IMAGE FUSION HARDWARE IMPLEMENTATION WITH AN OPTIMIZED REDUCTION CIRCUIT

A Thesis

by

Furkan Aydın

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Electrical and Electronics Engineering

Özyeğin University
August 2018

Copyright © 2018 by Furkan Aydın

IMAGE FUSION HARDWARE IMPLEMENTATION WITH AN OPTIMIZED REDUCTION CIRCUIT

Approved by:

Assoc. Prof. H. Fatih Uğurdağ, Advisor
Department of Electrical and Electronics Engineering
Özyeğin University

Asst. Prof. Göktürk Poyrazoğlu
Department of Electrical and Electronics Engineering
Özyeğin University

Asst. Prof. Onur Demir
Department of Computer Engineering
Yeditepe University

Date Approved: 9 August 2018

To my family, friends and teachers

ABSTRACT

In this thesis, the hardware implementation of a real-time image fusion algorithm using a High-Level Synthesis (HLS) tool is presented. Image fusion combines two or more images through a color transformation process. Different applications may require different frames per second (fps) and/or resolution. Yet the specifics of the image-processing algorithm may frequently change causing redesign. If the target platform is Field Programmable Gate Array (FPGA), usually rapid yet optimized hardware implementation is required. All these requirements cannot be met only by HLS. Clever approaches in terms of architectural techniques such as unorthodox ways of pipelining, coding Register Transfer Level (RTL) generators instead of RTL, and creative ways of porting interface logic/software allowed us to meet the requirements outlined above. With all these in our arsenal, we were able to get 3 versions of the image fusion algorithm (with different fps and/or resolution) running on Intel Altera Cyclone IV and Arria 10 FPGAs in a fairly short amount of time. In the image fusion algorithm, there is a standard deviation calculation, implementation of which requires accumulation of all pixel values of each frame. Hence, we designed two novel pipelined reduction circuits (and hence their RTL generators) that are named Area-Efficient Reduction Circuit (AERC) and High-Speed Area-Efficient Reduction Circuit (HSAERC). We generated several reduction circuits (AERC and HSAERC) using our own Verilog RTL generators. AERC and HSAERC designs were implemented on both Xilinx Virtex-II Pro and Virtex-5 FPGAs, and their synthesis results were compared with state-of-the-art designs in the literature. AERC design is better than the existing designs in the literature in terms of area utilization. On the other hand, HSAERC design is better than other designs in terms of performance.

ÖZETÇE

Bu tezde, bir Yüksek-Seviyeli Sentez (HLS) aracı kullanarak gerçek zamanlı bir görüntü birleştirme algoritmasının uygulaması sunulmuştur. Görüntü birleştirme bir veya birden fazla görüntüyü bir renk dönüştürme yöntemi kullanarak birleştirir. Farklı uygulamalar için saniyedeki kare sayısı (fps) ve/veya çözünürlük gereksinimi farklı olabilir. Görüntü işleme algoritmasının özellikleri sıkça değişebileceğinden yeni tasarım yapmak gerekebilir. Hedef platform Sahada Programlanabilir Kapı Dizisi (FPGA) olduğunda genellikle hızlı hatta optimize edilmiş donanım gerçeklemesi gereklidir. Bütün bu gereksinimler, sadece HLS aracı ile karşılanamaz. Alışılmışın dışında boru hattı yöntemleri, Yazmaç Transfer Seviyesi (RTL) kodlama yerine RTL üreten araç kodlama ve mantık/yazılım arayüz bağlantısını yapmanın yaratıcı yolları, yukarıda belirtilen gereksinimleri karşılamamıza izin verir. Bütün bu yaklaşımlarla beraber, görüntü birleştirme algoritmasının 3 farklı versiyonunu (farklı fps ve/veya çözünürlükte) Intel Altera Cyclone IV ve Arria 10 FPGA üzerinde oldukça kısa zamanda çalıştırdık. Görüntü birleştirme algoritmasında standart sapma hesaplaması bulunmaktadır, gerçeklemesi her görüntü karesindeki tüm piksel değerlerinin toplanmasını gerektirir. Bu nedenle, Alan-Verimli İndirgeme Devresi (AERC) ve Hız-Alan Verimli İndirgeme Devresi (HSAERC) diye adlandırılan iki özgün indirgeme devresi (ve dolayısıyla RTL üreteçleri) tasarladık. Kendi Verilog RTL kod üreteçlerimizi kullanarak birçok indirgeme devreleri (AERC ve HSAERC) ürettik. AERC ve HSAERC devreleri, Xilinx Virtex-II Pro ve Virtex-5 FPGA'leri üzerinde gerçeklenmiştir ve sentez sonuçları literatürdeki en gelişmiş tasarımlarla karşılaştırıldı. AERC tasarımı alan kullanımı bakımından literatürdeki tasarımlara göre daha iyidir. Diğer yandan HSAERC tasarımı ise performans bakımından diğer tasarımlardan üstündür.

ACKNOWLEDGMENTS

I am grateful to my thesis advisor Assoc. Prof. H. Fatih Uğurdağ for his invaluable encouragement and support. I would like to thank Asst. Prof. Göktürk Poyrazoğlu and Asst. Prof. Onur Demir for serving on my thesis examination committee. I would also like to thank to all my labmates at nEMESys Lab at Özyeğin University as well as all my other friends for their help and support.

This work work has been done as part of a project jointly supported by TÜBİTAK ARDEB-EEEAG and European Union's Artemis Joint Undertaking. On TÜBİTAK side, the project was called "M-RIVA: Methodology development for Real-time Implementation of Video Algorithms on FPGAs" (project no. 114E343). On Artemis side, the project was part of a bigger project called "ALMARVI: Algorithms, Design Methods, and Many Core Execution Platform for Low-Power Massive Data-Rate Video and Image Processing" (project no. 621439).

During the course of this work, I was a full-time research assistant supported by the above mentioned TÜBİTAK project (PI: Assoc. Prof. H. Fatih Uğurdağ) between March 2016 and April 2017, and starting with August 2017 I have been supported by TÜBİTAK ARDEB-EEEAG 1003 project (PI: Prof. Murat Uysal, co-PI: Assoc. Prof. H. Fatih Uğurdağ) named "Innovative Optical Wireless Communication Technologies for 5G and Beyond" (project no. 215E311). Last but not least, I must acknowledge the tuition waiver I received from Özyeğin University throughout my studies as well as the great research and working environment the university provides.

Finally, I would like to express my gratitude to my family for their love and support. Without them, it would be impossible for me to complete this thesis.

TABLE OF CONTENTS

DEDICATION . . . . **iii**

ABSTRACT . . . . **iv**

ÖZETÇE . . . . **v**

ACKNOWLEDGMENTS . . . . **vi**

LIST OF TABLES . . . . **ix**

LIST OF FIGURES . . . . **x**

I INTRODUCTION **1**

1.1 Problem Statement 1

1.2 Image Fusion Overview 2

1.3 Reduction Circuit Overview 3

1.4 Contributions of the Thesis 4

1.5 Outline of the Thesis 4

II HARDWARE IMPLEMENTATION OF IMAGE FUSION **5**

2.1 Previous Work 5

2.2 Image Fusion Method 6

2.3 Image Fusion Hardware 12

2.3.1 Frame Pipelining 13

2.3.2 FPGA Architecture 13

2.3.3 Details of FPUs 15

2.3.4 Pixel Pipelining 18

2.3.5 Host Implementation 19

2.4 Tools and Techniques 20

III PIPELINED REDUCTION CIRCUITS **23**

3.1 Previous Work 23

3.2 Reduction Problem 24

3.3 Proposed Reduction Circuit Architectures 27

3.3.1 Interface 27

3.3.2 Area-Efficient Reduction Circuit 28

3.3.3 High-Speed and Area-Efficient Reduction Circuit 36

3.3.4 AERC and HSAERC with Multiplication Operation 37

3.4 Reduction Circuit Verilog Code Generator 38

IV RESULTS 40

4.1 Image Fusion Implementation Results 40

4.2 Reduction Circuit Implementation Results 42

V CONCLUSION 44

REFERENCES 46

VITA 49

LIST OF TABLES

1 FPU Properties of Cyclone IV FPGA 16

2 FPU Properties of Arria 10 FPGA 17

3 Image Fusion Design Results 40

4 Image Fusion Clock Speed Results 41

5 Resource Utilization Results of Image Fusion Design 41

6 Resource Utilization Proportion Results of Image Fusion Design . . . 41

7 Comparison of Reduction Circuit Designs 42

8 Implementation Results of Reduction Circuits for Virtex-II Pro FPGA 43

9 Implementation Results of Reduction Circuits for Virtex-5 FPGA . . 43

LIST OF FIGURES

1 False Color Image Composition Methodology 7
2 Example of False Color Image Composition 8
3 Structure of Image Fusion Method 9
4 Examples of Image Fusion with Different Target Images 11
5 Structure of Image Fusion Hardware 12
6 Schedule of (a) Sequential (b) Frame Pipelining Implementation . . . 14
7 Top-Level Block Diagram of the FPGA for Image Fusion Design . . . 15
8 Example of Pixel Pipelining (a) Not pipelined (b) Pipelined 19
9 Implementation Methodology 21
10 Simple Accumulation Architecture 25
11 Binary Tree Architecture 26
12 Interface of Reduction Circuit 27
13 Architectural Details of AERC 28
14 Structure of Control Mechanism 29
15 Control Algorithm for AERC Design 30
16 Example of Partial Result Reduction 31
17 AERC Design Example with k = 2, n = 8 32
18 AERC Design Example with k = 2, n = 9 32
19 AERC Design Example with k = 3, n = 8 33
20 AERC Design Example with k = 3, n = 9 33
21 AERC Design Example with k = 4, n = 8 34
22 AERC Design Example with k = 8, n = 8 35
23 Architectural Details of HSAERC 36
24 Clock Frequency Details of HSAERC 38

CHAPTER I

INTRODUCTION

This chapter introduces the problem, and gives an overview about image fusion and reduction circuit. It also includes summary of our contributions and outline of the thesis.

1.1 Problem Statement

The complexity of algorithms and large data sets in many real-time image processing applications leads to high performance platforms. Field Programmable Gate Arrays (FPGAs) provide a desirable platform to implement real-time image processing applications because FPGAs offer significant advantages in terms of parallel operation and reconfiguration [1].

Image processing algorithms comprise a great variety of tasks that require many computational operations on each pixel in each frame [2]. Moreover, the quality and the size of image data is constantly increasing. In order to achieve high video rates and resolutions, it is fundamental to process the image pixels simultaneously. However, the development time of the design becomes a significant issue for the FPGA programmer as the algorithm becomes more complex [3]. Consequently, hardware designers need to find new ways in order to complete their designs rapidly. Recently, the usage and popularity of High-Level Synthesis (HLS) tools increase in FPGA community [3] because it can translate high-level language codes into Hardware Description Language (HDL) synthesizable projects [4].

This thesis work has been done as part of a project jointly supported by TÜBİTAK ARDEB-EEEAG and European Union's Artemis Joint Undertaking. Algorithms, Design Methods, and Many Core Execution Platform for Low-Power Massive Data-Rate Video and Image Processing (ALMARVI) [5] was an ARTEMIS project which was supported by both TÜBİTAK and European Union. Özyeğin University and Aselsan were Turkish partners of ALMARVI project which had 16 partners from 4 different countries, namely, Netherlands, Finland, Czech Republic, and Turkey. In the scope of ALMARVI project, we performed hardware implementation of an image fusion algorithm [6]. Our special task was to implement it with 30 frames per second at a resolution of 1920x1080 pixels.

In this thesis, we used Multiplexing Aware Function and Register Scheduler (MAFURES) HLS tool [3] in order to implement the real-time image fusion algorithm rapidly. The algorithm includes standard deviation calculation that requires accumulation of all pixels of each frame. However, the MAFURES HLS tool we used was not sufficient to implement reduction operation. We decided to design our own reduction circuits; hence, we initially made pipelined reduction circuit generators. Our reduction circuit designs contribute significantly in terms of performance and area with respect to other designs in the literature.

1.2 Image Fusion Overview

Human visual system is very sophisticated and can be influenced easily by various factors such as sunlight, shading, reflectance, and composition [7]. Therefore, the human visual system cannot always interpret surrounding objects perfectly [7]. However, it is easier to identify the details of views in some circumstances. For example, daylight images are more favorable than night images in order to identify features of scene because color in daylight images provides much more information.

In real-world applications, more than one image is combined to produce a vision-enhancing output images [7] and this technique is called image fusion. The goal in image fusion is to reduce uncertainty and minimize redundancy in the output while enhancing comprehensive information [8]. It achieves this goal by processing the multiple images that are acquired by different sensors on a particular algorithm [9].

Image fusion has been used in many application areas. In medicine, it has recently been used for diagnostics and treatment of patients [10]. In security and surveillance, it used by both defense and commercial sectors for different objectives including wide area border protection, harbor surveillance, and systems that can detect, track and classify individual targets [11]. In remote or satellite area, it used to obtain proper view of satellite vision [10]. In this thesis, the presented work can be used for surveillance applications.

1.3 Reduction Circuit Overview

Reduction circuits that are used to reduce a set of numbers to a single value are used in a wide range of fields ([12], [13], [14]) for many scientific applications such as vector summation, dot product, and discrete cosine transform. Although FPGAs provide various function units in order to perform these applications, it is challenging to design a reduction circuit in an FPGA environment when high performance is necessity because the design needs to be pure pipelined and also consists of pipelined function units. Moreover, due to the fact that input vectors or matrices can be too large to buffer in the hardware, each vector item need to be read sequentially every clock cycle. This further complicates the design in order to prevent data hazards. Therefore, many reduction circuit architectures had been proposed in the literature. In this thesis, we also proposed two novel reduction circuit designs. Since high throughput reduction circuits are typically hand designed for specific vector lengths, these circuits need to be modified when the set lengths are changed; therefore, we wrote reduction circuit generators at HDL to rapidly generate reduction circuit designs that can perform any reduction circuit operations, such as floating-point addition and multiplication.

1.4 Contributions of the Thesis

The contributions to the literature are following:

- We proposed an HLS centered design strategy in order to implement an image fusion algorithm rapidly.
- We proposed two novel reduction circuit architectures.
- We obtained superior timing and area results for our reduction circuits.

1.5 Outline of the Thesis

Chapter II presents previously proposed image fusion implementations, used image fusion algorithm and its hardware implementation in detail. Chapter III presents related work, reduction circuit problem, and our novel reduction circuit architectures. Chapter IV shows the timing and area synthesis results for both image fusion designs and reduction circuit designs.

CHAPTER II

HARDWARE IMPLEMENTATION OF IMAGE FUSION

In this chapter, we first present previous work and used image fusion method. Then, we propose image fusion hardware in detail. We also explain used tools and techniques for the hardware implementation of the image fusion.

2.1 Previous Work

Image fusion has been performed using several approaches and presented in the literature ([15], [16], [17], [18]). In [19], Mohamed provides a detailed survey. In [15], Besiris proposed a hardware implementation of pixel-level color image fusion. Their technique is composed of covariance estimation, Cholesky decomposition and transformation. In [20], Qu proposed a real-time image fusion system uses both FPGA and multiDSP in order to perform algorithm. However, the performance result of the system is low because DSPs are used as an algorithm processor. In [21], Song proposed hardware implementation of real-time Laplacian pyramid image fusion. It was implemented on Virtex-4 SX35 FPGA. The performance of the system was 25 frames per second (fps) for 640x480 resolution images [21]. In [16], Sims proposed hardware implementation of pattern-selective pyramidal image fusion technique that finds and extracts all image characteristics from input images in order to combine them. It was implemented on Virtex-II XC2VP100 FPGA that fused greyscale 640x480 resolution images with 30 fps. In [22], Popovic proposed a real-time implementation of a multi-resolution image blending algorithm that uses multi-resolution decomposition of source images. The design which was implemented on Virtex-7 FPGA supports pipelining [22]. The performance result was 94 fps for HD 1080 resolution [22]. All of these designs ([16], [20], [21], [22]) are complex and require too much effort for implementation.

2.2 Image Fusion Method

In this section, the fundamental principles of image fusion algorithm ([6], [23]) proposed briefly.

Image recoloring and color transferring are most common tasks in image processing. Reinhard et al. [23] presented a method for transferring image characteristics from an image to another image. Toet [6] showed that this method could be applied to transfer the natural characteristics of daylight color image to (fused) multiband night vision image.

Fig. 1 shows that the method is employed to combine display of visual (400 - 700 nm) and near infrared (700 - 900 nm) intensified low-light CCD images and thermal middle wavelength band (3 - 5 μm) infrared images [6]. The visual light image is mapped to R channel of an RGB image in Fig. 2(a). The near-infrared light image is mapped to G channel of the RGB image in Fig. 2(b). And, the infrared light image is mapped to B channel of the RGB image in Fig. 2(c). As shown in Fig.2(d), this composition image is called false color image [6] that is also represented as a source image. The important point is that the images of all three channels should be the image of the same scene.

For the image fusion, a daylight image is called target image that is required to transfer its natural color to the source image operations are performed on pixels of these two source and target images. Structure of the image fusion method is given in Fig. 3.

Firstly, the false color image generation is performed combining three identical scene images that are obtained by different sensors to the proper channels. Then, the source and target images are transformed from the RGB to the $l\alpha\beta$ space in order to minimize the correlation between channels. Then, mean and standard deviation

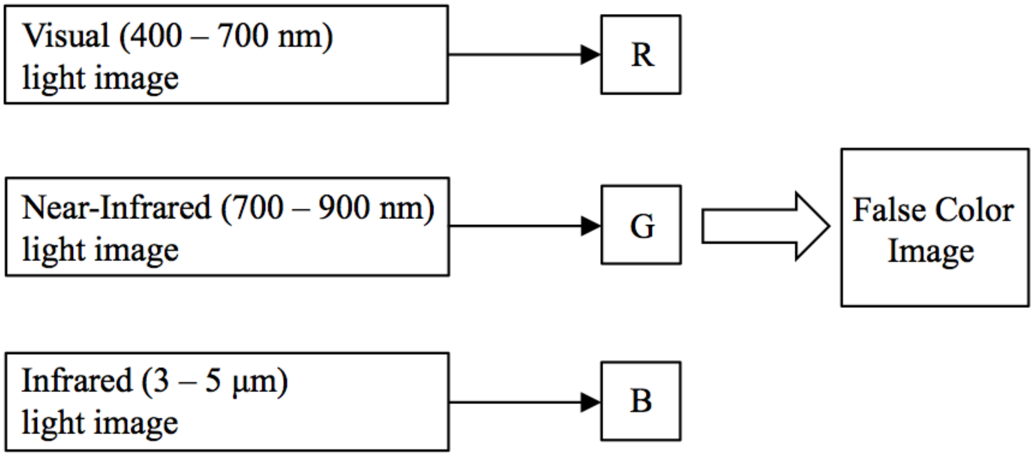

Visual (400 – 700 nm) light image
R
Near-Infrared (700 – 900 nm) light image
G
False Color Image
Infrared (3 – 5 μm) light image
B

Figure 1: False Color Image Composition Methodology

values of the source and target images are calculated. The calculated mean values are subtracted from each image pixel values. Thereafter, the source image is scaled with the ratio of the standard deviations of the source and target images. As a result of this process, a synthetic image is created and its standard deviation is conform to the standard deviation of the target image. Then, the mean value of the target image is added to the generated synthetic image pixel values. With this process, the mean value of the synthetic image becomes equal to the mean value of the target image. Finally, the synthetic image in $l\alpha\beta$ space is transformed into the RGB space.

The image fusion method proceeds as in the following [6]:

1. False color image is created and it is named as a source image. An image that is very similar to the color characteristic of the source daytime image is selected. This image is called the target image. The color characteristics of the target image are transferred to the source image.

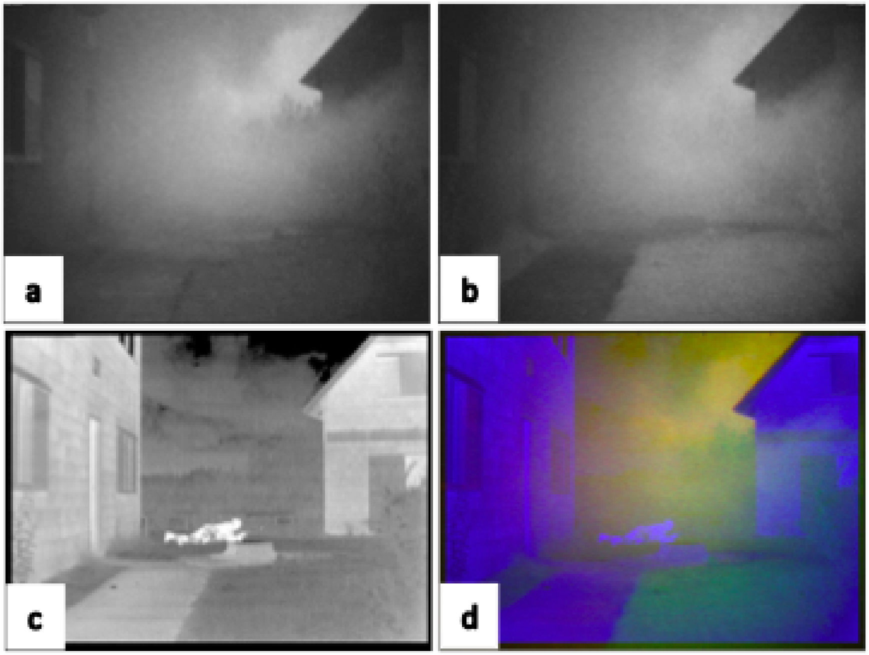


Figure 2: Example of False Color Image Composition

2. The source and the target image are transformed from RGB space to $l\alpha\beta$ space.

   (a) Transform RGB to LMS space.

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.3811 & 0.5783 & 0.0402 \\ 0.1967 & 0.7244 & 0.0782 \\ 0.0241 & 0.1288 & 0.8444 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}$$

   (b) Perform logarithm in order to eliminate a deal of skew.

$$\begin{aligned} L &= logL \\ M &= logM \\ S &= logS \end{aligned} \tag{2}$$

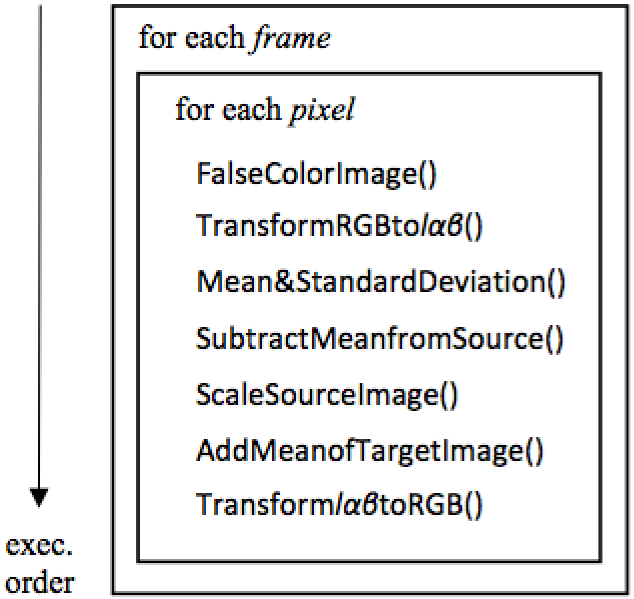


Figure 3: Structure of Image Fusion Method

3. Transform LMS to $l\alpha\beta$ space [6] to decorrelate the axes in the LMS space.

$$\begin{aligned}\begin{bmatrix} l \\ \alpha \\ \beta \end{bmatrix} &= \begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{6}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -2 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & -2\frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 0 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}\end{aligned} \tag{3}$$

4. The mean and standard deviations are calculated for the source and the target image.

5. The calculated mean values are subtracted from the source and target image data points.

$$\begin{aligned} l^* &= l - < l > \\ \alpha^* &= \alpha - < \alpha > \\ \beta^* &= \beta - < \beta > \end{aligned} \tag{4}$$

6. The source data points are scaled with the ratio of standard deviation of the source and the target image. At the end of this process, a synthetic image is created that is equal to the standard deviation of the target daylight image.

$$\begin{aligned} l'_s &= \frac{\sigma_t^l}{\sigma_t^l} l_s^* \\ \alpha'_s &= \frac{\sigma_t^\alpha}{\sigma_t^\alpha} \alpha_s^* \\ \beta'_s &= \frac{\sigma_t^\beta}{\sigma_t^\beta} \beta_s^* \end{aligned} \tag{5}$$

7. The mean of target image is added to the synthetic image; thus, the mean of the synthetic image is equalized to the mean of the target image.

8. Finally, the synthetic image in $l\alpha\beta$ space is transformed into RGB space.

    (a) $l\alpha\beta$ space to LMS space conversion

$$\begin{aligned} \begin{bmatrix} L \\ M \\ S \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -2 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{3} & 0 & 0 \\ 0 & \frac{\sqrt{6}}{6} & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} l \\ \alpha \\ \beta \end{bmatrix} \\ &= \begin{bmatrix} \frac{\sqrt{3}}{3} & \frac{\sqrt{6}}{6} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{3}}{3} & \frac{\sqrt{6}}{6} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{3}}{3} & -2\frac{\sqrt{6}}{6} & 0 \end{bmatrix} \begin{bmatrix} l \\ \alpha \\ \beta \end{bmatrix} \end{aligned} \tag{6}$$

(b) LMS space to RGB space conversion

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 4.4679 & -3.5873 & 0.1193 \\ -1.2186 & 2.3809 & -0.1624 \\ 0.0497 & -0.2439 & 1.2045 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix} \tag{7}$$

Fig. 4 shows varied synthetic images that are produced by using different daylight target images and a false color source image. The image in Fig. 4(a) represents source image (false color image). The images in Fig. 4(b) and in Fig. 4(d) are target images. The images in Fig. 4(c) and in Fig. 4(e) are result images.

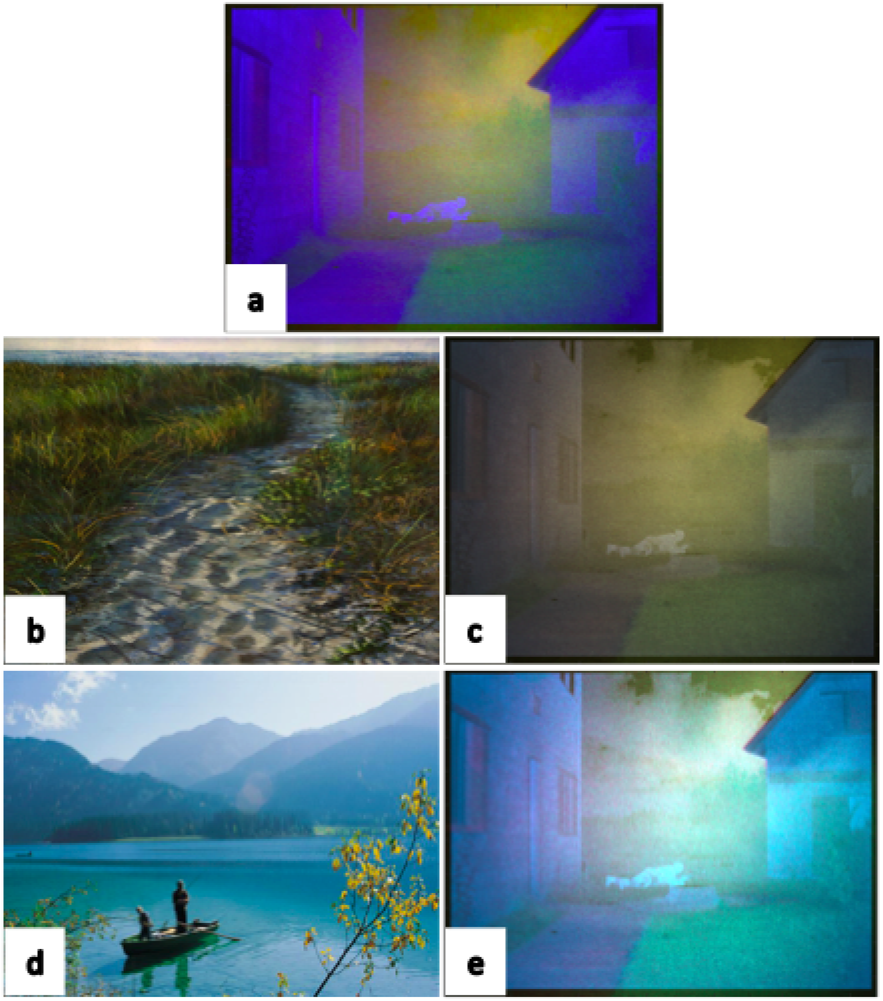


Figure 4: Examples of Image Fusion with Different Target Images

Standard Deviation Formula

The hardware implementation of standard deviation calculation that is given in (8) is challenging for designers because of data dependency issue. Hence, we used the mean of previous frame. In order accumulate pixel values, we used reduction circuit design that is explained in the next chapter.

$$\sigma(standard\_deviation) = \sqrt{\frac{1}{N}\sum_{x=1}^{N}(x_i - \frac{1}{N}\sum_{x=1}^{N} x_i)^2} \tag{8}$$

2.3 Image Fusion Hardware

The overall system is given in Fig. 5. A host processor is used to control video acquisition and display. It supplies video streams from a camera or storage to the FPGA over PCIe interface. The FPGA collects each frame data in its corresponding FIFOs in order to perform the image fusion process. After completing the image fusion process on the FPGA, each frame data is immediately transferred back to the FIFOs, and then to the Host. The Host collects video frames and transmits them to the FPGA, as well as captures processed frames from the FPGA and display the video on the monitor.

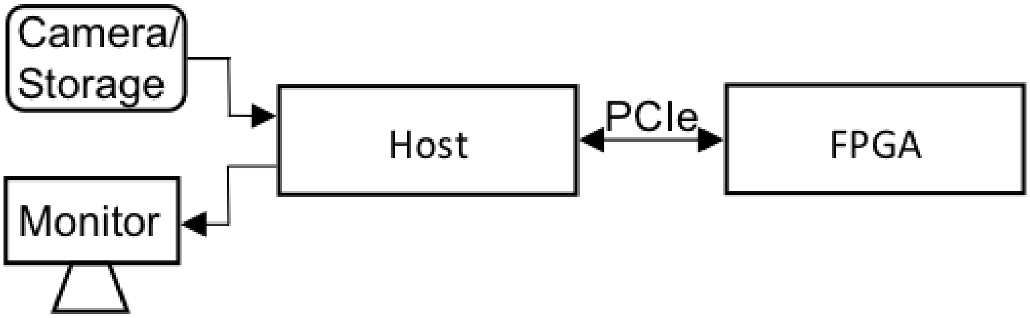


Figure 5: Structure of Image Fusion Hardware

2.3.1 Frame Pipelining

Fig. 6 shows the top-level schedule that is also our frame level schedule. In our hardware design, the current frame starts to be processed without waiting for the completion of the previous frame operations. This frame pipelining technique yields increment in number of processed frame per unit time. As shown in Fig. 6(b), Vin f2 starts before Vout f1 finishes in Fig. 6(b). Otherwise, the schedule of implementation is sequential like in Fig. 6(a). In order to make the figure easily readable, we shaded the block of odd numbered frames. We used Vin and Vout to indicate Video in and Video out. Vin and Vout are processes that are used to control the transaction between the Host, the FPGA and the PCIe interface. Each frame streamed out back to the Host after the image fusion process. In Fig. 6, Fs indicates the image fusion process on the FPGA.

2.3.2 FPGA Architecture

The structure of the design consists of PCIe interface, input and output FIFOs, wrapper, fusion main and standard deviation modules. Fig. 7 shows the top-level block diagram of the FPGA in the image fusion design. Pixel streaming is done over PCIe bus. SRAM FIFOs (made using FPGA Block RAM) provide pixel transaction between PCIe interface and the wrapper module to provide synchronization. The wrapper module controls input-output write and read enable signals between the fusion main and the standard deviation modules and also between input and output FIFOs.

The fusion main module is the most important module that contains all fusion method operations except standard deviation (8) because there is a data dependency issue. The standard deviation value of each frame is used once for each pixel scaling operation. Therefore, the fusion main module is needed of the standard deviation value of the current frame. However, this is quite costly because all pixels in the

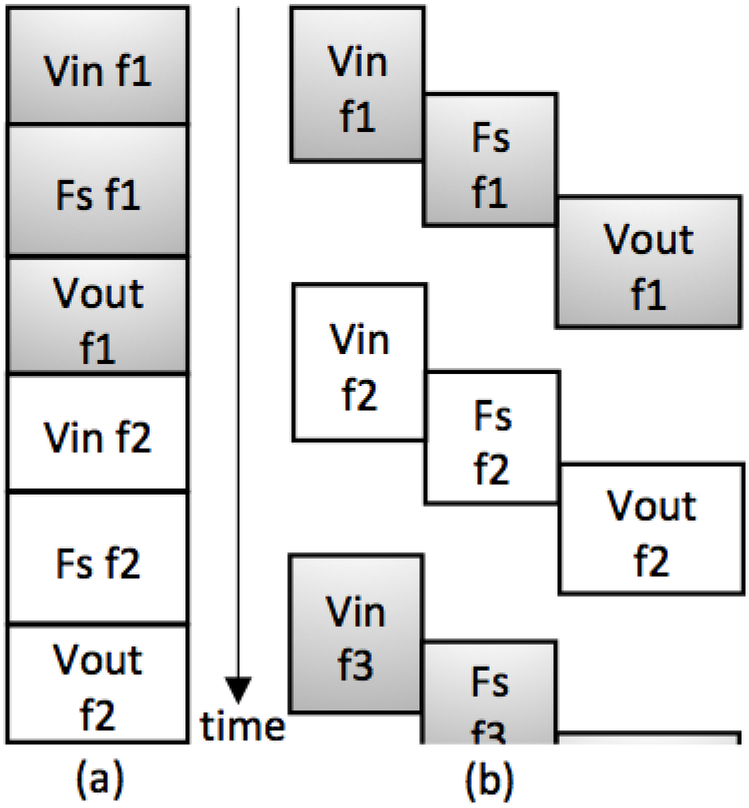


Figure 6: Schedule of (a) Sequential (b) Frame Pipelining Implementation

frame must be processed to calculate the standard deviation of the frame. Hence, we designed the standard Deviation module as a separate module that gives the standard deviation value of the previous frame to the fusion main module each time. The Wrapper module provides the connection between the Wrapper module and the fusion module.

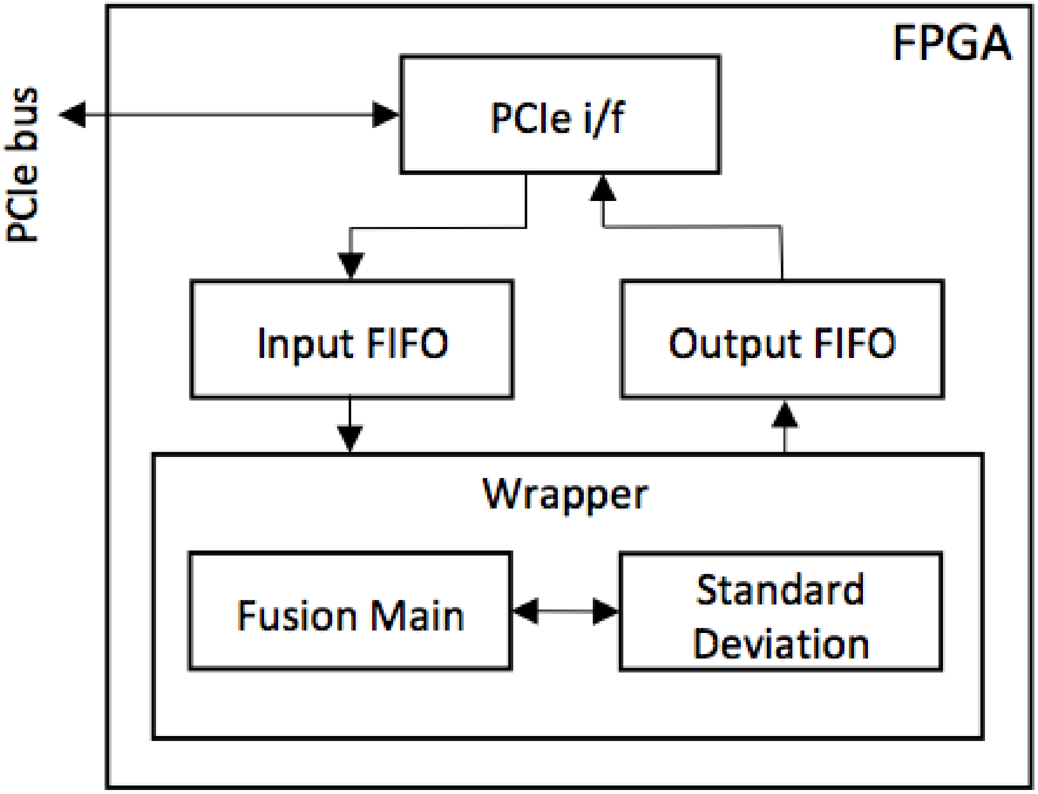


Figure 7: Top-Level Block Diagram of the FPGA for Image Fusion Design

FIFO Mechanism

Host sends pixel values to input FIFO via PCIe. If the input FIFO is not full, the pixel values are written to the input FIFO. If the input FIFO is not empty, the pixel values go to fusion process. After the fusion process, the pixel values are written to output FIFO. Finally, if the output FIFO is not empty, the pixel values go back to the Host via PCIe.

2.3.3 Details of FPUs

Altera has special floating-point IP cores to perform floating-point arithmetic operations. We created the necessary Floating-Point Units (FPUs) for the image fusion algorithm using Megawizard tool of Altera Quartus II. We determined the FPUs according to the fps and resolution values that we have targeted for the real-time image fusion application.

Our aim for the image fusion application was to process 30 frames per second at a resolution of 1920x1080 pixels. In order to achieve this goal, FPGA needs to be able to process 62,208,000 pixels per second. The calculation is given in (9).

$$\begin{aligned} Pixel\_number &= fps \cdot resolution \\ &= 30 \cdot 1920 \cdot 1080 \\ &= 62,208,000 \end{aligned} \tag{9}$$

The used FPU properties of Cyclone IV FPGA for the image fusion design are given in Table 1.

Table 1: FPU Properties of Cyclone IV FPGA

$FPU\_IP\_Core\_Name$	$Function\_Overview$	$f(MHz)$	$Latency(\#cycles)$
ADD_SUB	Adder/Subtracter	187	11
MULT	Multiplier	188	5
DIV	Divider	191	14
SQRT	Square Root	233	28
LOG	Natural Logarithm	184	21
EXP	Exponential	110	17
CONVERT (INT_to_FP)	Integer-to-Float	232	6
CONVERT (FP_to_INT)	Float-to-Integer	210	6

When we designed the system to work on Arria 10 FPGA, we created new FPUs because FPU properties for Arria 10 FPGA are different. The used FPU properties of Cyclone IV FPGA for the image fusion design are given in Table 2.

Table 2: FPU Properties of Arria 10 FPGA

$FPU\_IP\_Core\_Name$	$Function\_Overview$	$f(MHz)$	$Latency(\#cycles)$
ADD_SUB	Adder/Subtracter	521	11
MULT	Multiplier	371	5
DIV	Divider	449	25
SQRT	Square Root	844	28
LOG	Natural Logarithm	308	21
EXP	Exponential	804	50
CONVERT (INT_to_FP)	Integer-to-Float	299	6
CONVERT (FP_to_INT)	Float-to-Integer	488	6

Logarithm and Exponent FPUs

There are some logarithm operations in the image fusion algorithm. However, MegaWizard tool of Altera Quartus II does not support logarithm FPU with base 10. On the other hand, it supports natural logarithm. Hence, we used the formula that is given in (10) in order to perform logarithm operations with base 10.

$$log_{10}x = log_e x \cdot log_{10}e \tag{10}$$

In the above formula, we can find $log_{10}x$ on the left side of the equation by multiplying $log_e x$ and $log_{10}e$. Since $log_{10}e$ is a constant value, we only use FPU for $log_e x$ calculation.

We faced with similar problem for exponential operations because there is not exponent FPU with base 10. Therefore, we used similar methodology like logarithm calculation. We used the following formula that is given in (11) in order to perform 10^x.

$$10^x = e^{x \cdot log_e 10} \tag{11}$$

Convert FPUs

Since each pixel value that is read from the input FIFO is an integer value, we need to convert integer values to floating-point values. Hence, we used integer-to-float FPUs that operate each conversion with 6-clock cycle latency. Similarly, we used float-to-integer FPU to convert floating-point values to integer values before sending pixel values to the output FIFO. It also operates with 6-clock cycle latency. These latencies cause the system to increase its total latency. However, since our system is pipelined, it does not cause any problem.

Prevention of Not-a-Number (NaN) and Infinity Conditions

Output values of logarithm, square root and division FPUs can sometimes be NaN or Infinity. This affects our system adversely. For example, when input of the logarithm FPU is negative, output becomes NaN. Similarly, input of the square root FPU is negative or NaN, output becomes NaN. In order to prevent these undesirable conditions, we added control logic to check the signal value.

2.3.4 Pixel Pipelining

Fig. 8 shows our pixel-level pipelining that is similar to the frame pipelining. The left hand side of the Fig. 8 shows sequential execution without pipelining. The right hand side of the Fig. 8 shows the pipelined version of the order of computation in the Fig. 3. In the pipelined version, a new pixel operation starts before the current pixel is completed; hence, the time difference between the start time of pixel operation that is the most essential parameter of the design in terms of the throughput and fps is shorter than non-pipelined version. In addition to pixel-level pipelining, we used in our design the Altera floating-point IP cores that are also support pipelining.

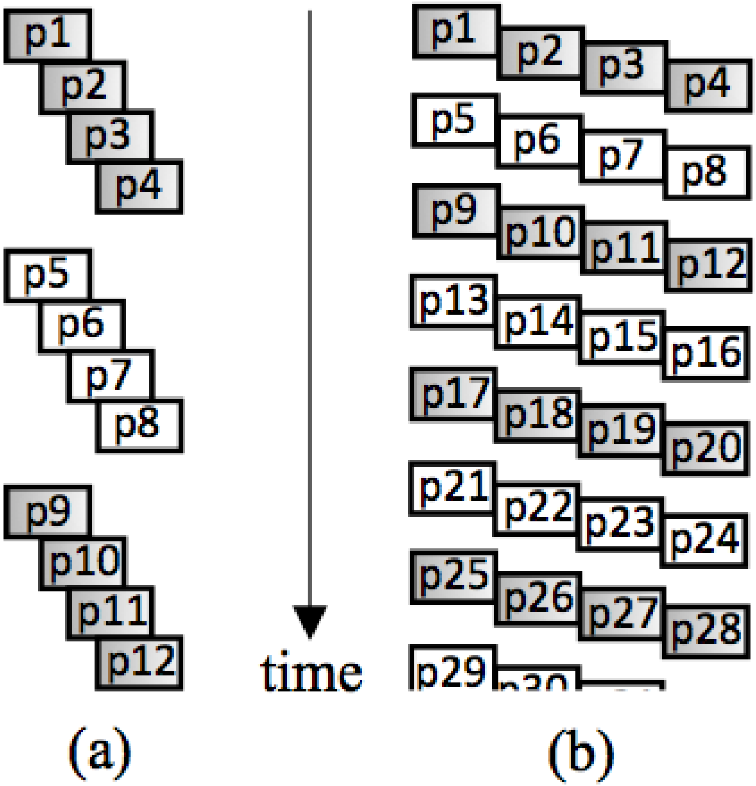


Figure 8: Example of Pixel Pipelining (a) Not pipelined (b) Pipelined

2.3.5 Host Implementation

The Host processor is running a software that performs different tasks simultaneously. The first task is capturing video from camera or storage (hard-disk). The second task is streaming video to the FPGA. Similarly, the third task is receiving the output stream from the FPGA to Host. The fourth task is displaying the output stream. These tasks in the Host form a pipeline with the blocks in the FPGA. If the software part of the system is not pipelined, the pipelined FPGA part will not work with high-throughput performance as in the pipeline. Hence, the systems cycle time (frame time) will be equal to the systems latency.

2.4 Tools and Techniques

Our overall implementation methodology is given in Fig. 9. Design refers to writing or generating the design in Verilog. Verification refers to simulation. After verification passes, we synthesize. Otherwise, we go back to the design step. If Synthesize does not meet timing or does not produce a design that fits the FPGA, then we go back to the design step. If synthesis passes successfully, we go to Validation, which refers to testing the design on the FPGA. If validation does not pass successfully, then we go back to the verification step.

HLS and Code Generation

We firstly determine the image fusion algorithm is suitable for MAFURES HLS tool. We code the algorithm in C++ language. This code is our golden reference for the fusion design. There are considerable numbers of arithmetic operations in the code. All operations are single-precision floating-point. However, there is a cyclic inter-iteration dependency in the algorithm. The standard deviation and the mean values should be calculated before they are subtracted from the each pixel values. Thus, we used the previous frame standard deviation and mean calculation for the current frame operations. However, this makes the algorithm complicated to generate the whole code in one through MAFURES HLS tool. Hence, we used a methodology like divide and conquer. We divide the design into two parts, the standard deviation and the fusion main operation module. The fusion main operation module is generated using MAFURES HLS tool. Accumulation parts of standard deviation module are generated using reduction circuit code generator and the others are hand-coded designs. Moreover, synchronization between the fusion main operation module and the standard deviation module is provided by hand-coded wrapper module.

In this work, we used Altera FPGAs as target devices and used Quartus II and Quartus Prime design software tools. These tools support floating-point IP cores

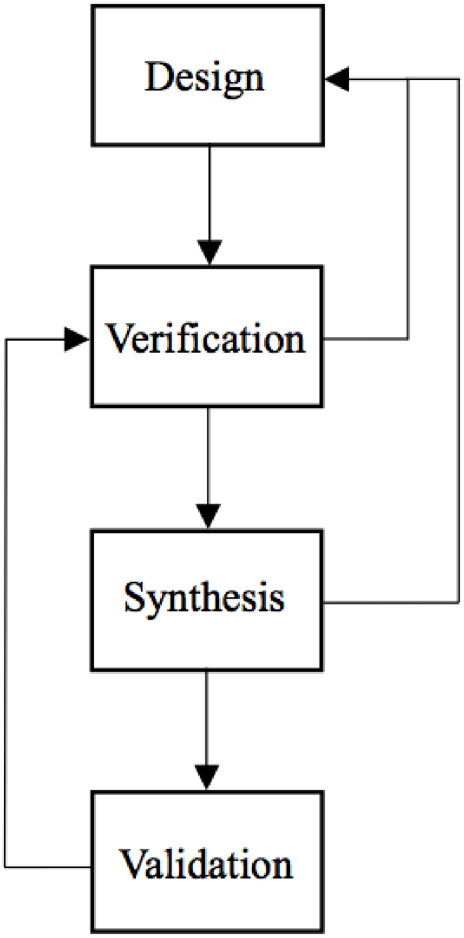


Figure 9: Implementation Methodology

for arithmetic operations. These cores support pipelining in similar manner as MAFURES HLS tool. We determine types of FPUs by analyzing our reference C++ code. Our fusion designs need seven different kinds of FPU. These are addition, subtraction, multiplication, division, square root, exponential, and logarithm FPUs. FPU modules that are used in the design should be declared into the FPU input file in the HLS tool. The name of the FPU operation, name of different modes, its latency, inputs/output port names and parameter settings should be included in the FPU module declarations file. The clock frequency of the design is affected by the clock frequency of the FPUs; hence we have chosen the FPUs to work at the maximum frequency. At that point, we enter the latency information of FPUs to MAFURES HLS tool as input. MAFURES also needs the information of Initiation Interval (II) that describes time between two consecutive loop iterations in terms of cycle. Ideally, II should be equal to 1 for the highest throughput. However, it cannot be possible due to resource constraints. In case of II more than one cycle, multiple operations can be assigned into single FPU with multiplexers. For the image fusion design, we initially specify the II value to be 3. Since there was no problem in terms of resource, we also implement the design by setting the initiation interval value to 1. We make only a few modifications on the wrapper module for both designs.

CHAPTER III

PIPELINED REDUCTION CIRCUITS

This chapter presents previous work of reduction circuits in the literature and introduces reduction problem. Also, it presents our proposed reduction circuit architectures and reduction circuit Verilog Register Transfer Level (RTL) code generator.

3.1 Previous Work

In the literature, there are many reduction circuit designs. Previously, proposed designs include various disadvantages in terms of area, performance, and implementation pitfall. They were designed to meet different requirements, some of which have beneficial features but some of which are unfavorable features. We will present a brief review in this section.

In [24], Kogge proposed a reduction method that uses lg(n) adders; however, design is not feasible when the input size are large because the design performs adder tree and requirement of adder usage is large. In [25], Ni and Hwang proposed a reduction design that use one adder and a fixed size buffer. Firstly, it reduces n inputs to α values, and then α values are reduced to a single value. This method stores input items in the memory because when reducing α values starts, it cannot accept new input item. Therefore, there is a large buffer requirement for large values of n. In [26], Luo and Martonasi proposed a self-alignment technique in order to implement a high-performance floating-point accumulator with using carry-save arithmetic in accumulator and delayed final adder. This technique that eliminates the interactions of the incoming number and the running sum makes pipeline possible. However, it is not fully pipelined because the accumulator needs to stall in order to prevent overflow. In [27] adapted self-alignment technique to implement single-precision floating-point multiply accumulator (FPMAC). The proposed design resolves scheduling restrictions between sequential FPMAC instructions and improve throughput with pipeline structure. In [28], Nagar and Bakos proposed a double precision accumulator design that reduces complexity of the control logic by integrating a coalescing reduction circuit within the low-level design of a base-converting floating-point adder. In [29], Zhou et al. proposed the fully compacted binary tree (FCBT), the dual striped adder (DSA) and the single striped adder (SSA). All of these designs can reduce multiple input sets of arbitrary sizes without stalling the pipeline. In these designs, the numbers of adders are fixed. The FCBT and the DSA uses two floating-point adders but SSA uses one floating-point adders. Further, the size of buffers is independent of the number of input sets for all of the three designs. The FCBT design needs background knowledge of the maximum number of element in a vector in advance. The DSA is more complex design with the lowest clock speed among the designs. Both DSA and SSA produce out-of-order outputs, when process variable size of vectors. In [30], Huang and Andrews proposed three different modular fully pipelined architectures. These are capable of performing reduction over data sets of arbitrary sizes without stalling, and generating results in order. The first one is named modular fully pipelined architecture (MFPA) that consists of ($\log_2$k+1) fully pipelined adders; however, these are not fully utilized. The second one is named area-efficient fully pipelined architecture (AeMFPA) that is better than MFPA in terms of area. It consists of two adders. The third one is named alternative design of AeMFPA (A^2eMFPA) that uses less buffer than AeMFPA.

3.2 Reduction Problem

A reduction circuit typically performs the same operation around the primitive operator in order to reduce given one or more set of numbers to a single value. One example of reduction circuit is an accumulator that performs a simple summation of

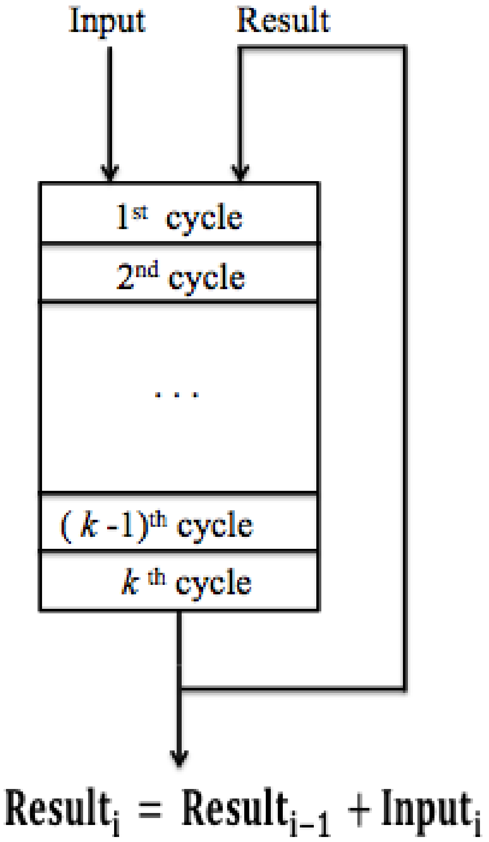


Figure 10: Simple Accumulation Architecture

the set of elements in Fig. 10. It uses a floating-point adder with pipeline latency k clock cycle that is larger than one clock cycle. There are k cycle stalls to obtain results for each addition. However, when new input value is fed to the accumulator in each cycle, data hazards will occur in this accumulation. Although a single-stage FPU works without the data hazard issue, its operating frequency is not high; therefore it is not desirable. Moreover, storing input values in FIFOs or registers may not work since it may cause buffer to overflow or pipeline stalls [31].

Reduction operations can be also implemented using a full binary tree that is given in Fig. 11. However, when the size of input element is high, we need more pipelined FPUs. Due to the resource constraints, full binary tree methodology is not desirable.

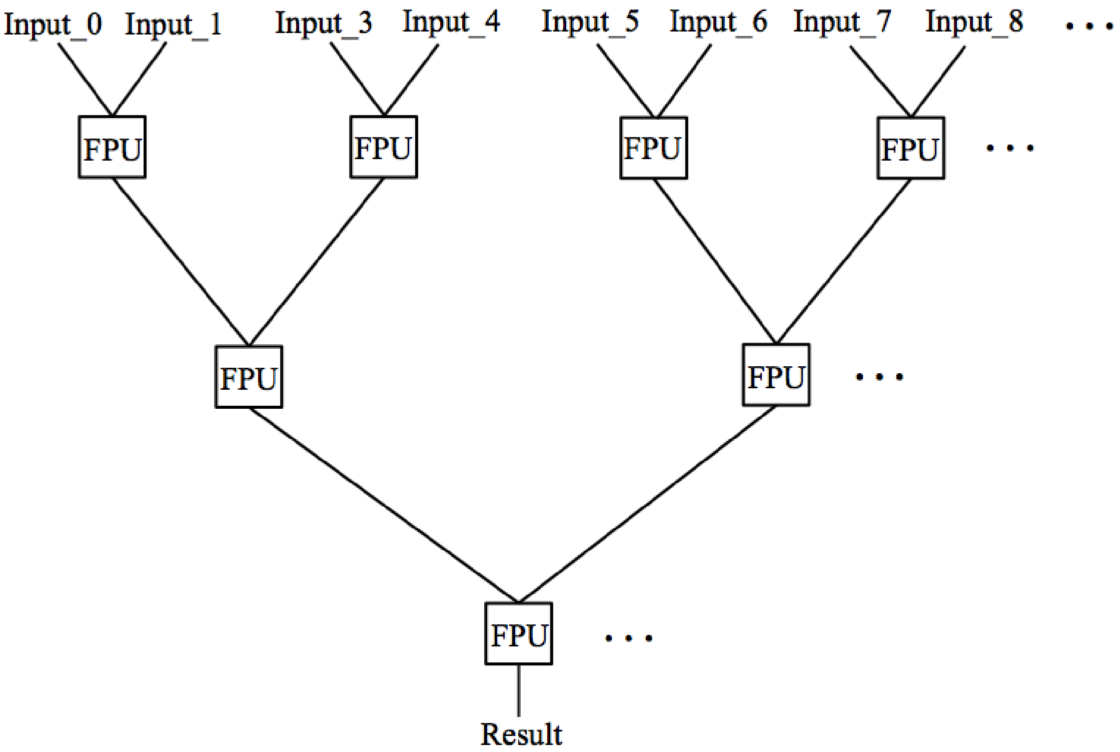


Figure 11: Binary Tree Architecture

Reduction Problem Formal Definition:

The reduction problem can be formally defined as follows:

Given a set S = $\{S_0, S_1 ... S_{n-1}\}$ of floating-point numbers, the problem is to reduce n values in the set into a single values without any data hazard. The used FPUs are pipelined with k pipeline stages and $k > 1$.

We met the following requirements to design our reduction circuits:

- Only one number enters to the FPU at each clock cycle.
- If the input is not ready, there will not be a problem. After all inputs are fetched into the FPU, they are reduced correctly without any data hazard.
- The uses of fixed number of FPUs are allowed, and they cannot stall.

3.3 Proposed Reduction Circuit Architectures

In this section, we first describe the interface of reduction circuit architectures. Then, we propose the AERC and HSAERC architectures in detail.

3.3.1 Interface

The interface of reduction circuit is shown in Fig. 12. The input and output interface of the reduction circuit consist of the following signals:

- *reset:* reset the status of registers
- *dataIn:* the input data item
- *inEn:* specify the validity of input items
- *result:* the reduced result
- *resultRdy:* specify the validity of the *result* signal

Since we produced reduction circuits using our code generators, there is no need to indicate the last item of data set as an input.

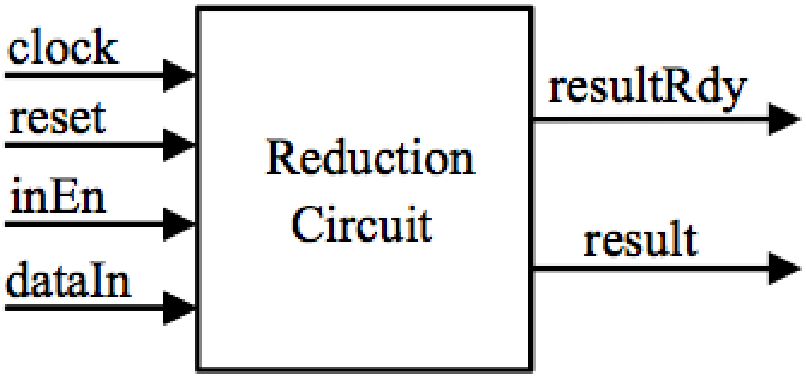


Figure 12: Interface of Reduction Circuit

3.3.2 Area-Efficient Reduction Circuit

The architecture of our area-efficient reduction circuit is shown in Fig. 13. It composed of one FPU, three 2-to-1 multiplexers (mux1, mux3 and mux4), one 4-to-1 multiplexer (mux2) and some control logic. The FPU performs summation with k pipeline stages. The control logic controls the statues of the multiplexer select signals and the resultRdy signal. The structure of control logic and the adder can vary depending on the size of input items entered into the code generator. The rest of the logic, multiplexers and one register are fixed in our proposed architecture.

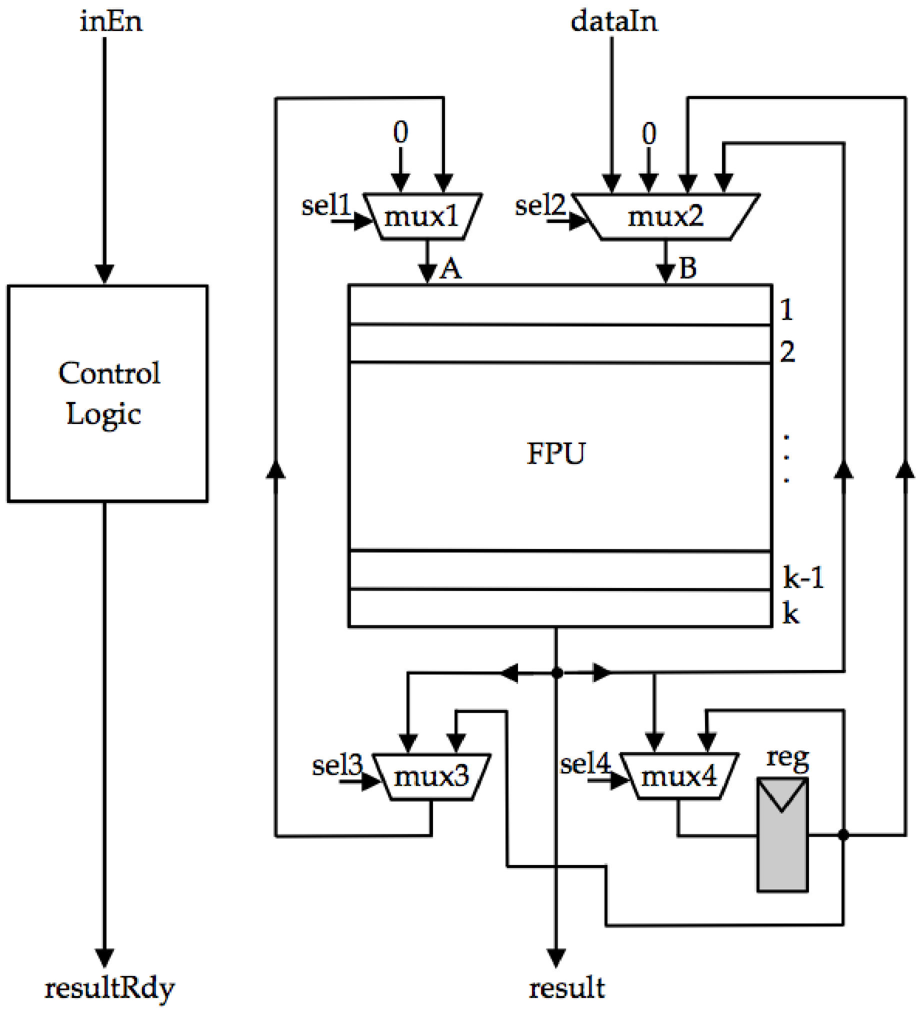


Figure 13: Architectural Details of AERC

The control logic is a little bit complicated; however, it does not pose any drawbacks in terms of design time because the designs can rapidly produced by the code generator. The control mechanism is given in Fig. 14 is structured as a basis of a 3-level reverse pyramid. At the level-1, 0 and input vector items (dataIn) are fed to the FPU. After k cycles, first partial result is obtained from the FPU that indicates the beginning of the level-2. At the level-2, partial result of the FPU and input vector items are fed to the FPU. When all input items are sent to the FPU, if modulo number of current cycle and k value is 0, the level-3 starts. However, unless modulo number of current cycle and k value is 0, the level-3 starts after k value and modulo number difference cycles. During this process, partial result and 0 values are fed to the FPU because all input vector items are finished. At the level-3, the partial results are fed to the FPU. However, partial results need to be registered because they will be operated with new partial results that will be obtained on next cycles.

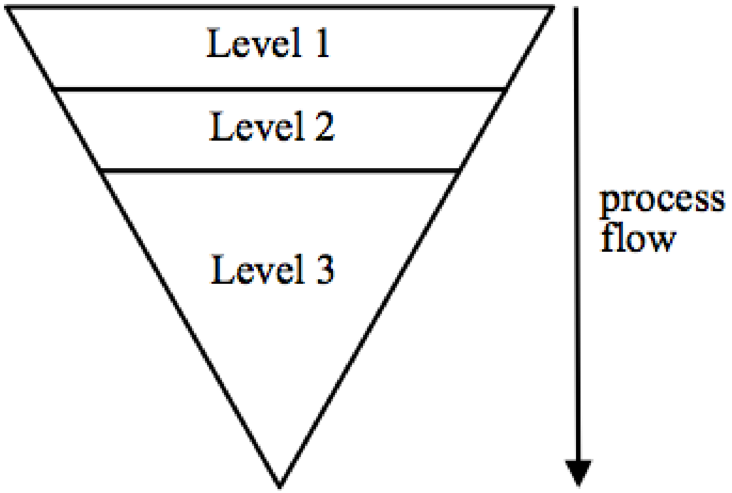
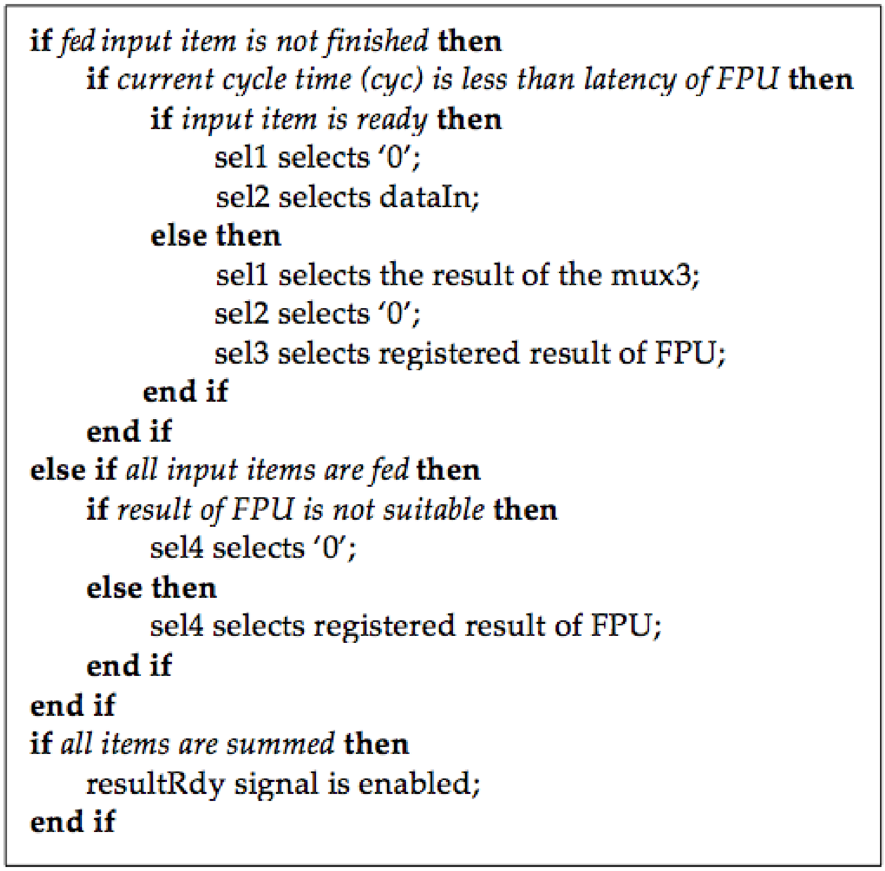


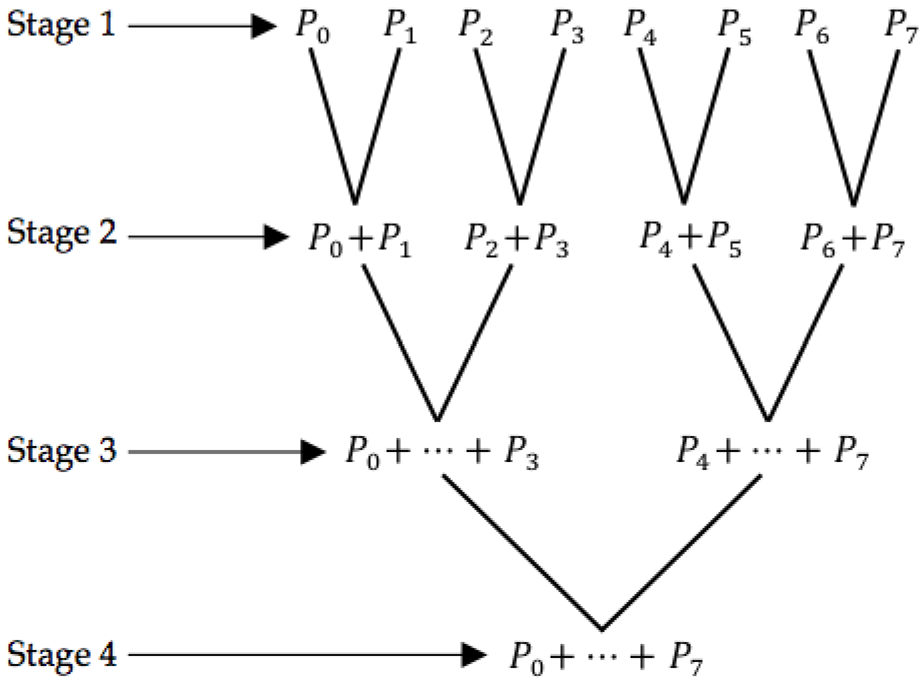
Figure 14: Structure of Control Mechanism

The control algorithm is given in Fig. 15. While the FPU performs operations without stalling, three selection controls sel0, sel1 and sel2 associated with the multiplexers are used to direct the data to the FPU. If the input data is not ready, the FPU still continues to operate. In order to prevent data hazard and keep result in order, 0 is fed to the input B of the FPU. After the ongoing significant operations in the FPU are completed, the result is registered until a new input is ready. Then, the registered result is processed with the new input in the FPU. This ensures that the final result of the reduction circuit is correct; however, this leads to increase in total latency per vector set.



```
if fed input item is not finished then
    if current cycle time (cyc) is less than latency of FPU then
        if input item is ready then
            sel1 selects '0';
            sel2 selects dataIn;
        else then
            sel1 selects the result of the mux3;
            sel2 selects '0';
            sel3 selects registered result of FPU;
        end if
    end if
else if all input items are fed then
    if result of FPU is not suitable then
        sel4 selects '0';
    else then
        sel4 selects registered result of FPU;
    end if
end if
if all items are summed then
    resultRdy signal is enabled;
end if
```

Figure 15: Control Algorithm for AERC Design



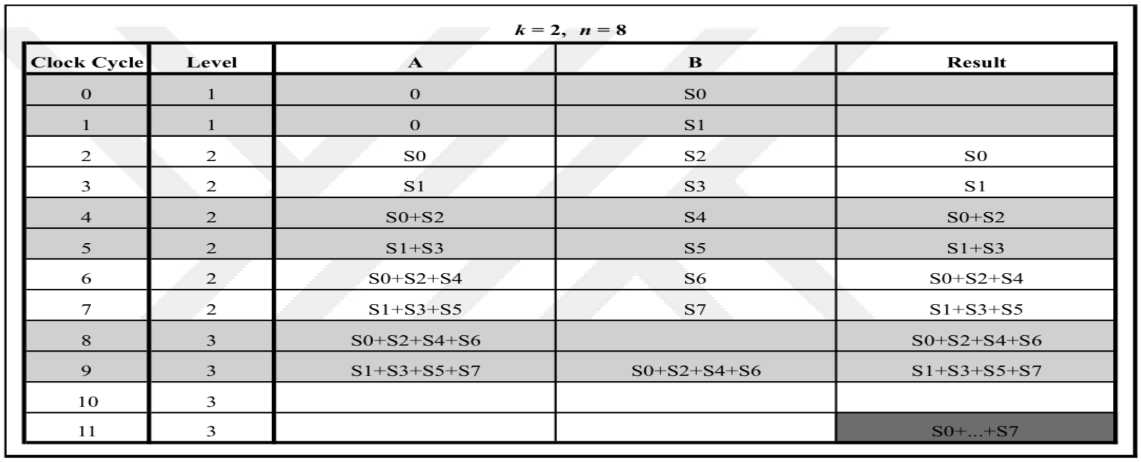
Stage 1
P_0 P_1 P_2 P_3 P_4 P_5 P_6 P_7
Stage 2
P_0+P_1 P_2+P_3 P_4+P_5 P_6+P_7
Stage 3
$P_0+\cdots+P_3$ $P_4+\cdots+P_7$
Stage 4
$P_0+\cdots+P_7$

Figure 16: Example of Partial Result Reduction

The AERC design reduces n inputs in equal or less than n + k - mod(n,k) + k($\log_2$k + 1) cycles. According to the structure of the control mechanism in Fig. 14, the design needs k cycles at level-1. At level-2, if mod(n,k) is equal to 0, it needs n - k cycles. However, unless mod(n,k) is equal to 0, it needs n cycles. At the level-3, it uses maximum k($\log_2$k + 2) cycles because it accumulates all partial results like a chain as shown in Fig. 16. In Fig. 16, 8 partial results are reduced with 4 stages using a chain of partial sums.

Examples of AERC design with different pipeline length and latency value are shown in Fig. 17-22. In Fig. 17, AERC design accumulates eight input items (n=8) using 2-pipeline stage (k=2) adder FPU. At the level-1, the input A of the FPU is 0, on the other hand, the input B of the FPU takes the input items each cycle. When the first result of the FPU emerges, the level-2 starts and it continues until all input items are finished. The level-3 starts at clock cycle 8 due to the fact that modulo number of current cycle (8) and k value (2) is 0 at clock cycle 8 when all input items are finished. On the other hand, in Fig. 18 , 0 is fed to input B of the FPU in the clock cycle 9 because modulo number of current cycle (9) and k value (2) is 1, hence, the level-3 starts at clock cycle 10. At the level 3, partial results of FPU are fed to both input A and input B of the FPU in order to accumulate all partial results.

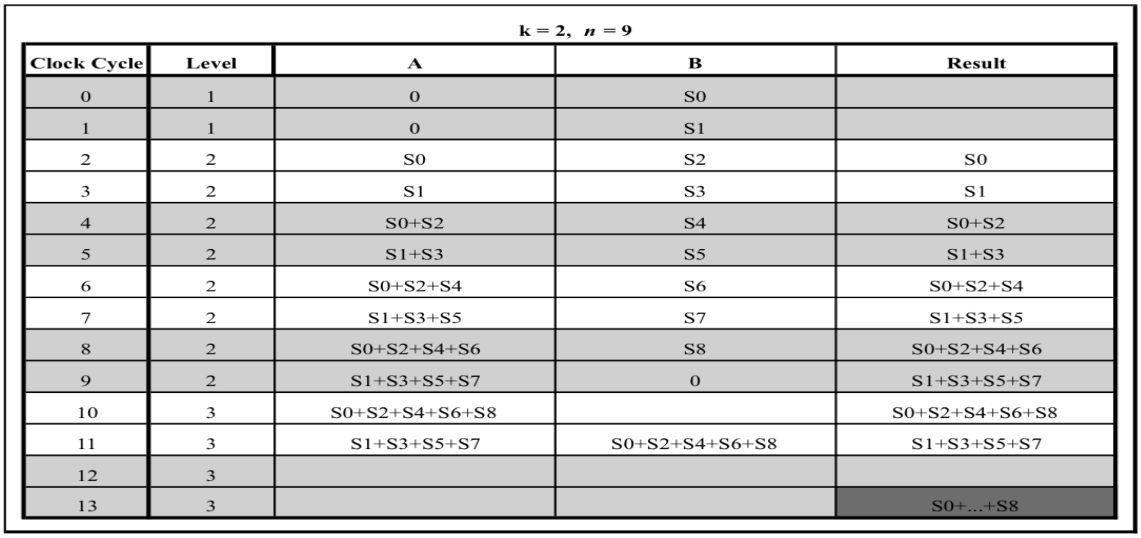
In the examples, some places are empty because they do not affect on our final results; therefore, we fed 0 to input or inputs of the FPU.



$k = 2,\ n = 8$

Clock Cycle	Level	A	B	Result
0	1	0	S0	
1	1	0	S1	
2	2	S0	S2	S0
3	2	S1	S3	S1
4	2	S0+S2	S4	S0+S2
5	2	S1+S3	S5	S1+S3
6	2	S0+S2+S4	S6	S0+S2+S4
7	2	S1+S3+S5	S7	S1+S3+S5
8	3	S0+S2+S4+S6		S0+S2+S4+S6
9	3	S1+S3+S5+S7	S0+S2+S4+S6	S1+S3+S5+S7
10	3			
11	3			S0+...+S7

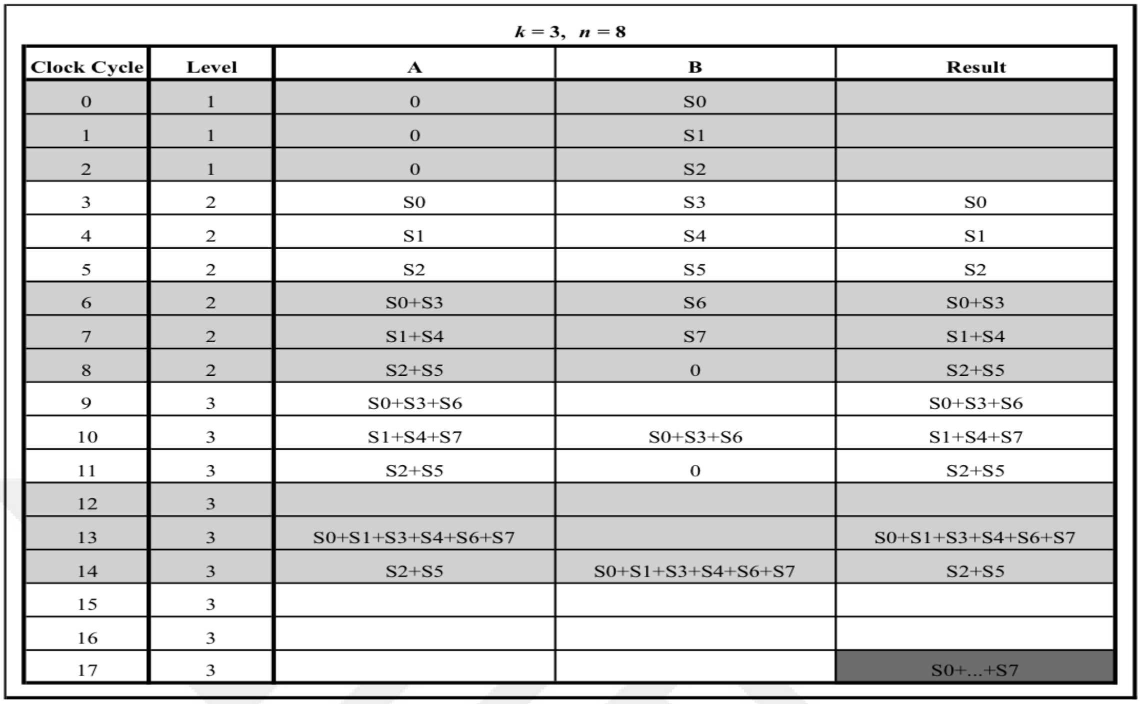
Figure 17: AERC Design Example with k = 2, n = 8



$k = 2,\ n = 9$

Clock Cycle	Level	A	B	Result
0	1	0	S0	
1	1	0	S1	
2	2	S0	S2	S0
3	2	S1	S3	S1
4	2	S0+S2	S4	S0+S2
5	2	S1+S3	S5	S1+S3
6	2	S0+S2+S4	S6	S0+S2+S4
7	2	S1+S3+S5	S7	S1+S3+S5
8	2	S0+S2+S4+S6	S8	S0+S2+S4+S6
9	2	S1+S3+S5+S7	0	S1+S3+S5+S7
10	3	S0+S2+S4+S6+S8		S0+S2+S4+S6+S8
11	3	S1+S3+S5+S7	S0+S2+S4+S6+S8	S1+S3+S5+S7
12	3			
13	3			S0+...+S8

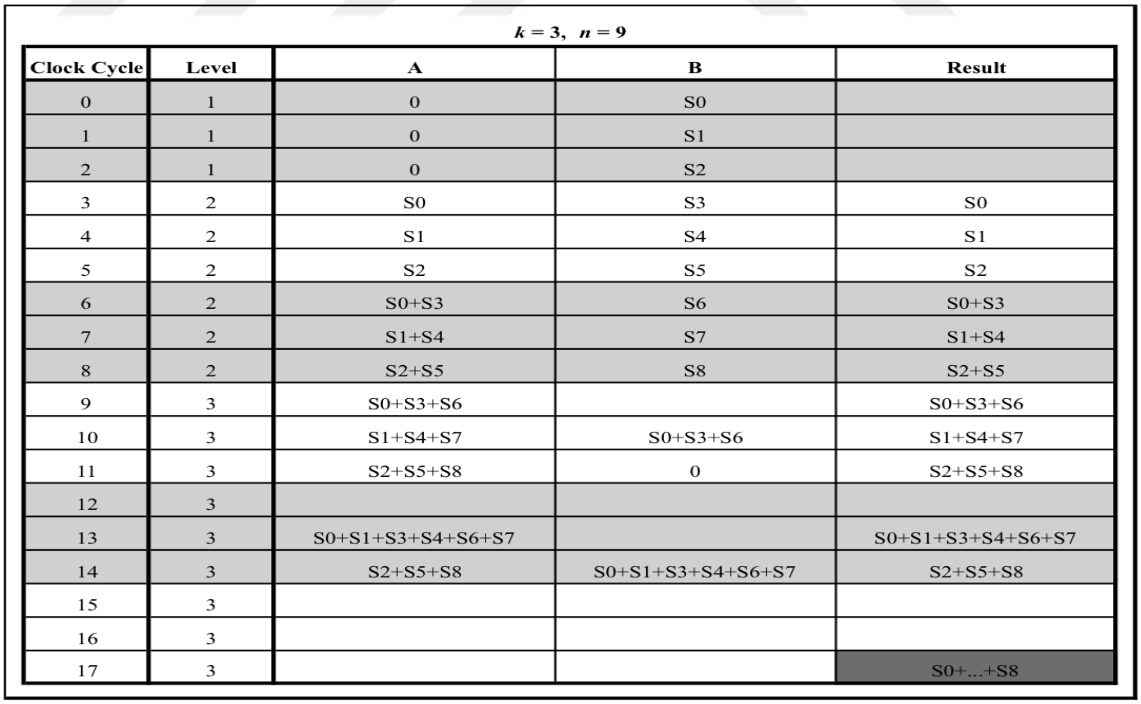
Figure 18: AERC Design Example with k = 2, n = 9



k = 3, *n* = 8

Clock Cycle	Level	A	B	Result
0	1	0	S0	
1	1	0	S1	
2	1	0	S2	
3	2	S0	S3	S0
4	2	S1	S4	S1
5	2	S2	S5	S2
6	2	S0+S3	S6	S0+S3
7	2	S1+S4	S7	S1+S4
8	2	S2+S5	0	S2+S5
9	3	S0+S3+S6		S0+S3+S6
10	3	S1+S4+S7	S0+S3+S6	S1+S4+S7
11	3	S2+S5	0	S2+S5
12	3			
13	3	S0+S1+S3+S4+S6+S7		S0+S1+S3+S4+S6+S7
14	3	S2+S5	S0+S1+S3+S4+S6+S7	S2+S5
15	3			
16	3			
17	3			S0+...+S7

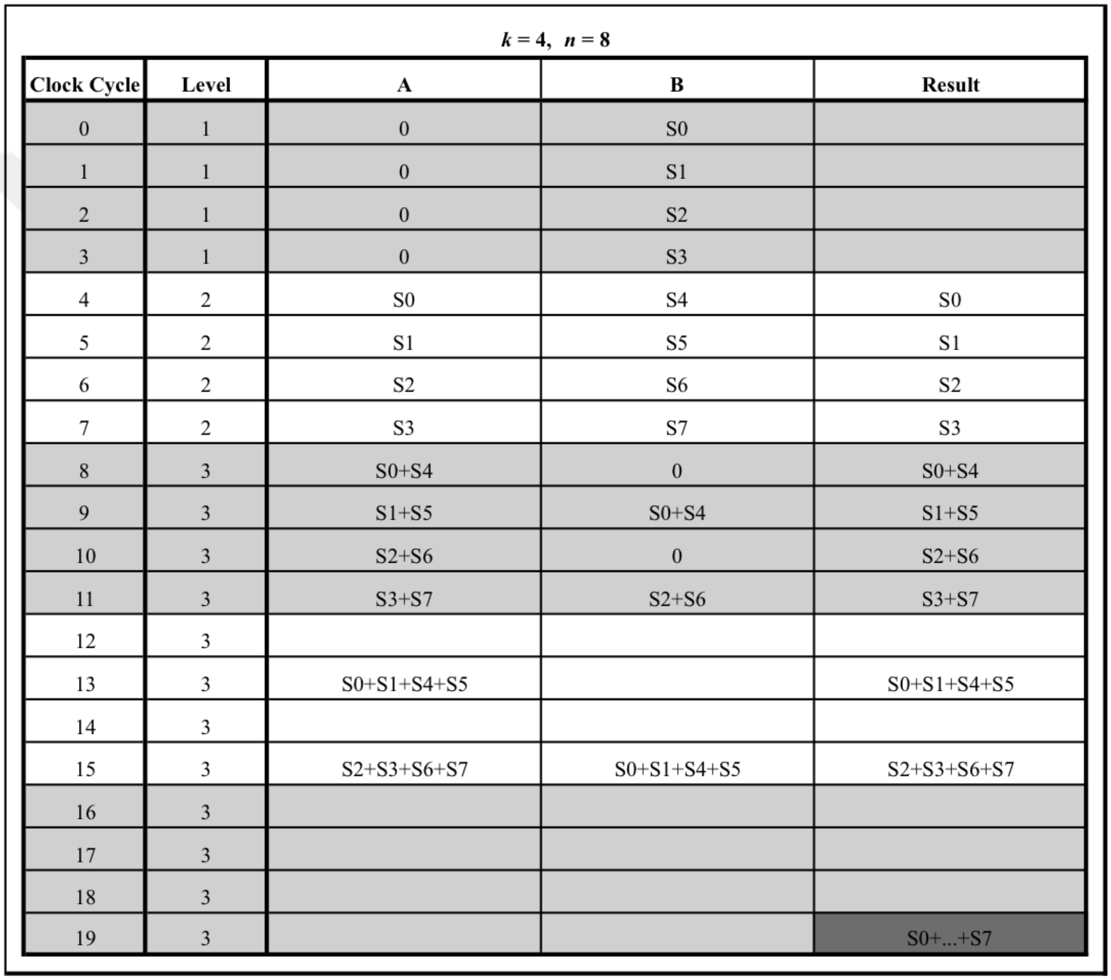
Figure 19: AERC Design Example with k = 3, n = 8



k = 3, *n* = 9

Clock Cycle	Level	A	B	Result
0	1	0	S0	
1	1	0	S1	
2	1	0	S2	
3	2	S0	S3	S0
4	2	S1	S4	S1
5	2	S2	S5	S2
6	2	S0+S3	S6	S0+S3
7	2	S1+S4	S7	S1+S4
8	2	S2+S5	S8	S2+S5
9	3	S0+S3+S6		S0+S3+S6
10	3	S1+S4+S7	S0+S3+S6	S1+S4+S7
11	3	S2+S5+S8	0	S2+S5+S8
12	3			
13	3	S0+S1+S3+S4+S6+S7		S0+S1+S3+S4+S6+S7
14	3	S2+S5+S8	S0+S1+S3+S4+S6+S7	S2+S5+S8
15	3			
16	3			
17	3			S0+...+S8

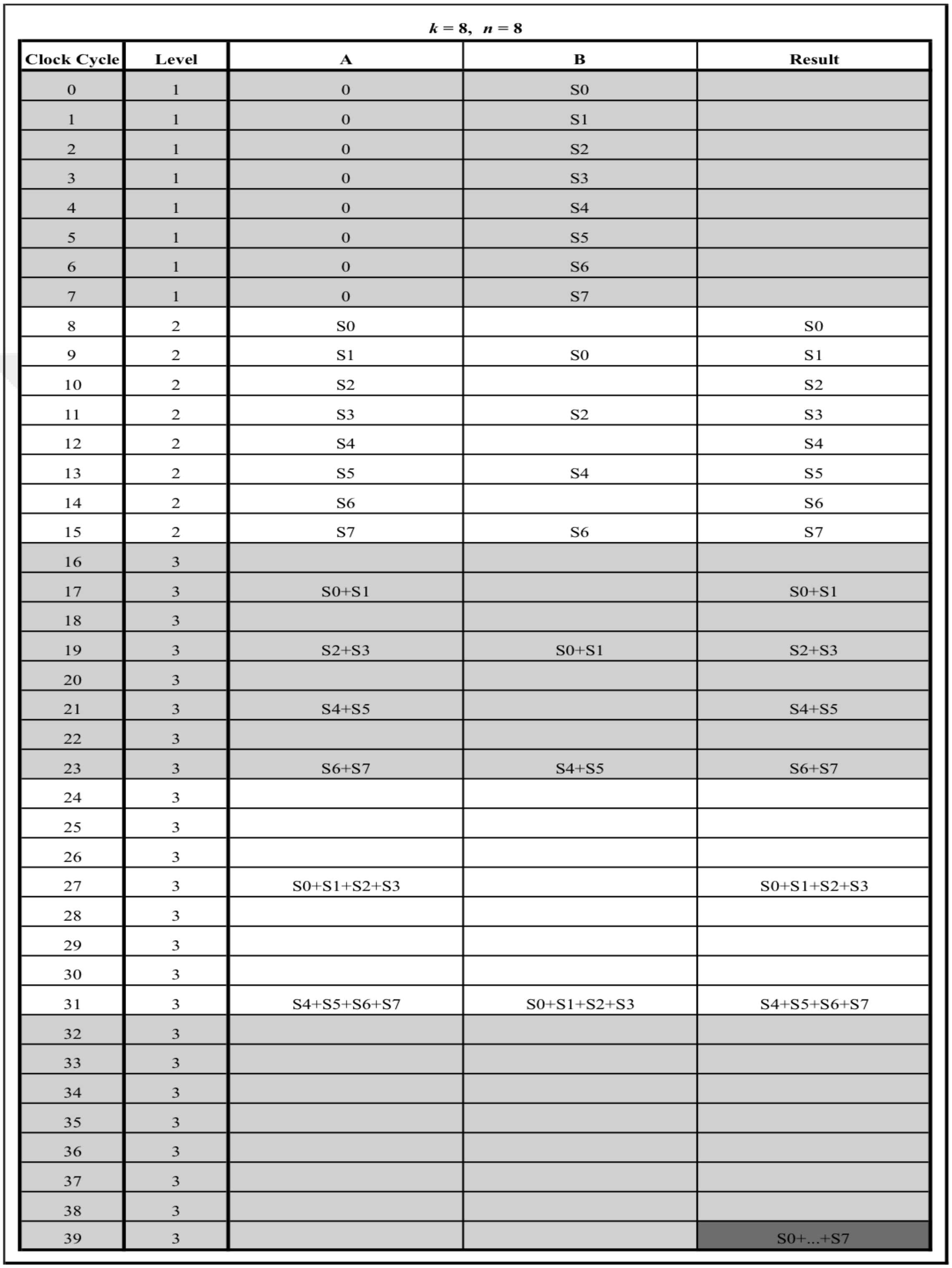
Figure 20: AERC Design Example with k = 3, n = 9



***k* = 4, *n* = 8**

Clock Cycle	Level	A	B	Result
0	1	0	S0	
1	1	0	S1	
2	1	0	S2	
3	1	0	S3	
4	2	S0	S4	S0
5	2	S1	S5	S1
6	2	S2	S6	S2
7	2	S3	S7	S3
8	3	S0+S4	0	S0+S4
9	3	S1+S5	S0+S4	S1+S5
10	3	S2+S6	0	S2+S6
11	3	S3+S7	S2+S6	S3+S7
12	3			
13	3	S0+S1+S4+S5		S0+S1+S4+S5
14	3			
15	3	S2+S3+S6+S7	S0+S1+S4+S5	S2+S3+S6+S7
16	3			
17	3			
18	3			
19	3			S0+...+S7

Figure 21: AERC Design Example with k = 4, n = 8



***k* = 8, *n* = 8**

Clock Cycle	Level	A	B	Result
0	1	0	S0	
1	1	0	S1	
2	1	0	S2	
3	1	0	S3	
4	1	0	S4	
5	1	0	S5	
6	1	0	S6	
7	1	0	S7	
8	2	S0		S0
9	2	S1	S0	S1
10	2	S2		S2
11	2	S3	S2	S3
12	2	S4		S4
13	2	S5	S4	S5
14	2	S6		S6
15	2	S7	S6	S7
16	3			
17	3	S0+S1		S0+S1
18	3			
19	3	S2+S3	S0+S1	S2+S3
20	3			
21	3	S4+S5		S4+S5
22	3			
23	3	S6+S7	S4+S5	S6+S7
24	3			
25	3			
26	3			
27	3	S0+S1+S2+S3		S0+S1+S2+S3
28	3			
29	3			
30	3			
31	3	S4+S5+S6+S7	S0+S1+S2+S3	S4+S5+S6+S7
32	3			
33	3			
34	3			
35	3			
36	3			
37	3			
38	3			
39	3			S0+...+S7

Figure 22: AERC Design Example with k = 8, n = 8

3.3.3 High-Speed and Area-Efficient Reduction Circuit

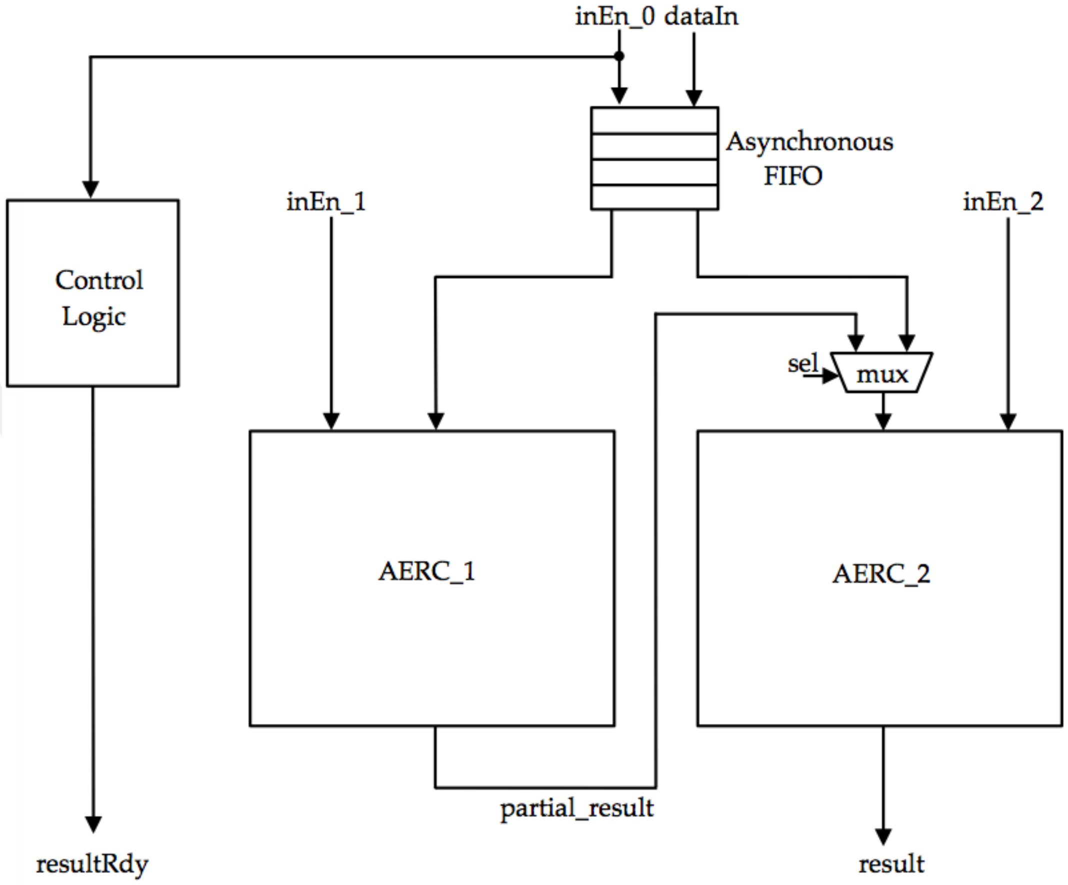


Figure 23: Architectural Details of HSAERC

HSAERC architecture is developed to achieve better performance results than AERC architecture in terms of clock frequency. The idea is based on using two prior AERC reduction circuits concurrently; hence, unlike the AERC architecture, our second design uses two FPUs. However, we keep the same interface as shown in Fig. 12 for the design. The architecture of HSAERC is shown in Fig. 23. It consists of one asynchronous FIFO, two same AERC designs (AERC_1 and AERC_2), and some control logics. The challenge of HSAERC design is to feed data to two AERC designs properly; therefore, we used an asynchronous FIFO. The asynchronous FIFO stores the input items and then feeds them into two AERC design modules consecutively.

When reduction modules finish all operations, the result of AERC_1 module enters the AERC_2 module as an input to produce the final result. Multiplexer controls AERC_2 module input data.

Each AERC module in the HSAERC architecture uses its own control mechanism that we have already explained. However, the HSAERC architecture contains control logic in order to control the input enable signals (inEn_1 and inEn_2) of each AERC module. This control logic controls status of sel signal of multiplexer (mux) that is given in Fig. 23. When partial result is ready, the status of sel signal changes and it allows the partial results to enter the AERC_2 module.

The clock frequency of asynchronous FIFO is twice as fast as the AERC modules, as shown in Fig. 24. inEn_0, inEn_1 and inEn_2 signals represent the enable signals that approve the entry of the vector items into the modules. There is a half-period phase shift between inEn_1 and inEn_2 signals in order to prevent buffer underflow case. The size of the asynchronous FIFO is Wx4. The data width of the asynchronous FIFO is W in which W is operand precision of the reduction circuit.

The HSAERC design reduces n inputs in equal or less than (n + (k - mod(n,k)))/2 + k($\log_2$k + 3) cycles. Each AERC module in the HSAERC architecture reduces vector sets 2 times faster than previous AERC design. That is, 2 divides total latency. However, there are also k cycles need to reduce results of AERC_1 and AERC_2 modules.

3.3.4 AERC and HSAERC with Multiplication Operation

The proposed AERC and HSAERC designs, which can be implemented with different type of arithmetic operations, require minor changes. In this section, we explain required modifications to implement reduction circuits with multiplication. The implementation of reduction circuits with multiplication demands only following changes:

- The FPU needs to be replaced by fully pipelined multiplier FPUs.

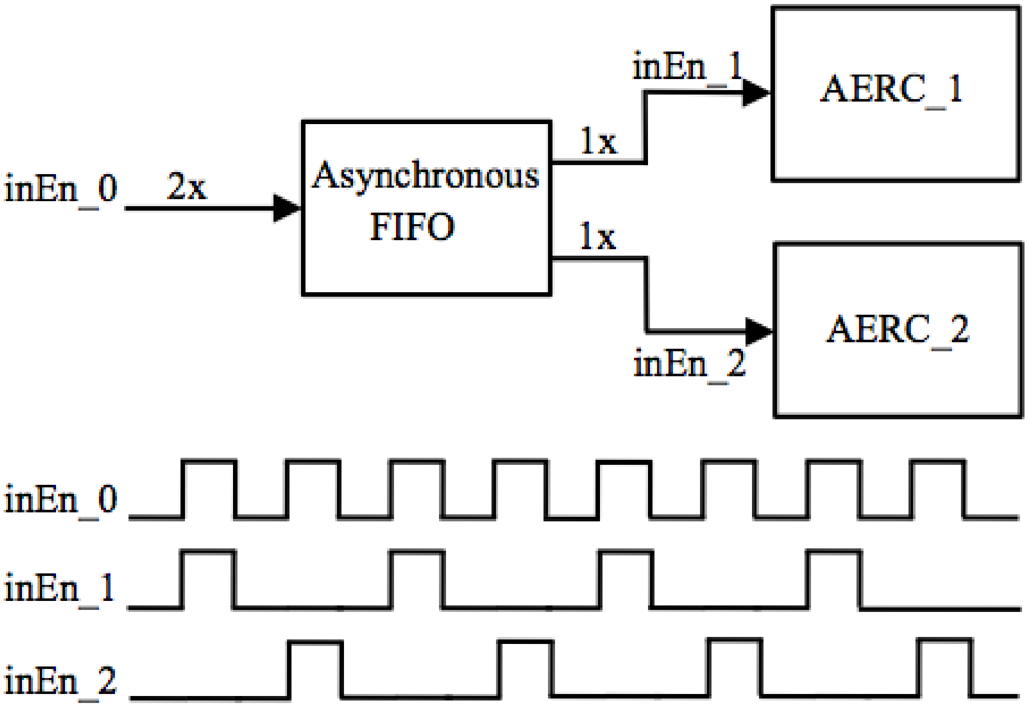


Figure 24: Clock Frequency Details of HSAERC

- The constant input values of mux1 and mux2 need to be 1.

3.4 Reduction Circuit Verilog Code Generator

In this section, reduction circuit Verilog RTL code generator is mentioned. We wrote codes in Python language to generate Verilog RTL codes for our reduction circuits.

Reduction circuits can have different requirements and features. For example, they can use single-precision floating format or double-precision floating format. Moreover, they can use FPU with different clock cycle latency and their operation type can be different. Writing Verilog RTL code for different kind of reduction circuit can take too much design time. Within this scope, we did code generators. We produced Verilog RTL codes for our proposed AERC and HSAERC architectures using our code generators.

The code generator only requires the information of FPU input and output names, FPU operator type, number of FPU latency cycle and input vector size.

CHAPTER IV

RESULTS

In this chapter, we present image fusion implementation result and reduction circuit implementation results.

4.1 Image Fusion Implementation Results

We have applied our tools and techniques to synthesize three image fusion designs that are fully verified and tested. The implementations are performed on both Intel Altera Cyclone IV (on Terasics De2i 150) and Intel Altera Arria 10 (on Nallatechs 385A) boards. We firstly implemented the design at II = 3 cycles on Cyclone IV board and then, we reduced II value equal to 1 cycle in order to achieve higher fps value. Afterwards we implemented the design on Arria 10 FPGA that is more superior to Cyclone IV. Hence, better performance results are achieved.

Performance results of the image fusion designs are shown in Table 3 and 4.

Table 3: Image Fusion Design Results

FPGA	*II*	*Resolution*	*fps*
Arria 10	1	Full HD (1920x1080)	86
Cyclone IV	1	VGA (640x480)	40
Cyclone IV	3	VGA (640x480)	15

Table 4: Image Fusion Clock Speed Results

$FPGA$	II	$0^{\circ}Cfreq.(MHz)$	$85^{\circ}Cfreq.(MHz)$	$100^{\circ}Cfreq.(MHz)$
Arria 10	1	198	-	188
Cyclone IV	1	104	95	-
Cyclone IV	3	78	73	-

Although Cyclone IV runs almost two times slower than Arria 10, fps result is much more than twice for Cyclone IV because the processor of the Terasic DE-150 board with Altera Cyclone IV FPGA has a slower read/write speed than the Nallatech 385A board with Arria 10 FPGA.

In Table 5 and 6, logic block utilization of the FPGAs are shown. Both Cyclone IV and Arria 10 FPGAs have sufficient resources for our image fusion designs.

Table 5: Resource Utilization Results of Image Fusion Design

$FPGA$	II	$\#LogicElement$	$\#Register$
Arria 10	1	22320	60994
Cyclone IV	1	79975	53349
Cyclone IV	3	39812	27252

Table 6: Resource Utilization Proportion Results of Image Fusion Design

$FPGA$	II	$LogicElement$	$Register$
Arria 10	1	5%	8%
Cyclone IV	1	53%	36%
Cyclone IV	3	27%	18%

Arria 10 and Cyclone IV have different IP cores so there is considerable difference in terms of resource utilization. Moreover, when we reduce the II value from 3 to 1, the resource utilization of the image fusion design also increases as shown in Table II

4.2 Reduction Circuit Implementation Results

We have implemented AERC and HSAERC designs on Xilinx Virtex-II Pro and Virtex-5 FPGAs. The characteristics of our proposed reduction circuit designs with other designs in the literature are given in Table 7. In table, “k” refers to pipeline stage of FPU and “n” refers to number of vector item.

AERC design is superior to other designs in terms of the usage of buffer size and FPU. On the other hand, HSAERC design work with minimum latency in comparison with other designs.

Table 7: Comparison of Reduction Circuit Designs

$Design$	$\#FPU$	$Buffer\_Size$	$Total\_latency$
AERC	1	0	$\leq n + (k - \mathrm{mod}(n,k)) + k(\log_2 k + 2)$
HSAERC	2	4	$\leq (n + (k - \mathrm{mod}(n,k)))/2 + k(\log_2 k + 3)$
PCBT[29]	$\log_2 n$	$2(\log_2 n)$	$n + k(\log_2 n)$
FCBT[29]	2	$3(\log_2 n)$	$\leq 3n + (k-1)(\log_2 n)$
DSA[29]	2	$k(\log_2 k + 1)$	$n + k(\log_2 k + 1)$
SSA[29]	1	$2k^2$	$\leq n + 2k^2$
MFPA[30]	$\log_2 k + 1$	k	$\leq n + k(\log_2 k + 2)$
AeMFPA[30]	2	$<2k$	$\leq n + k(\log_2 k + 2)$
A^2eMFPA[30]	2	k	$\leq n + k(\log_2 k + 2)$

Implementation results of different accumulator designs ($n = 128$, $k = 14$) on Xilinx Virtex-II Pro and Virtex-5 FPGAs are given in Table 8 and 9. AERC is the best design in terms of area, on the other hand, HSAERC is the best design in terms of performance.

Table 8: Implementation Results of Reduction Circuits for Virtex-II Pro FPGA

$Design$	$\#FPU$	$\#Slices$	$\#BlockRAM$	$freq.(MHz)$	$Total\_latency$
AERC	1	1,121	0	146	210
HSAERC	2	2,457	1	290	154
PCBT[29]	7	6,808	0	165	226
FCBT[29]	2	2,859	10	170	≤ 475
DSA[29]	2	2,215	3	142	≤ 232
SSA[29]	1	1,804	6	165	≤ 520
MFPA[30]	5	4,991	2	207	198
AeMFPA[30]	2	3,130	14	204	198
A^2eMFPA[30]	2	3,737	2	144	198

Table 9: Implementation Results of Reduction Circuits for Virtex-5 FPGA

$Design$	$\#FPU$	$\#Slices$	$\#BlockRAM$	$freq.(MHz)$	$Total\_latency$
AERC	1	412	0	226	210
HSAERC	2	2,185	1	450	154
MFPA[30]	5	1,692	2	367	198
AeMFPA[30]	2	1,234	14	321	198
A^2eMFPA[30]	2	1,309	2	247	198

CHAPTER V

CONCLUSION

In this thesis, we shared our experiences on hardware implementation of a real-time image fusion algorithm using MAFURES HLS tool and other design techniques in order to complete rapidly. In the literature, there are many image fusion hardware implementations ([15], [16], [17], [18]). However, most of them were implemented by using conventional methodology and without using HLS tools. Design time is very significant for designers because they can do more design if they reduce their design time. There is a trade-off between design time and productivity of hardware designers. If they reduce their design time, their productivity can increase. Thus, we anticipate that our experience of different design tools and techniques will pave the way for hardware designers in order to reduce their design time. We implemented 3 version of the image fusion algorithm (with different fps and/or resolution) on 2 different FPGAs (Cyclone IV and Arria 10) in a fairly short amount of time. Each of these designs on the average contains around 3500 lines of Verilog RTL code, 200 lines of C++ HLS input code and many vendor-specific FPGA IP files separately. The hardware implementation of the first version of the image fusion design takes much more time than the second and third version of the image fusion design. We did not modify so much code for the second design and the third design due to our clever design methodology and used tools and techniques. However, we did not modify so much code for the second design and the third design due to our clever design methodology and used tools and techniques.

In this thesis, we also proposed two novel reduction circuit architectures (AERC and HSAERC). We did Verilog RTL code generators in order to generate our proposed reduction circuit designs swiftly. We generated reduction circuit designs in different specifications and synthesized them. We obtained competitive results in terms of area and performance in comparison with previous designs in the literature.

For future work, it is possible to improve performance of our reduction circuit designs. Moreover, our designs require some modifications in order to handle different consecutive vector sets properly and produce systematical results. Our proposed reduction circuits can reduce accumulation and multiplication operations respectively. However, it does not perform dot-product calculation that is a widely used operation in many scientific applications. Hence, we want to add new features to our code generator in order to generate reduction circuits that perform any dot-product operations.

REFERENCES

[1] C. T. Johnston, K. T. Gribbon, and D. G. Bailey, "Implementing image processing algorithms on FPGAs," in *Proceedings of Electronics New Zealand Conference*, pp. 118–123, 2004.

[2] M. I. Alali, K. M. Mhaidat, and I. Aljarrah, "Implementing image processing algorithm in FPGA hardware," in *Proceedings of Applied Electrical Engineering and Computing Technologies (AEECT)*, pp. 1–5, 2003.

[3] A. E. Guzel, V. E. Levent, M. Tosun, M. A. Özkan, T. Akgun, D. Büyükaydin, C. Erbas, and H. F. Ugurdag, "Using high-level synthesis for rapid design of video processing pipes," in *Proceedings of East-West Design & Test Symposium (EWDTS)*, pp. 1–4, 2016.

[4] A. Cornu, S. Derrin, and D. Lavenier, "HLS tools for FPGA: Faster development with better performance," in *Proceedings of International Conference on Reconfigurable Computing: Architectures, Tools and Applications*, pp. 67–78, 2011.

[5] ALMARVI, *[Online]. Available: http://www.almarvi.eu/.* [Accessed: 20-Jul-2018].

[6] A. R. Toet, "Natural colour mapping for multiband nightvision imagery," *Information Fusion*, vol. 4, pp. 155–166, 2003.

[7] J. L. Bittner, M. T. Schill, F. Mohd-Zaid, and L. M. Blaha, "The effect of multispectral image fusion enhancement on human efficiency," *Cognitive Research*, vol. 2, pp. 1–18, 2017.

[8] A. Toet, M. A. Hogervorst, S. G. Nikolov, J. J. Lewis, T. D. Dixton, D. R. Bull, and C. N. Canagarajah, "Towards cognitive image fusion," *Information Fusion*, vol. 11, pp. 95–113, 2010.

[9] Z. Wu and H. Li, "Research on the technique of image fusion based on wavelet transform," in *Proceedings of ISECS International Colloquium on Computing, Communication, Control and Management*, pp. 165–168, 2009.

[10] Z. Omar and T. Stathaki, "Image fusion: An overview," in *Proceedings of International Conference on Intelligent Systems Modelling and Simulation (ISMS)*, pp. 306–310, 2014.

[11] T. Riley and M. Smith, "Image fusion technology for security and surveillance applications," in *Proceedings of SPIE 6402, Optics and Photonics for Counterterrorism and Crime Fighting II*, pp. 1–12, 2006.

[12] L. Zhuo and V. K. Prasanna, "Sparse matrix-vector multiplication on FPGAs," in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 63–74, 2005.

[13] A. Saeed, M. Elbably, and G. Abdelfadeel, "Efficient FPGA implementation of FFT/IFFT Processor," vol. 3, pp. 103–110, 2009.

[14] Y. Lee, Y. Choi, S.-B. Ko, and M. H. Lee, "Performance analysis of bit-width reduced floating-point arithmetic units in FPGAs: a case study of neural network-based face detector," *EURASIP Journal on Embedded Systems*, vol. 2009, pp. 1–11, 2009.

[15] D. Besiris, V. Tsagaris, N. Fragoulis, and C. Theoharatos, "An FPGA-based hardware implementation of configurable pixel-level color image fusion," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, pp. 362–373, 2012.

[16] O. N. Sims and J. Irvine, "An FPGA implementation of pattern-selective pyramidal image fusion," in *Proceedings of International Conference on Field Programmable Logic and Applications*, pp. 1–4, 2006.

[17] M. A. Mohamed and B. M. El-Den, "Implementation of image fusion techniques for multi-focus images using FPGA," in *Proceedings of National Radio Science Conference*, pp. 1–11, 2011.

[18] G. Mamatha, V. Sumalatha, and L. M. V, "FPGA implementation of satellite image fusion using wavelet substitution method," in *Proceedings of Science and Information Conference*, pp. 1155–1159, 2015.

[19] M. A. Mohamed and B. M. El-Den, "Implementation of image fusion techniques using FPGA," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 10, pp. 95–102, 2010.

[20] F. Qu, B. Liu, J. Zhao, and Q. Sun, "Image fusion real-time system based on FPGA and multi-DSP," *Optics and Photonics Journal*, vol. 3, pp. 76–78, 2013.

[21] Y. Song, K. Gao, G. Ni, and R. Lu, "Implementation of real-time Laplacian pyramid image fusion processing based on FPGA," in *Proceedings of SPIE 6833,Electronic Imaging and Multimedia Technology V*, pp. 1–7, 2007.

[22] V. Popovic, K. Seyid, A. Schmid, and Y. Leblebici, "Real-time hardware implementation of multi-resolution image blending," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2741–2745, 2013.

[23] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley, "Color transfer between images," *IEEE Computer Graphics and Applications*, vol. 21, pp. 34–41, 2001.

[24] P. M. Kogge, "The architecture of pipelined computers," *Hemisphere Publishing Corp.*, 1981.

[25] L. M. Ni and K. Hwang, "Vector-reduction techniques for arithmetic pipelines," *IEEE Transactions on Computers*, vol. 34, pp. 404–411, 1985.

[26] Z. Luo and M. Martonosi, "Accelerating pipelined integer and floating-point accumulations in configurable hardware with delayed addition techniques," *IEEE Transactions on Computers*, vol. 49, pp. 208–218, 2000.

[27] S. R. Vangal, Y. V. Hoskote, Y. V. Borkar, and A. Alvandpour, "A 6.2-GFlops floating-point multiply-accumulator with conditional normalization," *IEEE Journal of Solid-State Circuits*, vol. 41, pp. 2314–2323, 2006.

[28] K. K. Nagar and J. D. Bakos, "A high-performance double precision accumulator," in *Proceedings of International Conference on Field-Programmable Technology (FPT)*, pp. 500–503, 2009.

[29] L. Zhuo, G. R. Morris, and V. K. Prasanna, "A high-performance reduction circuits using deeply pipelined operators on FPGAs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, pp. 1377–1392, 2007.

[30] M. Huang and D. Andrews, "Modular design of fully pipelined reduction circuits on FPGAs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, pp. 1818–1826, 2013.

[31] L. Zhuo, G. R. Morris, and V. K. Prasanna, "Designing scalable FPGA-based reduction circuits using pipelined floating-point cores," in *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, pp. 147–154, 2005.

VITA

Name: Furkan Aydın

Date of Birth: 12/06/1991

Languages: Turkish, English

EDUCATION

- MS: Özyeğin University, Electrical and Electronics Engineering, 2015-2018
- BS: Özyeğin University, Electrical and Electronics Engineering, 2009-2014
- High School: Mersin Yusuf Kalkavan Anatolian High School, 2007-2009
- High School: Elazığ Anatolian High School, 2005-2007

HONOR

- 100 % Tuition waiver for both BS and MS

WORK EXPERIENCE

Özyeğin University | Istanbul / Turkey

Research Assistant * Feb 2016 – ongoing

During my MS studies, I have contributed to the following research projects: ALMARVI (Algorithms, Design Methods, and Many-core Execution Platform for Low-Power Massive Data-Rate Video and Image Processing), ARTEMIS 2013 - project no. 621439, M-RIVA (Methodology Development for Real-time Implementation of Video Algorithms on FPGAs), TÜBİTAK 1001 - project no. 114E343 and Innovative Optical Wireless Communication Technologies for 5G and Beyond, TÜBİTAK 1003 - project no. 215E311.

Özyeğin University | Istanbul / Turkey

Teaching Assistant * Feb 2016 – ongoing

Led the lab sessions and also assisted the activities in the rest of the course for:

– FPGA and System on Chip Design in Spring 2017-2018

– Digital Systems in Spring 2017-2018

– Digital Systems in Fall 2017-2018

– Microprocessors in Spring 2016-2017

– Digital Electronics and FPGA Design in Fall 2016-2017

Emerson Process Management | Istanbul / Turkey

Engineer * Mar 2014 – Aug 2014, Intern * Jul 2013 – Sep 2013

Developed a supervisory control and data acquisition system, which includes hardware/software design and functional specification for automated processes. Developed and implemented logic control schemes for industrial automation applications.

Özyeğin University | Istanbul / Turkey

Library Assistant * Oct 2010 – Jan 2011,

IT Support Services * Oct 2009 – May 2010

Was responsible for following up the delivery and return of library materials on the library system. Assisted the technical support staff on setting up audio-visual systems and Internet access.

TECHNICAL EXPERIENCE

Programming Languages: Verilog * C/C++ * Python * Perl * Java

Tools: Altera Quartus Synthesis * Xilinx ISE/Vivado Synthesis * ISim * ModelSim * MATLAB * Proteus

Experience in: FPGA Design * Implementing Video Processing and Communication Applications * Computer Arithmetic * HLS * Embedded Systems (Arduino, Raspberry Pi, etc) * LabVIEW FPGA

PUBLICATIONS

Accepted Papers (Not yet published):

- Conference Paper

  F. Aydin, H. F. Ugurdag, V. E. Levent, A. E. Güzel, N. F. R. Annafianto, M. A. Özkan, T. Akgün, C. Erbas, "Rapid Design of Real-Time Image Fusion on FPGA Using HLS and Other Techniques," *in Proceedings of the Second International Symposium on Multimedia Computing and Applications,* 2018.

- Journal Paper

  V. E. Levent, A. E. Güzel, M. Tosun, M. Buyukmihci, F. Aydin, S. Gören, C. Erbas, T. Akgün, H. F. Ugurdag, "Tools and Techniques for Implementation of Real-Time Video Processing Algorithms," *Journal of Signal Processing Systems,* 2018.