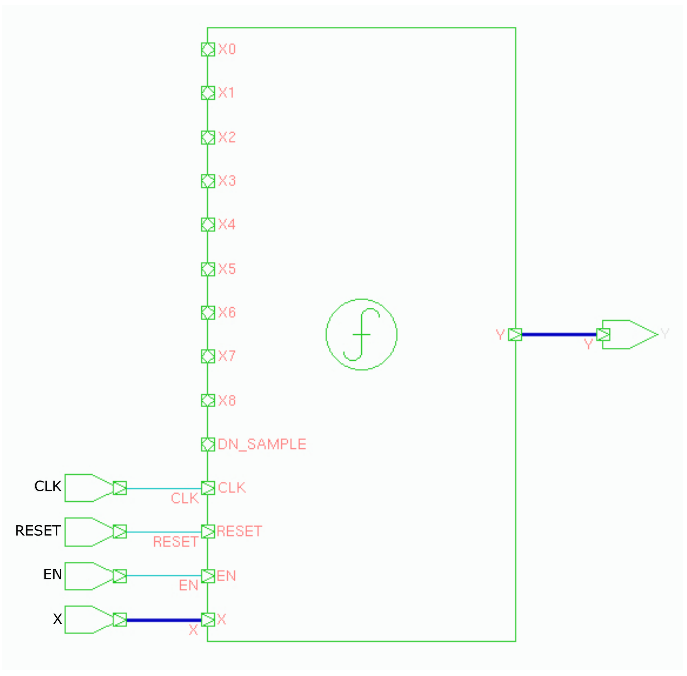


Figure 7.11: VCS schematic of a RNS-based filter of the image processor

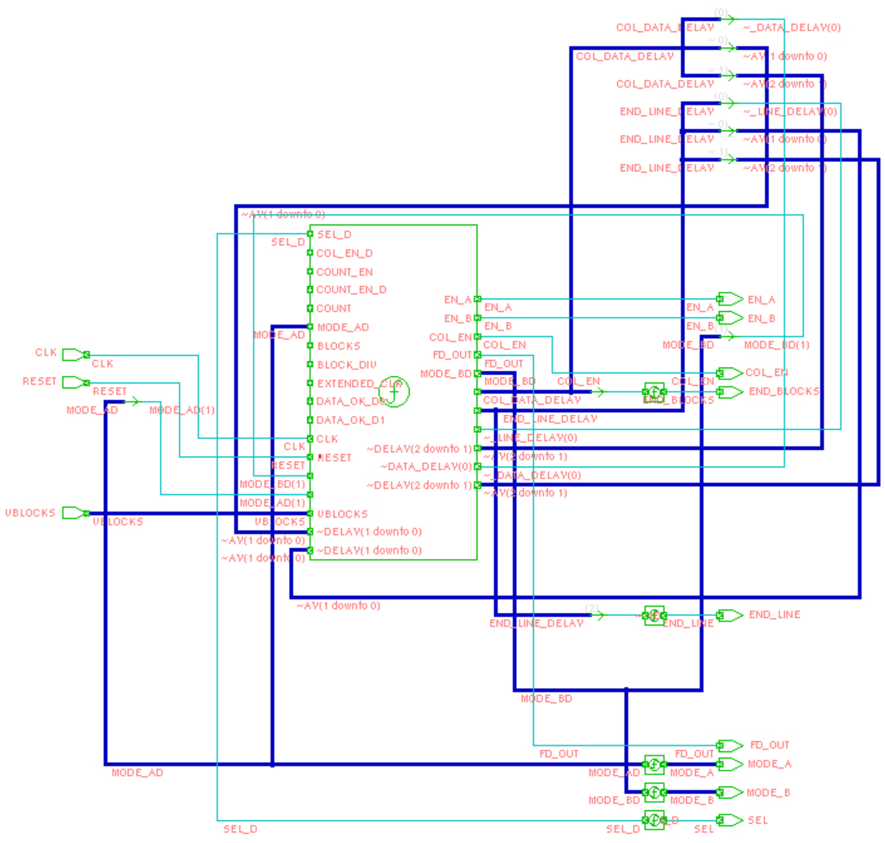


Figure 7.12: VCS schematic of the main controller of the image processor

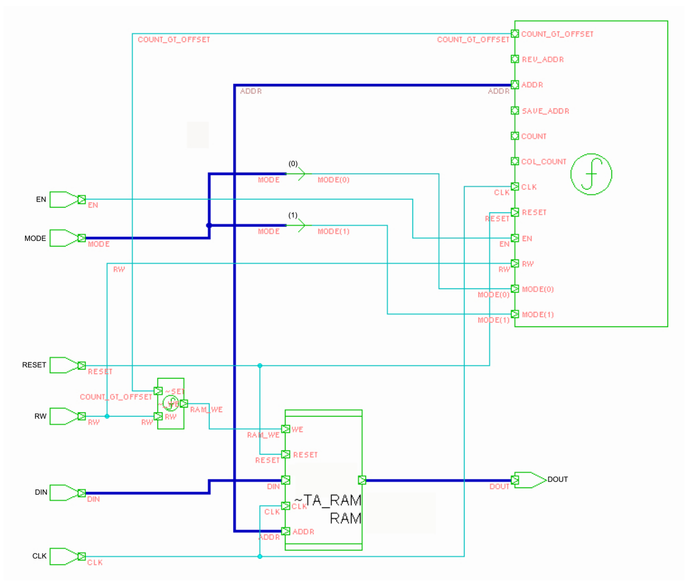


Figure 7.13: VCS schematic of a transposition unit of the image processor

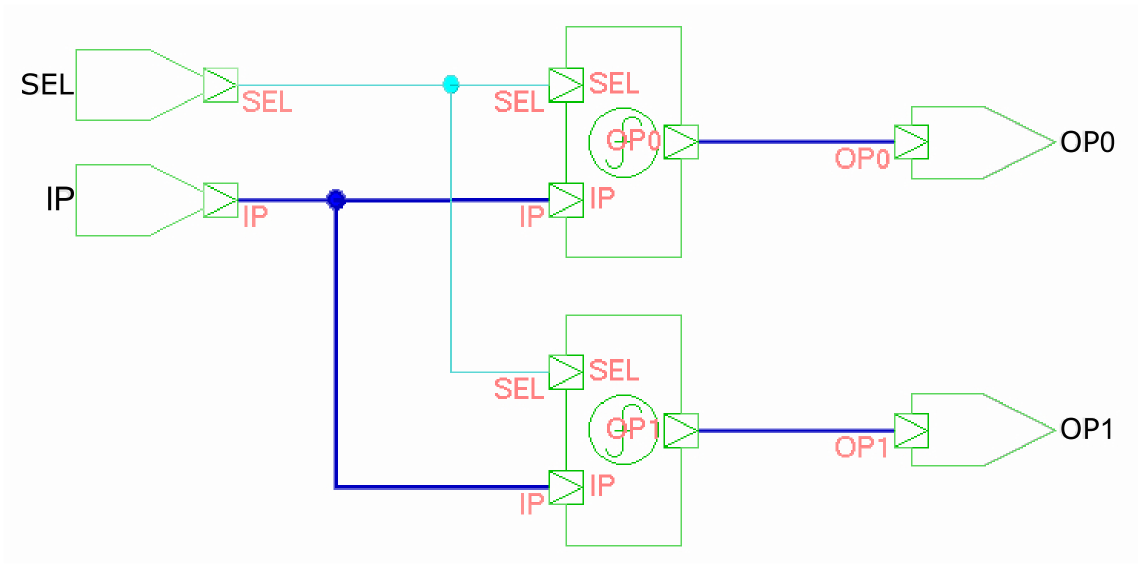


Figure 7.14: VCS schematic of demux module of the image processor

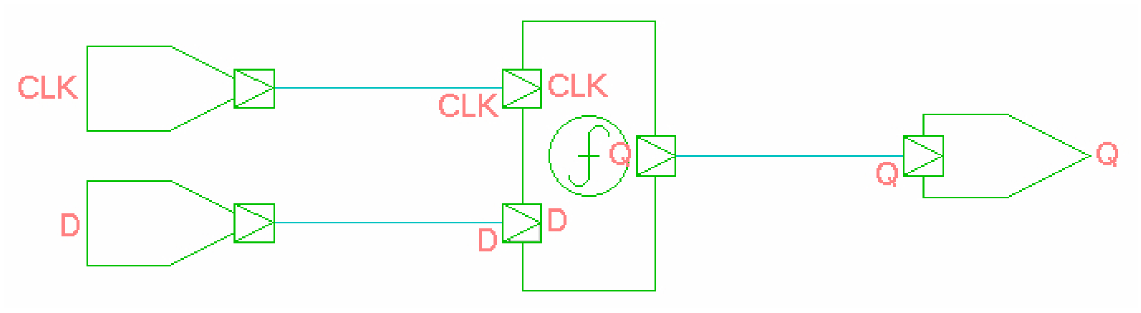


Figure 7.15: VCS schematic of a flip flop of the image processor

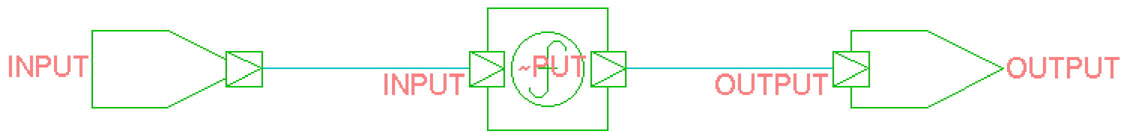


Figure 7.16: VCS schematic of a invert module the image processor

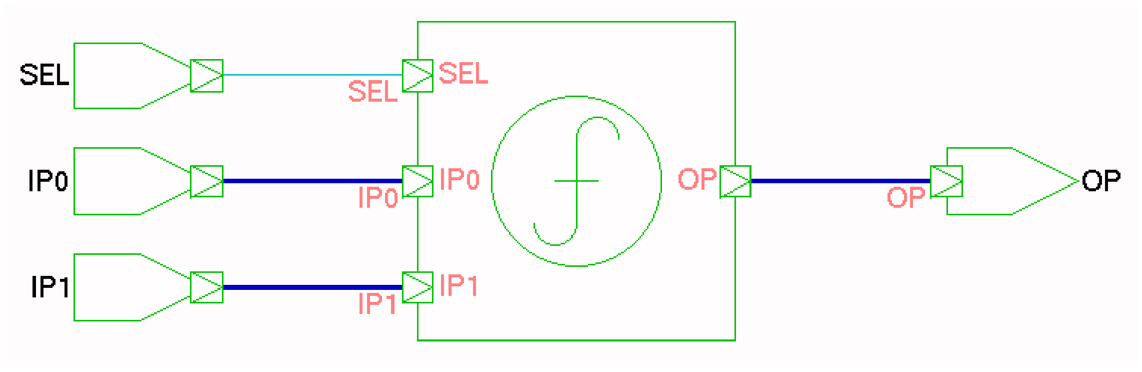


FIGURE 7.17: VCS schematic of mux module of the image processor

- Simulation

  Lastly, the simulation opens a GUI window to view the waveforms. Simulation uses the binary executable generated in the elaboration step. This step has also two modes of simulation: “optimised (batch)” or “debug (interactive)”. The debug mode is used in the initial phase of the design cycle, whereas the batch mode is used when most of the design issues are resolved.

```
## Analyze
   vhdlan -full64 /home/.../src/(top-level and all subdesigns).vhd

## Doing common elaboration using Chronologic VCS(TM)Version G-2012.09_Full64
   vcs -full64 RNS2D_tb -debug_all
```

After elaborating the design, `final.vpd` is generated and converted to the `final.vcd`. The VCD (Values Change Dump) file contains the switching activity information. This file is used for the multi-voltage power analysis in next section. Synopsys VCS-MX Tcl scripts for simulation of the proposed RNS-based image processor are provided in Appendix E.

```
## Generating VCD file using Synopsys VCS MX Compiled Simulator(simv) Version G-2012.09
   ./simv -ucli
   ucli\% dump -file final.vpd -type vpd
   VPD0
   ucli\% dump -add FILTER2D_FILTER2D_SCH_TB/uut/* -fid VPD0
   ucli\% run 500 us
   ucli\% dump -close
   ucli\% exit
```

```
## Generating VCD using VCD+ to VCD Translator G-2012.09_Full64
   ls *.vpd
   common.vpd  final.vpd  inter.vpd  testbench.vpd
   vpd2vcd final.vpd final.vcd
```

7.7 Multi-Voltage Power Analysis Using PrimeTime PX and UPF Flow

The power-performance of the proposed RNS-based image processor is optimised using the multi-voltage technique. It contains power domains with specific power behaviour. The extenders are in the low-power domain, which should be considered when analysing the power consumption of the whole system. The multi-voltage power analysis needs comprehensive information to accurately report the power consumption. Using internal and leakage power tables, an accurate power consumption can be calculated [162].

The power analysis of the proposed image processor is performed using the IEEE 1801 (UPF) standard. The Prime Time PX (PTPX) is a power analysis tool to perform timing and power analysis. An advantage of using PTPX is that it generates power consumption reports at the cell, block and chip level based on the switching activity, net capacitance and circuit connectivity. It also calculates the power consumption for each activity and generates average and peak power reports. Two power-analysis methodologies are available [162]:

- **Averaged power analysis** is based on the default switching activity of the circuit or the switching activity derived from RTL or gate level simulation, or user-defined switching.

- **Time-based power analysis** is the most accurate power analysis, based on the RTL or gate level simulation with respect to time.

The time-based power analysis is selected for the most accurate results. The link library is set to `saed90nm_max_hth` and `saed90nm_max`. These are the libraries previously used for mapping the design. Since the proposed design uses two different voltage levels, two link libraries are used. Figure 7.18 shows the files generated by Synopsys tools to be used for power analysis.

The proposed design is optimised using the multi-voltage technique, and compiled in DC topographical mode; hence, the multi-voltage power analysis is done using the IEEE standard using the UPF file. The MAPPED design, Milkyway library and TLU+

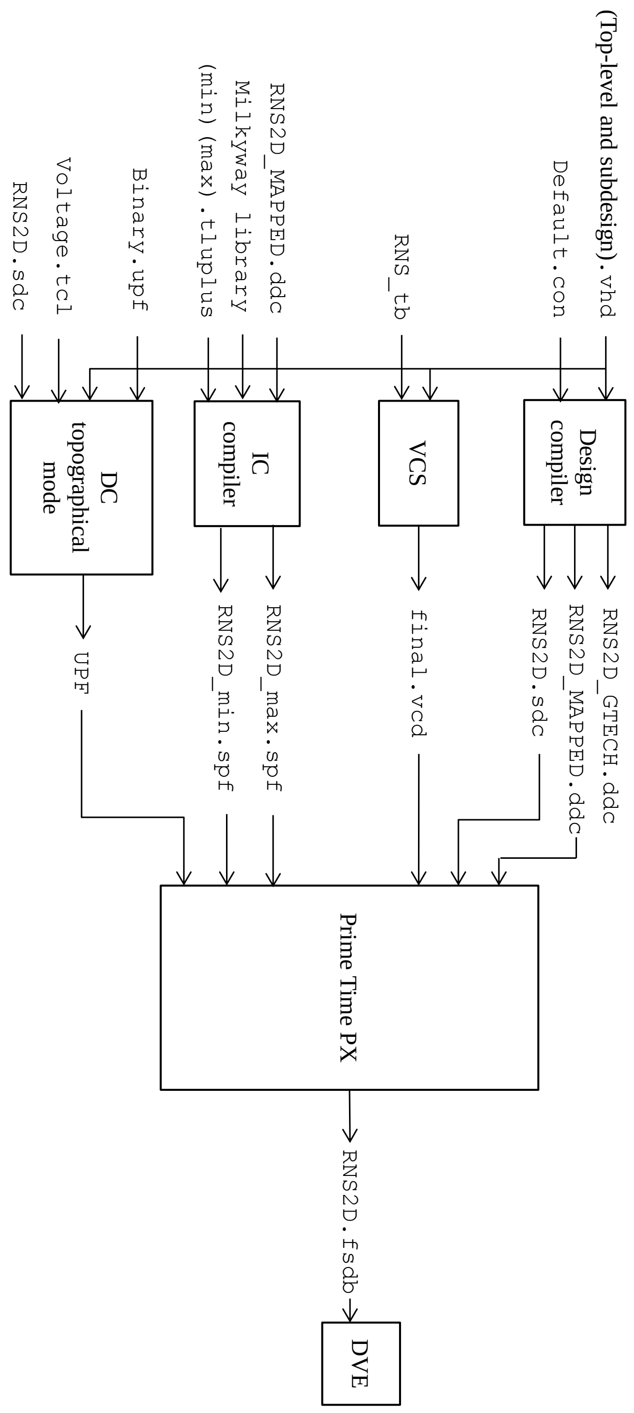


Figure 7.18: Files generated by Synopsys tools to be used for power analysis

Table 7.18: List of cells in the proposed processor

Cell	Reference	Area	Attributes
XLXI_1	control_logic	2228.43	hierarchical
XLXI_4	mux	338.23	hierarchical
XLXI_5	demux	442.37	hierarchical
XLXI_6	extender_0	5359468.00	hierarchical
XLXI_7	extender_1	5359473.50	hierarchical
XLXI_8	flip_flop	24.88	hierarchical
XLXI_9	invert_0	5.53	hierarchical
XLXI_10	invert_1	5.53	hierarchical
XLXI_43	cdf97_0	26398.16	hierarchical
XLXI_44	cdf97_1	26458.98	hierarchical
XLXI_70	scale	0.00	hierarchical
Total	12 cells	10774844.00	

files are used in the IC compiler to generate parasitic exchange formats as `.max.spf` and `.min.spf`. The event file which contains the switching activity is generated using VCS. Table 7.18 shows the cells of the design, the references used to resolve them, the area of each cell, and the cell attributes.

The three power domains defined for the design (TOP, CELL, EXTENDER) are shown in Table 7.19.

The four supply nets defined on design (VDD, VSS, VDDGS, VDDXS) are shown in Table 7.20.

The three supply sets defined for design (TOP.primary, CELL.primary, EXTENDER.primary) are shown in Table 7.21.

When all the input files are ready, PT-PX analyses the power. The power analyser is set, and the power analysis is time-based.

```
set power_enable_analysis true
set power_analysis_mode time_based
```

`RNS2D_MAPPED.ddc` is linked to the references, and the necessary files are read before update timing.

```
source /home/.../scripts/binary.upf
source /home/.../scripts/voltage.tcl
```

Table 7.19: Power domains of the multi-voltage processor

Power Domain	TOP	CELL	EXTENDER
Connections	Primary	Primary	Primary
Power	VDD	VDDXS	VDDGS
Ground	VSS	VSS	VSS
Supply sets	TOP.primary	CELL.primary	EXTENDER.primary
default_retention	TOP.default_retention	CELL.default_retention	EXTENDER.default_retention
default_isolation	TOP.default_isolation	CELL.default_isolation	EXTENDER.default_isolation
Elements	top level	XLXI_1	XLXI_6
		XLXI_4	XLXI_7
		XLXI_5	
		XLXI_8	
		XLXI_9	
		XLXI_10	
		XLXI_43	
		XLXI_44	
		XLXI_70	

Table 7.20: Supply nets of the multi-voltage processor

Supply Net	VDD	VSS	VDDGS	VDDXS
Scope	top level	top level	top level	top level
Power Domains	TOP EXTENDER CELL	TOP	TOP EXTENDER	TOP CELL
Supply Ports	VDD	VSS	VDDGS	VDDXS
PG_power_pin	N/A	N/A	N/A	N/A
Max-delay Voltage	1.2	0.0	0.7	1.2
Min-delay Voltage	1.2	0.0	0.7	1.2

Table 7.21: Supply sets of the multi-voltage processor

Supply set	TOP.primary	CELL.primary	EXTENDER.primary
Scope	top level	top level	top level
Power supply net association	VDD	VDDXS	VDDGS
Ground supply net association	VSS	VSS	VSS

```
read_parasitics /home/.../db/RNS2D.spf.min
read_parasitics /home/.../db/RNS2D.spf.max
update_timing
```

`final.vcd` is set, and the power is updated.

```
read_vcd -strip_path RNS2D_TB/UUT /home/.../db/vcd/final.vcd
update_power
```

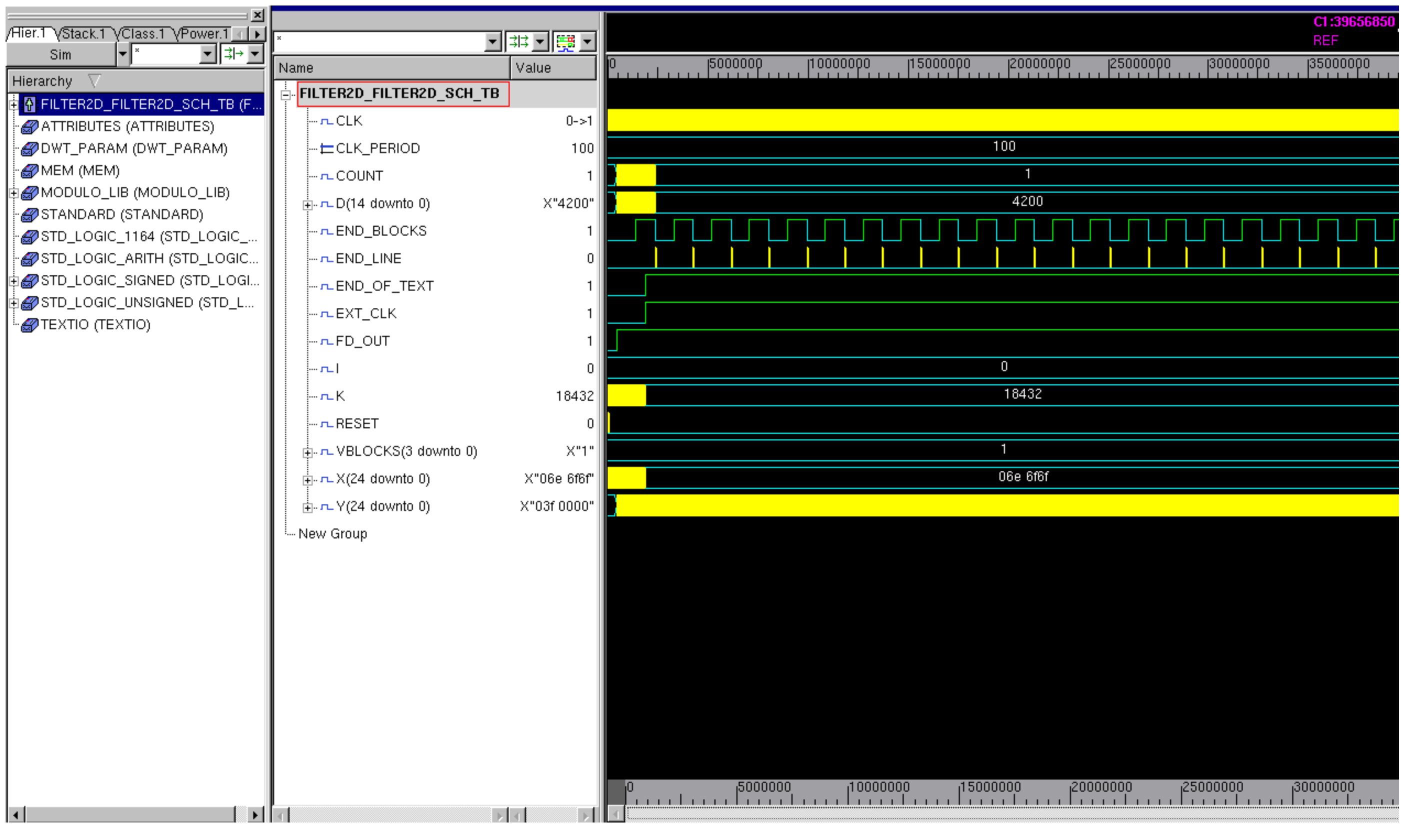
This step generates a series of power reports and `.fsdb` (Fast Signal Data Base), which is the waveform file. The waveform file can be viewed in graphical waveform viewing software Custom WaveView. The over-all and zoomed waveforms of the proposed image processor are shown in Figures 7.19 and 7.20, respectively. Synopsys PrimeTime PX Tcl scripts for multi-voltage power analysis of the proposed RNS-based image processor are provided in Appendix F. Synopsys DC report of post compile UPF of the proposed RNS-based image processor is provided in Appendix G.

7.8 Chapter Summary

The use of specialised number systems like RNS arithmetic is very efficient at improving the speed of computationally intensive processors, however this approach adds new concerns for lowering the total power consumption. In this chapter, the proposed two-dimensional RNS-based digital image processor with SVS implementation is implemented. The proposed design is synthesised using high-end Synopsys tools, and the performance improvement of the system is proven. Comparing the synthesis results of the proposed design with the 2's-complement contender, it is confirmed that, with a well-designed system including an appropriate arithmetic level and a well-established low-power method, the existing processors can be optimised to achieve higher performance, less hardware complexity and lower power dissipation.

Publication pertaining to this chapter:

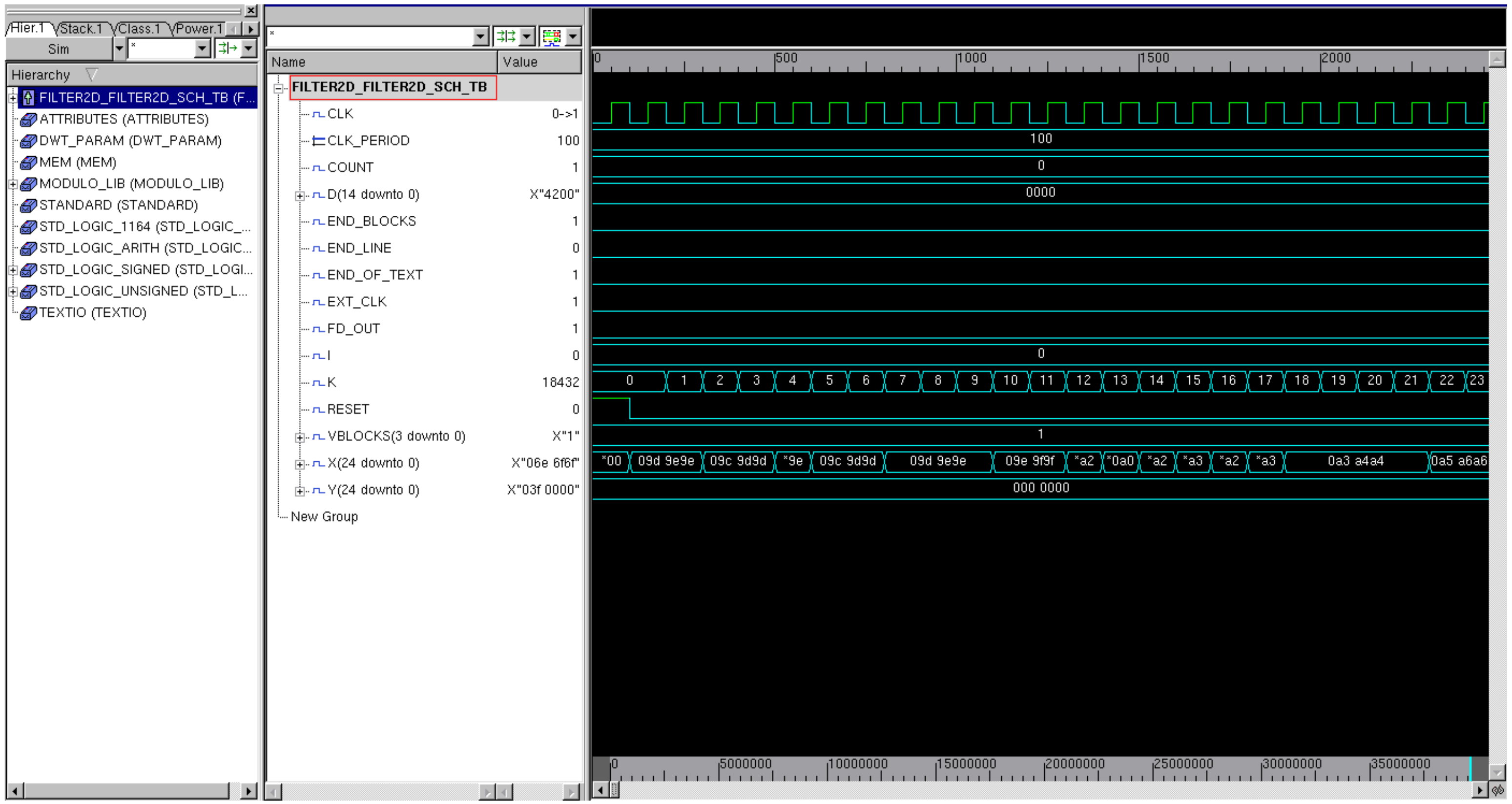
- Azadeh Safari, and Yinan Kong. *Power-Performance Enhancement of RNS-Based DWT Image Processor Using Multiple Voltage Domains*, In progress.



FILTER2D_FILTER2D_SCH_TB (F...
ATTRIBUTES (ATTRIBUTES)
DWT_PARAM (DWT_PARAM)
MEM (MEM)
MODULO_LIB (MODULO_LIB)
STANDARD (STANDARD)
STD_LOGIC_1164 (STD_LOGIC_...
STD_LOGIC_ARITH (STD_LOGIC...
STD_LOGIC_SIGNED (STD_LOGI...
STD_LOGIC_UNSIGNED (STD_L...
TEXTIO (TEXTIO)

Name	Value
FILTER2D_FILTER2D_SCH_TB	
CLK	0->1
CLK_PERIOD	100
COUNT	1
D(14 downto 0)	X"4200"
END_BLOCKS	1
END_LINE	0
END_OF_TEXT	1
EXT_CLK	1
FD_OUT	1
I	0
K	18432
RESET	0
VBLOCKS(3 downto 0)	X"1"
X(24 downto 0)	X"06e 6f6f"
Y(24 downto 0)	X"03f 0000"
New Group	

FIGURE 7.19: Overall view of the proposed RNS-based image processor waveforms



```
Hierarchy
FILTER2D_FILTER2D_SCH_TB (F...
ATTRIBUTES (ATTRIBUTES)
DWT_PARAM (DWT_PARAM)
MEM (MEM)
MODULO_LIB (MODULO_LIB)
STANDARD (STANDARD)
STD_LOGIC_1164 (STD_LOGIC_...
STD_LOGIC_ARITH (STD_LOGIC...
STD_LOGIC_SIGNED (STD_LOGI...
STD_LOGIC_UNSIGNED (STD_L...
TEXTIO (TEXTIO)
```

```
Name                      Value
FILTER2D_FILTER2D_SCH_TB
  CLK                     0->1
  CLK_PERIOD              100
  COUNT                   1
  D(14 downto 0)          X"4200"
  END_BLOCKS              1
  END_LINE                0
  END_OF_TEXT             1
  EXT_CLK                 1
  FD_OUT                  1
  I                       0
  K                       18432
  RESET                   0
  VBLOCKS(3 downto 0)     X"1"
  X(24 downto 0)          X"06e 6f6f"
  Y(24 downto 0)          X"03f 0000"
New Group
```

Figure 7.20: Zoomed view of the proposed RNS-based image processor waveforms

8 Physical Implementation Using Design Compiler Topographical Technology in ASIC Methodology

As the last step of the project, the proposed image processor is implemented using DC topographical technology in ASIC methodology. DC topographical technology is selected because it eliminates design iterations and reduces the overall design cycle. Milkyway technology and IC compiler are used for this purpose.

In this stage, some restrictions have been imposed on using the back-end view of libraries. The back-end view of libraries is not available to universities and non-commercial organisations. Synopsys provides an educational design kit ($SAED\_EDK90$) for educational and research purposes. This library is not designed for fabrication and does not contain foundary information due to intellectual property (IP) restrictions imposed by IC manufacturing foundaries. Having this restriction, the `saed90nm_typ_ht` library is used for physical implementation.

8.1 Physical Synthesis Using IC Compiler

To physically implement the proposed image processor, the set of files shown in Figure 8.1 has been generated during design synthesis in Chapter 7. `RNS2D_MAPPED.ddc` contains the netlist of the mapped design to the target library. The `RNS2D.sdc` file specifies the design constraints. `RNS2D.script` and UPF describe the power format and design development strategies used during synthesis. The logic and libraries are required for performing physical optimisations and resolving hierarchical references. The Milkyway reference libraries `RNS2D.mw` and technology files `saed90nm_icc_1p9m.tf` are also needed for physical library information. The TLUPlus (TLU+) files provide more accurate resistance coefficient (RC) extraction results. The TLU+ files store the RC coefficients in a binary table format and provide the effect of width, space, density and temperature on RC. The physical synthesis flow using the above mentioned file formats is followed to synthesise the proposed processor physically.

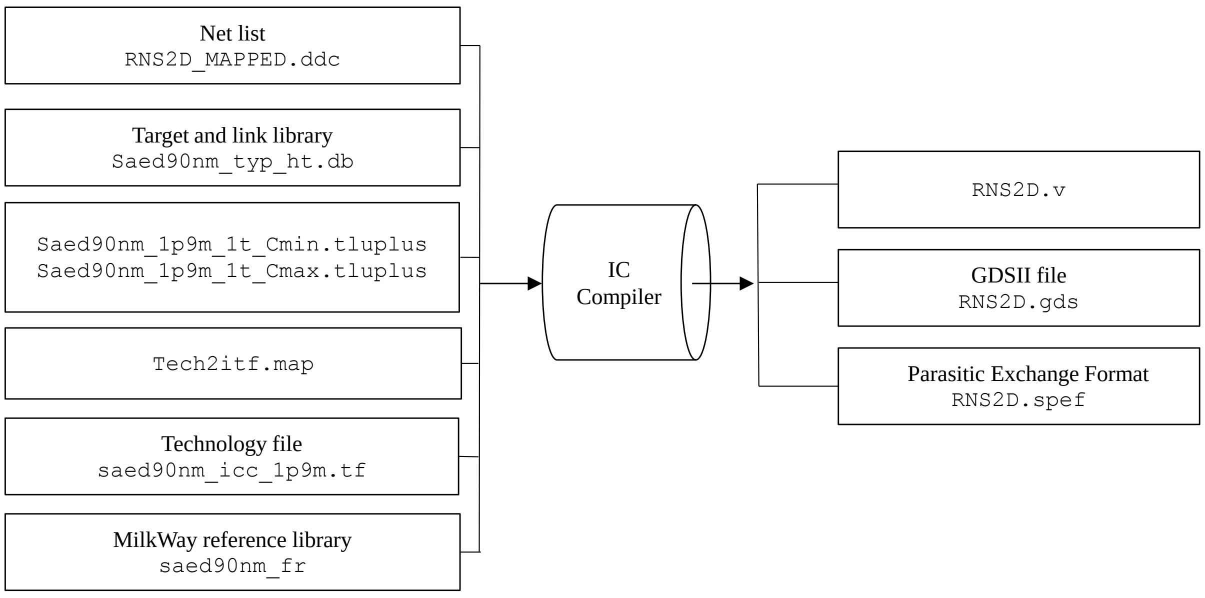


Figure 8.1: Input and output of IC compiler

8.2 Physical Synthesis Flow

As discussed in Chapter 7, using LPD techniques imposes some modifications on the digital standard-cell libraries as well as the synthesis flow. It also imposes some modifications in the physical synthesis flow, as shown in Figure 8.2. The IC compiler is set up with the target and link library `saed90nm_typ_ht.db`, and the working directory. Synopsys IC Compiler Tcl scripts for physical implementation of the proposed RNS-based image processor are Provided in Appendix H.

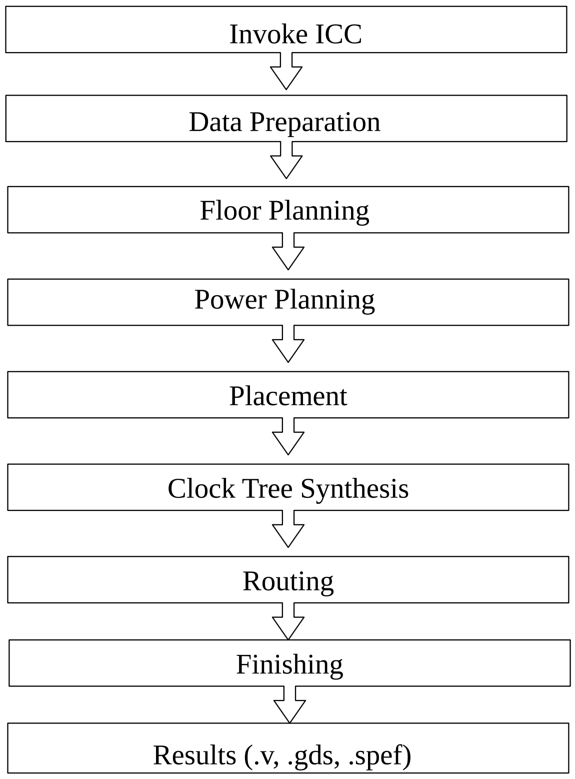


Figure 8.2: Physical synthesis flow used to implement the RNS-based processor

8.2.1 Library Data Preparation

Library data preparation is the first step for physical implementation. A new Milkyway library is created using the technology file `saed90nm_icc_1p9m.tf` and the reference library `saed90nm_fr`. The new Milkyway library is called `RNS2D`. The time, resistance and capacitance units in the created Milkyway library are:

Time Unit from Milkyway design library: *ns*

Resistance Unit from Milkyway design library: *kΩ*

Capacitance Unit from Milkyway design library: *pF*

The Milkyway technology file `.tf` contains information about the metal layers and vias required for place and route tools in the IC Compiler. The TLU+ files are used for RC estimation and provide information about all layers including active, poly, metal layers, etc. To accurately extract the RC results, there are TLU+ files available for each library. The TLU+ files are binary tables containing RC coefficients, and are important in the way that they specify the effect of width, space, density and temperature on RC.

The Milkyway technology files may also contain parasitic models of wires (as in the TLU+ files). If the TLU+ files are specified, the IC Compiler uses TLU+ files for RC estimations. Otherwise, the IC Compiler reads the parasitic information from the technology file.

It should be noted that using the TLU+ files gives more accurate RC estimations. Another advantage of using the TLU+ file is that different TLU+ files can be used for different RC corners (not PVT corners), while a technology file contains parasitic information for only one RC corner. Furthermore, different TLU+ files can be specified for different scenarios. The Max_TLU+, Min_TLU+ and the layer-mapping file between technology library and ITF file are shown in Table 8.1.

Table 8.1: The Max_TLU+, Min_TLU+, and the layer-mapping file between technology library and ITF file

Max_TLU+	`Saed90nm_1p9m_1t_Cmax.tluplus`
Min_TLU+	`Saed90nm_1p9m_1t_Cmin.tluplus`
Layer-mapping file	`Tech2itf.map`

8.2.2 Floorplanning

After setting the TLU+ files, the netlist of the mapped design `RNS2D_MAPPED.ddc` is imported. This design is mapped, compiled and saved using the same target library as when creating the Milkyway library. The GTECH design cannot be used for floorplanning because for un-mapped designs there is no Milkyway library.

The `RNS2D_MAPPED.ddc` is compiled using the default constraints `default.con`. Thus it is necessary to import `RNS2D.sdc` as well. Table 8.2 shows the planner summary of RNS-based image processor implementation.

Table 8.2: Planner summary of RNS-based image processor implementation

Row Direction	HORIZONTAL
Control Parameter	Aspect Ratio
Core Utilisation	0.350
Number Of Rows	1924
Core Width	5543.68
Core Height	5541.12
Aspect Ratio	1.000
Double Back	ON
Flip First Row	YES
Start From First Row	YES
Planner run	Successful

Table 8.3: The power and ground nets and pins setting of RNS-based image processor implementation

set power	VDD
set ground	VSS
set power port	VDD
set ground port	VSS
set `mw_logic0_net`	VSS
set `mw_logic1_net`	VDD

8.2.3 Placement

The power and ground rings are then added to the design in the PreRoute step. Before adding these rectangular rings, there are two more steps to accomplish: setting the power and ground ports, and defining the power and ground nets and pins. Table 8.3 shows the power and ground nets and pins of the RNS-based image processor implementation.

To create the rectangular rings, both VDD and VSS nets are set with a 0.2 offset (offsets applied after auto adjustments). The left- and right-side widths are 0.16, and the bottom- and top-side widths are 0.14 (around the core). Creating rectangular rings automatically connects the straps to the closest power and ground ring at or beyond both ends of the straps. Wide straps have an advantage over thin straps since they improve the quality of the placement and reduce the placement runtime. Table 8.4

Table 8.4: Creating rectangular rings for VDD and VSS nets of the RNS-based image processor

	Offset	Segment layer
Right	0.2	`M4`
Left	0.2	`M4`
Bottom	0.2	`M3`
Top	0.2	`M3`

shows the offset and segment layers of the rectangular rings.

Table 8.5: Place optimisation settings used in placement step of RNS-based image processor implementation

place_opt effort level	High
Congestion removal	Yes
Area recovery	No
Optimise dft	No
Clock tree synthesis	No
Optimise power	Yes
Optimise power mode	leakage

Core placement and optimisation have the most run-time for power optimisation in the physical implementation steps. The power optimisation is set to minimum congestion and high-effort compilation. During this step, standard cells are placed in horizontal placement rows. The design is then routed and optimised for power consumption. Table 8.5 shows the place optimisation settings. The chip summary report and legalise displacement report are provided in Tables 8.6 and 8.7, respectively.

8.2.4 Clock Tree Synthesis

For the core Clock Tree Synthesis (CTS) and optimisation, clock trees are built based on the clock-tree design rule constraints. It balances the loads and minimises the clock skew. The placements of the clock sinks are fixed, incremental logic and placement optimisation are performed, and the placement of both the buffers and registers on the clock tree are fixed. The clock_opt command performs clock-tree synthesis, routing of

Table 8.6: RNS-based image processor chip summary report

Std cell utilisation	34.70% (11565605/(33331376-0)) (Non-fixed + Fixed)
Chip area	33331376 sites, bbox (1.00 1.00 5544.68 5542.12)μm
Std cell area	11565605 sites, (non-fixed:11565605 fixed:0)
Cells	654321 (non-fixed:654321 fixed:0)
Macro cell area	0 sites 0 cells
Placement blockages	0 sites
Routing blockages	0 sites
Partial p/g net blockages	0 sites
Lib cell count	64
Avg. std cell width	4.88 μm
Site array	unit (width: 0.32 μm, height: 2.88 μm, rows: 1924)
Physical DB scale	1000 db_unit = 1 μm

Table 8.7: Legalise displacement of the chip

Avg cell displacement	0.760 μm (0.26 row height)
Max cell displacement	3.853 μm (1.34 row height)
Std deviation	0.574 μm (0.20 row height)
Number of cells moved	416 (out of 654321)

clock nets, extraction, optimisation, and optionally hold-time violation fixing on the current design. Table 8.8 shows the clock-tree summary.

TABLE 8.8: Clock tree summary

Timing Path Group	*clk*
Levels of Logic	62.00
Critical Path Length	37.14 (*ns*)
Critical Path Slack	62.26 (*ns*)
Critical Path Clk Period	100.00 (*ns*)
Total Negative Slack	0.00
No. of Violating Paths	0.00
Worst Hold Violation	0.00
Total Hold Violation	0.00
No. of Hold Violations	0.00
Cell count:	
Hierarchical Cell Count	126
Hierarchical Port Count	3571
Leaf Cell Count	654321
Buf/Inv Cell Count	36599
CT Buf/Inv Cell Count	0
Combinational Cell Count	423246
Sequential Cell Count	231075
Macro Count	0
Area (μm):	
Combinational Area	4904381.892862
Non-combinational Area	5754479.545292
Buf/Inv Area	418664.448485
Net Area	6352950.951578
Net XLength	25329066.00
Net YLength	23133108.00
Cell Area	10658861.438154
Design Area	17011812.389732
Net Length	48462176.00
Design rules:	
Total Number of Nets	879437
Nets With Violations	12
Max Trans Violations	9
Max Cap Violations	3

Each standard cell has power and ground pins which are connected to the straps and rings. The power and ground (VDD and VSS) rails in standard cells are connected. The standard cells are pre-routed before performing global routing. The priority is to ensure that the global router recognises these routing obstructions.

8.2.5 Routing

Finally, for routing the design, the core post-route and optimisation are executed on the design. This step performs routing for the broken nets. The empty spaces in the standard-cell rows are filled with instances of master filler cells in the library. Table 8.9 shows the route optimisation strategy in our design. The technology table contains 9 routable metal layers; this is considered as a 9-metal-layer design. Table 8.10 shows the filler-cell insertion.

Table 8.9: Route optimisation strategy for the design

Stage	Auto
Effort	Medium
Power mode	None
Search-Repair loops	10
ECO Search-Repair loops	4
Fix Hold Mode	Route_based
Route Violation threshold	3000

Table 8.10: Filler cell insertion

Placeable cells	654328
Cover cells	0
IO cells/pins	61
Cell instances	654389
Net pin threshold	33
Pre-routes for placement blockage/checking	120
Pre-routes for map congestion calculation	44393
Auto Set : first cut	Vertical
Design style	Horizontal

8.2.6 Design Verification

To verify the physical design, the design rules are checked using DRC (Design Rule Checking) and LVS (layout-versus-schematic). Details of design verification check are as follow.

```
-- LVS START : --
Total area error in layer 0 is 0.  Elapsed =    0:00:13, CPU =    0:00:13
Total area error in layer 1 is 0.  Elapsed =    0:00:35, CPU =    0:00:35
Total area error in layer 2 is 0.  Elapsed =    0:00:53, CPU =    0:00:53
Total area error in layer 3 is 0.  Elapsed =    0:07:52, CPU =    0:07:52
Total area error in layer 4 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 5 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 6 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 7 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 8 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 9 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 10 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 11 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 12 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 13 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 14 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 15 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
```

8.2.7 Finishing and Saving the Results

The design is saved in the last step of physical synthesis. A VHDL netlist, a VHDL file without power and ground ports, and a SPEF (Standard Parasitic Exchange Format) file are generated. To write the design stream, the design data is written in a specified library in GDS format. These files can be used for modifications and further optimisations in the future. Figures 8.3 and 8.4 show the path-to-path schematic, and the schematic of the proposed 2D RNS-based image processor, respectively. Synopsys IC Compiler reports for physical implementation of the proposed RNS-based image processor are provided in Appendix I.

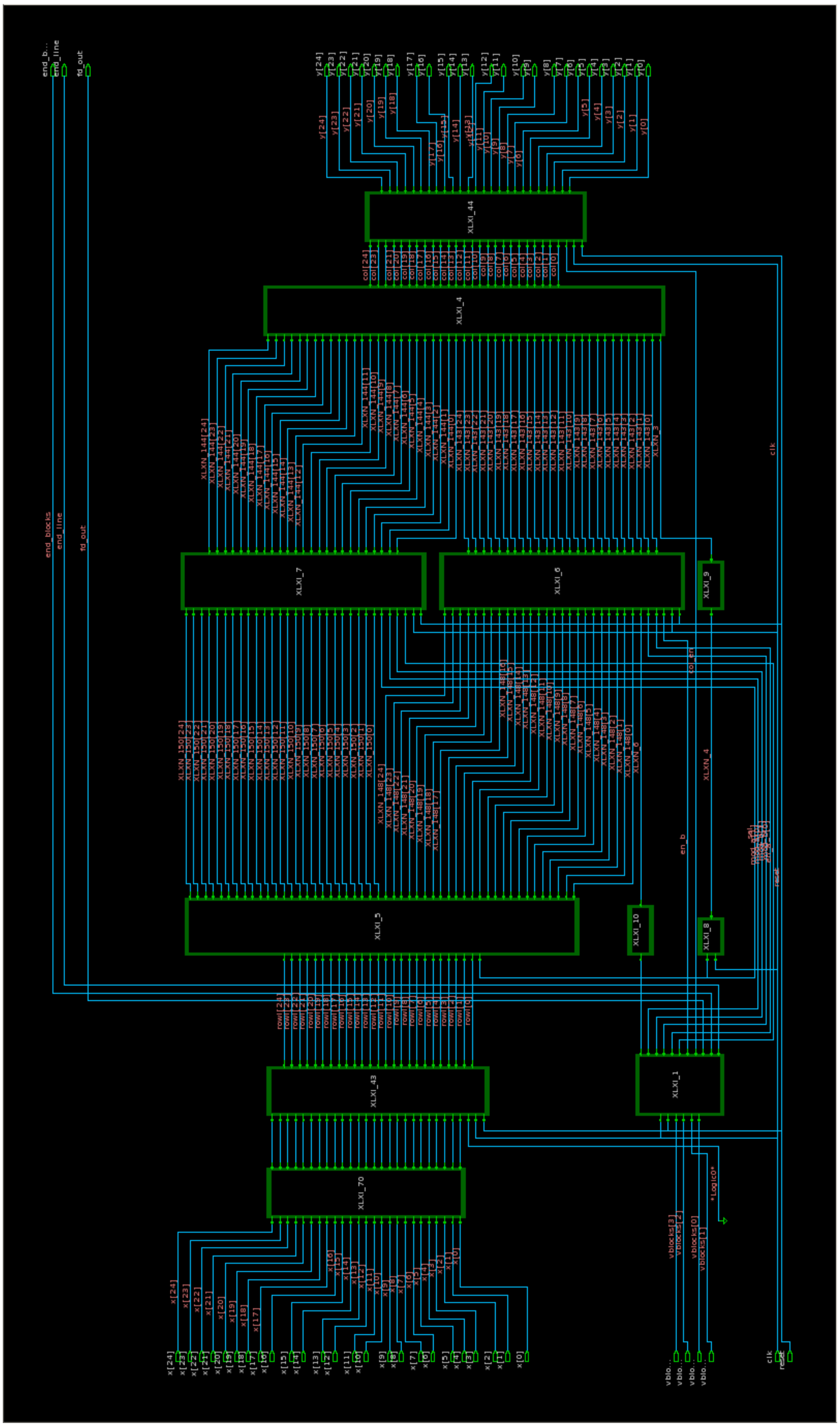


Figure 8.3: Path-to-path schematic of the proposed 2D RNS-based image processor

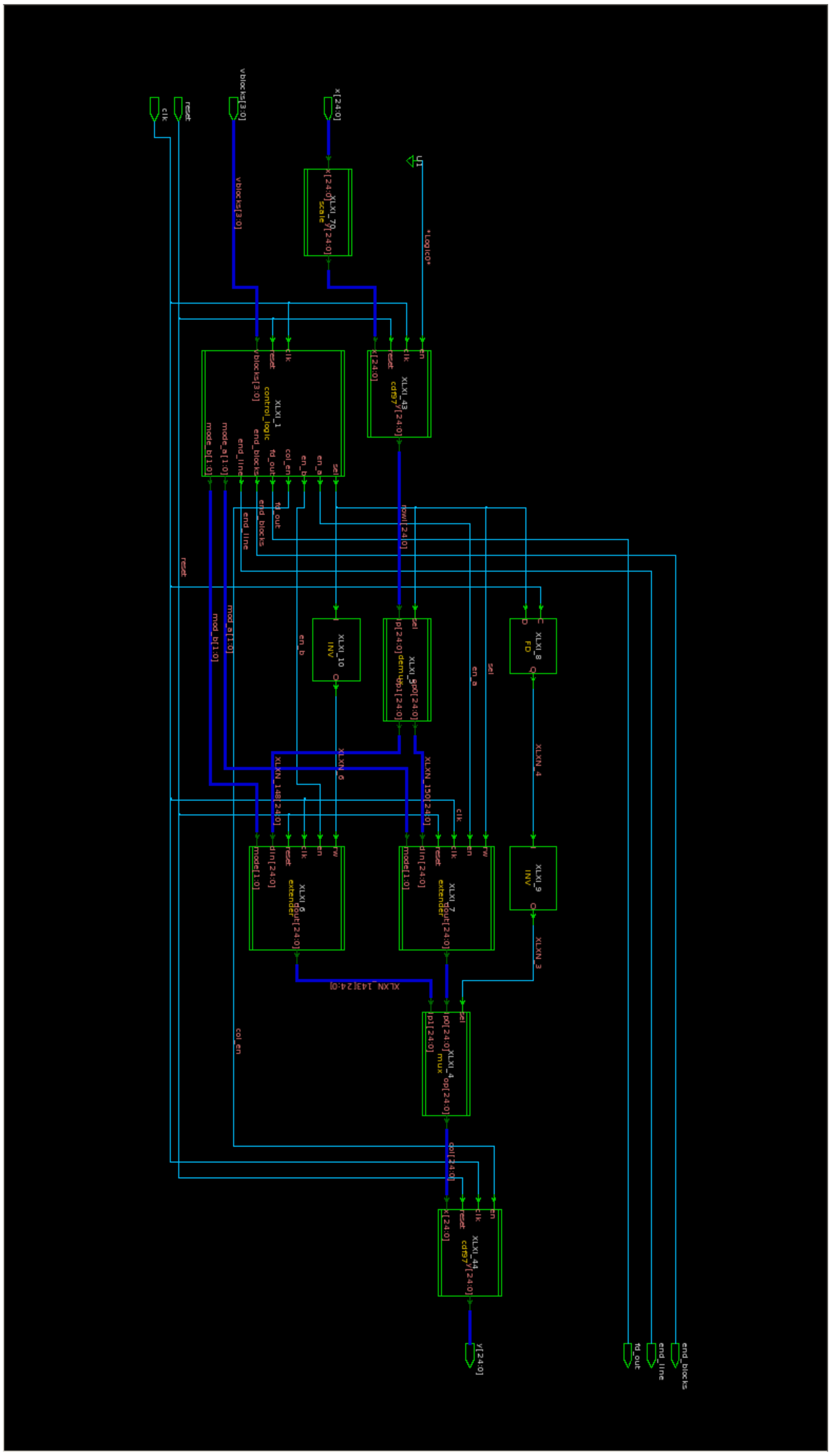


Figure 8.4: Schematic of the proposed 2D RNS-based image processor

8.3 Comparison of the Proposed RNS-Based Image Processor with Dedicated Hardware Designs

For comparison of the proposed RNS-based image processor with existing hardware designs, several 2D image-processor hardware implementations are considered. The majority of architectures have reported results based on an 8×8 block-size DCT or IDCT [151, 153, 163–173] except [151] that authors have reported the maximum frequency for 16×16 block sizes as well. The only paper presenting hardware implementation of a DWT image processor is in [174], which was able to process images without the need for fragmentation. They presented the hardware results of processing an 256×256 image using Haar wavelets. In this thesis, due to block artifacts and boundary distortions caused by DCT, the proposed image processor uses DWT; thus it is able to process 64×64 block sizes, which reduces the processing time and distortions significantly. It is able to process images of up to 1024×1024 pixels.

It is an inconsistent and incomplete comparison to compare processors that use different arithmetic operations, filter-bank features, power management techniques and technology process. Almost all of the architectures in literature are customised by designer preferences and contributions, which makes it hard to compare them. They differ in transform coefficient bits (accuracy), voltage, and technology process. They also vary in the methodology they have adopted for optimisations, like the cost-effective approach in [153] and [169], the low-power technique [170], high-throughput design [173], and low-complexity scheme [174]. The authors in [175] have proposed a methodology for evaluating area and delay scale when comparing designs with different technology processes. The authors have assumed a velocity saturated model, hence delay scales $1/S$. In Table 8.11, delay scale of each design is provided with regard to the proposed design in 90 nm.

If different technologies are taken into account, a trend of using smaller technologies in new designs is obvious. However, it does not conclude that new designs are necessarily smaller than previous architectures. It can even be seen that designs using similar transforms (IDCT) and technology (0.6 μm) have presented different core areas ([169] and [172]) which was due to different optimisation methodologies. [169] has developed the folding scheme to obtain a low gate count and high throughput, while [172] was a high-performance, low-cost design. The core area of our design is 30.28 mm^2 (5.54368 $mm \times$ 5.54112 mm), which is comparable to the efficient design in [172]. The area consumption of the presented approach is smaller than that of the low-power

Table 8.11: Comparison of the proposed RNS-based image processor with dedicated hardware designs

Design	Year	Transform	N	Technology	Delay Scale	Core area	Voltage	Frequency	Latency	Power	LPD
				(μm)	(mm^2)	(S)	(V)	(MHz)	(Cycles)	(mW)	
Totzek [151]	1990	DCT	8×8	1.5	16.67	91	-	45	-	3100	No
			16×16					22.5			
⋮	⋮	⋮									
Chang [170]	2000	DCT	8×8	0.6	6.67	50.6	2	100	198	138	Yes
Fanucci [171]	2002	DCT	8×8	0.18	2	-	1.6	33.6	227	8.7	Yes
Guo [172]	2003	IDCT	8×8	0.6	6.67	21	-	100	-	-	No
Gong [153]	2004	DCT	8×8	0.25		1.5	2.78−	150	1	-	No
Ruiz [173]	2005	DCT	8×8	0.35	3.89	3	3.3	300	178	-	No
Abhishek [176]	2006	DCT	104×128	0.5	5.55	2.99	3	$100KKz$	72	$80\mu W/frame$	Yes
Diaz [174]	2006	DWT	256×256	1.0	11.11	-	-	14	72	-	No
Nilchi[177]	2009	DWT	128×128	0.35	3.89	12.76	3.3	-	-	26.2	No
Shah[178]	2013	DWT	128×128	0.35	3.89	0.65	-	-	-	0.328	Yes
RNS processor	2014	DWT	64×64	0.09	1	30.28	1.08, 0.7	27	-	91	Yes

8×8 processor in [170]. The authors in [170] achieved low power consumption by simplifying the direct 2D algorithm and applying parallel distribute arithmetic. They used a 2 V power supply, and the TSMC 0.6 μm single-poly double-metal technology. The processor consumed 133 mW of power at 100 MHz and the maximum operation speed achieved was 133 MHz.

The design of a test chip for the computation of the 2D DCT/IDCT using 1.5 μm CMOS technology has been presented in [151]. It was tested under worst case conditions, and performed at 45 MHz for an 8×8 block size and 22.5 MHz for a block size of 16×16. Slawecki et al. [163] used cyclic convolution IDCT and ROM-based multiplications. Their design needed 1 W of power to process an 8×8 block-size image using 2 μm technology.

A 2D DCT core processor has been presented in 1996, in [166]. It was a 0.9 V, 150 MHz, 10 mW, 4 mm^2 core processor for a HDTV-resolution video compression and decompression core processor. The authors used a variable-threshold-voltage technique to reduce the active power dissipation, and a 0.3-μm CMOS triple-well, double-metal technology.

Rambaldi [168] proposed a 2D IDCT processor using LPD techniques to reduce power. The proposed design used 3.3 V and 1.1 V voltages, and was implemented using 0.5 μm CMOS technology. It used 200 K transistors and operated at 27 MHz. They achieved a low-power design: dissipates power as low as 67 mW.

Authors in [176] proposed an image processor architecture which was capable of performing modular and programmable matrix operations. They used 0.5μ N-well CMOS technology to produce a 100 kHz, 74 K transistor, and 80 $\mu W/frame$ $(VDD = 3 : 3V)$. A JPEG Encoder and DCT transform was used to process the image blocks of 104×128.

The "focal-plane algorithmically-multiplying CMOS computational image sensor" was proposed in [177]. It was used to compute 2D video frames of up to 8×8 pixels simultaneously.They used 0.35μm CMOS technology with 3.3 V supply voltage which used 26.2 mW for processing images using 8×8 kernel.

The performance characteristics of the physical implementation of the proposed RNS-based image processor are provided in Table 8.12.

Our chip is implemented using a Synopsys 90 nm Generic Library $(SAED90nmEDK)$ technology process, and compresses 64×64 block images at a frequency of 27 MHz using the CDF97 algorithm. Multi-voltage UPF flow is used which makes the chip a low-power RNS-based design. It operates with one clock-cycle latency (one pixel per

Table 8.12: Chip characteristics

Internal word-length	25 bits
Technology	Synopsys 90 nm Generic Library ($SAED90nmEDK$)
Core size	5.54368 $mm \times$ 5.54112 mm
Clock rate	27 MHz
Block size	64×64
Supply voltage	Multi-voltage 1.08, 0.7
Power	91 mW

clock cycle), and requires 30.28 mm^2 of core area including the transposition modules (RAM). The power consumption of the proposed compression chip is comparable with designs in the literature. The current processor can be used to design a compact, full-custom, complete image coder, not just the transform processor.

8.4 Chapter Summary

As the last step of the project, the proposed image processor is implemented using DC topographical technology in ASIC methodology. DC topographical technology is selected because it eliminates design iterations and reduces the overall design cycle. The main purpose of ASIC prototyping is to identify any bugs in the design. The synthesised design is ready for tape-out and final testing and verification.

9 Thesis Conclusion and Recommendations for Future Work

9.1 Thesis Conclusion

Using specialised number systems like RNS is a promising new paradigm in improving the speed of computationally intensive digital image processors. An optimised RNS-based system performs mathematical operations on independent modular channels simultaneously, and accomplishes high-throughput hardware architectures for real-time encoding, which can not be achieved using conventional number systems.

In this thesis, a two-dimensional RNS-based digital image processor with SVS implementation has been presented. The proposed design has been synthesised using high-end Synopsys tools, and the performance improvement of the system was proven.

Contributions are made in designing the multiplier-less filters and transposition module with embedded symmetric extension to avoid blurring or spatial dislocations. A strong relationship between the hardware utilisation and the power consumption indicated that, by using overlapped blocks and reducing the amount of hardware, memory usage as well as power consumption are saved.

The power consumption of the proposed processor has been refined for portable-device applications with high speed and superior functionality requirements such as cell phones, GPS devices, laptops and ultra mobile PCs that should be either always on or have intermittent usage.

High speed and more functionality of the proposed processor resulted in an increased power consumption, facing us with power-management issues. The situation became even worse when the design is implemented with a 90 nm technology process. The increase in the power consumption was mainly due to leakage power in memory based transposition modules (Extenders). The problem of power consumption was solved by employing a multi-voltage LPD technique.

Comparing the synthesis results of the proposed RNS-based image processor with initial binary processor, it was confirmed that, with a well-designed system including an appropriate arithmetic level and a well-established low-power method, the existing processors can be optimised to achieve higher performance, less hardware complexity and lower power dissipation. The current processor can be used to design a compact, full-custom, complete image coder, not just the transform processor.

Finally, a number of limitations and difficulties need to be considered. Synopsys tools are extremely complicated and the hurdle of setting up and learning the tools should not be ignored. In addition, some restrictions have been imposed for using the back-end view of libraries as they are not available to universities and non-commercial organisations. Hence, an educational design kit ($SAED\_EDK90$) is provided by Synopsys for educational and research purposes. This library is not designed for fabrication and does not contain foundary information due to intellectual property (IP) restrictions imposed by IC manufacturing foundaries.

9.2 Future Research Directions

This research has laid significant groundwork for further investigation in designing fast, simple and efficient digital image processors. Generally, RNS is very useful in designing an *Inner Product Step Processor* (IPSP). Real-time applications demanding new high-speed processors are limitless. The proposed image processor can be used in video processing.

A further contribution can be made by decomposing the low-cost moduli set into smaller "sub-moduli". It is a good idea to design multi-level RNS-based designs. In this suggestion, each modular channel can use sub-moduli itself. It would be very useful in RNS-based systems that require large dynamic range, and modular channels using a large number of bits (such as 1024-bit modular channels).

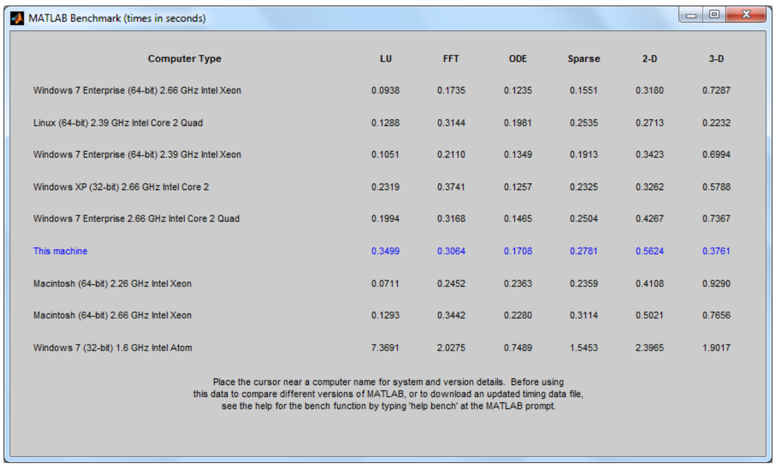
The issue of scaling is an intriguing one which can be usefully explored in further research. There are several ways to reduce the hardware complexity of individual components of scalers. One particular hardware reduction is to pipeline the $(2^n - 1)$ and $(2^n + 1)$ adders. This can be done by using a multiplexer to connect the End-Around Carry (EAC), rather than using two separate adders.

Another recommended future research is using the proposed image processor as a sub-processor (or co-processor) to implement further processing of images. One can use residue-to-binary converters to generate binary coefficients, and proceed with further steps such as quantisation and entropy coding.

A

Computer Specifications Used for Running MATLAB

A.1 Computer Benchmark for Running MATLAB



MATLAB Benchmark (times in seconds)

Computer Type	LU	FFT	ODE	Sparse	2-D	3-D
Windows 7 Enterprise (64-bit) 2.66 GHz Intel Xeon	0.0938	0.1735	0.1235	0.1551	0.3180	0.7287
Linux (64-bit) 2.39 GHz Intel Core 2 Quad	0.1288	0.3144	0.1981	0.2535	0.2713	0.2232
Windows 7 Enterprise (64-bit) 2.39 GHz Intel Xeon	0.1051	0.2110	0.1349	0.1913	0.3423	0.6994
Windows XP (32-bit) 2.66 GHz Intel Core 2	0.2319	0.3741	0.1257	0.2325	0.3262	0.5788
Windows 7 Enterprise 2.66 GHz Intel Core 2 Quad	0.1994	0.3168	0.1465	0.2504	0.4267	0.7367
This machine	0.3499	0.3064	0.1708	0.2781	0.5624	0.3761
Macintosh (64-bit) 2.26 GHz Intel Xeon	0.0711	0.2452	0.2363	0.2359	0.4108	0.9290
Macintosh (64-bit) 2.66 GHz Intel Xeon	0.1293	0.3442	0.2280	0.3114	0.5021	0.7656
Windows 7 (32-bit) 1.6 GHz Intel Atom	7.3691	2.0275	0.7489	1.5453	2.3965	1.9017

Place the cursor near a computer name for system and version details. Before using this data to compare different versions of MATLAB, or to download an updated timing data file, see the help for the bench function by typing 'help bench' at the MATLAB prompt.

Figure A.1: Computer benchmark used for running MATLAB

A.2 MATLAB Benchmark

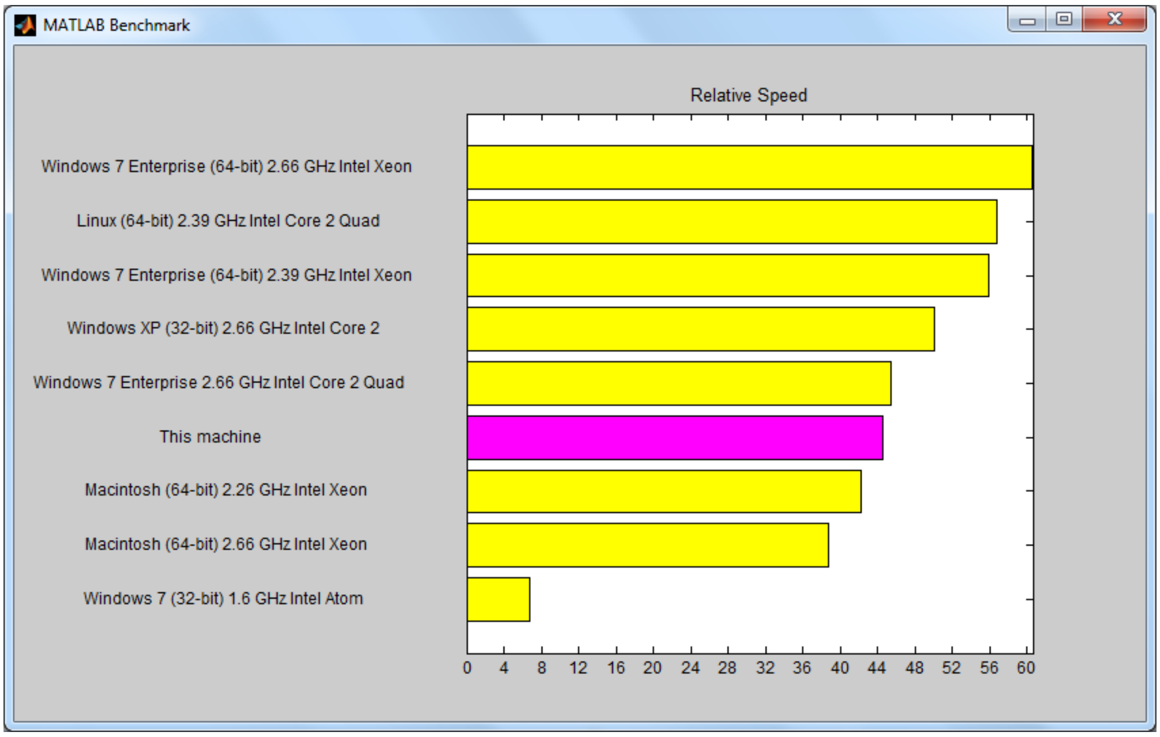


FIGURE A.2: MATLAB benchmark

B

Synopsys Design Compiler Synthesis Results of Full-Adder-Based Scaler

Table B.1: Synopsys DC synthesis results of Chang scaler-Scale 1

n	Area	Delay	Power
8	Combinational area: 123.500000	Data required time 8.05	Cell internal power = 92.1400 μW (14%)
	Buf/Inv area: 28.000000	Data arrival time -7.97	Net switching power = 555.8295 μW (86%)
	Net Interconnect area:18.067170	Slack (MET)	Total dynamic power = 647.9695 μW (100%)
	Total area:141.5671701		Cell leakage power= 20.4122 nW
16	Combinational area: 319.750000	Data required time 8.05	Cell internal power = 221.1405 μW (16%)
	Buf/Inv area: 72.000000	Data arrival time -7.92	Net switching power = 1.1231 mW (84%)
	Net Interconnect area:62.567820	Slack (MET)	Total dynamic power = 1.3442 mW (100%)
	Total area:382.3178201		Cell leakage power= 67.4990 nW
32	Combinational area: 992.000000	Data required time 8.05	Cell internal power = 520.8665 μW (18%)
	Buf/Inv area: 211.500000	Data arrival time -8.05	Net switching power = 2.3196 mW (82%)
	Net Interconnect area:231.396075	Slack (MET)	Total dynamic power = 2.8404 mW (100%)
	Total area:1223.3960751		Cell leakage power= 139.4617 nW
64	Combinational area: 1711.750000	Data required time 8.05	Cell internal power = 923.1697 μW (17%)
	Buf/Inv area: 437.750000	Data arrival time -8.05	Net switching power = 4.5279 mW (83%)
	Net Interconnect area:377.153214	Slack (MET)	Total dynamic power = 5.4510 mW (100%)
	Total area:2088.9032141		Cell leakage power= 289.1568 nW
128	Combinational area: 3581.250000	Data required time 8.05	Cell internal power = 1.9460 μW (18%)
	Buf/Inv area: 954.500000	Data arrival time -8.05	Net switching power = 9.1557 mW (82%)
	Net Interconnect area:747.609416	Slack (MET)	Total dynamic power = 11.1017 mW (100%)
	Total area:4328.8594161		Cell leakage power= 598.3668 nW

Table B.2: Synopsys DC synthesis results of Chang scaler-Scale 2

n	Area	Delay	Power
8	Combinational area: 349.750000 Buf/Inv area: 53.250000 Net Interconnect area:69.588500 Total area:419.3385001	Data required time 8.05 Data arrival time -8.03 Slack (MET)	Cell internal power = 228.7735 μW (26%) Net switching power = 641.7675 μW (86%) Total dynamic power = 870.5410 μW (100%) Cell leakage power= 58.9548 nW
16	Combinational area: 784.750000 Buf/Inv area: 123.500000 Net Interconnect area:153.570939 Total area:938.3209391	Data required time 8.05 Data arrival time -8.05 Slack (MET)	Cell internal power = 505.5667 μW (29%) Net switching power = 1.2525 mW (71%) Total dynamic power = 1.7581 mW (100%) Cell leakage power= 122.2448 nW
32	Combinational area: 2136.500000 Buf/Inv area: 384.000000 Net Interconnect area:451.580613 Total area:2588.0806131	Data required time 8.05 Data arrival time -8.03 Slack (MET)	Cell internal power = 1.3466 mW (34%) Net switching power = 2.6478 mW (66%) Total dynamic power = 3.9944 mW (100%) Cell leakage power= 362.7282 nW
64	Combinational area: 3986.500000 Buf/Inv area: 718.00000 Net Interconnect area:810.449539 Total area:4796.9495391	Data required time 8.05 Data arrival time -8.05 Slack (MET)	Cell internal power = 2.5088 mW (32%) Net switching power = 5.2551 mW (68%) Total dynamic power = 7.7639 mW (100%) Cell leakage power= 659.369 nW
128	Combinational area: 8256.000000 Buf/Inv area: 1744.500000 Net Interconnect area:1703.805073 Total area:9959.8050731	Data required time 8.05 Data arrival time -8.05 Slack (MET)	Cell internal power = 5.3354 mW (33%) Net switching power = 10.6002 mW (67%) Total dynamic power = 15.9356 mW (100%) Cell leakage power= 1.3880 nW

TABLE B.3: Synopsys DC synthesis results of Chang scaler-Scale 3

n	Area	Delay	Power
8	Combinational area: 429.500000 Buf/Inv area: 71.000000 Net Interconnect area:99.710729 Total area:529.2107291	Data required time 8.05 Data arrival time -8.03 Slack (MET)	Cell internal power = 235.5096 μW (27%) Net switching power = 649.0065 μW (73%) Total dynamic power = 884.5161 μW (100%) Cell leakage power= 70.2844 nW
16	Combinational area: 1222.750000 Buf/Inv area: 237.000000 Net Interconnect area:294.316377 Total area:1517.0663771	Data required time 8.05 Data arrival time -8.04 Slack (MET)	Cell internal power = 567.4903 μW (30%) Net switching power = 1.3140 mW (70%) Total dynamic power = 1.8815 mW (100%) Cell leakage power= 171.6459 nW
32	Combinational area: 2516.750000 Buf/Inv area: 553.500000 Net Interconnect area:570.948027 Total area:3087.698271	Data required time 8.05 Data arrival time -8.05 Slack (MET)	Cell internal power = 1.1131 mW (30%) Net switching power = 2.5529 mW (70%) Total dynamic power = 3.6661 mW (100%) Cell leakage power= 354.1719 nW
64	Combinational area: 5292.750000 Buf/Inv area: 1227.000000 Net Interconnect area:1157.999501 Total area:6450.7495011	Data required time 8.05 Data arrival time -8.53 slack (VIOLATED) -0.48	Cell internal power = 2.4024 mW (32%) Net switching power = 5.1059 mW (68%) Total dynamic power = 7.5083 mW (100%) Cell leakage power= 761.9608 nW
128	Combinational area: 9959.000000 Buf/Inv area: 2223.000000 Net Interconnect area:2264.369268 Total area:12223.3692681	Data required time 8.05 Data arrival time -9.72 slack (VIOLATED) -1.67	Cell internal power = 4.4119 mW (3%) Net switching power = 10.0173 mW (69%) Total dynamic power = 14.4292 mW (100%) Cell leakage power= 1.4141 μW

Table B.4: Synopsys DC synthesis results of Chang scaler

n	Area	Delay	Power
8	Combinational area: 828.500000 Buf/Inv area: 124.500000 Net Interconnect area:398.737708 Total area:2618.9877081	Data required time 8.05 Data arrival time -8.05 Slack (MET)	Cell internal power = 3.7500 mW (69%) Net switching power = 1.6935 mW (31%) Total dynamic power = 5.4435 mW (100%) Cell leakage power= 406.2321 nW
16	Combinational area: 1834.750000 Buf/Inv area: 268.750000 Net Interconnect area:911.556704 Total area:5392.0567041	Data required time 8.05 Data arrival time -8.04 Slack (MET)	Cell internal power = 7.2566 mW (69%) Net switching power = 3.3235 mW (31%) Total dynamic power = 10.5801 mW (100%) Cell leakage power= 836.9442 nW
32	Combinational area: 5831.000000 Buf/Inv area: 1234.500000 Net Interconnect area:1318.317576 Total area:7149.3175761	Data required time 8.05 Data arrival time -8.05 Slack (MET)	Cell internal power = 3.1309 mW (29%) Net switching power = 7.5583 mW (71%) Total dynamic power = 10.6893 mW (100%) Cell leakage power= 936.1253 nW
64	Combinational area: 11406.500000 Buf/Inv area: 2625.750000 Net Interconnect area:2488.695020 Total area:13895.1950201	Data required time 8.05 Data arrival time -8.73 slack (VIOLATED) -0.68	Cell internal power = 6.2138 mW (29%) Net switching power = 15.0227 mW (71%) Total dynamic power = 21.2366 mW (100%) Cell leakage power= 1.8163 μW
128	Combinational area: 21715.500000 Buf/Inv area: 5110.250000 Net Interconnect area:4872.010286 Total area:26587.5102861	Data required time 8.05 Data arrival time -9.90 slack (VIOLATED) -1.85	Cell internal power = 12.1563 mW (29%) Net switching power = 30.0377 mW (71%) Total dynamic power = 42.1940 mW (100%) Cell leakage power= 3.5212 μW

C
Tcl Scripts for Synthesising Initial Binary and the Proposed RNS-based Image Processors Using Synopsys DC

C.1 Synopsys DC Setup File-Setup.tcl

```
set search_path "/home/.../models
/home/.../work /home/.../src /home/.../db /"
set link_library  "* saed90nm_typ_ht.db"
set target_library  "saed90nm_typ_ht.db"

alias h history
alias rc "report_constraint -all_violators"

define_design_lib WORK -path "/home/.../work"

set source_path "/home/.../src/"
set report_path "/home/.../reports/"
set script_path "/home/.../scripts/"
set db_path "/home/.../db/"
```

C.2 Synopsys DC Constraints File-Defaults.con

```
# Define system clock period
set clk_period 10

# Create real clock if clock port is found
if {[sizeof_collection [get_ports clk]] > 0} {
   set clk_name clk
   create_clock -period $clk_period clk
}

# Create virtual clock if clock port is not found
if {[sizeof_collection [get_ports clk]] == 0} {
   set clk_name vclk
   create_clock -period $clk_period -name vclk
}

# Apply default drive strengths and typical loads for I/O ports
set_load 1.5 [all_outputs]
set_driving_cell -no_design_rule -lib_cell INVX0 [all_inputs]

# If real clock, set infinite drive strength
if {[sizeof_collection [get_ports clk]] > 0} {
   set_drive 0 clk
}

# Apply default timing constraints for modules
set_input_delay 1.2 {vblocks[3] vblocks[2] vblocks[1] vblocks[0] x[24] x[23]
x[22] x[21] x[20] x[19] x[18] x[17] x[16] x[15] x[14] x[13] x[12] x[11] x[10]
x[9] x[8] x[7] x[6] x[5] x[4] x[3] x[2] x[1] x[0] reset} -clock $clk_name

set_output_delay 1.5 [all_outputs] -clock $clk_name
set_clock_uncertainty -setup 0.45 $clk_name

# Set operating conditions
set_operating_conditions TYPICAL
# Turn on auto wire load selection
# (library must support this feature)
set auto_wire_load_selection true
```

D

Tcl Scripts for Multi-Voltage Synthesis of Initial Binary and the Proposed RNS-Based Image Processors Using Synopsys DC Topographical Mode

D.1 Synopsys DC Topographical Mode Setup File-Setup_topo.tcl

```
source "/home/.../scripts/path.tcl"

analyze -library WORK -format vhdl
{/home/.../src/dwt_param.vhd
/home/.../src/Modular_lib.vhd
/home/.../src/RNS2D.vhd
/home/.../src/scale.vhd
/home/.../src/ram.vhd
/home/.../src/mux.vhd
/home/.../src/mem.vhd
/home/.../src/invert.vhd
/home/.../src/flip_flop.vhd
/home/.../src/extender.vhd
/home/.../src/demux.vhd
/home/.../src/control_logic.vhd
/home/.../src/cdf97.vhd
/home/.../src/OR_gate.vhd}

elaborate BINARY -architecture BEHAVIORAL -library WORK -update

link
set upf_create_implicit_supply_sets false

source -verbose -echo /home/.../inputs/binary.upf
source -verbose -echo /home/.../scripts/voltage.tcl
source -verbose -echo /home/.../inputs/chiptop+_s0.sdc
compile_ultra
source -verbose -echo /home/.../scripts/report.tcl
uplevel #0 { report_power -analysis_effort medium -verbose }
```

D.2 Multi-voltage Setting Tcl File for Topographical Mode Synthesis -Voltage.tcl

```
set_voltage 0.7 -obj {VDDGS}
set_voltage 1.08 -obj {VDDXS}
set_voltage 1.08 -obj {VDD}
set_voltage 0.000 -obj {VSS}
name_format -level_shift_prefix "LS_"
set_operating_conditions -max TYPICAL -min TYPICAL
```

D.3 UPF File for Compiling the Proposed RNS-based Image Processor Using Synopsys DC Topographical Mode-binary.upf

```
## CREATE POWER DOMAIS
#####################
create_power_domain TOP
create_power_domain CELL -elements {XLXI_1 XLXI_4 XLXI_5
XLXI_43 XLXI_44 XLXI_70 XLXI_72 XLXI_74 XLXI_75 XLXI_76}

create_power_domain EXTENDER  -elements {XLXI_6 XLXI_7}

## TOPLEVEL CONNECTIONS
#######################
# VDD
create_supply_port VDD
create_supply_net  VDD   -domain TOP
connect_supply_net VDD   -ports VDD

# VSS
create_supply_port VSS
create_supply_net  VSS   -domain TOP
create_supply_net  VSS   -domain EXTENDER -reuse
create_supply_net  VSS   -domain CELL -reuse
connect_supply_net VSS   -ports VSS

create_supply_net  VDD   -domain EXTENDER -reuse
create_supply_net  VDD   -domain CELL -reuse

# VDDG
create_supply_port VDDGS
create_supply_net  VDDGS   -domain TOP
create_supply_net  VDDGS   -domain EXTENDER -reuse
connect_supply_net VDDGS  -ports VDDGS

# VDDX
create_supply_port VDDXS
create_supply_net  VDDXS   -domain TOP
create_supply_net  VDDXS   -domain CELL -reuse
connect_supply_net VDDXS   -ports VDDXS

## PRIMARY POWER NETS
#####################
set_domain_supply_net TOP   -primary_power_net VDD   -primary_ground_net VSS
set_domain_supply_net CELL   -primary_power_net VDDXS   -primary_ground_net VSS
set_domain_supply_net EXTENDER  -primary_power_net VDDGS -primary_ground_net VSS

# ADD PORT STATE INFO
#####################
add_port_state VDD    -state {HV  1.08}
add_port_state VDDXS  -state {HV  1.08}
add_port_state VDDGS  -state {LV  0.7}
add_port_state VSS    -state {GND  0}
```

E

Synopsys VCS-MX Tcl Scripts for Simulation of the Proposed RNS-based Image Processor

E.1 Synopsys DVE Setup File-Setup_DVE.tcl

```
VCS_HOME=/usr/synopsys/vcs
export VCS_HOME
PATH=$VCS_HOME/bin:$PATH
export PATH
echo $PATH
/usr/synopsys/vcs/bin:/usr/kerberos/sbin:
/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:
/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin
cd /usr/synopsys/vcs/bin/
vhdlan -full64 /home/.../src/test_bench.vhd
vcs -full64 filter2d_filter2d_sch_tb -debug_all
./simv -gui
```

E.2 Tcl File Used to Generate VCD File

```
./simv -ucli

                   Synopsys VCS MX Compiled Simulator(simv)
                      Version G-2012.09 -- Aug 24, 2012
                   Copyright (c) 1991-2012 by Synopsys Inc.
                             ALL RIGHTS RESERVED

ucli% dump -file temp.vpd -type vpd
VPD0
ucli% dump -add LEGALL_TEST/uut/* -fid VPD0
1
ucli% run 600000 ns
600000 NS
ucli% dump -close
ucli% exit
ls *.vpd
common.vpd  inter.vpd  temp.vpd
vpd2vcd temp.vpd temp.vcd
####################################################
VCD+ to VCD Translator
Copyright (c) 1991-2012 by Synopsys Inc.
#############Version : G-2012.09_Full64 ############
Done with vcd+ hierarchy read.
Done with vcd hierarchy write.
Done writing vcd value changes.

Done
```

F

Synopsys PrimeTime PX Tcl Scripts for Multi-voltage Power Analysis of the Proposed RNS-based Image Processor

F.1 Tcl Script for PrimeTime PX Post Lay Power Analysis

```
source "/home/.../post_lay/scripts/setup.tcl"
set power_enable_analysis TRUE
set power_analysis_mode time_based
read_ddc /home/.../db/RNS2D_MAPPED.ddc
current_design
link
remove_design -all
set power_enable_analysis TRUE
set power_analysis_mode time_based
set search_path "/home/.../ref/models /home/.../src /home/.../db ."
set target_library  "saed90nm_typ_ht.db"
set link_library  [concat $target_library "*"]
```

```
read_ddc /home/.../db/RNS2D_MAPPED.ddc
current_design
link
read_sdc /home/.../db/RNS2D.sdc
check_timing
update_timing
update_power
write_sdc > /home/asafari/Prj/final/db/RNS2D_pt.sdc
```

G

Synopsys DC Report of Post Compile UPF of the Proposed RNS-based Image Processor

G.1 Visual UPF - Post Compile UPF

```
create_power_domain TOP -include_scope
create_power_domain CELL -elements {XLXI_1 XLXI_4 XLXI_5
XLXI_43 XLXI_70 XLXI_72 XLXI_74 XLXI_75 XLXI_76 XLXI_44 }
create_power_domain EXTENDER -elements {XLXI_6 XLXI_7 }
create_supply_port VDD -direction in
create_supply_net VDD -domain TOP
connect_supply_net VDD -ports VDD
create_supply_port VSS -direction in
create_supply_net VSS -domain TOP
create_supply_net VSS -domain EXTENDER -reuse
create_supply_net VSS -domain CELL -reuse
connect_supply_net VSS -ports VSS
create_supply_net VDD -domain EXTENDER -reuse
create_supply_net VDD -domain CELL -reuse
create_supply_port VDDGS -direction in
create_supply_net VDDGS -domain TOP
create_supply_net VDDGS -domain EXTENDER -reuse
```

```
connect_supply_net VDDGS -ports VDDGS
create_supply_port VDDXS -direction in
create_supply_net VDDXS -domain TOP
create_supply_net VDDXS -domain CELL -reuse
connect_supply_net VDDXS -ports VDDXS
set_domain_supply_net TOP -primary_power_net VDD -primary_ground_net VSS
set_domain_supply_net CELL -primary_power_net VDDXS -primary_ground_net VSS
set_domain_supply_net EXTENDER -primary_power_net VDDGS -primary_ground_net VSS
add_port_state VDD -state {HV 1.080000}
add_port_state VDDXS -state {HV 1.080000}
add_port_state VDDGS -state {LV 0.700000}
add_port_state VSS -state {GND 0.000000}
name_format -isolation_suffix _UPF_ISO -level_shift_prefix LS_ -level_shift_suffix _UPF_LS
set derived_upf true
connect_supply_net VDD -ports {
XLXI_6/LS_rw_UPF_LS/VDDH LS_ip1[0]_UPF_LS/VDDH LS_ip1[1]_UPF_LS/VDDH
LS_ip1[2]_UPF_LS/VDDH LS_ip1[3]_UPF_LS/VDDH LS_ip1[4]_UPF_LS/VDDH
LS_ip1[5]_UPF_LS/VDDH LS_ip1[6]_UPF_LS/VDDH LS_ip1[7]_UPF_LS/VDDH
LS_ip1[8]_UPF_LS/VDDH LS_ip1[9]_UPF_LS/VDDH LS_ip0[0]_UPF_LS/VDDH
LS_ip0[1]_UPF_LS/VDDH LS_ip0[2]_UPF_LS/VDDH LS_ip0[3]_UPF_LS/VDDH
LS_ip0[4]_UPF_LS/VDDH LS_ip0[5]_UPF_LS/VDDH LS_ip0[6]_UPF_LS/VDDH
LS_ip0[7]_UPF_LS/VDDH LS_ip0[8]_UPF_LS/VDDH LS_ip0[9]_UPF_LS/VDDH
XLXI_6/LS_en_UPF_LS/VDDH XLXI_7/LS_en_UPF_LS/VDDH
XLXI_7/LS_rw_UPF_LS/VDDH XLXI_6/LS_mode[0]_UPF_LS/VDDH
XLXI_6/LS_mode[1]_UPF_LS/VDDH XLXI_7/LS_mode[0]_UPF_LS/VDDH XLXI_7/LS_mode[1]_UPF_LS/VDDH
XLXI_6/LS_reset_UPF_LS/VDDH XLXI_7/LS_reset_UPF_LS/VDDH
XLXI_6/LS_din[5]_UPF_LS/VDDH XLXI_6/LS_din[6]_UPF_LS/VDDH
XLXI_6/LS_din[7]_UPF_LS/VDDH XLXI_6/LS_din[8]_UPF_LS/VDDH
XLXI_6/LS_din[9]_UPF_LS/VDDH XLXI_6/LS_din[0]_UPF_LS/VDDH
XLXI_6/LS_din[1]_UPF_LS/VDDH XLXI_6/LS_din[2]_UPF_LS/VDDH
XLXI_6/LS_din[3]_UPF_LS/VDDH XLXI_6/LS_din[4]_UPF_LS/VDDH
XLXI_7/LS_din[5]_UPF_LS/VDDH XLXI_7/LS_din[6]_UPF_LS/VDDH
XLXI_7/LS_din[7]_UPF_LS/VDDH XLXI_7/LS_din[8]_UPF_LS/VDDH
XLXI_7/LS_din[9]_UPF_LS/VDDH XLXI_7/LS_din[0]_UPF_LS/VDDH
XLXI_7/LS_din[1]_UPF_LS/VDDH XLXI_7/LS_din[2]_UPF_LS/VDDH
XLXI_7/LS_din[3]_UPF_LS/VDDH XLXI_7/LS_din[4]_UPF_LS/VDDH}
connect_supply_net VSS -ports { XLXI_6/LS_rw_UPF_LS/VSS
LS_ip1[0]_UPF_LS/VSS LS_ip1[1]_UPF_LS/VSS LS_ip1[2]_UPF_LS/VSS
LS_ip1[3]_UPF_LS/VSS LS_ip1[4]_UPF_LS/VSS LS_ip1[5]_UPF_LS/VSS
LS_ip1[6]_UPF_LS/VSS LS_ip1[7]_UPF_LS/VSS LS_ip1[8]_UPF_LS/VSS
LS_ip1[9]_UPF_LS/VSS LS_ip0[0]_UPF_LS/VSS LS_ip0[1]_UPF_LS/VSS
LS_ip0[2]_UPF_LS/VSS LS_ip0[3]_UPF_LS/VSS LS_ip0[4]_UPF_LS/VSS
LS_ip0[5]_UPF_LS/VSS LS_ip0[6]_UPF_LS/VSS LS_ip0[7]_UPF_LS/VSS
LS_ip0[8]_UPF_LS/VSS LS_ip0[9]_UPF_LS/VSS XLXI_6/LS_en_UPF_LS/VSS
XLXI_7/LS_en_UPF_LS/VSS XLXI_7/LS_rw_UPF_LS/VSS
XLXI_6/LS_mode[0]_UPF_LS/VSS XLXI_6/LS_mode[1]_UPF_LS/VSS
XLXI_7/LS_mode[0]_UPF_LS/VSS XLXI_7/LS_mode[1]_UPF_LS/VSS
XLXI_6/LS_reset_UPF_LS/VSS XLXI_7/LS_reset_UPF_LS/VSS
XLXI_6/LS_din[5]_UPF_LS/VSS
XLXI_6/LS_din[6]_UPF_LS/VSS XLXI_6/LS_din[7]_UPF_LS/VSS
XLXI_6/LS_din[8]_UPF_LS/VSS XLXI_6/LS_din[9]_UPF_LS/VSS
XLXI_6/LS_din[0]_UPF_LS/VSS XLXI_6/LS_din[1]_UPF_LS/VSS
XLXI_6/LS_din[2]_UPF_LS/VSS XLXI_6/LS_din[3]_UPF_LS/VSS
XLXI_6/LS_din[4]_UPF_LS/VSS XLXI_7/LS_din[5]_UPF_LS/VSS
XLXI_7/LS_din[6]_UPF_LS/VSS XLXI_7/LS_din[7]_UPF_LS/VSS
XLXI_7/LS_din[8]_UPF_LS/VSS XLXI_7/LS_din[9]_UPF_LS/VSS
XLXI_7/LS_din[0]_UPF_LS/VSS XLXI_7/LS_din[1]_UPF_LS/VSS
XLXI_7/LS_din[2]_UPF_LS/VSS XLXI_7/LS_din[3]_UPF_LS/VSS
XLXI_7/LS_din[4]_UPF_LS/VSS}
connect_supply_net VDDGS -ports { XLXI_6/LS_rw_UPF_LS/VDDL
LS_ip1[0]_UPF_LS/VDDL LS_ip1[1]_UPF_LS/VDDL LS_ip1[2]_UPF_LS/VDDL
LS_ip1[3]_UPF_LS/VDDL LS_ip1[4]_UPF_LS/VDDL LS_ip1[5]_UPF_LS/VDDL
LS_ip1[6]_UPF_LS/VDDL LS_ip1[7]_UPF_LS/VDDL LS_ip1[8]_UPF_LS/VDDL
LS_ip1[9]_UPF_LS/VDDL LS_ip0[0]_UPF_LS/VDDL LS_ip0[1]_UPF_LS/VDDL
LS_ip0[2]_UPF_LS/VDDL LS_ip0[3]_UPF_LS/VDDL LS_ip0[4]_UPF_LS/VDDL
LS_ip0[5]_UPF_LS/VDDL LS_ip0[6]_UPF_LS/VDDL LS_ip0[7]_UPF_LS/VDDL
LS_ip0[8]_UPF_LS/VDDL LS_ip0[9]_UPF_LS/VDDL XLXI_6/LS_en_UPF_LS/VDDL
XLXI_7/LS_en_UPF_LS/VDDL XLXI_7/LS_rw_UPF_LS/VDDL
XLXI_6/LS_mode[0]_UPF_LS/VDDL XLXI_6/LS_mode[1]_UPF_LS/VDDL
XLXI_7/LS_mode[0]_UPF_LS/VDDL XLXI_7/LS_mode[1]_UPF_LS/VDDL
XLXI_6/LS_reset_UPF_LS/VDDL XLXI_7/LS_reset_UPF_LS/VDDL
XLXI_6/LS_din[5]_UPF_LS/VDDL XLXI_6/LS_din[6]_UPF_LS/VDDL
XLXI_6/LS_din[7]_UPF_LS/VDDL XLXI_6/LS_din[8]_UPF_LS/VDDL
XLXI_6/LS_din[9]_UPF_LS/VDDL XLXI_6/LS_din[0]_UPF_LS/VDDL
XLXI_6/LS_din[1]_UPF_LS/VDDL XLXI_6/LS_din[2]_UPF_LS/VDDL
XLXI_6/LS_din[3]_UPF_LS/VDDL XLXI_6/LS_din[4]_UPF_LS/VDDL
XLXI_7/LS_din[5]_UPF_LS/VDDL XLXI_7/LS_din[6]_UPF_LS/VDDL
XLXI_7/LS_din[7]_UPF_LS/VDDL XLXI_7/LS_din[8]_UPF_LS/VDDL
XLXI_7/LS_din[9]_UPF_LS/VDDL XLXI_7/LS_din[0]_UPF_LS/VDDL
XLXI_7/LS_din[1]_UPF_LS/VDDL XLXI_7/LS_din[2]_UPF_LS/VDDL
XLXI_7/LS_din[3]_UPF_LS/VDDL XLXI_7/LS_din[4]_UPF_LS/VDDL}
set derived_upf false

# Visual UPF added these lines...
#Power State Table
create_pst pst -supplies { VDD VDDXS VDDGS VSS }
```

```
add_pst_state s0 -pst pst -state { HV HV LV GND }
```

H

Synopsys IC Compiler Tcl Scripts for Physical Implementation of the Proposed RNS-based Image Processor

H.1 Synopsys IC Compiler Setup File-Setup_icc.tcl

```
set search_path "/home/.../icc/ref/models /home/.../icc/work /home/.../icc/source /"
set link_library  "* saed90nm_typ_ht.db"
set target_library  "saed90nm_typ_ht.db"

alias h history
alias rc "report_constraint -all_violators"

#define_design_lib WORK -path "/home/.../icc/work"

set source_path "/home/.../icc/source/"
set report_path "/home/.../icc/reports/"
set script_path "/home/.../icc/scripts/"
set db_path "/home/.../icc/db/"
```

H.2 Creat MilkyWay Library “RNS2D”

```
create_mw_lib  -technology /home/.../icc/ref/techfiles/saed90nm_icc_1p9m.tf
                -mw_reference_library {/home/.../icc/ref/saed90nm_fr }
                -hier_separator {/}
                -bus_naming_style {[%d]}
                -open  ./RNS2D
```

H.3 Set TLU+ Files

```
set_tlu_plus_files -max_tluplus
/home/.../icc/ref/tluplus/saed90nm_1p9m_1t_Cmax.tluplus -min_tluplus
/home/.../icc/ref/tluplus/saed90nm_1p9m_1t_Cmin.tluplus -tech2itf_map
/home/.../icc/ref/tluplus/tech2itf.map
```

H.4 Import the Design and Constraints File

```
import_designs -format ddc {/home/.../icc/source/RNS2D.ddc}
read_sdc  -version Latest "/home/.../icc/source/RNS2D.sdc"
```

H.5 Initialise Floorplan

```
create_floorplan -core_utilization 0.35 -left_io2core 1 -bottom_io2core 1
                                        -right_io2core 1 -top_io2core 1
```

H.6 Set Power and Ground Nets and Pins

```
set power               "VDD"
set ground              "VSS"
set powerPort           "VDD"
set groundPort          "VSS"
set mw_logic0_net       "VSS"
set mw_logic1_net       "VDD"
derive_pg_connection -power_net VDD
                     -ground_net VSS
```

```
                      -power_pin VDD
                      -ground_pin VSS
```

H.7 Add Rectangular Power and Ground Rings

```
create_rectangular_rings  -nets  {VSS VDD}  -left_offset 0.2
-left_segment_layer M4 -right_offset 0.2
-right_segment_layer M4 -bottom_offset 0.2
-bottom_segment_layer M3 -extend_bh -top_offset 0.2 -top_segment_layer M3
```

H.8 Create Power Straps

```
create_power_straps  -direction horizontal  -nets  {VDD}
-layer M4 -configure groups_and_step
-num_groups 28 -step 3

create_power_straps  -direction horizontal  -start_at 1.5
-nets  {VSS}  -layer M4 -configure groups_and_step
-num_groups 28 -step 3

create_power_straps  -direction vertical  -nets  {VDD}
-layer M3 -configure groups_and_step
-num_groups 28 -step 3

create_power_straps  -direction vertical  -start_at 1.5 -nets  {VSS}
-layer M3 -configure groups_and_step
-num_groups 28 -step 3
```

H.9 Core Placement and Optimisation

```
place_opt  -effort high -power -continue_on_missing_scandef -congestion
```

H.10 Clock Tree Synthesis

```
clock_opt -only_psyn -continue_on_missing_scandef
```

H.11 Preroute Standard Cells

```
preroute_standard_cells -nets  {VDD VSS}  -connect horizontal
-port_filter_mode off -cell_master_filter_mode off -cell_instance_filter_mode off
-voltage_area_filter_mode off -route_type {P/G Std. Cell Pin Conn}
```

H.12 Core Route and Optimisation

```
route_opt -effort low
```

H.13 Insert Fillers

```
insert_stdcell_filler
```

H.14 Verification

```
verify_drc
verify_lvs
```

H.15 Save the File

```
write_verilog  -no_physical_only_cells /home/.../icc/results/RNS2D_fm.v
write_parasitics -output      {/home/.../icc/results/RNS2D.spf}
set_write_stream_options -map_layer /home/.../icc/ref/saed90nm.gdsout.map
                         -output_filling fill    -child_depth 20
                         -output_outdated_fill    -output_pin {text geometry}
write_stream -format gds -lib_name /home/.../icc/work/RNS2D -cells {RNS2D} RNS2D.gds
write_stream -format gds -lib_name /home/.../icc/work/RNS2D -cells {RNS2D } write_stream
```

I

Synopsys IC Compiler Reports for Physical Implementation of the Proposed RNS-based Image Processor

I.1 Linking the Design

```
Linking design ‘‘RNS2D’’ Using the following designs and libraries:
--------------------------------------------------------------------------
{*} (129 designs) ../icc/RNS2D_MAPPED.ddc,etc
saed90nm_typ_ht (library) ../icc/ref/models/saed90nm_typ_ht.db
Info: Creating auto CEL.
Preparing data for query..................
Information: Performing CEL netlist consistency check. (MWDC-118)
Information: CEL consistency check PASSED. (MWDC-119)
Information: Saved design named RNS2D. (UIG-5)
Preparing data for query..................
1
```

I.2 Sanity Check on TLU+ Files

```
1. Checking the conducting layer names in ITF and mapping file ...
{[} Passed! {]}

2. Checking the via layer names in ITF and mapping file ... {[} Passed!
{]}

3. Checking the consistency of Min Width and Min Spacing between MW-tech
and ITF ... {[} Passed! {]}

----------------- Check Ends ------------------

TLUPlus based RC computation is enabled. (RCEX-141)

The distance unit in Capacitance and Resistance is 1 micron. (RCEX-007)

The RC model used is TLU+. (RCEX-015)
```

I.3 Floorplanning

```
Planner Summary:

This floorplan is created by using tile name (unit).
Row Direction = HORIZONTAL
Control Parameter = Aspect Ratio
Core Utilization = 0.350
Number Of Rows = 1924
Core Width = 5543.68
Core Height = 5541.12
Aspect Ratio = 1.000
Double Back ON
Flip First Row = YES
Start From First Row = YES
Planner run through successfully.
```

I.4 Placement

```
Beginning Coarse Placement
  --------------------------

Information:
```

```
Running stand-alone coarse placer in a separate process using temp directory '/tmp'.

Warning: Scan DEF information is required. (PSYN-1099)
...13%...25%...38%...50%...63%...75%...88%...100% done.

  Coarse Placement Complete
  --------------------------

Information: connected 716661 power ports and 716661 ground ports
```

I.5 Chip Summary

```
****************************************
  Report : Chip Summary
  Design : RNS2D
  Version: G-2012.06-ICC-SP3
****************************************
Std cell utilization: 35.28%  (11838374/(33558849-0))
(Non-fixed + Fixed)
Std cell utilization: 35.28%  (11838374/(33558849-0))
(Non-fixed only)
Chip area:            33558849 sites, bbox (1.00 1.00 5562.28 5562.28) um
Std cell area:        11838374 sites, (non-fixed:11838374 fixed:0)
                      709901   cells, (non-fixed:709901 fixed:0)
Macro cell area:      0        sites
                      0        cells
Placement blockages:  0        sites, (excluding fixed std cells)
                      0        sites, (include fixed std cells & chimney area)
                      0        sites, (complete p/g net blockages)
Routing blockages:    0        sites, (partial p/g net blockages)
                      0        sites, (routing blockages and signal pre-route)
Lib cell count:       53
Avg. std cell width:  4.46 um
Site array:           unit     (width: 0.32 um, height: 2.88 um, rows: 1931)
Physical DB scale:    1000 db_unit = 1 um
```

I.6 Legalize Displacement

```
****************************************
```

```
  Report : pnet options
  Design : RNS2D
  Version: G-2012.06-ICC-SP3
****************************************


--------------------------------------------------------------------
Layer      Blockage   Min_width   Min_height   Via_additive     Density
--------------------------------------------------------------------
M1         none          ---         ---       via additive      ---
M2         none          ---         ---       via additive      ---
M3         none          ---         ---       via additive      ---
M4         none          ---         ---       via additive      ---
M5         none          ---         ---       via additive      ---
M6         none          ---         ---       via additive      ---
M7         none          ---         ---       via additive      ---
M8         none          ---         ---       via additive      ---
M9         none          ---         ---       via additive      ---
Legalizing 709901 illegal cells...
Starting legalizer.
Initial legalization:  100% (6 sec)
Optimizations pass 1: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% (28.7 sec)
Optimizations pass 2: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% (29.4 sec)
Optimizations pass 3: 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% (28.3 sec)
Legalization complete (106 total sec)

****************************************
Report : Legalize Displacement
Design : RNS2D
Version: G-2012.06-ICC-SP3
****************************************
avg cell displacement: 0.760 um ( 0.26 row height)
max cell displacement: 3.853 um ( 1.34 row height)
std deviation: 0.574 um (0.20 row height)
number of cell moved: 416 cells (out of 654321 cells)
Total 0 cells has large displacement (e.g. > 8.640 um or 3 row height)

Placement Optimization Complete
```

I.7 Clock Tree Summary

```
****************************************
Report : qor
```

```
Design : RNS2D
Version: G-2012.06-ICC-SP3
****************************************


  Timing Path Group 'clk'
  -----------------------------------
  Levels of Logic:              62.00
  Critical Path Length:         37.14
  Critical Path Slack:          62.26
  Critical Path Clk Period:    100.00
  Total Negative Slack:          0.00
  No. of Violating Paths:        0.00
  Worst Hold Violation:          0.00
  Total Hold Violation:          0.00
  No. of Hold Violations:        0.00
  -----------------------------------


  Cell Count
  -----------------------------------
  Hierarchical Cell Count:        126
  Hierarchical Port Count:       3571
  Leaf Cell Count:             654321
  Buf/Inv Cell Count:           36599
  CT Buf/Inv Cell Count:            0
  Combinational Cell Count:    423246
  Sequential Cell Count:       231075
  Macro Count:                      0
  -----------------------------------


  Area
  -----------------------------------
  Combinational Area:  4904381.892862
  Noncombinational Area:
                       5754479.545292
  Buf/Inv Area:         418664.448485
  Net Area:            6352950.951578
  Net XLength        :    25329066.00
  Net YLength        :    23133108.00
  -----------------------------------
  Cell Area:          10658861.438154
  Design Area:        17011812.389732
  Net Length        :     48462176.00


  Design Rules
  -----------------------------------
  Total Number of Nets:        934475
  Nets With Violations:             0
  Max Trans Violations:             0
  Max Cap Violations:               0
  -----------------------------------

ROPT:    (SETUP) WNS: 0.0000 TNS: 0.0000  Number of Violating Path: 0
ROPT:    (HOLD) WNS: 0.0000 TNS: 0.0000  Number of Violating Path: 0
ROPT:    Number of DRC Violating Nets: 0
ROPT:    Number of Route Violation: 0
1
```

I.8 Filler Cell Insertion

```
=== Filler Cell Insertion ======
PARAM: respectMacroPadding = FALSE
PARAM: respectPlacementBlockage = TRUE

Initializing Data Structure ...
  Reading technology information ...
    Technology table contains 9 routable metal layers
    This is considered as a 9-metal-layer design
    Reading library information from DB ...
  Reading netlist information from DB ...
    709087 placeable cells
    0 cover cells
    61 IO cells/pins
    709148 cell instances
  Sorting cells, nets, pins ...
    net pin threshold = 33
  Reading misc information ...
    array <unit> has 0 vertical and 1931 horizontal rows
    GRC ref loc X corrected
    GRC ref loc Y corrected
    227 pre-routes for placement blockage/checking
    121642 pre-routes for map congestion calculation
    Auto Set : first cut = vertical
  Checking information read in ...
    design style = Horizontal masters, Horizontal rows
  Processing std cells for voltage threshold type...
  Preprocessing design ...
    processing macro cells (if any)
    processing preroute blockages (if any)
    processing hard placement blockages (if any)
    processing soft placement blockages (if any)
    Auto Set : first cut = vertical
    processing std cells
    Pass I: adjust placeable rows
    Pass II: mark placed cells
  Processing filler cells...
Hierarchical update for new filler cells

INFO: Fillers rules in use ...

  ** LR Filler Rules **

  ** VT Filler Rules **
=== End of Filler Cell Insertion ===
```

I.9 Design Verification

```
-- LVS START : --
Total area error in layer 0 is 0.  Elapsed =    0:00:13, CPU =    0:00:13
Total area error in layer 1 is 0.  Elapsed =    0:00:35, CPU =    0:00:35
Total area error in layer 2 is 0.  Elapsed =    0:00:53, CPU =    0:00:53
Total area error in layer 3 is 0.  Elapsed =    0:07:52, CPU =    0:07:52
Total area error in layer 4 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 5 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 6 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 7 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 8 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 9 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 10 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 11 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 12 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 13 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 14 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
Total area error in layer 15 is 0.  Elapsed =    0:07:55, CPU =    0:07:55
```

List of Acronyms/Abbreviations

ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BDWT	Bi-orthogonal Discrete Wavelet Transform
BK	Brent-Kung
BPP	Bits Per Pixel
BTC	Block Truncation Coding
CDF	Cumulative Distribution Function
CR	Compression Ratio
CRT	Chinese Remainder Theorem
CWT	Continuous Wavelet Transform
DC	Design Compiler
DCT	Discrete Cosine Transform
DEF	Design Exchange Format
DPCM	Differential Pulse Code Modulation
DRC	Design Rule Checking
DSP	Digital Signal Processing
DVE	Discovery Visual Environment
DWT	Discrete Wavelet Transform
EAC	End-Around Carry
EDA	Electronic Design Automation
FA	Full Adder

FNT	Fermat Number Transform
FPGA	Field Programmable Gate Array
FSDB	Fast Signal Data Base
FT	Fourier Transform
GCD	Greatest Common Divisor
GDSII	Graphical Design Station Version II
GFNT	Generalised Fermat Number Transform
GIF	Graphics Interchange Format
GTECH	Generic Technology
GUI	Graphical User Interface
HDL	Hardware Description Language
HH	High High
HL	High Low
IC	Integrated Circuit
ICC	IC Compiler
IEC	International Electro-technical Commission
I/O	Input/Output
IP	Intellectual Property
ISO	International Organisation for Standardisation
ITF	Interconnect Technology Format
ITU	International Telecommunication Union
JBIG	Joint Bi-level Image Experts Group
JPEG	Joint Photogaphic Expert Group
LEF	Library Exchange Format
LH	Low High
LL	Low Low
LOCO-I	LOw Complexity LOssless COmpression Image
LPD	Low-Power Design

LS	Level Shifter
LSB	Least-Significant Bit
LUT	Look-Up Table
LVS	Layout Versus Schematic
LZW	Lempel-Ziv-Welch
MCMM	Multicorner-Multimode
MRC	Mixed-Radix Converter
MSB	Most-Significant Bit
MSE	Mean Square Error
MV	Multi-Voltage
NCF	Netlist Constraints File
NTICE	Number-Theory-based Image Compression Encryption
OASIS	Open Artwork System Interchange Standard
PNG	Portable Network Graphics
PSNR	Peak Signal-to-Noise Ratio
PVT	Process Voltage Temperature
QoR	Quality of Results
RC	Resistance Coefficient
RGB	Red Green Blue
ROM	Read-Only Memory
RNS	Residue Number System
RTL	Register Transfer Level
SDC	Synopsys Design Constraints
SPEF	Standard Parasitic Exchange Format
Tcl	Tool Command Language
UCF	User Constraints File
UPF	Unified Power Format
VCD	Values Change Dump

VCS	Synopsys Verilog Compiler Simulator
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration
VQ	Vector Quantisation
WLM	Wire Load Model
WT	Wavelet Transform
2D	Two Dimensional

References

[1] C.-H. Chang and J. Low. *Simple, fast, and exact RNS scaler for the three-moduli set.* Circuits and Systems I: Regular Papers, IEEE Transactions on **58**(11), 2686 (2011). xx, 7, 51, 54, 55, 59, 61, 62, 64, 65, 68, 72, 80, 81, 82, 101

[2] M. Martina and G. Masera. *Low-complexity, efficient 9/7 wavelet filters VLSI implementation.* Circuits and Systems II: Express Briefs, IEEE Transactions on **53**(11), 1289 (2006). xx, 113

[3] Y. Liu and E.-K. Lai. *Design and implementation of an RNS-based 2-D DWT processor.* Consumer Electronics, IEEE Transactions on **50**(1), 376 (2004). xx, 2, 3, 36, 109, 113, 115, 135, 136, 167

[4] A. Omondi and B. Premkumar. *Residue number systems: theory and implementation* (Imperial College Press, 2007). xx, 52, 114, 115, 116, 117

[5] Synopsys, Inc. *Design compiler used guide*, g-2012.06 ed. (2012). xxi, 141, 142, 143, 144, 145, 147, 148, 150, 152

[6] J. M. Williams. *Digital VLSI Design with Verilog: A Textbook from Silicon Valley Technical Institute* (Springer, 2008). xxi, 161

[7] M. I. Mahmoud, M. I. Dessouky, S. Deyab, and F. H. Elfouly. *Comparison between haar and daubechies wavelet transformions on FPGA technology.* World Academy of Science, Engineering and Technology **26**, 68 (2007). xxiii, 36, 37

[8] A. Skodras, C. Christopoulos, and T. Ebrahimi. *The JPEG 2000 still image compression standard.* Signal Processing Magazine, IEEE **18**(5), 36 (2001). xxiii, 40

[9] K. M, F. D, A. R, G. A, and S. K. *Low power methodology manual for system-on-chip design* (springer.com, 2007). xxv, 157, 158, 159, 160

[10] C. Huang, D. Peterson, H. Rauch, J. Teague, and D. Fraser. *Implementation of a fast digital processor using the residue number system.* Circuits and Systems, IEEE Transactions on **28**(1), 32 (1981). 1, 50, 59

[11] S. Waser and M. J. Flynn. *Introduction to arithmetic for digital systems designers* (Harcourt Brace College Publishers, 1995). 1

[12] B. Rejeb, H. Henkelmann, and W. Anheier. *Hardware/software implementation of real-time fractal image coding* (Citeseer, 2001). 2

[13] A. Safari and Y. Kong. *Four tap daubechies filter banks based on RNS.* In *Communications and Information Technologies (ISCIT), 2012 International Symposium on*, pp. 952–955 (IEEE, 2012). 2

[14] J. Ramírez, U. Meyer-Bäse, F. Taylor, A. García, and A. Lloris. *Design and implementation of high-performance RNS wavelet processors using custom IC technologies.* Journal of VLSI signal processing systems for signal, image and video technology **34**(3), 227 (2003). 2, 3, 167

[15] S. Srinivasan. *Modulo transforms-an alternative to lifting.* Signal Processing, IEEE Transactions on **54**(5), 1864 (2006).

[16] W. L. Freking and K. K. Parhi. *Low-power FIR digital filters using residue arithmetic.* In *Signals, Systems & Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on*, vol. 1, pp. 739–743 (IEEE, 1997).

[17] M. G. Arnold. *The residue logarithmic number system: theory and implementation.* In *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, pp. 196–205 (IEEE, 2005).

[18] D. K. Taleshmekaeil and A. Mousavi. *The use of residue number system for improving the digital image processing.* In *Signal Processing (ICSP), 2010 IEEE 10th International Conference on*, pp. 775–780 (IEEE, 2010). 4, 52

[19] A. Mousavi and D. K. Taleshmekaeil. *Pipelined residue logarithmic numbers system for general modules set $(2^n - 1, 2^n, 2^n + 1)$.* In *Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on*, pp. 699–703 (IEEE, 2010). 2

[20] W. Jenkins and B. Leon. *The use of residue number systems in the design of finite impulse response digital filters.* Circuits and Systems, IEEE Transactions on **24**(4), 191 (1977). 2

[21] J. Ramírez, A. García, P. Fernández, L. Patrilla, and A. Lloris. *RNS-FPL merged architectures for orthogonal DWT*. Electronics Letters **36**(14), 1198 (2000). 2

[22] J. Ramirez, A. Garcia, L. Parrilla, A. Lloris, and P. Fernandez. *Implementation of RNS analysis and synthesis filter banks for the orthogonal discrete wavelet transform over FPL devices*. In *Circuits and Systems, 2000. Proceedings of the 43rd IEEE Midwest Symposium on*, vol. 3, pp. 1170–1173 (IEEE, 2000). 2, 33, 36

[23] A. Ammar, A. Al Kabbany, M. Youssef, and A. Amam. *A secure image coding scheme using residue number system*. In *Radio Science Conference, 2001. NRSC 2001. Proceedings of the Eighteenth National*, vol. 2, pp. 399–405 (IEEE, 2001). 2, 54, 57

[24] G. C. Cardarilli, A. Del Re, A. Nannarelli, and M. Re. *Low power and low leakage implementation of RNS FIR filters*. In *Proc. of 39th Asilomar Conference on Signals, Systems, and Computers*, pp. 1620–1624 (2005). 3

[25] W. Wang, M. Swamy, and M. O. Ahmad. *RNS application for digital image processing*. In *System-on-Chip for Real-Time Applications, 2004. Proceedings. 4th IEEE International Workshop on*, pp. 77–80 (IEEE, 2004). 3, 57

[26] M. Keating. *Low Power Methodology Manual: For System on Chip Design* (Springer, 2007). 4, 162

[27] T. Toivonen and J. Heikkila. *Video filtering with fermat number theoretic transforms using residue number system*. Circuits and Systems for Video Technology, IEEE Transactions on **16**(1), 92 (2006). 4, 17, 25, 124

[28] V. Jagannathan, A. Mahadevan, R. Hariharan, and S. Srinivasan. *Number theory based image compression encryption and application to image multiplexing*. In *Signal Processing, Communications and Networking, 2007. ICSCN'07. International Conference on*, pp. 59–64 (IEEE, 2007). 4

[29] G. Bernocchi, G. Cardarilli, A. Del Re, A. Nannarelli, and M. Re. *Low-power adaptive filter based on RNS components*. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 3211–3214 (2007). 4, 61

[30] A. K. Jain. *Fundamentals of digital image processing*, vol. 3 (Prentice-Hall Englewood Cliffs, 1989). 11

[31] J. Rosenthal. *JPEG Image Compression using an FPGA*. Ph.D. thesis, UNIVERSITY OF CALIFORNIA (2006). 12

[32] P. Symes. *Digital video compression* (McGraw Hill Professional, 2004). 12

[33] M. J. Weinberger, G. Seroussi, and G. Sapiro. *The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS*. Image Processing, IEEE Transactions on **9**(8), 1309 (2000). 12

[34] Q. Lu, L. Du, and B. Hu. *Low-power JPEG2000 implementation on DSP-based camera node in wireless multimedia sensor networks*. In *Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC'09. International Conference on*, vol. 1, pp. 300–303 (IEEE, 2009). 12

[35] M. Bhuyan, N. Amin, M. A. H. Madesa, and M. S. Islam. *FPGA realization of lifting based forward discrete wavelet transform for JPEG 2000*. International Journal of Circuits, Systems and Signal Processing **1**(2), 124 (2007). 12, 25

[36] M. Vishwanath. *The recursive pyramid algorithm for the discrete wavelet transform*. Signal Processing, IEEE Transactions on **42**(3), 673 (1994). 12, 33

[37] C.-C. Chang and Y.-L. Lin. *A dual mode (5, 3)/(9, 7) FDWT/IDWT hardware accelerator IP* (2004). 13, 40

[38] V. Jagannathan, A. Mahadevan, R. Hariharan, and E. Srinivasan. *Simultaneous color image compression and encryption using number theory*. In *Proceedings of ICIS*, vol. 5, p. 1 (Citeseer, 2005). 13

[39] P. Symes. *Video compression demystified* (McGraw-Hill Professional, 2000). 13

[40] J. Watkinson. *Art of digital audio* (CRC Press, 2013). 13

[41] M. Yang and N. Bourbakis. *An overview of lossless digital image compression techniques*. In *Circuits and Systems, 2005. 48th Midwest Symposium on*, pp. 1099–1102 (IEEE, 2005). 13, 24

[42] U. S. Mehta, K. S. Dasgupta, and N. M. Devashrayee. *Run-length-based test data compression techniques: how far from entropy and power bounds?a survey*. VLSI Design **2010**, 1 (2010). 14

[43] N. V. Boulgouris, D. Tzovaras, and M. G. Strintzis. *Lossless image compression based on optimal prediction, adaptive lifting, and conditional arithmetic coding.* Image Processing, IEEE Transactions on **10**(1), 1 (2001). 16

[44] Z. Xiong, K. Ramchandran, M. T. Orchard, and Y.-Q. Zhang. *A comparative study of DCT and wavelet-based image coding.* Circuits and Systems for Video Technology, IEEE Transactions on **9**(5), 692 (1999). 17, 20

[45] V. Spiliotopoulos, N. Zervas, Y. Andreopoulos, G. Anagnostopoulos, and C. E. Goutis. *Quantization effect on VLSI implementations for the 9/7 DWT filters.* In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, vol. 2, pp. 1197–1200 (IEEE, 2001). 17

[46] A. S. Lewis and G. Knowles. *Image compression using the 2-D wavelet transform.* Image Processing, IEEE Transactions on **1**(2), 244 (1992). 20, 38

[47] C. M. Brislawn, J. N. Bradley, R. J. Onyshczak, and T. Hopper. *FBI compression standard for digitized fingerprint images.* In *SPIE's 1996 International Symposium on Optical Science, Engineering, and Instrumentation*, pp. 344–355 (International Society for Optics and Photonics, 1996). 24

[48] M. Boliek. *JPEG 2000 final committee draft.* ISO/IEC FCD15444 **1** (2000). 24

[49] M. M. Dewasthale and P. Mukherji. *FPGA implementation of wavelet transform based on lifting scheme.* In *Information Management and Engineering, 2009. ICIME'09. International Conference on*, pp. 456–460 (IEEE, 2009). 25

[50] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. *Image coding using wavelet transform.* Image Processing, IEEE Transactions on **1**(2), 205 (1992). 25

[51] H. Liao, M. K. Mandal, and B. F. Cockburn. *Efficient architectures for 1-D and 2-D lifting-based wavelet transforms.* Signal Processing, IEEE Transactions on **52**(5), 1315 (2004). 25

[52] H. Olkkonen and J. T. Olkkonen. *Simplified biorthogonal discrete wavelet transform for VLSI architecture design.* Signal, Image and Video Processing **2**(2), 101 (2008). 25, 26

[53] M. Grangetto, M. Martina, G. Masera, G. Piccinini, F. Vacca, and M. Zamboni. *FPGA power efficient inverse lifting wavelet IP.* In *Signals, Systems and*

Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on, vol. 2, pp. 1325–1329 (IEEE, 2001). 25

[54] P. McCanny, S. Masud, and J. McCanny. *An efficient architecture for the 2-D biorthogonal discrete wavelet transform.* In *Image Processing, 2001. Proceedings. 2001 International Conference on*, vol. 3, pp. 314–317 (IEEE, 2001). 25, 40

[55] K. Andra, C. Chakrabarti, and T. Acharya. *A VLSI architecture for lifting-based forward and inverse wavelet transform.* Signal Processing, IEEE Transactions on **50**(4), 966 (2002). 26

[56] J. M. Jou, Y.-H. Shiau, and C.-C. Liu. *Efficient VLSI architectures for the biorthogonal wavelet transform by filter bank and lifting scheme.* In *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, vol. 2, pp. 529–532 (IEEE, 2001). 25, 26

[57] F. Iqbal. *Wavelet transform based image compression on FPGA.* Ph.D. thesis, FLORIDA STATE UNIVERSITY (2004). 26

[58] P.-C. Wu and L.-G. Chen. *An efficient architecture for two-dimensional discrete wavelet transform.* Circuits and Systems for Video Technology, IEEE Transactions on **11**(4), 536 (2001). 31

[59] J.-D. Wu and C.-H. Liu. *Investigation of engine fault diagnosis using discrete wavelet transform and neural network.* Expert Systems with Applications **35**(3), 1200 (2008). 32

[60] S. G. Mallat. *Multifrequency channel decompositions of images and wavelet models.* Acoustics, Speech and Signal Processing, IEEE Transactions on **37**(12), 2091 (1989). 32, 33

[61] M. Unser. *Vanishing moments and the approximation power of wavelet expansions.* In *Image Processing, 1996. Proceedings., International Conference on*, vol. 1, pp. 629–632 (IEEE, 1996). 32

[62] G. Uytterhoeven, D. Roose, and A. Bultheel. *Wavelet transforms using the lifting scheme.* ITA-Wavelets Report WP **1** (1997). 32

[63] M. Nagabushanam and S. Ramachandran. *Design and implementation of parallel and pipelined distributive arithmetic based discrete wavelet transform IP core.* European Journal of Scientific Research, ISSN pp. 378–392 (2009). 33, 109

[64] I. Daubechies. *Ten lectures on wavelets, CBMS-NSF regional conference series in Applied Mathematics, vol. 61, society for industrial and applied mathematics (SIAM), Philadelphia, PA, 1992.* MR MR1162107 (93e: 42045) . 33, 37

[65] S. Mallat. *A wavelet tour of signal processing* (Access Online via Elsevier, 1999). 35

[66] M. Lightstone, E. Majani, and S. K. Mitra. *Low bit-rate design considerations for wavelet-based image coding.* Multidimensional systems and signal processing **8**(1-2), 111 (1997). 36, 39

[67] A. Lewis and G. Knowles. *VLSI architecture for 2D Daubechies wavelet transform without multipliers.* Electronics letters **27**(2), 171 (1991). 36, 124

[68] J. S. Walker. *A primer on wavelets and their scientific applications* (CRC press, 2002). 37

[69] N. Kingsbury and J. Mugarey. *Wavelet transforms in image processing.* Signal Analysis and Prediction p. 27 (1998). 37

[70] M. J. Smith and S. L. Eddins. *Analysis/synthesis techniques for subband image coding.* Acoustics, Speech and Signal Processing, IEEE Transactions on **38**(8), 1446 (1990). 39

[71] D. B. Tay. *Rationalizing the coefficients of popular biorthogonal wavelet filters.* Circuits and Systems for Video Technology, IEEE Transactions on **10**(6), 998 (2000). 40

[72] B. Parhami. *Computer arithmetic: algorithms and hardware designs* (Oxford University Press, Inc., 2009). 47, 48, 50, 59, 63

[73] P. A. Mohan. *Residue number systems: algorithms and architectures* (Springer, 2002). 48, 52, 54, 61, 66, 75

[74] N. S. Szabo and R. I. Tanaka. *Residue arithmetic and its applications to computer technology*, vol. 24 (McGraw-Hill New York, 1967). 49, 50, 59, 60, 63

[75] O. Abdelfattah. *Data Conversion in Residue Number System.* Ph.D. thesis, McGill University (2011). 49

[76] M. A. P. Shenoy and R. Kumaresan. *A fast and accurate RNS scaling technique for high speed signal processing.* IEEE Trans. Acoust., Speech, Signal Process **37**(6), 929 (1989). 49, 59, 60, 61, 82

[77] R. Conway and J. Nelson. *Fast converter for 3 moduli RNS using new property of CRT.* Computers, IEEE Transactions on **48**(8), 852 (1999). 51

[78] C. Huang, D. Peterson, H. Rauch, J. Teague, and D. Fraser. *Implementation of a fast digital processor using the residue number system.* Circuits and Systems, IEEE Transactions on **28**(1), 32 (1981). 51

[79] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor. *Residue number system arithmetic: Modern applications in digital signal processing.* IEEE Press. (1986). 52

[80] M.-H. Sheu, S.-H. Lin, C. Chen, and S.-W. Yang. *An efficient VLSI design for a residue to binary converter for general balance moduli* $(2^n - 3, 2^n + 1, 2^n - 1, 2^n + 3)$. Circuits and Systems II: Express Briefs, IEEE Transactions on **51**(3), 152 (2004). 52, 54

[81] M. Abdallah and A. Skavantzos. *On multimoduli residue number systems with moduli of forms* $(r^a, r^b - 1, r^c + 1)$. Circuits and Systems I: Regular Papers, IEEE Transactions on **52**(7), 1253 (2005).

[82] S. Tan and J. Vandcwallc. *Canonical forms for singular systcms.* In *35th IEEE Coil/. Decision Contr.*, pp. 214–2149 (1986). 52

[83] M. Abdallah and A. Skavantzos. *A systematic approach for selecting practical moduli sets for residue number systems.* In *Proceedings of the 27th Southeastern Symposium on System Theory (SSST'95)*, pp. 445–449 (IEEE Computer Society, 1995). 53

[84] H. H. Rosenbrock. *Structural properties of linear dynamical systems.* Inr. J. Cuntr **20**, 191 (1974). 54

[85] G. Verghese, B. Levy, and T. Kailath. *A generalized state-space for singular systems.* Automatic Control, IEEE Transactions on **26**(4), 811 (1981). 54

[86] M. Abdallah and A. Skavantzos. *On multimoduli residue number systems with moduli of forms* $(r^a, r^b - 1, r^c + 1)$. Circuits and Systems I: Regular Papers, IEEE Transactions on **52**(7), 1253 (2005). 54

[87] C. B. Dutta, P. Garai, and A. Sinha. *A scheme for improving bit efficiency for residue number system*. In *Advances in Computing and Information Technology*, pp. 649–656 (Springer, 2013). 54

[88] W. Wang, M. Swamy, M. O. Ahmad, and Y. Wang. *A study of the residue-to-binary converters for the three-moduli sets*. Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on **50**(2), 235 (2003). 54, 59, 109

[89] A. Hiasat and S. Abdel-Aty-Zohdy. *Residue-to-binary arithmetic converter for the moduli set* $(2^k, 2^k - 1, 2^{k-1} - 1)$. Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on **45**(2), 204 (1998). 54

[90] Y. Wang, X. Song, M. Aboulhamid, and H. Shen. *Adder based residue to binary number converters for* $(2^n - 1, 2^n, 2^n + 1)$. Signal Processing, IEEE Transactions on **50**(7), 1772 (2002). 54, 59

[91] W. Wang, M. Swamy, and M. O. Ahmad. *Moduli selection in RNS for efficient VLSI implementation*. In *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*, vol. 4, pp. 512–515 (IEEE, 2003). 54

[92] A. Garcia, U. Meyer-Base, A. Lloris, and F. J. Taylor. *RNS implementation of FIR filters based on distributed arithmetic using field-programmable logic*. In *Circuits and Systems, 1999. ISCAS'99. Proceedings of the 1999 IEEE International Symposium on*, vol. 1, pp. 486–489 (IEEE, 1999). 54

[93] Y. Kong and B. Phillips. *Fast scaling in the residue number system*. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **17**(3), 443 (2009). 55, 61, 62

[94] U. Meyer-Base and T. Stouraitis. *New power-of-2 RNS scaling scheme for cell based IC design*. IEEE Transaction Very Large Scale Integr. (VLSI) Syst **11**(2), 280 (2003). 55, 61

[95] M. Griffin, F. Taylor, and M. Sousa. *New scaling algorithms for the chinese remainder theorem*. In *Signals, Systems and Computers, 1988. Twenty-Second Asilomar Conference on*, vol. 1, pp. 375–378 (IEEE, 1988). 55, 60, 61, 66, 82

[96] M. S. M. Griffin and F. Taylor. *Efficient scaling in the residue number system*. In *Int. Conf. Acoust., Speech, Signal Process., Glasgow, U.K.*, pp. 1075–1078 (1989). 60, 74

[97] H. O.Aichholzer. *A fast method for modulus reduction in residue number system.* In *Economicall parallel process, Vienna, Austria*, pp. 41–54 (1993). 55

[98] A. Halbutogullari and C. K. Koc. *Parallel multiplication in GF(2^k) using polynomial residue arithmetic.* Designs, Codes and Cryptography **20**(2), 155 (2000). 55, 56

[99] D. C. Feldmeier. *Fast software implementation of error detection codes.* IEEE/ACM Transactions on Networking (TON) **3**(6), 640 (1995). 55

[100] A. Skavantzos and F. J. Taylor. *On the polynomial residue number system [digital signal processing].* Signal Processing, IEEE Transactions on **39**(2), 376 (1991).

[101] A. Skavantzos and T. Stouraitis. *Polynomial residue complex signal processing.* Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on **40**(5), 342 (1993). 55

[102] S. Pontarelli, G.-C. Cardarilli, M. Re, and A. Salsano. *A novel error detection and correction technique for RNS based FIR filters.* In *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS'08. IEEE International Symposium on*, pp. 436–444 (IEEE, 2008). 55

[103] J. Chu and M. Benaissa. *Polynomial residue number system GF(2^M) multiplier using trinomials.* In *17th European Signal Processing Conference* (2009). 56

[104] M. Roshanzadeh, A. Ghaffari, and S. Saqaeeyan. *Using residue number systems for improving QoS and error detection & correction in wireless sensor networks.* In *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, pp. 1–5 (IEEE, 2011). 55, 56

[105] A. H. Navin, A. S. Khashandarag, A. R. Oskuei, and M. Mirnia. *A novel approach cryptography by using residue number system.* In *Computer Sciences and Convergence Information Technology (ICCIT), 2011 6th International Conference on*, pp. 636–639 (IEEE, 2011). 55

[106] M. Ciet, M. Neve, E. Peeters, and J.-J. Quisquater. *Parallel FPGA implementation of RSA with residue number systems-can side-channel threats be avoided?* In *Circuits and Systems, 2003 IEEE 46th Midwest Symposium on*, vol. 2, pp. 806–810 (IEEE, 2003). 55

[107] J. Chu and M. Benaissa. *Error detecting AES using polynomial residue number systems.* Microprocessors and Microsystems **37**(2), 228 (2013). 56

[108] D. Kheirandish and A. Safari. *Using one hot residue number system (OHRNS) for digital image processing.* In *AISP 2012* (2012). 56

[109] M. Labafniya and M. Eshghi. *An efficient adder/subtracter circuit for one-hot residue number system.* In *Electronic Devices, Systems and Applications (ICEDSA), 2010 Intl Conf on*, pp. 121–124 (IEEE, 2010). 56

[110] R. Farshidi, A. Habibi Zadnavin, and E. Gholami. *A novel multiple valued logic OHRNS adder circuit for modulo* $(r^n - 1)$*.* In *ADVCOMP 2010, The Fourth International Conference on Advanced Engineering Computing and Applications in Sciences*, pp. 166–170 (2010). 57

[111] S. J. Jassbi, M. Hosseinzadeh, S. Gorgin, and K. Navi. *One-hot multi-level residue number system.* IEEE EWDTS, Yerevan pp. 733–738 (2007). 57

[112] M. Hosseinzadeh, S. J. Jassbi, and K. Navi. *A novel multiple valued logic OHRNS modulo-*r^n *adder circuit.* International Journal of Electronics, Circuits and Systems **1**(4), 245 (2007). 57

[113] P. Pirsch and H.-J. Stolberg. *VLSI implementations of image and video multimedia processing systems.* Circuits and Systems for Video Technology, IEEE Transactions on **8**(7), 878 (1998). 57

[114] B. Vinnakota and V. V. B. Rao. *Fast conversion technique for binary-residue number system.* IEEE Trans, Circuits Syst. I, Fundam. Theory Appl. **41**(12), 927 (1994). 59

[115] G. Jullien. *Residue number scaling and other operations using ROM arrays.* Computers, IEEE Transactions on **C-27**(4), 325 (1978). 59

[116] M. Kameyama and T. Higuchi. *A new scaling algorithm in symmetric residue number system based on multiple-valued logic.* In *Proc. IEEE International Symposium on Circuits and Systems, Tokyo*, pp. 189–192 (1979).

[117] C. Su and H. Lo. *An algorithm for scaling and single residue error correction in the residue number system.* IEEE Trans. Computers **39**(8), 1053 (1990). 59

[118] K. H. O'Keefe and J. L. Wright. *Remarks on base extension for modular arithmetic.* Computers, IEEE Transactions on **100**(9), 833 (1973). 60

[119] G. A. Jullien. *Residue number scaling and other operations using ROM arrays.* Computers, IEEE Transactions on **100**(4), 325 (1978). 60

[120] F. J. Taylor and C. H. Huang. *A floating-point residue arithmetic unit.* Journal of the Franklin Institute **311**(1), 33 (1981). 60

[121] F. J. Taylor and C. H. Huang. *An autoscale residue multiplier.* Computers, IEEE Transactions on **100**(4), 321 (1982). 60

[122] D. D. Miller and J. N. Polky. *An implementation of the LMS algorithm in the residue number system.* Circuits and Systems, IEEE Transactions on **31**(5), 452 (1984). 60

[123] M. C. Z. D. Ulman and J. M. Zurada. *Effective RNS scaling algorithm with the chinese remainder theorem decomposition.* In *Proc. IEEE Pacific Rim Conf. Commun., Comput., Signal Process., Victoria, BC, Canada*, pp. 528–531 (1993). 61, 62, 66, 82

[124] F. Barsi and M. C. Pinotti. *Fast base extension and precise scaling in RNS for look-up table implementations.* Signal Processing, IEEE Transactions on **43**(10), 2427 (1995). 61, 74

[125] A. García and A. Lloris. *A look-up scheme for scaling in the RNS.* IEEE Trans. Comput. **48**, 748 (1999). 61, 81, 82

[126] M. Dasygenis, K. Mitroglou, D. Soudris, and A. Thanailakis. *A full-adder-based methodology for the design of scaling operation in residue number system.* Circuits and Systems I: Regular Papers, IEEE Transactions on **55**(2), 546 (2008). 61, 80, 81, 82

[127] P. Benardson. *Fast memoryless,over 64 bits, residue-to-binary convertor.* Circuits and Systems, IEEE Transactions on **32**(3), 298 (1985). 61, 62

[128] A. Safari and Y. Kong. *Simple, fast and synchronous hybrid scaling scheme for the 8-bit moduli set.* Journal of Emerging Trends in Computing and Information Sciences **3**(6), 949 (2012). 61

[129] D. Soudris, M. Dasygenis, K. Mitroglou, K. Tatas, and A. Thanailakis. *A full adder based methodology for scaling operation in residue number system, electronics, circuits and systems.* In 9th *International Conference on*, vol. 3, pp. 891–894 (2002). 61

[130] S. Ma, J. Hu, Y. Ye, L. Zhang, and X. Ling. *A 2^n scaling scheme for signed RNS integers and its VLSI implementation.* Sci, China, Inf.Sci **53**(1), 203 (2010). 61, 62

[131] A. Lindström, M. Nordseth, L. Bengtsson, and A. Omondi. *Arithmetic circuits combining residue and signed-digit representations.* In *Advances in Computer Systems Architecture*, pp. 246–257 (Springer, 2003). 62

[132] S. Ma, J. Hu, L. Zhang, and X. Ling. *An efficient RNS parity checker for moduli set $(2^n - 1, 2^n + 1, 2^{2n} + 1)$ and its applications.* Science in China Series F: Information Sciences **51**(10), 1563 (2008). 62

[133] K. Rosen. *Elementary Number Theory and its Applications* (Addison-Wesley Educational Publishers Inc, 2D, 1985), reprint ed. 64

[134] B. Parhami. *Computer Arithmetic Algorithm and Hardware Designs* (Oxford: Oxford University, Press, 2000). 66

[135] R. P. Brent and H. Kung. *The area-time complexity of binary multiplication.* Journal of the ACM (JACM) **28**(3), 521 (1981). 66

[136] H. Vergos, C. Efstathiou, and D. Nikolos. *Diminished-one modulo $2^n + 1$ adder design.* Computers, IEEE Transactions on **51**(12), 1389 (2002). 80

[137] G. Dimitrakopoulos, D. Nikolos, H. Vergos, D. Nikolos, and C. Efstathiou. *New architectures for modulo $2^n - 1$ adders.* In *Electronics, Circuits and Systems, 2005. ICECS 2005. 12th IEEE International Conference on*, pp. 1–4 (2005). 80

[138] Z. D. Ulman and M. Czyzak. *Highly parallel, fast scaling of numbers in nonredundant residue arithmetic.* IEEE Transaction on signal processing **46**(2), 487 (1998). 80

[139] M. J. F. S. Waser. *Introduction to Arithmetic for Digital Systems Designers* (New York: HRW, 1982). 81

[140] S. W. Smith *et al. The scientist and engineer's guide to digital signal processing* (1997). 109

[141] A. Safari, N. CV, and Y. Kong. *VLSI architecture of multiplier-less DWT image processor.* In *TENCON Spring Conference, 2013 IEEE*, pp. 280–284 (IEEE, 2013). 112

[142] H. T. Vergos, C. Efstathiou, and D. Nikolos. *Diminished-one modulo* 2^n+1 *adder design*. Computers, IEEE Transactions on **51**(12), 1389 (2002). 117

[143] O. Fatemi and S. Panchanathan. *VLSI architecture of a scalable matrix transposer*. In *Innovative Systems in Silicon, 1996. Proceedings., Eighth Annual IEEE International Conference on*, pp. 382–391 (IEEE, 1996). 118, 132

[144] S. Panchanathan. *Universal architecture for matrix transposition*. Computers and Digital Techniques, IEE Proceedings E **139**(5), 387 (1992).

[145] B. Bilgic, B. K. Horn, and I. Masaki. *Efficient integral image computation on the GPU*. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pp. 528–533 (IEEE, 2010).

[146] D. L. Zhang, Y. Yang, Y. K. Song, and G. M. Du. *Design and implement of large dimension matrix transpose based on DDR3 SDRAM*. Advanced Materials Research **760**, 1423 (2013).

[147] S. Hsu, A. Agarwal, M. Anders, S. Mathew, H. Kaul, F. Sheikh, and R. Krishnamurthy. *A 280mV-to-1.1 V 256b reconfigurable SIMD vector permutation engine with 2-dimensional shuffle in 22nm CMOS*. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pp. 178–180 (IEEE, 2012).

[148] J. C. Bowman and M. Roberts. *Adaptive matrix transpose algorithms for distributed multicore processors* (2013). 118

[149] C.-T. Hsu and J.-L. Wu. *Hidden digital watermarks in images*. Image Processing, IEEE Transactions on **8**(1), 58 (1999). 123

[150] G. Strang. *Wavelets and filter banks* (Wellesley Cambridge Press, 1996). 124

[151] U. Totzek, F. Matthiesen, S. Wohlleben, and T. Noll. *CMOS VLSI implementation of the 2D-DCT with linear processor arrays*. In *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*, pp. 937–940 (IEEE, 1990). 135, 193, 194, 195

[152] S.-i. Uramoto, Y. Inoue, A. Takabatake, J. Takeda, Y. Yamashita, H. Terane, and M. Yoshimoto. *A 100-MHz 2-D discrete cosine transform core processor*. IEICE TRANSACTIONS on Electronics **75**(4), 390 (1992). 135

[153] D. Gong, Y. He, and Z. Cao. *New cost-effective VLSI implementation of a 2-D discrete cosine transform and its inverse.* Circuits and Systems for Video Technology, IEEE Transactions on **14**(4), 405 (2004). 135, 136, 193, 194

[154] C. Cheng and K. K. Parhi. *High-speed VLSI implementation of 2-D discrete wavelet transform.* Signal Processing, IEEE Transactions on **56**(1), 393 (2008). 136

[155] X. Tian, L. Wu, Y.-H. Tan, and J.-W. Tian. *Efficient multi-input/multi-output VLSI architecture for two-dimensional lifting-based discrete wavelet transform.* IEEE transactions on computers **60**(8), 1207 (2011). 136

[156] B. K. Mohanty, A. Mahajan, and P. K. Meher. *Area-and power-efficient architecture for high-throughput implementation of lifting 2-D DWT.* Circuits and Systems II: Express Briefs, IEEE Transactions on **59**(7), 434 (2012). 136

[157] Synopsys, Inc. *Design vision user guide*, g-2012.06 ed. (2012). 142

[158] Synopsys, Inc. *Data Preparation for IC Compiler User Guide*, h-2013.03 ed. (2013). 143, 161

[159] C. Neau and K. Roy. *Optimal body bias selection for leakage improvement and process compensation over different technology generations.* In *Proceedings of the 2003 international symposium on Low power electronics and design*, pp. 116–121 (ACM, 2003). 158

[160] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada. *1 V power supply high-speed digital circuit technology with multithreshold-voltage CMOS.* Solid-State Circuits, IEEE Journal of **30**(8), 847 (1995). 159

[161] Synopsys, Inc. *VCS MX/VCS MXi user guide*, f-2011.12-sp1 ed. (2012). 168

[162] Synopsys, Inc. *PrimeTime PX user guide*, g-2012.06 ed. (2012). 173

[163] D. Slawecki and W. Li. *Dct/idct processor design for high data rate image coding.* Circuits and Systems for Video Technology, IEEE Transactions on **2**(2), 135 (1992). 193, 195

[164] SGS-THOMSON Microelectronics. *2-D Discrete Cosine Transform Image Processor*, product no. imsa121 ed.

[165] Y.-T. Chang and C.-L. Wang. *New systolic array implementation of the 2-D discrete cosine transform and its inverse*. Circuits and Systems for Video Technology, IEEE Transactions on **5**(2), 150 (1995).

[166] T. Kuroda, T. Fujita, S. Mita, T. Nagamatsu, S. Yoshioka, K. Suzuki, F. Sano, M. Norishima, M. Murota, M. Kako, *et al. A 0.9-V, 150-MHz, 10-mW, 4 mm^2, 2-D discrete cosine transform core processor with variable threshold-voltage (VT) scheme*. Solid-State Circuits, IEEE Journal of **31**(11), 1770 (1996). 195

[167] J. Hunter and J. V. McCanny. *Discrete cosine transform generator for VLSI synthesis*. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 5, pp. 2997–3000 (IEEE, 1998).

[168] R. Rambaldi, A. Ugazzoni, and R. Guerrieri. *A 35 μw 1.1 V gate array 8× 8 IDCT processor for video-telephony*. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 5, pp. 2993–2996 (IEEE, 1998). 195

[169] T.-H. Chen. *A cost-effective* 8×8 *2-D IDCT core processor with folded architecture*. Consumer Electronics, IEEE Transactions on **45**(2), 333 (1999). 193

[170] H.-C. Chang, J.-Y. Jiu, L.-L. Chen, and L.-G. Chen. *A low power* 8×8 *direct 2-D DCT chip design*. Journal of VLSI signal processing systems for signal, image and video technology **26**(3), 319 (2000). 193, 194, 195

[171] L. Fanucci and S. Saponara. *Data driven VLSI computation for low power DCT-based video coding*. In *Electronics, Circuits and Systems, 2002. 9th International Conference on*, vol. 2, pp. 541–544 (IEEE, 2002). 194

[172] J.-I. Guo and J.-C. Yen. *An efficient IDCT processor design for HDTV applications*. Journal of VLSI signal processing systems for signal, image and video technology **33**(1-2), 147 (2003). 193, 194

[173] G. A. Ruiz, J. A. Michell, and A. BurÃ^{3}n. *High throughput 2D DCT/IDCT processor for video coding*. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 3, pp. III–1036 (IEEE, 2005). 193, 194

[174] F. J. Díaz, A. M. Burón, and J. M. Solana. *Haar wavelet based processor scheme for image coding with low circuit complexity*. Computers & Electrical Engineering **33**(2), 109 (2007). 193, 194

[175] C. H. Ho, P. H. W. Leong, W. Luk, S. J. Wilton, and S. López-Buedo. *Virtual embedded blocks: A methodology for evaluating embedded elements in fpgas*. In *Field-Programmable Custom Computing Machines, 2006. FCCM'06. 14th Annual IEEE Symposium on*, pp. 35–44 (IEEE, 2006). 193

[176] A. Bandyopadhyay, J. Lee, R. W. Robucci, and P. Hasler. *MATIA: a programmable 80 μw/frame CMOS block matrix transform imager architecture*. Solid-State Circuits, IEEE Journal of **41**(3), 663 (2006). 194, 195

[177] A. Nilchi, J. Aziz, and R. Genov. *Focal-plane algorithmically-multiplying CMOS computational image sensor*. Solid-State Circuits, IEEE Journal of **44**(6), 1829 (2009). 194, 195

[178] D. Shah and C. Vithalani. *Fpga based hardware design flow of distributed arithmetic (DA) based 2D discrete wavelet transform (DWT) for the proposed image compression algorithm* (2013). 194