Advanced Techniques for Improving Radar Performance

by

Mohammed Adel Shoukry
B.Sc., Military Technical College, Egypt
M.Sc., Military Technical College, Egypt

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

Advanced Techniques for Improving Radar Performance

by

Mohammed Adel Shoukry
B.Sc., Military Technical College, Egypt
M.Sc., Military Technical College, Egypt

Supervisory Committee

Dr. Panajotis Agathoklis, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Fayez Gebali, Co-Supervisor
(Department of Electrical and Computer Engineering)

Dr. Sudhakar Ganti, Outside Member
(Department of Computer Science)

Supervisory Committee

Dr. Panajotis Agathoklis, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Fayez Gebali, Co-Supervisor
(Department of Electrical and Computer Engineering)

Dr. Sudhakar Ganti, Outside Member
(Department of Computer Science)

ABSTRACT

Wideband beamforming have been widely used in modern radar systems. One of the powerful wideband beamforming techniques that is capable of achieving a high selectivity over a wide bandwidth is the nested array (NA) beamformer. Such a beamformer consists of nested antenna arrays, 2-D spatio-temporal filters, and multirate filterbanks. Speed of operation is bounded by the speed of the hardware implementation.

This dissertation presents the use of a systematic methodology for design space exploration of the NA beamformer basic building blocks. The efficient systolic array design in terms of the highest possible clock speed of each block was selected for hardware implementation. The proposed systolic array designs and the conventional designs were implemented in FPGA hardware to verify their functionality and compare their performance. The implementations results confirm that the proposed systolic array implementations are faster and requires less hardware resources than the published designs. The overall beamformer FPGA implementation is constructed based on the analysis of efficient systolic arrays designs of the beamformer building blocks. The implemented overall structure is then validated to ensure its proper operation. Further, the implementation performance is evaluated in terms of accuracy

and error analysis in comparison to the MATLAB simulations. The new methodology is based on the systematic methodology to close the gap between the modern wideband radar I/O rates and the silicon operating speed. This new metodology is applied to the interpolator block as an example. The proposed methodology is simulated and tested using MATLAB object oriented programming (OOP) to ensure the proper operation.

Contents

Supervisory Committee **ii**

Abstract **iii**

Table of Contents **v**

List of Tables **x**

List of Figures **xii**

List of Acronyms **xvii**

Acknowledgements **xx**

1 Introduction **1**

1.1 Problem Description 3

1.2 Organization of the Dissertation 3

2 Background **6**

2.1 Continuous ST wideband PWs Modeling 7

2.2 Sampled PWs in Space and Time 9

2.3 Uniform vs Nested Antenna Arrays 10

2.4 Wideband Beamforming 12

2.4.1 Fullband Beamforming using Uniform Linear Array and Trapezoidal Filters 13

2.4.2 Subband Beamformer Using Nested Arrays and Trapezoidal Filters 14

2.5 2-D Trapezoidal Filter design 16

2.6 Simulation Results 16

2.7 Conclusions 20

3 Design Space Exploration of Decimators **21**
3.1 Introduction and related work 21
3.2 Systematic methodology for systolic array design applied to the decimator 24
3.3 Decimator dependence graph (DG) 25
3.4 Decimator scheduling function 26
3.5 Decimator node projection 29
3.6 Systolic array design space exploration 30
3.6.1 Design-Option #1: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_1 = [1 \quad 0]^T$ . . . 31
3.6.2 Design-Option #2: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_3 = [1 \quad -1]^T$ . . 34
3.6.3 Design-Option #3: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$ . . . 35
3.6.4 Design-Option #4: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_1 = [1 \quad 0]^T$ . . . 36
3.6.5 Design-Option #5: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_2 = [1 \quad 1]^T$ . . . 37
3.6.6 Design-Option #6: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$ . . . 39
3.6.7 Comparing the proposed designs 39
3.7 Alternative decimator structures 40
3.7.1 Conventional Design 40
3.7.2 Polyphase Design 42
3.8 FPGA implementation results 43
3.9 Conclusion 45

4 Design Space Exploration of 2-D Beamforming Filter **50**
4.1 Introduction and Related Work 50
4.2 Preliminaries 52
4.2.1 Algorithm Computational Domain 52
4.2.2 Index Dependence of the Algorithm Variables 52
4.2.3 Task Scheduling 53
4.2.4 Point Projection 53
4.3 Computational Domain (CD) of the 2-D BB BF Filter 54
4.4 Dependence Matrices of the 2-D BB BF Filter Variables 55
4.4.1 Nullvectors of the Dependence Matrices 55
4.4.2 Feeding and Extraction Points of the 2-D BB BF Filter Variables 56
4.5 Scheduling Functions for the 2-D BB BF Filter 59
4.5.1 Scheduling Option #1 60
4.5.2 Scheduling Option #2 61
4.6 Point Projection for the 2-D BB BF Filter 62

4.7 Comparing Multiplier Based and Distributed Arithmetic Based multiplier/accumulator . . . 64
4.8 Exploration of Processor array structures . . . 66
4.8.1 Design #1: $\boldsymbol{s}_1 = [1 \quad 1 \quad I]$ and $\boldsymbol{P}_{ab} = [0 \quad 0 \quad 1]$ . . . 67
4.8.2 Design #2: $\boldsymbol{s}_1 = [1 \quad 1 \quad I]$ and $\boldsymbol{P}_{ac} = [0 \quad 1 \quad 0]$ . . . 70
4.8.3 Design #3: $\boldsymbol{s}_1 = [1 \quad 1 \quad I]$ and $\boldsymbol{P}_{bc} = [1 \quad 0 \quad 0]$ . . . 70
4.8.4 Design #4: $\boldsymbol{s}_2 = [1 \quad J \quad 1]$ and $\boldsymbol{P}_{ab} = [0 \quad 0 \quad 1]$ . . . 72
4.8.5 Design #5: $\boldsymbol{s}_2 = [1 \quad J \quad 1]$ and $\boldsymbol{P}_{ac} = [0 \quad 1 \quad 0]$ . . . 72
4.8.6 Design #6: $\boldsymbol{s}_2 = [1 \quad J \quad 1]$ and $\boldsymbol{P}_{bc} = [1 \quad 0 \quad 0]$ . . . 72
4.9 Hardware Implementation Results . . . 72
4.9.1 Conventional Design Implementation . . . 73
4.9.2 Implementation Results . . . 73
4.10 Performance Comparison . . . 74
4.11 Conclusion . . . 75

5 Design Space Exploration of the Interpolators 82
5.1 Introduction and Related Work . . . 82
5.2 Systematic Methodology for Systolic Array Design Applied to the Interpolators . . . 83
5.3 Interpolator Dependence Graph (DG) . . . 84
5.4 Interpolator Scheduling Function . . . 85
5.5 Interpolator Node Projection . . . 89
5.6 Systolic Array Design Space Exploration . . . 90
5.6.1 Design-Option #1: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_1 = [1 \quad 0]^T$ . . . 91
5.6.2 Design-Option #2: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_3 = [1 \quad -1]^T$ . . . 95
5.6.3 Design-Option #3: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$ . . . 96
5.7 Hardware Implementation Results . . . 97
5.7.1 Conventional Design Implementation . . . 98
5.7.2 Proposed Interpolator Design Implementation . . . 98
5.7.3 Implementation Results . . . 98
5.8 Design Complexity Comparison . . . 101
5.9 Conclusions . . . 102

6 Efficient FPGA Implementation of the Beamformer 104
6.1 Efficient FPGA Implementation of the Beamformer . . . 104

6.1.1 FPGA Implementation Validation 105
6.2 Finite-Precision Model Accuracy 114
6.3 Error sources 117
6.3.1 Finite Word-Length Effect on the Signal-to-Error Ratio (SER) 118
6.3.2 The Effect of Using a Conventional MACs on the system SER 120
6.4 Maximum Speed and Resources Utilization Estimation 121
6.5 Conclusions 122

7 Closing the Speed Performance Gap Between Radar I/O Rates and Silicon Speed 123
7.1 The Proposed Methodology for Closing the speed performance gap . . 124
7.2 Partitioning 125
7.3 Re-timer 128
7.3.1 Interconnection Network 130
7.4 Carry-Save Addition of the Outputs from the Interconnection Network 133
7.5 Final Addition 135
7.6 Ultra-High Speed Interpolator General Block Diagram 136
7.7 System Clocking 136
7.8 Conclusions 137

8 Conclusions and Future Work 138
8.1 Conclusions 138
8.2 Summary and Significance of Dissertation Contributions 138
8.2.1 Processor Array Design Space Exploration for High-Speed Decimators 139
8.2.2 Processor Array Design Space Exploration for High-Speed 2-D BF Filters 139
8.2.3 Processor Array Design Space Exploration for High-Speed Interpolators 140
8.2.4 Finite word-length effect on the beamformer accuracy 140
8.2.5 Closing the speed gap between high-speed ADC and silicon speed 141
8.3 Future Work 141
8.3.1 Design Space Exploration of the 3-D Beamforming Filter . . . 141
8.3.2 Application of Closing Speed Gap Methodology to the Decimator and 2-D BF Filter Blocks 142

8.3.3 Implementation on SDR 142

Bibliography **143**

List of Tables

Table 3.1 The possible systolic array design-options 31
Table 3.2 Input/output timing for PE_0 and PE_1 in Fig. 3.10. 38
Table 3.3 Performance comparison between conventional, polyphase and proposed decimators at different decimation factors $M = 2, 4,$ and 8 for the case of number of FIR filter coefficients $J = 8$. . . 44
Table 3.4 Performance comparison between conventional, polyphase and proposed decimators at different decimation factors $M = 2, 4, 8,$ and 16 for the case of number of FIR filter coefficients $J = 16$. 45
Table 3.5 Performance comparison between conventional, polyphase and proposed decimators at different decimation factors $M = 2, 4, 8, 16,$ and 32 for the case of number of FIR filter coefficients $J = 32$. 46
Table 3.6 Performance comparison between conventional, polyphase and proposed decimators at different decimation factors $M = 2, 4, 8, 16, 32,$ and 64 for the case of number of FIR filter coefficients $J = 64$. 47

Table 4.1 Relation between the sensor index i and the sensor sample delay D for Design #1. 69
Table 4.2 Comparison between resource utilization for conventional and proposed designs for the case of $I = 15$, $J = 32$ and PE word size = 8 bits. 74

Table 5.1 The possible systolic array design-options 90
Table 5.2 PE_0 and PE_1 activities for Design-Option #1 after applying the non-linear scheduling function in Eq. (5.25) when $J = 8$ and $L = 4$. 93
Table 5.3 Comparison between resource utilization for conventional and proposed designs for different values of $L = 2$, 4, and 8 for the case of $J = 8$. 99

Table 5.4 Comparison between resource utilization for conventional and proposed interpolators for different values of $L = 2$, 4, 8, and 16 for the case of $J = 16$. 100
Table 5.5 Comparison between resource utilization for conventional and proposed designs for different values of $L = 2$, 4, 8, 16, and 32 for the case of $J = 32$. 101
Table 5.6 Comparison between resource utilization for conventional and proposed designs for different values of $L = 2$, 4, 8, 16, 32, and 64 for the case of $J = 64$. 101

Table 6.1 The scaling factors for the system shown in Fig. 6.4. 110
Table 6.2 The cumulative scaling factors and the number of bits for Fig. 6.4 points. 115

Table 7.1 Example of the partial results that required by the output sample y_{10}. 128

List of Figures

Figure 2.1 The plane wave propagating from a certain direction. 8
Figure 2.2 The ROS of the wideband PWs. 9
Figure 2.3 Nested array vs ULA structures. 11
Figure 2.4 The passband area of TF encloses the ROS of the desired PW. 13
Figure 2.5 The structure of the NA beamformer. 14
Figure 2.6 ROS of $\boldsymbol{F}(f_z, f_{ct})$, and the passband area of the 2-D TF. 15
Figure 2.7 Bandwidth of the received signals. 17
Figure 2.8 Analysis FIR filters amplitude responses. 17
Figure 2.9 ROS of 2-D FT of: (a)-(d) the signals received by the l^{th} sub-array, (e)(h) the analysis filters output, and (i)-(l) the output of downsamplers by 2^{l-1}. 18
Figure 2.10 ROS of 2-D trapezoidal filter. 19
Figure 2.11 Beamformer output and desired signal in the time-domain. . . . 19

Figure 3.1 M-to-1 decimator dependence graph for the case when $M = 4$ and $J = 8$. Empty circles denote multiply/accumulate operations. 25
Figure 3.2 The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_1$ in Eq. (3.9). 27
Figure 3.3 The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_2$ in Eq. (3.10). 28
Figure 3.4 The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_3$ in Eq. (3.11). 28
Figure 3.5 The DAG for Design-Option #1: using $\boldsymbol{s}_1 = [1 \ -1]$ and $\boldsymbol{d}_1 = [1 \quad 0]^T$. 32
Figure 3.6 Systolic array for Design-Option #1. (a) The systolic array when $J = 8$ and $M = 4$. (b) PE details. 33
Figure 3.7 The DAG for Design-Option #2: using $\boldsymbol{s}_1 = [1 \ -1]$ and $\boldsymbol{d}_3 = [1 \ -1]^T$. 34

Figure 3.8 The DAG for Design-Option #3: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$. 35
Figure 3.9 The DAG for Design-Option #4: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_1 = [1 \quad 0]^T$. 36
Figure 3.10Systolic array for Design-Option #4. (a) The systolic array when $J = 8$ and $M = 4$. (b) PE details. 37
Figure 3.11The DAG for Design-Option #5: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_2 = [1 \quad 1]^T$. 39
Figure 3.12The DAG for Design-Option #6: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$. 40
Figure 3.13The conventional decimator implementation. (a) The downsampler details. (b) The systolic array of the conventional FIR filter. (c) The FIR filter PE details (Form 2). 41
Figure 3.14The double-precision polyphase decimator implementation. (a) The polyphase decimator structure. (b) The systolic array for the FIR filter section. 42
Figure 3.15Maximum frequency comparison chart between different filter structures at different decimation factors for $J = 64$. 48
Figure 3.16Power consumption comparison chart between different filter structures at different decimation factors for $J = 64$. 49

Figure 4.1 Subdomain $\mathcal{D}_h$ for the input variable instance $h(c_1, c_2)$. 56
Figure 4.2 Pipelining in the i-direction first then in the j-direction. 61
Figure 4.3 Pipelining in the j-direction first then in the i-direction. 62
Figure 4.4 The normalized delay associated with the conventional, DA based, and proposed MAC implementation at different word-sizes for Xilinx Virtex-7 FPGAs. 66
Figure 4.5 Systolic array for Design #1. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details. 71
Figure 4.6 Systolic array for Design #2. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details. 76
Figure 4.7 Systolic array for Design #3. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details. 77
Figure 4.8 Systolic array for Design #4. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details. 78

Figure 4.9 Systolic array for Design #5. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details. 79
Figure 4.10 Systolic array for Design #6. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details. 80
Figure 4.11 The conventional 2-D BB BF filter implementation. (a) The systolic array of the conventional 2-D BB BF filter. (b) The systolic array of the 1-D FIR filter. (c) The 1-D FIR filter PE details. 81

Figure 5.1 1-to-L interpolator dependence graph for the case when $L = 4$ and $J = 8$. Empty circles denote multiply/accumulate operations. 85
Figure 5.2 The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_1$ in Eq. (5.9). 87
Figure 5.3 The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_2$ in Eq. (5.10). 88
Figure 5.4 The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_3$ in Eq. (5.11). 88
Figure 5.5 The DAG for Design-Option #1: using $\boldsymbol{s}_1 = [1 \ -1]$ and $\boldsymbol{d}_1 = [1 \ 0]^T$. 92
Figure 5.6 Systolic array for Design-Option #1. (a) The systolic array when $J = 8$ and $L = 4$. (b) PE details. 94
Figure 5.7 The DAG for Design-Option #2: using $\boldsymbol{s}_1 = [1 \ -1]$ and $\boldsymbol{d}_3 = [1 \ -1]^T$. 95
Figure 5.8 The DAG for Design-Option #3: using $\boldsymbol{s}_1 = [1 \ -1]$ and $\boldsymbol{d}_4 = [0 \ 1]^T$. 96
Figure 5.9 The conventional interpolator implementation. (a) The upsampler details. (b) The systolic array of the conventional FIR filter. (c) The conventional FIR filter PE details. 97

Figure 6.1 Single beamformer channel diagram. 105
Figure 6.2 ROS of the 2-D FT of the signals received by the 3^{rd} subarray. 107
Figure 6.3 2-D passband of the trapezoidal filter. 107
Figure 6.4 Finite-precision fixed point implementation diagram. 109
Figure 6.5 Simulations for the 3^{rd} channel FPGA implementations. (a) Decimator array outputs. (b) 2-D BF filter outputs. (c) Interpolator (channel) outputs. 111

Figure 6.6 ST representation of the finite-precision decimator array outputs. 112
Figure 6.7 ST frequency domain representation of the finite-precision decimator array outputs. 113
Figure 6.8 Finite-precision 2-D BB BF filter center censor output. 114
Figure 6.9 Finite-precision interpolator output. 114
Figure 6.10ST representation of the full-precision decimator array outputs. 116
Figure 6.11ST frequency domain representation of the full-precision decimator array outputs. 116
Figure 6.12Full-precision 2-D BB BF filter output. 117
Figure 6.13Full-precision interpolator output. 118
Figure 6.14Finite word-length and generated errors effects on SER using proposed MAC. (a) Channel #1. (b) Channel #2. (c) Channel #3. (d) Channel #4. 119
Figure 6.15Finite word-length and generated errors effects on SER using conventional MAC. (a) Channel #1. (b) Channel #2. (c) Channel #3. (d) Channel #4. 121

Figure 7.1 Interpolator DAG in case of $J = 16$ and $L = 4$. 124
Figure 7.2 Interpolator partitioning #1 for the DAG in Fig. 7.1. 126
Figure 7.3 Interpolator partitioning #2 for the DAG in Fig. 7.1. 127
Figure 7.4 Interpolator partitioning #3 for the DAG in Fig. 7.1. 128
Figure 7.5 Partitioning block diagram. (a) Distribution of the interpolator inputs among the partitions. (b) PE details. 129
Figure 7.6 The outputs of the partitions in case of $L = 4$ and $P = 3$. . . . 130
Figure 7.7 The timing for the addition of the partitions outputs in case of $L = 4$ and $P = 3$. 131
Figure 7.8 The timing for the outputs of the re-timer and the desired final output samples in case of $L = 4$ and $P = 3$. 132
Figure 7.9 Re-timer block diagram. 133
Figure 7.10Interconnection network block diagram. 134
Figure 7.11Partitions outputs carry-save addition block diagram. (a) Adders array. (b) Adder detals. 134
Figure 7.12Final addition block diagram. 135
Figure 7.13General block diagram of the ultra-high speed interpolator. . . 136
Figure 7.14Interpolator basic structure. 136

Figure 7.15System clocking required for ultra high speed interpolator. . . . 137

List of Acronyms

ADC	Analog to Digital Converter
ALU	Arithmetic Logic Unit
2-D	Two-Dimensional
BB	Broadband
BF	Beamforming
CD	Computational Domain
CF	Computational Filter
CFAR	Constant False Alarm Rate
CIC	Cascaded Integrator Comb
CPU	Central Processing Unit
CSA	Carry-Save Adder
CS-MAC	Carry-Save Multiplier/Accumulator
DA	Distributed Arithmetic
DAG	Directed Acyclic Graph
DBF	Digital Beamforming
DCA	Digital to Analog Converter
DDC	Digital Down Converter
DFT	Discrete Fourier Transform
DG	Dependence Graph
DoA	Direction of Arrival
DoF	Degree of Freedom
DSP	Digital Signal Processor
D-2	Decimation-by-2
D-4	Decimation-by-4
D-8	Decimation-by-8
EMW	Electromagnetic Wave
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FT	Fourier Transform
GPU	Graphic Processing Unit
GSC	Generalized Sidelobe Canceller
GS/s	Gigasamples per second
GUI	Graphical User Interface

IFT	Inverse Fourier Transform
IIR	Infinite Impulse Response
I-2	Interpolation-by-2
I-4	Interpolation-by-4
I-8	Interpolation-by-8
I/O	Input/Output
LFM	Linear Frequency Modulation
LSB	Least Significant Bit
LUT	Lookup Table
MAC	Multiply/Accumulate
MD	Multi-Dimensional
MDSP	Multi-Dimensional Signal Processing
MIMO	Multiple-Input Multiple-Output
MRFIR	Multi-rate FIR Filter
MSB	Most Significant Bit
MVDR	Minimum Variance Distortionless Response
NA	Nested Arrays
OOP	Object Oriented Programming
PE	Processing Element
PR	Perfect Reconstruction
PW	Plane Wave
QMF	Quadrature Mirror Filter
RCA	Ripple Carry Adder
RF	Radio Frequency
RIA	Regular Iterative Algorithm
ROM	Random Access Memory
ROS	Region of Support
SBP	Spatially Bandpass
SDR	Software Defined Radio
SER	Signal-to-Error ratio
SNR	Signal-to-Noise ratio
SOI	Signal of Interest
SPCP	Single-Precision Carry-Propagate
ST	Spatio-Temporal
STAP	Space Time Adaptive Processing

TF	Trapezoidal Filter
ULA	Uniform Linear Array

ACKNOWLEDGEMENTS

Praise be to Allah who gave me health, strength and patience to conduct this work. I would like to thank everyone who supported me during my Ph.D. journey, and I would like to express my sincere gratitude and appreciation to the following special people:

Prof. Panajotis Agathoklis, for all the support, mentorship, encouragement, insight, and all of his interesting inputs on a variety of topics he gave me. I have learned from him the way to think, the way to approach the problems, and the way to be a good supervisor. Without his guidance and constant feedback this PhD would not have been achievable.

Prof. Fayez Gebali, for all his patience, invaluable advise, and guidance throughout the study period. I sincerely admire his professional ethics because of his ability to connect with his students on a professional and personal level simultaneously. His office door has been always open for me long and late hours, and even in the weekends. I believe my research style has been strongly influenced by the way that he approaches the problems. Rather than jumping to the solution, he had a remarkable ability to break down a complicated problem into simpler steps which follow a logical line so that I could understand not only how to deal with the problem, but why it was solved that way.

Prof. Sudhakar Ganti, for his participation as member of the respected committee.

My beloved father soul, Adel Abdelhamid, who was the origin of my success.

My mother, Abla Tantawy, for her great support, love, continuous prayers for me to excel in my Ph.D. she always believing in me and encouraging me to follow my dreams.

My wife, Shimaa Ibrahim, for her continuous love, patience, emotional support, assurance in difficult and frustrating moments, helping in whatever way she could, and encouragement during during this challenging period. She remembered me all the time in her prayers and ensured that I had the proper environment to excel in my studies.

My daughter, Shahd Shoukry, for her family and social times she spent with me, especially during the weekends.

My brother, Ahmed Adel, for his continuous prayers and support.

The Programmer Analyst, Kevin Jones, for his great help and kind assistance with all my technical problems either in software or hardware. He was always available for us and ready to help.

Mohammed Shoukry

Chapter 1

Introduction

Radio detection and ranging (radar) performs two major tasks, detecting the presence of a target and determining its range. The round trip of a radar signal includes transmitting an electromagnetic wave (EMW) to cover an area of interest, scattering of the wave by target(s) inside this area, receiving the scattered EMW at the receiver side, and finally processing the received signal to extract the desired information [1].

Radar applications and modes are diverse. For example, radars are used on aircraft, missiles, satellites, ships, ground vehicles, and tripods. They attempt to detect, locate, characterize, and possibly track aircraft, missiles, ships, satellites, personnel, metallic objects, moving ground vehicles, buried objects even mold growing within building walls. With such a wide variety of radar platforms and targets, the process of classifying specific radars and their goals is a hard task [2].

Recently, radar tasks are not limited to just detecting and measuring the range of targets; they are expanding to include target speed, height, shape, size, and trajectory. In addition, they are used in missile guidance, tracking, and surveillance. All these tasks require fine range resolution and high measurement accuracy. Given that range resolution is inversely proportional to waveform bandwidth [1], modern radar systems transmit across a much wider frequency, typically wideband linear frequency modulated (LFM) waveform, than the classical radar systems. For example, resolutions on the order of a half foot or less are common, requiring a 1 GHz or greater bandwidth [2]. In addition to achieving fine range resolution, these wideband signals makes the radars very difficult to be detected because its spectrum is buried in the white Gaussian noise. Modern advanced radar systems need to be run-time adaptable to suit their environmental and operational requirements, which is driving the need toward digital radar systems. The need for more digital signal processing is pushing

the conversion of analog radar signals into digital as early as possible by moving the analog-to-digital converter (ADC) closer to the antenna. This in turn introduces a number of challenging system-level considerations [3]. Recently, gigasample per second (GSPS) ADCs have been introduced with resolution up to 12-bits [4]. These high speed ADCs help in pushing the digitization point in the radar systems toward the antenna. Supporting digitization close to the antenna means that the digital signal processing platform, typically an FPGA, can be used right after the antenna. However, connecting the FPGA platform to the high speed ADC brings major technical challenges. One of these challenges is that the implemented signal processing algorithms on the FPGA should be optimized for speed to be able to handle the high data rate output from the ADC. Another major challenge is that in some applications the speed of the optimized design is still lesser than the high data rate output from the ADC. We label this challenge the *speed performance gap* between high speed ADC and silicon speed.

Digital beamforming (DBF) is the signal processing technique that is used at both the transmitting and receiving ends of the radar system (right before the radar transmitting antenna and right after radar receiving antenna). Beamforming techniques are used for directional signal transmission or reception in order to achieve spatial selectivity. DBF is much preferred in modern radar over the conventional analog beamforming methods because of the latter's limitations. A noticable advance in radar functionality is being realized in the next generation of phased array antennas by replacing analog combiner networks in the antenna with DBF. The basic idea of DBF is to transmit/receive multiple independent weighted beams formed by an array of antenna elements. The received multiple signals by each receive antenna element are then down converted before the ADC stage. Following the ADC, many DBF algorithms can be used. The benefits of DBF come at the expense of needing more receivers and higher computational throughput to perform operations digitally that were formerly done with analog hardware. Fortunately, as digital computing technology has continued to advance at a rapid rate, these more demanding computing requirements have become increasingly easier to accommodate [2]. The primary motivation for implementing DBF in modern radars is the ability to process multiple spatial channels in the digital computer for advanced signal processing algorithms. However, the narrowband beamforming techniques is insufficient to perform beamforming in modern radars that employ wideband signals [5].

Wideband operation introduces an additional set of challenges for the wideband

radar designers such as, how to develop beamforming techniques capable of preserving the selectivity over wide bandwidth and how the implementation of such techniques can match the high data rates of the wideband radars.

1.1 Problem Description

Wideband beamforming operates on spatio-temporal data, where the number of samples at each time step equal the number of antenna elements. The data rate is typically from 500 MHZ to 6 GHz. Performing wideband beamforming is a challenging task as it requires achieving a high selectivity over a wide bandwidth and accommodating with the highly throughput input/output data rates. Beamforming can be implemented using GPUs, DSPs, and multicore CPUs alone or in combination, as well as with FPGAs.

In this dissertation a high data rate FPGA implementations of wideband nested array (NA) beamformers are performed. Based on a systematic methodology, massively parallel systolic-arrays architectures are proposed for the implementation of the NA beamformer. The systematic methodology helps in exploring the systolic arrays design space for the considered beamformer based on the dependence graph (DG) of the beamformer basic building blocks difference equations. The obtained architectures provide us the flexibility to choose the high-speed and low-hardware complexity systolic-array architectures that meets hardware constraints for specific values of system parameters compared to the conventional counterparts.

1.2 Organization of the Dissertation

This dissertation consists of eight chapters and is organized as follows: A wideband beamformer using NAs, 2-D filter, and multirate filter banks is presented in Chapter 2. The processor array design space exploration and efficient hardware implementation for the basic building blocks of the NA beamformer (decimators, 2-D FIR ST beamforming filters, and interpolators) is explained in details in Chapter 3, Chapter 4, and Chapter 5, respectively. In Chapter 6, the overall beamformer FPGA implementation is constructed and the effect of the finite word-length on the beamformer accuracy is experimentally evaluated. Although, this performs well for medium-to-low frequencies, for high frequencies there may be a speed gap between the I/O rates of the ADC and hardware implementations even with the fastest possible designs. In

Chapter 7, a new methodology is proposed as a solution to close the gap between the high speed ADC and hardware implemented designs silicon speed. Chapter 8, provides concluding remarks and suggestions for the future work.

In Chapter 2, the general theory of a NA beamformer and its capabilities of achieving beamforming with high selectivity over the wide bandwidth are introduced. In this chapter, first the basic idea of the characteristics of the wideband plane waves (PWs) in the 2-D spatio-temporal (ST) frequency domain is explained. Then the structure and properties of the NAs are presented and compared with the uniform linear arrays (ULAs). The NAs used here consist of several ULAs of increasing distance between sensors (each one called subarray) where the distance between elements in each subarray is two times larger than in the previous one. Combining nested arrays, 2-D filters, and multirate filter banks is comprehensively explained for a linear array with MATLAB simulations. The beamformer consists of subband beamformers, each one consists of three basic building blocks (decimator, 2-D filter, and interpolator). Each subaband beamformer uses the signals obtained from one of the subarrays as the input. These signals are filtered and downsampled so that the ROS of the resulting 2-D signals in the 2-D frequency domain are the same for all subbands. The same 2-D trapezoidal filter design can therefore be used for all subarray beamformers to pass the desired signal and eliminate interferences. The use of nested arrays leads to larger effective aperture at low temporal frequencies and thus, better selectivity for low frequencies. Further, NAs are known to require a lower sensor density for alias free sampling than ULAs.

In Chapter 3, a systematic methodology presented in [6] was applied to the beamformer's first basic building block (decimator) difference equation. This methodology is used to develop a single dependence graph that reflects the actions of the anti-aliasing filter and the downsampler. Different scheduling and projection functions were used to perform the design space exploration. Three scheduling functions and 4 projection direction were possible which produces 12 valid designs. Six designs was chosen which satisfy the fastest possible system clock speed. One of the 6 designs was chosen since it required the least area. The results of this chapter are published as [7]

In Chapter 4, the same systematic methodology presented in [6] was adopted to the beamformer second basic building block (2-D BF filter) difference equation. This methodology is used to develop a single 3D computational domain that reflects the actions of the 2-D BF filter. Different scheduling and projection functions were used

to perform the design space exploration. Three scheduling functions and 4 projection directions were possible which produces 12 valid designs. Six design options were chosen which satisfy the fastest possible system clock speed. One of the 6 designs was chosen since it displayed the least area. The results of this chapter are submitted to [8].

Chapter 5, the systematic methodology presented in [6] was applied to the beamformer third basic building block (interpolator) difference equations. This methodology is used to develop a single dependence graph that reflects the actions of the upsampler and the anti-imaging filter. Different scheduling and projection functions were used to perform the design space exploration. Three scheduling functions and 4 projection directions were possible which produces 12 valid design options. Three design options were chosen which satisfy the fastest possible system clock speed. One of the 6 designs was chosen since it displayed the least area. The results of this chapter are published as [9]

In Chapter 6, the results of a finite word-length MATLAB implementation is evaluated and compared with an FPGA implementation. The effect of finite word-length errors on the accuracy of the complete beamformer is studied. The accuracy analysis is performed by calculating the signal-to-error ratio (SER) at the output of the beamformer for different word-lengths. The SER results show that a good accuracy of the implemented system is obtained with a word-length of 12-bits and that the quality of accuracy increases significantly with increased word-lengths. The results of this chapter is published as [10]

In Chapter 7, a possible speed gap between the I/O rate and the silicon processor rate is identified for high bandwidth beamformers. A new methodology is proposed for closing this speed gap and is applied to the interpolator block. The approach starts by partitioning the DAG. The number of partitions required depends on the desired dilation. The number of clock phases required is based on the number of partitions. Simulation results indicate that this approach leads to satisfactory results.

Chapter 2

Background

Radar systems generally operate by connecting an antenna to a powerful radio transmitter to radiate EMWs. The transmitter is then disconnected and the antenna is connected to a sensitive receiver which amplifies any echos returned from reflecting objects (targets). By processing the echos, radar receiver can extract the required information about the targets. The environment in which a radar must operate includes many sources of electromagnetic radiation, which can mask the relatively weak echoes from its own transmission. Beamforming improves the radar performance in a specific spatial region – in both azimuth and elevation – while nulling out interference, noise, and extraneous signals, including those from jammers, in other regions.

Moden radar systems transmits signals of large bandwidth for performing better range/spatial resolution, lower probability of intercept, detectable material penetration, and easier target information recovery than using the narrowband signals. A signal that has a ratio of bandwidth to its center frequency (fractional bandwidth) larger than 1% has to be considered as a wideband signal since the frequency-dependence of the array manifolds and the beampattern should be considered in this case. The large bandwidth, however, results in potentially huge data rates. For various modern radars, such as air defense, air traffic control, astronomy, Doppler navigation, terrain avoidance, and weather mapping, the design of wideband beamformers would have different considerations and specifications in order to meet the requirements of specified signals. When the signal bandwidth increases, the performance of the conventional narrowband beamforming techniques will degrade significantly [5]. So, some of the major challenges faced by wideband radar designers are how to develop beamforming techniques capable of preserving high selectivity over the wide bandwidth and how to the implementation of such techniques capable of compatibility with high data rates

of the wideband radars.

This work proposes using NA beamformer for wideband radar applications. Such a beamformar is capable of achieving a high selectivety over a wide bandwidth [11]. In addition, this work introduces the high performance implementation of the NA beamformer to accommodate the wideband radar high data rates.

2.1 Continuous ST wideband PWs Modeling

wideband radio frequency (RF) signals from far-field sources or wideband signals reflected from far field objects etc., are often arrive at the antenna array over a significant angular range. Such type of signals can be modeled as wideband spatio-temporal (ST) PWs arriving at the antenna array from a certain direction of arrival (DoA). An ideal 4D continuous domain wideband ST PW can be modeled, as illustrated in Fig. 2.1, as $pw_{C-4D}(d_x x + d_y y + d_z z + ct)$, where $\boldsymbol{d} = (d_x, d_y, d_z)$ is the unit vector defining the DOA in 3-D space, c is the speed of light and $pw_{C-4D}(l)/\forall l = (d_x x + d_y y + d_z z + ct)$ is the 1-D intensity function propagating along the DOA, and $t \in \boldsymbol{R}^1$ is time [12]. The polarization of the signal is not considered here, and the following analysis is for non-polarized waves.

Figure 2.1 shouws the DOA vector in polar coordinates is given by:

$$[d_x \quad d_y \quad d_z] = [\sin\theta\cos\phi \quad \sin\theta\sin\phi \quad \cos\theta] \tag{2.1}$$

where are (θ,ϕ) the elevation and azimuth angles respectively.

When pw_{C-4D} is received by a linear 1-D sensor array located at $y = z = 0$, the received signal can be represented in the 2-D ST domain $(x, t) \in \boldsymbol{R}^2$ as:

$$pw_{C-2D}(x, ct) = pw_{C-4D}(x, y, z, ct)|_{y=z=0} \tag{2.2}$$

The corresponding continuous domain 2-D ST frequency representation of Eq. (2.2) can be obtained by the 2-D Fourier transform (FT) as:

$$PW_{C-2D}(\omega_x, \omega_{ct}) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} pw_{C-2D}(x, ct) e^{j(\omega_x x + \omega_{ct} ct)} dx\ dct \tag{2.3}$$

For a wideband ST PW with arbitrary DOA vector of, $d_x = [\sin\theta \ \cos\phi]$, with an intensity function of $pw_{C-2D}(x, ct)$, Eq. (2.3) can be rewritten as,

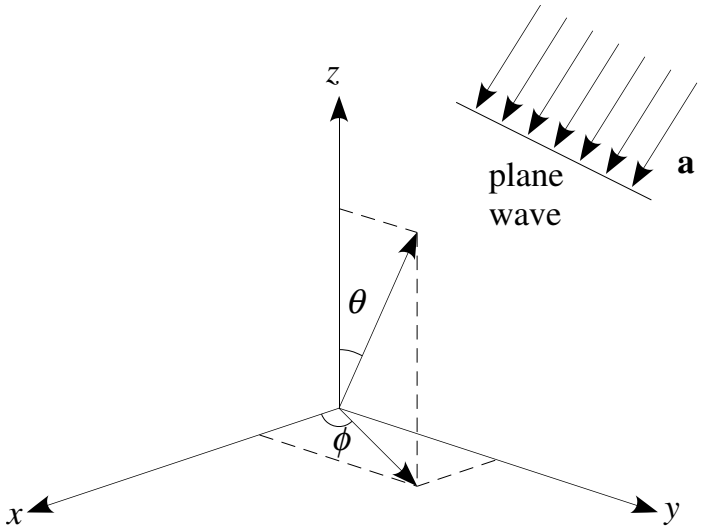


Figure 2.1: The plane wave propagating from a certain direction.

$$
\begin{aligned}
PW_{C-2D}(\omega_x, \omega_{ct}) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} pw_{C-2D}(d_x x, ct) e^{j(\omega_x x + \omega_{ct} ct)} dx\ dct \\
&= \delta(\omega_x - dx\omega_{ct}) PW(\omega_{ct}) \qquad (2.4)
\end{aligned}
$$

where $(\omega_x, \omega_{ct}) \in \boldsymbol{R}^2$, $\omega_{ct} = \omega_t/c$, and $\omega_t \in \boldsymbol{R}^2$ is the continuous domain temporal frequency of the wideband ST intensity function $pw(ct)$ where its FT is $PW(\omega_{ct})$. By examining $PW_{C-2D}(\omega_x, \omega_{ct})$, it can be observed that the region of support (ROS) of the spectrum of $PW(\omega_x, \omega_{ct})$ in Eq. (2.4) lies on the line $\omega_x - \sin\theta\omega_{ct} = 0$ which passes through the origin and makes an angel equal to $\alpha = tan^{-1}(\sin(\theta))$ with w_{ct} axis. Figure. 2.2 shows the ROSs of the spectra of two PWs (desired-unwanted) arriving from two different directions. Since $-90° \leq \theta \leq 90°$, α can be from $-45°$ to $45°$.

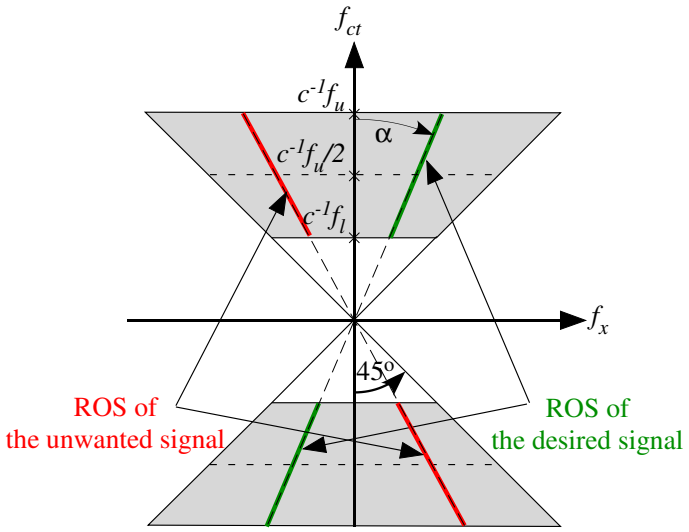


Figure 2.2: The ROS of the wideband PWs.

Up to this point, the 1-D sensor array is assumed to be of infinite extent.

2.2 Sampled PWs in Space and Time

Assume the continuous PWs with frequency response shown in Fig. 2.2 are sampled in space using a ULA of an infinite extent instead of the continous aperture. The spatially sampled signal is further temporally sampled at the rate of $f_s = 1/T_s$. This

spatially-temporally sampled signal represented by $f_D(n_t T_s + c^{-1}\cos(\theta) n_x d)$ where f_D is a discrete version of the continuous $f(t + c_{-1}\cos(\theta)x)$, and its 2-D FT consists of periodically repeated copies of $\boldsymbol{F}(f_x, f_{ct})$ which is given by:

$$\boldsymbol{F}_D(e^{j\omega_x}, e^{j\omega_t}) = \sum_{m_t=-\infty}^{\infty} \sum_{m_x=-\infty}^{\infty} \frac{\boldsymbol{F}(\frac{\omega_x - 2\pi m_x}{2\pi d}, \frac{\omega_t - 2\pi m_t}{2\pi c T_s})}{d(cT_s)} \tag{2.5}$$

where $\omega_x = 2\pi d f_x$ and $\omega_x = 2\pi T_s f_t$ are the normalized frequencies. Inside the Nyquist box, i.e. $|\omega_x| \leq \pi$ and $|\omega_t| \leq \pi$ $\boldsymbol{F}_D(e^{j\omega_x}, e^{j\omega_t})$ is equal to the 2-D continuous FT of the PW (scaled by $1/dcT_s$) provided no aliasing has happened. In order to avoid aliasing, the distance d between antenna elements must be less than $\lambda_h/2$ ($\lambda_h = c/f_h$) and $T_s \leq \alpha/2f_{max}$ ($0 \leq \alpha < 1$) [13].

2.3 Uniform vs Nested Antenna Arrays

Antenna arrays generally perform spatial sampling of the incoming PWs. wideband BF is one of the major applications of the antenna arrays. To avoid aliasing, the distance between antennas (d) must be less than or equal to $\lambda_h/2$ where λ_h is the wavelength associated with the highest frequency [13]. In addition, the aperture size depends on the ratio of the highest to the lowest frequency [13]. In order to achieve wideband BF for the signals with a large bandwidth, a large aperture with a large number of antennas should be employed. In this work, the class of nonuniform antenna arrays structure which is capable of achieving the same aperture size and significantly reduce the number of antennas are considered. This class of arrays is called "nested arrays" (NAs) as they are obtained by combining two or more ULAs with different apertures and increasing inter-sensor spacing. The inter-sensor spacing arranged so that some antenna arrays are superimposed as shown in Fig. 2.3.

NAs have been widely used in different radar applications as: For the clutter suppression in airborne radar, a fully adaptive space time adaptive processing (STAP) with nested arrays was proposed, where deep nulls along clutter ridge and a narrow mainlobe in the desired direction were achieved [14]. However, the spatial and Doppler frequencies of all jamming and clutter sources are required to be prior known. The minimum variance distortionless response (MVDR) beamformer with nested arrays was proposed in [15], where a spatial smoothing method was used to construct a covariance matrix with a larger dimension than the physical one. A robust beamforming

method in nested array based on interference-plus-noise covariance matrix reconstruction and steering vector estimation was proposed in [16]. A new nested MIMO array design approach utilizing the nested arrays, which features on having a closed-form expression for the sensor locations and the number of achievable degrees of freedom (DoFs) was proposed in [17]. The solution for the problem of radar detection in a 3D target parameter space using a digital video broadcasting-terrestrial-based passive radar systems, with a nonuniform linear array in the surveillance channel and spatial filtering in the frequency domain proposed in [18].

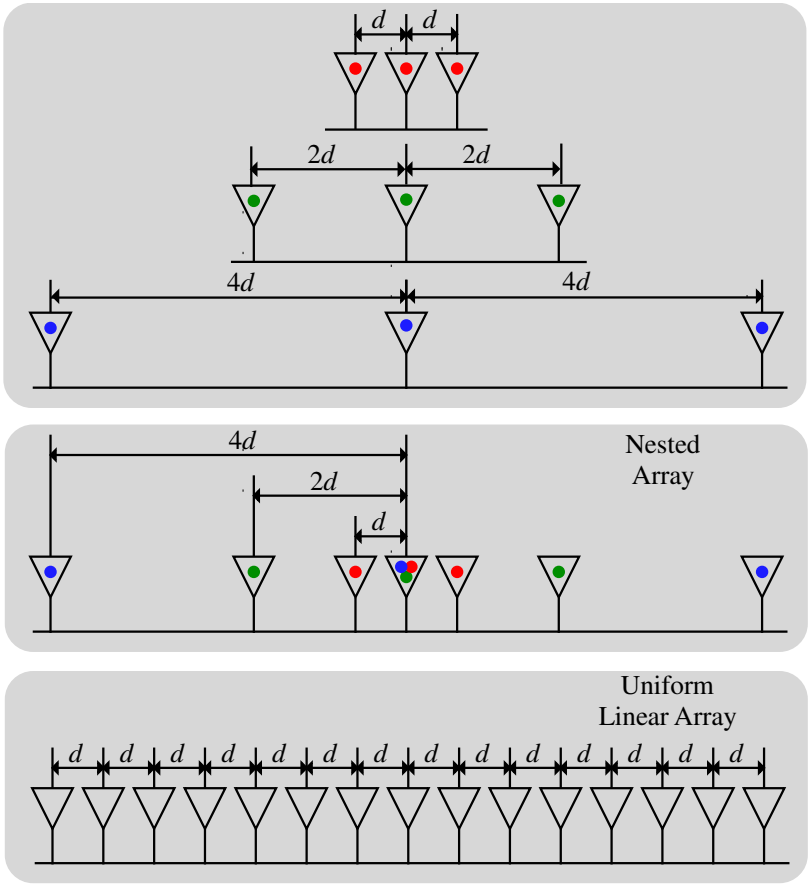


Figure 2.3: Nested array vs ULA structures.

The main advantage of NAs is that it can achieve longer aperture compared to ULAs with the same number of antennas. On the other hand, NAs can be implemented with much less antennas compared to ULAs with the same aperture size. Furthermore, NAs increase the effective aperture and consequently the selectivity at low frequencies and due to larger interelement spacing, the mutual coupling between antenna elements can be eliminated.

The beamformer used in this work consists of subarray beamforming, each one uses the output signals from one of the nested arrays as the input. These signals are filtered and downsampled in a certain way so that the ROS of the resulting 2-D signals in the 2-D frequency domain are the same for all subbands. Therefore, the same TF design can be used for all subarray beamformers to pass the desired signal and eliminate unwanted ones.

2.4 Wideband Beamforming

Beamforming means passing a PW propagating from a desired direction and reject the others. Generally, beamformer is a spatial-temporal filter which is designed to pass energy from a special direction at some desired frequencies [19]. Taking advantage of antenna array, the received signal can be spatially sampled and processed. Then, a delay line connected to each sensor can be used to perform temporal processing.

From signal bandwidth point of view, beamforming could be classified into narrow band and wideband beamformers.

For narrowband signals, no temporal filtering is involved and beamformer can be interpreted as a spatial filter [13]. In this case, beamforming can be achieved by an instantaneous linear combination of the received array signals. Delay-and-sum is one of the simplest approaches for narrowband beamforming [5].

For wideband signals, an additional temporal processing for the effective operation has to be employed [5]. For this case, beamformers can be more classified into fullband and subband [13]. In the fullband beamforming, the whole frequency spectrum of the received signal by the antenna is processed by one beamformer. The requirement of high selectivity fullband beamformer can be simply performed by increasing the number of elements. However, this will increase the costs of the system due to the increase of the number of RF modules, analog-to-digital converters (ADC), etc. On the contrary, subband beamforming is referred to an approach in which the full spectrum is decomposed into several subbands, and then each subband is processed separately. The use of NAs in conjunction with the subband beamforming can still retaining the same specification of the high selectivity as the fullband beamforming with reduced number of antenna elements and the corresponding RF modules, ADC, etc.

2.4.1 Fullband Beamforming using Uniform Linear Array and Trapezoidal Filters

As explained in Section 2.1, the ROS of 2-D FT of the PW received by a ULA is located on a line (Fig. 2.6). To do beamforming, one can use a 2-D FIR TF [12]. Ideally, the TF can be designed to have a unity gain within its passband area and zero gain elsewhere. The passband area should encloses the ROS of the desired PW as shown in Fig. 2.4 where the passband area of the TF represented by a yellow shaded area. Four parameters are used to define the passband area of the TF. The

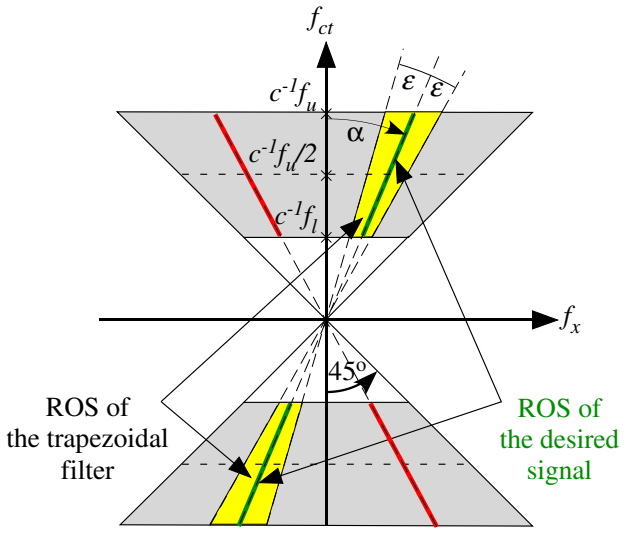


Figure 2.4: The passband area of TF encloses the ROS of the desired PW.

first parameter is the angle α and can be obtained from the DOA θ according to the following relation:

$$\alpha = tan^{-1}(\sin\theta) \tag{2.6}$$

The second parameter is the selectivity angle ϵ which controls the selectivity of the passband area around the desired PW. The third and fourth parameters, $c^{-1}f_u$ and $c^{-1}f_l$ control the upper and lower bounds along the temporal frequency axis f_{ct} where c represents the speed of light, f_u and f_l represent the upper and lower frequencies of desired PW, respectively.

2.4.2 Subband Beamformer Using Nested Arrays and Trapezoidal Filters

Consider a Wideband PW having temporal bandwidth $[f_l,f_u]$ $(f_l > 0)$ is propagating with a given DOA. Without loss of generality it can be assumed that $f_u/f_l = 2^L$, where L is a positive integer. The objective is to recover $f(t)$, the temporal intensity function of the Wideband PW received from the desired DOA, without distortion and reject interference signals with different DOAs and noise.

The beamformer deploys a NA of L ULAs [11, 20]. These nested ULAs have the effect of subsampling the incoming PW in space. Each ULA is part of one of the L different subband beamformers. The received signal at each array element is temporally sampled by the rate of $F_s = 2f_u/\alpha$ where $0 < \alpha \leq 1$.

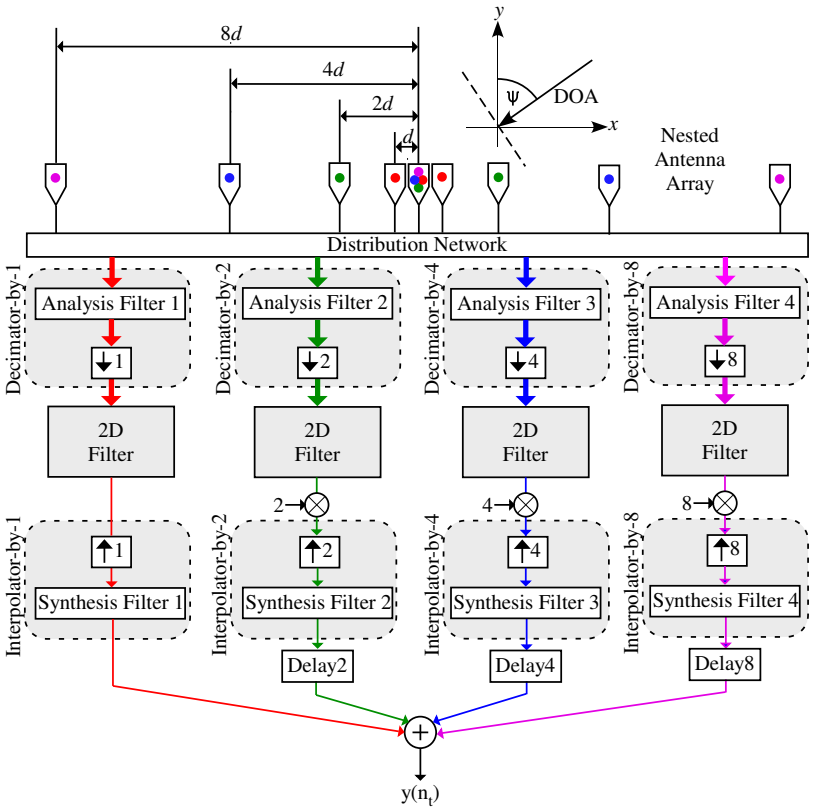


Figure 2.5: The structure of the NA beamformer.

Fig. 2.5 shows an example of a four-octave beamformer $(L = 4)$. The first ULA with elements spaced at $d_1 = \lambda/2$ $(\lambda_u = c/f_u$ $)$ is connected to the first subband beamformer and is processing the highest octave, $f_u/2 < f_t < f_u$. The l^{th} ULA ($l = 1, 2, ..., L$) with elements spaced at $d_l = 2^{l-1}d_1$ is connected to the l^{th} subband

beamformer which processes the l^{th} octave, $f_u/2^l < f_t < f_u/2^{l-1}$. Each array element of the l^{th} subband beamformer is connected through the distribution network to a decimator which consists of an analysis filter with unity gain within the l^{th} octave and zero elsewhere, to extract the related octave, and downsampler by 2^{l-1}. The downsampled signal is processed by a 2-D TF whose magnitude response in the (f_{ct}, f_z) plane is shown in Fig. 2.6. The passband area of the 2-D TF is designed to pass

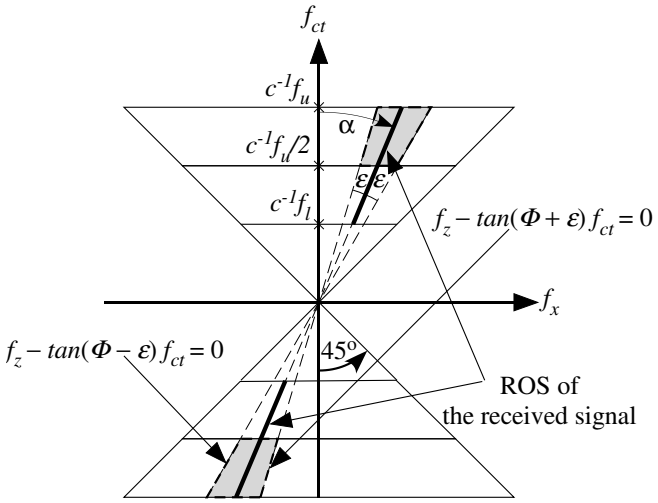


Figure 2.6: ROS of $\boldsymbol{F}(f_z, f_{ct})$, and the passband area of the 2-D TF.

Wideband PWs with the desired DOA and temporal frequencies within the range $c^{-1}f_u/2 < f_{ct} < c^{-1}f_u$. The signal in the desired DOA is obtained using the center output ($n_z = 0$) of the 2-D TF, $f_l(0, n_t)$ and is multiplied by 2^{l-1} to compensate for downsampling. Next, in order to go back to the original sampling rate, i.e. F_s, the output is applied to an interpolator which consists of upsampler by 2^{l-1} followed by a synthesis filter, to remove all replicas of the signal spectrum generated by upsampling except for the baseband copy. The synthesis filter that used has the same magnitude response as the analysis filter. To align the outputs of the subband beamformers, appropriate delays are added. The aligned signals are added and the resulting signal is obtained.

The NA beamformer architecture illustrated in Fig. 2.5 is for the one dimensional case, and it can be extended to two dimension [21].

2.5 2-D Trapezoidal Filter design

Spatial and temporal subsampling lead to the same frequency specifications for the TFs in all subband beamformers. The passband of this 2-D TF depends on the following two parameters; α and ϵ. The former is obtained based on the DOA of the desired signal, and the later controls the beam width around Φ . The passband is the area surrounded by the following lines:

$$f_x - \tan(\Phi - \epsilon) f_{ct} = 0, \quad f_x - \tan(\Phi + \epsilon) f_{ct} = 0 \tag{2.7}$$

$$f_{ct} \pm c^{-1} f_u = 0, \quad f_{ct} \pm c^{-1} f_u/2 = 0 \tag{2.8}$$

This is shown in Fig. 2.6. A linear phase FIR 2-D TF can be obtained by using inverse Fourier transform (IFT) of the ideal frequency response. A 2-D rectangular window is used here to truncate the impulse response:

$$\text{2-D window}(n_x,n_t) = \begin{cases} 1 & |n_x| \le N_x \text{ and } |n_t| \le N_t \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

2.6 Simulation Results

The simulation is provided to illustrate how the method works. Two Wideband PWs are considered; one as the desired PW (DOA = 50°) and the other one as an interference (DOA = 120°). For both cases the spectrum of the signal is equal to 1 within $f_{ct} = [10 \quad 160]$ MHz . Since $160/10 = 2^4$, a four-octave beamformer is needed. The sampling frequency $2f_u/\alpha$ is chosen to be 400 MHz ($\alpha = 0.8$). Analysis and synthesis FIR filters were designed using the Hamming window-based technique. Their amplitude responses are shown in Fig. 2.8.

To achieve almost PR, it is also required that the signal in the transition band of the analysis and synthesis filters is available at the output of the TF. For this reason, the passband specification of the 2-D TF is selected $f_{ct} = [40 \quad 200]$ instead of $f_{ct} = [80 \quad 160]$. The other parameters of the 2-D TF, i.e. Φ and ϵ, are set to 29.83° and 5°, respectively. The performance of the subband beamformer is illustrated in the frequency domain in Fig. 2.9. The ROS of the 2-D FT of the signals received by the l^{th} subarray are shown in Fig. 2.9(a)-(d). The horizontal and vertical axes are ω_z and ω_{ct}, respectively. In Fig. 2.9(b)-(d), aliasing can be observed due to spatial subsampling. Using the analysis filters to extract the related octave of the signal

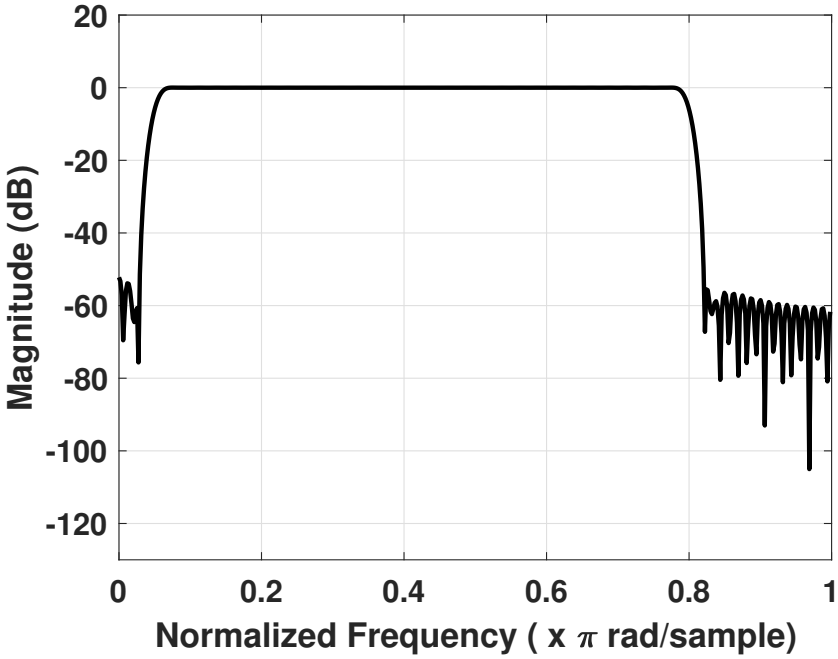


Figure 2.7: Bandwidth of the received signals.

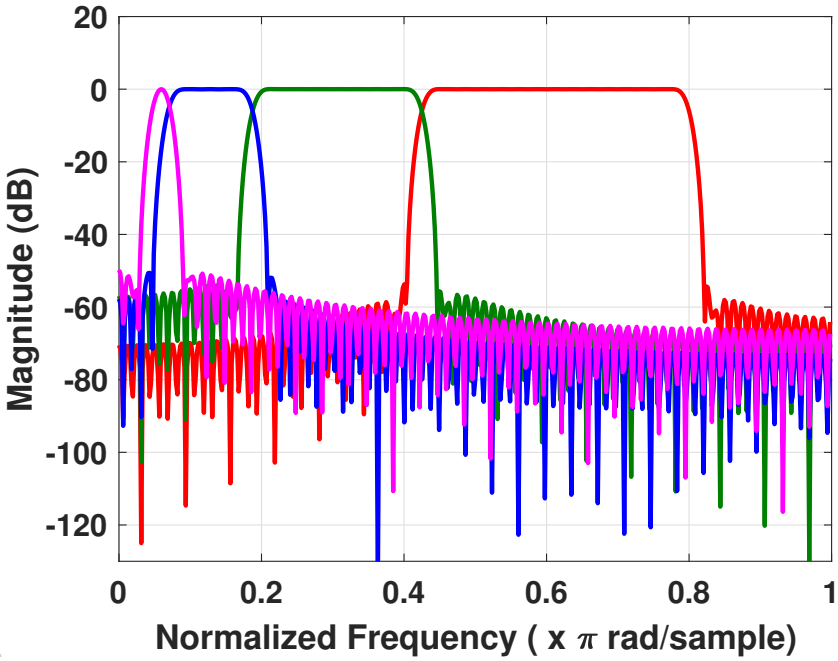


Figure 2.8: Analysis FIR filters amplitude responses.

spectrum, aliasing is eliminated as it can be seen from the ROS of the 2-D FT of the analysis filters output, in Fig. 2.9(e)-(h). Then, the output of the analysis filters is downsampled by 2^{l-1}. The ROS of the 2-D FT of the resulting output is shown in Fig. 2.9(i)-(l). Clearly, the ROS for all subbands outputs are the same and thus the same TF can be used in all subbands. The amplitude response of the designed TF for $N_z = N_t = 64$ is shown in Fig. 2.10. TF passes the desired signal and attenuates the interference. The output of the TF is upsamled, multiplied by 2^{l-1}, and filtered by the synthesis filter. In this example since the length of analysis (synthesis) filters are the same; there is no need for adding delay. Clearly, the output $y(n)$ of the beamformer is almost the delayed version of the desired PW as shown in Fig. 2.11. The amount of delay is the sum of delays due to the analysis, trapezoidal, and synthesis filters.

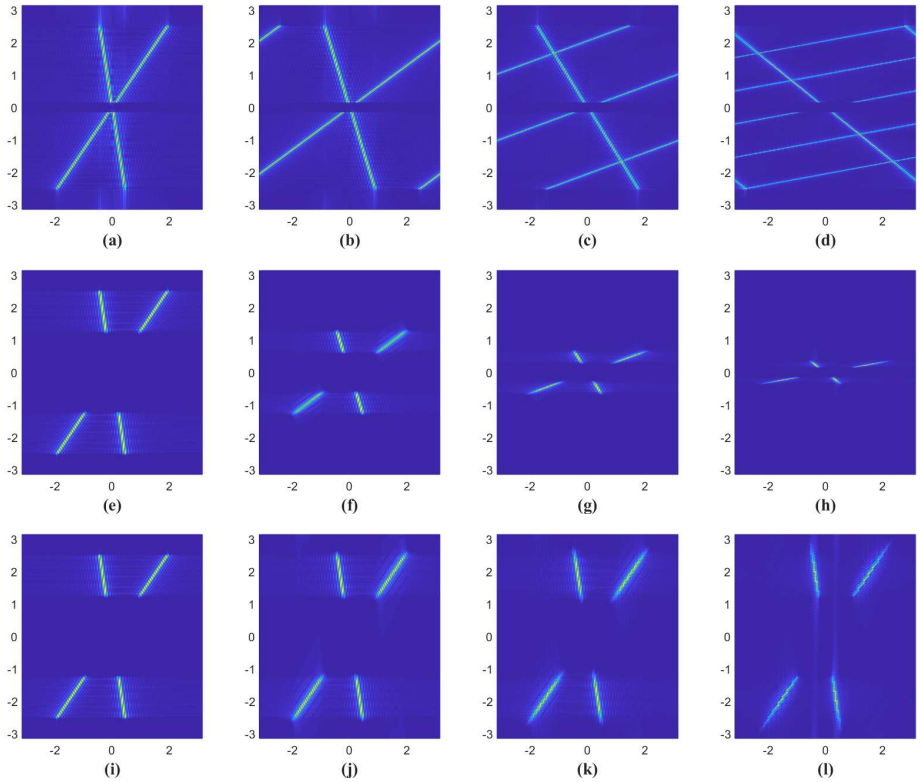


Figure 2.9: ROS of 2-D FT of: (a)-(d) the signals received by the l^{th} subarray, (e)(h) the analysis filters output, and (i)-(l) the output of downsamplers by 2^{l-1}.

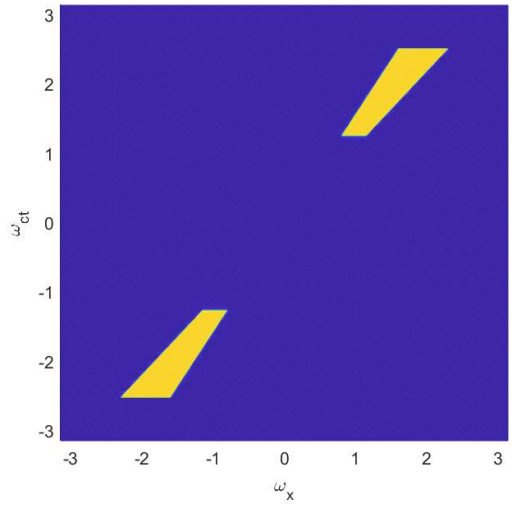


Figure 2.10: ROS of 2-D trapezoidal filter.

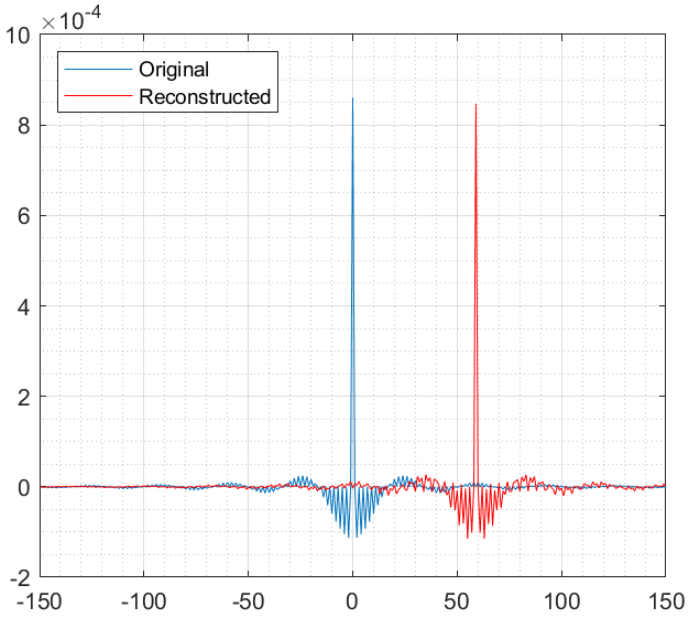


Figure 2.11: Beamformer output and desired signal in the time-domain.

2.7 Conclusions

A wideband beamforming technique using nested arrays, multirate filter banks, and 2-D ST filter is introduced. A comparison between NA and ULA structures is presented. MATLAB simulations for the NA beamformer are performed to show that it outperforms ULA beamformer in achieving high selectivity specially at low frequencies.

Chapter 3

Design Space Exploration of Decimators

This chapter presents a new systolic array structure for a decimator that merges the antiliasing FIR filter with the downsampler. The development of the structure is based on a systematic methodology. Using this methodology, a dependence graph for the decimator was obtained that combined the antialiasing filter and the downsampler. Different data scheduling and projection operations were developed to obtain the different proposed designs. Six systolic array design options were obtained and evaluated. The fastest design was selected for hardware implementation and compared with the other two well-known decimator designs; namely conventional design in which the antialiasing filter is followed by a downsample and the polyphase design in which a commutator is followed by the polyphase antialiasing filter. FPGA implementations for the proposed and the other two designs confirm that the proposed decimator implementation outperforms in terms of area, speed, and power as the decimation factor increases regardless of the number of FIR filter coefficients.

3.1 Introduction and related work

Generally, decimators are used in multirate systems to generate signals with lower data rates. Examples where decimators are an essential component include FIR filters with steep transition band [22][23], nested arrays broadband bemformers [11][24] which is the scope of this dissertation, the baseband digital signal processing (DSP) of software-defined radio (SDR) to enable configurable sample rates [25], the digital

down converter (DDC) of a 4G receiver systems [26], the digitally-enhanced high speed analog-to-digital converter (ADC) for achieving higher signal-to-noise ratio (SNR) in wireless and software defined radio applications [27][28], the quadrature mirror filter (QMF) for equalizing wireless communications channels [29], and discrete Fourier transform (DFT) filter bank beamformers [30]. Decimators can also be found in other general areas such as radar [31][32][33][34], communications [35], speech [36][37] and image processing [38][39].

Decimators structure consists of an antialiasing filter followed by a downsampler. The structure of the downsampler is much simpler than the structure of the filter. This is the reason why optimizing the design of decimators focuses on the optimizing the design of the antialiasing filter.

Rahate et al. [37], presented a design analysis of decimation filters for hearing aid applications on FPGA. The decimation filter is a combination of digital integrator and digital differentiator stages, which can perform the operation of digital low pass filtering and decimation at the same time.

Mehra and Singh [40], proposed an efficient structure for rational sampling rate converter by combining interpolator and decimator with a low pass filter.

Harize et al. [41], presented a methodology for the implementation of decimation FIR filters on FPGA. The methodology was based on using distributed arithmetic (DA) to replace the multipliers with LUTs. The methodology was also used to realize the decimator using polyphase structure. However, two factors contributed to limiting the speed of their proposed design. The use of DA resulted in a lower sampling rate than the system clock frequency by a factor equal to the data word size.

Liu et al. [42], presents an implementation of digital down conversion method based on applying the parallel structure of polyphase filter banks.

Jetly et al. [43], presents a method for the implementation decimation and interpolation FIR filters to be integrated with the digital baseband receiver chain of a vehicular communication platform. The decimation filter used consists of a polyphase decomposed FIR filter of order 7 and a downsampler with downsampling factor equals to 4.

Jayaprakasan and Madheswaran [44], presented the implementation of a two stage FIR decimation filter and compared it with single stage implementation for WiMAX applications. Results show that the two stage FIR filter utilizes less LUTs and consumes less power than the single stage one.

Zheng et al. [45], decomposed the multirate FIR filter (MRFIR) into output

computational threads. Each thread represents an instance of the finite inner-product required to produce a single output of the MRFIR. The filter is thus viewed as a finite collection of concurrent threads.

Mehra and Arora, [46] presented an efficient multiplier-less technique to design and implement a high speed cascaded integrator comb (CIC) decimator for wireless applications. The proposed design can operate at an estimated frequency of 276.6 MHz and uses a relatively few hardware resources.

Some other approaches deals with improving the design of the individual elements of the circuits. Such approaches are the following:

Aljuffri et al. [47], used Wallace Tree and Vedic multipliers for implementation of 8-tap and 16-tap sequential and parallel micro programmed FIR filters architectures. The designs are realized using FPGA. The sequential FIR filters architecture designed using Wallace Tree multiplier seems to be more efficient as compared to Vedic multipliers.

Prasanna and Rani [48], implemented 16-tap symmetric FIR filter using a reduced parallel LUT decomposed DA approach which is implemented using a FPGA device. The design reduces the number of LUTs and offers 60.5% less delay than a systolic DA based design.

Thakur and Khare [49], presented a high speed FPGA implementation of FIR filter. The design offers a minimum period of 4.255 ns and maximum frequency of 235.026 MHz.

In this chapter, a systematic methodology presented in [6] is used to develop systolic array structures for decimators. This methodology is used to find best data scheduling strategies and explore possible structures. Six structures with output data pipelining will be considered due to promising low clock speeds. The one requiring the least power and area will be selected and implemented using FPGA. The performance of this implementation will be compared with implementations of existing well-known decimator structures. Results indicate that the proposed design has higher clock speed and lower area and power requirements, especially when the decimation ratio is increased, than the well-known decimator structures.

3.2 Systematic methodology for systolic array design applied to the decimator

In order to perform systolic array design space exploration of the decimator, we use a systematic methodology that was proposed in [6] for regular iterative algorithms (RIAs). The decimator algorithm is given by:

$$u(i) = \sum_{j=0}^{J-1} h(j)x(i-j) \tag{3.1}$$

$$y(i) = u(iM) \tag{3.2}$$

where M is the decimation factor, J is the number of the antialiasing FIR filter coefficients, $h(j)$ is the FIR filter coefficients, $x(i-j)$ is the decimator input samples, $u(i)$ is the FIR filter output and is the input to the downsampler, and $y(i)$ is the decimator output.

The methodology of [6] specifies several steps which are adapted here for the decimator algorithm as follows:

1. The difference equations of the decimator algorithm, expressed as an RIA, are shown in Eqs. (3.1) and (3.2).

2. Define a computational domain $\mathcal{D} \subset \mathcal{Z}^n$ of the algorithm based on the RIA. Since Eqs. (3.1) and (3.2) use two indices i and j, the algorithm is defined in the 2-dimensional integer domain $\mathcal{Z}^2$. In Section 3.3 the computational domain of the decimator $\mathcal{D} \subset \mathcal{Z}^2$ is defined through investigation of the ranges of indices i and j.

3. Define the subdomain of each variable in $\mathcal{D}$ using the dependence of the algorithm variables on the iteration indices. This is explained in Section 3.3.

4. Obtain the scheduling functions that satisfy the input and output data timing specifications and constrains. In Section 4.5, valid scheduling functions for the decimator algorithm are explored.

5. Obtain the projection functions that satisfy the scheduling functions and any hardware restrictions. In Section 4.6, projection directions associated with valid scheduling functions for the decimator algorithm are being explored.

The term systolic array design will be used to describe the architecture and the functionality of a systolic array while systolic array implementation represents the actual implementation of this design in hardware.

3.3 Decimator dependence graph (DG)

The decimator Eqs. (3.1) and (3.2) define a sequential evaluation of the decimation algorithm. The algorithm is iterative and depends on two indices i and j. There are two input variables $h(j)$ and $x(i-j)$. There is one intermediate variable $u(i)$ and one output variable $y(i)$. We use the powerful systematic technique of reference [6] to perform systolic array design space exploration of the decimator structure based on the iterations defined by Eqs. (3.1) and (3.2). Fig. 3.1 shows the dependence graph of the decimator for the case when $M = 4$ and $J = 8$. The horizontal axis is the i-axis (range $i \geq 0$) and vertical axis is the j-axis (range $0 \leq j < 8$).

The decimator output $y(i)$ is shown at the top of Fig. 3.1. $y(i)$ is shown by the thick vertical lines since each output sample depends on the i index only. Note that sample $y(i)$ corresponds to the intermediate sample $u(iM)$ according to Eq. (3.2).

The empty circles indicate useful filtering operations that result in the generation of the output samples $y(i)$.

In this work, the systematic methodology is employed using different scheduling

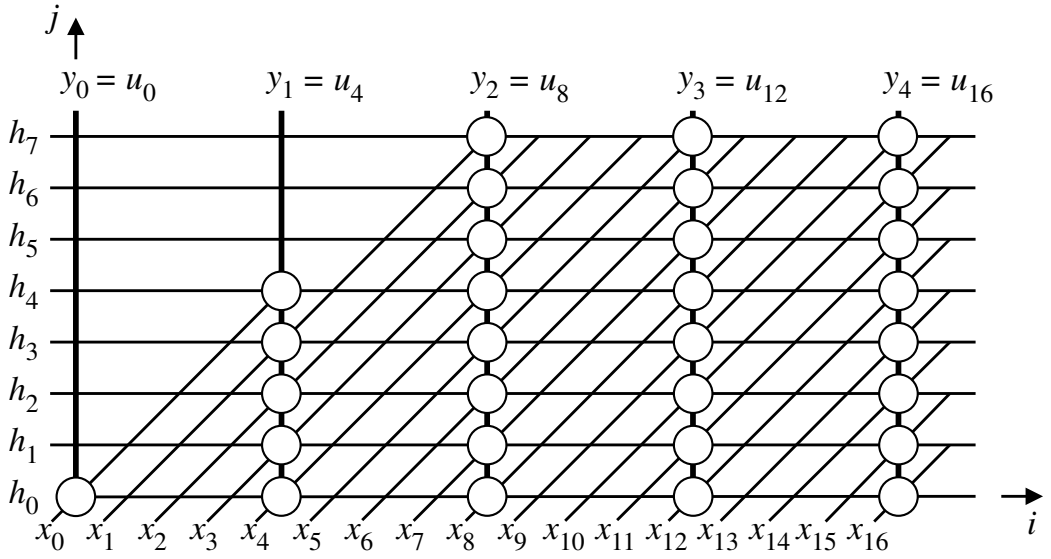


Figure 3.1: M-to-1 decimator dependence graph for the case when $M = 4$ and $J = 8$. Empty circles denote multiply/accumulate operations.

and projection operations for systolic array design space exploration of the decimator.

3.4 Decimator scheduling function

The scheduling function assigns a time index value for the operation of each point in the DG of Fig. 3.1. A simple linear scheduling function is used to assign time index values to the DG nodes:

$$t(\boldsymbol{p}) = \boldsymbol{s}\boldsymbol{p} \tag{3.3}$$

where $\boldsymbol{s} = [\alpha \quad \beta]$ is the schedule row vector and $\boldsymbol{p} = [i \quad j]^T$ is a point in the i-j plane of Fig. 3.1. Therefore the time associated with a node is given by:

$$t(\boldsymbol{p}) = i\alpha + j\beta \tag{3.4}$$

An edge connecting two nodes having the same time value are said to be lying on the same equitemporal zone. Data flowing between these two nodes is said to be broadcast since the data value is shared by the two nodes at the same time value. An edge connecting two nodes with different time values indicate that data is pipelined from the node with lower time value to the node with higher time value. The scheduling function transforms the DG to a directed acyclic graph (DAG) [6].

The scheduling vector components α and β are determined subject to input and output data specifications and the decision whether to pipeline or broadcast a variable.

It is assumed that the input data $x(i)$ arrive at consecutive time steps, we can write:

$$t(\boldsymbol{p}_2) = t(\boldsymbol{p}_1) + 1 \tag{3.5}$$

Assuming further that $x(i)$ sample is supplied to the DG at point $\boldsymbol{p} = [i \quad 0]^T$, we have:

$$t(\boldsymbol{p}_1) = \alpha\, i \tag{3.6}$$

$$t(\boldsymbol{p}_2) = \alpha(i+1) \tag{3.7}$$

Equations (3.5)-(3.7) result in $\alpha = 1$. A valid scheduling vector that satisfies the above assumptions about input data timing is given by

$$\boldsymbol{s} = [1 \quad \beta] \tag{3.8}$$

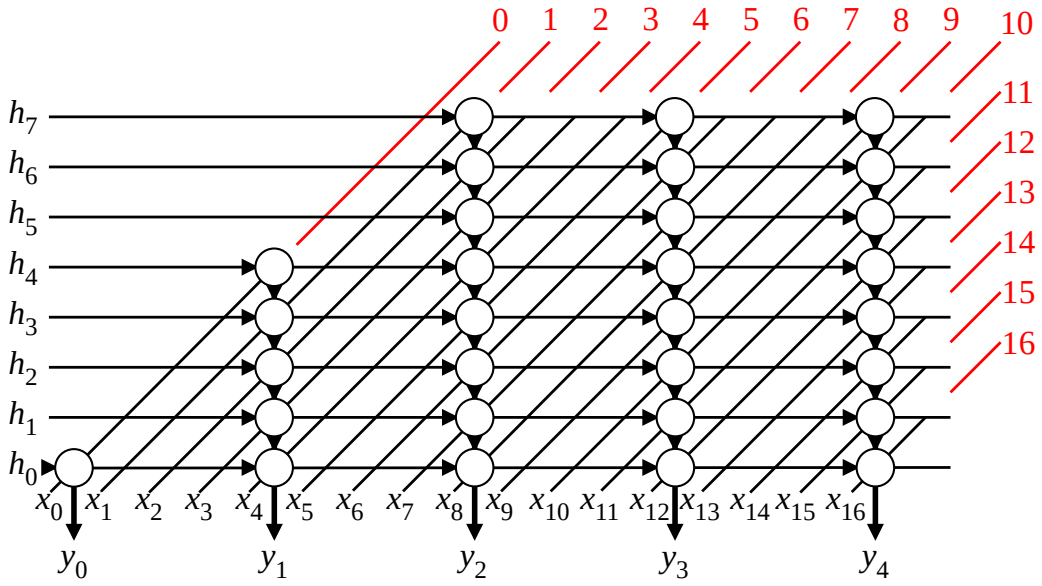


Figure 3.2: The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_1$ in Eq. (3.9).

The value of β will be determined by our choice of whether we need to pipeline or broadcast the output sample $y(i)$. We have three possible valid scheduling vectors that we can employ:

$$\boldsymbol{s}_1 = [1 \quad -1] \tag{3.9}$$

$$\boldsymbol{s}_2 = [1 \quad 1] \tag{3.10}$$

$$\boldsymbol{s}_3 = [1 \quad 0] \tag{3.11}$$

Figs. 3.2, 3.3, and 3.4 show the DAG corresponding to scheduling vectors $\boldsymbol{s}_1$, $\boldsymbol{s}_2$, and $\boldsymbol{s}_3$ respectively. The equitemporal zones and the time index values are indicated by the red lines and the red numbers respectively. The inputs $x(i-j)$ and $h(j)$ are indicated by the arrows and the output $y(i)$ is indicated by the vertical lines. The empty circles indicate all the multiply/accumulate operations for each output sample.

The scheduling vector $\boldsymbol{s}_1$ results in broadcast input $x(i-j)$ and pipelined output $y(i)$. The scheduling vector $\boldsymbol{s}_2$ results in pipelined input $x(i-j)$ and pipelined output $y(i)$. The scheduling vector $\boldsymbol{s}_3$ results in pipelined input $x(i-j)$ and broadcast output

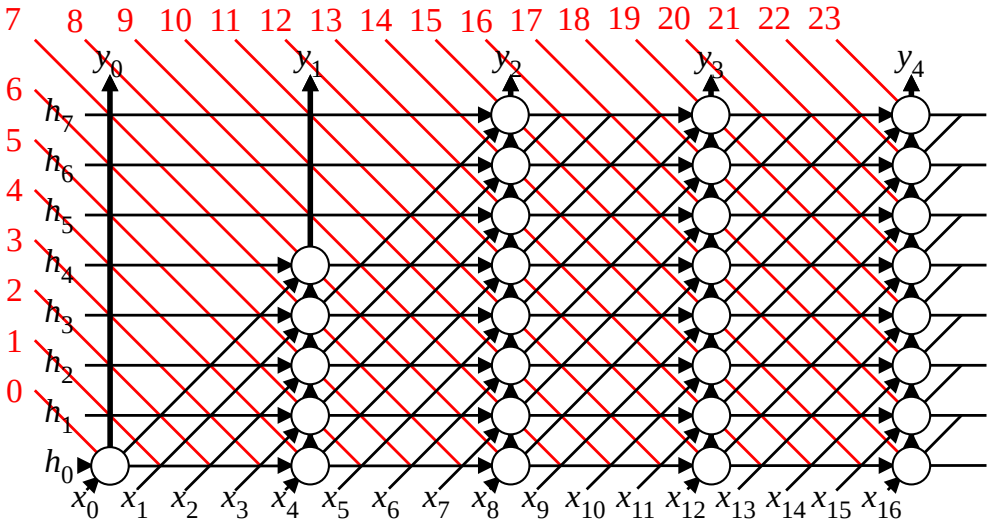


Figure 3.3: The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_2$ in Eq. (3.10).

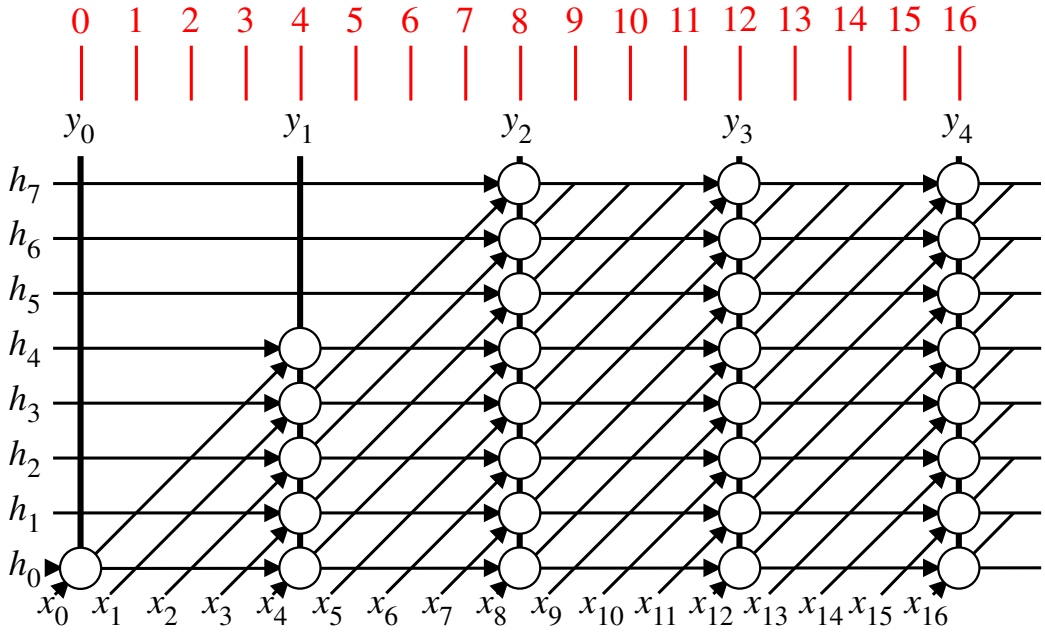


Figure 3.4: The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_3$ in Eq. (3.11).

$y(i)$. Since Pipelined output allow a fastest clock rate and broadcast output results in expensive hardware and/or slow speed, the scheduling vector $\boldsymbol{s}_3$ shall be excluded from the design space exploration choices.

3.5 Decimator node projection

Linear projection is defined as the mapping of several points in an n-dimensional DG to processing element PE_k where k is the processor index that given by [6]:

$$k = \boldsymbol{P}\boldsymbol{p} \tag{3.12}$$

where $\boldsymbol{P}$ is the projection matrix that projects a point $\boldsymbol{p} \in \mathcal{D}$ to a point k in a 1-D domain.

According to [6], the projection matrix $\boldsymbol{P}$ is determined by selecting a projection direction $\boldsymbol{d}$ that is in the nullspace of $\boldsymbol{P}$. The selection of a valid projection direction should satisfy:

$$\boldsymbol{s}\boldsymbol{d} \neq 0 \tag{3.13}$$

where $\boldsymbol{s}$ is the chosen scheduling vector. In this work, it is aimed to map the points in the 2-D DG shown in Fig. 3.1 to a 1-D domain.

Applying condition (3.13) to the three scheduling vectors in Eq. (3.9), (3.10) and (3.11), four different projection directions are obtained:

$$\boldsymbol{d}_1 = [1 \quad 0]^T \tag{3.14}$$

$$\boldsymbol{d}_2 = [1 \quad 1]^T \tag{3.15}$$

$$\boldsymbol{d}_3 = [1 \quad -1]^T \tag{3.16}$$

$$\boldsymbol{d}_4 = [0 \quad 1]^T \tag{3.17}$$

Only three of the previous projection directions are valid for each schehuling vector. For $\boldsymbol{s}_1$, the vectors $\boldsymbol{d}_1$, $\boldsymbol{d}_3$, and $\boldsymbol{d}_4$ are valid. For $\boldsymbol{s}_2$, the vectors $\boldsymbol{d}_1$, $\boldsymbol{d}_2$, and $\boldsymbol{d}_4$ are valid. For $\boldsymbol{s}_3$, the vectors $\boldsymbol{d}_1$, $\boldsymbol{d}_2$, and $\boldsymbol{d}_3$ are valid.

Based on the above discussion, the three scheduling vectors and their associated projection directions produce nine different systolic arrays. This is discussed in more details in Section 4.8

These projection directions are then used to calculate the associated projection

matrices according to the procedure described in [6]. Given a project direction $\boldsymbol{d} = [\gamma \quad \delta]^T$, and knowing that $\boldsymbol{d}$ is the nullvector of the projection matrix $\boldsymbol{P}$ [6], There are two possible forms:

$$\boldsymbol{P} = [\delta \quad -\gamma] \quad \text{or} \quad \boldsymbol{P} = [-\delta \quad \gamma] \tag{3.18}$$

The proper form to use is the one that results in positive PE indices after the project operation.

Using Eqs. (3.12) and (3.18) we get :

$$k = i\delta - j\gamma \tag{3.19}$$

For more flexibility in this work, a non-linear node projection operation have been adopted of the form [6]:

$$k' = \left\lfloor \frac{k}{m} \right\rfloor \tag{3.20}$$

where k was given in Eq. (3.12) and m is the desired number of points in DAG that will be assigned to one PE.

3.6 Systolic array design space exploration

In this section, the systolic array design space of linear processor array will be explored for the decimator using the calculated scheduling vectors and projection directions.

Table 3.1 illustrates how nine possible systolic array design options are obtained. The nine possible design options are obtained based on the combination of the scheduling functions and projection directions, as explained in the table. The advantages and disadvantages were also summarized in the table.

Based on the discussion in Section 4.5, scheduling vector $\boldsymbol{s}_3$ will not be considered. Therefore, there is only one valid scheduling vectors $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ with their associated valid projection directions. This gives a total of six possible design options that allow the fastest clock rate. The following subsections discuss the six design options to be explored.

Table 3.1: The possible systolic array design-options

Design-Option #	Scheduling Vector	Projection Direction	Main Features
1	$\boldsymbol{s}_1$	$\boldsymbol{d}_1$	Pipelined output
2		$\boldsymbol{d}_3$	
3		$\boldsymbol{d}_4$	
4	$\boldsymbol{s}_2$	$\boldsymbol{d}_1$	Pipelined output
5		$\boldsymbol{d}_2$	
6		$\boldsymbol{d}_4$	
7	$\boldsymbol{s}_3$	$\boldsymbol{d}_1$	Broadcast output
8		$\boldsymbol{d}_2$	
9		$\boldsymbol{d}_3$	

3.6.1 Design-Option #1: using $\boldsymbol{s}_1 = [1 \ -1]$ and $\boldsymbol{d}_1 = [1 \quad 0]^T$

The DAG corresponding to $\boldsymbol{s}_1$ is shown in Fig. 3.5. The time index associated with any point $\boldsymbol{p}$ is given by:

$$t(\boldsymbol{p}) = i - j \tag{3.21}$$

According to Eq. (3.18), the projection matrix corresponding to the linear projection direction $\boldsymbol{d}_1$ is given by:

$$\boldsymbol{P}_1 \ = \ [0 \quad 1] \tag{3.22}$$

Using Eq. (3.19) and (3.22), the processing element (PE) index associated with point $\boldsymbol{p}$ is given by:

$$p' = j \tag{3.23}$$

The PE index values are indicated by the numbers inside the circles in Fig. 3.5. The number of PEs will be finite and equal to the number of antialiasing filter coefficients J.

By observing Fig. 3.5 we found that there are only two PEs active during each time step. This indicates that we must use nonlinear node projection operation that maps each node to only one of two PEs. For Design-Option #1, a general non-linear node projection operation we use is defined as [6]:

$$p'' = \left\lfloor \frac{j}{J/M} \right\rfloor \tag{3.24}$$

where p'' is the new PE index. For the case when $J = 8$ and $M = 4$ we would have only two PEs as was observed in Fig. 3.5. The nodes that map to PE_0 and PE_1 are shown as the grey and blue circles respectively in Fig. 3.5. Fig. 3.5 now shows that only two processors are active at each time step. The number of PEs in general is given by J/M. Fig. 3.5 indicates that the nodes implement the basic operation of multiply/accumulate (MAC) operation. To reduce roundoff effects and computational noise, double-precision MAC is implemented in the form:

$$2^w e_h + e_l = a \times b + 2^w d_h + d_l \tag{3.25}$$

where a, b are single-precision multiplier and multiplicand inputs, d_h, d_l are the high-order and low-order word input to the double-precision adder, e_h, e_l is the high-order and low-order word output from the double-precision adder, and w is the number of bits in the high-order and low-order words. The delay to perform the multiplication followed by a double-precision addition is $4T$ where T is the delay of the multiplier or the double-precision adder.

In the proposed designs Fig.3.6(b), and later in Fig. 3.10(b), a high-speed, double-precision carry-save multiplier/accumulator (CS-MAC) developed in [50] is used as :

$$2^w \left(s_{out} + c_{out}\right) + l_{out} = a \times b + 2^w \left(s_{in} + c_{in}\right) + l_{in} \tag{3.26}$$

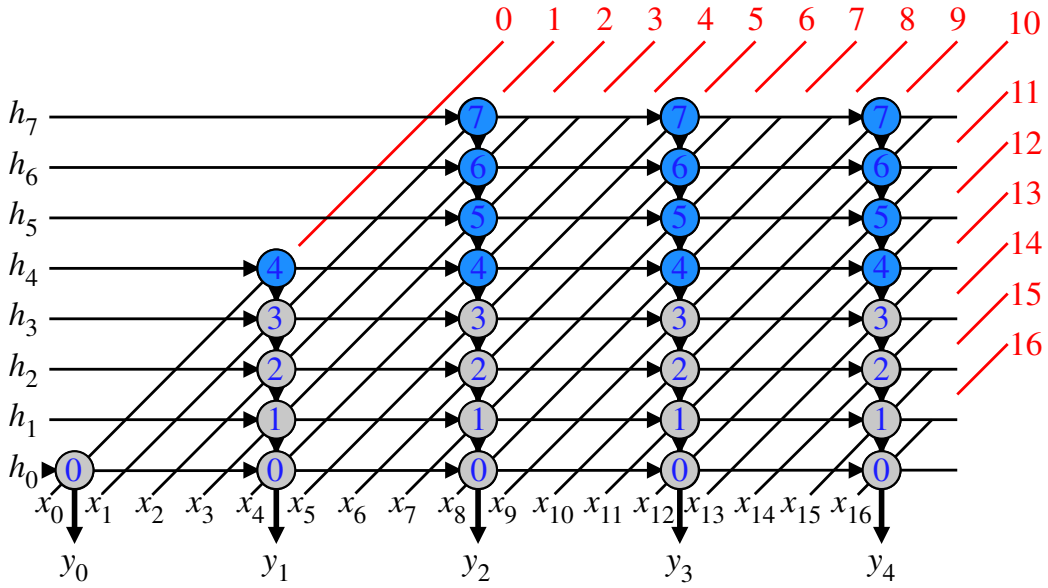


Figure 3.5: The DAG for Design-Option #1: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_1 = [1 \quad 0]^T$.

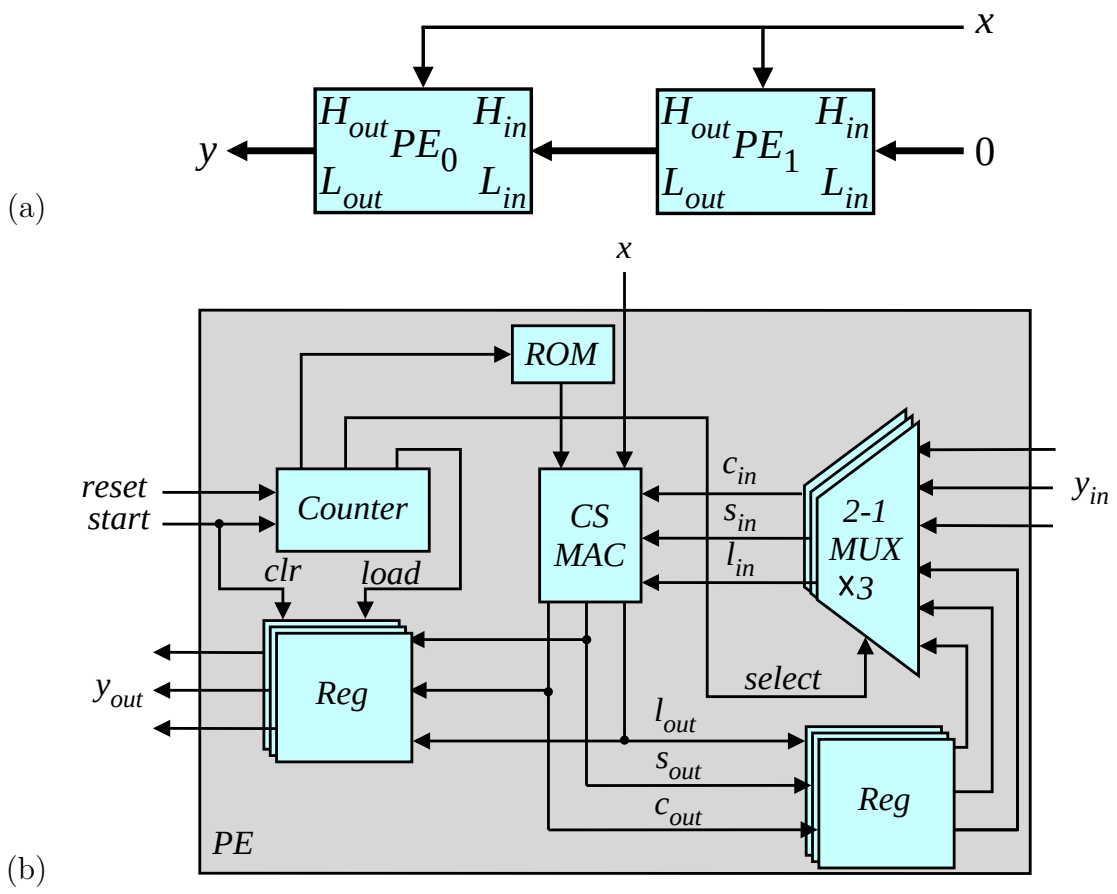


Figure 3.6: Systolic array for Design-Option #1. (a) The systolic array when $J = 8$ and $M = 4$. (b) PE details.

where s_{out}, c_{out} are the high-order sum and carry word output from the CS-MAC, l_{out} is the low order word output from the CS-MAC, s_{in}, c_{in} are the high-order sum and carry word input to the CS-MAC, and l_{in} is the low order word input to the CS-MAC.

A reduction in the computation time ranging from 20% to 50% compared with other schemes has been achieved using the CS-MAC without a significant increase in the required area [50].

Fig. 3.6 shows the systolic array and the detailed PE structure that results after applying the scheduling function $\boldsymbol{s}_1$ and the non-linear projection direction $\boldsymbol{p''}$ for Design-Option #1. Fig. 3.6(a) shows the resulting systolic array. Two PEs are shown since $J = 8$ and $M = 4$. The input data x is broadcast and the double-precision output $y_{out} = H_{out}2^w + L_{out}$ is pipelined. The output data flows from PE_1

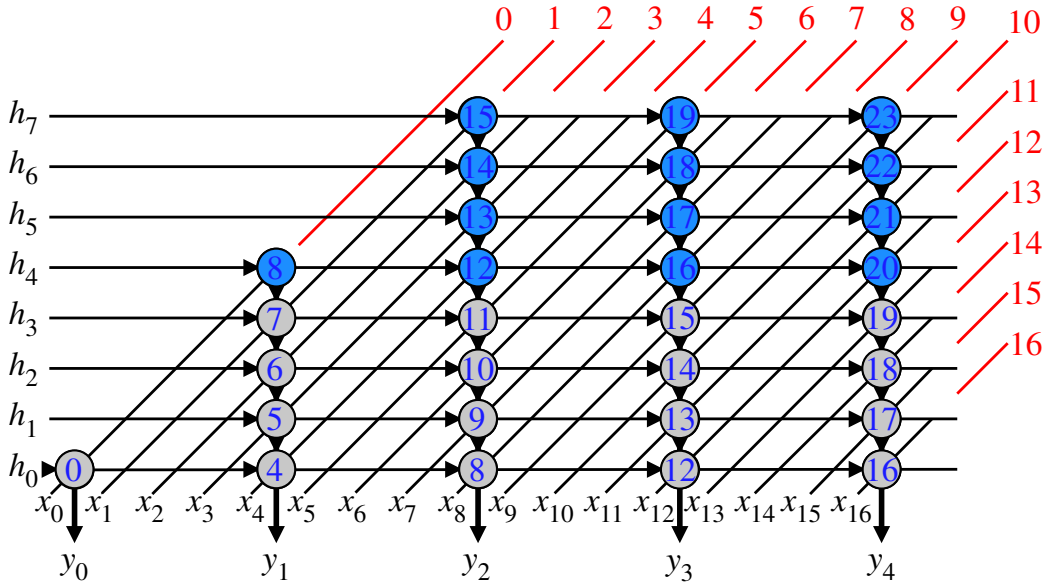


Figure 3.7: The DAG for Design-Option #2: using $\boldsymbol{s}_1 = [1 \ \ -1]$ and $\boldsymbol{d}_3 = [1 \ \ -1]^T$.

to PE_0. Fig. 3.6(b) shows the PE details. Examination of Fig. 3.5 reveals that the antialiasing filter coefficients allocated to each PE are stored in reverse order. For example, PE_0 will store its filter coefficients in the ROM as $\{h_3,\ h_2,\ h_1,\ h_0\}$. The Ripple Carry Adder (RCA), shown in Fig. 3.6(b), is used to reduce the high-order sum and carry output s_{out}, c_{out} from the CS-MAC to the high order word H_{out}.

3.6.2 Design-Option #2: using $\boldsymbol{s}_1 = [1 \ \ -1]$ and $\boldsymbol{d}_3 = [1 \ \ -1]^T$

In this design-Option, we use the same scheduling function $\boldsymbol{s}_1$ with different linear projection direction $\boldsymbol{d}_3$. The corresponding projection matrix will be:

$$\boldsymbol{P}_3 \ = \ [1 \quad 1] \tag{3.27}$$

Using Eq. (3.19) and (3.27), the processing element (PE) index associated with point $\boldsymbol{p}$ is given by:

$$p' = i + j \tag{3.28}$$

Fig. 3.7 shows the DAG for Design-Option #2 together with the equitemporal zones and the PE indices as shown inside the circles. We notice that the total number of PEs is infinite in this case since index i extends to ∞. However, we also notice that only two PE are active at any given time step. We could use the same non-linear

node projection used in Eq. (3.24) for Design-Option #1.

As we use the same scheduling function $\boldsymbol{s}_1$ and the non-linear projection direction $\boldsymbol{p''}$ of Design-Option #1, The systolic array will be the same as in Fig. 3.6(a).

3.6.3 Design-Option #3: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$

The projection matrix corresponding to the linear projection direction $\boldsymbol{d}_4$ will be:

$$\boldsymbol{P}_4 \quad = \quad [1 \quad 0] \tag{3.29}$$

Using Eq. (3.19) and (3.29), the processing element (PE) index associated with point $\boldsymbol{p}$ is given by:

$$p' = i \tag{3.30}$$

The total number of PEs is infinite since index i extends to ∞.

Fig. 3.8 shows the DAG for Design-Option #3 together with the equitemporal zones and the PE indices as shown inside the circles. We also notice that only two PE are active at any given time step. We could use the same non-linear node projection used in Eq. (3.24) for Design-Options #1 and #2.

As we use the same scheduling function $\boldsymbol{s}_1$ and the non-linear projection direction $\boldsymbol{p''}$ of Design-Option #1. The systolic array will be the same as in Fig. 3.6(a).

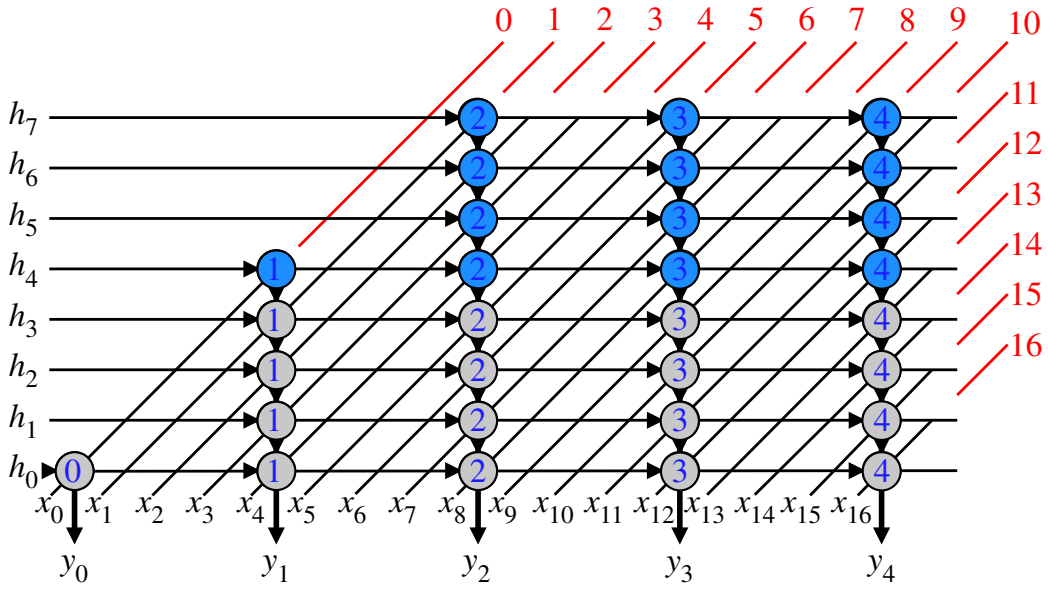


Figure 3.8: The DAG for Design-Option #3: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$.

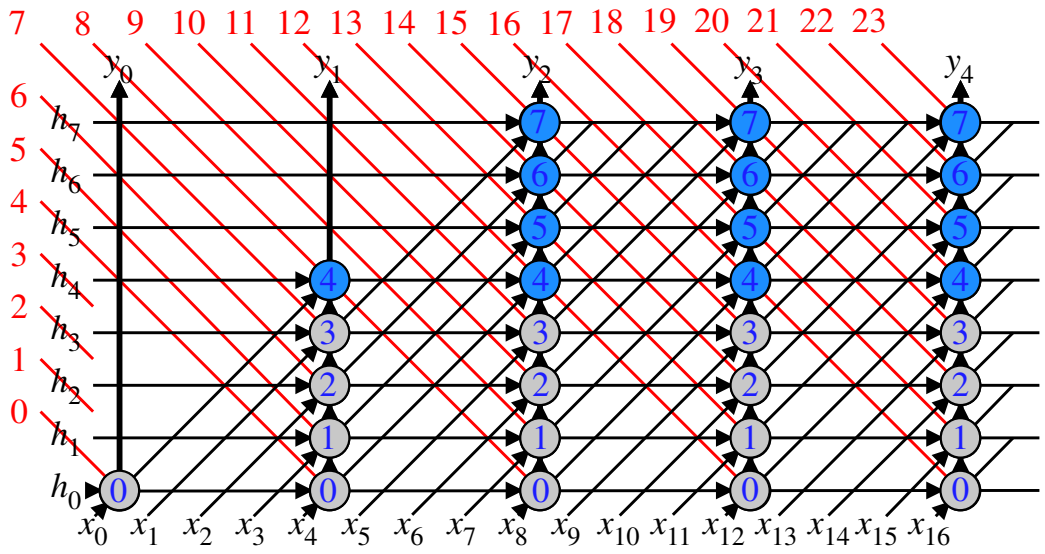


Figure 3.9: The DAG for Design-Option #4: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_1 = [1 \quad 0]^T$.

3.6.4 Design-Option #4: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_1 = [1 \quad 0]^T$

Design-Option #4 uses scheduling function $\boldsymbol{s}_2$. Fig. 3.9 shows the equitemporal zones. The time index associated with point $\boldsymbol{p}$ is given by:

$$t(\boldsymbol{p}) = i + j \tag{3.31}$$

Design-Option #4 uses the same linear projection direction $\boldsymbol{d}_1$ that was used in Design-Option #1. The corresponding projection matrix is given by Eq. (3.22). The PE index associated with a point $\boldsymbol{p}$ in the DG is given by Eq. (3.23). Similar to Design-Option #1, the number of processors is finite and also two PEs are active at any given time step. We also use the non-linear node projection $\boldsymbol{p''}$ given by Eq. (3.24) to project the DAG to a set of tasks that can be executed concurrently in hardware systolic arrays.

The output data timing of Fig. 3.9 requires careful analysis. We notice, for example, that output y_4 is ready at time 24 from PE_1. This output should have been obtained at time 16. This indicates a latency or 8 clock cycles. Also, the input data sequence must be resynchronized before it fed to the CS-MAC.

Referring to Fig. 3.9, the timing information for x_{in}, x_{int}, and y_{out} is shown in Table 3.2.

Fig. 3.10 shows the systolic array and the PE structure that results after applying the scheduling function $\boldsymbol{s}_2$ and the non-linear projection direction $\boldsymbol{p''}$. Fig. 3.10(a)

shows the resulting systolic array for Design-Option #4. As in Design-Option #1, the number of PEs is two. Both the input data x and output data y are pipelined. The output data flows from PE_0 to PE_1. Fig. 3.10(b) shows the PE details. The antialiasing filter coefficients are stored in the ROM in a normal order. However, the input data x_{in} are used by the PE's in reverse order. For example, PE_0 will use the input data x in the first high rate four clock cycles 0–3 in the order $\{x_3,\ x_2,\ x_1,\ x_0\}$. The input data x should be delayed by a value of $2M$.

We infer from Table 3.2 and Fig. 3.10(b) that we need more hardware resources for the delay and the reordering compared to the design-Option in Fig. 3.6(b)

3.6.5 Design-Option #5: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_2 = [1 \quad 1]^T$

In this design-Option, we use the same scheduling function $\boldsymbol{s}_3$. The time index associated with any point $\boldsymbol{p}$ is given by Eq. (3.31).

The projection matrix corresponding to the linear projection direction $\boldsymbol{d}_2$ is given

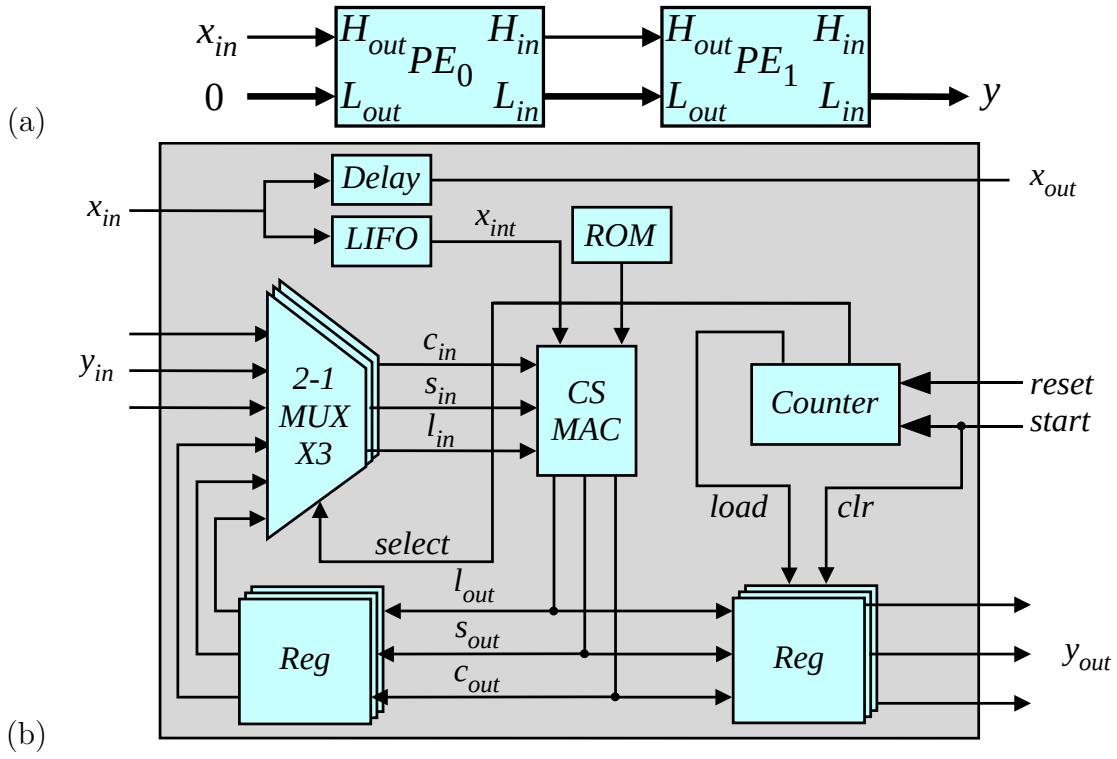


Figure 3.10: Systolic array for Design-Option #4. (a) The systolic array when $J = 8$ and $M = 4$. (b) PE details.

Table 3.2: Input/output timing for PE_0 and PE_1 in Fig. 3.10.

Time	PE_0			PE_1		
	x_{in}	x_{int}	y_{out}	x_{in}	x_{int}	y_{out}
0	x_0	x_0	0	0	0	0
1	x_1	0		0	0	
2	x_2	0		0	0	
3	x_3	0		0	0	
4	x_4	x_4	y'_0	0	0	0
5	x_5	x_3		0	0	
6	x_6	x_2		0	0	
7	x_7	x_1		0	0	
8	x_8	x_8	y'_1	x_0	x_0	y_0
9	x_9	x_7		x_1	0	
10	x_{10}	x_6		x_2	0	
11	x_{11}	x_5		x_3	0	
12	x_{12}	x_{12}	y'_2	x_4	x_4	y_1
13	x_{13}	x_{11}		x_5	x_3	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

by:

$$\boldsymbol{P}_2 \;=\; [1 \;\; -1] \tag{3.32}$$

Using Eq. (3.19) and (3.32), the processing element (PE) index associated with point $\boldsymbol{p}$ is given by:

$$p' = i - j \tag{3.33}$$

The total number of PEs is infinite since index i extends to ∞.

Fig. 3.11 shows the DAG for Design-Option #5 together with the equitemporal zones and the PE indices as shown inside the circles. We also notice that only two PE are active at any given time step. We could use the same non-linear node projection used in Eq. (3.24) for the previous design-Options.

As we use the same scheduling function $\boldsymbol{s}$ and the non-linear projection direction $\boldsymbol{p''}$ of Design-Option #4. The systolic array will be the same as in Fig. 3.10.

3.6.6 Design-Option #6: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$

This design-Option uses the same scheduling function $\boldsymbol{s}_2$ that was used in Design-Options #4 and #5. The time index associated with any point $\boldsymbol{p}$ is given by Eq. (3.31).

It also uses the same linear projection $\boldsymbol{d}_4$ that was used in Design-Option #3. The corresponding projection matrix is given by Eq. (3.29) The PE index associated with a point $\boldsymbol{p}$ in the DG is given by Eq. (3.30).

The total number of PEs is infinite since index i extends to ∞.

Fig. 3.12 shows the DAG for Design-Option #6 together with the equitemporal zones and the PE indices as shown inside the circles. We also notice that only two PE are active at any given time step. We could use the same non-linear node projection used in Eq. (3.24) for the all previous design-Options.

As we use the same scheduling function $\boldsymbol{s}_2$ and the non-linear projection direction $\boldsymbol{p''}$ of Design-Option #4. The systolic array will be the same as in Fig. 3.10.

3.6.7 Comparing the proposed designs

Six different systolic array design options were obtained. Design-Options #1, #2 and #3 map to one systolic array. Design-Options #4, #5 and #6 map to another type of systolic array. The former systolic array requires less hardware resources and is the one that proposed in this work.

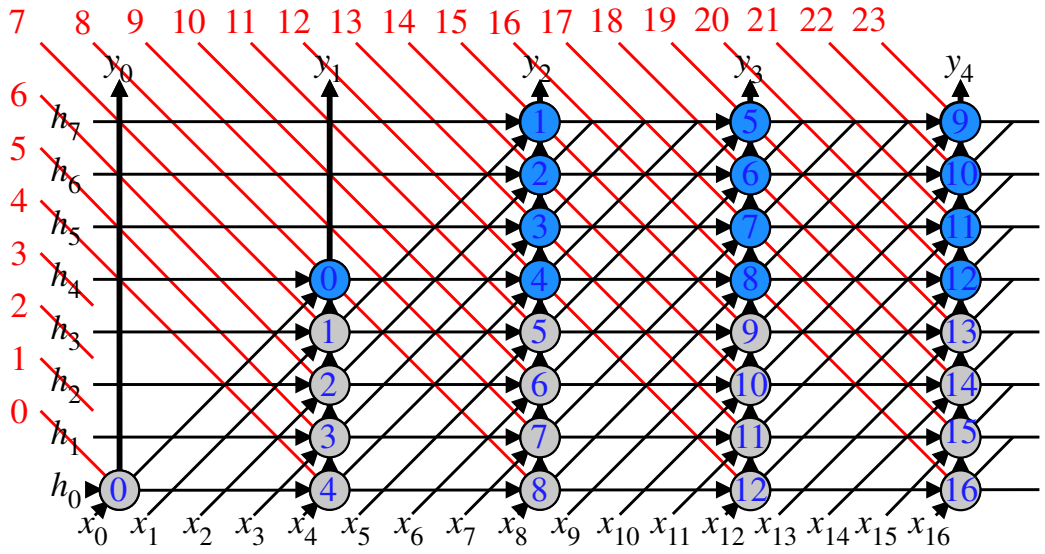


Figure 3.11: The DAG for Design-Option #5: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_2 = [1 \quad 1]^T$.

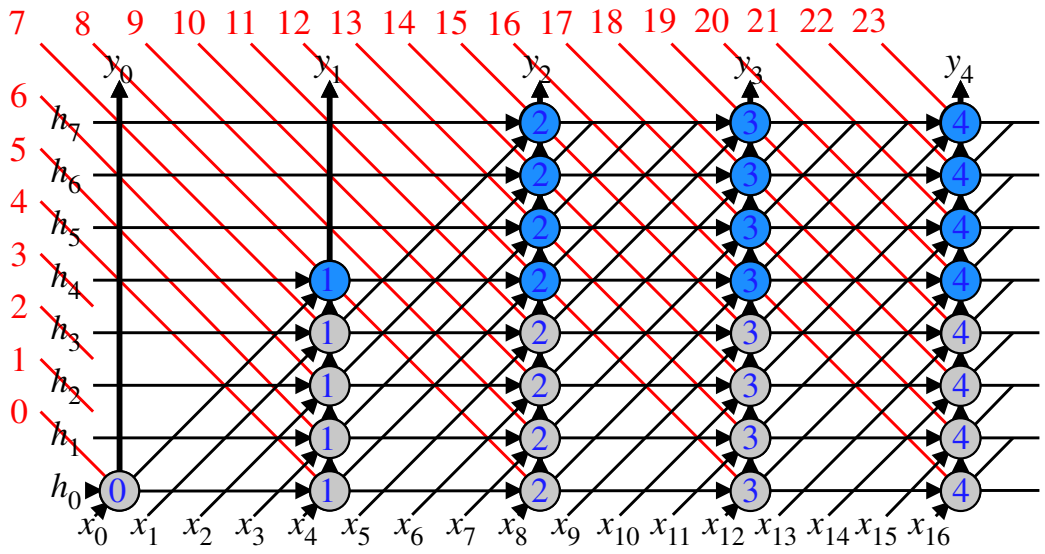


Figure 3.12: The DAG for Design-Option #6: using $\boldsymbol{s}_2 = [1 \quad 1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$.

Fig. 3.6 shows the proposed systolic array design. It can be seen from the Fig. 3.6 that the design requires broadcast of the input data and neighbour-to-neighbour communication for the intermediate data. This ensures the maximum processing speed.

3.7 Alternative decimator structures

There are two well known decimator design approaches. The first approach uses an antialiasing filter followed by a downsampler. We call this approach the conventional design. The second approach uses a polyphase decimator structure.

3.7.1 Conventional Design

The conventional design consists of an anti-imaging filter followed by an downsampler. Fig. 3.13 shows the systolic array design of the conventional decimator.

Fig. 3.13(a) shows the downsampler details which accepts the high rate data samples y_h and produces data samples y_l with M lower rate.

Fig. 3.13(b) shows the systolic array of the conventional FIR filter, which corresponds to a form 2 implementation of an FIR filter. The number of PEs is equal to the number of filter coefficients J. We assume the double precision mode of operation, where data exchanged between the PEs is in double precision format. This is

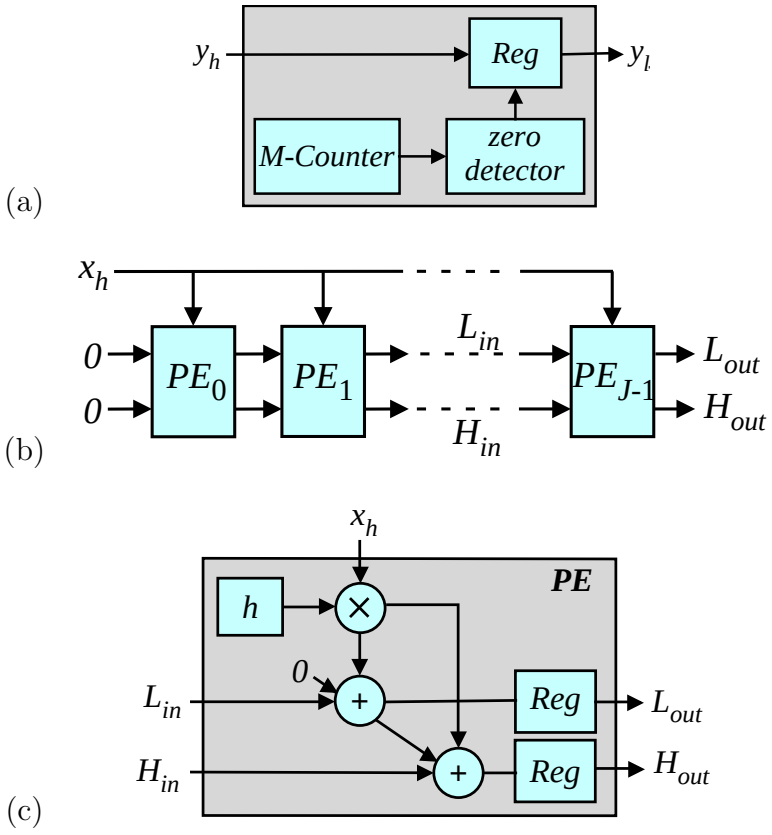


Figure 3.13: The conventional decimator implementation. (a) The downsampler details. (b) The systolic array of the conventional FIR filter. (c) The FIR filter PE details (Form 2).

explicitly shown in Fig. 3.13 as the partial sums L and H.

Fig. 3.13(c) shows the conventional FIR filter PE details. The PE consists of a traditional multiplier with double-precision output. The multiplier multiplies the input data samples x_h by the dedicated FIR filter coefficient stored in a register. The lower bits of the multiplier output are added to the lower bits of the previous PE output using a single-precision carry-propagate (SPCP) adder to produce the lower bits of the PE output. Similarly, the higher bits of the multiplier output are added to the higher bits of the previous PE output using a SPCP adder to produce the higher bits of the PE output.

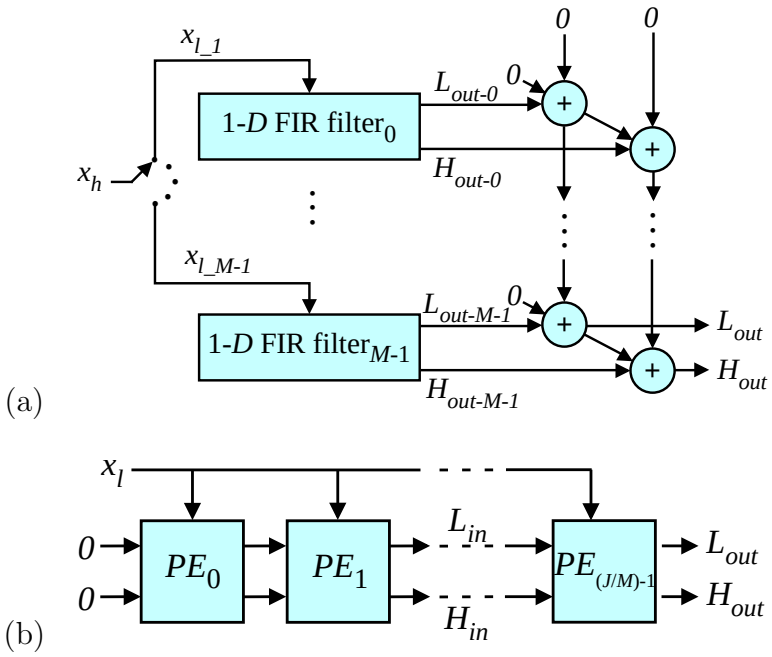


Figure 3.14: The double-precision polyphase decimator implementation. (a) The polyphase decimator structure. (b) The systolic array for the FIR filter section.

3.7.2 Polyphase Design

An alternative approach to implement the decimator is by using the polyphase structure [51]. In this approach the input data is applied to a commutator before the polyphase FIR filter structure. Fig. 3.14 shows the systolic array design of the polyphase decimator.

Fig. 3.14(a) shows the double-precision polyphase decimator structure. The FIR filter is decomposed into M components. Each component corresponds to a form 2 implementation of an FIR filter operating at the lower clock rate. We assume the double precision mode of operation.

Fig. 3.14(b) shows the systolic array for one of the FIR filter sections. This systolic array is similar to the systolic array shown in Fig. 3.13(b). However, the number of PEs is equal to J/M. We assume the double precision mode of operation, as in the conventional design, where data exchanged between the PEs is in double precision format. This is explicitly shown in Fig. 3.14 as the partial sums L and H.

The FIR filter PE details is the same as in the conventional design as shown in Fig. 3.13(c).

3.8 FPGA implementation results

This section compares the FPGA performance (area, speed, and power) of the three different approaches discussed in Section 3.7. There are several reasons for adopting this strategy:

1. Different FPGA devices and families are being used.
2. Variability in the design goals and strategies used (area reduction, timing performance, power optimization or balanced).
3. Reported results are sometimes unclear in terms of the number of filter coefficients or the decimation factors used.
4. In some of the published designs, the system sampling rate is different (lower) than the system clock speed. The latter is reported but not the former.

FPGA implementations of the conventional, polyphase and the proposed designs were implemented using VHDL on Xilinx Virtex 6 ML605 development board.

Several decimation factors were used to compare the performance among the three designs. The decimation factors used were 2, 4, 8, etc (D-2, D-4, D-8, etc). The number of antialiasing FIR filter coefficients were chosen as 8, 16, 32 and 64.

Tables 3.3–3.6 show the performance in terms of FPGA resources, speed, and power consumption for the three designs for decimation factors $M = 2^i$, with $1 \leq i \leq \log_2 J$; for $8 \leq J \leq 64$, respectively. As expected, the conventional design shows almost the same performance regardless of the decimation factor M. This is because the same FIR filter used in each decimator. The impact of the downsampler on the performance is minimal since a one-bit counter is needed for decimation by 2, a two-bit counter is needed for decimation by 4 and a three-bit counter is needed for decimation by 8.

The performance of the polyphase design depends on M. This is because the number of polyphase branches is directly proportional to M.

For the proposed design, the performance depends on M as in the polyphase design. This is because the number of PEs is inversely proportional to M.

Fig. 3.15 shows the speed results of Table 3.6. The speed of the conventional decimator is constant regardless of the value of M. The structure of the antialiasing filter in the conventional decimator is independent of M and this explains why the speed is not affected by the value of M.

Table 3.3: Performance comparison between conventional, polyphase and proposed decimators at different decimation factors $M = 2, 4$, and 8 for the case of number of FIR filter coefficients $J = 8$.

Decimation factor	2–8	2		4		8	
Decimator structure	Conv.	Poly.	Prop.	Poly.	Prop.	Poly.	Prop.
# Slice Registers	191	87	183	85	103	81	53
# Slice LUTs	352	151	331	150	218	247	117
# Fully used LUT-FF	131	85	102	84	55	67	36
Maximum Frequency (MHz)	135	299	232	164	212	106	149
Delay (ns)	7.4	3.3	4.3	6.1	4.7	9.4	5.6
Power (mW)	132	126	128	127	118	131	113

The speed of the polyphase decimator is better than the other two designs when $M = 2$. The speed decreases for increasing the value of M. The polyphase design becomes the slowest design for values of $M > 4$. The reduction in speed can be explained due to the extra adder delay since increasing M leads to increasing the number of the polyphase branches.

The speed of the proposed decimator decreases at a slow rate as M increases. This can be explained by the fact that the size of the distributed RAM that stores the PE filter coefficients increases as M increases.

For $M > 2$ the speed of the proposed design is higher than the other two designs.

Fig. 3.16 shows the power consumption for the three designs for decimation factors $M = 2^i$, with $1 \leq i \leq 6$; for $J = 64$. The power consumption of the conventional decimator is constant regardless of the value of M. The structure of the antialiasing filter in the conventional decimator is independent of M and this explains why the power is not affected by the value of M.

Table 3.4: Performance comparison between conventional, polyphase and proposed decimators at different decimation factors $M = 2, 4, 8$, and 16 for the case of number of FIR filter coefficients $J = 16$.

Decimation factor	2-16	2		4		8		16	
Decimator structure	Conv.	Poly.	Prop.	Poly.	Prop.	Poly.	Prop.	Poly.	Prop.
# Slice Registers	386	182	380	165	194	153	102	145	64
# Slice LUTs	722	335	698	277	489	345	242	601	158
# Fully used LUT-FF	259	179	179	164	102	126	57	119	37
Maximum Frequency (MHz)	135	293	209	173	186	104	182	72	178
Delay (ns)	7.4	3.4	4.8	5.8	5.4	9.6	5.5	13.9	5.6
Power (mW)	136	131	132	130	122	133	115	139	114

The power consumption of the polyphase decimator is better than the other two designs when $M = 2$. The power consumption is almost constant for all values of $2 < M \leq 64$. Increasing the values of M decreases the number of PEs in each branch, however the number of branches increases. Therefore the net power consumption is almost constant.

The power consumption of the proposed decimator decreases as M increases. This can be explained by the fact that the number of PEs decreases as M increases.

For $M > 4$ the power consumption of the proposed design is lower than the other two designs.

3.9 Conclusion

In this chapter we reviewed how a systematic methodology was used to find best data scheduling strategies and explore possible decimator structures. Nine possible

Table 3.5: Performance comparison between conventional, polyphase and proposed decimators at different decimation factors $M = 2, 4, 8, 16$, and 32 for the case of number of FIR filter coefficients $J = 32$.

Decimation factor	2-32	2		4		8		16		32	
Decimator structure	Conv.	Poly.	Prop.	Poly.	Prop.	Poly.	Prop.	Poly.	Prop.	Poly.	Prop.
# Slice Registers	787	415	770	349	389	316	204	289	109	274	63
# Slice LUTs	2230	1024	1540	533	871	601	498	804	280	1245	177
# Fully used LUT-FF	515	414	394	341	198	282	106	191	57	223	38
Maximum Frequency (MHz)	135	275	194	158	181	102	178	69	179	45	175
Delay (ns)	7.4	3.6	5.1	6.3	5.5	9.7	5.5	14.6	5.6	22.4	5.7
Power (mW)	148	140	147	134	129	137	120	141	114	139	111

designs which satisfy input data timing were possible. Out of those six structures were promising in terms of fast clock speed due to output data pipelining. One design, which had the least area and power requirements out of these six, was selected. A detailed discussion was provided for two well-known decimator designs; namely conventional design in which the antialiasing filter is followed by a downsample and the polyphase design in which a commutator is followed by the polyphase antialiasing filter. FPGA implementations are performed for the proposed design as well as existing well-known decimator structures to verify their functionality and compare their performance. FPGA implementation results confirm that the proposed decimator implementation outperforms in terms of area, speed, and power as the decimation factor increases regardless of the number of FIR filter coefficients.

Table 3.6: Performance comparison between conventional, polyphase and proposed decimators at different decimation factors $M = 2, 4, 8, 16, 32$, and 64 for the case of number of FIR filter coefficients $J = 64$.

Decimation factor	2-64	2		4		8		16		32		64	
Decimator structure	Conv.	Poly.	Prop.	Poly.	Prop.	Poly.	Prop.	Poly.	Prop.	Poly.	Prop.	Poly.	Prop.
# Slice Registers	1724	938	1537	820	770	686	393	612	213	625	118	600	54
# Slice LUTs	4362	2518	3422	1185	1758	1099	1057	1287	617	2140	321	2636	195
# Fully used LUT-FF	1028	931	794	810	392	656	202	503	105	254	58	473	39
Maximum Frequency (MHz)	131	308	193	158	178	100	170	71	173	52	166	24	159
Delay (ns)	7.7	3.2	5.2	6.3	5.6	10.0	5.9	14.0	5.8	19.4	6.0	41.0	6.3
Power (mW)	181	156	177	143	145	144	130	148	120	151	114	154	108

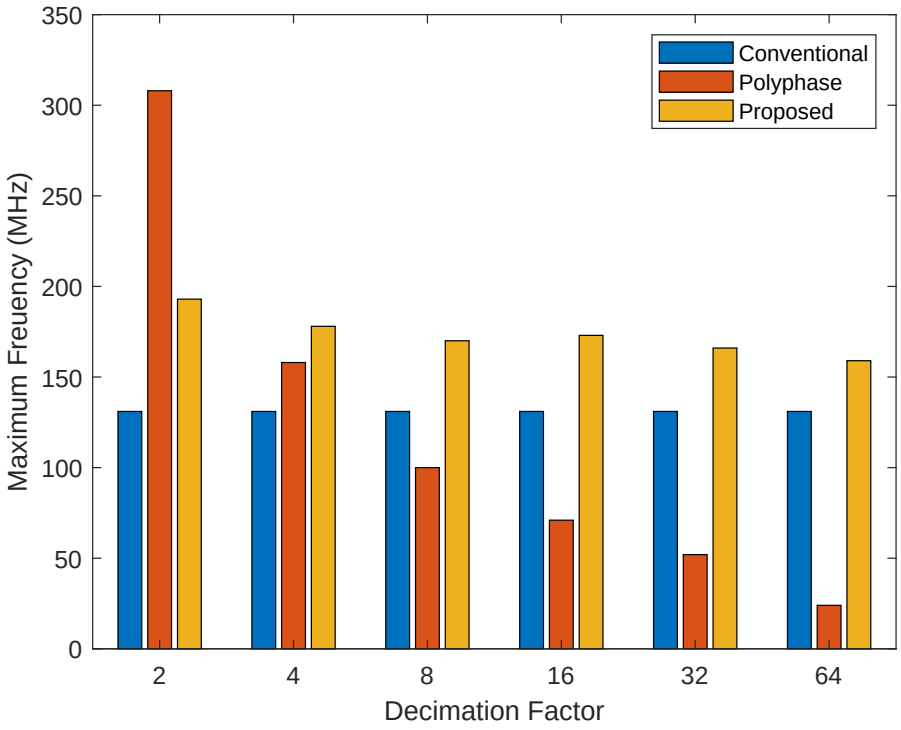


Figure 3.15: Maximum frequency comparison chart between different filter structures at different decimation factors for $J = 64$.

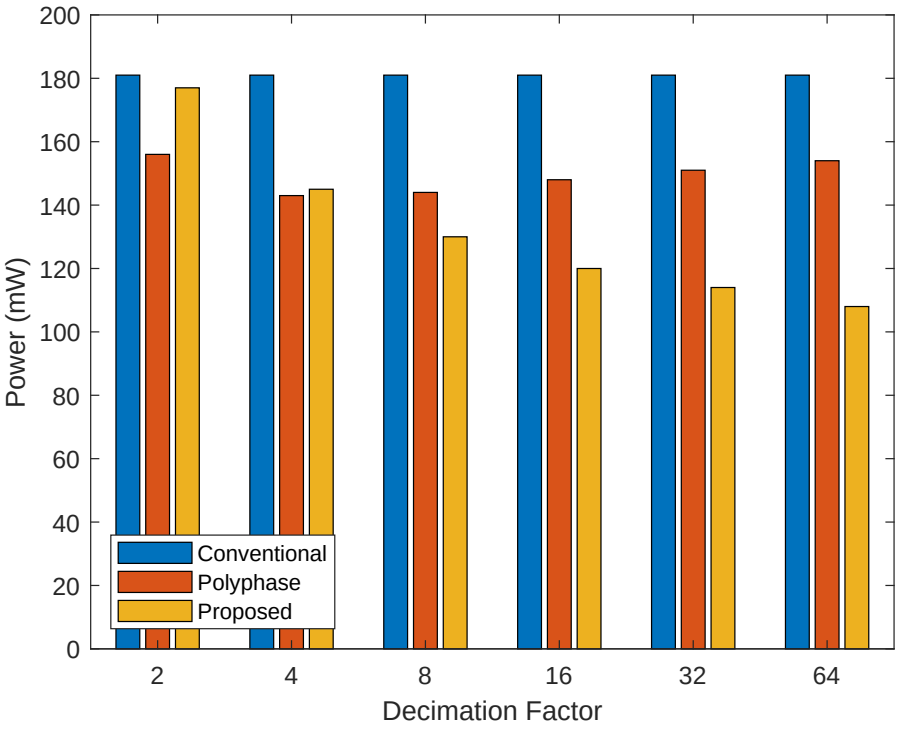


Figure 3.16: Power consumption comparison chart between different filter structures at different decimation factors for $J = 64$.

Chapter 4

Design Space Exploration of 2-D Beamforming Filter

This chapter presents the development and comparison of two dimensional (2-D) broadband (BB) beamforming (BF) filter processor array designs and implementations. A systematic methodology was applied to the difference equations defining the input/output (I/O) relations of the 2-D BB BF filter. This systematic methodology is based on finding a computational domain of the iterative algorithm for the BB BF filter and exploring different task scheduling and projection options to obtain a processor array designs. Six such designs were obtained, implemented and evaluated. The multiply/accumulate (MAC) operation of the resulting designs could be implemented using either distributed arithmetic (DA) technique or parallel multipliers. The analysis and comparison between DA based and multiplier based (conventional - proposed) MAC implementations are performed. FPGA implementations for the conventional design using conventional MAC and the six proposed designs using proposed MAC confirm that four of the proposed designs produced a speedup factor of 3.48 compared to conventional designs.

4.1 Introduction and Related Work

Broadband (BB) beamforming (BF) has many applications such as radar and navigation, wireless communications, radio astronomy, microwave imaging and microphone arrays [52, 53, 54, 55, 56, 57, 58, 59, 60]. BF algorithms may be classified as fixed/adaptive and narrowband/broadband [53, 54, 19, 5]. An important class of

fixed broadband beamformers are designed using multi-dimensional filters such as: 2-D filters having passbands of (beam [61, 24], fan [62, 63], and trapezoidal [64, 65]) shapes for one-dimensional (1-D) antenna arrays, and three-dimensional (3-D) filters having passbands of (cone [66, 67], and frustum [68, 69]) shapes for 2-D antenna arrays.

Hardware implementation of BB BF filters have been presented in the literature. In [70], Ariyarathna et al. presents the implementation of a 2-D FIR trapezoidal filter based beamforming array receiver. The 2-D FIR trapezoidal filter is designed and implemented on FPGA as a filter-and-sum architecture. In [71], Kumar et al. presented an efficient architecture for 2-D FIR filter for block processing using DA formulation in order to eliminate the use of multipliers in the design. In [72], Teja and Hema presented a systematic design strategy to derive area-delay-power-efficient architectures for two-dimensional finite impulse response filter based on the analyzed memory footprint and combinational complexity. In [73], Sravan et al. presents a receive-mode digital-RF beamformer based on spatially bandpass (SBP) 2nd order 2-D IIR beam filters. The 2-D IIR SBP beam filter is implemented as a systolic array of several parallel core processing modules. In [74], Kumar et al. proposed two hardware reconfigurable structures for the 2-D IIR filters based on DA techniques. Although the implementations of the 2-D FIR filters exist in the literature, there is still interest in exploring the possibilities for efficient hardware implementations using systematic approaches.

In this chapter, the processor array implementation of fixed BB BF based on 2-D FIR filters is considered. The 2-D BB BF filter design space exploration is performed using a systematic technique that was proposed in [6]. This systematic technique allows us to merge all the iterations of the 2-D BB BF filter in a unified computational domain (CD). Six possible designs will be presented. FPGA implementations of these designs were done to verify the functionality and evaluate the performance of the proposed designs. The MAC operation of the resulting designs could be implemented using either DA technique or parallel multipliers. A comparison is performed between DA based and multiplier based MAC implementations. Compared to the conventional design, the proposed designs are faster at the cost of increasing the required hardware resources.

4.2 Preliminaries

The 2-D FIR BB BF filter I/O relations is given by:

$$y(n_t) = \sum_{i=-(I-1)/2}^{(I-1)/2} \sum_{j=0}^{J-1} h(i,j)x(i, n_t - j) \tag{4.1}$$

where $y(n_t)$ represents the output samples, $h(i,j)$ represents the filter coefficients, and $x(i, n_t - j)$ represents the input samples from the sensor arrays. We assume a right-sided output samples sequences, which implies $n_t \geq 0$. The number of the 2-D BB filter coefficients is $I \times J$ where I is the number of sensors in the sensor array.

In order to perform processor array design space exploration of the 2-D BB BF filter, we use a systematic methodology that was proposed in [6] for regular iterative algorithms (RIAs). The methodology specifies several steps which are adapted here for the 2-D BB BF filtering algorithm as follows:

4.2.1 Algorithm Computational Domain

The difference equation of the 2-D BB BF filter, expressed as an RIA, are shown in Eq. (4.1). The ranges of the indices in Eq. (4.1) define the computational domain $\mathcal{D} \subset \mathcal{Z}^n$ of the algorithm. Since Eq.(4.1) has three indices n_t, i and j, the algorithm is defined in the 3-dimensional integer domain $\mathcal{Z}^3$. Section 4.3 discusses the computational domain of the 2-D BB BF filter.

4.2.2 Index Dependence of the Algorithm Variables

Equation (4.1) defines one output variable $y(n_t)$ and two input variables $h(i,j)$ and $x(i, n_t - j)$. The algorithm variables could be broadcast or pipelined in the resulting processor array. A broadcast input variable implies that the input variable is distributed throughout the processor array using a broadcast bus, where a given input sample is supplied to all the processing elements (PEs) at the same time. Further, a broadcast output variable implies that all the computations needed to produce a certain output sample are done at the same time instance. On the other hand, a pipelined input or output variable implies that the computations needed are done over consecutive time instances.

Each instance of the variable is associated with a set of points in a computational

subdomain that is a subset of the $\mathcal{D}$. This subdomain is defined through the index dependence and the index range of a variable using the dependence matrix $\boldsymbol{A}$, see Sec. 11.5 in [6]. The rank deficiency of $\boldsymbol{A}$ define the dimensionality of the computational subdomain. The nullvectors of $\boldsymbol{A}$ form the basis vectors describing the coordinates of the points in the computational subdomain.

Each sample of a broadcast/pipelined input variable is associated with a feeding point through which the data is supplied to the processor array. Each sample of a broadcast output variable is associated with an extraction point from which the data is obtained from the processor array. Each sample of a pipelined output variable is associated with a feeding point through which the pipelined is initialized. The pipelined output variable is also associated with an extraction point from which the data is obtained from the processor array.

For each sample of the input or output variable the feeding or extraction point is located at one of the corners of the associated subdomain.

Section 4.3 explains the dependence of the 2-D BB BF filter variables.

4.2.3 Task Scheduling

Task scheduling describes the time sequence of the operations associated with every point in $\mathcal{D}$. Valid scheduling functions must satisfy any input and output data timing specifications and constrains. An affine scheduling function is defined through a scheduling row vector $\boldsymbol{s}$. The inner product of $\boldsymbol{s}$ and the null vectors of a variable determine whether the variable will be pipelines or broadcast. Section 4.5 provides exploration of scheduling functions for the 2-D BB BF filter algorithm.

4.2.4 Point Projection

A point projection maps each point in $\mathcal{D}$ to a processing element in the resulting processor array. Point projection is effected through a projection matrix $\boldsymbol{P}$ which is defined by the projection direction $\boldsymbol{d}$. Section 4.6 provides exploration of projection directions associated with valid scheduling functions for the 2-D BB BF filter algorithm.

The term processor array design in the rest of the paper will be used to describe the architecture and the functionality of a processor array while processor array implementation represents the actual implementation of this design in FPGA. Section 4.6 provides exploration of projection directions for the 2-D BB BF filter algorithm.

4.3 Computational Domain (CD) of the 2-D BB BF Filter

The dimensionality n of $\mathcal{D}$ is determined from Eq. (4.1). Since the algorithm depends on three indices n_t, i and j, we can write $\mathcal{D} \subset \mathcal{Z}^3$, where $\mathcal{Z}^3$ is the 3-D integer space.

The algorithm indices are organized in the form of a vector as:

$$\boldsymbol{p} = [n_t \quad i \quad j]^T \tag{4.2}$$

where T means vector transpose.

The boundaries of the 3-D space describing our algorithm are defined by the restrictions imposed on the values of the indices as: $n_t \geq 0$, $-(I-1)/2 \leq i \leq (I-1)/2$, and $0 \leq j \leq J-1$.

$\mathcal{D}$ can be represented as a convex hull with lower and upper hulls defining its boundaries. The lower and upper hulls represent the starting and ending values of the filtering indices respectively.

The lower hull is obtained from Eq. (4.1) using the inequalities:

$$[1 \quad 0 \quad 0]\boldsymbol{p} \geq 0 \tag{4.3}$$

$$[0 \quad 1 \quad 0]\boldsymbol{p} \geq -(I-1)/2 \tag{4.4}$$

$$[0 \quad 0 \quad 1]\boldsymbol{p} \geq 0 \tag{4.5}$$

The above three inequalities simply state that the lower bound on points in the CD is described by the equations of planes defining the bottom surfaces of the CD as: $n_t \geq 0$, $i \geq -(I-1)/2$, and $j \geq 0$,

From Eq. (4.1), the upper hull of the algorithm is described by:

$$[0 \quad 1 \quad 0]\boldsymbol{p} \leq (I-1)/2 \tag{4.6}$$

$$[0 \quad 0 \quad 1]\boldsymbol{p} \leq J-1 \tag{4.7}$$

The above two inequalities state that the upper bound on points in the CD is described by the equations of planes defining the top surfaces of the CD as: $i \leq (I-1)/2$, and $j \leq J-1$. The lack of an inequality for the index n_t indicates that the upper bound in this direction extends to infinity, i.e., the CD contain a ray in the n_t-direction according to [6].

4.4 Dependence Matrices of the 2-D BB BF Filter Variables

There are two input variables $h(i,j)$ and $x(i,n_t-j)$ and one output variable $y(n_t)$. For the output variable y, the dependence matrix can be given as:

$$\boldsymbol{A}_y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{4.8}$$

For the input variable h:

$$\boldsymbol{A}_h = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

For the input variable x:

$$\boldsymbol{A}_x = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \tag{4.10}$$

The dependence matrices $\boldsymbol{A}_y$, $\boldsymbol{A}_h$, and $\boldsymbol{A}_x$ are rank deficient matrices. Since the rank of $\boldsymbol{A}_y$ is equal to 1, then the number of independent nullvectors associated with it is 2. The rank of $\boldsymbol{A}_h$, and $\boldsymbol{A}_x$ are equal to 2, and there is only one independent nullvector associated with each one of them.

4.4.1 Nullvectors of the Dependence Matrices

This subsection will discuss the nullvectors associated with the dependence matrices of the algorithm variables. These nullvectors will help us perform the design space exploration of the desired systolic arrays.

Figure 4.1 illustrates all the points $\boldsymbol{p} \in \mathcal{D}$ that use the instance $h(c_1,c_2)$ of the input variable $h(i,j)$. This set of points define a subdomain $\mathcal{D}_h \subset \mathcal{D}$. All the points $\boldsymbol{p} \in \mathcal{D}_h$ can be described using the basis vector $e_h = [1 \quad 0 \quad 0]^T$ of the subdomain $\mathcal{D}_h$. We notice that $\boldsymbol{e}_h$ is the nullvector of the dependence matrix $\boldsymbol{A}_h$.

The nullvectors associated with the variable y are obtained using the dependence matrix $\boldsymbol{A}_y$ and can be given as:

$$\boldsymbol{e}_{y1} = [0 \quad 1 \quad 0]^T \tag{4.11}$$

$$\boldsymbol{e}_{y2} = [0 \quad 0 \quad 1]^T \tag{4.12}$$

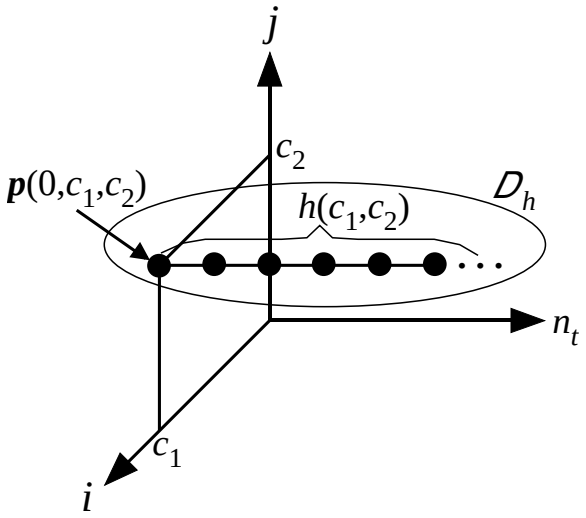


Figure 4.1: Subdomain $\mathcal{D}_h$ for the input variable instance $h(c_1, c_2)$.

Similarly the nullvector associated with the variable h is:

$$\boldsymbol{e}_h = [1 \quad 0 \quad 0]^T \tag{4.13}$$

The nullvector associated with the variable x is:

$$\boldsymbol{e}_x = [1 \quad 0 \quad 1]^T \tag{4.14}$$

Section 4.5 will explains how these nullvectors are used to explore valid scheduling schemes for executing our algorithm.

4.4.2 Feeding and Extraction Points of the 2-D BB BF Filter Variables

Given the algorithm computation domain $\mathcal{D}$, we need to find points where the input variables can be supplied. These are the feeding points. Similarly, we need to find points in $\mathcal{D}$ from which the output of the algorithm can be extracted. These are the extraction points.

For example, the input variable instance $h(c_1, c_2)$ is supplied to the algorithm at point $\boldsymbol{p}(0, c_1, c_2) \in \mathcal{D}$ as shown in Fig. 4.1, which defines the feeding point for the instance $h(c_1, c_2)$. The point $\boldsymbol{p}(0, c_1, c_2)$ was chosen because it lies on the boundary

of $\mathcal{D}_h$ and intersects with the lower hull of $\mathcal{D}$. Similarly, the input variable instance $x(c_1, c_2)$ is supplied to the algorithm at point $\boldsymbol{p} \in \mathcal{D}$, which is the intersection between $\mathcal{D}_x$ and the lower hull of $\mathcal{D}$. On the other hand, the output variable instance $y(c_1)$ is extracted from the algorithm at a point $\boldsymbol{p} \in \mathcal{D}$, which is the intersection between $\mathcal{D}_y$ and the upper or lower hulls of $\mathcal{D}$.

Mathematically, in order to find the feeding/extraction points, that lie on the upper/lower hulls of $\mathcal{D}$, the dependence matrices of each variable should be augmented to make them full rank. The dependence matrices are augmented using the basis vectors of Eqs. (4.3) - (4.7) that describe the lower and upper hulls.

For the ouput variable y, in order to find the feeding/extraction point on the lower hull, we augment the dependence matrix $\boldsymbol{A}_y$ with the vectors from (4.3)-(4.5). This makes $\boldsymbol{A}_y$ full rank. The augmented matrix $\boldsymbol{A}_{y,aug}$ becomes:

$$\boldsymbol{A}_{y,aug} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

The feeding/extraction point $\boldsymbol{p}_{1y,i/o}$ at the lower hull is determined from the linear equation:

$$\boldsymbol{A}_{y,aug} \;\; \boldsymbol{p}_{1y,i/o} \;\; = \;\; [c_1 \quad -(I-1)/2 \quad 0]^T \tag{4.16}$$

which gives:

$$\boldsymbol{p}_{1y,i/o} \;\; = \;\; [c_1 \quad -(I-1)/2 \quad 0]^T \tag{4.17}$$

since $\boldsymbol{A}_{y,aug}$ is a unity matrix.

In a similar fashion, to find the feeding/extraction point of the variable y on the upper hull, we augment the dependence matrix $\boldsymbol{A}_y$ with the vectors of (4.6) and (4.7). The resulting augmented matrix $\boldsymbol{A}_{y,aug}$ is the same as in Eq. (4.15). The feeding/extraction point at the upper hull is determined from the set of linear equations:

$$\boldsymbol{A}_{y,aug} \;\; \boldsymbol{p}_{2y,i/o} \;\; = \;\; [c_1 \quad (I-1)/2 \quad J-1]^T \tag{4.18}$$

which gives:

$$\boldsymbol{p}_{2y,i/o} = [c_1 \quad (I-1)/2 \quad J-1]^T \tag{4.19}$$

For the input variable h, the augmented matrix of the dependence matrix $\boldsymbol{A}_h$ is given by:

$$\boldsymbol{A}_{h,aug} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{4.20}$$

Because h is an input variable, we are interested only in the feeding point $\boldsymbol{p}_{h,in}$ on the lower hull. The set of linear equations will be:

$$\boldsymbol{A}_{h,aug} \ \boldsymbol{p}_{h,in} = [c_1 \quad c_2 \quad 0]^T \tag{4.21}$$

which gives:

$$\boldsymbol{p}_{h,in} = [0 \quad c_1 \quad c_2]^T \tag{4.22}$$

Similarly, for the input variable x, the augmented matrix of the dependence matrix $\boldsymbol{A}_x$ is given by:

$$\boldsymbol{A}_{x,aug} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.23}$$

Also, we are interested only in the feeding point $\boldsymbol{p}_{x,in}$ because x is an input variable. The set of linear equations will be:

$$\boldsymbol{A}_{x,aug} \ \boldsymbol{p}_{x,in} = [c_1 \quad c_2 \quad 0]^T \tag{4.24}$$

which gives:

$$\boldsymbol{p}_{x,in} = [c_2 \quad c_1 \quad 0]^T \tag{4.25}$$

4.5 Scheduling Functions for the 2-D BB BF Filter

The scheduling function divides the tasks in the algorithm into stages. At each stage, a group of tasks gets executed in parallel so that the final result is correct. All these tasks are assigned the same time value.

Broadcasting an algorithm variable results in assigning all points in its broadcast subdomain the same time value. Points in the subdomain of a pipelined variable, on the other hand, are assigned different time values. For a pipelined output variable, a partial result that is generated by each PE is accumulated and passed to the next PE until the final result is accumulated by the last PE. One simple scheduling function that is used to schedule the computation tasks is the linear scheduling function [6]:

$$t(\boldsymbol{p}) = \boldsymbol{s}\boldsymbol{p} - s \tag{4.26}$$

where $t(\boldsymbol{p})$ is the time value assigned to a point $\boldsymbol{p}$ in $\mathcal{D}$, $\boldsymbol{s} = [\alpha \quad \beta \quad \gamma]$ is the schedule row vector, and s is a scalar constant. To broadcast an algorithm variable whose nullvector is $\boldsymbol{e}$, we must have [6]:

$$\boldsymbol{s}\boldsymbol{e} = 0 \tag{4.27}$$

and to pipeline this variable, we must have:

$$\boldsymbol{s}\boldsymbol{e} \neq 0 \tag{4.28}$$

Conditions in (4.27) and (4.28) are the minimum constrains that can be used to get a valid scheduling function.

Since broadcast output variable would result in a slower system that requires gathering all the partial outputs and somehow using them to produce the output value, we chose to pipeline the output variable. So, the valid scheduling vector should satisfy (4.28). By multiplying a scheduling vector $\boldsymbol{s}$ by the null vectors of the output variable y, defined by Eqs. (4.11) and (4.12), we obtain:

$$\boldsymbol{s}\boldsymbol{e}_{y1} \neq 0 \tag{4.29}$$

$$\boldsymbol{s}\boldsymbol{e}_{y2} \neq 0 \tag{4.30}$$

which leads to β and γ being $\neq 0$.

For the output variable y we have the condition that it should be obtained from the algorithm at consecutive time steps. For the two adjacent samples in n_t-direction, we have: $\boldsymbol{p}_1 = [n_t \quad i \quad j]^T$ and $\boldsymbol{p}_2 = [n_t + 1 \quad i \quad j]^T$

$$t(\boldsymbol{p}_2) = t(\boldsymbol{p}_1) + 1 \tag{4.31}$$

$$\boldsymbol{s}\boldsymbol{p}_2 = \boldsymbol{s}\boldsymbol{p}_1 + 1 \tag{4.32}$$

By substituting $\boldsymbol{s}$ in Eq. (4.32), we get $\alpha = 1$.

The scheduling vector so far is:

$$\boldsymbol{s} = [1 \quad \beta \quad \gamma] \tag{4.33}$$

Each output sample in Eq. (4.1) is evaluated over two indices i and j. We can choose to pipeline or broadcast the calculations over each index independently of the other index.

Since our objective is to achieve the fastest possible system clock, a fully pipelined output is the best choice. Fully pipelined output means pipeline the output partial calculation over both i and j indices. Therefore we have only one option which is $\alpha = 1$, $\beta \neq 0$ and $\gamma \neq 0$. In the following subsections we can identify two scheduling options.

4.5.1 Scheduling Option #1

In this design option, we choose to pipeline in the i-direction first then in the j-direction.

For pipelining in the positive i-direction, the calculations associated the two adjacent points $\boldsymbol{p}_a = [n_t \quad i \quad j]^T$ and $\boldsymbol{p}_b = [n_t \quad i+1 \quad j]^T$ in Fig. 4.2 should be performed in two consecutive time steps:

$$t(\boldsymbol{p}_b) = t(\boldsymbol{p}_a) + 1 \tag{4.34}$$

$$\boldsymbol{s}\boldsymbol{p}_b = \boldsymbol{s}\boldsymbol{p}_a + 1 \tag{4.35}$$

By substituting Eq. (4.33) in Eq. (4.35), we get $\beta = 1$. Choosing to pipeline the output in the negative i-directions leads to identical results and will not be pursued further in this work.

For pipelining in the positive j-direction, the calculations associated the two ad-

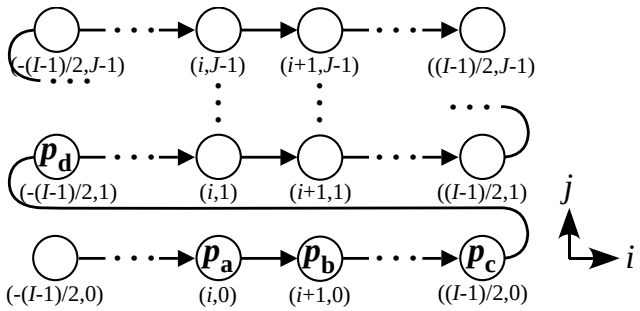


Figure 4.2: Pipelining in the i-direction first then in the j-direction.

jacent points $\boldsymbol{p}_c = [n_t \quad (I-1)/2 \quad j]^T$ and $\boldsymbol{p}_d = [n_t \quad -(I-1)/2 \quad j+1]^T$ in Fig. 4.2 should be performed in two consecutive time steps:

$$t(\boldsymbol{p}_d) = t(\boldsymbol{p}_c) + 1 \tag{4.36}$$

$$\boldsymbol{s}\boldsymbol{p}_d = \boldsymbol{s}\boldsymbol{p}_c + 1 \tag{4.37}$$

By substituting Eq. (4.33) with $\beta = 1$ in Eq. (4.37), we get $\gamma = I$. Choosing to pipeline the output in the negative j-directions leads to identical results and will not be pursued further in this work.

The scheduling vector that effects our pipelining choice is given by:

$$\boldsymbol{s}_1 = [1 \quad 1 \quad I] \tag{4.38}$$

4.5.2 Scheduling Option #2

In this design option, we choose to pipeline in the j-direction first then in the i-direction.

For pipelining in the positive j-direction, the calculations associated the two adjacent points $\boldsymbol{p}_a = [n_t \quad i \quad j]^T$ and $\boldsymbol{p}_b = [n_t \quad i \quad j+1]^T$ in Fig. 4.3 should be performed in two consecutive time steps:

$$t(\boldsymbol{p}_b) = t(\boldsymbol{p}_a) + 1 \tag{4.39}$$

$$\boldsymbol{s}\boldsymbol{p}_b = \boldsymbol{s}\boldsymbol{p}_a + 1 \tag{4.40}$$

By substituting Eq. (4.33) in Eq. (4.40), we get $\gamma = 1$. Choosing to pipeline the output in the negative j-directions leads to identical results and will not be pursued

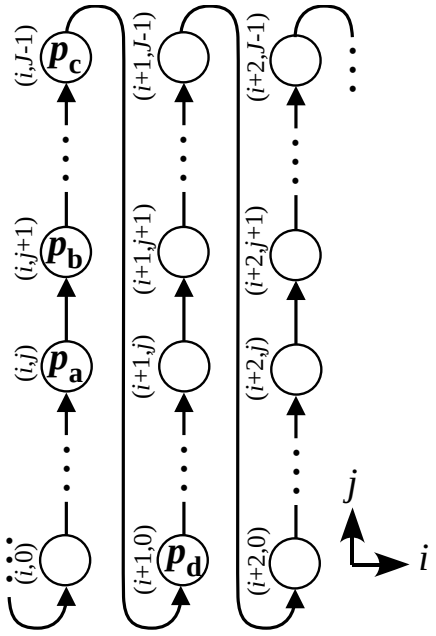


Figure 4.3: Pipelining in the j-direction first then in the i-direction.

further in this work.

For pipelining in the positive i-direction, the calculations associated the two adjacent points $\boldsymbol{p}_c = [n_t \quad i \quad J-1]^T$ and $\boldsymbol{p}_d = [n_t \quad i+1 \quad 0]^T$ in Fig. 4.3 should be performed in two consecutive time steps:

$$t(\boldsymbol{p}_d) = t(\boldsymbol{p}_c) + 1 \tag{4.41}$$

$$\boldsymbol{s}\boldsymbol{p}_d = \boldsymbol{s}\boldsymbol{p}_c + 1 \tag{4.42}$$

By substituting Eq. (4.33) with $\gamma = 1$ in Eq. (4.42), we get $\beta = J$. Choosing to pipeline the output in the negative i-directions leads to identical results and will not be pursued further in this work.

The scheduling vector that effects our pipelining choice is given by:

$$\boldsymbol{s}_2 = [1 \quad J \quad 1] \tag{4.43}$$

4.6 Point Projection for the 2-D BB BF Filter

The linear projection operation is the projection matrix $\boldsymbol{P}$ that maps a point in the domain of dimension n to point $\boldsymbol{p'}$ in the k-dimensional computational domain, where

$k \leq n$ [6], according to the equation:

$$\boldsymbol{p'} = \boldsymbol{P}\boldsymbol{p} \tag{4.44}$$

Where $\boldsymbol{p'}$ is the new linear projected point that produces the processor index associated with point $\boldsymbol{p}$. The projection matrix $\boldsymbol{P}$ is determined by the associated projection direction $\boldsymbol{d}$, which is the nullspace of the matrix.

According to [6], the selection of the valid projection directions should satisfy the following conditions:

First, they should satisfy:

$$\boldsymbol{s}\boldsymbol{d}_i \neq 0 \tag{4.45}$$

where $\boldsymbol{s}$ is the chosen scheduling vector. In this work, it is aimed to map the points in the 3-D CD to a 1-D domain.

Second, they should be perpendicular to each other.

Further, to simplify the node projection analysis they should ly in the principle direction of the CD.

Applying the previous conditions to the valid scheduling vectors $\boldsymbol{s}_1$, and $\boldsymbol{s}_2$ (Eqs. (4.38) and (4.43)), Three valid projection directions are obtained:

$$\boldsymbol{d}_a = [1 \quad 0 \quad 0]^T \tag{4.46}$$

$$\boldsymbol{d}_b = [0 \quad 1 \quad 0]^T \tag{4.47}$$

$$\boldsymbol{d}_c = [0 \quad 0 \quad 1]^T \tag{4.48}$$

These projection directions are then used to calculate the associated projection matrix according to the procedure described in [6]. Given a project directions $\boldsymbol{d}_a$, $\boldsymbol{d}_b$, and $\boldsymbol{d}_c$. and knowing that the projection matrix $\boldsymbol{P}$ is the nullvector of each pair of these projection directions [6], we get only one valid projection matrix for each pair as:

$$\boldsymbol{P}_{ab} = [0 \quad 0 \quad 1] \tag{4.49}$$

$$\boldsymbol{P}_{ac} = [0 \quad 1 \quad 0] \tag{4.50}$$

$$\boldsymbol{P}_{bc} = [1 \quad 0 \quad 0] \tag{4.51}$$

By substituting Eqs. (4.49), (4.50), and (4.51) into Eq. (4.44), we get that $p' = j$, $p' = i$, and $p' = n_t$, respectively. which means that all operations will be performed

on the j-direction, i-direction, and n_t-direction, respectively.

Since, n_t have a large values compared to i and j values, using projection matrix $\boldsymbol{P}_{bc}$ leads to a large number of PEs that required to perform the operations. So, the projection matrix $\boldsymbol{P}_{bc}$ will be excluded from the design space exploration.

This projection matrices $\boldsymbol{P}_{ab}$ and $\boldsymbol{P}_{ac}$ have an effect on the data variables directions as follows:

The pipeline/broadcast direction for the input/output data variables is mapped to the point e' given by:

$$e' = \boldsymbol{P}\boldsymbol{e} \tag{4.52}$$

4.7 Comparing Multiplier Based and Distributed Arithmetic Based multiplier/accumulator

Each point in the projected domain obtained in Sec. 4.6 represents a MAC operation

$$d = a \times b + c \tag{4.53}$$

The highest sampling rate for the 2-D BB BF filter designs are obtained when the filter output is pipelined according to Sec. 4.5. This results in a system clock speed determined by the delay of the operations performed in each point in the projected domain. Therefore the system speed is determined by estimating the delay of the basic MAC operation.

There are three options to implement the MAC that performs the operation in the above equation (the conventional, DA based, and proposed MAC implementation). The conventional and the proposed MAC are multiplier based and the DA based one is multiplier-less. The conventional way to implement the multiplier based one which is MAC uses a multiplier followed by an accumulator. In the DA based MAC, lookup tables (LUTs) followed by adders are used to eliminate the multipliers in the hardware implementations.

In this work we use a new multiplier based MAC implementation that uses the merged MAC proposed in [50]. The merged MAC is a high-speed, double-precision carry-save multiplier/accumulator (CS-MAC) that is approximately 4 times faster than the conventional MAC. The CS-MAC multiplies two numbers and produces a double precision result in carry-save format. The accumulation operation is performed

in double precision and is merged, and concurrent, with the multiply operation. This can be expressed as

$$2^w (s_{out} + c_{out}) + l_{out} = a \times b + 2^w (s_{in} + c_{in}) + l_{in} \tag{4.54}$$

where s_{out}, c_{out} are the high-order sum and carry word output from the CS-MAC, l_{out} is the low order word output from the CS-MAC, s_{in}, c_{in} are the high-order sum and carry word input to the CS-MAC, and l_{in} is the low order word input to the CS-MAC. A reduction in the computation time ranging from 20% to 50% compared with other schemes has been achieved using the CS-MAC, without a significant increase in the required area [50].

To obtain expressions for the delays of the conventional, the DA based, and the proposed 2-D BB BF filter designs, we assume that the data size is W bits. The delay of the MAC operation in Eq. (4.53) depends on the details of the implementation as discussed in the following paragraphs.

The conventional MAC implementation uses $W \times W$ parallel multipliers followed by a double precision accumulator. Assuming T is the delay of a 1-bit adder, the normalized delay associated with the conventional MAC implementation is given by:

$$T_{conv} = 4W \tag{4.55}$$

The value of T is technology and implementation dependent. As an example, T can be estimated from the maximum frequency of the Xilinx adder IP core [75].

The DA based MAC implementation typically replace the combinational logic multipliers with lookup tables (LUT) in the form of random access memory (RAM) or read only memory (ROM) modules. The normalized delay associated with the DA based MAC implementation is given by:

$$T_{DA} = 2W + T_{mem}/T \tag{4.56}$$

$$= 2W + R \tag{4.57}$$

where the first term on the right hand side represents the normalized double-precision adder delay and the second term, T_{mem} is the read access delay of the memory and R is the ratio of the memory access delay relative to one bit adder delay. As an example, T_{mem} can be estimated from the performance of Xilinx block memory generator data sheet [76].

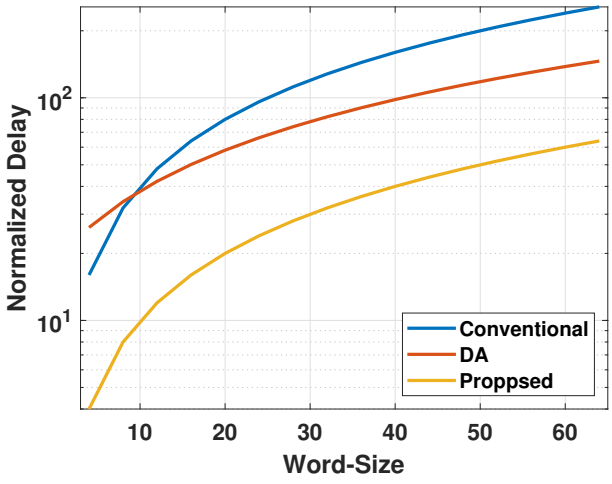


Figure 4.4: The normalized delay associated with the conventional, DA based, and proposed MAC implementation at different word-sizes for Xilinx Virtex-7 FPGAs.

The proposed MAC implementation uses $W \times W$ merged MACs. The normalized delay associated with the proposed MAC implementation is given by:

$$T_{prop} = W \tag{4.58}$$

Based on Eqs. (4.55), (4.56), and (4.58), the normalized delay associated with the conventional, DA based, and proposed MAC implementations at different word sizes are shown in Fig. 4.4. The values of T, T_{mem}, and R were estimated for Xilinx Virtex-7 FPGAs [75, 76]. From the figure, the proposed MAC implementation outperforms, in terms of speed, both the DA based and conventional MAC implementations for all values of word-sizes W. We observe also that the conventional MAC implementation outperforms the DA based MAC implementation for small word sizes. The opposite is true for larger word sizes. The intersection point of the blue and red curves occurs when $R = 2W$, which is technology dependent.

4.8 Exploration of Processor array structures

Now, we have six systolic array designs to be explored based on our choices for the scheduling vectors $\boldsymbol{s}_1$, $\boldsymbol{s}_1$ and the projection matrices $\boldsymbol{P}_{ab}$, $\boldsymbol{P}_{ac}$, and $\boldsymbol{P}_{bc}$. The effect of these choices on the algorithm variables will result in different systolic array structures

and different PE details, as will be explained in the following subsections.

4.8.1 Design #1: $\boldsymbol{s}_1 = [1 \quad 1 \quad I]$ and $\boldsymbol{P}_{ab} = [0 \quad 0 \quad 1]$

We begin by studying the effects of $\boldsymbol{s}_1$ and $\boldsymbol{P}_{ab}$ on the output variable y.

Applying the scheduling vector $\boldsymbol{s}_1$ to the null vectors of y, gives:

$$\boldsymbol{s}_1 \boldsymbol{e}_{y1} = 1 \tag{4.59}$$

$$\boldsymbol{s}_1 \boldsymbol{e}_{y2} = I \tag{4.60}$$

Equation (4.59) implies that the output data variable y is pipelined in the positive i-direction and the pipeline delay is one sample. Equation (4.60) implies that the output data variable y is pipelined in the positive j-direction and the pipeline delay is I samples.

Applying the projection matrix $\boldsymbol{P}_{ab}$ to the null vectors of y, gives:

$$\boldsymbol{P}_{ab} \boldsymbol{e}_{y1} = 0 \tag{4.61}$$

$$\boldsymbol{P}_{ab} \boldsymbol{e}_{y2} = 1 \tag{4.62}$$

Equation (4.61) implies that the output data variable y is localized viz; all the calculations required to produce each output sample are performed in the same PE. Equation (4.62) implies that the output data variable y is moving in j-direction viz; all the calculations required to produce each output sample are exchanged between the PEs.

Applying the projection matrix $\boldsymbol{P}_{ab}$ to the output sample extraction point in Eq. (4.19), gives:

$$\boldsymbol{P}_{ab} p2y, i/o = J - 1 \tag{4.63}$$

therefore, the output sample $y(n_t)$ is obtained from processing element with index $J-1$ (PE_{J-1}).

Next we study the effects of $\boldsymbol{s}_1$ and $\boldsymbol{P}_{ab}$ on the input variable h. Applying the scheduling vector $\boldsymbol{s}_1$ on the null vector of h gives:

$$\boldsymbol{s}_1 \boldsymbol{e}_h = 1 \tag{4.64}$$

therefore, the input data variable h is pipelined in a positive n_t-direction and the

pipeline delay is one sample.

Applying the projection matrix $\boldsymbol{P}_{ab}$ to the null vectors of h, gives:

$$\boldsymbol{P}_{ab}\boldsymbol{e}_h = 0 \tag{4.65}$$

therefore, the input data variable h is localized.

Equations (4.64) and (4.65) indicate that each PE will use the same assigned filter coefficient at each time step.

Applying the scheduling vector $\boldsymbol{s}_1$ to the input sample $h(i,j)$ feeding point in (4.22), gives:

$$\boldsymbol{s}_1\boldsymbol{p}_{h,in} + s = i + Ij + (I-1)/2 \tag{4.66}$$

therefore, the input sample $h(i,j)$ will be used at the time step number $i + Ij + (I - 1)/2$.

Applying the projection matrix $\boldsymbol{P}_{ab}$ to the input sample $h(i,j)$ feeding point in (4.22), gives:

$$\boldsymbol{P}_{ab}\boldsymbol{p}_{h,in} = j \tag{4.67}$$

therefore, the input sample $h(i,j)$ is applied to the PE with index j (PE_j) where $0 \leq j < J - 1$.

Finally, we study the effects of $\boldsymbol{s}_1$ and $\boldsymbol{P}_{ab}$ on the input variable x. Applying the scheduling vector $\boldsymbol{s}_1$ to the null vector of x, gives:

$$\boldsymbol{s}_1\boldsymbol{e}_x = I + 1 \tag{4.68}$$

therefore, the input data variable x is pipelined in the positive n_t-j-plane direction and the pipeline delay is $I + 1$ samples.

Applying the projection matrix $\boldsymbol{P}_{ab}$ to the null vectors of x, gives:

$$\boldsymbol{P}_{ab}\boldsymbol{e}_x = 1 \tag{4.69}$$

therefore, the input data variable x is moving in n_t-j-plane direction

Applying the scheduling vector $\boldsymbol{s}_1$ to the input sample $x(i, n_t - j)$ feeding point in (4.25), gives:

$$\boldsymbol{s}_1\boldsymbol{p}_{x,in} = n_t + i + (I-1)/2 \tag{4.70}$$

therefore, each input sample from sensor i should be applied to the systolic array

Table 4.1: Relation between the sensor index i and the sensor sample delay D for Design #1.

$\boldsymbol{i}$	$\boldsymbol{D}$
$-(I-1)/2$	0
$-(I-1)/2+1$	1
$\vdots$	$\vdots$
0	$(I-1)/2$
$\vdots$	$\vdots$
$(I-1)/2-1$	$I-2$
$(I-1)/2$	$I-1$

after a latency of $i+(I-1)/2$ time steps, i.e., the outputs of the sensors should be applied to a one-step triangular delay before they have applied to the PE.

Table 4.1 shows the relation between the input sample time and the sensor index.

Applying the projection matrix $\boldsymbol{P}_{ab}$ to the input sample $x(i, n_t - j)$ feeding point in (4.25), gives:

$$\boldsymbol{P}_{ab}\boldsymbol{p}_{x,in} = 0 \tag{4.71}$$

therefore, all the input samples $x(i, n_t - j)$ is applied to the PE with index 0 (PE_0).

A point $\boldsymbol{p} \in \mathcal{D}$ maps to PE_k where k is given by:

$$k = \boldsymbol{P}_{ab}\boldsymbol{p} = j \tag{4.72}$$

hence the number of PEs in the systolic array is equal to J.

The resulting systolic array for Design # 1 is shown in Fig. 4.5. Figure 4.5(a) shows the interconnection between PEs. Figure 4.5(b) shows the details of each PE.

In the following paragraphs, we will explain how we obtained this systolic array and its PE details.

From Eqs. (4.59) and (4.61), in each PE, the output samples are pipelined and the pipeline delay is one sample as shown in Fig. 4.5(b).

From Eqs. (4.60) and (4.62), the output samples of each PE are pipelined and moving at each I time steps between PEs as shown in Fig. 4.5(a). Based on this conclusion, each PE should contains I multipliers and I accumulators that operates in parallel.

In order to perform high speed multiply/accumulate calculations in the proposed

Design #1, a high-speed, double-precision carry-save multiplier/accumu-lator (CS-MAC) that was reported in [50] is used. The CS-MAC shown in Fig. 4.5(b) multiplies two numbers and produces a double precision result in carry-save format. The accumulation operation is performed in double precision and is merged, and concurrent, with the multiply operation. This can be expressed as

$$2^w (s_{out} + c_{out}) + l_{out} = a \times b + 2^w (s_{in} + c_{in}) + l_{in} \tag{4.73}$$

where s_{out}, c_{out} are the high-order sum and carry word output from the CS-MAC, l_{out} is the low order word output from the CS-MAC, s_{in}, c_{in} are the high-order sum and carry word input to the CS-MAC, and l_{in} is the low order word input to the CS-MAC.

A reduction in the computation time ranging from 20% to 50% compared with other schemes has been achieved using the CS-MAC, without a significant increase in the required area [50].

From Eqs. (4.64) and (4.65), at each PE the assigned beamforming filter coefficients $h(i, j)$ are pipelined and the pipeline delay is one sample.

From Eqs. (4.68) and (4.69), the input samples $x(i, n_t - j)$ to each PE are pipelined and moving at each $I + 1$ time steps between PEs as shown in Fig. 4.5(a).

From Eqs. (4.70) and (4.71), the input samples $x(i, n_t - j)$ should applied to a one-step triangular delay before they have applied to the first PE as shown in Fig. 4.5(a).

4.8.2 Design #2: $\boldsymbol{s}_1 = [1 \quad 1 \quad I]$ and $\boldsymbol{P}_{ac} = [0 \quad 1 \quad 0]$

We study the effects of $\boldsymbol{s}_1$ and $\boldsymbol{P}_{ac}$ on the output variable y and the input variables h and x. Following the same steps used in Design #1, the resulting systolic array for Design #2 is shown in Fig 4.6. Figure 4.6(a) shows the interconnection between PEs. Figure 4.6(b) shows the details of each PE.

4.8.3 Design #3: $\boldsymbol{s}_1 = [1 \quad 1 \quad I]$ and $\boldsymbol{P}_{bc} = [1 \quad 0 \quad 0]$

We study the effects of $\boldsymbol{s}_1$ and $\boldsymbol{P}_{bc}$ on the output variable y and the input variables h and x. Following the same steps used in Design #1, any point $p = [n_t \; i \; j]^T \in \mathcal{D}$, maps to PE_k where $k = n_t$. That the mapping leads to an infinite number of PEs, which is not practical. However, the scheduling vector $\boldsymbol{s}_1$, indicates that any PE is

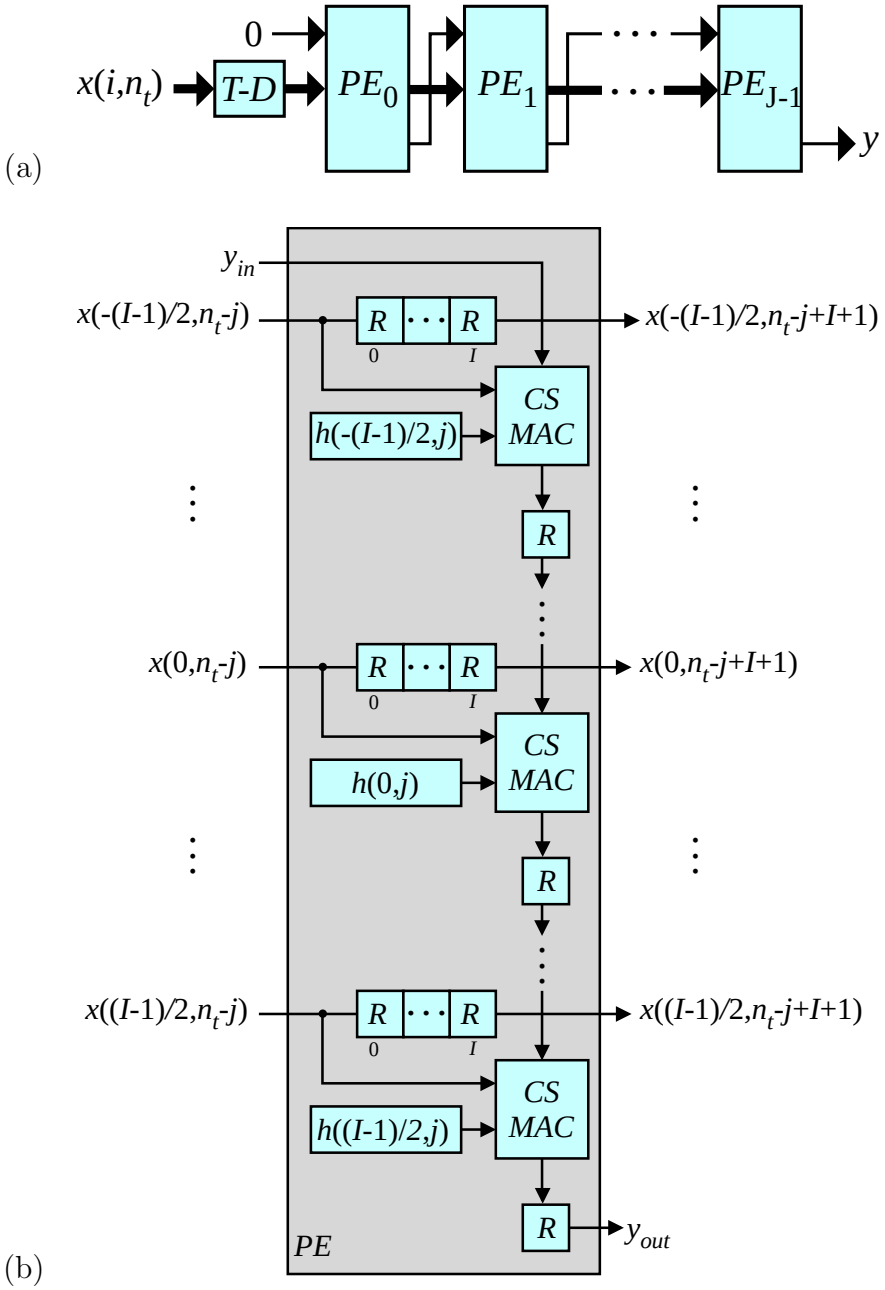


Figure 4.5: Systolic array for Design #1. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details.

active for IJ time steps. Similarly, at any given time step, the number of active PEs is IJ also.

Based on above conclusions, the resulting systolic array for Design #3 is as shown in Fig 4.7. Figure 4.7(a) shows the interconnection between PEs. Figure 4.7(b) shows

the details of each PE.

4.8.4 Design #4: $\boldsymbol{s}_2 = [1 \quad J \quad 1]$ and $\boldsymbol{P}_{ab} = [0 \quad 0 \quad 1]$

We study the effects of $\boldsymbol{s}_2$ and $\boldsymbol{P}_{ab}$ on the output variable y and the input variables h and x. Following the same steps used in Design #1, the resulting systolic array for Design #4 is as shown in Fig 4.8. Figure 4.8(a) shows the interconnection between PEs. Figure 4.8(b) shows the details of each PE.

4.8.5 Design #5: $\boldsymbol{s}_2 = [1 \quad J \quad 1]$ and $\boldsymbol{P}_{ac} = [0 \quad 1 \quad 0]$

We study the effects of $\boldsymbol{s}_2$ and $\boldsymbol{P}_{ac}$ on the output variable y and the input variables h and x. Following the same steps used in Design #1, the resulting systolic array for Design #5 is as shown in Fig 4.9. Figure 4.9(a) shows the interconnection between PEs. Figure 4.9(b) shows the details of each PE.

4.8.6 Design #6: $\boldsymbol{s}_2 = [1 \quad J \quad 1]$ and $\boldsymbol{P}_{bc} = [1 \quad 0 \quad 0]$

We study the effects of $\boldsymbol{s}_2$ and $\boldsymbol{P}_{bc}$ on the output variable y and the input variables h and x. Following the same steps used in Design #1 and Design #3, the resulting systolic array for Design #6 is as shown in Fig 4.10. Figure 4.10(a) shows the interconnection between PEs. Figure 4.10(b) shows the details of each PE.

4.9 Hardware Implementation Results

In this section the results of the hardware implementations will be presented. The hardware platform used for all designs is Xilinx Artix 7 XC7A100T on Digilent BASYS 3 FPGA board.

The six designs presented in the previous section were implemented, as well as, a conventional 2-D BB BF filter. The implementation of the conventional 2-D BB BF filter used here will be discussed in Sec. 4.9.1 and the results of the implementations in terms of hardware resources and speed, for all designs are presented in Sec. 4.9.2. In the next section the performance of the various implementations will be compared.

FPGA implementations of the conventional and the proposed designs were implemented using Xilinx Vivado Design Suite (version 2017.4) and VHDL.

4.9.1 Conventional Design Implementation

The conventional systolic array implementation of the 2-D BB BF filter used here consists of an array of 1-D FIR filters followed by an array of adders, and it is shown in Fig. 4.11.

Figure 4.11(a) shows the systolic array of the conventional 2-D BB BF filter. The number of 1-D FIR filter is equal to I.

Figure 4.11(b) shows the systolic array of the 1-D FIR filter, which corresponds to a form 2 implementation of an FIR filter. The number of PEs is equal to the number of 1-D FIR filter coefficients J. We assume double precision mode of operation, where data exchanged between the PEs is in double precision format. This is explicitly shown in the figure as the partial sums L and H.

Figure 4.11(c) shows the 1-D FIR filter PE details. The register h stores the appropriate filter coefficient. The multiplier performs a 2's complement multiplication operations with double-precision output. The lower bits of the multiplier output are added to the lower bits of the previous PE output using a single-precision ripple-carry adder (RCA) to produce the lower bits of the PE output L_{out}. Similarly, the higher bits of the multiplier output are added to the higher bits of the previous PE output using a RCA adder to produce the higher bits of the PE output H_{out}.

4.9.2 Implementation Results

The conventional and proposed Designs #1 to #6 are implemented using FPGA. The number of FIR filter coefficients were chosen to be 480 corresponding to $I = 15$ and $J = 32$.

Table 4.2 shows the performance (speed, LUTs, Registers, etc.) of the conventional and the six proposed designs for the 2-D BB BF filter.

The clock duration (delay) in Table 4.2 for the proposed designs is determined from Fig. 4.5, 4.6, 4.8, and 4.9 to be the delay of the carry-save multiplier accumulator. On the other hand, the clock duration of the conventional design is determined by the adder required to add all the results of the 1-D FIR filters.

The speedup factor in Table 4.2 is defined as:

$$S = \frac{f}{f_c} \tag{4.74}$$

where f is the clock speed of the proposed designs and f_c is the clock speed of the

conventional design.

The percentage speedup increase in Table 4.2 is defined as:

$$\Delta S = \frac{f - f_c}{f_c} \times 100\ \% \tag{4.75}$$

Table 4.2: Comparison between resource utilization for conventional and proposed designs for the case of $I = 15$, $J = 32$ and PE word size = 8 bits.

2-D BB BF filter structure	Conv.	Proposed Designs					
(logic utilization)	Design	#1	#2	#3	#4	#5	#6
# Slice Registers	7,682	26,400	25,935	33,119	27,843	18,960	25,919
# Slice LUTs	12,026	49,643	49,563	132,163	38,551	38,777	124,483
# bonded IOBs	154	282	162	139	162	162	139
Max. Freq. (MHz)	74	253	249	133	266	262	133
Delay (ns)	13.596	3.949	4.014	7.546	3.763	3.824	7.546
Speedup factor S	1	3.4	3.4	1.8	3.6	3.5	1.8
Speedup increase ΔS (%)	0	241.9	236.5	79.7	259.4	254.0	79.7

4.10 Performance Comparison

Table 4.2 shows that the six proposed designs outperform the conventional design in terms of speed, which was the main design goal of this work. This increase in speed, however, comes at a cost of increased hardware resources required.

The average speedup factor in Eq. 4.74 is approximately 3.48 for Designs #1, #2, #4, and #5. The average speedup factor is 1.8 for Designs #3, and #6.

The clock duration for designs #1, #2, #4, and #5 is determined from Figs. 3.6, 3.10, 4.8 and 4.9 to be the delay of the CS-MAC. On the other hand, the clock duration for designs #3 and #6 is determined from Figs. 4.7 and 4.10 to be the delay

of the CS-MAC plus the delay of the RCA adder. This explains why the speed ups of Designs #1, #2, #4, and #5 are approximately twice those of Designs #3 and #6.

Based on these observations, Designs #1, #2, #4, and #5 are good candidates for the implementation of a high-speed 2-D BB BF filter.

The number of registers required for the proposed designs exceeds those required by the conventional design by a factor 2.5 to 3.4. The increased use of registers is due to the extensive use of pipelining to increase the clock speed.

The number of LUTs required for the proposed designs exceeds those required by the conventional design by a factor 3.2 to 4.1. The increased use of LUTs to construct the extra logic and registers in the proposed designs to increase the clock speed.

Based on the number of registers and LUTs requirements, Designs #4 and #5 are better candidates compared to Designs #1 or #2.

The number of bonded IOBs in Designs #4 and #5 are the same, which is comparable to the IOBs requirements for the conventional design.

From all the above observations, Design #4 is deemed to offer the best speedup compared to the other proposed designs and the conventional design.

4.11 Conclusion

The development of systolic array implementations for 2-D BB BF filters is explored in this chapter using a systematic methodology [6]. This methodology is used to perform a general design space exploration of the possible 2-D BB BF implementation structures. This design space exploration was conducted through selection of different scheduling and projection functions that satisfied the system's I/O restrictions. Extensive data pipelining was used for the scheduling function selection to obtain the fastest possible system clock speed. A significant improvement in reducing the total delay by a factor of four while at the same time useing double precision operations was made possible by the use of Merged CS-MAC.

Six different systolic array designs were obtained and evaluated. The conventional and proposed systolic array designs, were implemented in FPGA hardware to verify their functionality and compare their performance. On the average four of the proposed designs produced a speedup factor of 3.48 compared to the conventional design. This speedup came at the cost of increased register and LUT hardware requirements. Based on speedup and hardware resources requirements, it is concluded that Design #4 offers the best performance.

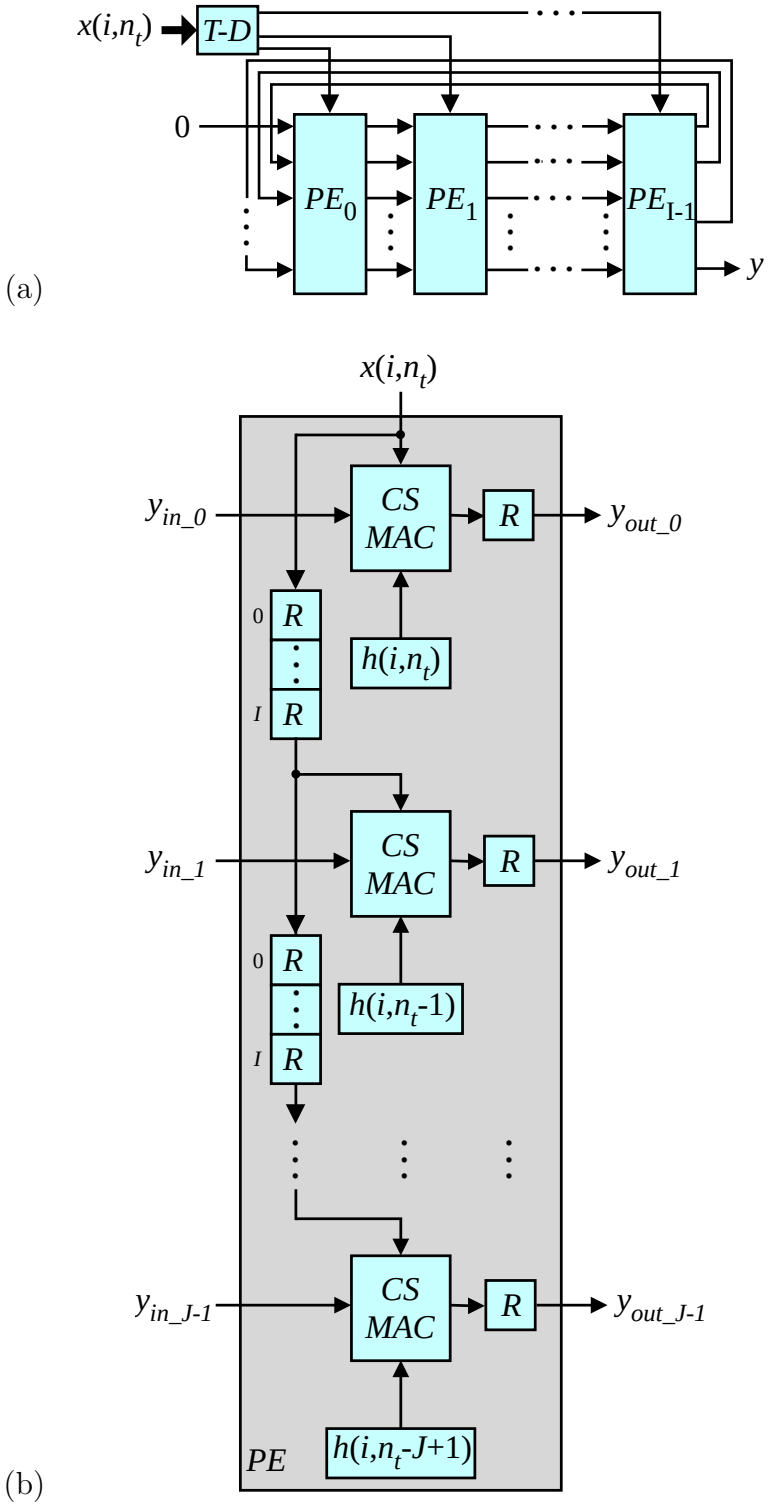


Figure 4.6: Systolic array for Design #2. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details.

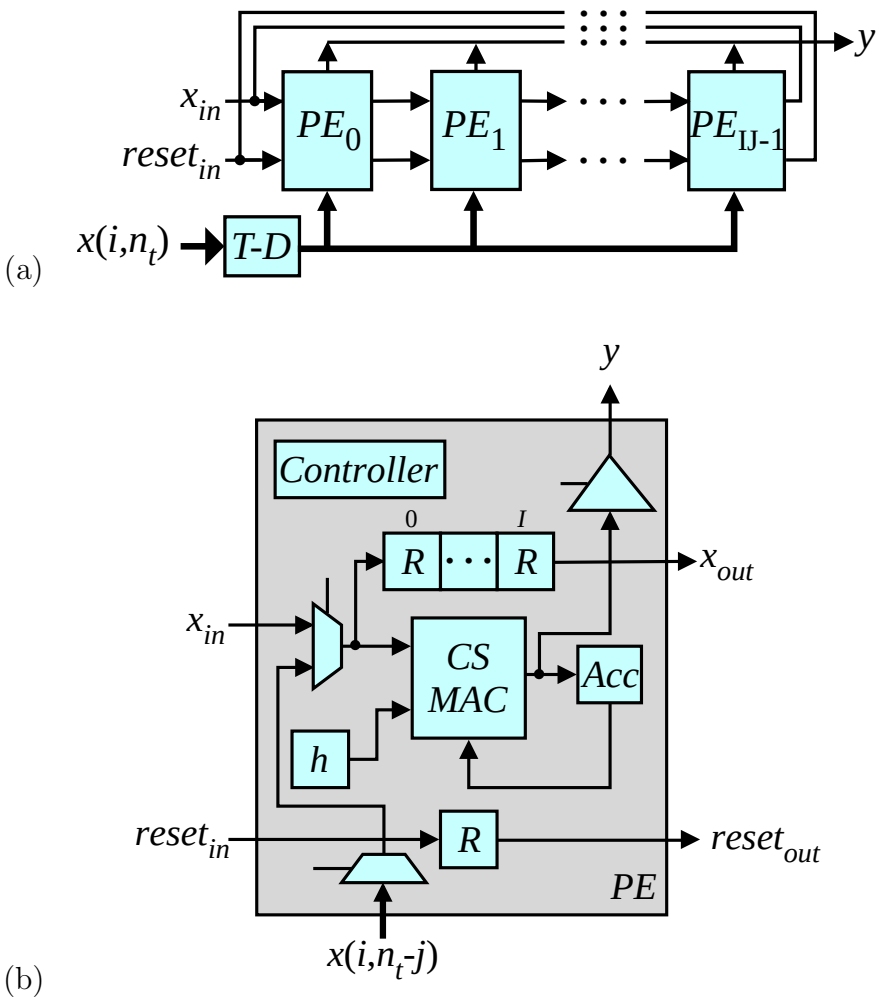


Figure 4.7: Systolic array for Design #3. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details.

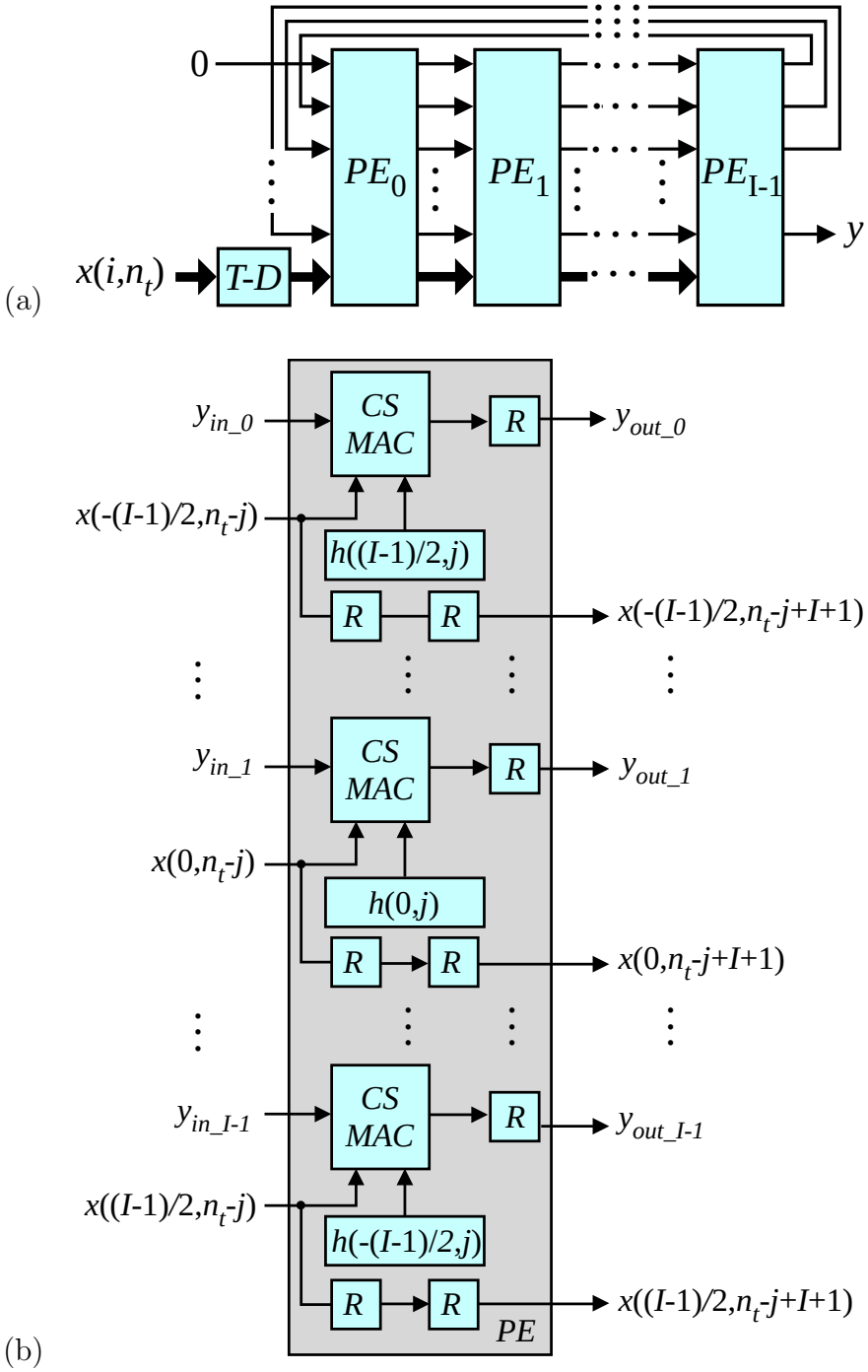
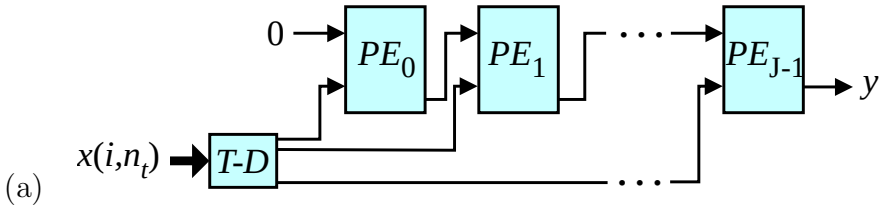
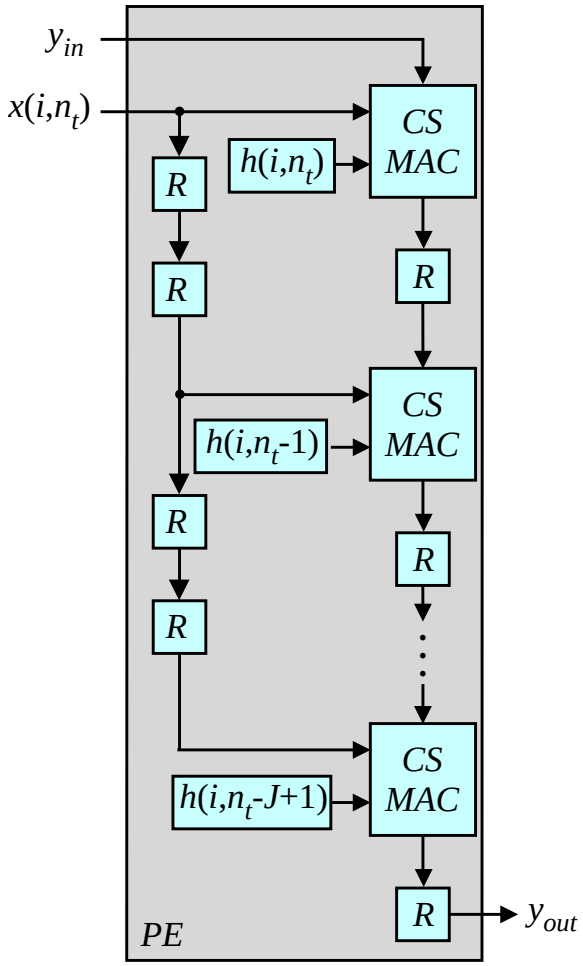


Figure 4.8: Systolic array for Design #4. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details.

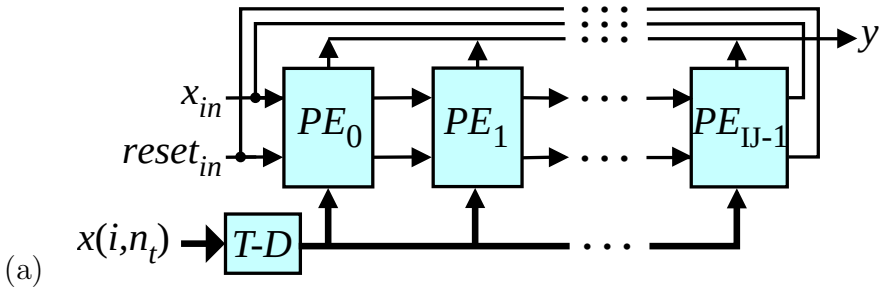


(a)

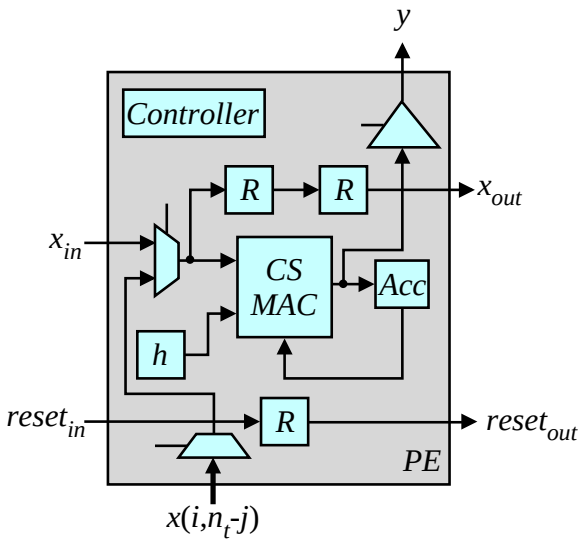


(b)

Figure 4.9: Systolic array for Design #5. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details.



(a)



(b)

Figure 4.10: Systolic array for Design #6. (a) The systolic array preceded by a triangular delay (T-D). (b) PE details.

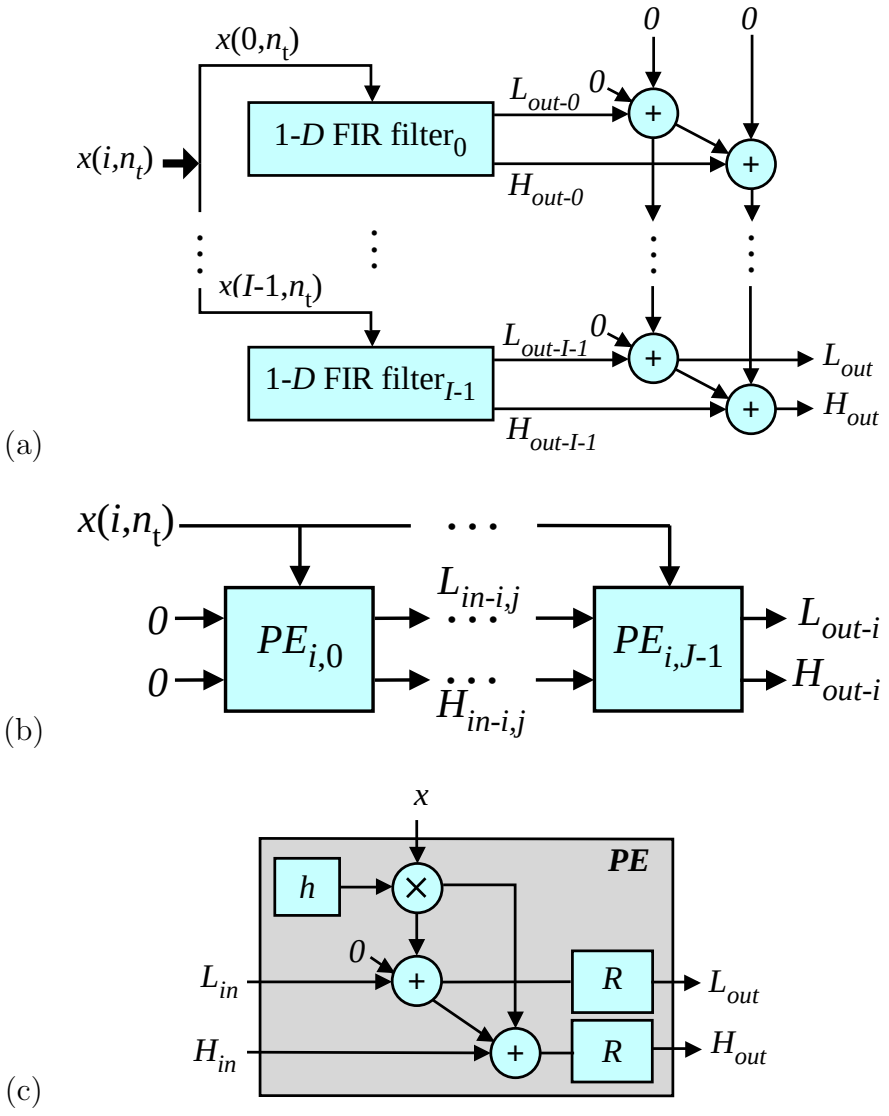


Figure 4.11: The conventional 2-D BB BF filter implementation. (a) The systolic array of the conventional 2-D BB BF filter. (b) The systolic array of the 1-D FIR filter. (c) The 1-D FIR filter PE details.

Chapter 5

Design Space Exploration of the Interpolators

This chapter presents the development and comparison of interpolator systolic array designs and implementations. Systematic methodology was applied to the difference equations defining the interpolator algorithm. A dependence graph for the interpolator was obtained that combined the upsampler and the anti-imaging filter. Different data scheduling and projection operations were developed. Nine systolic array design options were obtained and evaluated. The fastest design was selected for hardware implementation. FPGA implementations for the conventional and proposed designs confirm that the proposed interpolator implementation requires no more than 61.7% of the hardware resources required in the conventional design and are at least 63.9% faster than the conventional design.

5.1 Introduction and Related Work

Multirate digital signal processing is used in systems where signals are processed at different data rates [77, 78]. Interpolators are used in multirate systems to generate signals with higher data rates. Examples where interpolators are an essential component include implementation of the interpolation filters [79], nested arrays broadband bemformers [11, 24] and DFT filter bank beamformers [30]. Interpolators can also be found in other general areas such as radar (MIMO [34] - SAR [80, 81]), communications [35], and image processing [38, 39].

Kaya and Seke, proposed a memory based upsampling/interpolating FIR filter

modification/extension to distributed arithmetic based FIR filters that can be used for any filter coefficient set [82]. Using distributed arithmetic removed the need to use multipliers and the design was implemented on FPGAs and lookup tables (LUTs). The number of LUTs exponentially depends on the data word size. Abd El-Azeem et al. presented a Computational Filter (CF) that employs a sample calculation functional block [83]. This CF significantly reduces the hardware requirements to realize an interpolation filter. Al-Haj, presented a description of a parallel and high speed, single-chip implementation of the fundamental multirate filter banks. The hardware implementation platform is based on Virtex field programmable gate arrays (FPGAs) [84]. Zheng et al. decomposed the multirate FIR filter (MRFIR) into output computational threads. Each thread represents an instance of the finite inner-product required to produce a single output of the MRFIR. The filter is thus viewed as a finite collection of concurrent threads [45]. The number of threads for the interpolator equals the number of the FIR filter coefficients. Porat and Gebali published a polyphase implementation of an interpolator structure where the upsampler succeeded the polyphase filter structure [85, 6].

5.2 Systematic Methodology for Systolic Array Design Applied to the Interpolators

In order to perform systolic array design space exploration of the interpolator, we use a systematic methodology that was proposed in [6] for regular iterative algorithms (RIAs). The interpolator algorithm is given by:

$$u(i) = x(i/L), \qquad i \geq 0 \tag{5.1}$$

$$y(i) = \sum_{j=0}^{J-1} h(j)u(i-j) \tag{5.2}$$

where L is the interpolation factor and J is the number of the anti-imaging FIR filter coefficients.

The methodology of [6] specifies several steps which are adapted here for the interpolator algorithm as follows:

1. The difference equations of the interpolator algorithm, expressed as an RIA, are shown in Eqs. (5.1) and (5.2).

2. Define a computational domain $\mathcal{D} \subset \mathcal{Z}^n$ of the algorithm based on the RIA. Since Eqs. (5.1) and (5.2) use two indices i and j, the algorithm is defined in the 2-dimensional integer domain $\mathcal{Z}^2$. In Section 5.3 the computational domain of the interpolator $\mathcal{D} \subset \mathcal{Z}^2$ is defined through investigation of the ranges of indices i and j.

3. Define the subdomain of each variable in $\mathcal{D}$ using the dependence of the algorithm variables on the iteration indices. This is explained in Section 5.3.

4. Obtain the scheduling functions that satisfy the input and output data timing specifications and constrains. In Section 5.4, valid scheduling functions for the interpolator algorithm are explored.

5. Obtain the projection functions that satisfy the scheduling functions and any hardware restrictions. In Section 5.5, projection directions associated with valid scheduling functions for the interpolator algorithm are being explored.

It should be pointed out that the systematic methodology is applicable when the algorithm can be expressed as an RIA according to Step 1 above. Furthermore, efficient hardware structures are obtained when the filter used in the interpolator is an FIR filter whose number of coefficients is an integer multiple of the interpolation factor. Zero padding can always be used to circumvent the above limitation.

The term systolic array design will be used to describe the architecture and the functionality of a systolic array while systolic array implementation represents the actual implementation of this design in hardware.

5.3 Interpolator Dependence Graph (DG)

The interpolator Eqs. (5.1) and (5.2) define a sequential evaluation of the interpolation algorithm. The algorithm is iterative and depends on two indices i and j. There are two input variables $h(j)$ and $x((i-j)/L)$. There is one intermediate variable $u(i-j)$ and one output variable $y(i)$. Note that sample $x(i)$ corresponds to the intermediate sample $u(iL)$ according to Eq. (5.1).

The powerful systematic technique of reference [6] is used to perform systolic array design space exploration of the interpolator structure based on the iterations defined by Eqs. (5.1) and (5.2). Figure 5.1 shows the dependence graph of the interpolator

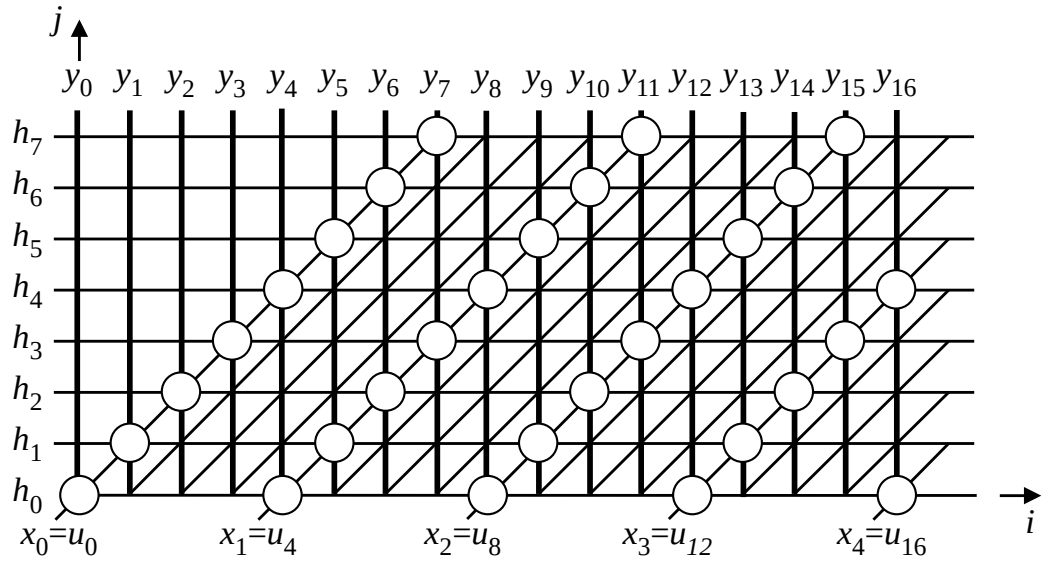


Figure 5.1: 1-to-L interpolator dependence graph for the case when $L = 4$ and $J = 8$. Empty circles denote multiply/accumulate operations.

for the case when $L = 4$ and $J = 8$. The horizontal axis is the i-axis (range $i \geq 0$) and vertical axis is the j-axis (range $0 \leq j < 8$).

The interpolator output $y(i)$ is shown at the top of the figure. $y(i)$ is shown by the thick vertical lines since each output sample depends on the i index only.

The empty circles indicate useful filtering operations that result in the generation of the output samples $y(i)$.

In this work, the systematic methodology is employed using different scheduling and projection operations for systolic array design space exploration of the interpolator.

5.4 Interpolator Scheduling Function

The scheduling function assigns a time index value for the operation of each point in the DG of Fig. 5.1. A simple linear scheduling function is used to assign time index values to the DG nodes:

$$t(\boldsymbol{p}) = \boldsymbol{sp} \tag{5.3}$$

where $\boldsymbol{s} = [\alpha \quad \beta]$ is the scheduling row vector and $\boldsymbol{p} = [i \quad j]^T$ is a point in Fig. 5.1. Therefore the time associated with a node is given by:

$$t(\boldsymbol{p}) = i\alpha + j\beta \tag{5.4}$$

An edge connecting two nodes having the same time value are said to be lying on the same equitemporal zone. Data flowing between these two nodes is said to be broadcast since the data value is shared by the two nodes at the same time value. An edge connecting two nodes with different time values indicate that data is pipelined from the node with lower time value to the node with higher time value. The scheduling function transforms the DG to a directed acyclic graph (DAG) [6].

The scheduling vector components α and β are determined subject to input and output data specifications and the decision whether to pipeline or broadcast a variable.

It is assumed that the input data $x(i)$ arrive at consecutive time steps, we can write:

$$t(\boldsymbol{p}_2) = t(\boldsymbol{p}_1) + 1 \tag{5.5}$$

Assuming further that $x(i)$ sample is supplied to the DG at point $\boldsymbol{p} = [i \quad 0]^T$, we have:

$$t(\boldsymbol{p}_1) = \alpha\, i \tag{5.6}$$

$$t(\boldsymbol{p}_2) = \alpha(i+1) \tag{5.7}$$

Equations (5.5)-(5.7) result in $\alpha = 1$. A valid scheduling vector that satisfies the above assumptions about input data timing is given by

$$\boldsymbol{s} = [1 \quad \beta] \tag{5.8}$$

The value of β will be determined by our choice of whether we need to pipeline or broadcast the output sample $y(i)$. We have three possible valid scheduling vectors that we can employ:

$$\boldsymbol{s}_1 = [1 \quad -1] \tag{5.9}$$

$$\boldsymbol{s}_2 = [1 \quad 1] \tag{5.10}$$

$$\boldsymbol{s}_3 = [1 \quad 0] \tag{5.11}$$

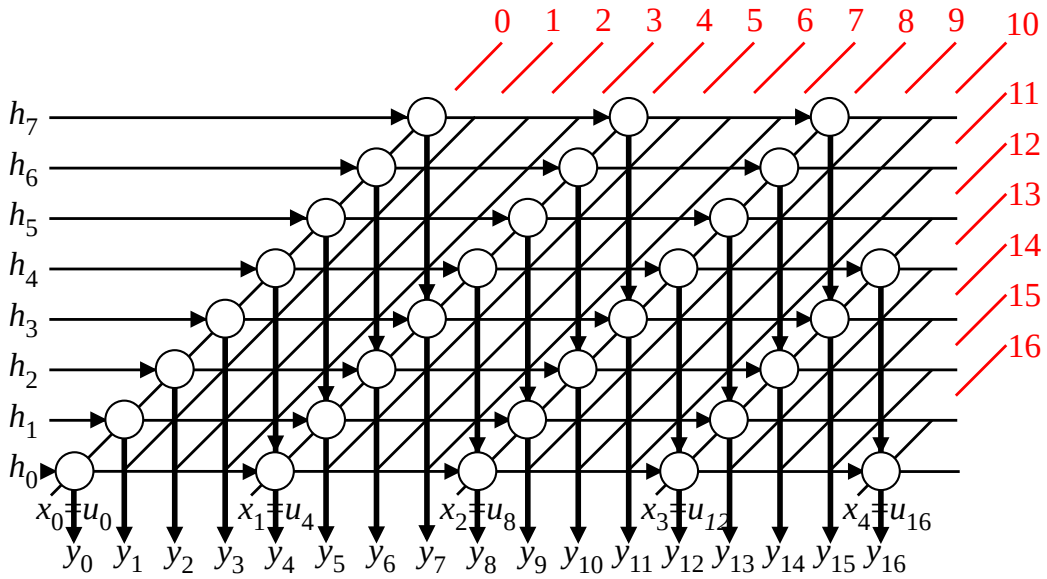


Figure 5.2: The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_1$ in Eq. (5.9).

Figures 5.2, 5.3, and 5.4 show the DAG corresponding to scheduling vectors $\boldsymbol{s}_1$, $\boldsymbol{s}_2$, and $\boldsymbol{s}_3$ respectively. The equitemporal zones and the time index values are indicated by the red lines and the red numbers respectively. The inputs $x(i-j)$ and $h(j)$ are indicated by the arrows and the output $y(i)$ is indicated by the vertical lines. The empty circles indicate all the multiply/accumulate operations for each output sample.

The scheduling vector $\boldsymbol{s}_1$ results in broadcast input $x(i-j)$ and pipelined output $y(i)$. Pipelined output allow a fastest clock rate.

The scheduling vector $\boldsymbol{s}_2$ results in pipelined input $x(i-j)$ and pipelined output $y(i)$. There are few output data timing problems associated with the scheduling vector $\boldsymbol{s}_2$. The first problem is, the delay associated with each output sample is not uniform. For example, y_1 is obtained at time $t=2$ instead of $t=1$, y_8 is obtained at time $t=12$ instead of $t=8$, and y_{12} is obtained at time $t=16$ instead of $t=12$. Another output data timing problem is appearance of more than one sample at the same time step. For example, both output samples y_6 and y_8 are obtained at time $t=12$ and both output samples y_7 and y_9 are obtained at time $t=14$. For these reasons, scheduling vector $\boldsymbol{s}_2$ shall be excluded as well from the design space exploration choices.

The scheduling vector $\boldsymbol{s}_3$ results in pipelined input $x(i-j)$ and broadcast output $y(i)$. Since broadcast output results in expensive hardware and/or slow speed, the

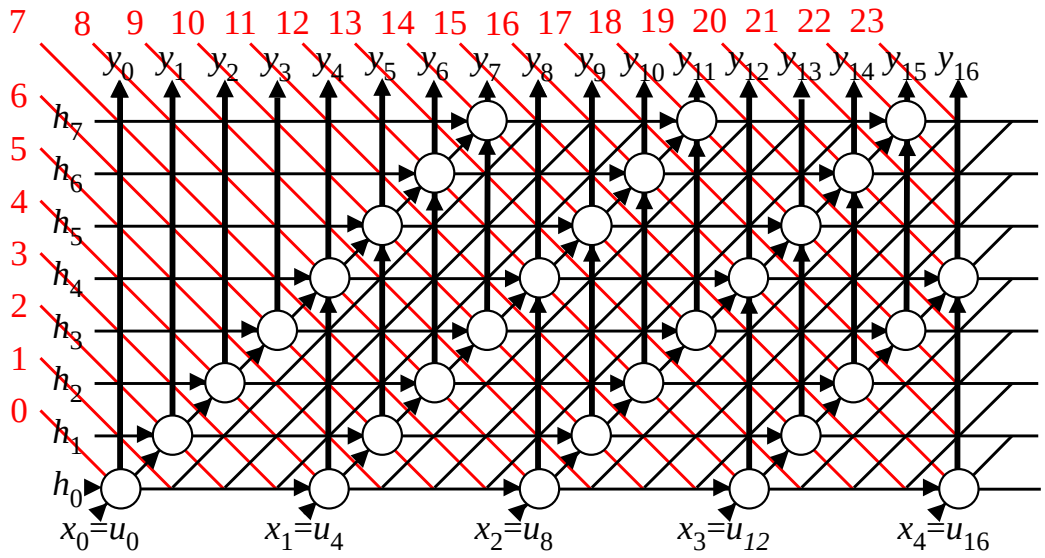


Figure 5.3: The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_2$ in Eq. (5.10).

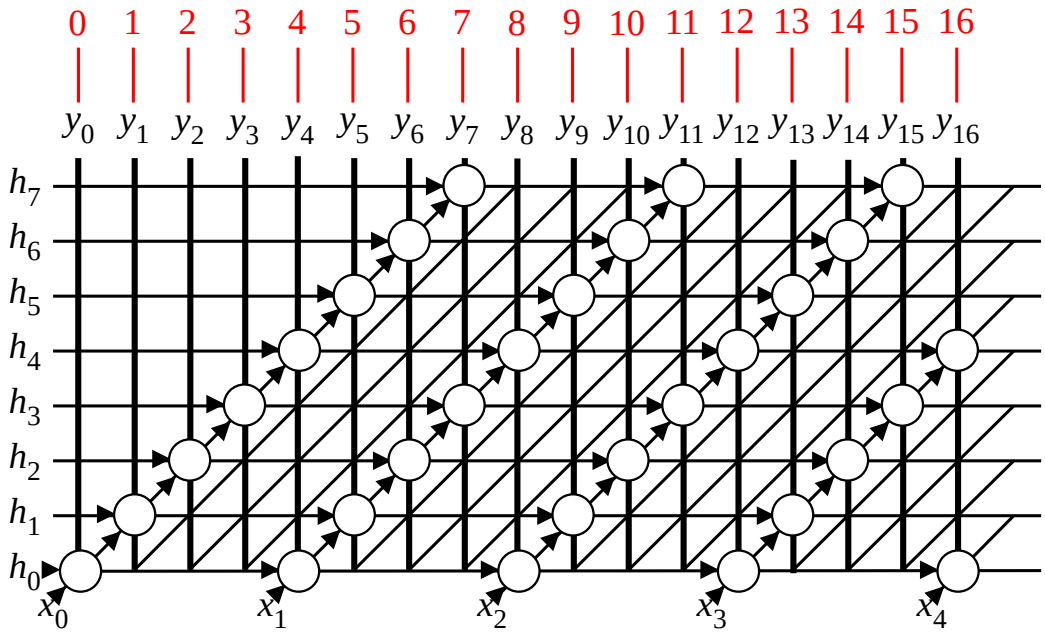


Figure 5.4: The DAG and the Equitemporal zones for scheduling vector $\boldsymbol{s}_3$ in Eq. (5.11).

scheduling vector $\boldsymbol{s}_3$ shall be excluded from the design space exploration choices.

5.5 Interpolator Node Projection

Linear projection is defined as the mapping of several points in an n-dimensional DG to processing element PE_k where k is the processor index that given by [6]:

$$k = \boldsymbol{P}\boldsymbol{p} \tag{5.12}$$

The projection matrix $\boldsymbol{P}$ is determined by the associated projection direction $\boldsymbol{d}$, which is the nullspace of the matrix.

According to [6], the selection of a valid projection direction should satisfy the following condition:

$$\boldsymbol{s}\boldsymbol{d}_i \neq 0 \tag{5.13}$$

where $\boldsymbol{s}$ is the chosen scheduling vector. In this work, it is aimed to map the points in the 2-D DG shown in Fig. 5.1 to a 1-D domain.

Applying condition (5.13) to the three scheduling vectors in Eq. (5.9), (5.10) and (5.11), four different projection directions are obtained:

$$\boldsymbol{d}_1 = [1 \quad 0]^T \tag{5.14}$$

$$\boldsymbol{d}_2 = [1 \quad 1]^T \tag{5.15}$$

$$\boldsymbol{d}_3 = [1 \quad -1]^T \tag{5.16}$$

$$\boldsymbol{d}_4 = [0 \quad 1]^T \tag{5.17}$$

Only three of the previous projection directions are valid for each schehuling vector. For $\boldsymbol{s}_1$, the vectors $\boldsymbol{d}_1$, $\boldsymbol{d}_3$, and $\boldsymbol{d}_4$ are valid. For $\boldsymbol{s}_2$, the vectors $\boldsymbol{d}_1$, $\boldsymbol{d}_2$, and $\boldsymbol{d}_4$ are valid. For $\boldsymbol{s}_3$, the vectors $\boldsymbol{d}_1$, $\boldsymbol{d}_2$, and $\boldsymbol{d}_3$ are valid.

Based on the above discussion, the three scheduling vectors and their associated projection directions produce nine different systolic arrays. This is discussed in more details in Section 5.6

These projection directions are then used to calculate the associated projection matrices according to the procedure described in [6]. Given a project direction $\boldsymbol{d} = [\gamma \quad \delta]^T$, and knowing that $\boldsymbol{d}$ is the nullvector of the projection matrix $\boldsymbol{P}$ [6], There are two possible forms:

$$\boldsymbol{P} = [\delta \quad -\gamma] \quad \text{or} \quad \boldsymbol{P} = [-\delta \quad \gamma] \tag{5.18}$$

Table 5.1: The possible systolic array design-options

Design-Option #	**Scheduling Vector**	**Projection Direction**	**Main Features**
1	$\boldsymbol{s}_1$	$\boldsymbol{d}_1$	Pipelined output
2		$\boldsymbol{d}_3$	
3		$\boldsymbol{d}_4$	
4	$\boldsymbol{s}_2$	$\boldsymbol{d}_1$	Timing problems
5		$\boldsymbol{d}_2$	
6		$\boldsymbol{d}_4$	
7	$\boldsymbol{s}_3$	$\boldsymbol{d}_1$	Broadcast output
8		$\boldsymbol{d}_2$	
9		$\boldsymbol{d}_3$	

The proper form to use is the one that results in positive PE indices after the project operation.

Using Eqs. (5.12) and (5.18) we get :

$$k = i\delta - j\gamma \tag{5.19}$$

For more flexibility in this work, a non-linear node projection operation have been adopted of the form [6]:

$$k' = \left\lfloor \frac{k}{m} \right\rfloor \tag{5.20}$$

where k was given in Eq. (5.12) and m is the desired number of points in DAG that will be assigned to one PE.

5.6 Systolic Array Design Space Exploration

In this section, the systolic array design space of linear processor array will be explored for the interpolator using the calculated scheduling vectors and projection directions.

Table 5.1 illustrates how nine possible systolic array design options are obtained. The nine possible design options are obtained based on the combination of the scheduling functions and projection directions, as explained in the table. The advantages and disadvantages were also summarized in the table.

Based on the discussion in Section 5.4, scheduling vectors $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$ will not be considered. Therefore, there is only one valid scheduling vector $\boldsymbol{s}_1$ with its associated valid projection directions. This gives a total of three possible design options that allow the fastest clock rate. The following subsections discuss the three design options to be explored.

5.6.1 Design-Option #1: using $\boldsymbol{s}_1 = [1 \ \ -1]$ and $\boldsymbol{d}_1 = [1 \ \ 0]^T$

The DAG corresponding to the scheduling vector $\boldsymbol{s}_1$ is shown in Fig. 5.5. The time index associated with any point $\boldsymbol{p}$ is given by:

$$t(\boldsymbol{p}) = i - j \tag{5.21}$$

According to Eq. (5.18), the projection matrix corresponding to the linear projection direction $\boldsymbol{d}_1$ is given by:

$$\boldsymbol{P}_1 \ = \ [0 \quad 1] \tag{5.22}$$

By substituting $\boldsymbol{P}_1$ in Eq. (5.19), we obtain:

$$k = j \tag{5.23}$$

The PE associated with each point are indicated by the numbers inside the circles in Fig. 5.5. The number of PEs will be finite and equal to the number of anti-imaging filter coefficients J. The communications between the PEs is inferred from the interconnection between the points in Fig. 5.5.

It is observed in Fig. 5.5 that active nodes lie on the lines defined by the equation:

$$i - j = kL, \quad k = 0, 1, 2 \cdots \tag{5.24}$$

This indicates that there are J PEs but they are active once every L time steps. We can reduce the hardware resources if we only use J/L PEs that are active now in all L time steps. This can be accomplished using non-linear scheduling and projection functions. The non-linear scheduling function used is of the form:

$$t'(\boldsymbol{p}) = i - j + \mathrm{rem}\,(j/L) \tag{5.25}$$

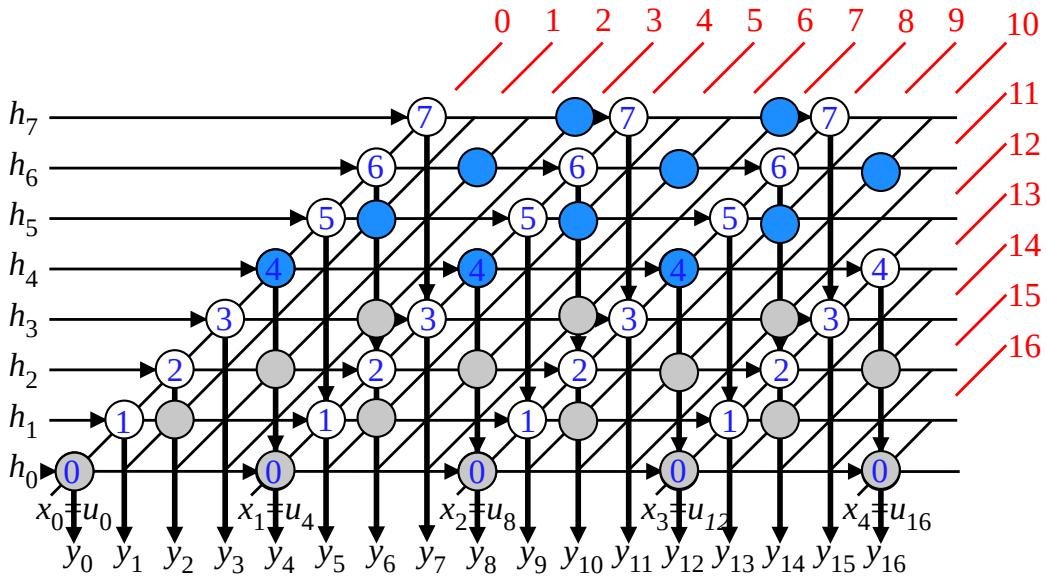


Figure 5.5: The DAG for Design-Option #1: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_1 = [1 \quad 0]^T$.

where *rem(x/y)* is the remainder of dividing x by y. The above equation applies to the points on the lines that satisfy Eq. (5.24).

After applying the non-linear function it is found that there is only J/L active nodes at each time step. Each of those node is assigned a separate PE using a non-linear projection function:

$$k' = \left\lfloor \frac{j}{J/L} \right\rfloor \tag{5.26}$$

where k' is the new PE index. For the case when $J = 8$ and $L = 4$ we would have only two PEs as was observed in Fig. 5.5. The nodes that map to PE_0 and PE_1 are shown as the grey and blue circles respectively in Fig. 5.5. The figure now shows that only two processors are active at each time step. The number of PEs in general is given by J/L. Figure 5.5 indicates that the nodes implement the basic operation of multiply/accumulate (MAC) operation. To reduce roundoff effects and computational noise, double-precision MAC is implemented in the form:

$$2^w e_h + e_l = a \times b + 2^w d_h + d_l \tag{5.27}$$

where a, b are single-precision multiplier and multiplicand inputs, d_h, d_l are the high-order and low-order word input to the double-precision adder, e_h, e_l is the high-order and low-order word output from the double-precision adder, and w is the number of bits in the high-order and low-order words. The delay to perform the multiplication

Table 5.2: PE_0 and PE_1 activities for Design-Option #1 after applying the non-linear scheduling function in Eq. (5.25) when $J = 8$ and $L = 4$.

$\boldsymbol{t}$	$\boldsymbol{x}$	$\boldsymbol{PE}_0$	$\boldsymbol{PE}_1$	$\boldsymbol{y}$
0	x_0	$y_0 = x_0h_0$	$y_{4tmp} = x_0h_4$	$y_0 = x_0h_0$
1		$y_1 = x_0h_1$	$y_{5tmp} = x_0h_5$	$y_1 = x_0h_1$
2		$y_2 = x_0h_2$	$y_{6tmp} = x_0h_6$	$y_2 = x_0h_2$
3		$y_3 = x_0h_3$	$y_{7tmp} = x_0h_7$	$y_3 = x_0h_3$
4	x_1	$y_4 = x_1h_0 + y_{4tmp}$	$y_{8tmp} = x_1h_4$	$y_4 = x_1h_0 + x_0h_4$
5		$y_5 = x_1h_1 + y_{5tmp}$	$y_{9tmp} = x_1h_5$	$y_5 = x_1h_1 + x_0h_5$
6		$y_6 = x_1h_2 + y_{6tmp}$	$y_{10tmp} = x_1h_6$	$y_6 = x_1h_2 + x_0h_6$
7		$y_7 = x_1h_3 + y_{7tmp}$	$y_{11tmp} = x_1h_7$	$y_7 = x_1h_3 + x_0h_7$
8	x_2	$y_8 = x_2h_0 + y_{8tmp}$	$y_{12tmp} = x_2h_4$	$y_8 = x_2h_0 + x_1h_4$
9		$y_9 = x_2h_1 + y_{9tmp}$	$y_{13tmp} = x_2h_5$	$y_9 = x_2h_1 + x_1h_5$
10		$y_{10} = x_2h_2 + y_{10tmp}$	$y_{14tmp} = x_2h_6$	$y_{10} = x_2h_2 + x_1h_6$
11		$y_{11} = x_2h_3 + y_{11tmp}$	$y_{15tmp} = x_2h_7$	$y_{11} = x_2h_3 + x_1h_7$
12	x_3	$y_{12} = x_2h_0 + y_{12tmp}$	$y_{17tmp} = x_3h_4$	$y_{12} = x_3h_0 + x_2h_4$
13		$y_{13} = x_2h_1 + y_{13tmp}$	$y_{18tmp} = x_3h_5$	$y_{13} = x_3h_1 + x_2h_5$
14		$y_{14} = x_2h_2 + y_{14tmp}$	$y_{19tmp} = x_3h_6$	$y_{14} = x_3h_2 + x_2h_6$
15		$y_{15} = x_2h_3 + y_{15tmp}$	$y_{20tmp} = x_3h_7$	$y_{15} = x_3h_3 + x_2h_7$
16	x_4	$y_{16} = x_3h_0 + y_{16tmp}$	$y_{21tmp} = x_4h_4$	$y_{16} = x_4h_0 + x_3h_4$
⋮	⋮	⋮	⋮	⋮

followed by a double-precision addition is $2T$ where T is the delay of the multiplier or the double-precision adder.

In the proposed design Fig.5.6(b), a high-speed, double-precision carry-save multiplier/accumulator (CS-MAC) developed in [50] is used as :

$$2^w\left(s_{out} + c_{out}\right) + l_{out} = a \times b + 2^w\left(s_{in} + c_{in}\right) + l_{in} \tag{5.28}$$

where s_{out}, c_{out} are the high-order sum and carry word output from the CS-MAC, l_{out} is the low order word output from the CS-MAC, s_{in}, c_{in} are the high-order sum and carry word input to the CS-MAC, and l_{in} is the low order word input to the CS-MAC.

A reduction in the computation time ranging from 20% to 50% compared with other schemes has been achieved using the CS-MAC, without a significant increase in the required area [50].

Table 5.2 shows the activity for PE_0 (grey circles) and PE_1 (blue circles) in Fig.

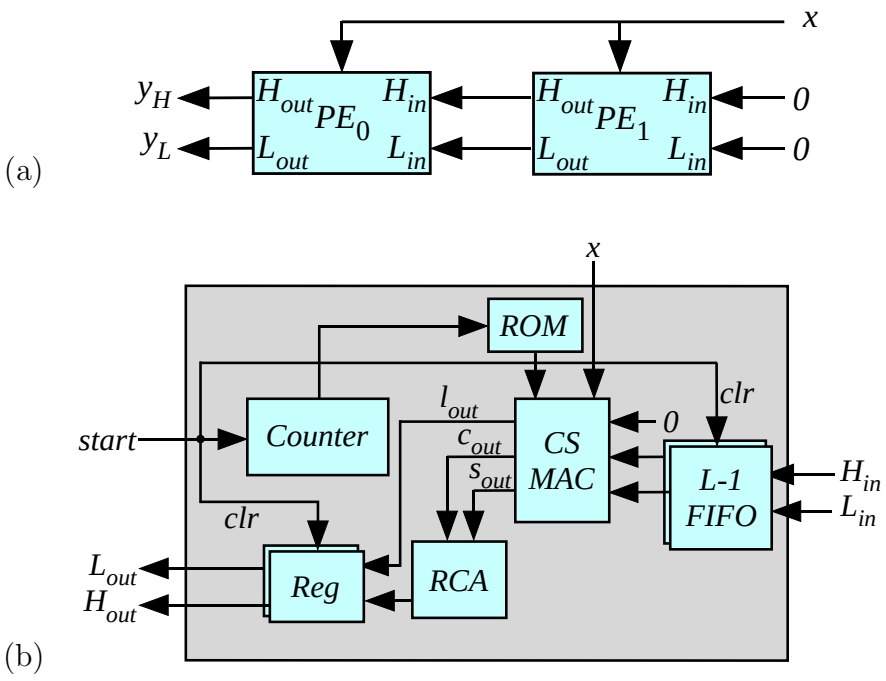


Figure 5.6: Systolic array for Design-Option #1. (a) The systolic array when $J = 8$ and $L = 4$. (b) PE details.

5.5 after applying the non-linear scheduling function in Eq. (5.25). Both PE_0 and PE_1 are active at each time step. The data output is obtained from PE_0. PE_1 performs the output partial results. These partial results are then forwarded to PE_0 after L-clock cycles.

The resulting systolic array and PE details can be obtained from Fig. 5.6 and Table 5.2, which are shown in Fig. 5.6. Figure 5.6(a) shows the resulting systolic array. Two PEs are shown since $J = 8$ and $L = 4$. The input data x is broadcast and the double-precision output $y_{out} = H_{out}2^w + L_{out}$ is pipelined. The output data flows from PE_1 to PE_0. Figure 5.6(b) shows the PE details. The Ripple Carry Adder (RCA), shown in Fig. 5.6(b), is used to reduce the high-order sum and carry output s_{out}, c_{out} from the CS-MAC to the high order word H_{out}.

An $L - 1$-stage First-in-First-out (FIFO) buffer is required to delay the data coming from the previous PE before it is applied to the CS-MAC.

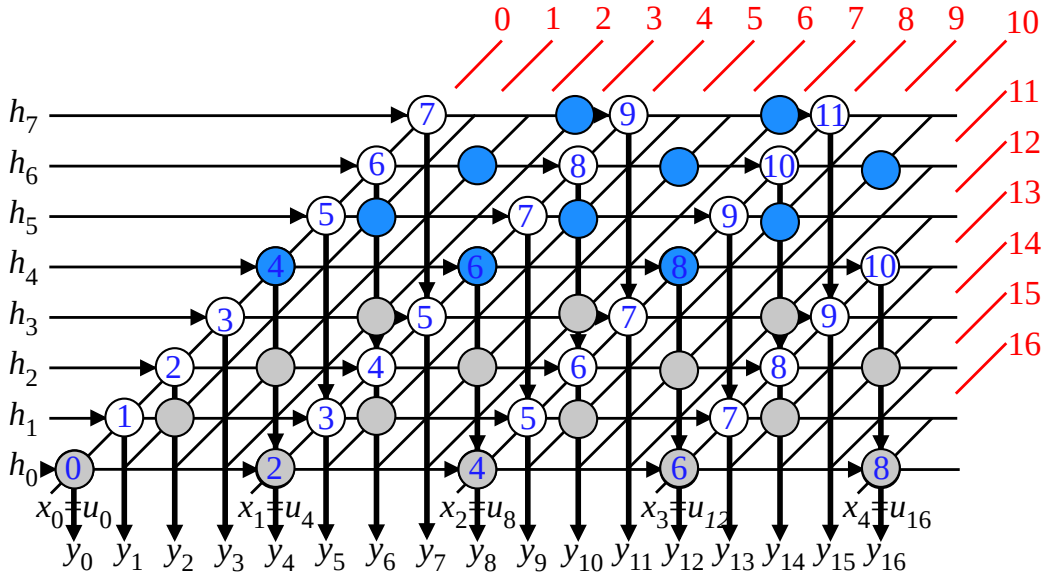


Figure 5.7: The DAG for Design-Option #2: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_3 = [1 \quad -1]^T$.

5.6.2 Design-Option #2: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_3 = [1 \quad -1]^T$

In this design-Option, the same scheduling function $\boldsymbol{s}_1$ is used with different linear projection direction $\boldsymbol{d}_3$. The corresponding projection matrix will be:

$$\boldsymbol{P}_3 \quad = \quad [1 \quad 1] \tag{5.29}$$

By substituting $\boldsymbol{P}_3$ in Eq. (5.19), we obtain:

$$k = i + j \tag{5.30}$$

Figure 5.7 shows the DAG for Design-Option #2 together with the equitemporal zones and the PE indices as shown inside the circles. It is noticed that the total number of PEs is infinite in this case since index i extends to ∞. However, the non-linear scheduling and projection functions, equations (5.25) and (5.26) respectively, could be used. It is noticed that only two PEs are active at any given time step.

As the same non-linear scheduling function $t'(\boldsymbol{p})$ and the non-linear projection direction k' of Design-Option #1 are used, The systolic array for Design-Option #2 will be the same as in Fig. 5.6.

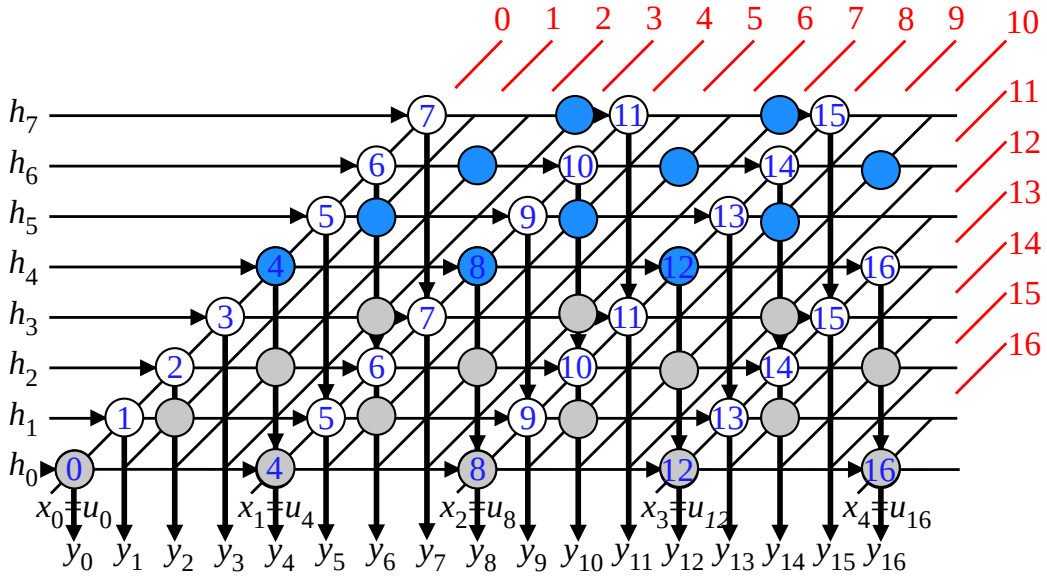


Figure 5.8: The DAG for Design-Option #3: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$.

5.6.3 Design-Option #3: using $\boldsymbol{s}_1 = [1 \quad -1]$ and $\boldsymbol{d}_4 = [0 \quad 1]^T$

The projection matrix corresponding to the linear projection direction $\boldsymbol{d}_4$ will be:

$$\boldsymbol{P}_4 \quad = \quad [1 \quad 0] \tag{5.31}$$

By substituting $\boldsymbol{P}_4$ in Eq. (5.19), we obtain:

$$k = i \tag{5.32}$$

The total number of PEs is infinite since index i extends to ∞.

Figure 5.8 shows the DAG for Design-Option #3 together with the equitemporal zones and the PE indices as shown inside the circles. The same non-linear node projection used in Eq. (5.26) could be used for Design-Options #1 and #2. It is also noticed that only two PEs are active at any given time step.

As the same non-linear scheduling function $t'(\boldsymbol{p})$ and the non-linear projection direction k' of Design-Option #1 are used, The systolic array for Design-Option #3 will be the same as in Fig. 5.6.

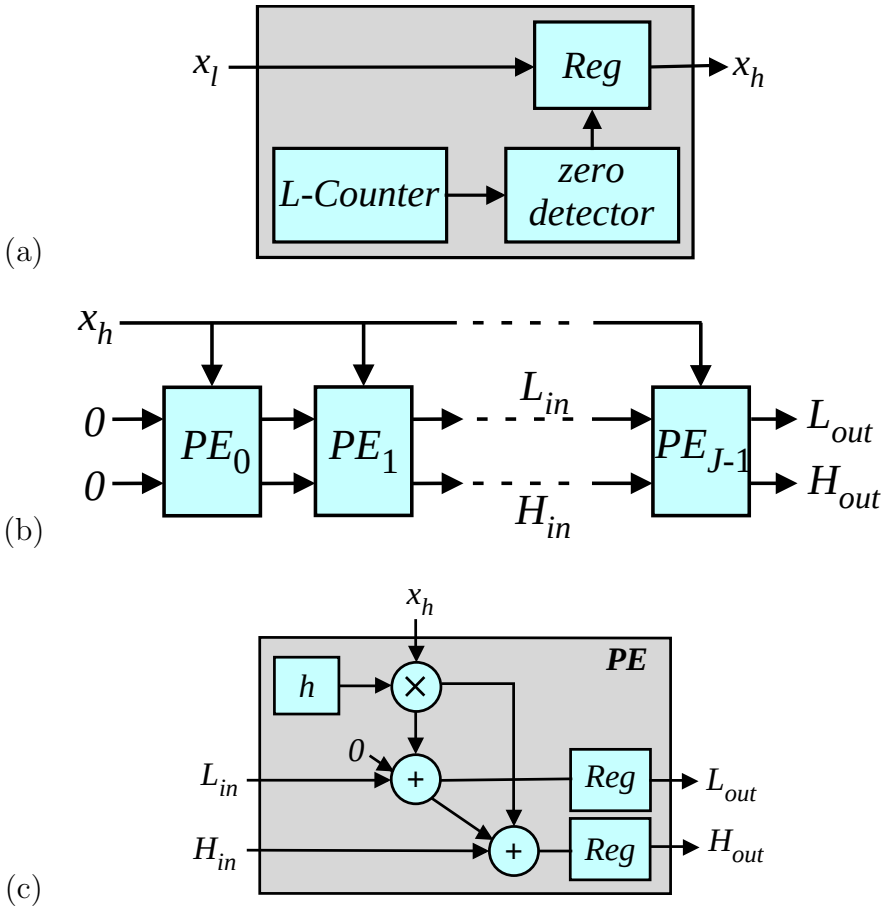


Figure 5.9: The conventional interpolator implementation. (a) The upsampler details. (b) The systolic array of the conventional FIR filter. (c) The conventional FIR filter PE details.

5.7 Hardware Implementation Results

This section discuss the hardware implementations of the interpolator. The conventional implementation of the interpolator is discussed in Sec. 5.7.1. The proposed implementation of the interpolator is discussed in Sec. 5.7.2.

The performance, in terms of hardware resources and speed, for both designs are reported for different values of interpolation factor L.

FPGA implementations of the conventional and the proposed designs were implemented using VHDL on Xilinx Virtex 6 ML605 development board.

5.7.1 Conventional Design Implementation

The conventional design is consists of an upsampler followed by an anti-imaging filter. Figure 5.9 shows the systolic array design of the conventional interpolator.

Figure 5.9(a) shows the upsampler details which accepts the low rate data samples x_l and produces data samples x_h with L higher rate.

Figure 5.9(b) shows the systolic array of the conventional FIR filter, which corresponds to a form 2 implementation of an FIR filter. The number of PEs is equal to the number of filter coefficient J. We assume the double precision mode of operation, where data exchanged between the PEs is in double precision format. This is explicitly shown in the figure as the partial sums L and H.

Figure 5.9(c) shows the conventional FIR filter PE details. The PE consists of a traditional multiplier with double-precision output. The multiplier multiplies the input data samples x_h by the dedicated FIR filter coefficient stored in a register. The lower bits of the multiplier output are added to the lower bits of the previous PE output using a single-precision carry-propagate (SPCP) adder to produce the lower bits of the PE output. Similarly, the higher bits of the multiplier output are added to the higher bits of the previous PE output using a SPCP adder to produce the higher bits of the PE output.

5.7.2 Proposed Interpolator Design Implementation

Figure 5.6 shows the proposed systolic array design which will be implemented. We notice from the figure that the design requires broadcast of the input data and neighbour-to-neighbour communication for the intermediate data. This ensures the maximum processing speed.

5.7.3 Implementation Results

The conventional and proposed designs performed interpolation by 2, 4, 8, etc (I-2, I-4, I-8, etc). The number of anti-imaging FIR filter coefficients were chosen as 8, 16, 32 and 64 in order to see the effect of the number of filter coefficients on the performance.

Table 5.3 shows the performance (speed, LUTs, Registers, etc.) of the conventional and proposed designs for different values of $L = 2$, 4, and 8 for an FIR filter with $J = 8$. It is noticed that the conventional design shows almost the same performance

Table 5.3: Comparison between resource utilization for conventional and proposed designs for different values of $L = 2$, 4, and 8 for the case of $J = 8$.

Structure of interpolator	Conv.	Proposed		
(logic utilization)	I-2/4/8	I-2	I-4	I-8
# Slice Registers	195	80	42	10
# Slice LUTs	1129	432	140	71
# fully used LUT-FF pairs	129	49	28	10
# bonded IOBs	91	26	26	26
Maximum Frequency (MHz)	153	328	331	740
Delay (ns)	6.524	3.041	3.014	1.350

regardless of the interpolation factor L. This is because the same FIR filter used in each interpolator. The impact of the upsampler on the performance is minimal since a one-bit counter is needed for interpolation by 2, a two-bit counter is needed for interpolation by 4 and a three-bit counter is needed for interpolation by 8. On the other hand, the performance of the proposed design depends on the interpolation factor. This is because the proposed design combines the FIR and upsampler operations and only the necessary computations are performed. The hardware resources of the proposed design are 61.7% less than the conventional design for the worst case when $L = 2$. Better savings are achieved for $L > 2$. The speed of the proposed design is 114.5% faster than the conventional design.

Table 5.4 shows the performance of the conventional and proposed designs for different values of $L = 2$, 4, 8 and 16 for an FIR with $J = 16$. The performance exhibit the same characteristics discussed in relation to Table 5.3. The hardware resources needed naturally increase for both implementations for all values of L since the number of filter coefficients is doubled. The hardware resources of the proposed design are 51.1% less than the conventional design for the worst case when $L = 2$. Better savings are achieved for $L > 2$. The speed of the proposed design is 85.4% faster than the conventional design.

Table 5.4: Comparison between resource utilization for conventional and proposed interpolators for different values of $L = 2$, 4, 8, and 16 for the case of $J = 16$.

Structure of interpolator	Conv.	Proposed			
(logic utilization)	I-2/4/8	I-2	I-4	I-8	I-16
# Slice Registers	387	202	106	43	10
# Slice LUTs	2207	1079	302	186	71
# fully used LUT-FF pairs	257	114	61	36	10
# bonded IOBs	155	26	26	26	26
Maximum Frequency (MHz)	145	294	331	270	740
Delay (ns)	6.866	3.395	3.014	3.702	1.350

Table 5.5 shows the performance of the conventional and proposed designs for different values of $L = 2$, 4, 8, 16 and 32 for an FIR with $J = 32$. The performance exhibit the same characteristics discussed in relation to Tables 5.3 and 5.4 where $J = 8$ and 16 respectively. The hardware resources needed naturally increases for all designs since the number of filter coefficients is increased. The hardware resources of the proposed design are 39.9% less than the conventional design for the worst case when $L = 2$. Better savings are achieved for $L > 2$. The speed of the proposed design is 63.9% faster than the conventional design.

Table 5.6 shows the performance of the conventional and proposed designs for different values of $L = 2$, 4, 8, 16, 32 and 64 for an FIR with $J = 64$. The performance exhibit the same characteristics discussed in relation to Tables 5.3, 5.4 and 5.5 where $J = 8$, 16 and 32 respectively. The hardware resources needed naturally increases for all designs since the number of filter coefficients is increased. The hardware resources of the proposed design are 56.6% less than the conventional design for the worst case when $L = 2$. Better savings are achieved for $L > 2$. The speed of the proposed design is 63.9% faster than the conventional design.

Table 5.5: Comparison between resource utilization for conventional and proposed designs for different values of $L = 2$, 4, 8, 16, and 32 for the case of $J = 32$.

Structure of interpolator	Conv.	Proposed				
(logic utilization)	I-2/4/8	I-2	I-4	I-8	I-16	I-32
# Slice Registers	772	455	236	95	51	16
# Slice LUTs	4373	2459	622	390	186	95
# fully used LUT-FF pairs	513	247	119	84	39	11
# bonded IOBs	283	26	26	26	26	26
Maximum Frequency (MHz)	144	294	319	270	270	337
Delay (ns)	6.920	3.393	3.134	3.694	3.702	2.967

Table 5.6: Comparison between resource utilization for conventional and proposed designs for different values of $L = 2$, 4, 8, 16, 32, and 64 for the case of $J = 64$.

Structure of interpolator	Conv.	Proposed					
(logic utilization)	I-2/4/8	I-2	I-4	I-8	I-16	I-32	I-64
# Slice Registers	1538	967	490	188	103	70	11
# Slice LUTs	8600	5168	1248	797	390	206	98
# fully used LUT-FF pairs	1025	516	243	181	87	47	11
# bonded IOBs	539	26	26	26	26	26	26
Maximum Frequency (MHz)	141	294	300	270	270	231	260
Delay (ns)	7.083	3.395	3.329	3.694	3.694	4.322	3.840

5.8 Design Complexity Comparison

We will use the conventional design as a benchmark for comparing the hardware performance figures of the conventional design shown in Fig. 5.9 and the design

proposed in this chapter.

From Tables 5.3–5.6 in Section 5.7, several general conclusions could be made:

1. The speed of the proposed design are at least 63.9% faster than the conventional design for a given number of FIR filter coefficients and for all interpolation factors.

2. The hardware resources of the proposed interpolator design are 61.7% less than the conventional design. This is true regardless of the number of FIR filter coefficients.

3. The hardware required for the proposed interpolator design in case of interpolation by 2 (I-2) consumes the most area compared to the other proposed interpolators in cases of interpolation by 4 (I-4), interpolation by 8 (I-8), etc. This is true regardless of the number of FIR filter coefficients.

4. In the conventional designs, the hardware resources depend only on the number of FIR filter coefficients and change by less than 0.5% when the interpolation factor changes.

5. The number of FPGA LUTs needed in distributed arithmetic is $\mathcal{O}(J \times 2^{2w})$. In this work, the number of FPGA LUTs needed is $\mathcal{O}(J \times w^2/L)$.

6. The number of threads or PEs needed in thread decomposition is $\mathcal{O}(J)$. In this work, the number of threads or PEs needed is $\mathcal{O}(J/L)$.

5.9 Conclusions

This chapter presented several contributions for the systolic array design space exploration of interpolators. A systematic methodology for systolic array design space exploration of interpolators was used since the interpolator algorithm is a RIA. Best hardware results are obtained when the interpolator utilizes an FIR filter whose number of coefficients is an integer multiple of the interpolation factor. Alternatively, the filter coefficients could be zero padded so that this limitation is avoided. The upsampler and the anti-imaging FIR filter are combined into a unified dependence graph (DG). Different data scheduling and projection operations were developed.

Nine different systolic array design options were obtained and evaluated. Design-Options #1, #2 and #3 produced the fastest possible clock speeds. The three design

options map to only one systolic array which uses a high-speed double-precision carry-save multiplier/accumulator (CS-MAC) to increase speed of arithmetic operations.

The proposed systolic array design and the conventional interpolator, were implemented in FPGA hardware to verify their functionality and compare their performance. FPGA implementations for the conventional and proposed designs confirm that the proposed interpolator implementation requires no more than 61.7% of the hardware resources required in the conventional design and are at least 63.9% faster than the conventional design. The results show that the proposed systolic array implementation performed better than the conventional implementation for all interpolation factors and for all values of FIR filter coefficients.

Chapter 6

Efficient FPGA Implementation of the Beamformer

In this chapter, the overall beamformer FPGA implementation is constructed based on the analysis of efficient systolic arrays designs of the beamformer building blocks. The implemented overall structure is then validated to ensure its proper operation. Further, the implementation performance is evaluated in terms of accuracy and error analysis in comparison to the MATLAB simulations.

In the previous chapters, the systolic arrays design space exploration for the basic building blocks of the beamformer (decimator, interpolator, and 2-D BB BF filter) were performed. For each beamformer building blocks, the most efficient systolic arrays design in terms of fastest clock speed was selected and implemented using FPFA.

6.1 Efficient FPGA Implementation of the Beamformer

As mentioned before in Chapter 2, the BB BF using NAs and multirate techniques consists of a number of channels as shown in Fig. 2.5. Each channel contains the basic building blocks of the beamformer.

The implementation of the overall beamforming system is performed through three steps. First, the implementations of the speed optimized systolic arrays designs for each of the building blocks that was proposed in Figs. #3.6, #4.8, and #5.6 in Chapters 3, 4, and 5 , respectively are connected together to implement one beam-

former channel as shown in Fig. 6.1. Second, the channels of the entire system are

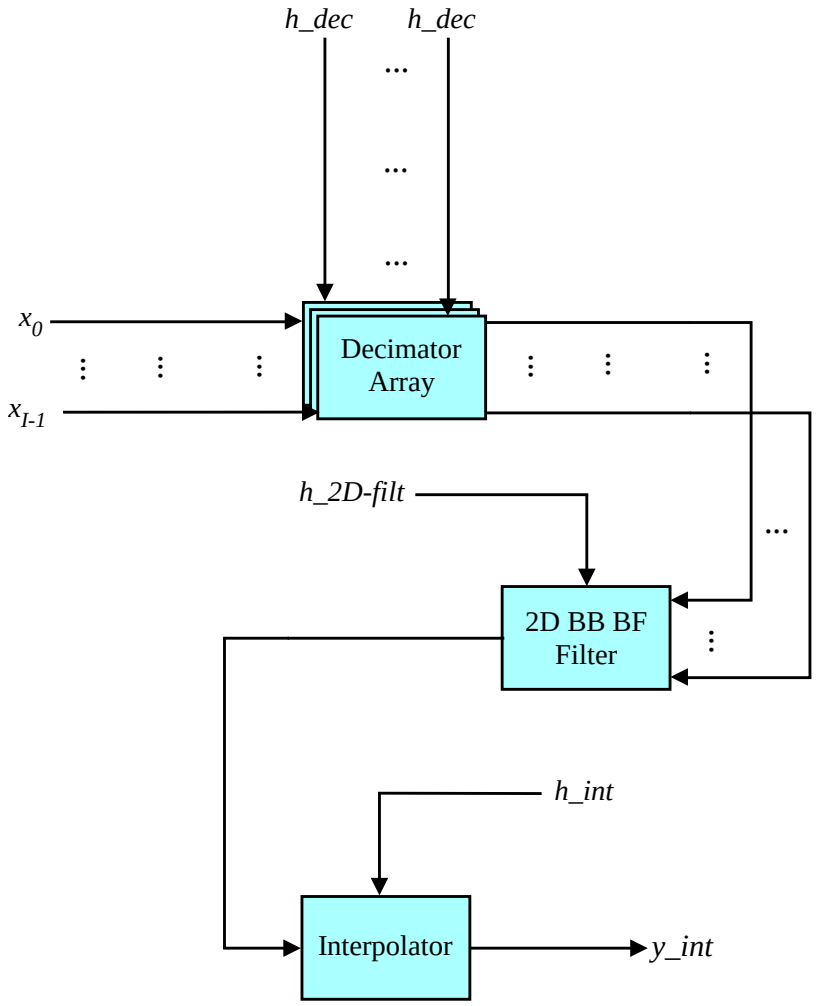


Figure 6.1: Single beamformer channel diagram.

constructed using the architecture shown in Fig. 6.1 using the proper decimation and interpolation factors for each channel. Finally, the obtained channels outputs are added together after applying a propoer delays to produce the beamformer output.

6.1.1 FPGA Implementation Validation

In order to validate the proper operation for the FPGA implemented overall structure of the beamformer, a three-step strategy have been used: (a) implement a single channel of beamforming system using FPGA; (b) perform finite-precision MATLAB model of the FPGA implementation; (c) compare the results of steps (a) and (b).

Validation Strategy: Single Channel FPGA Implementation

All the beamformer channels consist of the same building blocks. The difference between channels can be described using the following distinguishing factors:

- The value of the decimation/interpolation factor. Thus the number of PEs constituting each decimator/interpolator might differ but the structure of all PEs is the same.
- The values of the anti-aliasing FIR filter coefficients of the decimators/interpolatros. However, the coefficients ROMs all have the same size.

In this first validation step, only one channel will be implemented in FPGA. The selected channel is implemented and simulated using FPGA Xilinx Vivado Design Suite. In this implementation, the third channel of the beamformer is considered where the decimation and the interpolation factors equal 4 ($M = L = 4$). This channel consists of an array with 21 sensors that are uniformly spaced from $-10d$ to $10d$, where d equals to $4\lambda_h/2$. The same two BB PWs received from two different directions; one as the desired PW (DOA = 35°) and the other one as an interference (DOA= -50°), were used.

The ROS of the 2-D DFT of the signals received by the 3^{rd} subarray is shown in Fig. 6.2, where the spectrum of the signals is equal to 1 within $\pi/20 \leq \omega_{ct} \leq 4\pi/5$. The aliasing due to spatial subsampling can be observed in the upper-right and lower-left corners of Fig. 6.2.

The signals at this point are used as inputs to the implemented channel. The FIR filter coefficients of the analysis and synthesis filters, used in Chapter 2, are used in the decimator array and interpolator stages. The coefficients of the 2-D FIR TF which designed in Chapter 2 to have the passband containing the ROS of the desired signal at the decimator array outputs are used. Fig. 6.3 shows the 2-D passband of the TF used.

VHDL testbenches were designed for each block and functional simulations in Vivado logic simulator were executed. The input data was written in a file and then supplied to the testbench.

The hardware implementation used finite-precision (finite word-length) two's-complement fixed-point format for both the input signals and filter coefficients. The testbench data were obtained from the floating-point full-precision MATLAB (64-bits) data that were used in Chapter 2.

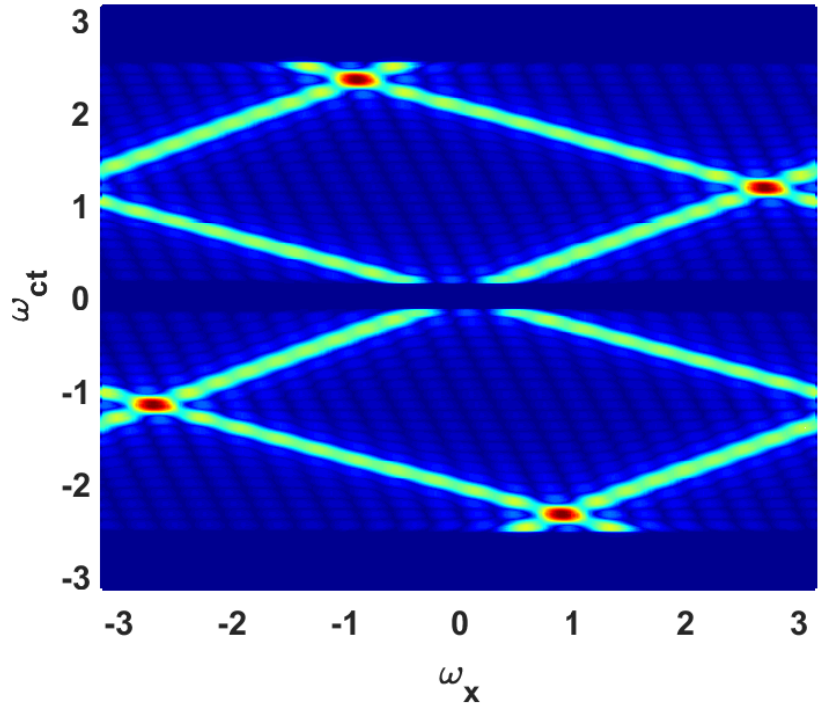


Figure 6.2: ROS of the 2-D FT of the signals received by the 3^{rd} subarray.

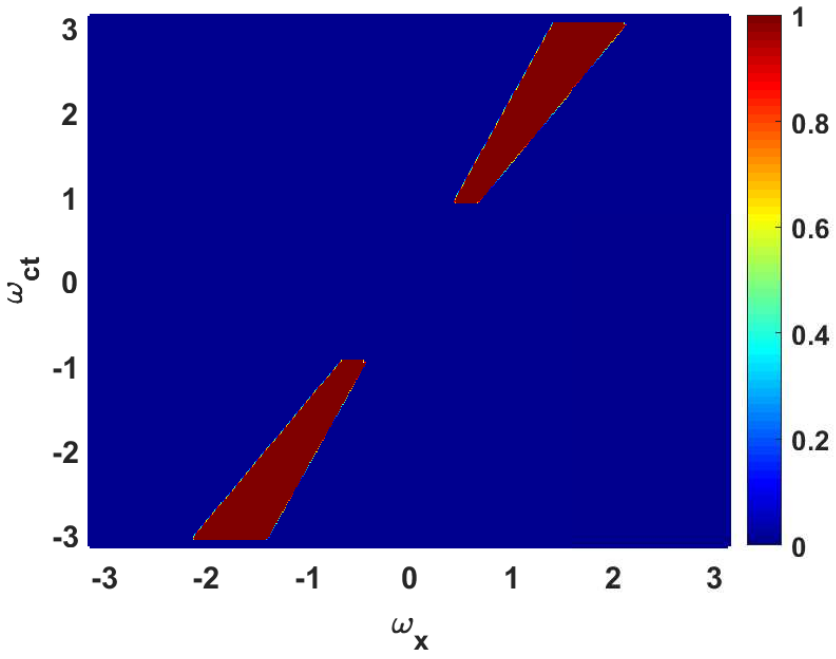


Figure 6.3: 2-D passband of the trapezoidal filter.

The full-precision data are multiplied by a scaling factor and then quantized to obtain the fixed-point values. The scaling factor is chosen such that the dynamic range of the input samples could be stored in W-bits 2's complement and in addition to avoid the overflow through the partial calculation. The scaling process that is performed prior to the quantization results in more efficient use of the available bits (optimally exploiting dynamic range of the quantizer), which in turn increase the system realization accuracy. A power of two scaling factor is used to minimize the number of arithmetic operations. The scaled values are converted to the two's-complement format and finally truncated to the required finite-precision (W-bits). For our FPGA implementations, finite-precision with $W = 16$ were used.

It should be noted that our building blocks used 16-bit single-precision data as input and produced 32-bit double-precision output data. Therefore connecting the building blocks together required truncating the least 16-bits of each module output. This truncation operation produces a certain error to the system, and this error are analyzed later in Section 6.3.

The filter coefficients are similarly expressed in form of 16-bits 2's-complement fixed point. Figure 6.4 shows the scaling, quantization and truncation for the inputs/outputs at the different points of the channel stages. The shaded blocks represent the main components of the channel, the triangular shapes represent the scaling operation that is applied to the samples values at the point before the triangles. The Q blocks represent the quantization operation that coverts the input data and the filters coefficients to the 2's complement finite-precision fixed-point format. The T blocks represent the truncation operation that truncates least significant bits (LSBs) and keeps the most significant 16-bits of the data between the blocks. The symbols k and represents the scaling factors.

The data used in the hardware simulations are the channel input data as well as the FIR filter coefficients for the decimators, 2-D BB BF, and interpolator. Referring to Fig. 6.4, and Table 6.1, the input data applied to the channel are scaled by scaling factor $K_{x1} = 2^{15}$. The decimators, 2-D BB BF, and interpolator FIR filter coefficients are scaled by scaling factors $K_{h4} = 2^{18}$, $K_{h10} = 2^{18}$, and $K_{h16} = 2^{18}$, respectively. The quantization is applied to the scaled stimuli to produce the required 2's complement 16-bits format.

The double-precision outputs obtained from the decimator array, 2-D BB BF filter, and interpolator are converted to single-precision data. This is effectively a two-step process. The first is right shifting the output by 16 bits which is equivalent to scaling

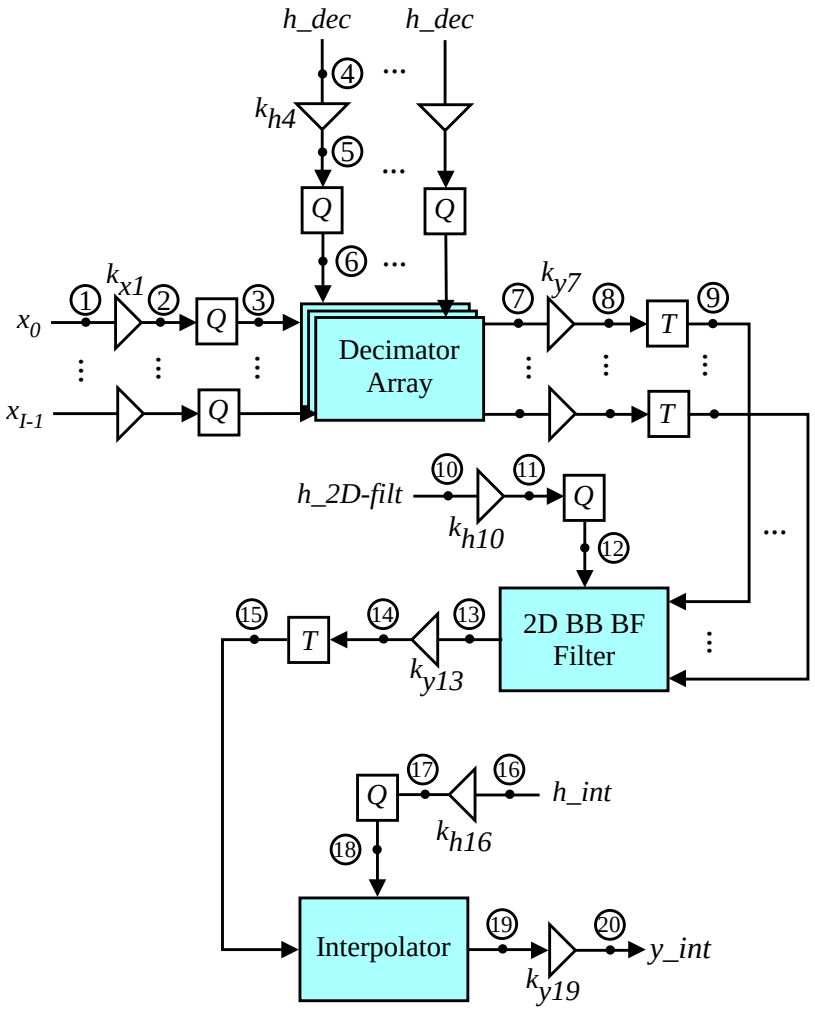


Figure 6.4: Finite-precision fixed point implementation diagram.

by K_y, shown in Fig. 6.4 and Table 6.1. The second step is discarding the least significant 16-bit digits by the truncation operation.

Table 6.1 summarizes the values of the scaling factors that are used through the channel.

In each testbench, all data sequences were read at each rising edge of a 100 MHz clock. The outputs of each stage are stored in a text files. Figure 6.5 shows snapshots of the simulated output traces for each stage of the FPGA implementation of the 3^{rd} beamformer channel. Figure 6.5-a shows the simulated output traces of the decimator array stage. Figure 6.5-b shows the simulated output traces of the 2-D BF filter stage. Figure 6.5-c shows the simulated output traces of the interpolator array stage which is the overall output of the 3^{rd} channel.

Table 6.1: The scaling factors for the system shown in Fig. 6.4.

Scaling Factor	**Value**
k_{x1}	2^{15}
k_{h4}	2^{18}
k_{y7}	2^{-16}
k_{h10}	2^{19}
k_{y13}	2^{-18}
k_{h16}	2^{16}
k_{y19}	2^{-34}

Validation Strategy: Single Channel Finite-Precision MATLAB Model

In this second step, MATLAB is used to develop a finite-precision model of the FPGA implementation presented in the first step. This model will be used to simulate and validate the FPGA implementation.

The data format used in the FPGA implementation was adopted for the MATLAB finite-precision model. Overflow checks were performed on the results of all the convolution operations and and data was sign extended to increase the dynamic range similarly as it was done for the FPGA implementation in the first step.

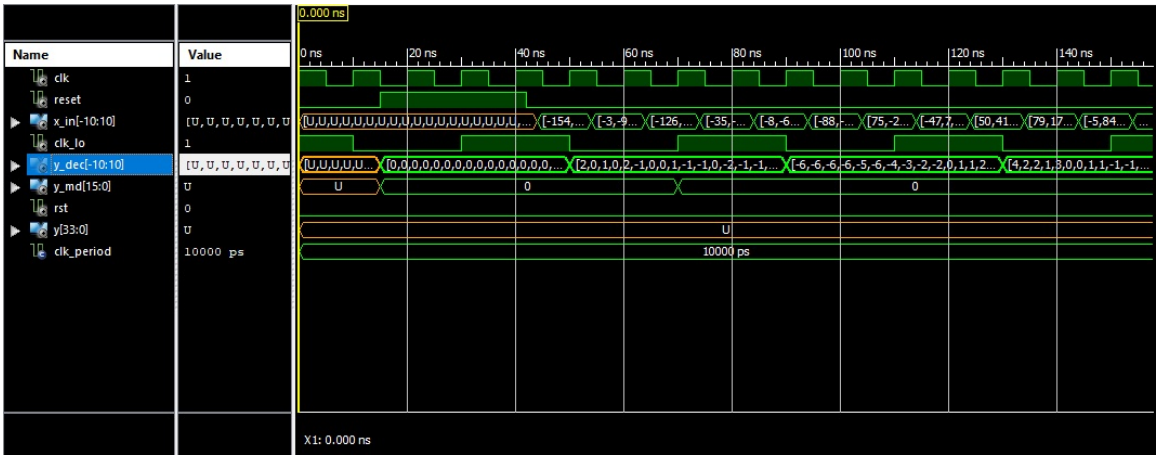
The output data produced by the decimator array are double-precision and scaled by $K_{x-dec} \times K_{h-dec} = 2^{33}$. The obtained outputs of the decimator array are stored in a text file. This was used in order to compare these results with the Hardware implementation results. These output data are considered as the input to 2-D BB BF filter and it should be in single-precision format. In order to do that, the output data are then truncated to the most 16-bits of the obtained results by discarding the remaining LSBs (scaling by $K_{y-dec} = 2^{-16}$), to be ready as a single-precision input to the 2-D BB BF filter (scaled by 2^{17}). This step is used to mimic the conversion from double to single precision format in the FPGA implementation.

For the sake of plotting the outputs of the decimator array, the truncated outputs are scaled by $1/(K_{x-dec} \times K_{h-dec} \times K_{y-dec}) = 2^{-17}$.

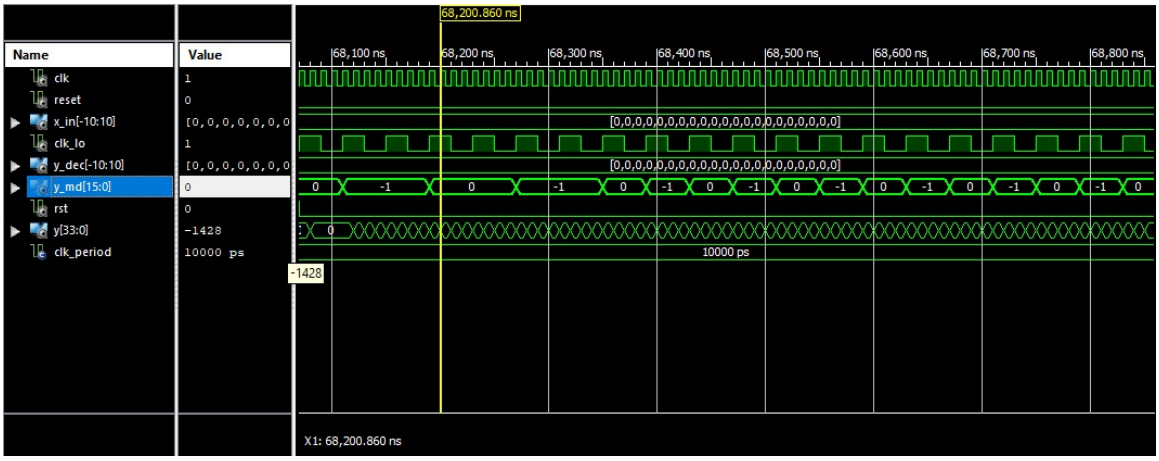
Fig. 6.6 shows the outputs of the finite-precision decimator array resulting in the spatio-temporal (ST), domain.

Fig. 6.7 shows the ROS of the 2-D Fourier transform (FT) for the decimators resulting outputs.

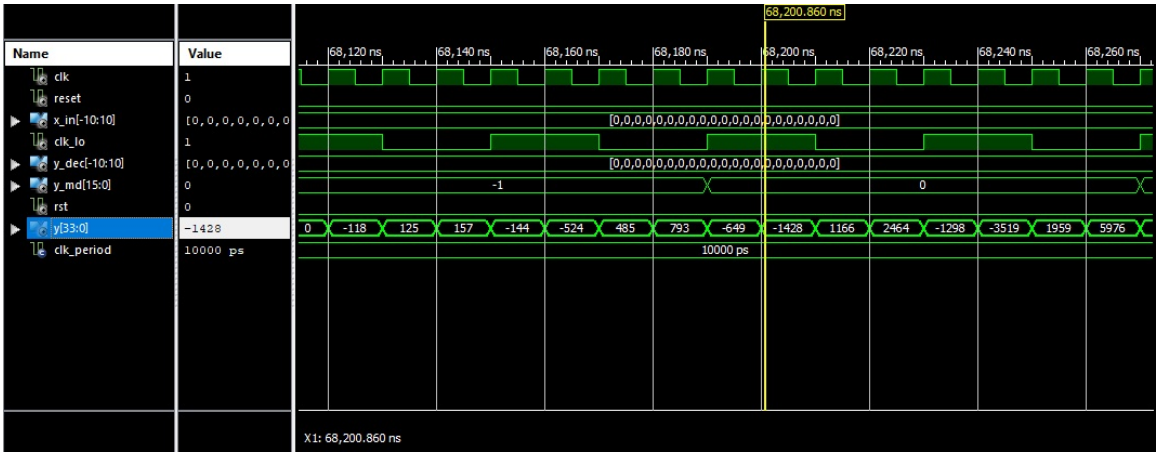
The 2-D BB BF filter coefficients are scaled by scaling factor $K_{h-2D-filt} = 2^{19}$. The



(a)



(b)



(c)

Figure 6.5: Simulations for the 3^{rd} channel FPGA implementations. (a) Decimator array outputs. (b) 2-D BF filter outputs. (c) Interpolator (channel) outputs.

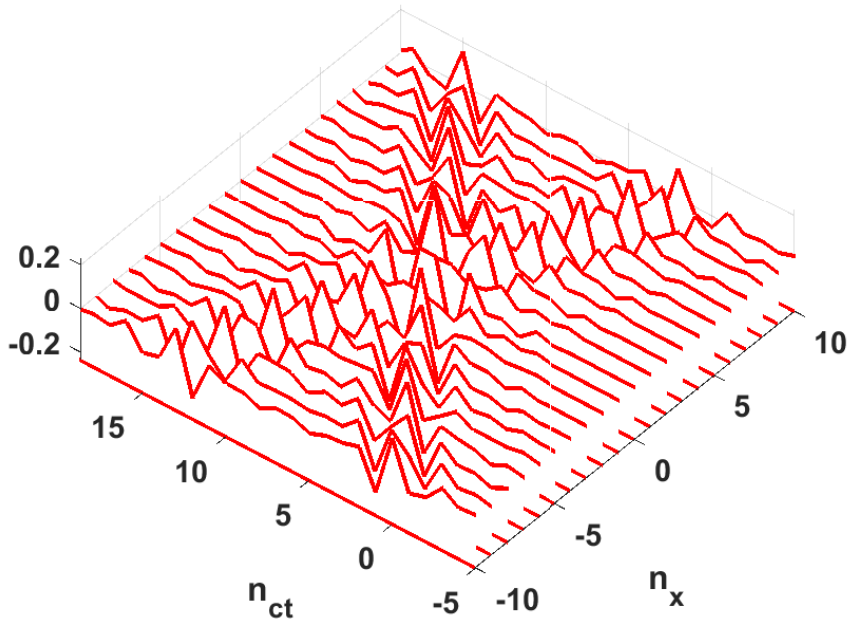


Figure 6.6: ST representation of the finite-precision decimator array outputs.

center sensor output is selected as the output of the TF. During the overflow check, it is found that the output data produced by the 2-D BB BF filter need a sign extension by 2-bits. So the obtained output data are scaled by $2^{17} \times K_{h-2D-filt} \times 2^2 = 2^{38}$ The obtained outputs of the 2-D BB BF filter are stored in a text file. Here, the outputs of the 2-D BB BF filter should be truncated by discarding the 18-LSBs (scaling by $K_{y-2D-filt} = 2^{-18}$) to be ready as a 16-bits input to the interpolator (scaled by 2^{20}).

For plotting the center sensor output of the TF the truncated outputs are scaled by $1/(2^{17} \times K_{h-2D-filt} \times 2^2 \times K_{y-2D-filt}) = 2^{-20}$ is shown in Fig. 6.8.

The interpolator FIR filter coefficients are scaled by scaling factor $K_{h-int} = 2^{16}$. The obtained interpolator outputs are 32-bits and is scaled by $K_{y-int} = 2^{36}$ due to the cumulative scaling and truncation at each of the channel stages. The obtained outputs of the interpolator are stored in a text file.

Figure 6.9 shows the interpolator output sampled after scaling by 2^{-36}.

Table 6.2 summarizes the cumulative scaling factors and the word-length for each of the points indicated in Fig. 6.4.

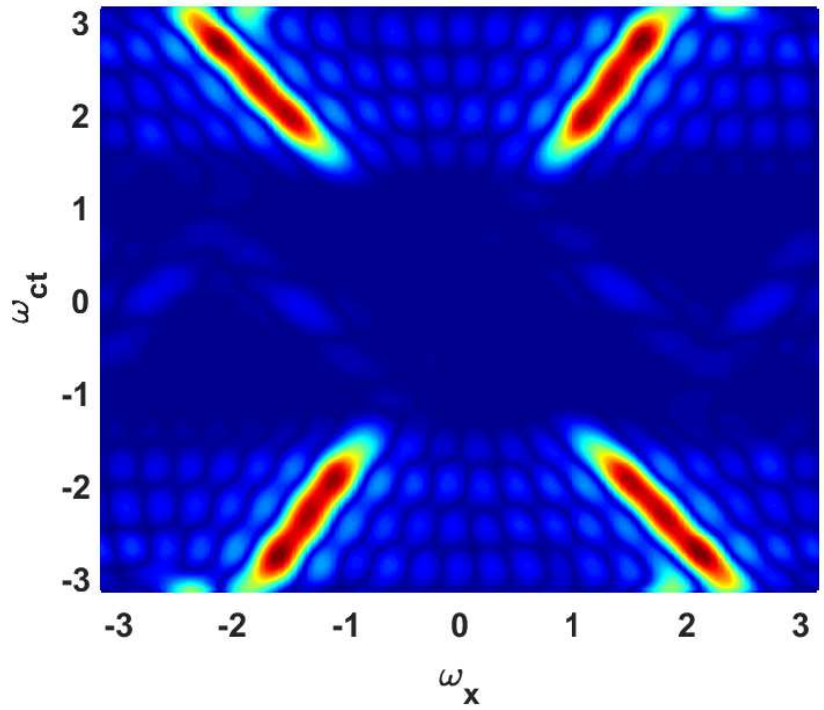


Figure 6.7: ST frequency domain representation of the finite-precision decimator array outputs.

Validation Strategy: Comparing FPGA and MATLAB Results

In the third step, the results obtained from FPGA implementations of the first step and MATLAB simulations of the second step for each building block are compared individually. This comparison aids in validating the proposed systolic arrays. The comparison is performed on the data that were saved in the text files as mentioned in the previous subsections.

The text files are collected in MATLAB workspace. A m script is then run to compare the results from FPGA and MATLAB simulations.

The comparisons show that the output results obtained from both FPGA implementations and MATLAB simulations are identical for each stage of the beamformer channel. This provides an assurance that the results from the FPGA implementations match well with the results from the finite precision MATLAB models.

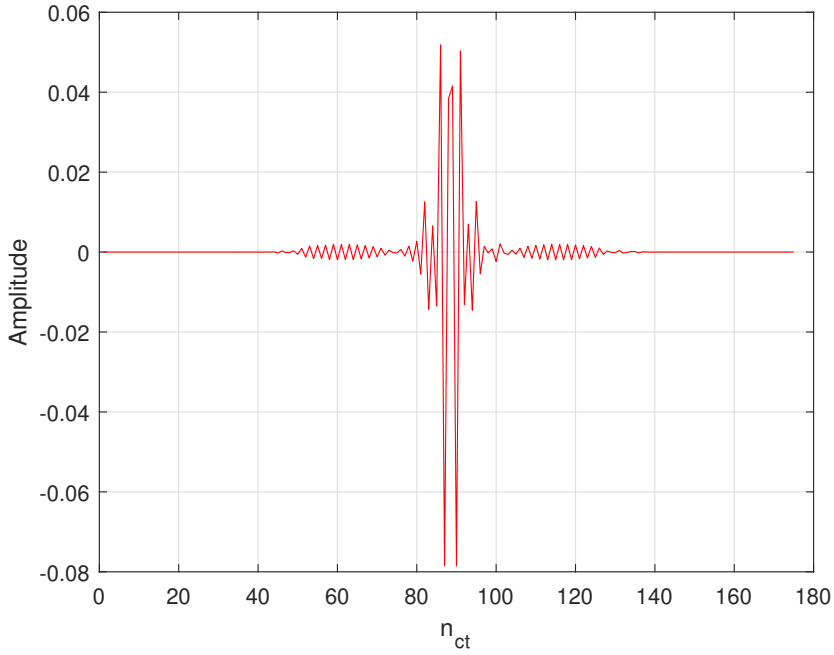


Figure 6.8: Finite-precision 2-D BB BF filter center censor output.

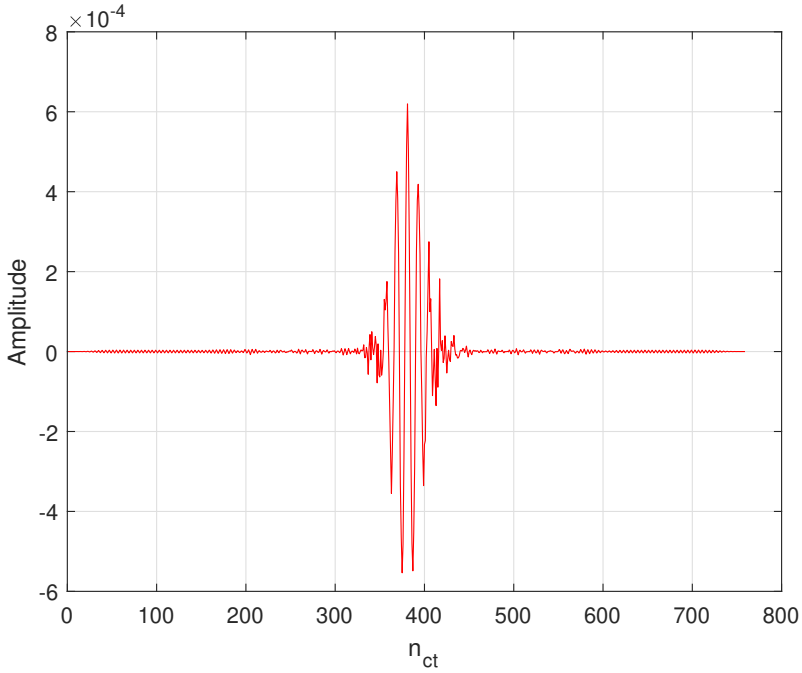


Figure 6.9: Finite-precision interpolator output.

6.2 Finite-Precision Model Accuracy

After establishing that the single channel FPGA implementations and the finite-precision MATLAB models are giving identical results, the implementation perfor-

Table 6.2: The cumulative scaling factors and the number of bits for Fig. 6.4 points.

Point	**Cumulative Scaling**	**Word-length**	**Comments**
1	1	64-bits	
2	2^{15}	64-bits	
3	2^{15}	16-bits	
4	1	64-bits	
5	2^{18}	64-bits	
6	2^{18}	16-bits	
7	2^{33}	32-bits	
8	2^{17}	32-bits	
9	2^{17}	16-bits	
10	1	64-bits	
11	2^{19}	64-bits	
12	2^{19}	16-bits	
13	2^{38}	34-bits	sign extended by 2-bits
14	2^{20}	34-bits	
15	2^{20}	16-bits	
16	1	64-bits	
17	2^{16}	64-bits	
18	2^{16}	16-bits	
19	2^{36}	32-bits	
20	1	32-bits	

mance analysis in terms of accuracy will be conducted. The accuracy analysis are performed by calculating the signal-to-error ratio (SER) between the finite-precision MATLAB models, that simulates the FPGA implementations, and full-precision MATLAB simulations, introduced in Chapter 2, as a benchmark. The third channel full-precision MATLAB simulations is used as follows: The signals at the outputs of the third channel antenna arrays are applied to the input of the third channel decimator array. The ROS of the resulting outputs of the decimator array in 2-D space-time domain are shown in Fig. 6.10.

The ROS of the 2-D FT of the resulting outputs of the decimator array are shown in Fig. 6.11.

The decimator array output signals are passed through the 2-D BB BF filter which performs the pass the SOI and reject the interfering signal, with group delay of $N_T F$. Figure 6.12 shows the TF center sensor output signal.

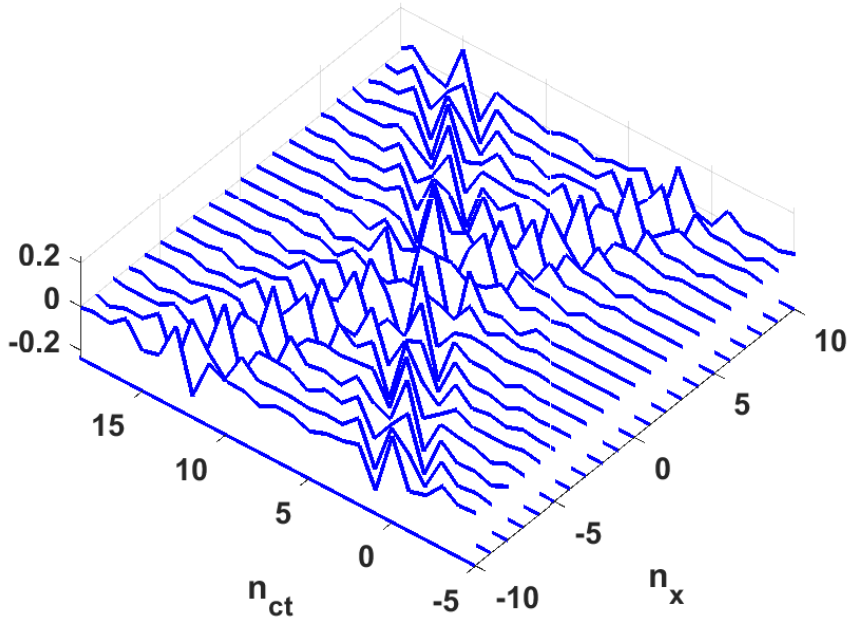


Figure 6.10: ST representation of the full-precision decimator array outputs.

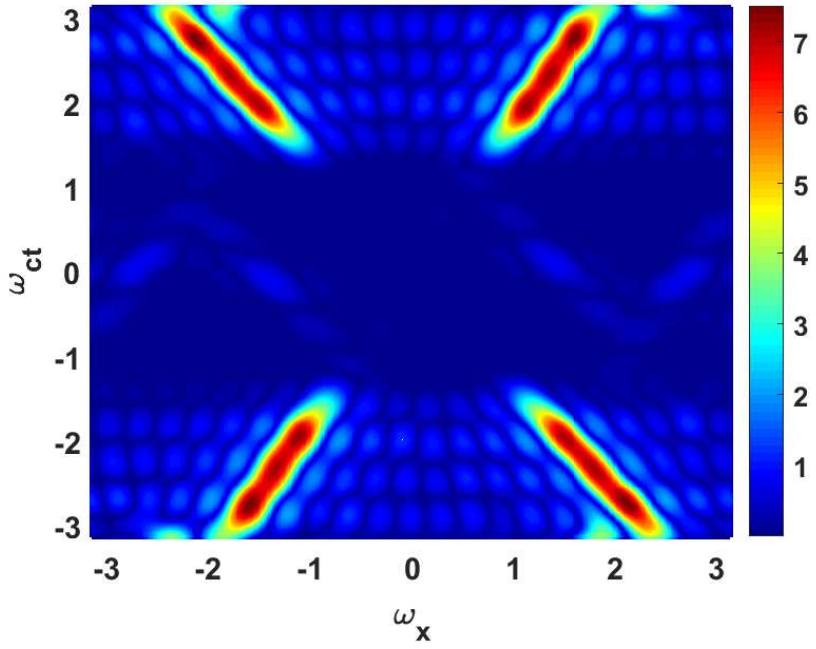


Figure 6.11: ST frequency domain representation of the full-precision decimator array outputs.

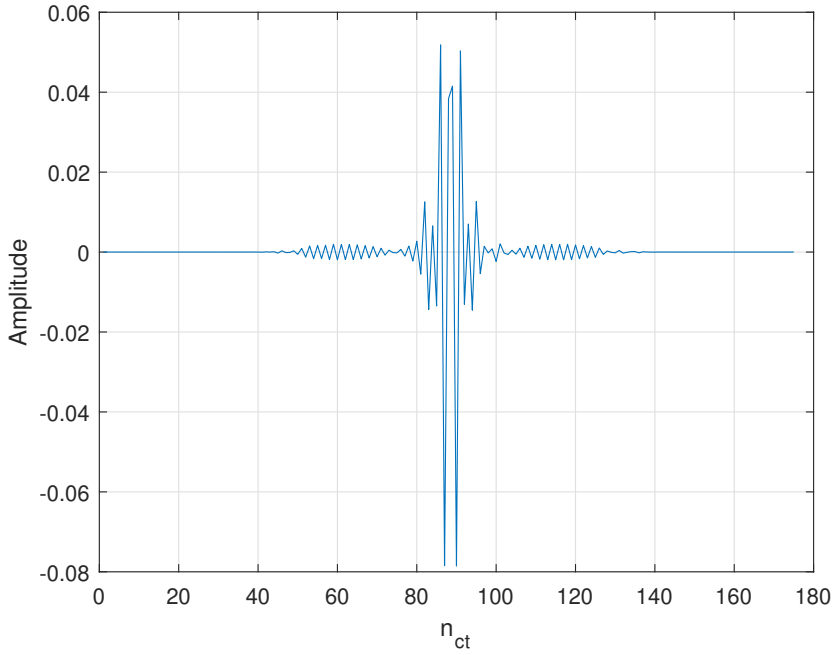


Figure 6.12: Full-precision 2-D BB BF filter output.

The TF center sensor output signal is passed through the interpolator which performs both upsampling by 4 and anti-imaging filtering, with group delay of N_I.

Figure 6.13 shows the interpolator output signal. This signal produces a perfect reconstructed SOI when added to the outputs of the remaining channels interpolators.

6.3 Error sources

In the validation channel shown in Fig. 6.4, two sources that induce errors in multiple locations will be considered.

1. Quantization errors: occur at points #3, 6, 12, and 18 due to the applied quantization process on the data samples. Quantization errors can be kept as low as desired by using a sufficiently large number of bits.

2. Truncation errors: appears at points # 9, and 15 due the truncation operations that are performed between the basic building blocks.

Arithmetic errors: There is another source of errors occurs due to the multiplication operations for the filtering operations in the three types of blocks (decimators,

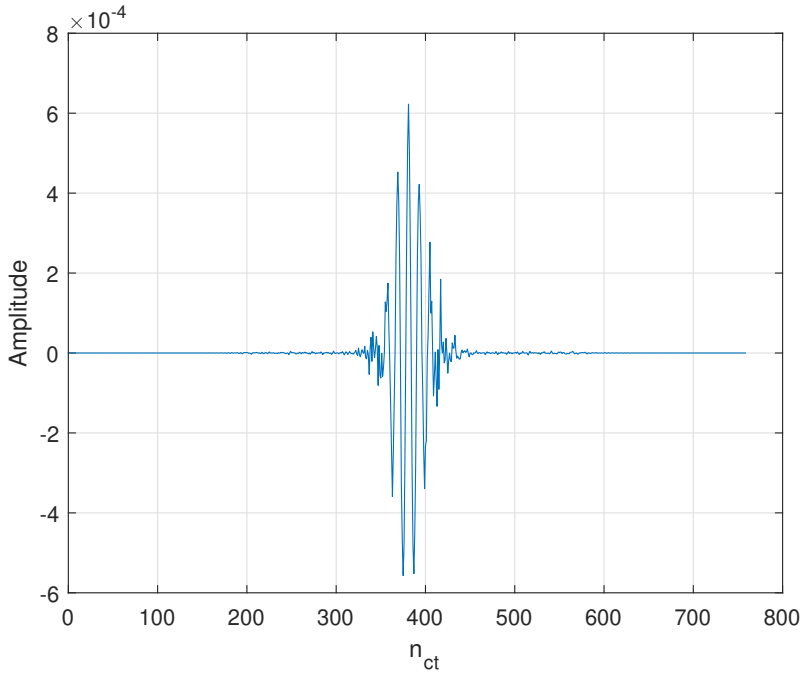


Figure 6.13: Full-precision interpolator output.

2-D filters, and interpolator). A filter that requires J multiplications will produce errors in the least significant $\log_2 J$ bits. These errors supposed to appear at points # 7, 13, and 19. However, in the proposed systolic array designs of beamforming system, arithmetic errors not present because the MACs used are of single-precision inputs and double-precision outputs which allows to perform all the multiplication and accumulation operations in double-precision format.

The absence of arithmetic errors in the proposed designs, greatly enhances the signal-to-error ratio compared to the standard beamformer implementations. This is explored in detail in Sec. 6.3.1.

6.3.1 Finite Word-Length Effect on the Signal-to-Error Ratio (SER)

The effect of the finite word-length on the SER is studied by calculating the SER at the inputs and the outputs of each stage for the all beamformer channels. The SER is calculated specifically at the points # 3, 7, 9, 13, 15, and 30 that are shown in Fig. 6.4. The SER is calculated with different word-lengths (8, 12, 16, 24, 32)-bits.

Figure 6.14 shows the effects of the word-length and the error that was generated

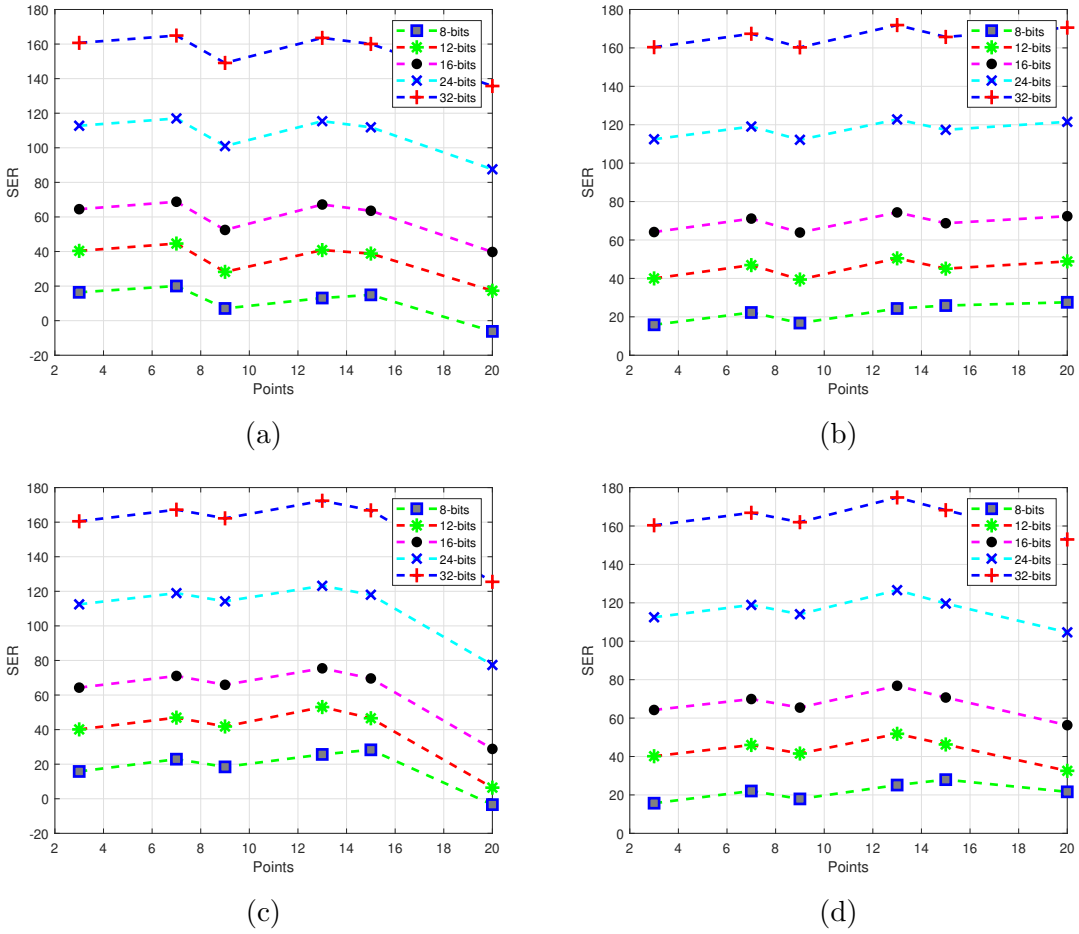


Figure 6.14: Finite word-length and generated errors effects on SER using proposed MAC. (a) Channel #1. (b) Channel #2. (c) Channel #3. (d) Channel #4.

in each channel stages on the values of SER. For the all channels the following can be observed:

1. The SER improves as the word-length increases.

2. The SER deteriorates between channel stages due to truncation effect.

3. The SER improves inside the decimators and 2-D filter stages because the FIR filters used in these stages are filtering out the unwanted signals with their generated errors.

For the interpolator stage, the SER in channel #2 improves as shown in Fig. 6.14-b and in channels #1, #3, and #4 decreases as shown in Fig. 6.14-a, 6.14-c, and 6.14-d. These changing effects are due to that the FIR filters used in the interpolators have

an amplification or attenuation gains to be able to achieve a perfectly reconstructed signals at the output of the beamformer.

From Fig. 6.14, it can be seen that the system will meet a satisfactory SER for the 16-bits and higher cases. However, with 8-bits case the SER decreses up to negative values in channels #1 (Fig. 6.14-a) and #3 (Fig. 6.14-c) and with 12-bits case the SER tends to be zero in channel #3 (Fig. 6.14-c). Thus, we see that at least 16-bits word-length are required for implementation of this class of beamforming systems.

6.3.2 The Effect of Using a Conventional MACs on the system SER

Traditionally, the multiplier-accumulator (MAC) used in the implementation of the FIR filters have the same input/output precision. For example, if at each sampling time the input signal represented by $b1$ digits is multiplied by a coefficient represented by $b2$ digits, a product having as many as $b1 + b2$ digits is generated. If a uniform register length should be used throughout the filter, each multiplier output must be rounded or truncated before processing can continue. The rounding or truncation operations propagate through the filter and give rise to output error commonly referred to as output round-off error.

In the proposed designs analysis in the previous section, a single-precision input double-precision output MAC were used. This means that all the partial calculation are performed in the double-precision format. Now, this way provides a round-off error free systems in addition to reduces the delay and the required hardware resources. In the cases where connecting components together are needed as in the beamformer channels, just the final outputs of each stage are truncated to match the single-precision input of the next stage.

For studying the effect of using the conventional MAC versus the double-precision MAC, the same experiments that were used in Section 6.3.1 are simulated for the beamformer implementation using conventional single-precision MACs.

Figure 6.15 shows finite word-length and generated errors due to using conventional MAC effects on the SER of the beamformer channels.

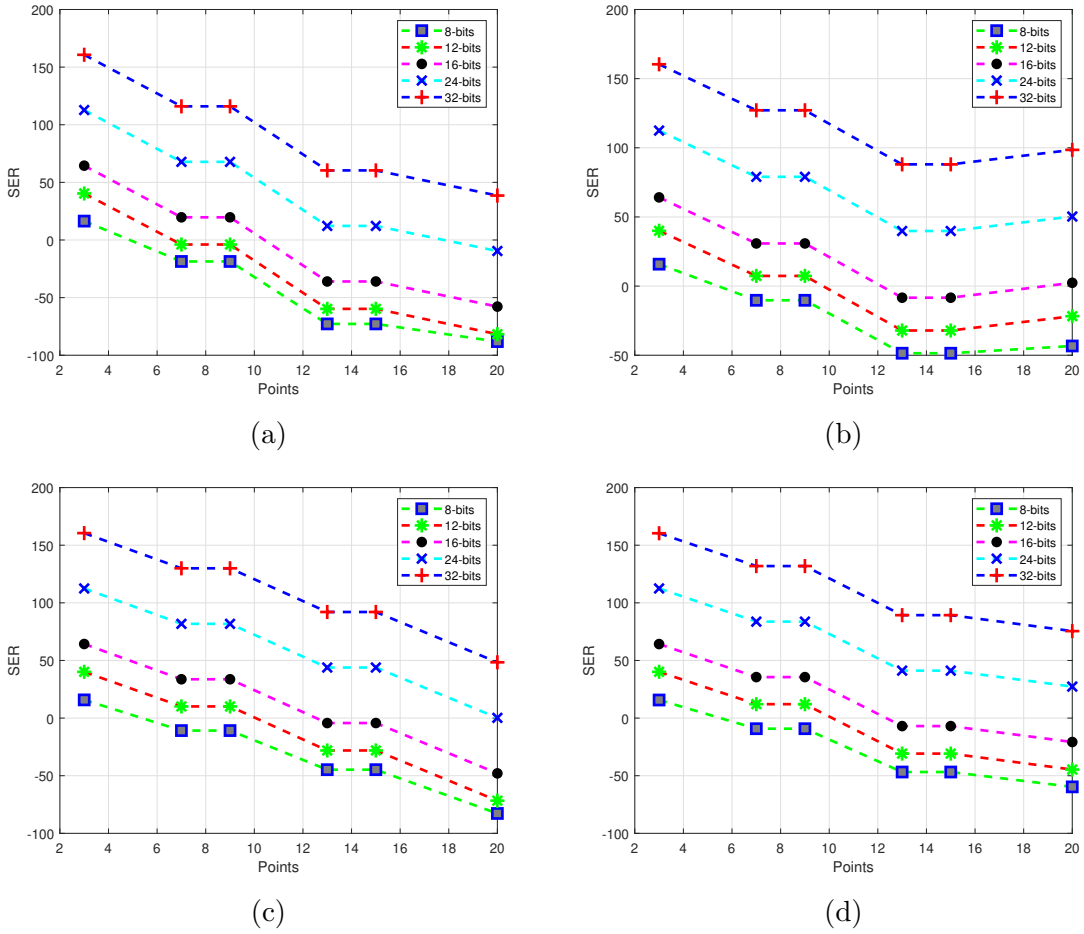


Figure 6.15: Finite word-length and generated errors effects on SER using conventional MAC. (a) Channel #1. (b) Channel #2. (c) Channel #3. (d) Channel #4.

6.4 Maximum Speed and Resources Utilization Estimation

In this section, the maximum speed and the total number of hardware resources (# slice registers and # LUTs) for the overall NA beamformer are estimated. The estimation is performed for the conventional and the proposed implementations in case of four channels and $J = 64$ for the decimators and interpolators and $I = 15$ and $J = 32$ for the 2-D filter as follows:

The maximum speed of the overall system is limited by the minimum frequency of the individual basic building blocks. Using Tables 3.6, 4.2, and 5.6, the maximum speed using the proposed implementation can be estimated to be 200 MHz and us-

ing the conventional implementation to be 74 MHz. So the maximum speed of the proposed implementation is 2.7 faster than that of the conventional implementation.

The total number of hardware resources can be estimated as follows:

$$A_t = N_{ch}[N_{ant}A_{dec} + A_{2D-filt} + A_{int}] \tag{6.1}$$

where N_{ch} represents the number of beamformer channels, N_{ant} represents the number of sensors in each subarray, A_{dec} represents the number of hardware resources in the decimator block, $A_{2D-filt}$ represents the number of hardware resources in the 2-D filter block, and A_{int} represents the number of hardware resources in the interpolator block.

Using Table 3.6, the total resources are calculated for the proposed decimators to be 225345, and the conventional decimators to be 365160. Using Table 4.2, the total resources are calculated for the proposed 2-D filter to be 265576, and the conventional 2-D filter to be 78832. Using Table 5.6, the total resources are calculated for the proposed interpolators to be 18996, and the conventional interpolators to be 40551. For the overall system, by substituting in Eq. (6.1) using the conventional implementation, we get the total resources equals 509917 and using proposed implementation, we get the total resources equals 484543. So the total resources required in the proposed implementation equals 1.05 of that are required in the conventional implementation.

6.5 Conclusions

A certain strategy is used to integrate the overall NA beamformer. The accuracy analysis is performed by calculating the SER at different locations along the BF channels and at different word-lengths. The SER results show that the acceptable accuracy of the implemented system is obtained with a word-length of 12-bits and that the quality of accuracy increases significantly with increased word-lengths. The FPGA implementation results confirm that the proposed processor array implementation are 2.7 faster with an increase of 5% of the hardware resources required by the conventional implementation.

Chapter 7

Closing the Speed Performance Gap Between Radar I/O Rates and Silicon Speed

In some radar applications that requires operating at high frequencies, consequently higher bandwidth beamformers are required.

The fastest possible designs for the NA beamformer basic building blocks that were obtained in the previous chapters performed well for medium-to-low frequencies. The upper frequency bound is limited by the delay of the MAC operation in each PE. The silicon speed is limited by the arithmetic logic unit (ALU) critical path of the processor. From Chapters 3, 4, and 5, we conclude that the decimator block is the slowest component with frequency about 200 MHz.

When higher operating frequencies are needed, a speed performance gap between the I/O rates of the ADC and hardware implementation exist is appeared. To address this issue, a new methodology is proposed in this chapter to close this speed performance gap. The new methodology is based on the systematic methodology that was used in Chapters 3, 4, and 5 and applied to the interpolator block as an example. The proposed methodology is verified and tested using MATLAB environment to ensure the proper operation.

7.1 The Proposed Methodology for Closing the speed performance gap

The methodology proposed in this section is general and applies to many RIAs. We consider applying this methodology for the interpolator as a working example. The proposed methodology specifies several steps. The methodology is applied after obtaining the DAG of the interpolator algorithm. Figure 7.1 for the case of $J = 16$, $L = 4$, and the scheduling function specified in Eq. (5.9).

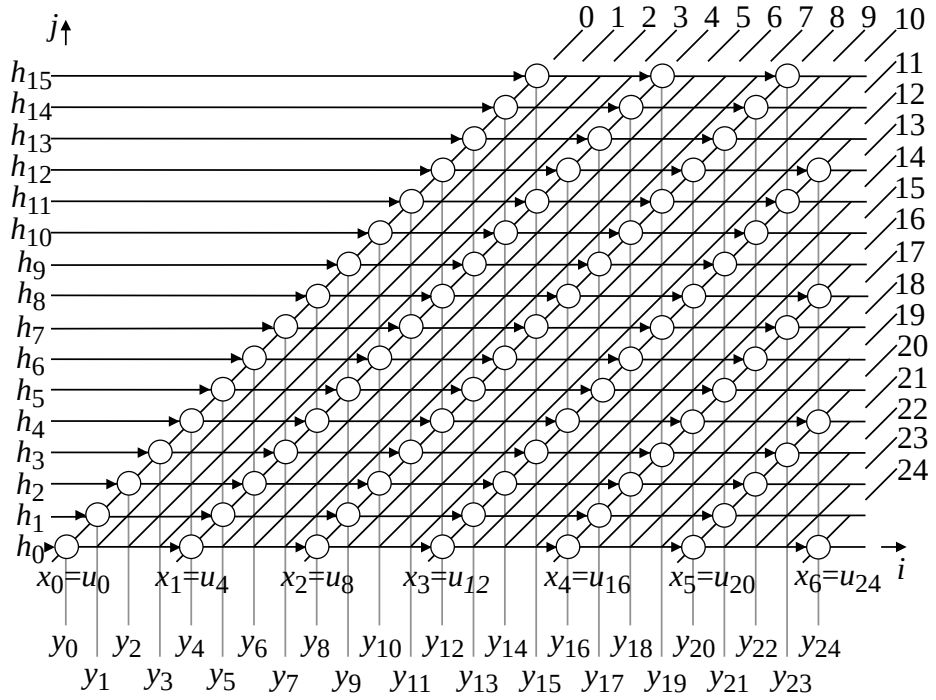


Figure 7.1: Interpolator DAG in case of $J = 16$ and $L = 4$.

The DAGs in Chapters 3, 4, and 5 assumed that the tasks performed in each node require one clock cycle to be completed. In Fig. 7.1, we assume that the tasks performed in each node require $D > 1$ clock cycles to be completed. We label the variable D the *dilation factor*.

Depending on the value of D, two situations could be encountered:

1. Case when $1 < D \leq L$. In this case, the reduction in the number of PEs performed in Chapter 5 will no longer apply. But data exchange between the communicating nodes is feasible since the input data is ready before it is required.

2. Case when $D > L$. In this case, data exchange between the communicating nodes is not feasible since the input data is not ready when it is required.

The following steps are performed after obtaining the DAG to make the interpolator I/O rate independent of the value of D as follows:

1. Partitioning of the DAG into sub-DAGs according to the value of D.
2. Re-timing the sub-DAGs (partitions) partial output results in order to reduce the number of simultaneous calculations.
3. Select the appropriate partial results obtained from the re-timer using an interconnection network.
4. Addition of the outputs from the interconnection network using carry-save accumulator.
5. Addition of the sum and carry outputs obtained from the carry-save accumulator to obtain the final interpolator output samples.

The methodology simulations were done on MATLAB environment using object oriented programming (OOP) to ensure simple coding and reduce the probability of software bugs.

7.2 Partitioning

In order to allow the interpolator to match the high data rates while each node takes dilation D clock cycles, we propose partitioning of the DAG. Partitioning of the DAG is proposed by dividing it into sub-DAGs such that the tasks in the nodes and the communications between the nodes take place at multiple clock cycles. This partitioning provides more time for the execution of the multiplication and addition operations. By summing the corresponding outputs of each partition together, the interpolator output samples can be obtained. In this way, the interpolator will be able to match the high data rates of the modern wideband radar systems.

In general the number of partitions P is given by:

$$P = \left\lceil \frac{D}{L} \right\rceil \tag{7.1}$$

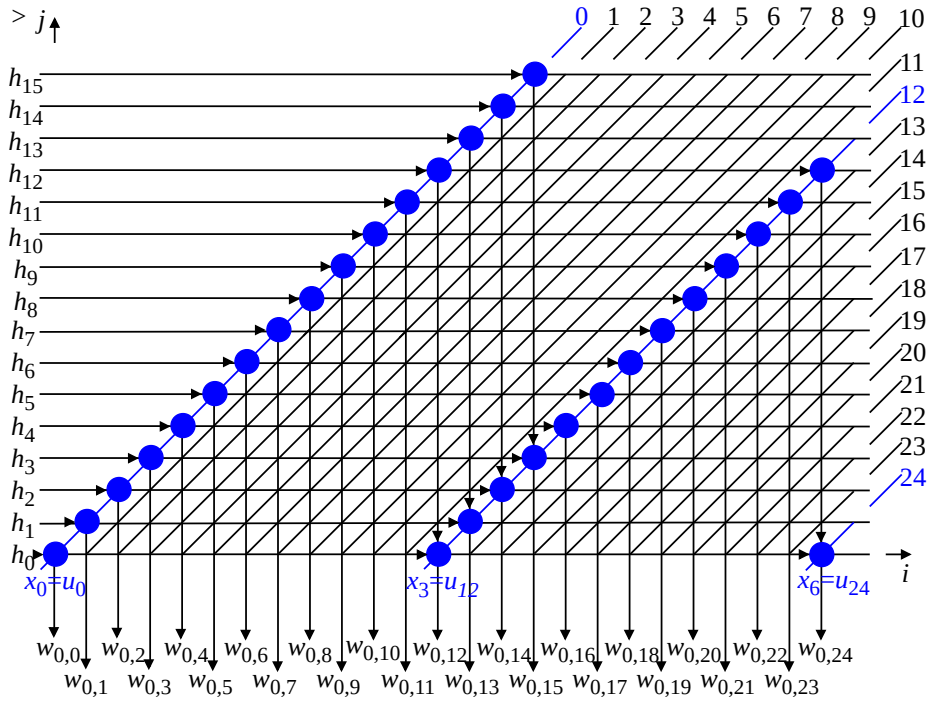


Figure 7.2: Interpolator partitioning #1 for the DAG in Fig. 7.1.

For example, if $L = 4$ and $D = 12$ then form 7.1 we get $P = 3$. Therefore, the DAG in Fig. 7.1 is partitioned into 3 sub-DAGs shown in Figs. 7.2, 7.3, and 7.4.

By observing the sub-DAGs it can be seen that the available time associated to each node is relaxed accordingly. This relaxed time allows the nodes to perform their tasks over a wider number of clock cycles. Notice that after partitioning, each node takes $D = 12$ clock cycles.

Figure 7.5 shows how the interpolator input x is distributed among the partitions and how the data from each partition is obtained.

The input data x is applied to the re-sampler operating with L-clock rate clk_L to distribute the input samples on the partitions. The partitioned input samples at the output of the downsampler are broadcast to each of the partitions PEs. The double-precision outputs obtained from each PE of each partition is collected in data buses to be ready for addition operation. Figure 7.5(b) shows the PE details. We notice from the figure that polyphase clocking is needed where the number of clock phases is the number of partitions. The assigned anti-imaging filter coefficients for each PE are stored in the register h. The same MAC that was used in the previous chapters are used here to performs the required multiplication and accumulation operations. In

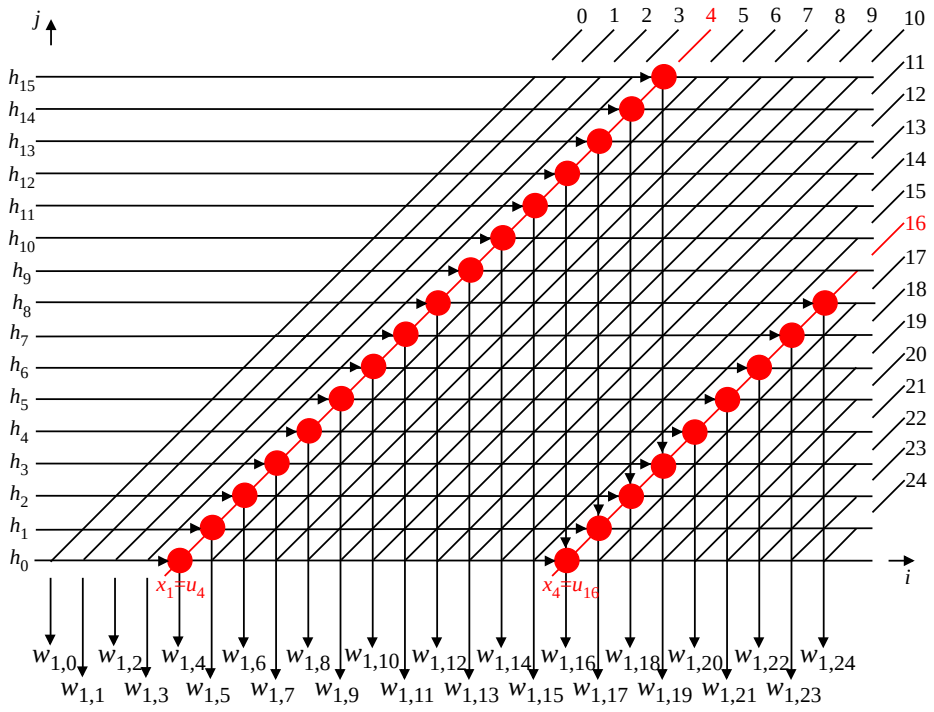


Figure 7.3: Interpolator partitioning #2 for the DAG in Fig. 7.1.

order to synchronize the PEs, each group of registers at the output of the MAC should be cleared at definite times according to the location of the PE in each partition.

Figure 7.6 shows the time indices at which the outputs obtained from each partition. OOP in MATLAB is used to obtain these results. It is noticed from the figure that each partition produces groups of PL partial results. This groups are separated in time by PL time steps. For a given output the adjacent partitions groups, the indices of each partial results in each partition are separated by L. and the time at which each and L time steps in the time index.

Careful observation of Fig. 7.6 reveals the following problems:

1. The groups of PL partial results from each single partition are obtained at the same time. If these partial results are added horizontally to obtain the interpolator final output samples, we will end up with an L successive groups of samples and each group have the same time as shown in Fig. 7.7. However, it is desired that successive output samples are obtained at each time step.

2. For a given output sample, the corresponding partial output results from each group have different indices. As an example, the output sample y_{10} requires the

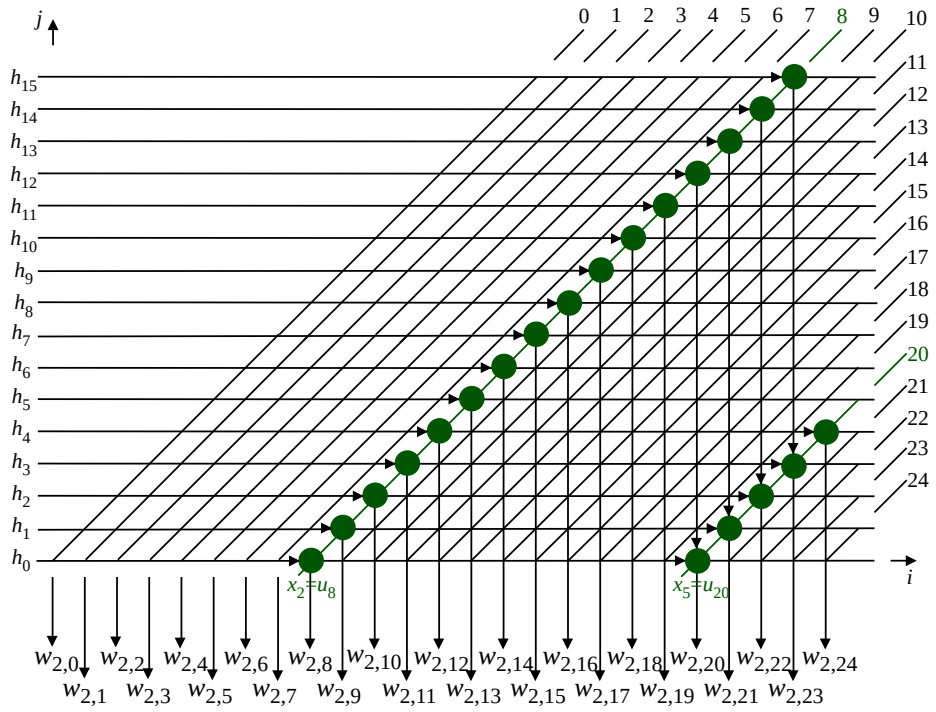


Figure 7.4: Interpolator partitioning #3 for the DAG in Fig. 7.1.

partial results that are obtained at different times as represented in Table 7.1.

Table 7.1: Example of the partial results that required by the output sample y_{10}.

Partition #	0	1	2
Partial result	$y_{0,10}$	$y_{1,6}$	$y_{2,2}$
Time	12	16	20

3. For a given output sample, the corresponding partial outputs of each partition are separated in time be L samples. However, the addition dilation $D > L$ and

Each of the previously stated problems require a specialized hardware to address them as will be illustrated in the following sections.

7.3 Re-timer

In order to address the observed problem of the interpolator output samples timing (problem #1) shown in Fig. 7.7, a re-timing should be applied to the partial results

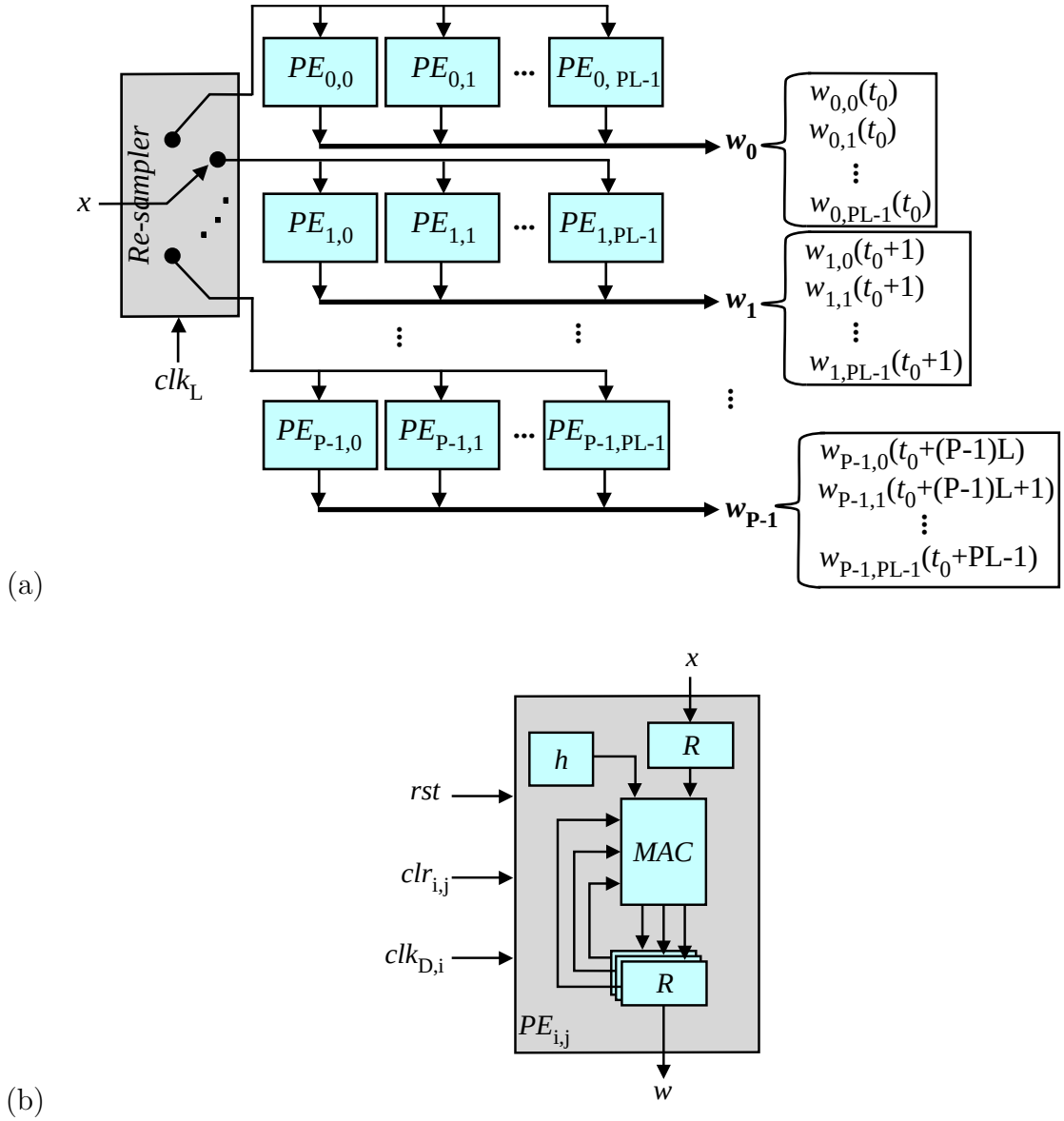


Figure 7.5: Partitioning block diagram. (a) Distribution of the interpolator inputs among the partitions. (b) PE details.

$\boldsymbol{w}$ obtained from the partitions to adjust the timing of the final interpolator output samples in order to be at each time step as shown in Fig. 7.8.

The re-timer consists of a group of P sawtooth delay elements as shown in Fig. 7.9. Each sawtooth delay element accepts the same time partial results of each partition and performs a triangular delay at each L samples.

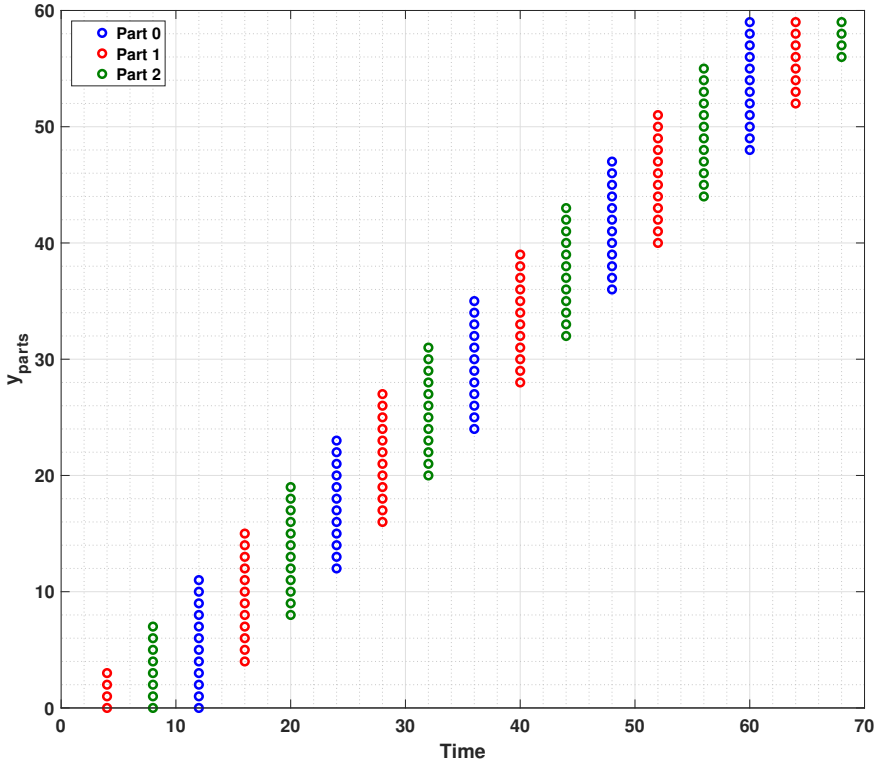


Figure 7.6: The outputs of the partitions in case of $L = 4$ and $P = 3$.

7.3.1 Interconnection Network

In order to address the observed problem of the different partial results indices explained in Table 7.1 (problem #2), an interconnection network should be used to select the appropriate partial results $\boldsymbol{u}_i$, $0 \leq i \leq P-1$ obtained from the re-timer shown in Fig. 7.9.

The re-timer generates a group of P^2L partial results at different times as represented in Fig. 7.9. These partial results generate a group of PL output samples y_i, $0 \leq i \leq PL-1$. The index i of y_i can be expressed as the relation:

$$i = q \times PL + j \tag{7.2}$$

where $q = \lfloor i/PL \rfloor$ and $j = i \mod PL$.

Each output sample y_i requires P dedicated partial results. The function of the interconnection network shown in Fig. 7.10 is to select these P partial results to be

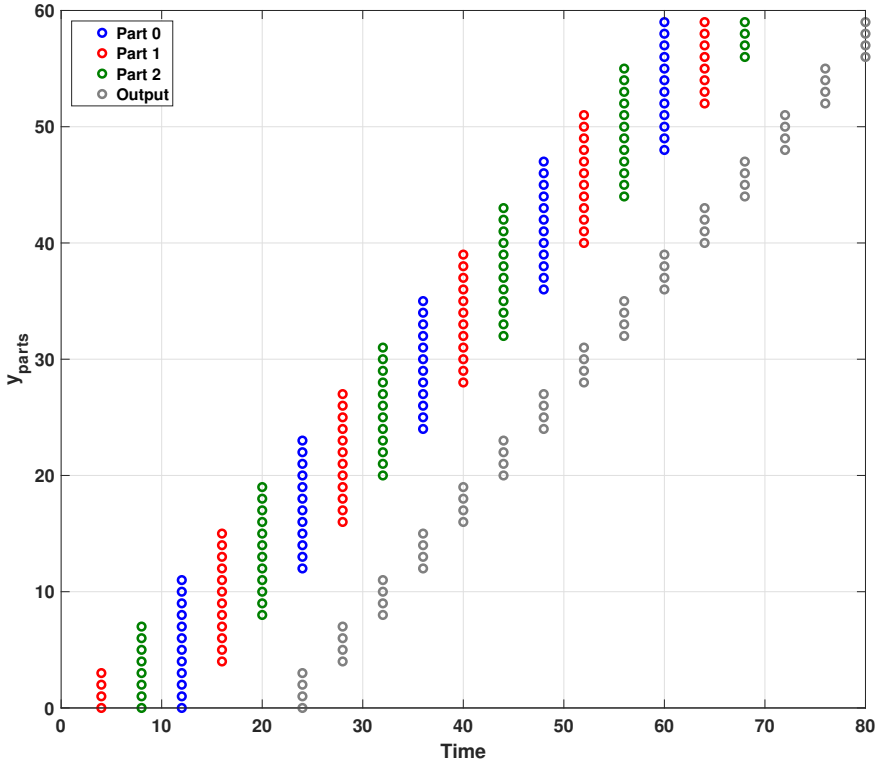


Figure 7.7: The timing for the addition of the partitions outputs in case of $L = 4$ and $P = 3$.

ready for the addition.

The interconnection network I/O relation can be represented as:

$$\begin{aligned} v_j &= w_{(k,l)}(t_j + j),\ w_{((k+1) \mod P, l-L)}(t_j + j + L), ..., \\ &\quad w_{((k+P-1) \mod P, l-(p-1)L)}(t_j + j + (p-1)L) \end{aligned} \tag{7.3}$$

where:

$$\begin{aligned} k &= (\left\lfloor \frac{j \mod PL}{L} \right\rfloor + 1) \mod P, \\ l &= (P-1)L + (j \mod L), \\ t_j &= q \times PL + (\left\lfloor \frac{j}{L} \right\rfloor L) + L + \delta, \\ \delta &= j \mod L \end{aligned}$$

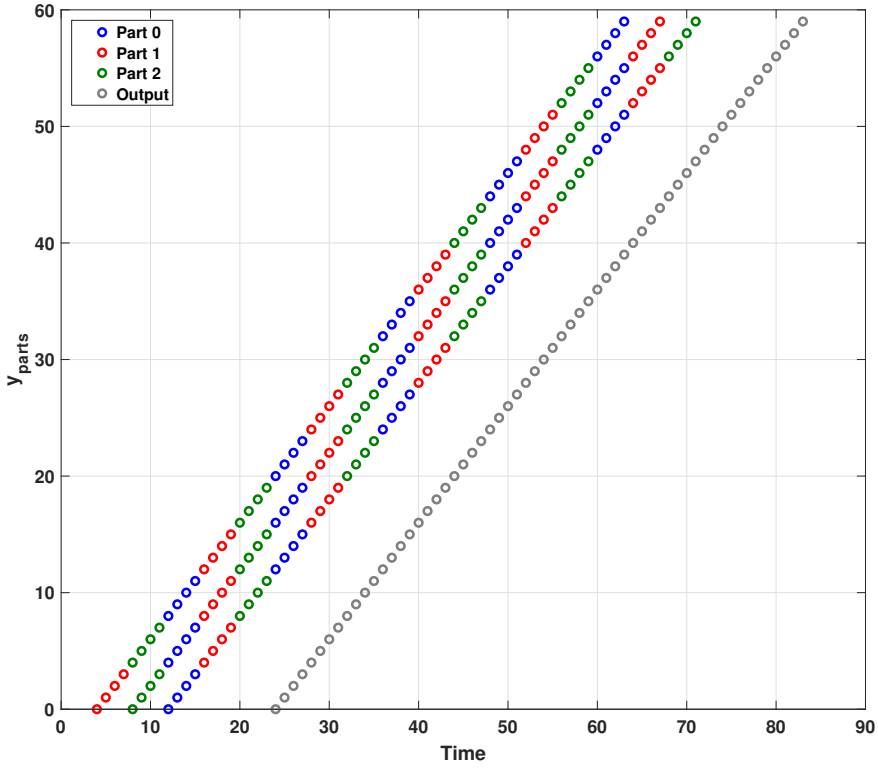


Figure 7.8: The timing for the outputs of the re-timer and the desired final output samples in case of $L = 4$ and $P = 3$.

According to Eq. (7.2), output y_i is obtained from output v_j by:

$$j = i \mod PL \tag{7.4}$$

At this point Eq. 7.3 indicates that we need to add P terms for each v_j to obtain the corresponding y_i. The problem arises because we have P partial results to be added before the next group arrives after PL time steps. These P terms are separated by L time steps apart while each addition operation requires D time steps. This situation can be solved according to the discussion in Section 7.4.

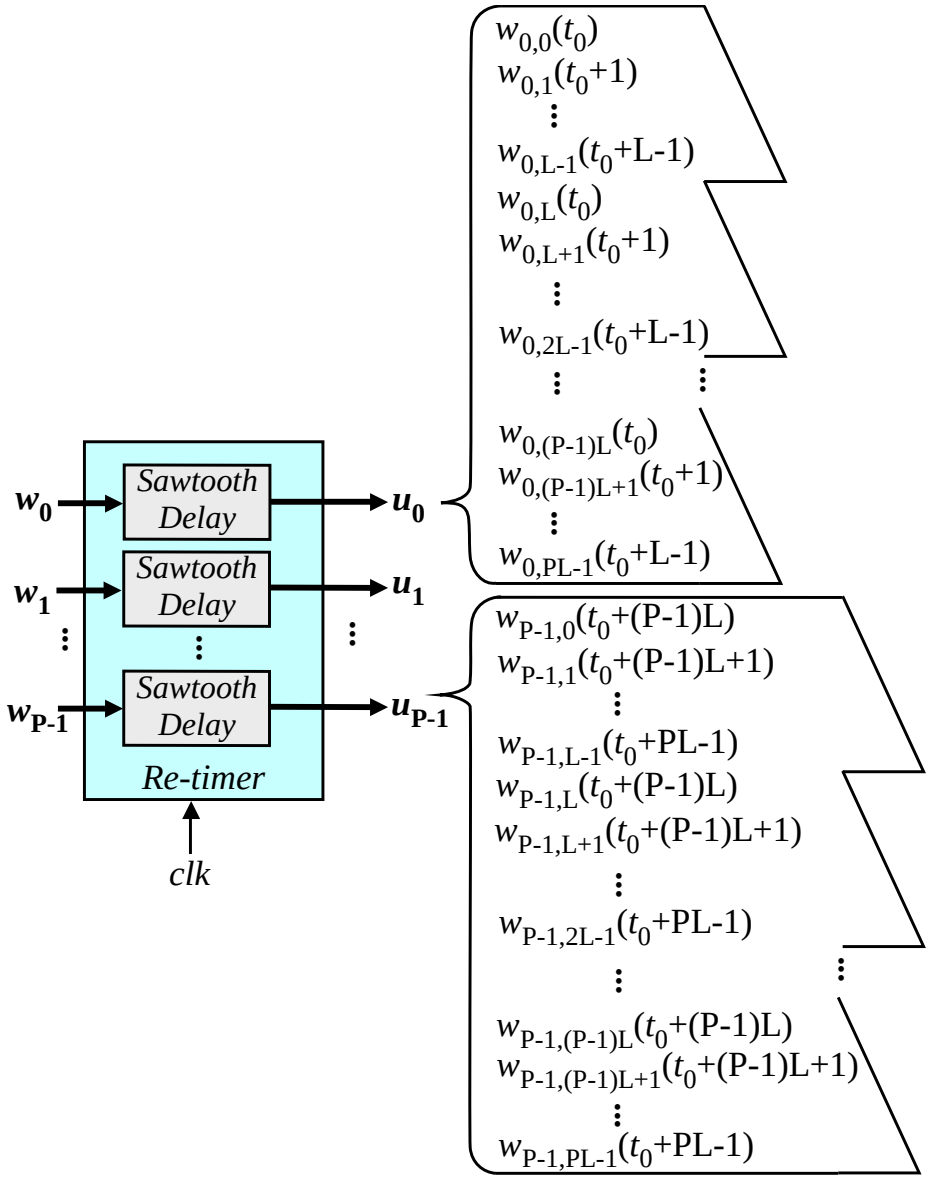


Figure 7.9: Re-timer block diagram.

7.4 Carry-Save Addition of the Outputs from the Interconnection Network

Th addition in Eq. 7.3 can be solved using carry-save accumulator (CSA) as shown in Fig. 7.11. Figure 7.11(a) shows an array of P CSAs to simultaneously add the P partial results. Each CSA converts a group of P inputs to a sum and carry words. The outputs of the CSA array uses tri-state buffers to produce the stream of sum and carry words and the spacing between the words is one clock cycle. At this stage we

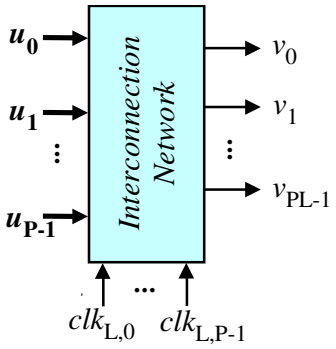


Figure 7.10: Interconnection network block diagram.

have produced a pair of words that mut be added to produce the output samples y_i.

$$y_i = s_i + c_i \tag{7.5}$$

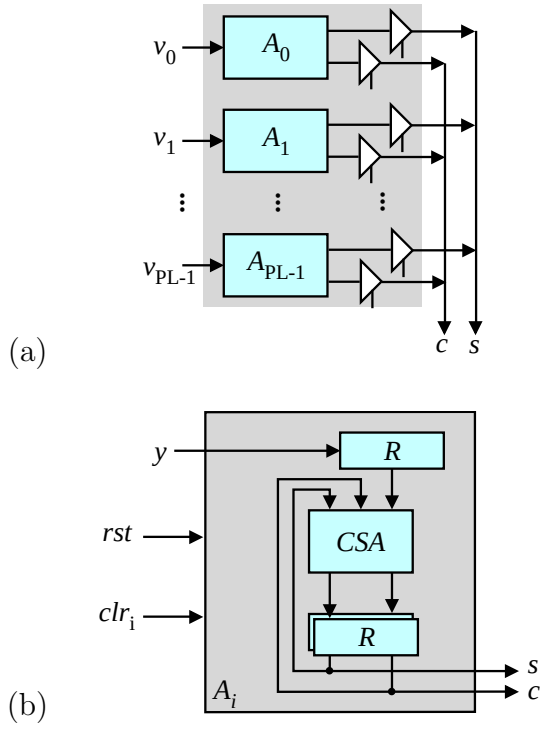


Figure 7.11: Partitions outputs carry-save addition block diagram. (a) Adders array. (b) Adder detals.

At this point Eq. 7.5 indicates that we need to add P terms for each v_j to obtain the corresponding y_i.

7.5 Final Addition

This stage is used to add the sum s and carry c outputs obtained from the previous stage. A deep pipeline ripple-carry adder (RCA) is employed as shown in Fig. 7.12. The box labeled D represents a one-bit register to store the carries between the pipeline stages. Based on the number of digits that the RCA can add within one clock cycle, the word-size of the obtained results from the previous stage is divided into a number of digits N. The number of pipeline stages is equal to N

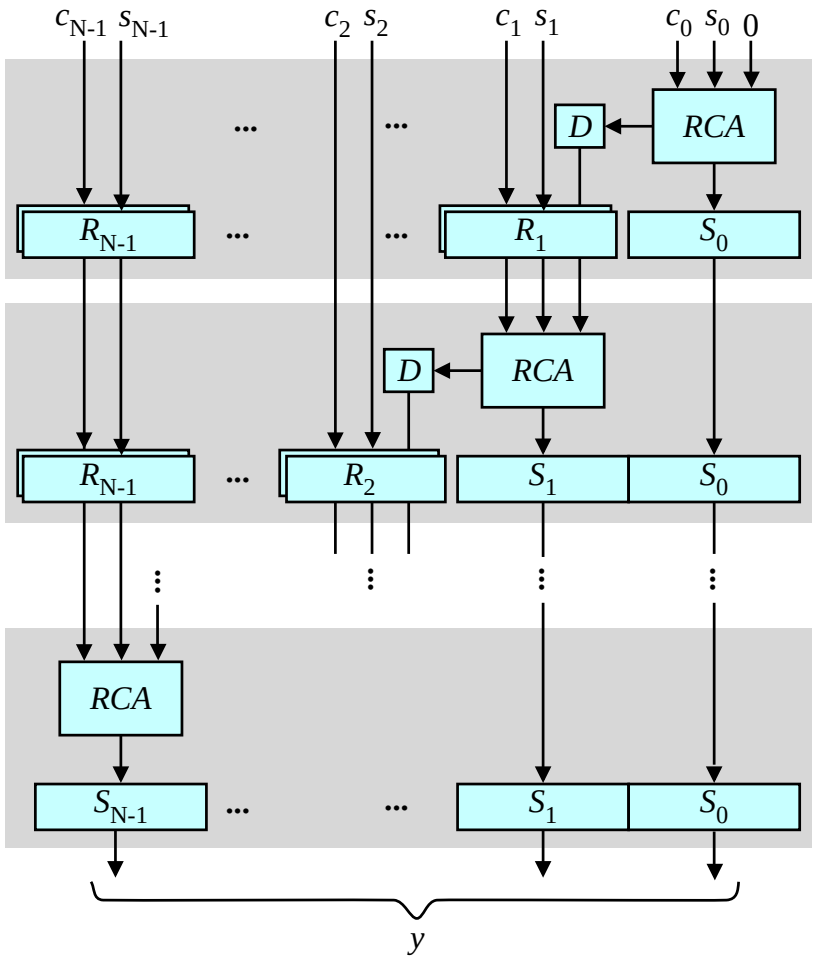


Figure 7.12: Final addition block diagram.

7.6 Ultra-High Speed Interpolator General Block Diagram

Based on the discussions on the previous sections, the overall ultra-high speed interpolator that capable of accommodating the advanced radars high I/O data rates is shown in Fig. 7.13. The top speed of this structure is limited by the delay of a one-bit full adder. On the other hand, Fig. 7.14 shows the standard interpolator basic structure where the top speed is governed by the critical path delay of the processor ALU. This critical path delay typically governed by the multiplier delay.

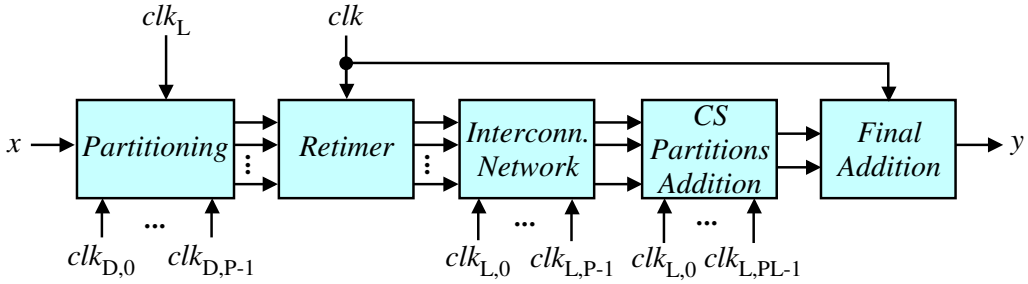


Figure 7.13: General block diagram of the ultra-high speed interpolator.

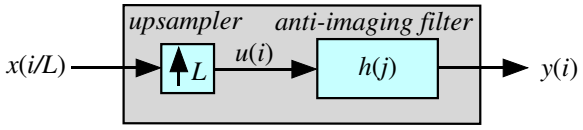


Figure 7.14: Interpolator basic structure.

7.7 System Clocking

As shown in Fig. 7.13, for the ultra-high speed interpolator to perform the interpolation operations, it is required to operate at different clock rates. Figure 7.15 shows the clocks used in the system for the case of $D = 8$ and $L = 4$. The clock that is represented in red color is the fastest clock and is required by the retimer and final addition blocks. The clocks that are represented in blue color are the clocks that are required by the partitioning, interconnection network, and partitions addition blocks. These clocks are related to the interpolation factor L and there is a phase shift one fast clock cycle between each succeeding clock. The clocks that are represented in

black color are the clocks that are required by only the partitioning block. These clocks are related to the dilation D and there is a phase shift one fast clock cycle between each succeeding clock.

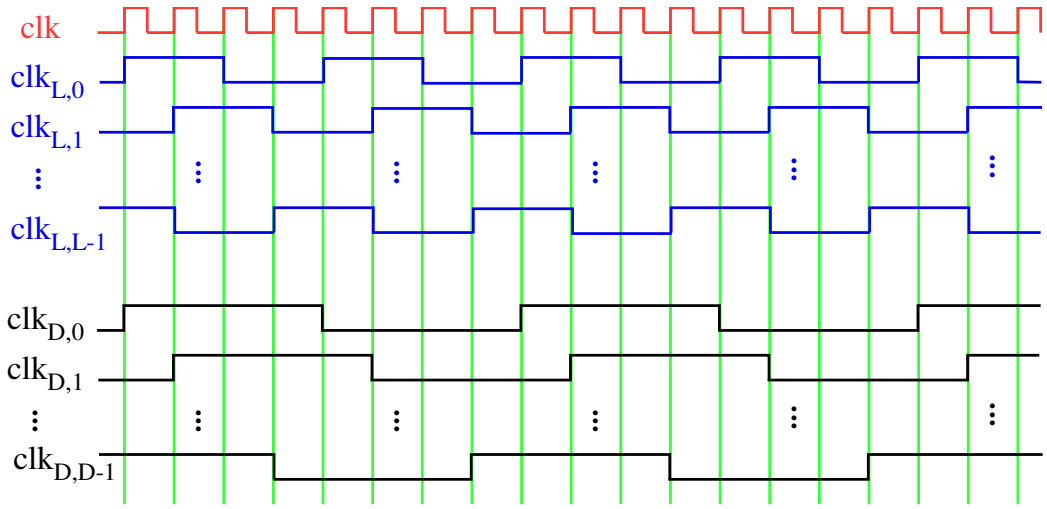


Figure 7.15: System clocking required for ultra high speed interpolator.

7.8 Conclusions

The potential speed performance gap between the modern wideband radar I/O rate and silicon operating speed was identified. A new methodology was proposed to close this speed performance gap. The methodology specifies several steps and applied here for the interpolator algorithm. MATLAB simulation results indicate that this approach leads to satisfactory results.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The main focus of this dissertation has been the enhancement of radar performance by the application of NA beamforming. In Chapter 2, a wideband beamformer has been proposed using nested arrays, multirate systems, and 2-D FIR filters for wideband radar interference rejection. In Chapters 3, 4, and 5, a systematic methodology was used to find best data scheduling strategies and explore possible NA beamformer basic building blocks structures. For each building block, one design which had the fastest clock speed out of different alternative structures was selected. FPGA implementations are performed for the proposed designs as well as existing well-known structures to verify their functionality and compare their performance. FPGA implementation results confirm that the proposed implementation is 2.7 faster than the conventional implementation with increase of 5% in the hardware resources required. In Chapter 6, the validation of the NA beamformer was performed. in addition, the error sources and effects of the finite word-length errors was studied. Chapter 7 proposed a new techniques for closing the speed gap. This techniques allows using slow processor clocks but requires high resolution for the phase shift between the phases which depends on speed of the ADC.

8.2 Summary and Significance of Dissertation Contributions

The contributions of this dissertation are summarized in the following subsections:

8.2.1 Processor Array Design Space Exploration for High-Speed Decimators

In this dissertation a systematic methodology presented in [6] was applied to the decimator difference equations. This methodology is used to develop a single dependance graph that reflects the actions of the anti-aliasing filter and the downsampler. Different scheduling and projection functions were used to perform the design space exploration. Three scheduling functions and 4 projection directions were possible which produces 12 valid designs. Six designs was chosen which satisfy the fastest possible system clock speed. One of the 6 designs was chosen since it displayed the least area.

Earlier works adopted an ad hoc techniques that designed the downsampler apart from the anti-aliasing filter. This produced a single decimator design with little opportunity to optimize the design based on speed, area or power. Any optimization effort was a compromise between speed and area.

The significance of our contribution is allowing us to merge the operations of the downsampler and filter. This resulted in an immediate reduction in the hardware complexity and perform true design space exploration and 12 designs were obtained. Our approach allowed us to select the best design that conforms to the speed and area design specifications.

8.2.2 Processor Array Design Space Exploration for High-Speed 2-D BF Filters

The systematic methodology presented in [6] was applied to the 2-D BF filter difference equation. This methodology is used to develop a single 3D computational domain that reflects the actions of the 2-D BF filter. Different scheduling and projection functions were used to perform the design space exploration. 3 scheduling functions and 4 projection directions were possible which produces 12 valid designs. 6 design options was chosen which satisfy the fastest possible system clock speed. One of the 6 designs was chosen since it displayed the least area.

Earlier works adopted an ad hoc techniques that designed the 2-D BF filter. This produced a single 2-D BF filter design with little opportunity to optimize the design based on speed, area or power.

The significance of our contribution is allowing us to perform true design space

exploration and 12 designs were obtained. Our approach allowed us to select the best design that conforms to the speed or area design specifications.

8.2.3 Processor Array Design Space Exploration for High-Speed Interpolators

The systematic methodology presented in [6] was applied to the interpolator difference equations. This methodology is used to develop a single dependance graph that reflects the actions of the upsampler and the anti-imaging filter. Different scheduling and projection functions were used to perform the design space exploration. 3 scheduling functions and 4 projection directions were possible which produces 12 valid design options. 3 design options was chosen which satisfy the fastest possible system clock speed. One of the 6 designs was chosen since it displayed the least area.

Earlier works adopted an ad hoc techniques that designed the downsampler apart from the anti-imaging filter. This produced a single decimator design with little opportunity to optimize the design based on speed, area or power. Any optimization effort was a compromise between speed and area.

The significance of our contribution is allowing us to merge the operations of the upsampler and the filter. This resulted in an immediate reduction in the hardware complexity and perform true design space exploration and 12 designs were obtained. Our approach allowed us to select the best design that conforms to the speed and area design specifications.

8.2.4 Finite word-length effect on the beamformer accuracy

The results from finite word-length MATLAB implementation is evaluated by studying the effect of finite word-length errors on the accuracy of the complete beamformer. The accuracy analysis is performed by calculating the SER at the output of the beamformer for different word-lengths. The SER results show that a good accuracy of the implemented system is obtained with a word-length of 12-bits and that the quality of accuracy increases significantly with increased word-lengths.

In the previous work, most of the authers studed the SER for the individual system components separately. They also did not model the combined effects of the quantization in the ADC and truncation of the multiplication, addition, and accumulation operations.

In this work we were able to model the different sources of noise in the cascaded multi rate system comprising the beamformer. Based on the model, a comprehensive study was performed for the effect of the finite word length on the performance of the system. The minimum word length was determined. We also include in our model the combined effects of the multiplication, addition, and accumulation operations.

8.2.5 Closing the speed gap between high-speed ADC and silicon speed

In some radar applications that employ the high speed ADC, there will be a speed gap between the I/O rates of the ADC and hardware implemented designs even with the fastest possible designs.

In this work, we identified the speed gap between the application I/O rate and the silicon processor rate. This dissertation proposed closing the speed gap using three techniques: partitioning, dilation, and polyphase clocking.

The approach starts by partitioning the DAG. The number of partitions required depends on the desired dilation. The number of clock phases required is based on the number of partitions.

The significance of this contribution is that it allows us to implement high speed algorithms on low speed processors.

8.3 Future Work

In what follows, three research topics are discussed as a continuation of the work presented in this dissertation.

8.3.1 Design Space Exploration of the 3-D Beamforming Filter

In order to achieve beamforming in both elevation and azimuth angles, 2-D arrays need to be deployed. This requires to employ a 3D FIR frustum shaped filter [21] in conjunction with planner antenna array. The 3-D frustum filter I/O relations is given by:

$$y(n_t) = \sum_{i=-(I-1)/2}^{(I-1)/2} \sum_{j=-(J-1)/2}^{(J-1)/2} \sum_{k=0}^{K-1} h(i,j,k)x(i,j,n_t-k) \tag{8.1}$$

where $y(n_t)$ represents the output samples, $h(i, j, k)$ represents the filter coefficients, and $x(i, j, n_t - k)$ represents the input samples from the sensor arrays. We assume a right-sided output samples sequences, which implies $n_t \geq 0$. The number of the 2-D BB filter coefficients is $I \times J \times K$ where $I \times J$ is the number of sensors in the sensor array.

The ranges of the indices in Eq. (8.1) define the computational domain $\mathcal{D} \subset \mathcal{Z}^n$ of the algorithm. Since Eq.(4.1) has three indices n_t, i, j and k, the algorithm is defined in the 4-dimensional integer domain $\mathcal{Z}^4$.

The possibility to apply the systematic methodology that used in Chapter 4 for the frustum filters would be interesting.

8.3.2 Application of Closing Speed Gap Methodology to the Decimator and 2-D BF Filter Blocks

The new methodology proposed in Chapter 7 was applied to the interpolator block as an example. It would be interesting to extend the application of this proposed methodology to the other two building blocks of the NA beamformer (decimator and 2-D BF filter).

8.3.3 Implementation on SDR

The obtained efficient systolic array designs for the basic building blocks of the NA beamforming system was targeting the SDR a cognitive radar system. From a practical point of view, the main future work is to implement a real time NA beamforming system on SDR that can adaptively switch between different 2-D BF filter coefficients according to the desired DOA.

Bibliography

[1] M. Skolnik, *Radar handbook, Third Edition.* McGraw-Hill Education, 1990.

[2] W. L. Melvin and J. A. Scheer, *Principles of Modern Radar: Advanced Techniques.* Edison, NJ: SciTech Publishing, 2013.

[3] D. Bosworth, "The demand for digital: Challenges and solutions for high speed analog to digital converters and radar systems," Tech. Rep., 2014.

[4] [Online]. Available: https://news.ti.com/industrys-fastest-12-bit-ADC-meets-most-demanding-requirements-tomorrows-test-and-measurement-and-defense-applications

[5] W. Liu and S. Weiss, *Wideband beamforming: concepts and techniques.* New York, NY: John Wiley & Sons, 2010, vol. 17.

[6] F. Gebali, *Algorithms and parallel computing.* New York, NY: John Wiley & Sons, 2011.

[7] M. Shoukry, F. Gebali, and P. Agathoklis, "Decimator systolic arrays design space exploration for multirate signal processing applications," *IET Circuits, Devices & Systems*, pp. 1–10, (Accepted on 8th August 2019).

[8] ——, "Design and analysis of processor arrays for implementation of 2-D FIR broadband beamformers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–12, (submitted on 24th September 2019)).

[9] ——, "Systolic array design space exploration of interpolators for multirate systems," *IET Circuits, Devices & Systems*, pp. 1032–1038, 2019.

[10] ——, "High performance implementation of nested array beamformer for wideband radar applications," in *2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM).* IEEE, 2019, pp. 1–5.

[11] I. Moazzen and P. Agathoklis, "Broadband beamforming using 2D trapezoidal filters and nested arrays," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim)*. IEEE, 2011, pp. 488–493.

[12] T. K. Gunaratne and L. T. Bruton, "Beamforming of broad-band bandpass plane waves using polyphase 2-D FIR trapezoidal filters," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 3, pp. 838–850, 2008.

[13] H. L. Van Trees, *Optimum array processing: Part IV of detection, estimation, and modulation theory*. New York, NY: John Wiley & Sons, 2004.

[14] P. Vouras, "Fully adaptive space-time processing on nested arrays," in *2015 IEEE Radar Conference (RadarCon)*. IEEE, 2015, pp. 0858–0863.

[15] P. Pal and P. Vaidyanathan, "Nested arrays: A novel approach to array processing with enhanced degrees of freedom," *IEEE Transactions on Signal Processing*, vol. 58, no. 8, pp. 4167–4181, 2010.

[16] J. Yang, G. Liao, and J. Li, "Robust adaptive beamforming in nested array," *Signal Processing*, vol. 114, pp. 143–149, 2015.

[17] M. Yang, L. Sun, X. Yuan, and B. Chen, "A new nested MIMO array with increased degrees of freedom and hole-free difference coarray," *IEEE Signal Processing Letters*, vol. 25, no. 1, pp. 40–44, 2017.

[18] N. del Rey-Maestre, D. Mata-Moya, M.-P. Jarabo-Amores, P.-J. Gómez-del Hoyo, J.-L. Bárcena-Humanes, and J. Rosado-Sanz, "Passive radar array processing with non-uniform linear arrays for ground target's detection and localization," *Remote sensing*, vol. 9, no. 7, p. 756, 2017.

[19] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing Prentice-Hall Signal Processing Series*. Prentice-Hall, Englewood Cliffs, NJ, 1984.

[20] Y. Zheng, R. Goubran, and M. El-Tanany, "A broadband adaptive beamformer using nested arrays and multirate techniques," in *Proc. IEEE DSP Workshop*, vol. 3, 2000, pp. 200–205.

[21] I. Moazzen and P. Agathoklis, "Broadband beamfoming using nested planar arrays and 3d fir frustum filters," in *2012 IEEE International Symposium on Circuits and Systems*. IEEE, 2012, pp. 53–56.

[22] T. Do-Hong and P. Russer, "A new design method for digital beamforming using spatial interpolation," *IEEE antennas and wireless propagation letters*, vol. 2, no. 1, pp. 177–181, 2003.

[23] L. J. Gudino, S. Jagadeesha, and J. X. Rodrigues, "A novel filter design for spatially interpolated beamformer," in *International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS)*. IEEE, 2009, pp. 1–4.

[24] V. Ariyarathna, A. Madanayake, P. Agathoklis, and L. T. Bruton, "Mixed microwave-digital and multi-rate approach for wideband beamforming applications using 2-D IIR beam filters and nested uniform linear arrays," *Multidimensional Systems and Signal Processing*, vol. 29, no. 2, pp. 703–718, 2018.

[25] C. Andrich, A. Ihlow, J. Bauer, N. Beuster, and G. Del Galdo, "High-precision measurement of sine and pulse reference signals using software-defined radio," *IEEE Transactions on Instrumentation and Measurement*, pp. 1132–1141, 2018.

[26] K. Amulya and G. Sadashivappa, "Design and implementation of a reconfigurable digital down converter for 4G systems using MATLAB and FPGA- a review," in *Conference on Emerging Devices and Smart Systems*. IEEE, 2018, pp. 265–268.

[27] T. Y. Lee, T. Butcher, T. Ishida, A. Panigada, and D. Meacham, "Digitally enhanced high speed ADC for low power wireless applications," in *International Conference on Microwaves for Intelligent Mobility*. IEEE, 2017, pp. 64–67.

[28] M. Balakrishnan, K. A. Meerja, K. K. Gundugonti, and S. R. K. Kalva, "Design of interfaces between high speed data converters and high performance fpgas for software defined radio applications," *Telecommunication Systems*, pp. 1–14, 2019.

[29] K. K. A.Swetha, S. Jagadeesh, "FPGA implementation of quadrature mirror filter in performance of adaptive equalizers in wireless communication channel," *International Journal of electronics & communication technology*, vol. 5, pp. 183–185, 2014.

[30] C.-L. Liu and P. Vaidyanathan, "Coprime DFT filter bank design: Theoretical bounds and guarantees," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 3861–3865.

[31] M. Newey, G. R. Benitz, D. J. Barrett, and S. Mishra, "Detection and imaging of moving targets with limit SAR data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 6, pp. 3499–3510, 2018.

[32] T.-l. WANG, J.-g. YANG, and M. Yu, "FIR multiphase decimation filtering method based on base 2 FFT," in *International Conference on Computer, Communication and Network Technology*. DEStech, 2018, pp. 330–337.

[33] S. I. Kelly and M. E. Davies, "A fast decimation-in-image back-projection algorithm for SAR," in *IEEE Radar Conference*. IEEE, 2014, pp. 1046–1051.

[34] G. Babur, O. A. Krasnov, A. Yarovoy, and P. Aubry, "Nearly orthogonal waveforms for MIMO FMCW radar," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 3, pp. 1426–1437, 2013.

[35] S. Arunkumar and P. G. Kumar, "Performance and analysis of transmultiplexers using decimator and interpolator," *Journal of Circuits, Systems and Computers*, p. 1950009, 2018.

[36] T. Rahate, S. Ladhake, and U. Ghate, "Decimator filter for hearing aid application based on FPGA," *International Research Journal of Engineering and Technology (IRJET)*, vol. 5, pp. 1175–1177, 2018.

[37] T. V. Rahate, S. A. Ladhake, and U. S. Ghate, "FPGA based implementation of decimator filter for hearing aid application," *International Research Journal of Engineering and Technology (IRJET)*, vol. 5, pp. 2289–2293, 2018.

[38] J. W. Woods, *Subband image coding*. Springer Science & Business Media, 2013, vol. 115.

[39] J. Hegarty, R. Daly, Z. DeVito, J. Ragan-Kelley, M. Horowitz, and P. Hanrahan, "Rigel: Flexible multi-rate image processing hardware," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 85, 2016.

[40] R. Mehra and V. Singh, "Rational sampling rate converter using coefficient symmetry," *International Journal of Computer Applications*, vol. 129, no. 4, pp. 1–7, 2015.

[41] S. Harize, M. Benouaret, and N. Doghmane, "A methodology for implementing decimator FIR filters on FPGA," *International Journal of Electronics and Communications*, vol. 67, no. 12, pp. 993–1004, 2013.

[42] T. Liu, J. Han, and Z. Li, "Broadband DDC based on polyphase filter and its FPGA implementation," in *IEEE International Conference on Electronic Information and Communication Technology*. IEEE, 2016, pp. 170–173.

[43] A. Jetly, R. Mehra, and R. Rana, "Reconfigurable channel interference reduction for vehicular communication applications," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 5, no. 5, pp. 1068–1073, 2017.

[44] V. Jayaprakasan and M. Madheswaran, "FPGA implementation of FIR based decimation filter structure for WiMAX application," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, pp. 2830–2837, 2013.

[45] J. X. Zheng, K. Nguyen, and Y. He, "Optimized FPGA implementation of multirate FIR filters through thread decomposition," in *IEEE Aerospace Conference*. IEEE, 2010, pp. 1–10.

[46] R. Mehra and R. Arora, "FPGA-based design of high-speed CIC decimator for wireless applications," *power*, vol. 1, p. 2, 2011.

[47] A. A.AlJuffri, A. S. Badawi, M. S. BenSaleh, A. M. Obeid, and S. M. Qasim, "FPGA implementation of scalable microprogrammed FIR filter architectures using wallace tree and vedic multipliers," in *Third International Conference on Technological Advances in Electrical, Electronics and Computer Engineering*. IEEE, 2015, pp. 159–162.

[48] S. Prasanna and S. J. V. Rani, "Area and speed efficient implementation of symmetric FIR digital filter through reduced parallel LUT decomposed DA approach," *Circuits and Systems*, vol. 7, no. 08, p. 1379, 2016.

[49] R. Thakur and K. Khare, "High speed FPGA implementation of FIR filter for DSP applications," *International Journal of Modeling and Optimization*, vol. 3, no. 1, pp. 92–94, 2013.

[50] A. Tawfik, F. Gebali, M. Fahmi, E. Abdel-Raheem, and P. Agathoklis, "High-speed area-efficient inner-product processor," *Canadian Journal of Electrical and Computer Engineering*, vol. 19, no. 4, pp. 187–191, 1994.

[51] L. Tan and J. Jiang, *Digital signal processing: fundamentals and applications*. Academic Press, 2018.

[52] S. Haykin, *Array signal processing*. Englewood Cliffs, NJ, Prentice-Hall., 1985.

[53] L. C. Godara, "Application of antenna arrays to mobile communications. II. beam-forming and direction-of-arrival considerations," *Proceedings of the IEEE*, vol. 85, no. 8, pp. 1195–1245, 1997.

[54] B. D. Van Veen and K. M. Buckley, "Beamforming: A versatile approach to spatial filtering," *IEEE ASSP magazine*, vol. 5, no. 2, pp. 4–24, 1988.

[55] J. Benesty, J. Chen, and Y. Huang, *Microphone array signal processing*. Springer Science & Business Media, 2008, vol. 1.

[56] I. Moazzen, S. Harrison, P. Agathoklis, and P. Driessen, "A nested microphone array for broadband audio signal processing," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim)*. IEEE, 2013, pp. 377–382.

[57] P. E. Dewdney, P. J. Hall, R. T. Schilizzi, and T. J. L. Lazio, "The square kilometre array," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1482–1496, 2009.

[58] N. W. Liyanage, L. T. Bruton, P. Agathoklis, and C. Edussooriya, "Space-time digital filtering of radio astronomical signals using 3-D cone filters," in *RFI mitigation workshop*, vol. 107. SISSA Medialab, 2010, p. 026.

[59] T. K. Gunaratne and L. T. Bruton, "Broadband beamforming of dense aperture array (DAA) and focal plane array (FPA) signals using 3D spatio-temporal filters for applications in aperture synthesis radio astronomy," *Multidimensional Systems and Signal Processing*, vol. 22, no. 1-3, pp. 213–236, 2011.

[60] N. K. Nikolova, "Microwave imaging for breast cancer," *IEEE microwave magazine*, vol. 12, no. 7, pp. 78–94, 2011.

[61] L. Bruton and N. Bartley, "Three-dimensional image processing using the concept of network resonance," *IEEE Transactions on Circuits and Systems*, vol. 32, no. 7, pp. 664–672, 1985.

[62] K. Nishikawa, T. Yamamoto, K. Oto, and T. Kanamori, "Wideband beamforming using fan filter," in *IEEE International Symposium on Circuits and Systems*, vol. 2. IEEE, 1992, pp. 533–536.

[63] N. Rajapaksha, A. Madanayake, and L. T. Bruton, "2D space-time wave-digital multi-fan filter banks for signals consisting of multiple plane waves," *Multidimensional systems and signal processing*, vol. 25, no. 1, pp. 17–39, 2014.

[64] T. K. Gunaratne and L. T. Bruton, "Broadband beamforming of bandpass plane waves using 2D FIR trapezoidal filters at baseband," in *Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 2006, pp. 546–549.

[65] R. T. Wijesekara, C. U. Edussooriya, L. T. Bruton, and P. Agathoklis, "A low-complexity 2-D spatially-interpolated FIR trapezoidal filter for enhancing broadband plane waves," in *10th International Workshop on Multidimensional (nD) Systems (nDS)*. IEEE, 2017, pp. 1–6.

[66] L. Bruton and N. Bartley, "The design of highly selective adaptive three-dimensional recursive cone filters," *IEEE Transactions on Circuits and Systems*, vol. 34, no. 7, pp. 775–781, 1987.

[67] C. Wijenayake, A. Madanayake, and L. Bruton, "Broadband multiple cone-beam 3-D IIR digital filters applied to planar dense aperture arrays," *IEEE Transactions on Antennas and Propagation*, vol. 60, no. 11, pp. 5136–5146, 2012.

[68] T. K. Gunaratne, L. Bruton, and P. Agathoklis, "Broadband beamforming of focal plane array (FPA) signals using real-time spatio-temporal 3D FIR frustum digital filters," *IEEE Transactions on Antennas and Propagation*, vol. 59, no. 6, pp. 2029–2040, 2011.

[69] H. Shubayli, C. U. Edussooriya, I. Moazzen, P. Agathoklis, and L. Bruton, "Implementation and performance analysis of 3-D cone and frustum filters," in *IEEE*

Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim). IEEE, 2015, pp. 450–455.

[70] V. Ariyarathna, V. A. Coutinho, S. Pulipati, A. Madanayake, R. T. Wijesekara, C. U. Edussooriya, L. T. Bruton, T. K. Gunaratne, and R. J. Cintra, "Real-time 2-D FIR trapezoidal digital filters for 2.4 GHz aperture receiver applications," in *Moratuwa Engineering Research Conference (MERCon).* IEEE, 2018, pp. 350–355.

[71] P. Kumar, P. C. Shrivastava, M. Tiwari, and G. R. Mishra, "High-throughput, area-efficient architecture of 2-D block FIR filter using distributed arithmetic algorithm," *Circuits, Systems, and Signal Processing*, vol. 38, no. 3, pp. 1099–1113, 2019.

[72] V. S. TEJA and D. HEMA, "A novel design of 2-D finite impulse response filters with reduced memory footprint," *International Journal of Emerging Technology in Computer Science & Electronics*, vol. 11, no. 3, pp. 60–63, 2014.

[73] S. Pulipati, V. Ariyarathna, and A. Madanayake, "A 16-element 2.4-GHz digital array receiver using 2-D IIR spatially-bandpass plane-wave filter," in *MTT-S International Microwave Symposium.* IEEE, 2018, pp. 667–670.

[74] P. Kumar, P. C. Shrivastava, M. Tiwari, and A. Dhawan, "ASIC implementation of area-efficient, high-throughput 2-D IIR filter using distributed arithmetic," *Circuits, Systems, and Signal Processing*, pp. 2934–2957, 2018.

[75] [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ru/c-addsub.html

[76] [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/blk_m blk-mem-gen.pdf

[77] R. E. Crochiere and L. R. Rabiner, *Multirate Digial Signal Processing.* Englewood Cliffs, New Jersey: Prentice-Hall, 1983.

[78] E. Gopi, *Multi-Disciplinary Digital Signal Processing.* Springer, 2018.

[79] S. Sridevi, R. Dhuli, and K. Baboji, "Low power and low complexity implementation of LPTV interpolation filter," in *International Conference on Computer Communication and Informatics (ICCCI).* IEEE, 2016, pp. 1–8.

[80] Y. Torres, K. Premaratne, F. Amelung, and S. Wdowinski, "An efficient polyphase filter-based resampling method for unifying the PRFs in SAR data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 10, pp. 5741–5754, 2017.

[81] V. T. Vu and M. I. Pettersson, "Fast backprojection algorithms based on subapertures and local polar coordinates for general bistatic airborne SAR systems," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 5, pp. 2706–2712, 2016.

[82] Z. Kaya and E. Seke, "Direct generation of upsampled FIR filter response a simple extension to filters with distributed arithmetic," in *10th International Conference on Digital Technologies*. IEEE, 2014, pp. 110–113.

[83] R. S. Abd El-Azeem, M. A. El-Moursy, A. M. Nassar, A. Gharib, N. T. Abou El-Kheir, and M. S. El-Kharashi, "High performance interpolation filter using direct computation," in *11th International Design & Test Symposium (IDT)*. IEEE, 2016, pp. 121–124.

[84] A. Al-Haj, "An efficient configurable hardware implementation of fundamental multirate filter banks," in *IEEE SSD 5th International Multi-Conference on Systems, Signals and Devices*. IEEE, 2008, pp. 1–5.

[85] B. Porat, *A Course in Digital Signal Processing*. New York, NY: John Wiley & Sons, 1996.