2D Digital Filter Implementation on a FPGA

by

Danny T. Tsuei

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Danny T. Tsuei

Abstract

The use of two dimensional (2D) digital filters for real-time 2D data processing has found important practical applications in many areas, such as aerial surveillance, satellite imaging and pattern recognition. In the case of military operations, real-time image processing is extensively used in target acquisition and tracking, automatic target recognition and identification, and guidance of autonomous robots. Furthermore, equal opportunities exist in civil industries such as vacuum cleaner path recognition and mapping and car collision detection and avoidance. Many of these applications require dedicated hardware for signal processing. It is not efficient to implement 2D digital filters using a single processor for real-time applications due to the large amount of data. A multiprocessor implementation can be used in order to reduce processing time.

Previous work explored several realizations of 2D denominator separable digital filters with minimal throughput delay by utilizing parallel processors. It was shown that regardless of the order of the filter, a throughput delay of one adder and one multiplier can be achieved. The proposed realizations have high regularity due to the nature of the processors. In this thesis, all four realizations are implemented in a Field Programming Gate Array (FPGA) with floating point adders, multipliers and shift registers. The implementation details and design trade-offs are discussed. Simulation results in terms of performance, area and power are compared.

From the experimental results, realization four is the ideal candidate for implementation on an Application Specific Integrated Circuit (ASIC) since it has the best performance, dissipates the lowest power, and uses the least amount of logic when compared to other realizations of the same filter size. For a filter size of 5×5, realization four can produce a throughput of 16.3 million pixels per second, which is comparable to realization one and about 34% increase in performance compared to realization one and two. For the given filter size, realization four dissipates the same amount of dynamic power as realization one, and roughly 54% less than realization three and 140% less than realization two. Furthermore, area reduction can be applied by converting floating point algorithms to fixed point algorithms. Alternatively, the denormalization and normalization stage of the floating point pipeline can be eliminated and fused together in order to save hardware resources.

Acknowledgments

I would like to give thanks to my supervising professors, Professor Mohamed Yahia Dabbagh and Professor Manoj Sachdev for their dedication, encouragement, and guidance during my graduate studies here at the University of Waterloo. I would like to thank them for reading and critiquing this thesis. I would also like to thank my thesis readers, Professor Bill Bishop and Professor Sebastian Fischmeister for their feedbacks and corrections on various parts of this thesis.

I would like to show my appreciation to the administrative staff in the Electrical and Computer Engineering department, Wendy Boles, Wendy Stoneman, Irena Baltaduonis, Susan King, and Annette Dietrich, for their assistance with the necessary paper work to fulfill my graduation requirement. Also, I would like to express my thanks to Philip Regier and Paul Ludwig for their technical assistance with regard to computer hardware.

To the members of CMOS Design and Reliability (CDR) group, David Li, Pierce Chuang, Jaspal Shah, Tasreen Charania and others, I couldn't have made it through graduate studies without your help. Thank you all for the constant encouragement through good times and bad. It was a pleasure to have you all by my side.

Lastly, I would like to express my gratitude to my parents, who supported me both financially and mentally, and for that I am grateful. I hope I have made you proud and I can continue to do so in the future.

Dedication

This thesis is dedicated to

My parents, Edward and Gina
for their unconditional love and support

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The application of Digital Signal Processing (DSP) algorithms impacts many aspects of our contemporary standard of living [1].Much of our technological advancements in modern wireless and wired communications, medical imaging equipment, audio and visual products are driven by new and emerging DSP algorithms [2, 3]. The advent of more complex DSP algorithms with smaller and faster Integrated Circuits (ICs) contribute to the eventuality of complex systems. DSP algorithms typically, but not always, process real time data as the data is acquired. This implies that DSP algorithms must produce a deterministic run time to satisfy system-wide timing constraints.

A 2D digital filter is a type of DSP algorithm used to process 2D data and has found important practical applications in many areas, including aerial surveillance, satellite imaging, pattern recognition, target acquisition and tracking [1, 2, 4, 5].

Traditionally DSP algorithms are implemented using digital signal processors that execute a set of instructions on a set of given input data. Special mathematical circuitry is often found on digital signal processors in order to facilitate complex mathematical functions. Designers are required to write in a low level language in order to produce optimized instructions for digital signal processors. However, in recent years, DSP designers have been shifting toward the use of Field Programming Gate Arrays (FPGA) for hardware development. Majority of the controversy between the digital signal processor versus FPGA surrounds the issue of processing capability, as measured in millions of instructions per second (MIPS) [6, 7]. Other inherent advantages of the FPGA, including reliability and maintainability, are often ignored.

Since a digital signal processor is a special microprocessor that requires a stream of instructions to execute, not all the instructions are data related. Some instructions are

control related, some are protocol related, while some are DSP related. Resources in the digital signal processor, such as internal registers, external memory, Direct Memory Access (DMA) controller, transfer buses and external input/output (I/O) signals are shared by all instructions. This can cause unexpected behavior as one instruction attempts to modify the results of another instruction. DSP algorithms are often required to run in real-time and any unexpected delays can result in system failure. FPGAs inherently negate this problem by issuing dedicated resources to the executing instructions. Dedicated data and control paths are placed and assigned to predetermined locations on the FPGA to ensure the functionality correctness while meeting timing constraints. Memory blocks are distributed throughout the FPGA and each instruction is allocated the proper amount of memory for its execution. I/O signals are clearly defined from modules to modules, and eliminates unexpected interactions between instructions. This also allows the designer to more easily locate and isolate bugs [8]. Not only does FPGA guarantee dedicated run-time resources, verification is much more simplified on the FPGA than digital signal processors.

Operating system (OS) , or more commonly called kernels, is used to control resource sharing on the digital signal processor. Allowable execution time, memory allocation and peripheral I/O access time are all managed by the operating system. However, there is an inherent conflict of interest between instruction efficiency and operating system intervention. For the instruction to work at optimal efficiency, it would be ideal to have zero intervention from the operating system. Compounding these difficulties is the lack of ability to verify functional correctness. It is impossible to ensure all possible permutations of the instructions plus OS interventions are tested during verification based on the large number of test cases. Challenges arises from complete functional verification is equally daunting on FPGAs. However, at the fundamental level, much of the design and verification tools are shared between FPGA and its Application Specific Integrated Circuit (ASIC) counterpart. The FPGA designers are able to benefit from this mutual relationship since ASIC designers are extremely intolerant of design bugs. It would take millions of dollars in fabrication cost to fix an implementation error [8]. From these arguments, it is clear that FPGA is an excellent hardware development platform for DSP designers.

1.1 Research Contribution

In this thesis, multiple implementations of two dimensional (2D) digital filters are presented. The implementations are designed and optimized using Altera's proprietary Quartus II software and simulated using ModelSim. Results in this thesis show that with minor modifications on the filter realization, the same throughput as derived from previous work

can be obtained.

A framework for future 2D filter implementations is also presented in this thesis. The 2D digital filters are implemented as custom hardware blocks in the Altera System on Programmable Chip (SOPC) Builder. This provides flexibility in customizing the DSP system using additional SOPC components, such as Nios II processor, Phase Locked Loops (PLLs), external Random Access Memory (RAM), and parallel I/Os that can be easily integrated into the design to tailor to different applications.

Several optimization techniques pertaining to Hardware Register Transfer level (RTL) are also discussed in the thesis. These techniques include selecting floating point (FP) operators, secondary clock divider circuit, specifying synopsis design constraint (SDC) file constraint for timing oriented fitting and using the Design Space Explorer (DSE) for timing optimization. Future work carried out in the implementation of 2D digital filters on FPGA will benefit from these optimization techniques as they are transferable across different RTL lanaguges and hardware platforms.

1.2 Thesis Overview

The remainder of this thesis is organized as follows. Chapter 2 presents the general form of four different 2D denominator separable digital filters. Chapter 3 discusses the derivation of filter coefficients from a sample impulse function as well as any architectural changes due to complex poles. Chapter 4 discusses architectural changes related to these realizations based on timing considerations. Chapter 5 describes how the digital filter fits into the system from a top level perspective. Chapter 6 shows and analyzes the simulation results. Finally, chapter 7 concludes the thesis.

Chapter 2

Realization of 2D Separable Denominator Digital Filters

A 2D separable denominator digital filter is a special class of the 2D digital filter, that has a transfer function with a separable denominator. There are several advantages to this type of filter. First, the design of such a filter is easier than the non-separable general filter. Second, stability tests are simpler and more similar to the 1D filter stability tests. Third, a general 2D non-separable filter can be approximated by a separable denominator filter. Fourth, circularly symmetric and fan filters, which are widely used in practice, have separable denominator transfer functions. Lastly, well established techniques for implementing 1D filter realizations can be used to derive several realizations for the 2D separable denominator filters.

In section 2.1, the general form of 2D separable denominator digital filter is presented. In the four sections that follow, Sections 2.2 to 2.5, four realizations of the 2D separable denominator digital filter are presented. These realizations take advantage of pipelined, multi-processor parallel processing to achieve high throughput. The high level block diagram and throughput are shown for each realization. This chapter is based on previous work in [9].

2.1 General Form

The transfer function of a separable denominator 2D digital filter of order $M_1 \times M_2$ is given by:

$$
\begin{aligned}
H(z_1, z_2) &= \frac{G(z_1, z_2)}{F(z_1, z_2)} \\
&= \frac{\sum_{i=0}^{M_1} \sum_{j=0}^{M_2} b_{i,j} z_1^{-i} z_2^{-j}}{\sum_{i=0}^{M_1} \alpha_i z_1^{-i} \sum_{j=0}^{M_2} \beta_j z_2^{-j}} \\
&= \frac{p(z_1, z_2)}{q_1(z_1) q_2(z_2)}
\end{aligned}
\tag{2.1}
$$

where $F(z_1, z_2)$ and $G(z_1, z_2)$ are the 2D Z-transforms of the input $f(n_1, n_2)$ and the output $g(n_1, n_2)$. $b_{i,j}$, α_i and β_j are constant coefficients that determine the characteristics of the filter. The polynomials $p(z_1, z_2)$, $q_1(z_1)$ and $q_2(z_2)$ are functions of the horizontal delay (z_1^{-1}) and the vertical delay (z_2^{-1}). M_1 is the horizontal dimension and M_2 is the vertical dimension of the 2D digital filter. Also, we assume that the numerator and denomiator are coprime and $\alpha_0 = \beta_0 = 1$.

2.2 Realization One

The transfer function $H(z_1, z_2)$ in (2.1) can be rewritten as a product of two subfilters:

$$
H(z_1, z_2) = H_1(z_1) H_2(z_1, z_2) \tag{2.2}
$$

where

$$
H_1(z_1) = \frac{1}{q_1(z_1)} = \frac{1}{\sum_{i=0}^{M_1} \alpha_i z_1^{-i}} \tag{2.3}
$$

$$
H_2(z_1, z_2) = \frac{p(z_1, z_2)}{q_2(z_2)} = \frac{\sum_{i=0}^{M_1} \sum_{j=0}^{M_2} b_{i,j} z_1^{-i} z_2^{-j}}{\sum_{j=0}^{M_2} \beta_j z_2^{-j}} \tag{2.4}
$$

Equation (2.2) suggests the realization of $H(z_1, z_2)$ as a cascade of two subfilters. The first subfilter $H_1(z_1)$ in (2.3) is an one dimensional (1D) Infinite Impulse Response (IIR) filter. This filter can be realized using the transposed direct form II, which is suitable for a multiprocessor implementation. The transfer function of the subfilter $H_2(z_1, z_2)$ can be rewritten as:

$$H_2(z_1, z_2) = \frac{\sum_{j=0}^{M_2} b_j(z_1) z_2^{-j}}{\sum_{j=0}^{M_2} \beta_j z_2^{-j}} \tag{2.5}$$

where

$$b_j(z_1) = \sum_{i=0}^{M_1} b_{i,j} z_1^{-i}, j = 0, 1, ..., M_2 \tag{2.6}$$

Again, 1D realization structures can be used for the realization of this transfer function. The transposed direct form II structure is used to realize the subfilter, where in Equation 2.5, $b_j(z_1), j = 0, 1, ...M_2$ represents a set of 1D FIR filters.

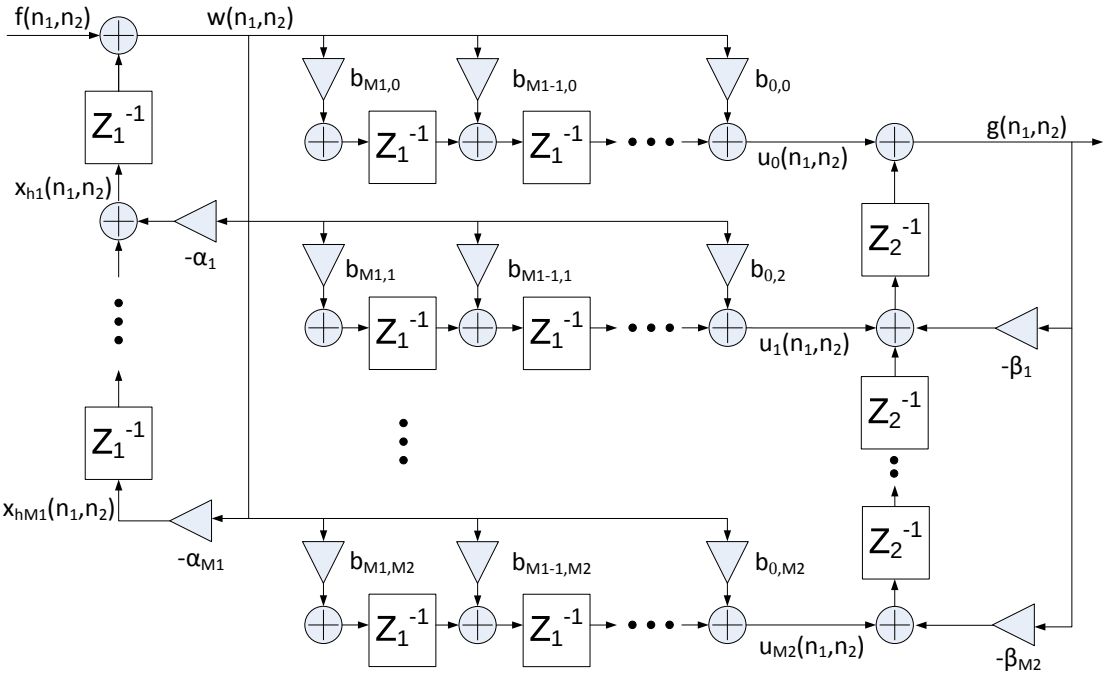


Figure 2.1: Realization One

Assuming the input is $f(n_1, n_2)$, output is $g(n_1, n_2)$, while $w(n_1, n_2)$ and $u_j(n_1, n_2)$ (where $j = 0...M_2$) are the intermediate state variables, the overall separable denominator filter has a completely decomposed realization as shown in Figure 2.1. The decomposed

realization consists of a cascade of three subfilters: a 1D horizontal IIR filter, a bank of 1D horizontal FIR filters, and a 1D vertical IIR filter. It can be seen that this realization has the minimum number of vertical shifts, namely z_2^{-1}. A minimum number of vertical shifts is a desired feature since the physical implementation of a vertical shift is far more expensive in terms of hardware compared to a horizontal shift. For example, in a raster scan image, a horizontal shift corresponds to a storage of one pixel of data, while a vertical shift corresponds to storage of an entire row of pixels. Since an FPGA is limited in terms of storage elements, be it RAM or registers, the less number of vertical shifts there are, the more likely the design will fit on the FPGA.

Pipelining and parallelism can be exploited effectively in the implementation of the decomposed realization in Figure 2.1. Since each subfilter consists of multiple repeating instances of multipliers, adders and shift registers, a module can be created to include one multiplier, one adder and one shift register. Cascading the module in parallel or series will produce the desired filter. Modules can also be called processors. The temporary data stored in the shift registers can also be called states.

The first subfilter has M_1 horizontal states and single intermediate output $w(n_1, n_2)$. The states and the intermediate output have a computation structure that requires one multiplication and two additions using M_1 processors. The second subfilter has $M_1(M_2+1)$ horizontal states and $(M_2 + 1)$ intermediate outputs. These states and the intermediate outputs can be computed in a single multiplication and a single addition using $(M_1 + 1)(M_2 + 1)$ processors. The third subfilter has M_2 vertical states and a single output $g(n_1, n_2)$. The computation of these states require the use of a processor that is composed of two adders, one multiplier and one shift register, assuming the first and the second subfilters are processing in parallel with the third subfilter. The required number of processors (P) and the cycle time (T) required for the realization are:

$$P = (M_1 + 1)(M_2 + 1) + M_1 + M_2 \tag{2.7}$$

$$T = t_{mult} + 2t_{add} + 1 \tag{2.8}$$

2.3 Realization Two

The second realization is based on the decomposition of the numerator polynomial of the transfer function in Equation (2.1) by the general decomposition theorem [10]. This

theorem decomposes the 2D polynomial as a sum of products of simple 1D polynomials. The numerator of the transfer function $H(z_1, z_2)$ in Equation (2.1) can be rewritten as

$$p(z_1, z_2) = \sum_{i=0}^{M_1} \sum_{j=0}^{M_2} b_{i,j} z_1^{-i} z_2^{-j} = Z_1^T B Z_2 \tag{2.9}$$

where

$$Z_1^T = [1, z_1^{-1}, z_1^{-2}, ..., z_1^{-M_1}] \tag{2.10}$$

$$Z_2 = [1, z_2^{-1}, z_2^{-2}, ..., z_2^{-M_2}] \tag{2.11}$$

$$B = [b_{i,j}] \tag{2.12}$$

B is a matrix of dimension $(M_1 + 1) \times (M_2 + 1)$. There are many decompositions for the matrix B. In particular, it can always be decomposed as a product of two matrices. The most commonly used decompositions in the literature are Singular Value Decomposition (SVD) [11], the Jordan Decomposition (JD) [12], and the Lower-Upper Decomposition (LUD) [13]. By decomposing the matrix B into a product of two matrices, multiplying, then expanding gives:

$$p(z_1, z_2) = \sum_{k=1}^{r} p_{k1}(z_1) p_{k2}(z_2) \tag{2.13}$$

where r is the rank of the matrix B, $p_{k1}(z_1)$ and $p_{k2}(z_2)$ are polynomials in z_1 and z_2, respectively, given by:

$$p_{k1}(z_1) = \sum_{i=0}^{M_1} g_{ki} z_1^{-i} \tag{2.14}$$

$$p_{k2}(z_2) = \sum_{j=0}^{M_2} h_{kj} z_2^{-j} \tag{2.15}$$

By using the decomposition in Equation (2.13), the transfer function $H(z_1, z_2)$ in Equation (2.1) is decomposed as the sum of products of 1-D IIR filters as:

$$H(z_1, z_2) = \sum_{k=1}^{r} H_{k1}(z_1) H_{k2}(z_2) \tag{2.16}$$

where

$$H_{k1}(z_1) = \frac{p_{k1}(z_1)}{q_1(z_1)}, \; H_{k2}(z_2) = \frac{p_{k2}(z_2)}{q_2(z_2)} \tag{2.17}$$

Equation (2.16) suggests a hybrid parallel-cascade realization of $H(z_1, z_2)$ as shown in Figure 2.2 below.

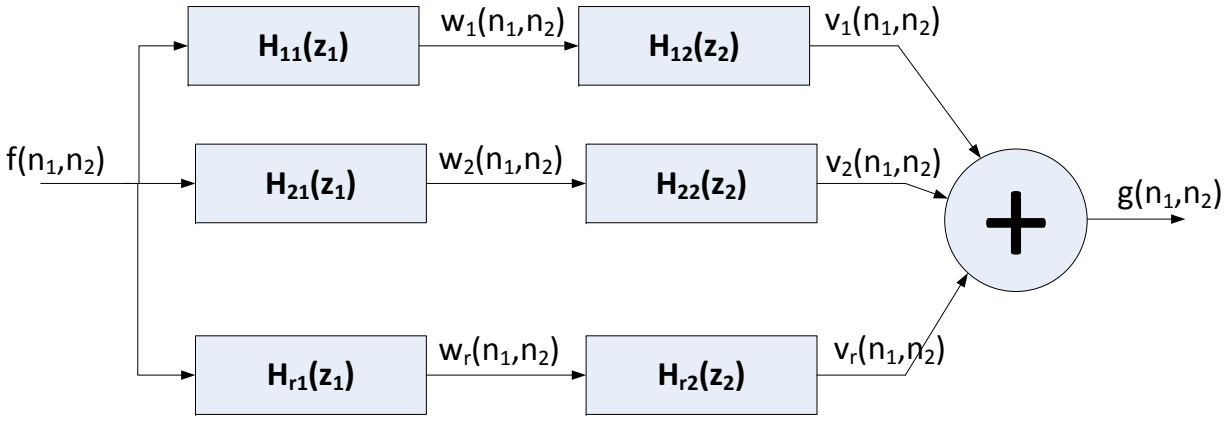


Figure 2.2: Realization Two: A Hybrid Parallel-Cascade Realization.

Each cascade is implemented as a pipeline of two 1D IIR filters that can be realized in any of the existing 1D realization. For example, $H_{k1}(z_1)$ (k = 1, 2, ... , r) can be realized in the observer canonical form, where $f(n_1, n_2)$ and $w_k(n_1, n_2)$ are its input and output. Similarly, the transfer function $H_{k2}(z_2)$ ($k = 1, 2, ..., r$) can be realized in the same form, where $w_k(n_1, n_2)$ and $v_k(n_1, n_2)$ are its input and output respectively. The output of the filter is then

$$g(n_1, n_2) = v_1(n_1, n_2) + v_2(n_1, n_2) + ... + v_r(n_1, n_2) \tag{2.18}$$

It can be shown that the filter realization has a number of processors (P) and a cycle time (T) given by:

$$P = r(2M_1 + 1 + 2M_2 + 1) \tag{2.19}$$

$$T = t_{mult} + 2t_{add} \tag{2.20}$$

where the summation at the filter output in Figure 2.3 is excluded from the computation of P in Equation (2.19). The number of processor required for realization two is almost double the amount required for realization one. This is due to the multiple realization of the denominator as can be seen in Equation (2.16).

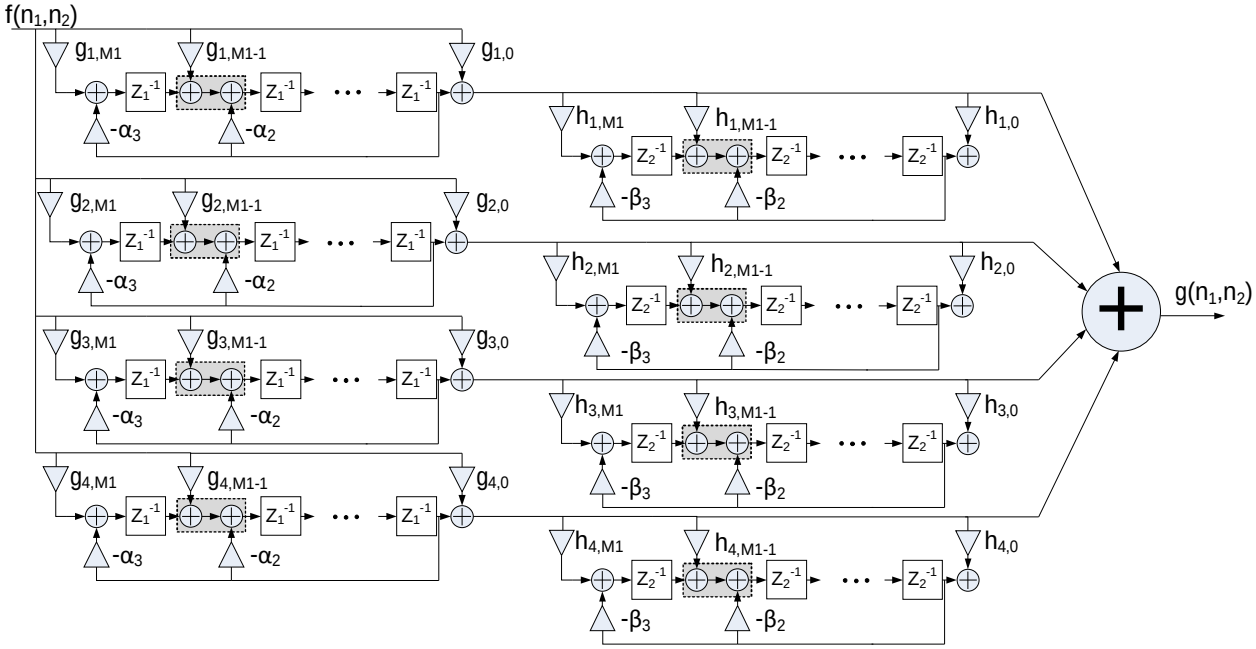


Figure 2.3: Realization Two.

2.4 Realization Three

For 1D filters, there always exists a diagonal state-space realization. The realization has high parallelism, low throughput delay and low roundoff noise. The realization can be obtained by applying partial-fraction expansion to the transfer function. In general, partial fraction does not exist for 2D filters. This is due to the lack of a fundamental theorem of algebra for factoring 2D and higher order polynomials. However, there is a partial-fraction expansion for the special case of the separable denominator transfer function $H(z_1, z_2)$ in Equation (2.1). The separable denominator transfer function $H(z_1, z_2)$ has a partial fraction expansion given by:

$$H(z_1, z_2) = \sum_{i=1}^{M_1} \sum_{j=1}^{M_2} \frac{K_{i,j}}{(1 - \gamma_i z_1^{-1})(1 - \lambda_j z_2^{-1})} + \sum_{i=1}^{M_1} \frac{K_{i,0}}{1 - \gamma_i z_1^{-1}} + \sum_{j=1}^{M_2} \frac{K_{0,j}}{1 - \lambda_j z_2^{-1}} + K_{0,0} \quad (2.21)$$

where γ_i $(i = 1, 2, ..., M_1)$ are the distinct roots of $q_1(z_1)$, and λ_j $(j = 1, 2, ..., M_2)$ are the distinct roots of $q_2(z_2)$ and $K_{i,j}$ are constants that can be determined from $H(z_1, z_2)$ in Equation (2.1). The proof of this theorem is given in [9].

Similar to 1D filters, the above theorem can be extended to the case of multiple poles. The expansion of $H(z_1, z_2)$ in Equation (2.21) suggests a highly parallel structure, which has an obvious multiprocessor realization as shown in Figure 2.4.

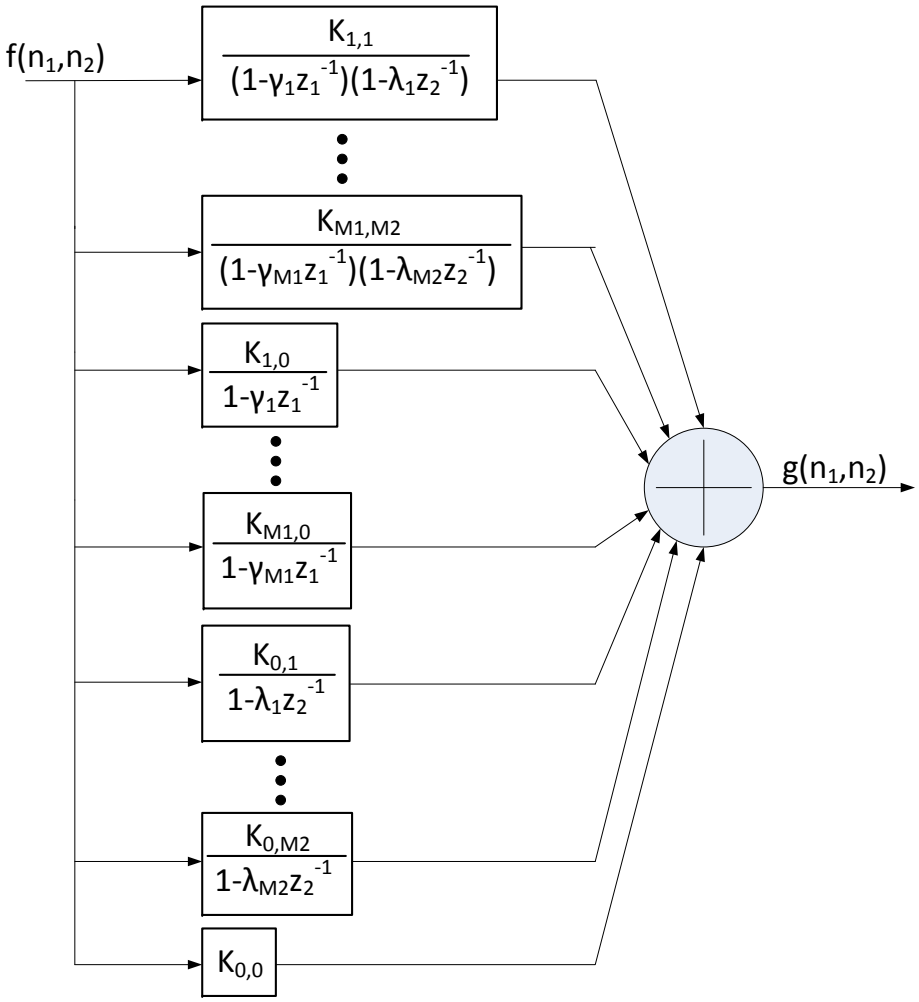


Figure 2.4: Realization Three

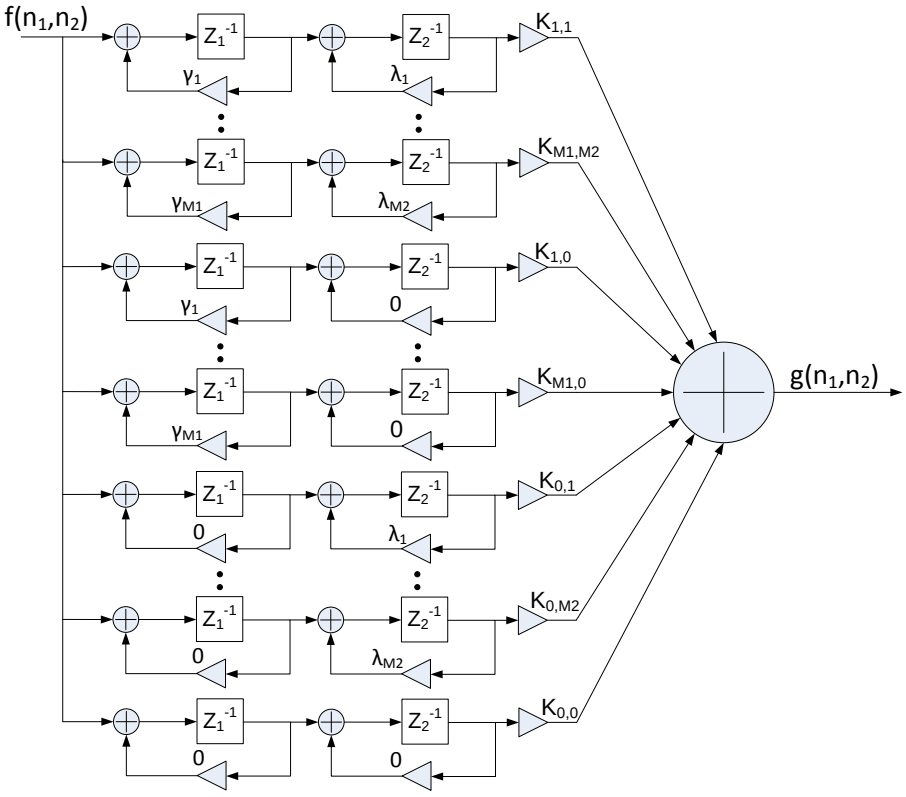


Figure 2.5: Realization Three Implementation

The throughput delay of this implementation is limited by the realization of each of the sub-transfer functions

$$H_{i,j}(z_1, z_2) = \frac{K_{i,j}}{(1 - \gamma_i z_1^{-1})(1 - \lambda_j z_2^{-1})} \tag{2.22}$$

One realization is shown in Figure 2.5. The filter can be broken down into three subfilters – two adder-multiplier processor connected back to back, followed by a single multiplier and a final adder that sums up all the intermediate outputs. The implementation of the filter has a number of processors (P) and a cycle time (T) given as:

$$P = 3M_1M_2 + 2(M_1 + M_2) \tag{2.23}$$

$$T = t_{mult} + t_{add} \tag{2.24}$$

where the number of processor used does not include the number of adders required to implement the pipelined summation adder. The number of processors required is higher than both realization one and two due to the duplicate realization of the poles. This structure, however, maintains a highly parallelized pipeline and is quite suitable for multi-processor implementation.

2.5 Realization Four

This realization is a modified realization of the realization three. The improvement here is that the multiple realization of the poles is avoided and hence the number of required processors is reduced. The realization is based on writing the partial fraction expansion in Equation (2.21) in matrix form as:

$$H(z_1, z_2) = P^T(z_1) * K * Q(z_2) \tag{2.25}$$

where

$$P(z_1) = \begin{bmatrix} 1 \\ \frac{1}{1-\gamma_1 z_1^{-1}} \\ . \\ . \\ \frac{1}{1-\gamma_{M_1} z_1^{-1}} \end{bmatrix}, Q(z_2) = \begin{bmatrix} 1 \\ \frac{1}{1-\lambda_1 z_2^{-1}} \\ . \\ . \\ \frac{1}{1-\gamma_{M_2} z_2^{-1}} \end{bmatrix}, K = \begin{bmatrix} K_{0,0} & K_{0,1} & . & . & . & K_{0,M_2} \\ K_{1,0} & K_{1,1} & & & & . \\ . & & & & & . \\ . & & & & & . \\ K_{M_1,0} & . & & . & . & K_{M_1,M_2} \end{bmatrix} \tag{2.26}$$

Equation (2.25) suggests realizing $H(z_1, z_2)$ as a pipeline of three stages. The first stage is a parallel realization of the vector $P(z_1)$. The second stage is the vector-multiplication operations on the columns of the matrix K followed by vector-summation operations. The third stage is the parallel realization of the vector $Q(z_2)$ followed by summation operations. The complete realization is shown in Figure 2.6.

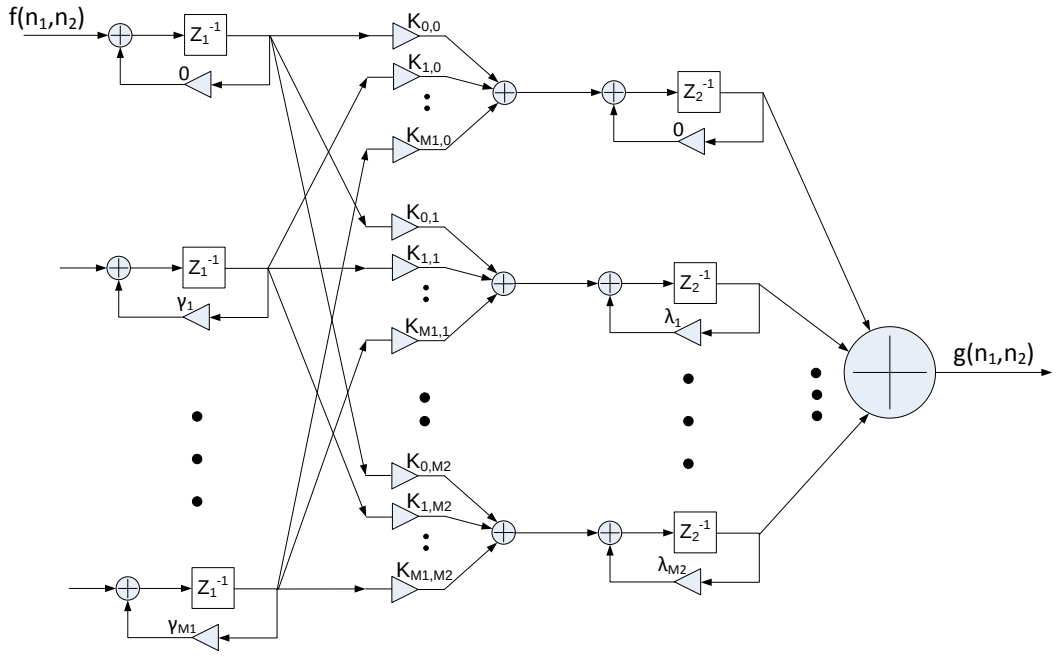


Figure 2.6: Realization Four.

The elements in $P(z_1)$ can be realized in parallel in one addition and one multiplication. The second stage involves parallel multiplication and addition, which can be implemented in parallel and pipeline fashion such that the cycle time is not increased, except for an initial delay. The third stage has a similar structure to the first stage, therefore it has the same cycle time as the first stage. The required number of processors (P) and the cycle time (T) are:

$$P = M_1 M_2 + 2(M_1 + M_2) \tag{2.27}$$

$$T = t_{mult} + t_{add} \tag{2.28}$$

It is worthwhile to note that the number of processors required is much less than

the previous realizations. In addition, this implementation can take advantage of highly parallelized addition and multiplication when calculating the intermediate state variables.

Chapter 3

Filter Coefficients Derivation

In order to apply a 2D separable denominator digital filter to two dimensional data, the corresponding filter coefficients must be first derived before data can be processed. The derived filter coefficients can then be used to verify the functional correctness of the implementation. A numerical example is taken out of existing publication in order to derive the coefficients. The 2D impulse response specification for a Quarter-Plane Gaussian Filter [14] is given by:

$$h_d(m,n) = 0.256322 \cdot \exp[-0.103203(m-4)^2 + (n-4)^2] \tag{3.1}$$

The resulting 2D separable denominator digital filter is given by Roesser local state-space (LSS) matrices:

$$\begin{bmatrix} x^h(i+1,j) \\ x^v(i,j+1) \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ 0 & A_4 \end{bmatrix} \begin{bmatrix} x^h(i,j) \\ x^v(i,j) \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} u(i,j) \tag{3.2}$$

$$y(i,j) = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} x^h(i,j) \\ x^v(i,j) \end{bmatrix} + du(i,j) \tag{3.3}$$

where $x^h(i,j)$ is a $M_1 \times 1$ horizontal state vector, $x^v(i,j)$ is a $M_2 \times 1$ vertical state vector, $u(i,j)$ is a scalar input, $y(i,j)$ is a scalar output, and

$$A_1 = \begin{bmatrix} 0.86382 & 0.27191 & 0.03899 \\ -0.27191 & 0.59513 & -0.36079 \\ 0.03899 & 0.36079 & 0.35615 \end{bmatrix} \tag{3.4}$$

$$A_2 = \begin{bmatrix} 0.42903 & 0.33791 & -0.12990 \\ 0.33791 & 0.26614 & -0.10231 \\ -0.12990 & -0.10231 & 0.03933 \end{bmatrix} \tag{3.5}$$

$$A_4 = \begin{bmatrix} 0.86382 & -0.27191 & 0.03899 \\ 0.27191 & 0.59513 & 0.36079 \\ 0.03899 & -0.36079 & 0.35615 \end{bmatrix} \tag{3.6}$$

$$b_1^t = \begin{bmatrix} 0.06361 & 0.05010 & -0.01926 \end{bmatrix} \tag{3.7}$$

$$b_2^t = \begin{bmatrix} 0.65500 & -0.51589 & -0.19831 \end{bmatrix} \tag{3.8}$$

$$c_1 = \begin{bmatrix} 0.65500 & -0.51589 & -0.19831 \end{bmatrix} \tag{3.9}$$

$$c_2 = \begin{bmatrix} 0.06361 & 0.05010 & -0.01926 \end{bmatrix} \tag{3.10}$$

$$D = 0.00943 \tag{3.11}$$

To derive the coefficients required for the realizations, the local state-space matrices need to be converted to a transfer function. Assume there is no loss of generality in representing 2D separable denominator digital filters by the LSS model in Equation 3.2 and 3.3. The LSS model is assumed to be asymptotically stable and minimal. The transfer function is given by [15]

$$H(z_1, z_2) = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} z_1 I_{M1} - A_1 & -A_2 \\ 0 & z_2 I_{M2} - A_4 \end{bmatrix}^{-1} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} + d \tag{3.12}$$

$$= \begin{bmatrix} 1 & c_1 (z_1 I_{M1} - A_1)^{-1} \end{bmatrix} \begin{bmatrix} d & c_2 \\ b_1 & A_2 \end{bmatrix} \begin{bmatrix} 1 \\ (z_2 I_{M2} - A_4)^{-1} b_2 \end{bmatrix} \tag{3.13}$$

By applying this transform, the resulting transfer function is:

$$H(z_1, z_2) = \frac{i(z_1, z_2)}{j(z_1) \times k(z_2)} \tag{3.14}$$

where

$$\begin{aligned} i(z_1, z_2) &= 0.009059397520 z_1^{-3} z_2^{-3} + 0.007524922930 z_1^{-3} z_2^{-2} + 0.002468223450 z_1^{-3} z_2^{-1} + \\ & 0.009244024377 z_1^{-3} + 0.007524922933 z_1^{-2} z_2^{-3} + 0.006245656370 z_1^{-2} z_2^{-2} + \\ & 0.00205458569 z_1^{-2} z_2^{-1} + 0.007675175463 z_1^{-2} + 0.002468223444 z_1^{-1} z_2^{-3} + \\ & 0.00205458571 z_1^{-1} z_2^{-2} + 0.000668282136 z_1^{-1} z_2^{-1} + 0.00252151860 z_1^{-1} + \\ & 0.009244024371 z_2^{-3} + 0.00767517545 z_2^{-2} + 0.00252151860 z_2^{-1} + \\ & 0.00943 \end{aligned}$$

$$j(z_1) = \frac{1}{1 - 1.515100000 z_1^{-1} + 1.236274491 z_1^{-2} - 0.3133116076 z_1^{-3}}$$

$$k(z_2) = \frac{1}{1 - 1.515100000 z_2^{-1} + 1.236274491 z_2^{-2} - 0.3133116076 z_2^{-3}}$$

Factoring the denominator of $j(z_1)$ and $k(z_2)$ gives:

$$\begin{aligned} j(z_1) &= \frac{1}{(1 - 0.3945011412 z_1^{-1})(1 - 1.120598859 z_1^{-1} + 0.7941969614 z_1^{-2})} \\ &= \frac{1}{(1 - 0.3945011412 z_1^{-1})(0.560295295 + 0.6562328205\mathrm{i} z_1^{-1})} \cdot \\ & \quad \frac{1}{(0.560295295 - 0.6562328205\mathrm{i} z_1^{-1})} \end{aligned}$$

and

$$\begin{aligned} k(z_2) &= \frac{1}{(1 - 0.3945011412 z_2^{-1})(1 - 1.120598859 z_2^{-1} + 0.7941969614 z_2^{-2})} \\ &= \frac{1}{(1 - 0.3945011412 z_2^{-1})(0.560295295 + 0.6562328205\mathrm{i} z_2^{-1})} \cdot \\ & \quad \frac{1}{(0.560295295 - 0.6562328205\mathrm{i} z_2^{-1})} \end{aligned}$$

so, the filter has two real poles at:

$$z_1 = z_2 = 0.3945011412$$

and four imaginary poles at:

$$z_1 = z_2 = 0.560295295 \pm 0.6552328205\text{i}$$

3.1 Realization One

Since the transfer function of realization one can be written as a product of two functions as shown in Equation 2.2, Equation 2.3, and Equation 2.4 , the filter coefficients can be directly collected from the transfer function without any transformation as follows:

$$\begin{aligned} \alpha_0 &= \beta_0 = 1 \\ \alpha_1 &= \beta_1 = -1.5151 \\ \alpha_2 &= \beta_2 = 1.236274491 \\ \alpha_3 &= \beta_3 = -0.3133116076 \end{aligned}$$

$$
\begin{aligned}
b_{3,3} &= 0.009059397520 \\
b_{3,2} &= 0.007524922930 \\
b_{3,1} &= 0.002468223450 \\
b_{3,0} &= 0.009244024377 \\
b_{2,3} &= 0.007524922933 \\
b_{2,2} &= 0.006245656370 \\
b_{2,1} &= 0.00205458569 \\
b_{2,0} &= 0.007675175463 \\
b_{1,3} &= 0.002468223444 \\
b_{1,2} &= 0.00205458571 \\
b_{1,1} &= 0.000668282136 \\
b_{1,0} &= 0.00252151860 \\
b_{0,3} &= 0.009244024371 \\
b_{0,2} &= 0.00767517545 \\
b_{0,1} &= 0.00252151860 \\
b_{0,0} &= 0.00943000000
\end{aligned}
$$

3.2 Realization Two

The filter coefficients for realization two are obtained from the decomposition of the matrix B into two submatrices. As shown in Equation 2.9, the numerator of the transfer function is separated into three matrices. Lower-Upper (triangular) Decomposition is used here to decompose the matrix B into a product of two matrices using Matlab.

$$
p(z_1, z_2) = \sum_{i=0}^{M_1} \sum_{j=0}^{M_2} b_{i,j} z_1^{-i} z_2^{-j} = Z_1^T \cdot B \cdot Z_2 = Z_1^T \cdot L \cdot U \cdot Z_2^T = (Z_1^T \cdot L)(U \cdot Z_2) \quad (3.15)
$$

and from Matlab, matrix B is:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.026739327677625 & 1 & 0 & 0 \\ 0.081391044114528 & -0.385454286773350 & 1 & 0 \\ 0.980278299363733 & 0.598912923175959 & 0.166861870971349 & 1 \end{bmatrix} \tag{3.16}$$

$$U = \begin{bmatrix} 0.0943 & 0.0025215186 & 0.00767517545 & 0.00924402437 \\ 0 & -0.000005954980906 & 0.000001995405592 & -0.000003566522894 \\ 0 & 0 & -0.000000479927738 & -0.000000259751483 \\ 0 & 0 & 0 & -0.000000139588430 \end{bmatrix} \tag{3.17}$$

The denominator coefficients remain the same as realization one.

$$\begin{aligned} \alpha_0 &= \beta_0 = 1 \\ \alpha_1 &= \beta_1 = -1.5151 \\ \alpha_2 &= \beta_2 = 1.236274491 \\ \alpha_3 &= \beta_3 = -0.3133116076 \end{aligned}$$

However, the numerator coefficients is less straightforward. According to Equation 2.14 and Equation 2.15, the filter coefficients that are z_1-dependent are denoted g_{k1} and filter coefficients that are z_2-dependent are denoted h_{kj}, where k represents the corresponding filter coefficients for each set of subfilters containing only one z_1 subfilter and one z_2 subfilter.

$$
\begin{aligned}
g_{00} &= 1 \\
g_{01} &= 0.0267393276 \\
g_{02} &= 0.0813910441 \\
g_{03} &= 0.9802782993 \\
g_{10} &= 0 \\
g_{11} &= 1 \\
g_{12} &= -0.3854542867 \\
g_{13} &= 0.5989129231 \\
g_{20} &= 0 \\
g_{21} &= 0 \\
g_{22} &= 1 \\
g_{23} &= 0.1668618709 \\
g_{30} &= 0 \\
g_{31} &= 0 \\
g_{32} &= 0 \\
g_{33} &= 1
\end{aligned}
$$

$$
\begin{aligned}
h_{00} &= 0.0943 \\
h_{01} &= 0.0025215186 \\
h_{02} &= 0.00767517545 \\
h_{03} &= 0.00924402437 \\
h_{10} &= 0 \\
h_{11} &= -0.0000059549 \\
h_{12} &= 0.0000019954 \\
h_{13} &= -0.0000035665 \\
h_{20} &= 0 \\
h_{21} &= 0 \\
h_{22} &= -0.0000004799 \\
h_{23} &= -0.0000002597 \\
h_{30} &= 0 \\
h_{31} &= 0 \\
h_{32} &= 0 \\
h_{33} &= -0.0000001395
\end{aligned}
$$

3.3 Realization Three

In order to produce the filter coefficients for realization three, the transfer function is partial fraction expanded and then its fraction numerator is solved. The chosen transfer function, however, contains four complex roots and two real roots. As a result, each two complex conjugate roots are combined to produce a real second order polynomial.

According to Equation 2.21, a 2D separable denominator digital filter of order 3×3 has the general partial fraction expanded form:

$$
\begin{aligned}
H(z_1, z_2) &= \frac{K_1}{(1-\gamma_1 z_1^{-1})(1-\lambda_1 z_2^{-1})} + \frac{K_2}{(1-\gamma_1 z_1^{-1})(1-\lambda_2 z_2^{-1})} + \\
&\quad \frac{K_3}{(1-\gamma_1 z_1^{-1})(1-\lambda_2^* z_2^{-1})} + \frac{K_4}{(1-\gamma_2 z_1^{-1})(1-\lambda_1 z_2^{-1})} + \\
&\quad \frac{K_5}{(1-\gamma_2 z_1^{-1})(1-\lambda_2 z_2^{-1})} + \frac{K_6}{(1-\gamma_2 z_1^{-1})(1-\lambda_2^* z_2^{-1})} + \\
&\quad \frac{K_7}{(1-\gamma_2^* z_1^{-1})(1-\lambda_1 z_2^{-1})} + \frac{K_8}{(1-\gamma_2^* z_1^{-1})(1-\lambda_2 z_2^{-1})} + \\
&\quad \frac{K_9}{(1-\gamma_2^* z_1^{-1})(1-\lambda_2^* z_2^{-1})} + \frac{K_{10}}{(1-\gamma_1 z_1^{-1})} + \frac{K_{11}}{(1-\gamma_2 z_1^{-1})} + \\
&\quad \frac{K_{12}}{(1-\gamma_2^* z_1^{-1})} + \frac{K_{13}}{(1-\lambda_1 z_2^{-1})} + \frac{K_{14}}{(1-\lambda_2 z_2^{-1})} + \frac{K_{15}}{(1-\lambda_2^* z_2^{-1})} + K_{16} \\
&= \frac{K_1}{(1-\gamma_1 z_1^{-1})(1-\lambda_1 z_2^{-1})} + \\
&\quad \frac{(K_2+K_3)z_2^{-1} + (K_2+K_3)}{(1-\gamma_1 z_1^{-1})(1-\beta_1 z_2^{-1} - \beta_2 z_2^{-2})} + \\
&\quad \frac{(K_4+K_7)z_1^{-1} + (K_4+K_7)}{(1-\alpha_1 z_1^{-1} - \alpha_2 z_1^{-2})(1-\lambda_1 z_2^{-1})} + \\
&\quad \{(K_5 \cdot \gamma_2^* \cdot \lambda_2^* + K_9 \cdot -\gamma_2 \cdot -\lambda_2) z_1^{-1} z_2^{-1} + \\
&\quad (K_5 \cdot -\gamma_2^* + K_9 \cdot -\gamma_2) z_1^{-1} + (K_5 \cdot -\lambda_2^* + K_9 \cdot -\lambda_2) z_2^{-1} + (K_5 + K_9)\} \cdot \\
&\quad \frac{1}{(1-\alpha_1 z_1^{-1} - \alpha_2 z_2^{-1})(1-\beta_1 z_1^{-1} - \beta_2 z_2^{-1})} + \\
&\quad \{(K_8 \cdot -\gamma_2 \cdot -\lambda_2^* + K_6 \cdot -\gamma_2^* \cdot -\lambda_2) z_1^{-1} z_2^{-1} + \\
&\quad (K_8 \cdot -\lambda_2^* + K_6 \cdot -\lambda_2) z_2^{-1} + (K_8 \cdot -\gamma_2 + K_6 \cdot -\gamma_2^*) z_1^{-1} + (K_8 + K_6)\} \cdot \\
&\quad \frac{1}{(1-\alpha_1 z_1^{-1} - \alpha_2 z_2^{-1})(1-\beta_1 z_2^{-1} - \beta_2 z_2^{-1})} + \\
&\quad \frac{K_{10}}{1-\gamma_1 z_1^{-1} 1} + \frac{(K_{11} \cdot -\gamma_2^* + K_{12} \cdot -\gamma_2) z_1^{-1} + (K_{11} + K_{12})}{1 - \alpha_1 z_1^{-1} - \alpha z_1^{-2}} + \\
&\quad \frac{K_{13}}{1-\lambda_1 z_2^{-1}} + \frac{(K_{14} \cdot -\lambda_2^* + K_{15} \cdot -\lambda_2) z_2^{-1} + (K_{14} + K_{15})}{1 - \beta_1 z_2^{-1} - \beta z_2^{-2}} + K_{16}
\end{aligned}
$$

After collecting similar powers and reducing, this simplifies to:

$$
\begin{aligned}
H(z_1, z_2) \;=\; & \frac{A z_1^{-1} z_2^{-1} + B z_1^{-1} + C z_2^{-1} + D}{(1 - \alpha_1 z_1^{-1} - \alpha_2 z_1^{-2})(1 - \beta_1 z_2^{-1} - \beta_2 z_2^{-2})} + && (3.18)\\
& \frac{E}{(1 - \gamma_1 z_1^{-1})(1 - \lambda_1 z_2^{-1})} + && (3.19)\\
& \frac{F z_2^{-1} + G}{(1 - \gamma_1 z_1^{-1})(1 - \beta_1 z_2^{-1} - \beta_2 z_2^{-2})} + && (3.20)\\
& \frac{H z_1^{-1} + I}{(1 - \alpha_1 z_1^{-1} - \alpha_2 z_1^{-2})(1 - \lambda_1 z_2^{-1})} + && (3.21)\\
& \frac{J}{(1 - \gamma_1 z_2^{-1})} + && (3.22)\\
& \frac{K z_1^{-1} + L}{(1 - \alpha_1 z_1^{-1} - \alpha_2 z_1^{-2})} + && (3.23)\\
& \frac{M}{(1 - \lambda_1 z_2^{-1})} + && (3.24)\\
& \frac{N z_2^{-1} + O}{(1 - \beta_1 z_2^{-1} - \beta_2 z_2^{-2})} + P && (3.25)
\end{aligned}
$$

Solving this system of 16 linear equations using matrices yields:

$$A \cdot x = B \tag{3.26}$$

where A is a matrix of dimension 16 $\times$ 16, and B is a column of dimension 16 $\times$ 1 and x represents the filter coefficients of dimension 1 $\times$ 16. Furthermore, matrix A is divided into four submatrices [1].

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \tag{3.27}$$

[1] $A1$, $A2$, $A3$, and $A4$ displayed here are with reduced precision due to page size restriction. Actual matrix precision is 10 digits after the decimal point.

$$A_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & -0.6636 & -1.1514 & 1 & -0.6636 & 0 \\ 0 & 0 & -0.6636 & 0 & 0.4721 & -0.6636 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.6636 & -1.1514 & 0 & -1.1514 & 1 \\ 1 & -0.6636 & -0.6636 & 0.4403 & 1.3259 & -1.1514 & 0.7641 & -1.1514 \\ -0.6636 & 0 & 0.4403 & 0 & -0.5436 & 0.7641 & 0 & 0.4721 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.28}$$

$$A_2 = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1.1514 & 1.8151 & 0 & 1.8151 & 1.1514 & 1 & 0.6636 & 1.8151 \\ 0.4721 & 1.2362 & 0 & 1.2362 & 0.4721 & -0.6636 & 0 & 1.2362 \\ 0 & -0.3133 & 0 & -0.3133 & 0 & 0 & 0 & -0.3133 \\ -0.6636 & -1.1514 & 1 & -0.6636 & -1.8151 & 0 & -1.8151 & -1.8151 \\ 0.7641 & 2.0900 & -1.8151 & 1.2045 & 2.0900 & -1.8151 & 1.2045 & 3.2945 \\ -0.3133 & -1.4235 & 1.2362 & -0.8203 & -0.8569 & 1.2045 & 0 & -2.2439 \\ 0 & 0.3607 & -0.3133 & 0.2079 & 0 & 0 & 0 & 0.5686 \end{bmatrix} \tag{3.29}$$

$$A_3 = \begin{bmatrix} 0 & -0.6636 & 0 & 0 & 0.4721 & 0 & 0.4721 & -0.6636 \\ -0.6636 & 0.4403 & 0 & 0 & -0.5436 & 0.4721 & -0.3133 & 0.7641 \\ 0.4403 & 0 & 0 & 0 & 0.2229 & -0.3133 & 0 & -0.3133 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.30}$$

$$A_4 = \begin{bmatrix} 0 & 0.4721 & -0.6636 & 0 & 1.2362 & 0 & 1.2362 & 1.2362 \\ 0 & -0.8569 & 1.2045 & 0 & -1.4235 & 1.2362 & -0.8203 & -2.2439 \\ 0 & 0.5836 & -0.8203 & 0 & 0.5836 & -0.8203 & 0 & 1.5283 \\ 0 & -0.1479 & 0.2079 & 0 & 0 & 0 & 0 & -0.3873 \\ 0 & 0 & 0 & 0 & -0.3133 & 0 & -0.3133 & -0.3133 \\ 0 & 0 & 0 & 0 & 0.3607 & -0.3133 & 0.2079 & 0.5686 \\ 0 & 0 & 0 & 0 & -0.1479 & 0.2079 & 0 & -0.3873 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.09816 \end{bmatrix} \tag{3.31}$$

and

$$B = \begin{bmatrix} 0.00943000000 \\ 0.00252151860 \\ 0.00767517545 \\ 0.00924402436 \\ 0.00252151860 \\ 0.00066828213 \\ 0.00205458571 \\ 0.00246822344 \\ 0.00767517546 \\ 0.00205458569 \\ 0.00624565635 \\ 0.00752492293 \\ 0.00924402436 \\ 0.00246822345 \\ 0.00752492293 \\ 0.00905939752 \end{bmatrix} \tag{3.32}$$

Solving the system of linear equations, by $x = A \backslash B$, the solution is obtained.

$$x = \begin{bmatrix} -2.1399700728 \\ 2.6315602263 \\ 1.4059630793 \\ -1.1315043079 \\ -1.4883426863 \\ -2.3631817875 \\ 2.2515878795 \\ -2.3631817864 \\ 0.52898585339 \\ -0.57836003994 \\ -0.20336809040 \\ 0.45656755241 \\ -0.57836003983 \\ -0.20336809035 \\ 0.45656755231 \\ 0.092288236346 \end{bmatrix} \tag{3.33}$$

Referring to Equation 3.18 to Equation 3.25, the coefficients are then [2]:

[2]Variable A and B presented here are coefficients for the partial fraction expanded 2D transfer function in Equation 3.18

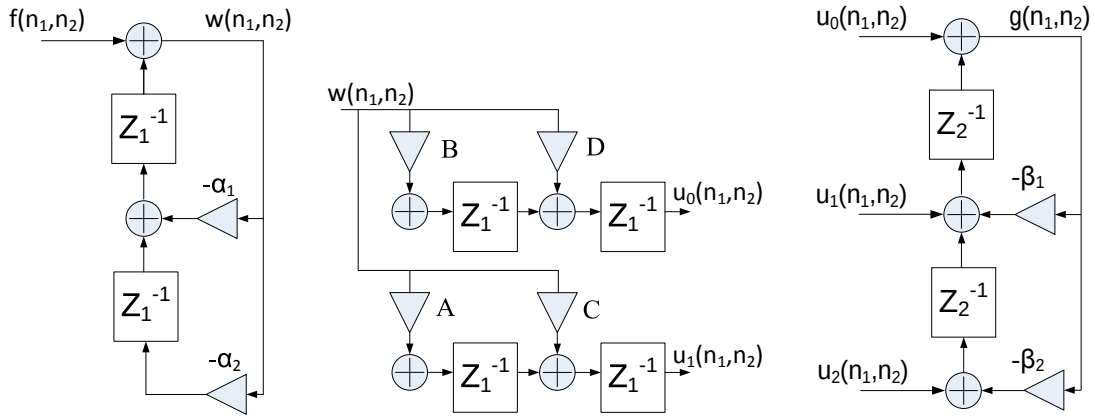
$$
\begin{aligned}
A &= -2.1399700728 \\
B &= 2.6315602263 \\
C &= 1.4059630793 \\
D &= -1.1315043079 \\
E &= -1.4883426863 \\
F &= -2.3631817875 \\
G &= 2.2515878795 \\
H &= -2.3631817864 \\
I &= 0.52898585339 \\
J &= -0.57836003994 \\
K &= -0.20336809040 \\
L &= 0.45656755241 \\
M &= -0.57836003983 \\
N &= -0.20336809035 \\
O &= 0.45656755231 \\
P &= 0.092288236346
\end{aligned}
$$

3.3.1 Modification to Realization Three

In Section 2.4, an architecture suitable for $M_1 \times M_2$ filter order is presented. However, the realization presented assume all the roots are real. As shown in Equation 3.18 to Equation 3.25, the roots are not all real. The roots shown in the numerical example are a single real, a complex, and a complex conjugate root in both the z_1 and z_2 direction. Due to this problem, the realization needs to be modified before it is suitable for implemention.

In order to accommodate complex roots, the complex roots are combined to form a real polynomial of order two that is realized using transposed direct form II. Figure 3.1(a) is the subfilter for realizing the combined complex roots in the z_1 direction and Figure 3.1(b) for the z_2 direction. Since the numerator is no longer a constant, but also contains coefficients multiplied by delays in z_1 and z_2 dimension, the numerator needs to to be modified as well. The transposed direct form of FIR filter is suitable to implement the numerator. Figure 3.1(c) shows the modified realization to implement the numerator portion of the first double root containing coefficients A, B, C and D in the form:

$$\frac{Az_1^{-1}z_2^{-1} + Bz_1^{-1} + Cz_2^{-1} + D}{(1 - \alpha z_1^{-1} - \alpha z_1^{-2})(1 - \beta z_2^{-1} - \beta z_2^{-2})} \tag{3.34}$$



(a) Modified z_1 subfilter (b) Modified multiplier subfilter (c) Modified z_2 subfilter

Figure 3.1: Implementation for the Numerator and the Denominator for Equation 3.34.

A detailed subfilter realization is shown in Appendix A.3.

3.4 Realization Four

Realization four is a modified version of realization three, where redundant implementation of the same pole is avoided. Coefficients involving the same pole are summed using a tree adder and then multiplied by the pole. As such, the coefficients used for realization four are the same as the coefficients used for realization three, which are presented in Section 3.3.

Writing realization four in matrix form using the derived coefficients A to P:

$$H(z_1, z_2) = P^T(z_1) \cdot K \cdot Q(z_2) \tag{3.35}$$

where

$$P(z_1) = \begin{bmatrix} 1 \\ \frac{1}{(1-\gamma_1 z_1^{-1})} \\ \frac{1}{(1-\alpha_1 z_1^{-1}-\alpha_2 z_1^{-2})} \end{bmatrix}, Q(z_2) = \begin{bmatrix} 1 \\ \frac{1}{(1-\lambda z_2^{-1})} \\ \frac{1}{(1-\beta_1 z_1^{-1}-\beta_2 z_1^{-2})} \end{bmatrix} \tag{3.36}$$

$$K = \left[\begin{array}{c|c|c} P & M & Nz_2^{-1}+O \\ J & E & Fz_2^{-1}+G \\ Kz_1^{-1}+L & Hz_1^{-1}+I & Az_1^{-1}z_2^{-1}+Bz_1^{-1}+Cz_2^{-1}+D \end{array}\right] \tag{3.37}$$

The filter realization have the structure similar to realization three. The circuitry that realizes the z_1 and z_2 poles and the coefficients is the same as the one shown in Figure 3.3.1. The difference, however, is that this realization does not realize the same pole multiple times. As a result, the wire connection will be different for realization four from realization three. A detailed subfilter realization is shown in Appendix A.4.

Chapter 4

Timing Considerations on Filter Implementation

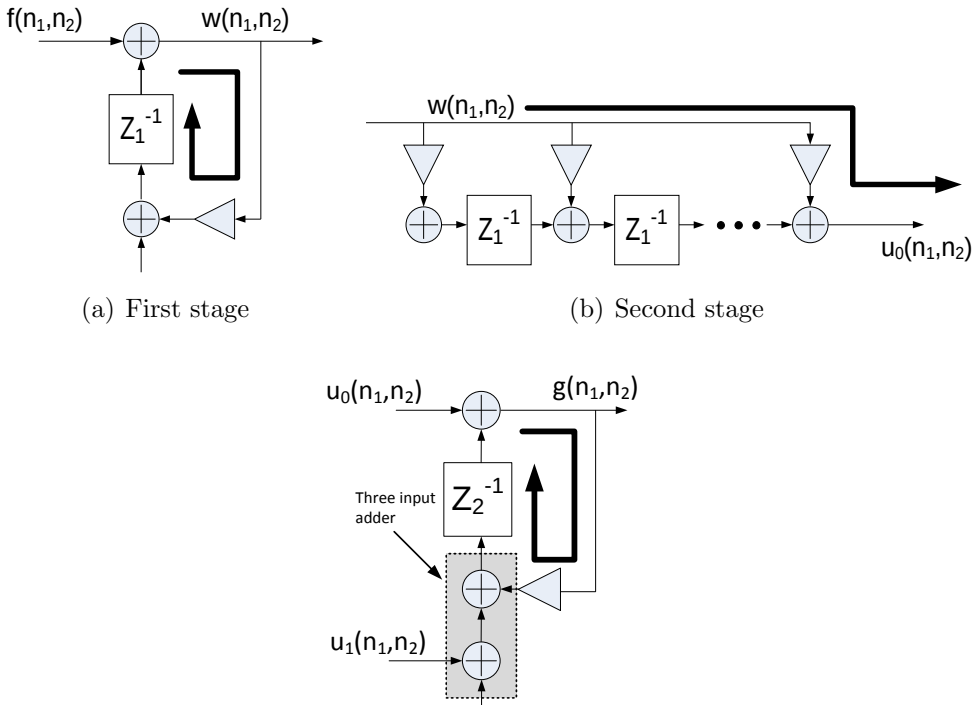
Before the system architecture of the digital signal processor can be presented, the timing of the filter realizations must be first investigated. As a whole, digital circuits are composed of two types of paths: data paths and control paths. Data paths propagate the data along an arithmetic channel while the control paths moderate the flow and the direction of the data traveling on the data paths. Some paths are timing critical, which implies that the paths are affecting the speed at which the design can operate. Therefore critical paths must be identified and optimized in order to improve the filter throughput.

Since the feedback paths stipulate the shortest time for which the output of the previous subfilter can be transferred to the input of the following subfilter, the feedback paths are also considered the critical paths. In this chapter, the critical path for each of the four realizations are identified and modifications are made to improve the performance.

4.1 Critical Data Path in Realization One

Since realization one is comprised of three subfilters, each subfilter will have a critical path. The critical path of the implementation is therefore the maximum of the three critical paths. For the first subfilter, the critical path is two adds and one multiply. For the second subfilter, the critical path is one multiply and one add. For the third subfilter, the critical path is two adds and one multiply. The three input adder in the third subfilter is broken down into two adders connected in series. The critical paths for the first, second and third

stages are shown in Figure 4.1. The actual throughput delay for the implementation is therefore two adds, one multiply plus one additional clock cycle required for the horizontal delay element to transfer the data from previous stage to the next stage. Next, since the subfilters are connected in cascade, critical path for the overall realization must be identified.



(a) First stage

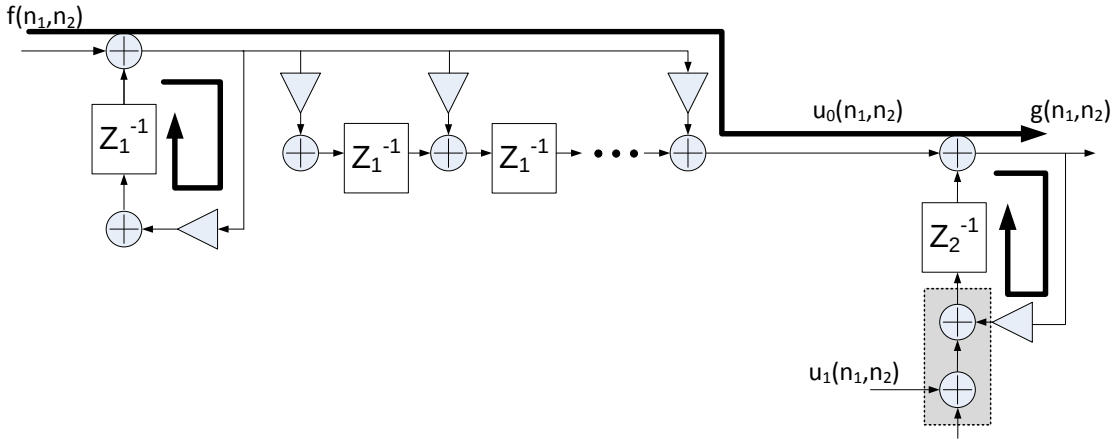
(b) Second stage

(c) Third Stage

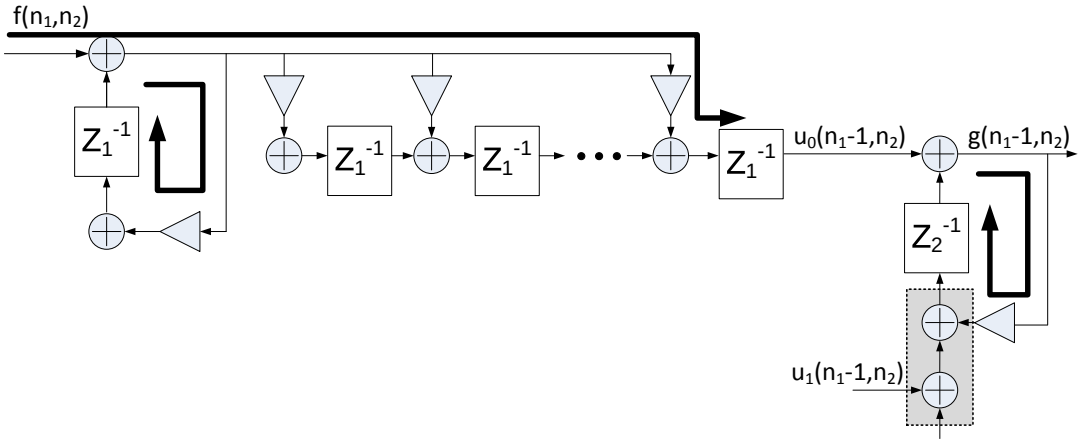
Figure 4.1: Critical Paths in First, Second and Third Subfilter of Realization One.

In the original realization shown in Figure 2.1, the critical path with the three subfilters combined is three add and one multiply, which is larger than the critical path of the individual subfilters. If the design remains unchanged, the throughput of the implementation will be three add and one multiply as shown in Figure 4.2(a) since the longest critical path dictates how fast the entire realization can be clocked. To increase the throughput, an additional horizontal delay is introduced at the output of the second subfilter. This horizontal delay breaks the overall critical path such that the overall implementation retains a throughput of two add and one multiply. However, since an extra horizontal delay is added, the output $u_i(n_1, n_2)$ and $g(n_1, n_2)$ now have an additional horizontal delay, as shown in

Figure 4.2(b). This extra horizontal delay adds to the overall latency of the digital filter, but does not affect the processed data in any way.



(a) Critical path of the overall realization

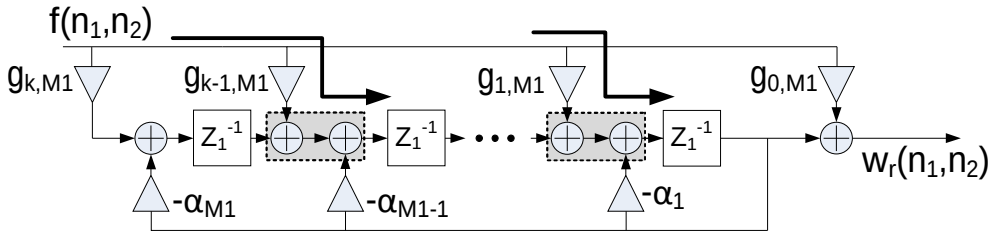


(b) Modified critical path of the overall realization

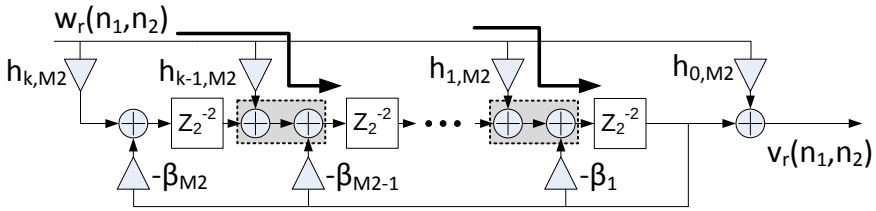
Figure 4.2: Critical Path and Modified Critical Path for Realization One.

4.2 Critical Data Path in Realization Two

Realization two contains two subfilters and a tree adder. Each subfilter has its own critical path, while the tree adder can be adjusted to accommodate the critical path found in the previous two subfilters. For the first subfilter, the critical path is one multiply and two adds. For the second subfilter, the critical path is also one multiply and two adds. The critical paths for the first and second subfilter is shown in Figure 4.2.



(a) First stage

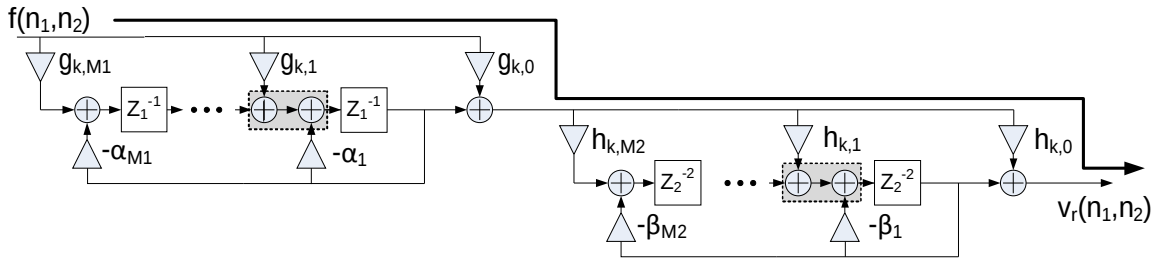


(b) Second stage

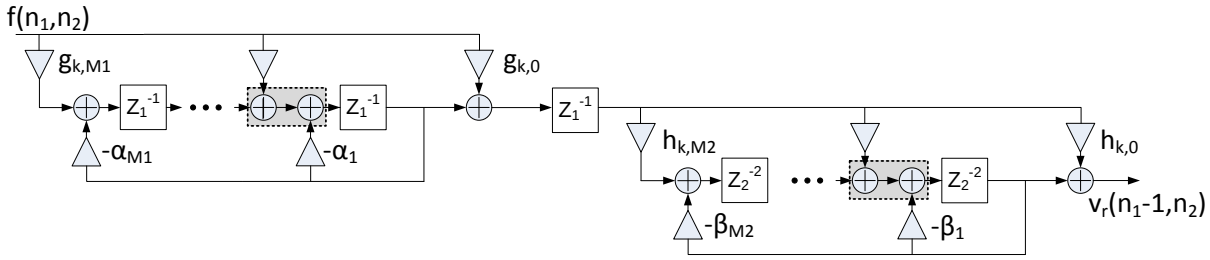
Figure 4.3: Critical Paths in First, Second and Third Stages of Realization Two.

In the original realization shown in Figure 2.3, the critical path with the two subfilters combined is two add and two multiply, which is larger than the critical path of the individual subfilters. If the design remains unchanged, the throughput of the implementation will be two add and two multiply as shown in Figure 4.4(a) since the longest critical path dictates how fast the entire realization can be operated. To increase the throughput, an additional horizontal delay is introduced at the output of the first subfilter. This horizontal delay breaks the overall critical path so that the overall implementation retains a throughput of two add and one multiply. Since an extra horizontal delay is added, the output $g(n_1, n_2)$ now has an additional horizontal delay, as shown in Figure 4.4(b). This extra horizontal delay increases the overall latency of the digital filter by one, but does not affect the output data. Since the minimum throughput is determined to be two add and one multiply, the tree adder that sums up all the $v_r(n_1, n_2)$ can now be determined to have two adds in its

critical path.



(a) Critical path of the overall realization

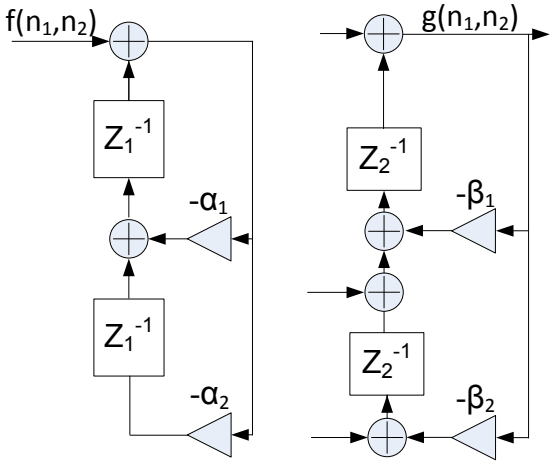


(b) Modified critical path of the overall realization

Figure 4.4: Critical Path and Modified Critical Path for Realization Two.

4.3 Critical Data Path in Realization Three and Four

Despite the structural difference between realization three and realization four, the basic subfilter that is used to construct realization three and realization four is the same. The z_1 and z_2 denominator realization is shown in Figure 4.5(a) and 4.5(b). In fact, this same denominator realization is used in realization one. The critical path is then two adds and one multiply as previously discussed.



(a) z_1 denominator realization (b) z_2 denominator realization

Figure 4.5: Denominator Realization for Realization Three and Four.

The numerator realization is similar to realization one. One horizontal delay is inserted at the end of the FIR filter chain to ensure a throughput of two add and one multiply. The numerator subfilter which realizes coefficient A, B, C, and D is shown in Figure 4.6.

Both realization three and four share the same subfilters for denominator and numerator realization. The only difference is realization four eliminates multiple realization of the same pole. This results in reduced hardware utilization and lower power dissipation. Appendix A.3 shows the detailed realization three subfilters and Appendix A.4 shows the detailed realization four subfilters.

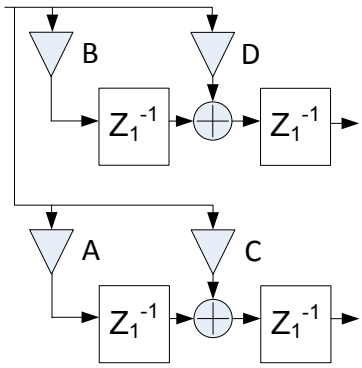
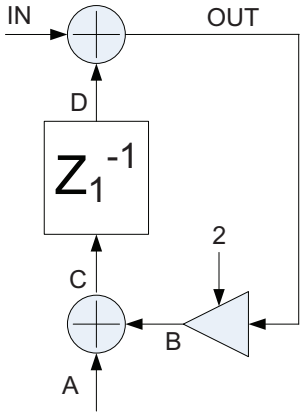


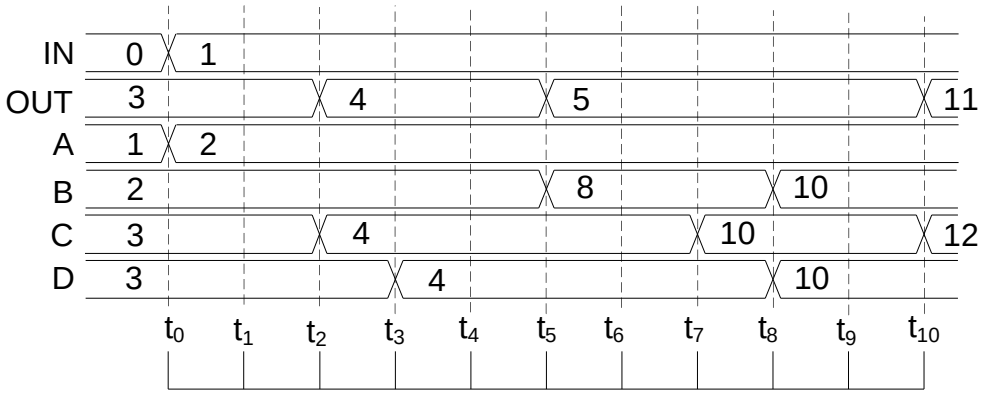
Figure 4.6: Numerator Realization

4.4 Control Path

Since adders, multipliers and shift registers are connected in series for all the realizations, control paths are required to reduce inadvertent data shifts. An example is provided to illustrate the significance of control paths. The example used for this section is subfilter one of realization one, as shown in Figure 4.7(a). It will be used to illustrate how incorrect data can be inadvertently shifted without proper control paths.



(a) Realization One subfilter one



(b) Timing diagram without control path

Figure 4.7: Example Timing Diagrams Without Control Paths.

For the sake of simplicity, an adder latency of 2 clock cycles, multiplier latency of 3 clock cycles, and shift register latency of 1 clock cycle is assumed. Figure 4.7(b) shows

the timing diagram of such a circuit assuming both adders, multiplier and shift register are operating simultaneously. Before zero time unit (t_0), IN is given the initial value of 0, A the value of 1, D the value of 3. Since OUT is the sum of IN and D, it is given the value of 3. For the same reason, C is the sum of A and B, and is given the value of 3, provided that the coefficient for the multiplier is 2 as shown. At zero time unit (t_0), IN switches from 0 to 1 and A switches from 1 to 2. After 2 time unit (t_2), adder on the top has finished calculating and OUT switches from 1 to 4. Since the adder on the bottom is calculating simultaneously as the top adder, C switches from 3 to 4. After 1 time unit (t_3), C has propagated to D via shift register, and D switches from 3 to 4. After 2 time unit (t_5), the multiplier has finished calculating and B switches from 2 to 8. At the same time, OUT switches from 4 to 5 due to a change on D. This result is undesirable since the change on OUT is propagated due to toggling of A to C to D. The correct result is finally attained after 5 time unit (t_{10}), when OUT switches from 5 to 11. Without proper control, the incorrect intermediate sum will propagate further down the signal processing pipeline and results in rapid toggling at the output. Clearly a control circuit is needed to remedy the problem.

To remedy the propagation of incorrect intermediate sums, a control circuitry is used to enable the adders, multiplier and shift register at the proper time. To produce the correct sum for the sample circuit in Figure 4.7(a), the top adder, the multiplier, the bottom adder, and the shift register must be enabled in this order. Figure 4.8 shows the enable signals that drive the corresponding output. ENA_OUT signals the top adder to turn on for two clock cycles to produce the correct sum at t_2. Following that, ENA_B signals the multiplier to turn on for 3 clock cycles to produce the correct product at t_5. ENA_C and ENA_D signals are applied to the bottom adder and the shift register to produce the correct sum. Notice the erroneous value of 5 no longer appears on OUT at t_5 using this control scheme.

Since all the realizations use a similar configuration of a multiplier, adders, and shift registers, the control scheme is applied to all the realizations to prevent incorrect calculations from propagating down the signal processing pipeline. This change in architecture is of course subject to trade-offs. Since extra circuitry is used to produce the correct enable signals for the corresponding adders, multipliers and shift registers, the hardware utilization will increase. Furthermore, additional routing resources are needed to fit the design as well.

Even though the same control scheme is applied to all four realizations, the exact cycle time required is different from one realization to another. The timing sequence shown in Figure 4.8 applies directly to realization one and realization two, since the critical path is two adds, one multiply and one horizontal shift. For realization three and realization four, however, the timing sequence is different. The critical path is actually one add, one

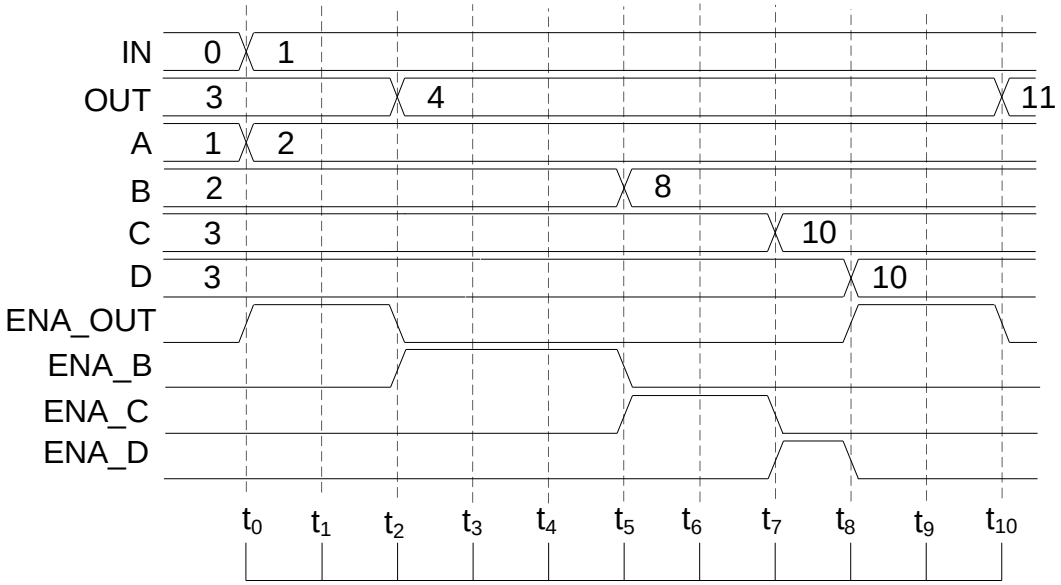


Figure 4.8: Timing Diagram with Control Path.

Table 4.1: Cycle Time for Each Realization.

Realiation	Cycle Time
One	$2t_{add} + t_{mult} + 1$
Two	$2t_{add} + t_{mult} + 1$
Three	$t_{add} + t_{mult} + 1$
Four	$t_{add} + t_{mult} + 1$

multiply and one horizontal shift. Table 4.1 summarizes the cycle time for each realization.

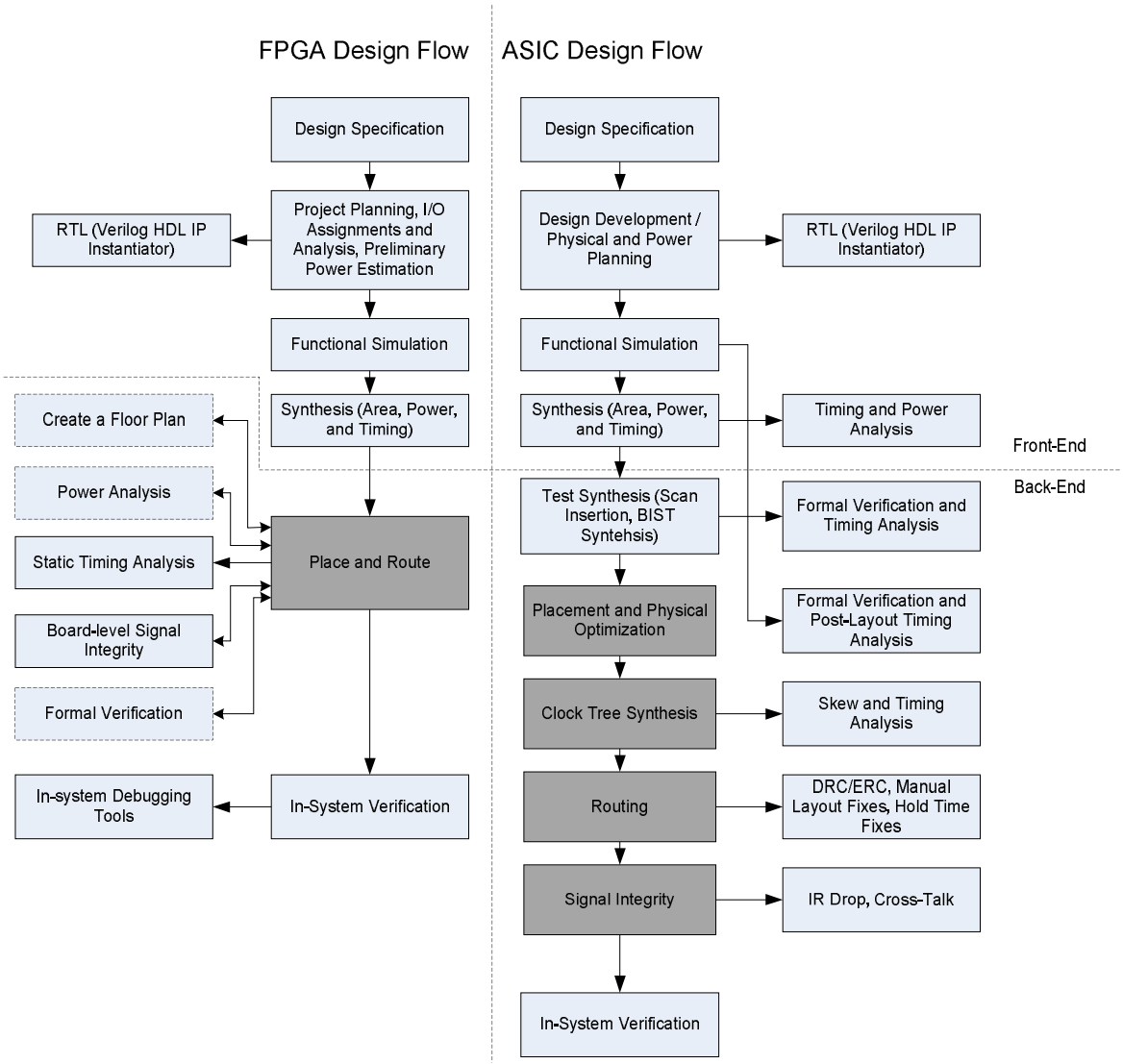
Chapter 5

System Implementation Details

To implement a 2D separable denominator digital filter and use it to process two dimensional data, a complete system is required to receive input from a source, buffer the input data, process the data using the digital filter, buffer the output data, and send the output data back to the source.

There are two ways to produce a robust and reusable system: full-custom ASIC flow or FPGA flow. A full-custom ASIC flow is a long and arduous process involving many specific design tools, as shown on the right hand side of Figure 5.1. The circuitry is first designed, functionally simulated, synthesized, layed out, manufactured and then tested. The entire process can occupy months for just one iteration of the design. The finished product is much more compact and better optimized in both power and performance when compared to an FPGA board.

On the contrary, the FPGA design flow is much more flexible and forgiving when compared to the ASIC flow. One iteration of the design from synthesis, place and route to testing can be done in a fraction of time that ASIC flow takes. The FPGA board can be reused many times, unlike ASIC which needs to be replaced with each new revision. Furthermore, many FPGA vendors handle both the front-end as well as back-end tasks in one manageable piece of software, saving both cost and precious development time. As a result, the digital signal processing system is built using a FPGA design flow, as shown on the left hand side of Figure 5.1. In the next section, the advantages and disadvantages of using fixed point versus floating point arithmetic for the digital signal processing system is discussed.


FPGA Design Flow
ASIC Design Flow
Design Specification
Project Planning, I/O Assignments and Analysis, Preliminary Power Estimation
RTL (Verilog HDL IP Instantiator)
Functional Simulation
Synthesis (Area, Power, and Timing)
Create a Floor Plan
Power Analysis
Static Timing Analysis
Board-level Signal Integrity
Formal Verification
Place and Route
In-system Debugging Tools
In-System Verification
Design Specification
Design Development / Physical and Power Planning
RTL (Verilog HDL IP Instantiator)
Functional Simulation
Synthesis (Area, Power, and Timing)
Timing and Power Analysis
Front-End
Back-End
Test Synthesis (Scan Insertion, BIST Syntehsis)
Formal Verification and Timing Analysis
Placement and Physical Optimization
Formal Verification and Post-Layout Timing Analysis
Clock Tree Synthesis
Skew and Timing Analysis
Routing
DRC/ERC, Manual Layout Fixes, Hold Time Fixes
Signal Integrity
IR Drop, Cross-Talk
In-System Verification

5.1 Fixed Point versus Floating Point

Traditional digital filters are implemented with fixed point calculations in mind, with which calculations are made with limited precision. Nowadays, hardware resources are becoming much cheaper and it is not unconventional to find floating point digital filters with much higher precision, albeit at the cost of higher hardware utilization as well. It is not an easy choice to make when it comes down to using fixed point or floating point arithmetic units, since both fixed point and floating point have advantages and disadvantages.

Fixed point processing is susceptible to finite word length effects, which occurs when the word length of the registers is less than the required precision needed to store the actual value. These effects introduce noise into the Digital Signal Processing system and create undesirable behavior, such as input quantization noise, coefficient quantization noise and arithmetic overflow [1].

Input quantization noise arises from the limited precision of the Analog to Digital Converter, when a continuous time signal is converted to a discrete time signal. For the case of filters, the input data is usually pre-sampled and quantized to acceptable levels before it is fed to the processors. As a result, input quantization noise is less of an issue for the matter at hand since it is assumed that this digital filter is a part of a larger system that reduce input quantization noise to an acceptable level.

On top of input quantization noise, there can also be noise associated with the coefficients that are used to describe the transfer function of the filter. Usually the coefficients assume the system has infinite precision. The Digital Signal Processing system has a limited amount of memory and is forced to truncate the coefficients. What is more, truncated coefficients no longer retain the precision of the original coefficients and can affect pole/zero locations, thus altering the frequency response of the digital filter. This effect is especially critical for IIR filters where the movement of the pole locations may lead to instability.

Lastly, addition and multiplication are two common operations in a Digital Signal Processing system. Multiplying two 16 bit number requires a 32 bit register to store the result. If the result is then multiplied with another 16 bit number, 48 (32 + 16) bits is required to store the result. If there are multiple multiplications within the systems, the number of bits required to store the intermediate output can increase very quickly. As a result, in a fixed point Digital Signal Processing system, results are usually rounded or truncated. Unfortunately, rounding limits the growth of word length at the expense of increased roundoff errors. The larger the Digital Signal Processing system, the more roundoff errors the result can accumulate.

To deal with the shortcomings of fixed point Digital Signal Processing systems, floating

Table 5.1: IEEE 745 Single and Double Precision Floating Point Internal Representation

Type	Sign	Exponent	Signficant	Total Bits
Single Precision	1	8	23	32
Double Precision	1	11	52	64

point adders and multipliers are used in the system. Floating point operations are intrinsically more expensive than fixed point operations since floating point numbers take more bits to encode. Floating point operations require the operands to be aligned before calculation can begin, a process called denormalization. After calculation is performed, the result needs to be converted back to single precision floating number, a process called normalization. Researchers agree in a pipelined DSP system, the denormalization and normalization stage of the floating point operators can be fused to reduce hardware complexity and increase performance [16]. For this research, this particular optimization technique was not applied since it requires C code. This idea, however, can be applied if the filter realizations are to be implemented on ASIC since the designer typically has more control in the design and layout on ASIC than a FPGA.

The standard IEEE 745 describes five floating point standards and two of these are popular among modern computing hardware: single precision and double precision. The internal representation of a single precision and a double precision floating point number is shown in Table 5.1.

Single precision floating point numbers are much like numbers represented in scientific notation. The sign bit represents the sign of the number. The exponent is an 8 bit signed integer ranging from -128 to 127. The significant is stored with an implicit leading bit (to the left of the binary point) with a value of 1 unless the significant is zero. The IEEE 745 standard also lists four floating point number special cases: signed zero, subnormal numbers, infinities, and not a number.

- Signed zero: Zero can represented as +0 or -0 in the IEEE floating point standard. The two values are numerically equal in computations. However, different operations can produce either +0 or -0.
- Subnormal numbers: Subnormal values occur when the result of a floating point operation is smaller in magnitude than the smallest possible representable value in the floating point number representation. Subnormal numbers are usually handled by the hardware.

- Infinities: Positive or negative infinity can result when a divide by zero exception occurs. It is not an error and can occur just as often as any regular number.
- Not a number (NaN): Not a number is the result of an invalid floating point operation. For example, dividing zero by zero or taking the square root of negative one. Using a NaN as an operand in an arthimetic operation will cause the result to be an NaN as well.

Fortunately, the Altera Quartus II IP MegaCore library comes with Floating Point (FP) adders and multipliers that can be configured to use single precision or double precision floating point, including the option to flag signed zero, subnormal numbers, infinities and NaNs. A design can also configure the FP multipliers and adders for different levels of pipeline depth which can affect the overall operating speed, power and area of the Digital Signal Processing system. For the implementation of this Digital Signal Processing system, IEEE single precision FP multipliers and adders are used to construct the system, since the system does not require the precision of double FP operators. In the next section, the optimum level of pipeline for a floating point multiplier and an adder is discussed.

5.2 Multiplier and Adder Pipeline Depth

Altera Quartus II Megafunction Wizard provides different pipeline depths for the FP Adders and Multipliers. For the FP Multiplier, pipeline depths of 5, 6, 10 and 11 can be selected; for the FP Adder, pipeline depths of 7, 8, 9, 10, 11, 12, 13, and 14 can be selected. This provides a trade-off between performance and register count. Figures 5.2 and 5.3 show the maximum operating frequency (f_{max}) and the register count versus the number of pipeline stages for FP Adder and Multiplier, respectively.

For the FP Multiplier, a maximum speed of 440 MHz is achieved using pipeline depths of 10 or 11. Since there is no performance increase by choosing a pipeline depth of 11, it would save some hardware resource by using pipeline depth of 10 for the FP Multiplier. For the FP Adder, a maximum speed of 482 MHz is achieved when the pipeline depth is 14. However, since the maximum speed of the design is restricted by the FP Multiplier's 440 MHz, it is unnecessary to use pipeline depth of more than 12 for the FP Adder. For constructing the Digital Signal Processing system, a pipeline depth of 10 is used for the FP Multiplier and a pipeline depth of 12 is used for the FP Adder. It should be noted that a speed difference of 100 MHz is observed when FP adder pipeline depth changes from 11 to 12. This sudden change in performance is unexpected. Documentations from Altera did not explain this drastic difference in performance.

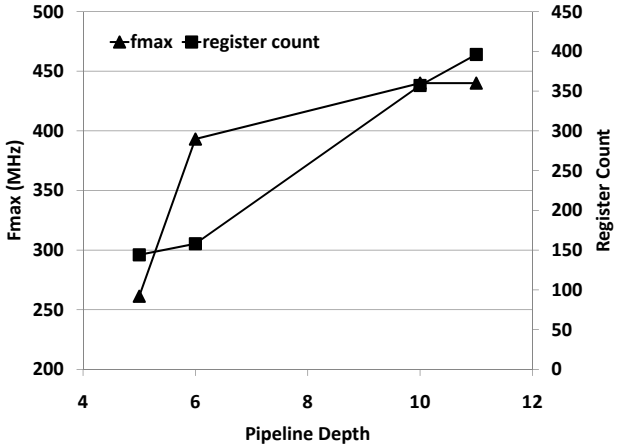


Figure 5.2: FP Multiplier $f_{\max}$ and Register Count versus Pipeline Depth

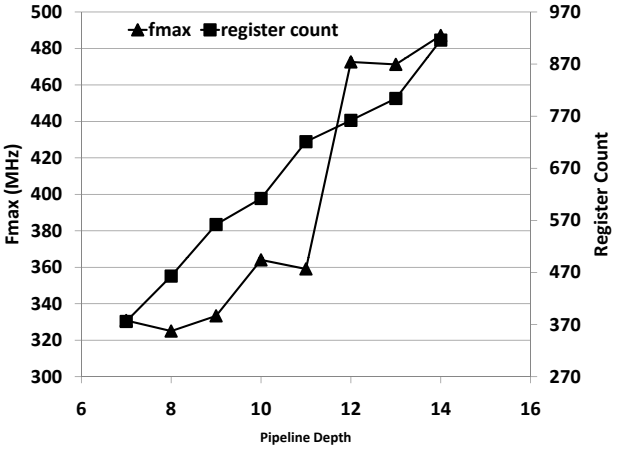


Figure 5.3: FP Adder $f_{\max}$ and Register Count versus Pipeline Depth

The hardware resource utilization for the selected multiplier and adder is indicated in Table 5.2. This information allows us to make a quantitative comparison on resource utilization from the realization to the implementation later on.

Table 5.2: Hardware Utilization for FP Multiplier and FP Adder.

FP Operator	Combinational ALUTs	Register Count	DSP 18 bit Multiplier	Latency
Multiplier	535	756	4	10
Adder	126	357	0	12

5.3 Shift Registers

Since the horizontal shift (z_1) is only one delay in pixel, it can be easily constructed using flip flops in parallel. The vertical shift (z_2), however, contains a line worth of pixels that is the width of the image. For a 64×64 image, each vertical shift needs to be 64 pixels deep. To accomplish this, a RAM-based shift register is used from the MegaWizard Plugin manager. The shift register is 32 bits wide to accommodate the single precision floating point algorithm used in the digital signal processing pipeline and is 64 elements deep to accommodate the image width. The hardware resource utilization used for horizontal and vertical shift register is shown in Table 5.3.

Table 5.3: Hardware Utilization for Horizontal and Vertical Shift Register.

Shift Register	Combinational ALUTs	Register Count	Block Memory Bits
Horizontal	0	32	0
Vertical	24	13	1984

5.4 Implementation of a DSP System

A generic system is required that allows test data to be passed into the 2D digital filter and allows it to process and transport the processed data back for display. The system must be able to transport the data in a timely manner and without fault. Furthermore, the system must be relatively cheap to construct. First, the communication protocol must be determined, which is discussed in subsection 5.4.1. Second, the overall DSP system architecture must be studied and developed, which is discussed in Subsection 5.4.2.

5.4.1 Communication Protocol to the FPGA

The Altera DE4 board provides many ways to accomplish the communication task since the board contains an array of high-speed IOs including Serial ATA, Gigabit Ethernet, PCI Express, Universal Serial Bus 2.0, and JTAG. The relevant specifications for each IO interface are listed below [17].

- Serial ATA 3.0: Support standard 6 Gbps signal rate.
- Gigabit Ethernet: Support Ethernet frame at 1 Gbps.
- PCI Express x8: Support speed for first generation at 2.5 Gbps/lane and second generation at 5 Gbps/lane. Can interface with PC motherboard with x8 or x16 PCI Express slot.
- USB 2.0: Support speed up to 480 Mbps. Also support both USB host and device.
- JTAG: Used to program and debug the FPGA via USB adapter. Speed is limited by the USB medium.

In fact, all these communication protocols are all viable choices and the only difference is the communication speed. Altera has also provided sample applications which conveniently work out of the box. The choice is then arbitrary and can be easily converted if more

communication protocols need to be included. The JTAG protocol is the ideal protocol to start with since it uses the existing USB cable that configures the FPGA and is the simplest protocol to use.

In essence, the USB cable is used to download a configuration bit stream to the FPGA device. Once the device powers up, the USB cable switches over to JTAG mode and enables debugging of onboard components using either the Avalon Memory Mapped (MM) interface or Streaming (ST) interface. There are hardware IPs embedded on the FPGA board that enable the required hardware transaction. The details of these hardware IPs are not discussed since they are beyond the scope of this thesis. More details with regard to Avalon MM interface and ST interface are discussed in Subsection 5.4.2.

5.4.2 DSP System Architecture

The proposed architecture of the digital signal processing system is shown in Figure 5.4. The DSP system is first configured via a byte stream from the host PC using the USB Blaster [1]. JTAG to Avalon MM interface is used to write filter coefficients to the digital filter. The actual image data is streamed into the input FIFO, processed by the digital filter, streamed back to the output FIFO, and then returned to the host PC.

The image data used for digital filtering is a gray scale image of 64×64 pixels. However, the image data streamed into the digital filter is not binary due to the lack of an available image encoder/decoder (CODEC) IP core. As a result, the image is first converted into integer values ranging from 0 to 255 and then converted again into single precision floating point numbers, since the digital filter operates using IEEE 754 single precision floating point standard. The same process is used to reverse the data that is received from the digital filter. This pseudo-CODEC is achieved using a mixture of Matlab and Java application on the host PC. For the same reason, the coefficients are also converted to single precision floating point numbers before they are written to the digital filter.

To write the filter coefficients to the digital filter, the Avalon MM interface is used [19]. Generally speaking, the Avalon MM interface creates a read/write interface in a memory-mapped system, where all the components are connected by interconnect fabric.

[1] For all intents and purposes, the USB Blaster is great for debugging. A new design can be quickly loaded into the FPGA and have test cases executed. For long term usage of the hardware, it is highly unlikely that a host PC will always be available to configure the FPGA every time the digital filter needs to be used. Instead, the configuration files can be stored in the onboard flash memory and have the onboard MAX II CPLD device automatically configure the FPGA on power-ups. [18]

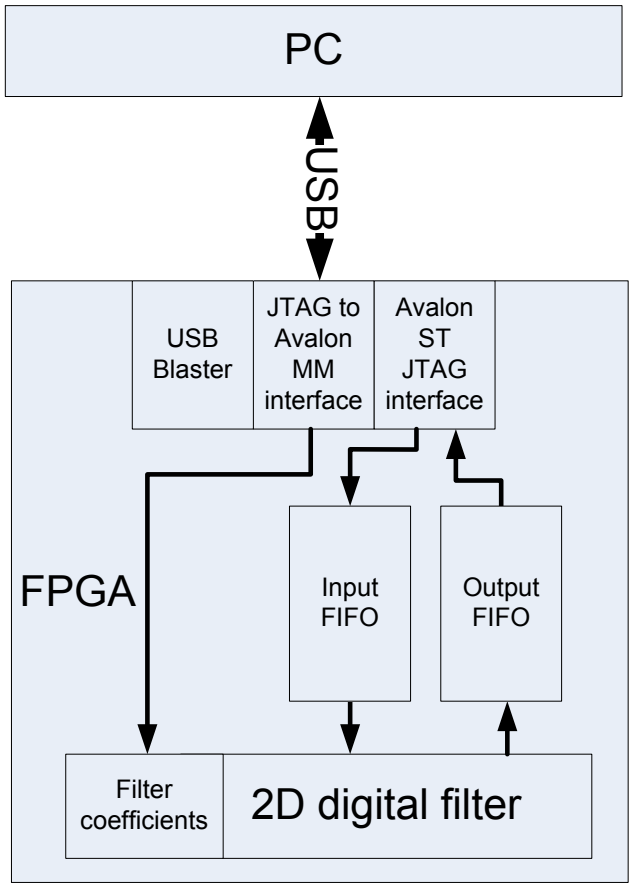


Figure 5.4: DSP System Architecture

Components such as the Nios II softcore processor, memories, communication interfaces and peripherals can all be interfaced using Avalon MM interface.

A detailed input/output (IO) interface and timing relations between JTAG to Avalon MM interface and digital filter is shown in Figure 5.5(a) and Figure 5.5(b) respectively.

1. address and read signals are asserted after the rising edge of clock by the master.
2. Address and read signals are sampled, and the slave responds with valid readdata.
3. Address, read and readdata lines are deasserted after read operation completes.
4. Address and write signal are asserted by the master. Valid data is asserted on the writedata line.
5. Write operation completes and the master deasserts address, write and writedata lines.

In total, four words are written to the slave. An actual Avalon MM interface transaction is shown in Figure 5.5(c), where coefficient $a1$ from realization one is given the hexadecimal value of bfe85332. Since coefficient a1 is the first of many coefficients in realization one, the address line is given the value of 00.

The actual image data is sent and received via the Avalon ST JTAG interface. In general, the Avalon ST interface is used for components that requires high bandwidth, low latency and unidirectional data [19], such as multiplexed streams, packets and digital signal processing data. The interface is capable of supporting complex protocols such as burst transfers and packet transfers with packets interleaved across multiple channels. For the proposed system, the Avalon ST interface is used as a traditional streaming interface where it supports a single stream of data without knowledge of channels or packet boundaries.

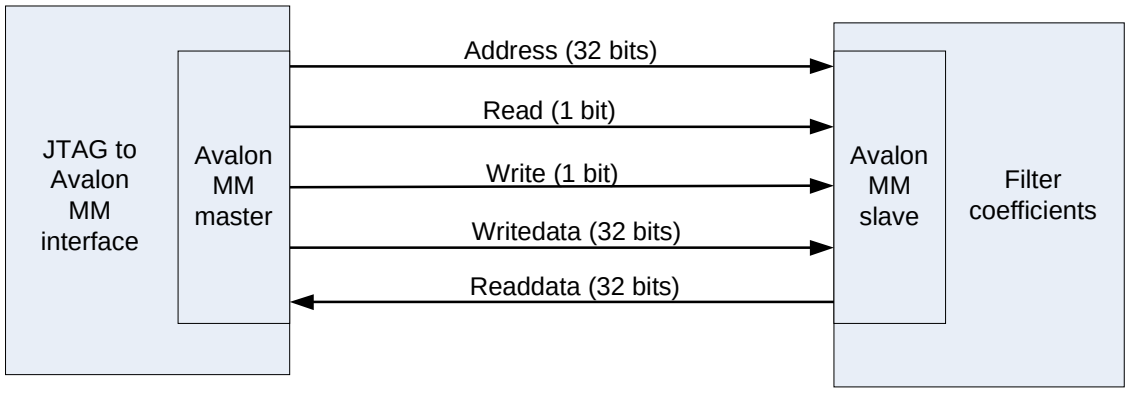
A detailed input/output interface of the Avalon ST interface from the Avalon ST JTAG interface to the digital fiter is shown in Figure 5.6(a), where the data flows from source to the sink. The Avalon ST interface uses backpressure to regulate the flow of data. Backpressure is a mechanism where the sink can signal to the source to stop sending data due to a full FIFO or congestion in its output ports. Backpressure is extensively used in this digital signal processing system to control data flow. Figure 5.6(b) illustrates a data transfer with backpressure.

1. Source provides data and asserts valid, even though the sink is not ready.
2. The source asserts ready and the first data is captured immediately two cycles after (1).

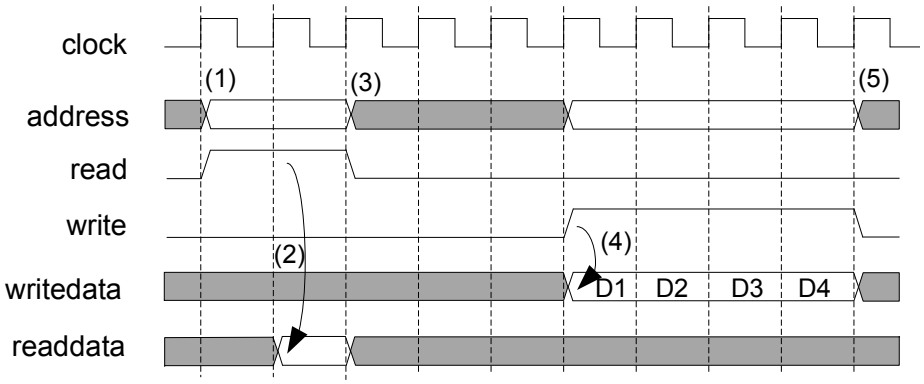
3. The next data is captured and the source deasserts valid, but the sink does not deassert ready until a cycle later.

In Figure 5.6(c), a transaction on the Avalon ST interface is shown where the input FIFO sends pixel data to the Avalon ST sink in the digital filter. The input FIFO, also called the ingress FIFO, asserts the hexadecimal value of 32353509 on the data lines. The Avalon ST sink on the digital filter captures the data on the same clock cycle. Reg_data is the input register at the input of the actual filter implementation.

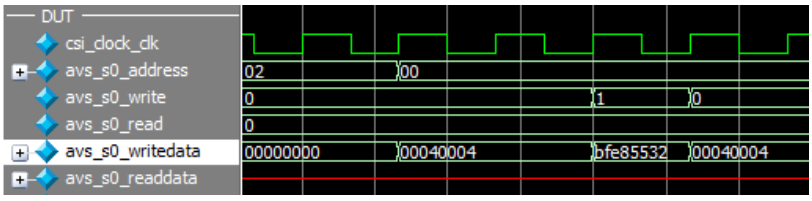
In conclusion, both the Avalon MM interface and the Avalon ST interface are used to construct the digital signal processing system. The Avalon MM interface is used to write and read filter coefficients to the 2D digital filter while the Avalon ST interface is used to transfer the raw input data to the FPGA and processed output data from the FPGA.



(a) Avalon MM interface I/O ports

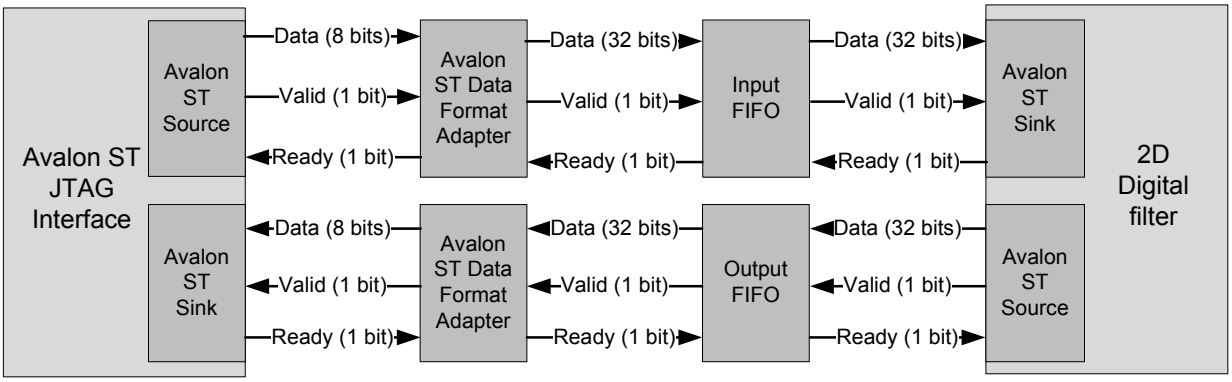


(b) Avalon MM interface timing relationship

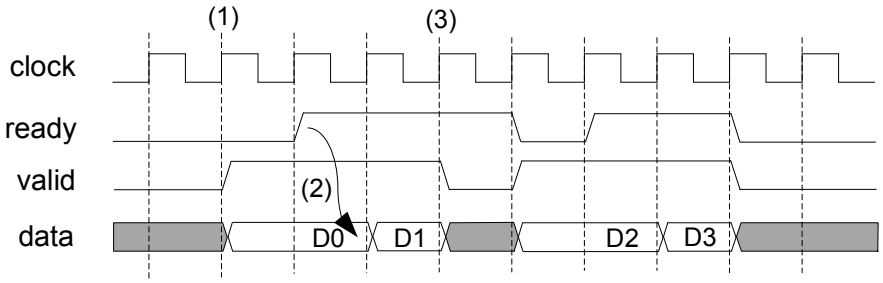


(c) A coefficient is written to the filter using Avalon MM interface

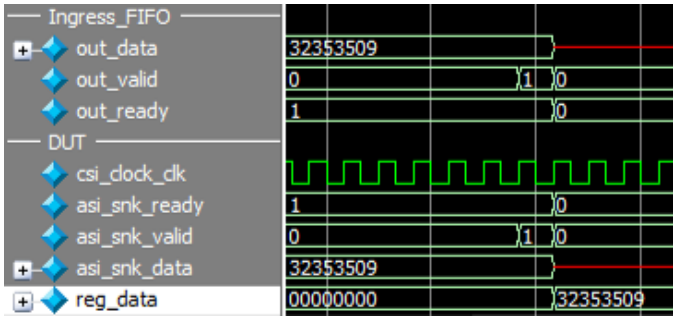
Figure 5.5: Avalon MM Interface Between the JTAG to Avalon MM Interface and the Filter Coefficient



(a) Avalon ST interface I/O ports



(b) Avalon ST interface timing relationship



(c) Pixel data is received by the Avalon ST sink in the digital filter

Figure 5.6: Avalon ST Interface Between the Avalon ST JTAG Interface and the Digital Filter

Chapter 6

Simulation Results

Given the derived coefficients in Chapter 3, the modifications made to the filter realization due to timing in Chapter 4, and the completed DSP system architecture in Chapter 5, filter realizations of various sizes can now be implemented in FPGA hardware. Once implementation is complete, the corresponding logic utilization, performance and power figures can be shown. In this chapter, images using the 3×3 filter coefficients derived from Chapter 3 is displayed first to show the filter implementations are indeed functional. Following that, performance, logic utilization, and power for other filter sizes are discussed.

6.1 Functional Correctness

As mentioned before, the input image used for the DSP system is a grayscale image of dimension 64×64. There is no particular for selecting a image dimension of 64×64 except demonstration. It should be noted that an image dimension larger than 64×64 can be used as input. The only change needed is to increase the storage capacity of the vertical shift registers (z_2), since a vertical shift register must be able to store the number of pixels equal to the width of the image. The sample image is shown in Figure 6.1. Since the impulse response is a low pass filter, the processed image looks blurred. The four processed images can be seen in Figure 6.2.

During experimentation, it was found that the filter impulse response is not normalized. This translates directly to output pixel values in excess of maximum allowable display value of 0 to 255. To ensure the output pixel values can be displayed, a Java program was written to normalize the output pixel value by linear scaling. The scaling factor is determined based

on the minimum and maximum values derived from the output pixel values. Any negative pixel values are rounded up to 0. During this scaling procedure, some details in the image are lost and can contribute to the perceived difference between the original image and the processed images.

Figure 6.1: Original Sample Image

Lines of black and gray can be seen on the top portion of the processed images, which were not present in the original image. This effect is commonly known as aliasing and is due to the initial condition of the digital filter. When the digital filter comes out of reset, all the intermediate pipeline registers are assigned a value of zero. Typically, when the digital filter starts processing the input data, it takes some time before meaningful information begins to arrive at the output of the digital filter. Inevitably some zeros will be observed at the output before the pixel values start to change. For gray scale images, a value of zero represents black and as a result, some black values are seen before the white appears.

Furthermore, the amount of aliasing is also affected by the scaling operation performed after the image is processed. Both realization three and realization four requires four times as much scaling per pixel when compared to realization one and realization two. Visually this can contribute to a processed image of higher contrast for realization three and four, and consequently, less aliasing.

The issue of aliasing is not a problem easily solved. Aliasing is more likely to happen to an image with a white background and processed right after the DSP system comes out of a reset. If the image contains a black background, aliasing will not be observed since the value of black is already zero in the pixel. Furthermore, if images are continuously processed, such as a video clip, the pixel values will likely not differ from the current image

to the previous image, and thus aliasing is less likely to occur.

It can also be seen that some of the image is warping from the right hand side to the left hand side. This effect is directly contributed by the insertion of control paths as discussed in Subsection 4.4. The control path dictates the transfer of a valid output every fixed amount of clock cycles. However, in all the realizations, the latency is larger than cycle time by several folds. Due to this difference, first few pixels output from the digital filter can be safely ignored as they do not contain any useful information. This issue can be easily fixed in software. As long as the software takes into account the latency of the given digital filter after the initial reset, the number of pixels that need to be discarded is equal to the number of cycles of latency divided by cycle time.

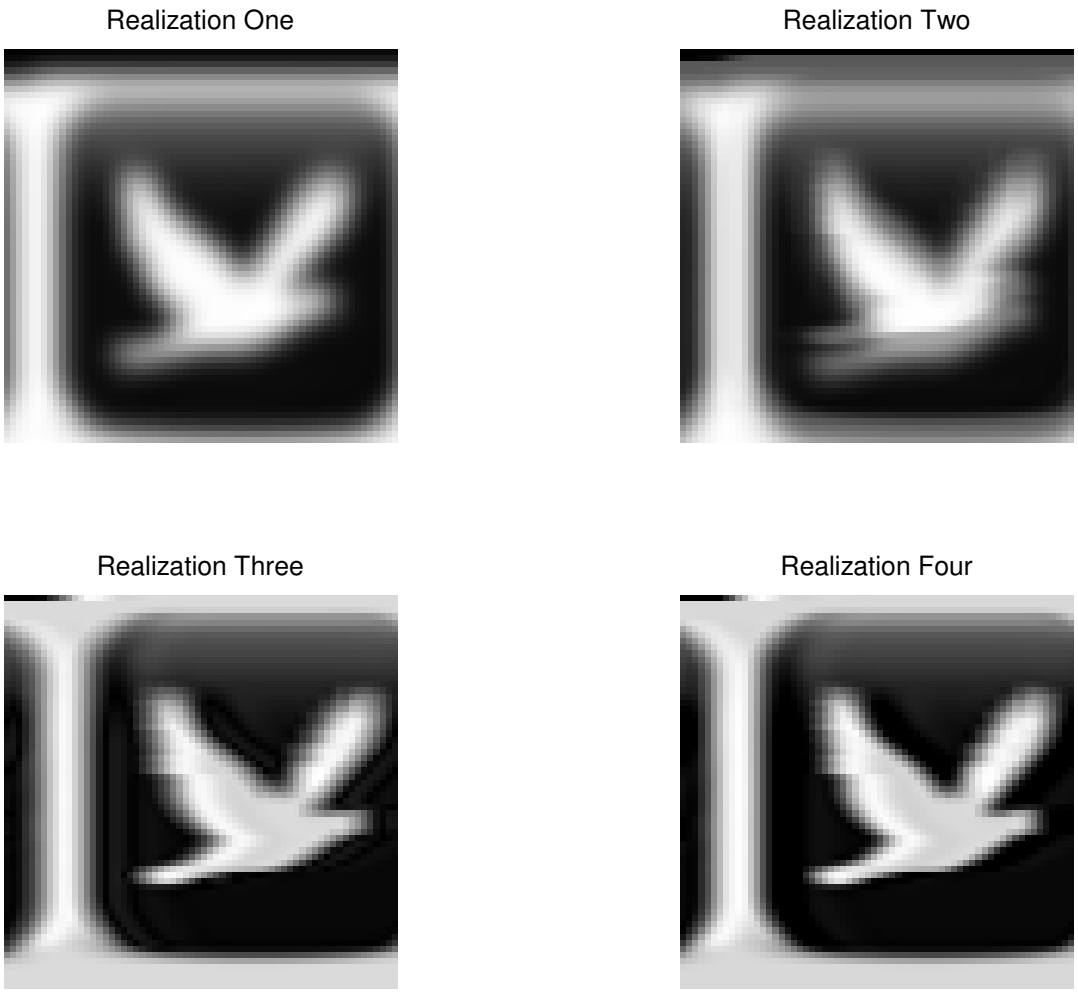


Figure 6.2: Processed Sample Image

6.2 Filters of Varying Sizes

Once the filter function correctness is confirmed, more filters of varying sizes can be constructed to observe the trend in performance, logic utilization, and power dissipation. It should be noted that the coefficients used to derive the power figures for filters with dimension larger than 3×3 are not derived from a pre-existing or pre-designed filter. As such, the power figures should only be considered as a general guideline of what power dissipation could look like. For filter dimension less than 3×3, the filter coefficients derived from Chapter 3 are truncated.

6.2.1 Performance

Opimitization of signal routing and module placement on FPGA is an intense and laborious process. To assist in the design, tools often provide several optimization settings that can significantly affect the performance, logic utilization and the power dissipation of the final design. Since the FPGA hardware is traditionally not optimized for power, the goal for this research is to optimize for maximum performance instead. A list of detailed settings for the synthesizer and fitter can be found in Appendix B. The maximum operating speed achieved using these settings is shown in Table 6.1. The maximum operating speed is limited to 375.09 MHz by the memory element used to construct the horizontal shift registers (z_1).

Table 6.1: Maximum Operating Speed (MHz) for Realization One With and Without DSE Optimization.

Filter Order	1x1	2x2	3x3	4x4	5x5	6x6	7x7
Without DSE	375.09	349.55	331.13	301.04	229.04	282.00	252.50
With DSE	375.09	375.09	375.09	375.09	375.09	345.07	329.49

With the aid of the Design Space Explorer (DSE), the performance can be increased dramatically. DSE is capable of enumerating a list of possible synthesizer/fitter configurations that can potentially give a better performance. A list of seed numbers is entered into the DSE and the optimization goal can be selected. Since this research is focused on performance enhancement, searching for the best performance option was selected. The maximum operating speed for realization one after using DSE is shown in Table 6.1. As one can see, the performance has improved dramatically and the f_{max} is close to 375.09 MHz for most filter sizes of realization one. The performance eventually deteriorates due to the increasing size of the design. It is evident that DSE can significantly increase the

performance of the design, and as a result, all realizations are DSE-optimized to ensure the maximum operating frequency is attained.

Table 6.2 shows a summary of the maximum operating speed for realization two, realization three, and realization four after DSE optimization has been applied. Realization one shows the least amount of performance deterioration as the filter order increases. Realization four performs better than realization three as filter order increases is no surprise, since realization four is realization three without instantiating the same poles multiple times. Realization two has the worst performance figure due to its hybrid parallel-cascade realization, which has a tendency of increasing fan out at the input of the digital filter and the input of the tree adder. If the intended field of application requires high performance 2D digital filter, realization one and realization four are preferred over realization two and realization three.

Table 6.2: Maximum Operating Speed (MHz) for Realization One, Two, Three and Four after Design Space Explorer.

Filter Order	1x1	2x2	3x3	4x4	5x5	6x6	7x7
One	375.09	375.09	375.09	375.09	375.09	345.07	329.49
Two	375.09	375.09	343.41	306.00	256.61	NA	NA
Three	375.09	372.30	362.32	344.47	281.53	NA	NA
Four	375.09	375.09	375.09	373.83	361.66	NA	NA

Operating speed, however, is only a part of the story. Throughput is the better indicator of performance, which is usually measured in bits per second, as it represents the maximum possible quantity of data that can be transmitted under ideal circumstances. To calculate throughput, maximum operating speed is multiplied by cycle time. The theoretical cycle time for each realization is presented in Table 4.1 and is shown again in Table 6.3, combined with the actual cycle time assuming t_{add} is 12, t_{mult} is 10, and t_{shift} is 1.

Table 6.3: Theoretical and Actual Cycle Time for Each Realization

Realization	Theoretical cycle time	Actual cycle time (clock cycles)
One	$2t_{add} + t_{mult} + 1$	$2 \times 12 + 10 + 1 = 35$
Two	$2t_{add} + t_{mult} + 1$	$2 \times 12 + 10 + 1 = 35$
Three	$t_{add} + t_{mult} + 1$	$12 + 10 + 1 = 23$
Four	$t_{add} + t_{mult} + 1$	$12 + 10 + 1 = 23$

Multiplying actual cycle time (from Table 6.3) by maximum operating frequency (from Table 6.2), maximum throughput is obtained in Table 6.4 in terms of million pixels per

second. As one can see, even though the maximum operating frequency is roughly the same across realizations of smaller filter order, the throughput is dramatically different due to cycle time. Realization four emerges as the definitive filter if performance is of major concern. Realization three shows promise as a close second to realization one. However, due to decrease in f_{max} as filter order increases, realization three is not as good as realization four.

Table 6.4: Throughput (million pixels/second) for Each Realization.

Filter Order	1x1	2x2	3x3	4x4	5x5	6x6	7x7
One	10.72	10.72	10.72	10.72	10.72	9.86	9.41
Two	10.72	10.72	9.81	8.74	7.33	NA	NA
Three	16.31	16.19	15.75	14.98	12.24	NA	NA
Four	16.31	16.31	16.31	16.25	15.72	NA	NA

6.2.2 Logic Utilization

As presented in Chapter 2, the number of processors required for each realization is derived from previous work [9]. The number of processors required for each realization is reiterated in Table 6.5, where M_1, M_2, and r are the horizontal dimension, vertical dimension and the rank of the matrix B, respectively. To determine the correlation of logic utilization between the realizations and the implementations, the number of processors used to construct each realization is translated into the number of combinational ALUTs, logic registers and 18 bit DSP multipliers, which is what the Quartus II tool uses when describing the logic utilization of a given implementation.

Table 6.5: Processors Required for Each Realization (does not include the final tree adder).

Filter	Processors required
One	$(M_1+1)(M_2+1)+M_1+M_2$
Two	$r(2M_1+1+2M_2+1)$
Three	$3M_1M_2+2(M_1+M_2)$
Four	$M_1M_2+2(M_1+M_2)$

As aforementioned, four modules from the Quartus II Megawizard Library are used to construct the filter implementations: FP adders, FP multipliers, horizontal shift registers and vertical shift registers. The corresponding logic utilization required for each module is

listed in Table 6.6. To determine the correlation of logic utilization between the realizations and the implementations, the relationship between the mathematical realization and the modules must be determined.

Table 6.6: Logic Utilization Required for Each Megawizard Library Module.

Megawizard module	ALUTs	Logic Registers	18 bit Multipliers	Memory Bits
FP Adder	126	357	0	0
FP Multiplier	535	756	4	0
z_1 Shift Register	0	32	0	0
z_1 Shift Register	24	13	0	1984

For realization one, M_1 processors are in the first subfilter, and each processor includes two adders, one multiplier and one horizontal shift register. $(M_1+1)(M_2+2)$ processors are in the second subfilter, and each processor includes a multiplier, an adder, and a horizontal shift register. M_2 processors are in the third subfilter, and each processor includes 2 adders, one multiplier, and one vertical shift register. Using these numbers, the expected hardware utilization can be calculated. Table 6.7 summarizes the expected logic utilization based on calculation versus actual logic utilization from the Quartus II tool for realization one.

Table 6.7: Expected versus Actual Logic Utilization for Realization One.

Filter Order	1x1	2x2	3x3	4x4	5x5	6x6	7x7
Expected Comb ALUT	4494	9271	15370	22791	31534	41599	52986
Actual Comb ALUT	6326	12653	21346	24614	33809	57081	70622
Expected Logic Register	8279	16632	27275	40208	55431	72844	92747
Actual Logic Register	8608	17272	28638	42028	57867	75763	96045
Expected 18 bit DSP	24	52	88	132	184	244	312
Actual 18 bit DSP	24	52	88	132	184	244	312
Expected memory bits	1984	3968	5952	7936	9920	11904	13888
Actual memory bits	1984	3968	5952	7936	9920	11904	13888

The 18 bit DSP multipliers and the memory bits used are the same between expected and actual logic utilization, however, the number of combinational ALUT and logic registers used are different. The actual implementation typically uses more logic than is required to implement the filters due to many reasons. For example, one fitter setting, called logic cell insert - logic duplication, enables the fitter to insert buffer logic cells and to duplicate logic cells to improve timing. When the setting is turned on, the fitter automatically inserts logic cells between two nodes without altering the functionality of the design. Buffer logic

cells are created from unused combinational ALUT and logic registers in the device. Yet another setting called duplicate logic for fan-out control duplicates logic or registers to improve fan-out. Duplicating logic or registers can help improve timing in paths where moving a register in a failing path to reduce routing delay creates another failing path, or where there are timing problems due to high fan-out nodes. A combination of these fitter settings helps to improve the maximum operating frequency of a design at the cost of higher logic utilization. The same analysis can be applied to other realizations/implementations, but the task would be redundant and is therefore not shown. Instead, the logic utilization for all implementations is shown.

Table 6.8: Actual Logic Utilization for Each Realization.

Filter Order		1x1	2x2	3x3	4x4	5x5	6x6	7x7
One	Comb ALUT	6326	12653	21346	24614	33809	57081	70622
	Logic Register	8608	17272	28638	42028	57867	75763	96045
	18 bit DSP	24	52	88	132	184	224	312
	Memory bits	1984	3968	5952	7936	9920	11904	13888
	Percentage (%)	5	10	17	24	33	46	58
Two	Comb ALUT	3847	27571	50060	79191	92393	NA	NA
	Logic Register	6624	35550	65584	104926	123746	NA	NA
	18 bit DSP	24	120	224	360	528	NA	NA
	Memory bits	9920	17856	31744	49600	103168	NA	NA
	Percentage (%)	4	22	40	63	78	NA	NA
Three	Comb ALUT	6317	18130	31573	51747	78403	NA	NA
	Logic Register	8651	27708	42066	68302	101306	NA	NA
	18 bit DSP	24	84	136	228	344	NA	NA
	Memory bits	3968	11904	17856	27776	49600	NA	NA
	Percentage (%)	5	15	25	41	63	NA	NA
Four	Comb ALUT	4667	10730	21052	31967	36164	NA	NA
	Logic Register	6543	14702	28100	41683	57070	NA	NA
	18 bit DSP	24	84	136	228	344	NA	NA
	Memory bits	1984	3968	5952	7936	11904	NA	NA
	Percentage (%)	4	9	17	25	34	NA	NA

Table 6.8 shows the logic utilization for all implementations of 2D digital filter for different varying sizes. The percentage (%) indicated in the table is the amount of FPGA hardware resource utilizated needed to fit the implementation. As it can be seen, all implementations have nearly the same logic utilization when the filter size is small. The

amount of growth for each implementation is different and there are several reasons that can contribute to the growth factor. A fixed amount of circuitry is required to construct the filter when filter size increases both in the horizontal and vertical direction. This entails additional FP multipliers, adders and shift registers are needed. Furthermore, as the amount of circuitry increases, the more difficult it is to meet timing. With aforementioned fitter settings, such as logic cell insert - logic duplication and duplicate logic for fan-out control, timing can be more easily met at the cost of additional logic utilization. As the filter size increases, these effect are compounded, and eventually contributes to significant increase in logic utilization.

Overall, both realization one and four are suitable candidates for area reduction. Both realizations use roughly the same amount of logic to synthesize and fit the design. If the research is to be carried out in standard cell design flow, realization one and realization four would be the ideal candidate since smaller logic utilization means less gate count and quicker turn over. Realization two and realization three, however, are terrible candidates. The logic utilization is twice as much or more for realization two and realization three, when compared to realization one and realization four. In actuality, the problem for realization two and realization three is similar. Both realizations implement the horizontal and vertical poles multiple time in the filter structure. This occupies a substantial amount of logic and leads to increase in logic utilization.

6.2.3 Power Dissipation

In the world of circuit design, trade offs are common. For FPGA design, this equates to trading power and area for flexibility. This does not always appeal to the ASIC crowd since area can translate directly to silicon cost and power can translate to heat. As a result it is unfair to compare the results obtained from FPGA hardware directly to its ASIC counterpart.

Power dissipation is linked directly to the input. Typically there is always a worst case input vector that will trigger the worst power dissipation. In other words, the power dissipation is directly related to the input vector. For example, a black image will dissipate significantly less power than the sample image, since many of the data paths do not toggle. To figure out the worst power vector, however, would take a long for a complex design. For this research, a 3×3 sample impulse response from previous work is presented, and 2D digital filter coefficients are derived from various transformation. However, that means there will not be enough coefficients for filter larger than 3×3. To cope with this problem, coefficients are duplicated in order to have the right number of coefficients. In some cases

the output is full of noise, or outright incorrect. Some may argue this produces bogus power data. But one must keep in mind that the work presented in this thesis is on the 2D digital filter implementation on FPGA, not the design of 2D digital filter.

Table 6.9: Power Dissipation for each Realization in mW.

Filter Order		1x1	2x2	3x3	4x4	5x5	6x6
One	Dynamic Power	236.79	513.65	862.10	1195.62	1720.07	2078.07
	Static Power	2.70	2.67	2.67	2.73	2.73	2.72
Two	Dynamic Power	134.73	717.84	1994.19	2671.14	3636.53	NA
	Static Power	2.74	2.71	2.70	2.71	2.75	NA
Three	Dynamic Power	261.92	793.77	1277.19	1489.56	2298.41	NA
	Static Power	2.67	2.69	2.69	2.7	2.71	NA
Four	Dynamic Power	187.8	435.11	828.69	1273.94	1729.01	NA
	Static Power	2.70	2.70	2.71	2.72	2.73	NA

The dynamic and static power dissipation for all realizations are shown in Table 6.9. The table shows realization one and realizations four are the more suitable candidate for ASIC design since they dissipate less power than realization two and realizations three. Typically power correlates very closely with logic utilization and switching frequency.

Chapter 7

Conclusions and Future Work

Four 2D separate denominator digital filters implementations are presented in this thesis. Due to timing constraint, optimization to the filter architecture is made by inserting registers in the DSP pipeline. With this change, the digital filter is able to achieve higher maximum operating speed. A flexible DSP system based on Altera System On Programmable Chip technology is presented in this thesis. It is used to functionally verify the correctness of the digital filter. Various synthesizer and fitter settings, such as logic duplication for fan-out control and logic cell insertion - logic duplication, are discussed in detail. Design Space Explorer is introduced and used to increase maximum operating frequency further.

7.1 Conclusions

Performance of the digital filter is directly related to the operating frequency and cycle time. With the help of DSE optimization, operating frequency is roughly equal for all realizations as filter size increases. However, the cycle time for realization three and four is much smaller than one and two due to less logic on the critical path. These two factors combined contributes to higher throughput for realization three and four. From the experimental result presented, realization four is able to achieve the highest throughput of 15.72 million pixels per second at 361.66 MHz for a filter size of 5×5. Realization three is a close second with 12.24 million pixels per second at 281.53 MHz. For high performance applications, realization three or realization four is recommended.

A smaller logic utilization translates to small area in ASIC. Since area directly translates to silicon cost, smaller logic utilization is better for low cost production. From the

results presented, realization one and realization four occupies 33% and 34 % of the FPGA resources for a 5 $\times$ 5 filter, respectively, which uses half as much resources as realization two and realization three. Realization one and realization four are better choice if the goal is to reduce production cost. Logic utilization also directly correlates with dynamic power dissipation. The results show that realization one and realization four dissipates the lowest amount of dynamic power at 1720.07 mW and 1729.01 mW for a filter size of 5 $\times$ 5.

7.2 Future Work

Realization four is the best candidate if the filter is to be implemented in ASIC flow. It has excellent performance, occupies lower area and dissipates lower power compared to all other realizations. For more area reduction on the ASIC implementation, floating point algorithms could be converted to fixed point algorithms to eliminate hardware complexity. Alternatively, the floating point pipeline can be customized so the denormalization and the normalization stage of the floating point operator can be eliminated altogether.

The floating point adder speed versus pipeline depth anomaly observed in Figure 5.3 should be studied further. Pipelined floating point operators should see a gradual increase in operating speed as pipeline increases. A sudden and large increase in operating speed is unexpected and requires more careful examination.

References

[1] Sen M. Kuo and Woon-Seng Gan. *Digital Signal Processing - Architecture, Implementation, and Applications*. 2004. 1, 45

[2] Qiu-Ting Wang and Ye Bin. A new recognizing and understanding method of military airfield image based on geometric structure. In *Wavelet Analysis and Pattern Recognition, 2007. ICWAPR '07. International Conference on*, volume 3, pages 1241 –1246, 2-4 2007. 1

[3] B. Lambrigtsen, A. Tanner, T. Gaier, P. Kangaslahti, and S. Brown. Prototyping a new earth observing sensor - geostar. In *Aerospace Conference, 2007 IEEE*, pages 1 –9, 3-10 2007. 1

[4] G. Linan-Cembrano, L. Carranza, C. Rind, A. Zarandy, M. Soininen, and A. Rodriguez-Vazquez. Insect-vision inspired collision warning vision processor for automobiles. *Circuits and Systems Magazine, IEEE*, 8(2):6 –24, second 2008. 1

[5] R.W.M. Smith. A generic scaleable image processing architecture for real-time military applications. In *High Performance Architectures for Real-Time Image Processing (Ref. No. 1998/197), IEE Colloquium on*, pages 3/1 –3/6, 12 1998. 1

[6] Scott Thibault. FPGA vs. DSP design reliability and maintenance. `http://www.altera.com/literature/wp/wp-01023.pdf`, 2011. Online; accessed 6-June-2011. 1

[7] K. Sakiyama, P.R. Schaumont, and I.M. Verbauwbede. Finding the best system design flow for a high-speed jpeg encoder. In *Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific*, pages 577 – 578, jan. 2003. 1

[8] Michael Parker. FPGA vs. DSP design reliability and maintenance. `http://www.altera.com/literature/wp/wp-01023.pdf`, 2011. Online; accessed 11-May-2011. 2

[9] Mohammed Yahia Dabbagh and Winser E. Alexander. Multiprocessor implementation of 2-D denominator sparable digital filters for real-time processing. *IEEE Trans. on Acoustic, Speech and Signal Processing*, 37:872 – 881, June 1989. 4, 11, 62

[10] A. N. Venetsanopoulos and B. G. Mertzios. A decomposition theorem and its implications to the design and realization of two-dimensional filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, CAS-33:239 – 249, December 1986. 7

[11] H. Andrews and III Patterson, C. Singular value decomposition (svd) image coding. *Communications, IEEE Transactions on*, 24(4):425 – 432, apr 1976. 8

[12] D. Coltuc and I. Pitas. Jordan decomposition filters. In *Circuits and Systems, 1995. ISCAS '95., 1995 IEEE International Symposium on*, volume 3, pages 1896 –1899 vol.3, 30 1995. 8

[13] C. Nikias, A. Chrysafis, and A. Venetsanopoulos. The LU decomposition theorem and its implications to the realization of two-dimensional digital filters. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 33(3):694 – 711, jun 1985. 8

[14] Tao Lin, Masayuki Kawamata, and Tatsuo Higuchi. Design of 2-d separable-denominator digital filters based on the reduced-dimensional decomposition. *IEEE Trans. on Circuits and Systems*, 34:934 – 941, August 1987. 17

[15] Takao Hinamoto, Toshiaki Takao, and Mitsuji Muneyasu. Synthesis of 2d separable-denominator digital filters with low sensitivity. *Journal of the Franklin Institute*, 329:1063 – 1080, November 1992. 18

[16] Altera Inc. Floating-point compiler: Increase performance with fewer resources. `http://www.altera.com/literature/wp/wp-01050-Floating-Point-Compiler-Increasing-Performance-With-Fewer-Resources.pdf`, 2007. Online; accessed 6-June-2011. 46

[17] Terasic Inc. Altera DE4 development and education board specifications. `http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=138&No=501&PartNo=2`, 2011. Online; accessed 11-May-2011. 49

[18] Altera Inc. Fast passive parallel configuration. `http://www.altera.com/support/devices/configuration/schemes/fast-passive-parallel/cfg-fpp.html`, 2011. Online; accessed 10-May-2011. 50

[19] Altera Inc. Avalon interface specifications. http://www.altera.com/literature/manual/mnl_avalon_spec.pdf, 2011. Online; accessed 8-May-2011. 50, 52

APPENDICES

Appendix A

Detailed filter implementation on FPGA

This section shows the complete and detailed filter realizations including the coefficients and interconnects for realization one to realization four. The numerical example included in this research assumes that each dimension, z_1 and z_2, contains three roots, one real and two complex. αs and βs are used to denote the polynomial coefficients when the two complex roots are combined in the z_1 and z_2 direction, respectively. λ and γ are used to represent the single real root in the z_1 and z_2 directions, respectively.

A.1 Realization One

Realization one contains three separate stages. First stage is composed of three subfilters. First subfilter is a 1D IIR filter in the z_1 direction. Second subfilter is a bank of 1D FIR filters. Third subfilter is composed of 1D FIR filter in the z_2 direction. The filter architecture is shown in Figure A.1.

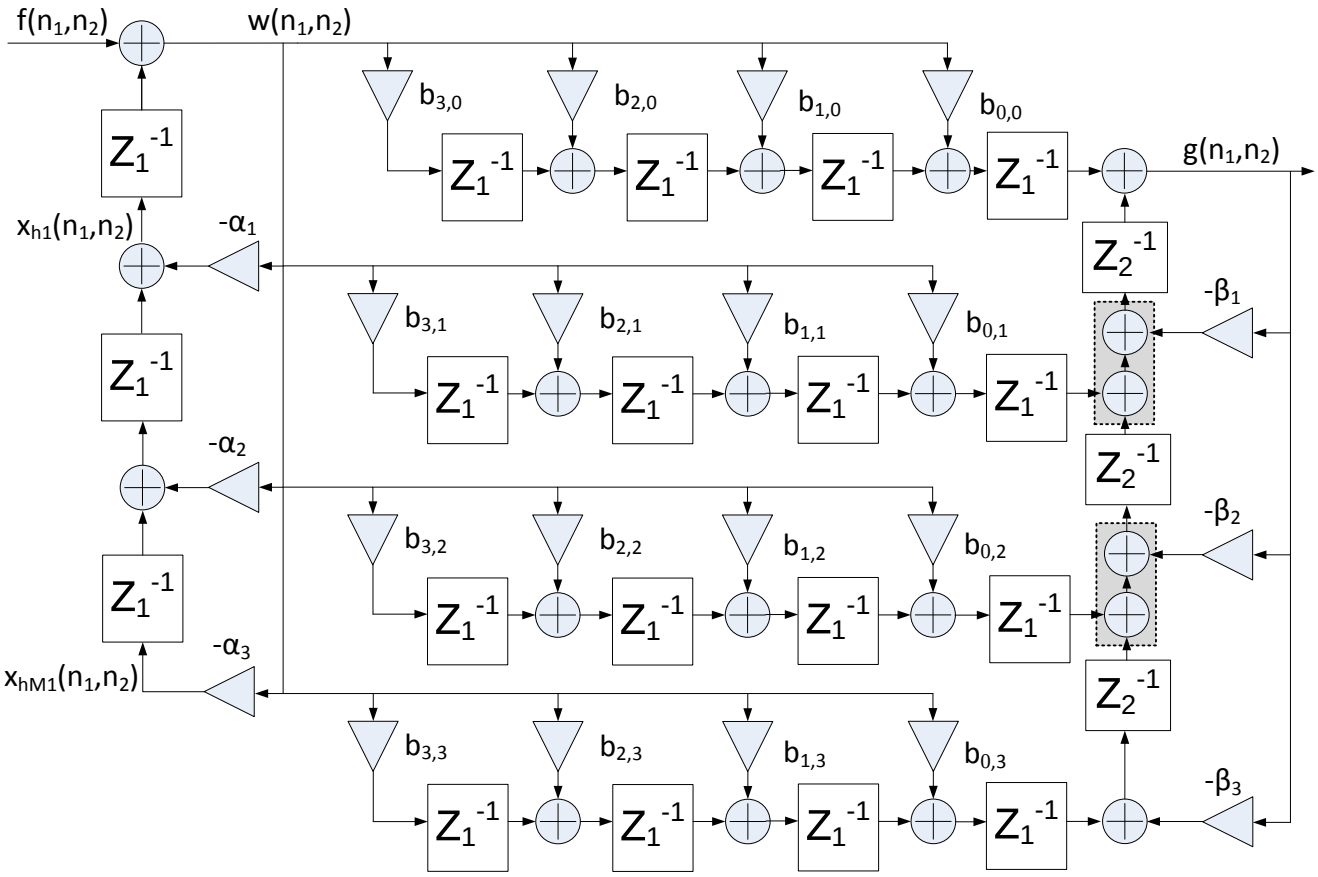


Figure A.1: Realization One

A.2 Realization Two

Realization two is a cascade of two filter banks as shown in Figure A.2. The number of banks in each subfilter is equivalent to the number of the rank of matrix B.

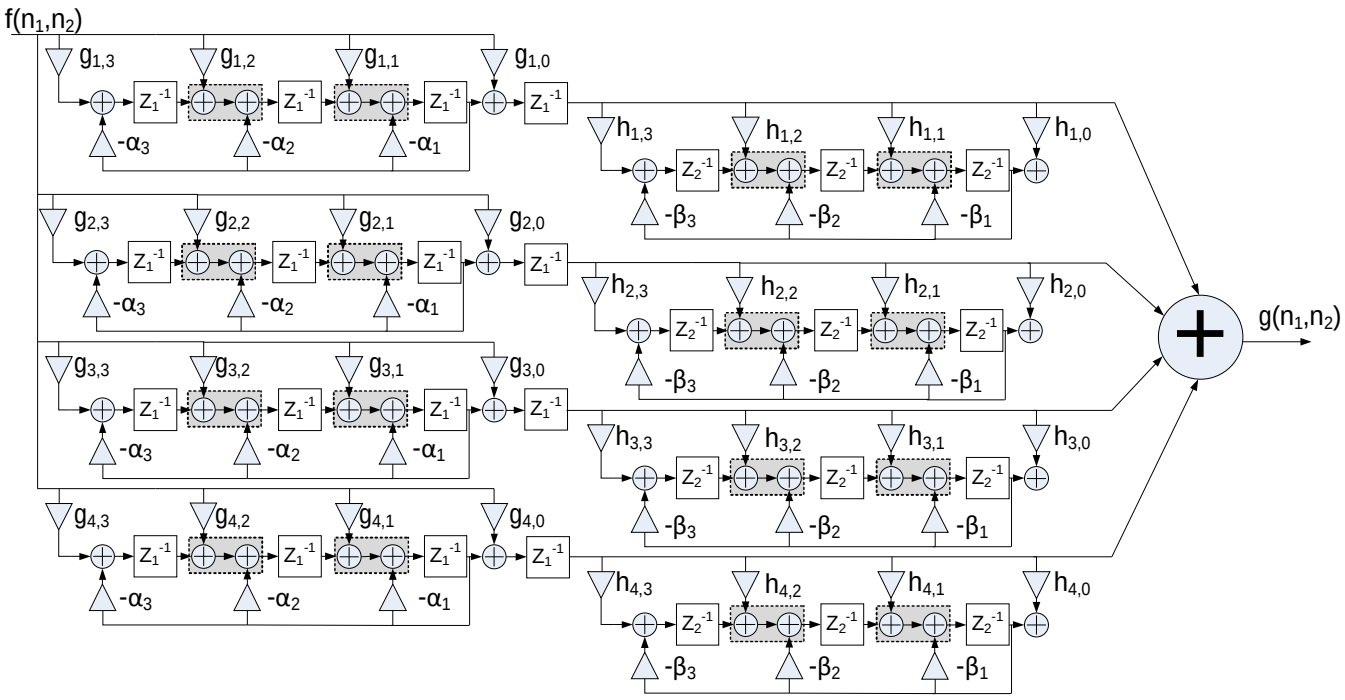


Figure A.2: Realization Two

A.3 Realization Three

Realization three is a combination of many parallel stages as shown in Figure A.3 to A.11 followed by a tree adder that sums up the intermediate results.

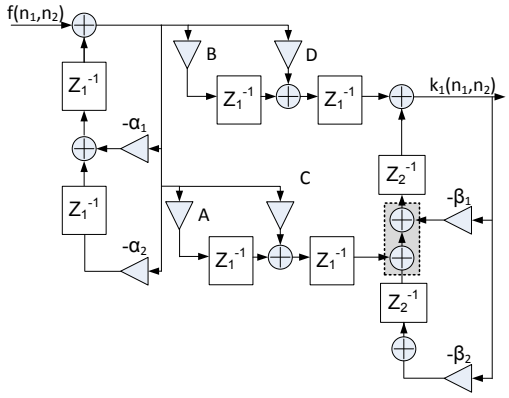


Figure A.3: Realization Three Subfilter One

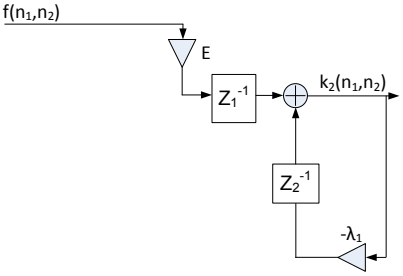


Figure A.4: Realization Three Subfilter Two

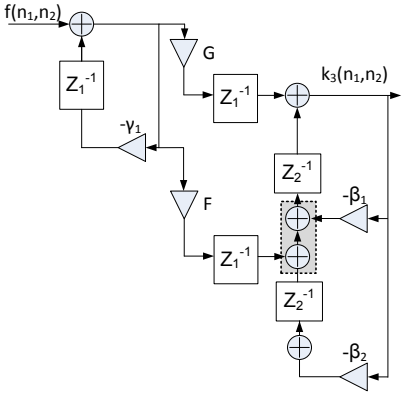


Figure A.5: Realization Three Subfilter Three

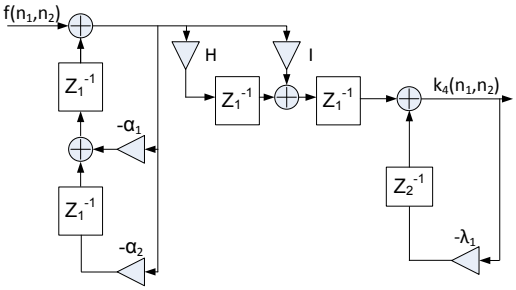


Figure A.6: Realization Three Subfilter Four

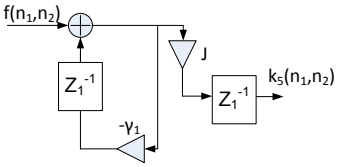


Figure A.7: Realization Three Subfilter Five

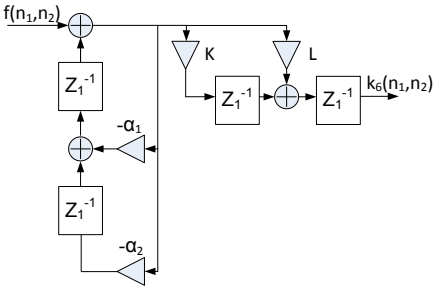


Figure A.8: Realization Three Subfilter Six

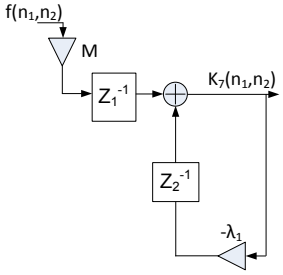


Figure A.9: Realization Three Subfilter Seven

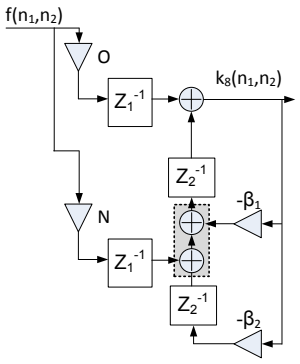


Figure A.10: Realization Three Subfilter Eight

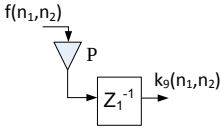


Figure A.11: Realization Three Subfilter Nine

A.4 Realization Four

Since realization four is a modified version of realization three, many subfilters that were used in realization three are reused in realization four. The interconnect between the subfilters are rewired to reduce redundant realization of the same pole. Figure A.12 shows the subfilter used to realize poles in the z_1 direction and Figure A.13 shows the subfilter used to realize the poles in the z_2 direction. Figure A.14 to A.17 shows the subfilter used to realize numerator coefficients A to P.

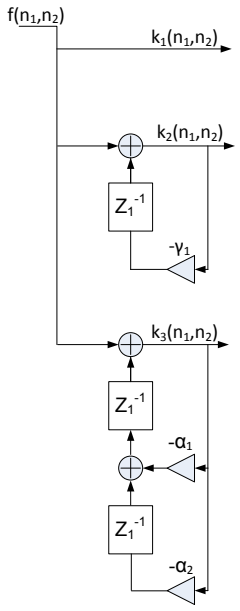


Figure A.12: Realization Four z_1 Roots

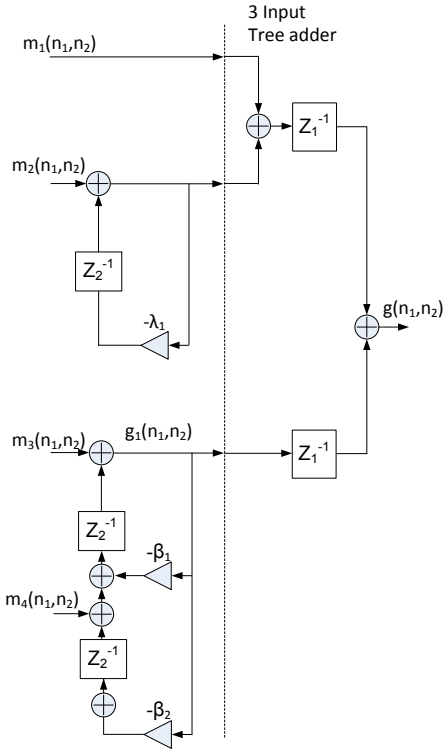


Figure A.13: Realization Four z_2 Roots

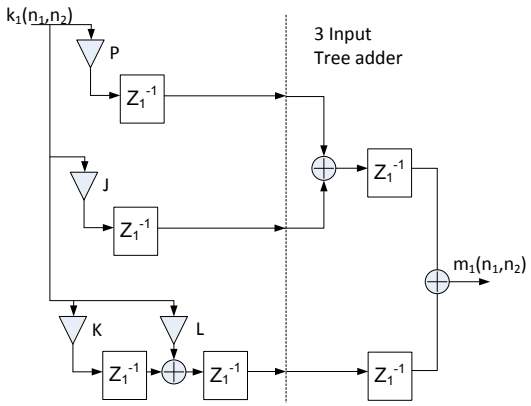


Figure A.14: Realization Four Numerator Coefficients P, J, K, and L

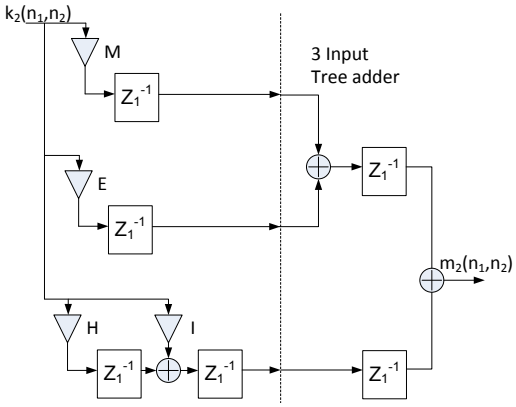


Figure A.15: Realization Four Numerator Coefficients M, E, H and I

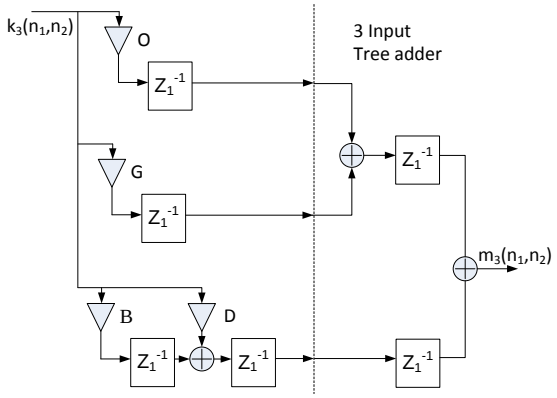


Figure A.16: Realization Four Numerator Coefficients B, D, G and O

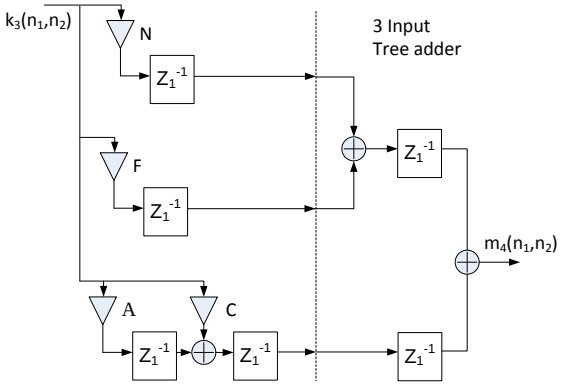


Figure A.17: Realization Four Numerator Coefficients A, C, F, and N

Appendix B

Synthesizer and Fitter Settings for Performance Optimization

B.1 Synthesizer Settings

Optimization Technique: Speed
Timing-Driven Synthesis: Checked
Power-Up Don't Care: Checked
PowerPlay power optimization: Off
Perform physical synthesis for combinational logic for performance: On
Perform register Duplication for Performance: On
Perform register retiminig for Performance: On
Logic cell insertion - logic duplication: On
Perform WYSIWYG (What You See Is What You Get) primitive resynthesis: On
Restructure multiplexers: Off
State machine processing: Auto

B.2 Fitter Settings

Seed number: 1
Optimize hold timing: All paths
Optimize multi-corner timing: Checked

PowerPlay power optimization: Off
Fitter Effort: Standard Fit (highest effort)
Router timing optimization level: Maximum
Auto packed registers: Normal
Enable beneficial skew optimization: On
Optimize hold timing: All paths
Duplicate logic for fan out control: Not set