# Msc THESIS

## Low Cost Techniques for High-Speed Implementation of Decision Feedback Equalizers for FPGAs

**Charalampos Papadopoulos**
**4192141**

## Abstract

In modern digital communications throughput rates in the order of gigabits per second are not uncommon. Hence there is a need for fast equalizing filters. While all feed forward filters can easily be pipelined, therefore sped up, the feedback filter (FBF) of decision feedback equalizers is the bottleneck of its performance. A simple, extremely fast, reformulation of the FBF, based on an obvious expansion of the Shannon's expansion theorem idea, can easily be devised. This particular design can be retimed in a straightforward manner to reach its iteration period bound, but is ultimately limited by the propagation delay of the registers. Even though we can unfold the circuit in an attempt to limit the register's delay relevance, its hardware overhead is exponentially dependent on the number of tap coefficients (L), therefore, very costly even for relatively small Ls. In order to cope with the exponential area growth a two stage pre-computation schema, similar to the reformulated FBF, will be introduced. We show that the area degrades significantly with the penalty of decreasing the maximum achievable frequency. To reach its iteration bound, this new design has to be unfolded. In our experiments we use the Xilinx ISE FPGA synthesizer, targeting the Virtex 5 family, the XC5VLX20T module, set to speed -2. We will approximate the minimum needed unfolding factor for an FPGA centric design, considering different word lengths and number of tap coefficients, to use as a base for the experimental phase. Our experiments with the unfolding factor will conclude when the performance is comparable with that of the first FBF reformulation. For that final design we will find the L where the hardware overhead improvement outweighs the hit in performance. Finally, based on the experimental results, we will show that the performance can be further increased, with the introduction of a retime approach. However according to our experiments the retiming transformation enhanced the performance by only 2%.

**Keywords:** Decision Feedback Equalizer, Feedback Filter, High speed, Low cost, Unfolding

Delft University of Technology

# Low Cost Techniques for High-Speed Implementation of Decision Feedback Equalizers for FPGAs

---

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Charalampos Papadopoulos
born in Athens, Greece

**Author**

Charalampos Papadopoulos (xaris.papadopoulos@hotmail.com)

**Title**

Low Cost Techniques for High-Speed Implementations of Decision Feedback Equalizers

**M.Sc. Presentation**

August 6th, 2015

**Graduation Committee**

Dr Georgi Gaydadjiev (Supervisor), Delft University of Technology
Dr van Genderen, Delft University of Technology
Dr. A Iosup, Delft University of Technology

# Contents

List of Figures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
List of Tables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .14

1 Introduction 16
- 1.1 Motivation and Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . 17
- 1.2 Thesis Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

Part I: Background 20

2 Introduction to Channel Equalization 22
- 2.1 Inter-symbol Interference . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
- 2.2 Design of Bandlimited Signals for Zero or Controlled ISI . . . . . . . . 23
  - 2.2.1 Zero ISI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .23
  - 2.2.2 Controlled ISI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
  - 2.2.3 Data Detection for Controlled ISI . . . . . . . . . . . . . . . . . . . . . . . 26
- 2.4 System Design in the Presence of Channel Distortion . . . . . . . . . . . 27
  - 2.4.1 Known Channel Characteristics . . . . . . . . . . . . . . . . . . . . . . . . 27
  - 2.4.2 Unknown Channel Characteristics and Equalization . . . . . . . . 28
- 2.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

3 Fast Adder and Multiplexer Implementations 38
- 3.1 Fast Adder Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
  - 3.1.1 Ripple Carry Adder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
  - 3.1.2 Carry Skip Adder and Linear Carry Look-Ahead Adder . . . . . 41
  - 3.1.3 Carry Select Adder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
  - 3.1.4 Square-root Carry Select Adder . . . . . . . . . . . . . . . . . . . . . . . . 43
  - 3.1.5 Parallel Prefix Adders . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .44
  - 3.1.6 Hybrid Adders . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
  - 3.1.7 Carry Save Adder and Multiple Operand Addition . . . . . . . . . 46

3.2 Fast Multiplexer Implementations . . . . . . . . . . . . . . . . . . . . . . . . . . .47
3.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49

4 Iteration Bound 51
4.1 Data Flow Graphs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
4.2 Iteration Period Bound . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.3 Pipelining and Cut Set Retiming . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.4 Unfolding . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .53
4.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

Part II: Implementation and Analysis 57

5 FBF Transformations 59
5.1 Faster Adders and Multiplier Transformation . . . . . . . . . . . . . . . . . . .59
5.2 Complete Circuit Overhaul . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
5.3 Partial pre-Computation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
5.4 Two Stage Pre-Computation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
5.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

6 FPGA Specific Implementations 67
6.1 Theoretical Estimation of the Unfolding Factor J . . . . . . . . . . . . . . .67
6.2 Constituent Units for the FPGA Design . . . . . . . . . . . . . . . . . . . . . . 68
    6.2.1 Adders for FPGA Designs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
    6.2.2 Multiplexers for FPGA Designs . . . . . . . . . . . . . . . . . . . . . . . . .71
6.3 Estimation for the Unfolding Factor J for FPGAs . . . . . . . . . . . . . . .72
6.4 Manual Retiming Approach . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
6.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79

7 Simulation Results 80
7.1 Retimed Multiplexer Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 80
7.2 Two Stage Pre-Computation FBF . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
7.3 Unfolded Circuits, J=2 through 7 . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
7.4 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85
7.5 Retimed circuit. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86

7.6 Validation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87

8 Conclusions and Recommendations 88
8.1 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88
8.2 Recommendations for Future Research . . . . . . . . . . . . . . . . . . . . . . . .89

Bibliography 90

Part III: Appendices 92

A Unfolded Circuits' Indices . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .94

B Area and Speed Measurements 97
B1. Retimed Multiplexer Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . .97
B2. Two Stage pre-Computation FBF . . . . . . . . . . . . . . . . . . . . . . . . . . . 97
B3. Unfolded Circuits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 98
B4. Retimed Circuit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 101

# List of Figures

2.1 Continuous-time Digital Communications' System . . . . . . . . . . . . . . . . . . . . . . . . . .22
2.2 Raised-cosine spectrum . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .24
2.3 Duobinary Signal Pulse in the time and frequency domain . . . . . . . . . . . . . . . . . . . .25
2.4 Modified Duobinary Signal Pulse in the time and frequency domain . . . . . . . . . . . .25
2.5 Equivalent discrete-time system model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .29
2.6 Equivalent discrete-time white noise system model . . . . . . . . . . . . . . . . . . . . . . . . 29
2.7 Dicrete-time channel models A,B,C and their amplitude spectra . . . . . . . . . . . . . . . 32
2.8 Decision Feedback Equalizer block diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .33
2.9 Error rate performance of linear 31 tap MSE equalizer . . . . . . . . . . . . . . . . . . . . . . .35
2.10 Error rate performance of DFE with $K_1$=15, $K_2$ =15 taps . . . . . . . . . . . . . . . . . . . 35
2.11 Performance comparison between MLSE simulation and DFE, both for the correct and estimated symbols feedback . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

3.1 Ripple carry adder with the inverters shifted away from the critical path . . . . . . . . .39
3.2 Canonical Design division . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .40
3.3 Ripple carry generator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .40
3.4 Group PG cells . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .40
3.5 Carry-ripple adder group PG network . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .41
3.6 Manchester carry-chain and its schematic representation . . . . . . . . . . . . . . . . . . . . . 41
3.7 Propagation predictor with Manchester chain generator . . . . . . . . . . . . . . . . . . . . . . 42
3.8 Linear carry look-ahead adder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
3.9 Square-root carry-select adder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .43
3.10 Conditional-sum adder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .44
3.11 Brent-Kung tree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
3.12 Sklansky tree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
3.13 Kogge-Stone tree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
3.14 Sparse-tree adder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
3.15 Carry save adder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .47
3.16 In order, static cmos, transmission gate and tri-state multiplexers . . . . . . . . . . . . . .47
3.17 Delay estimation/ fan-in of 4-to-1 cmos multiplexer and 4-to-1 tree of 2-to-1 cmos multiplexers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .48

4.1 (a) Block diagram and (b) its respective DFG . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
4.2 Fine grain pipelining where the multipliers where conveniently split in two subunits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .53

4.3 An application of cut set retiming . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .53
4.4 The 2-Unfolding of the DFG in (a) into (b) . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

5.1 FBF's critical path . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
5.2 FBF transformation to a single multiplexer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .61
5.3 The tree structure of the $M^L$-to-1 multiplexer. M multiplexers are the input to one multiplexer in the next stage . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
5.4 Retiming of the binary PAM 2-Tap FBF . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .62
5.5 Application of Shannon's decomposition theorem for iteration bound lowering . . . 63
5.6 2-term partial pre-computation and retiming . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .64
5.7 Two stage pre-computation of the FBF for binary PAM . . . . . . . . . . . . . . . . . . . . . .65

6.1 Adder delay growth with word length: x-axis(bits) y-axis(ns) . . . . . . . . . . . . . . . . . 69
6.2 Pipelining cut set for RCAs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
6.3 8bit 4Unfolded FBF with 6Tap coefficients . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .75
6.4 Step1. Move the available registers to all of their branches and Step2. retime the "feedback" multiplexers' cut sets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
6.5 Step3. The ad-hoc chosen cut sets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
6.6 Optimally retimed ASIC design for 8bit 4Unfolded FBF with 6Tap coefficients . . .78

7.1 Area growth laws for the Multiplexer FBF design . . . . . . . . . . . . . . . . . . . . . . . . . 81
7.2 Area growth laws for the 2 stage FBF design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82
7.3 Area and frequency measurements for unfolding factors 2 through 7 and W=8 . . . .83
7.4 Effects of the word length W on the area and frequency for J=4 . . . . . . . . . . . . . . . .84
7.5 Effects of the word length W on the area and frequency for J=7 . . . . . . . . . . . . . . . .84
7.6 Frequency and area comparison for the Multiplexer architecture and 7 Unfolded FBF, W=8 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85
7.7 Frequency and area comparison for the Multiplexer architecture and 7 Unfolded FBF, W=64 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85
7.8 Testing Board for the J Unfolded FBF . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .87

# List of Tables

6.1 Iteration bound for the ASIC Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
6.2 Appropriate Unfolding factor for the ASIC Design . . . . . . . . . . . . . . . . . . . . . . . . . 68
6.3 Delay and Area measurements of various Adder Implementations . . . . . . . . . . . . . 69
6.4 Logic Delay of Sklansky's tree adder vs. the synthesizer's adder mapping . . . . . . . 70
6.5 Logic Delay of CSA/Sklansky's tree adder vs. the synthesizer's adder mapping for 3 operand addition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .70
6.6 Multiplexer delay . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
6.7 Iteration bound for the (6.1) approximation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
6.8 Appropriate Unfolding factor for the (6.1) approximation . . . . . . . . . . . . . . . . . . . .72
6.9 Iteration bound for the (6.2) approximation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
6.10 Appropriate Unfolding factor for the (6.2) approximation . . . . . . . . . . . . . . . . . . .73
6.11 Factored form for the indices of y, for unfolding factor J=2 . . . . . . . . . . . . . . . . . .73

7.1 Area measurements for the Multiplexer FBF Design . . . . . . . . . . . . . . . . . . . . . . . 80
7.2 Area measurements of the 2 stage pre-computation FBF . . . . . . . . . . . . . . . . . . . . 81
7.3 max Frequency estimation for the 2 stage pre-computation FBF . . . . . . . . . . . . . . . 82
7.4 Area measurements for unfolding factors 2 through 7 and 8 bit words . . . . . . . . . . 83
7.5 max Frequency estimation for unfolding factors 2 through 7 and 8 bit words . . . . . 83
7.6: Simulation results for the retimed 4 unfolded FBF of figure 6.7 . . . . . . . . . . . . . . 86

# Introduction 1

For most of the early days of communication theory, analogue communications dominated the field. Noise was, and still is, a major issue and the only way to deal with it was to raise the transmitter's power a good deal above the noise level. Different modulations present of course different levels of sensitivity to noise, however, the trade-off for a more robust transmission per Watt is a larger effective bandwidth. And that is of great importance since different signals cannot be overlaid within the same bandwidth unless they are time multiplexed. The effective transmission of data types other than speech (or image) is also a limitation of analogue communications.

With the advent of digital communications error control became much more complicated, and yet very efficient. From the choice of channel codes to that of convoluted error correcting pre-encodings, very low (theoretically infinitesimally small) bit error rates for large transmission rates have been proven to exist and of course invented and extensively used in practice. For modern convolution codes error bursts on the scale of hundreds of words can be effectively corrected, through educated guesses. More than one signal can also share the same channel, through the advances of network coding theory. Other degradations like crosstalk, or parasitic bandwidth sharing can also be diminished or even taken advantage of. And at last, since all kinds of information can be treated in the same way through proper encoding, data types' limitations are a thing of the past.

Digital communication can in fact date all the way back to the Morse code or even to ancient times, with the lighting of bonfires signaling the presence of invading armies etc. However, what we now perceive as digital communications was really founded with the beginning of C.S. Shannon's work during WWII. Shannon is known as the father of information theory. Information theory views channels and transmitted data sequences through the eyes of probability theory. However at the physical level, digital signals, are still analogue in nature and as such are subject to the same maladies. Most importantly, as we will see in chapter two, the limited bandwidth and channel distortion affect the time duration of the pulses. That is exactly the spark that lights the fire of our subject matter. Bandwidth limitations will be shown to naturally lead to the idea of channel equalization.

## 1.1 Motivation and Problem Statement

The design of Decision Feedback Equalizers (DFEs) is in not a new or even vivid research area. Their theory is well developed since a few decades ago and their implementation pretty standard. However, for modern Giga bit transmission rates the design of the DFE need to go above and beyond the technology limitations to meet the performance standards. And even thought that is, on first sight, nothing but easily achievable, this is done at the expense of high hardware complexity cost.

Furthermore, specific DFE application themselves have an extensive enough "audience" pool to warrant an ideal ASIC implementation. However, the underlying idea of the adaptive digital decision feedback filter has a broader appeal that extends to many sub-disciplines of signal processing that do not qualify for such a privilege. For DSP applications more flexible approaches such as FPGAs are more widespread. For those, as we will see in later chapters, area is also a big concern.

In this thesis we will specifically develop and compare architectures for the part that predominately bottlenecks the DFE, the feedback filter. The development will eventually focus on FPGA applications. The extent of their usefulness will be evaluated using the Xilinx ISE synthesizer/simulator for our measurements' estimation and research for further improvements on the designs will be proposed.

## 1.2 Thesis Overview

The work in this thesis is divided into three main parts:

**Part I:** Theoretical background of the various fields needed to understand the motivation behind and the structure of the decision feedback equalizer- Chapter 2. The delay and area estimates used for the adder and multiplexer components throughout the thesis- Chapter 3. And the material that is necessary for the understanding of the a priori performance estimates for the architectures developed in part II and the justification for the various transformation choices- Chapter 4.

**Part II:** The development of the two different architectural choices for the FBF of the DFE and the theoretical estimates of their respective performance and area- Chapter 5. The necessary reevaluation of the units' design for a more structured FPGA architecture and the experimental estimation of the performance bound- Chapter 6. And finally a short exposition of the most relevant results of our simulations alongside with their comparative evaluation- Chapter7.

**Part III:** A short recapitulation of the conclusions drawn from the experimental results of Chapter 7 and recommendations for future research on and beyond the subject matter- Chapter 8.

# PART I:

## BACKGROUND

# Introduction to Channel Equalization 2

## Digital Pulse Amplitude Modulation Transmission Through Bandlimited Additive White Gaussian Noise Channels

The first step in studying any digital communication system is evaluating its performance on a single symbol basis. Unfortunately there is always an added level of complexity following the concatenation of symbols and a brand new array of problems communication theory has to solve in order to ensure an acceptable bit error rate.

Whenever a string of symbols is passed through a bandlimited channel, whether channel distortion is present or not, inter-symbol interference arises. That is of course due to the increased length of the pulses in the time domain and the insufficiently slow amplitude decrease rate their tails follow. In the following presentation a baseband system will be considered in order to introduce the basics of the subject unimpeded by further distractions.

### 2.1 Inter-symbol Interference

Following the signal flow in the block diagram of a PAM communication system

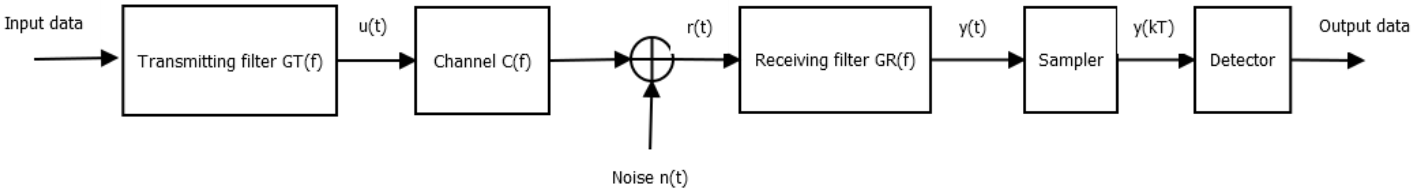


**Figure 2.1: Continuous-time Digital Communications' System**

$$\mathrm{u(t)} = \sum_{-\infty}^{+\infty} a_n g_T(\mathrm{t - nT}) \tag{2.1}$$

,where $a_n$ the k-bit symbol information sequence encoded into the $M=2^k$ levels of the M-ary PAM's amplitudes and $g_T$ the transmission pulse.

$$\mathrm{r(t)} = \sum_{-\infty}^{+\infty} a_n h(\mathrm{t - nT}) + n(t) \tag{2.2}$$

,where$\mathrm{h(t) = c(t) * g_T(t)}$and n(t) represents the AWGN.

$$\mathrm{y(t)} = \sum_{-\infty}^{+\infty} a_n \mathrm{x(t - nT) + v(t)} \tag{2.3}$$

,where $\mathrm{x(t) = g_T(t) * c(t) * g_R(t)}$ and $\mathrm{v(t) = n(t) * g_R(t)}$. Finally the sampler produces

$$y_k = \mathrm{y(kT)} = \sum_{-\infty}^{+\infty} a_n \mathrm{x(kT - nT) + v(kT) = x_0 a_0 + ISI + v_k} \tag{2.4}$$

Here the pulses are not sufficiently bounded in the time domain due to the finite bandwidth. This causes neighboring samples to leak into the intended output of the sampler, causing an additional source of noise known as inter-symbol interference. An added source of unpredictability is usually the unknown and time varying nature of the non-constant channel spectrum C(f). In theory the whole symbol sequence contributes to ISI, however, in practice we can truncate it to the L closest neighbors.

One more thing to note here is that the noise is no longer white and this will significantly hinder any efforts to evaluate the performance of the various equalization or estimation techniques. A noise whitening filter will be discussed and worked into the overall procedure shortly. But let's take things one step at a time and, for starters, consider a distortionless channel.

## 2.2 Design of Bandlimited Signals for Zero or Controlled Intersymbol Interference (ISI)

**2.2.1 Zero ISI:** That is,

$$\mathrm{x(nT)} = \begin{cases} 1 & \mathrm{n = 0} \\ 0 & \mathrm{n \neq 0} \end{cases} \tag{2.5}$$

The necessary and sufficient condition for the Fourier transform X(f) in order for x(t) to satisfy the above criterion is known as the Nyquist pulse shaping criterion or the Nyquist condition for zero ISI and it is stated as $\sum_{-\infty}^{+\infty} \mathrm{X}\left(\mathrm{f} + \frac{\mathrm{m}}{\mathrm{T}}\right) = \mathrm{T}$. The proof is easily extrapolated by following the steps used to prove its more notable sibling, the Nyquist sampling theorem, to reach

$$\mathrm{Z(f)} = \sum_{-\infty}^{+\infty} \mathrm{X}\left(\mathrm{f} + \frac{\mathrm{m}}{\mathrm{T}}\right) = \sum_{-\infty}^{+\infty} \mathrm{z_n e^{j2\pi nfT}} \tag{2.6}$$

From there,we'll notice that $z_n = Tx(-nT)$ which of course yields the desired transformation of the zero ISI signal to the Nyquist pulse shaping criterion.

Note that unlike the Nyquist sampling theorem, this time we will aim for overlapping replicas of X(f) (aliasing) in our effort for a flat frequency response Z(f)=T. In more detail, suppose the channel has bandwidth W, that is to say X(f)=0 for |f|>W

1. For $\frac{1}{T} > 2W$ the replicas are non overlapping therefore there is no way to ensure Z(f)=T.
2. For $\frac{1}{T} = 2W$(Nyquist rate) only X(f)=TΠ(f/2W) can provide the desired response. However the sampling function is non-causal and therefore non-realizable. In practice it can be approximated by a sufficiently delayed and truncated version. Yet it should be avoided all together as its tail decays extremely slowly, as 1/t. Hence a small mistiming error in the sampling will result in an infinite non convergent sum of ISI components.
3. Finally for $\frac{1}{T} < 2W$ we get overlapping replicas, which allow for numerous feasible choices of X(f) that satisfy Z(f)=T.

A widely used spectrum is the raised cosine

$$X_{rc} = \begin{cases} T, & 0 \le |f| \le \frac{1-\beta}{2T} \\ \frac{T}{2}\left[1 + \cos\frac{\pi T}{\beta}\left(|f| - \frac{1-\beta}{2T}\right)\right], & \frac{1-\beta}{2T} \le |f| \le \frac{1+\beta}{2T} \\ 0, & |f| > \frac{1+\beta}{2T} \end{cases} \qquad (2.7)$$

which of course is a modified, smoother, version of TΠ(f/2W) and converges to it for $\beta \to 0$. The excess bandwidth beyond the Nyquist frequency is usually expressed as a percentage, represented by the roll-off factor β ($0 \le \beta \le 1$).

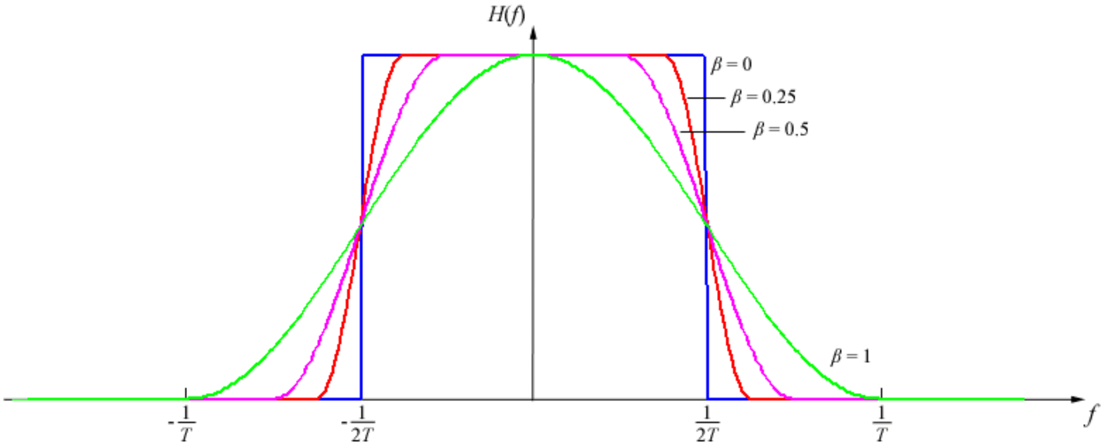


**Figure 2.2: Raised-cosine spectrum**

In the time domain

$$x(t) = \mathrm{sinc}\left(\frac{t}{T}\right)\frac{\cos\left(\frac{\pi\beta t}{T}\right)}{1-\frac{4\beta^2 t^2}{T^2}} \tag{2.8}$$

with its tails decaying as $\frac{1}{t}\frac{1}{t^2} = \frac{1}{t^3}$, which leads to a summable ISI series in case of mistiming. Practical transmitter and receiver filters approximating the overall desired behavior can easily be designed. And ideally $G_T(f) = G_R^*(f) = \sqrt{|X_{rc}(f)|}e^{-j2\pi f t_0}$ (matched filter for maximum SNR).

**2.2.2 Controlled ISI:** Obviously zero ISI is achieved at the expense of the transmission rate. But if we were to relax the zero ISI constraint we could raise the symbol rate to the Nyquist rate. Two commonly used choices are

$$x(nT) = \begin{cases} 1 & n = 0{,}1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad x(nT) = \begin{cases} 1 & n = 1 \\ -1 & n = -1 \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

, i.e. $Z(f) = T + Te^{-j2\pi fT}$ or $x(t) = \mathrm{sinc}(2Wt) + \mathrm{sinc}(2W(t-T))$ for $T = \frac{1}{2W}$, and

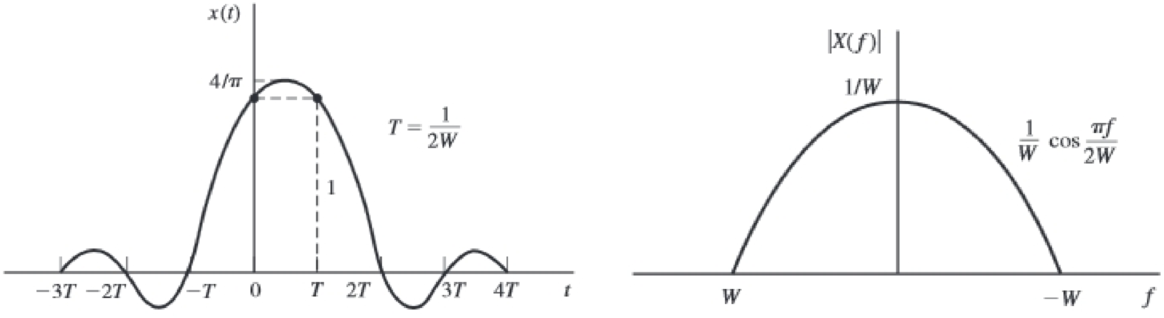


**Figure 2.3: Duobinary Signal Pulse in the time and frequency domain**

$$Z(f) = Te^{+j2\pi fT} - Te^{-j2\pi fT} \text{ or } x(t) = \mathrm{sinc}(2W(t+T)) - \mathrm{sinc}(2W(t-T)) \text{ for } T = \frac{1}{2W}$$

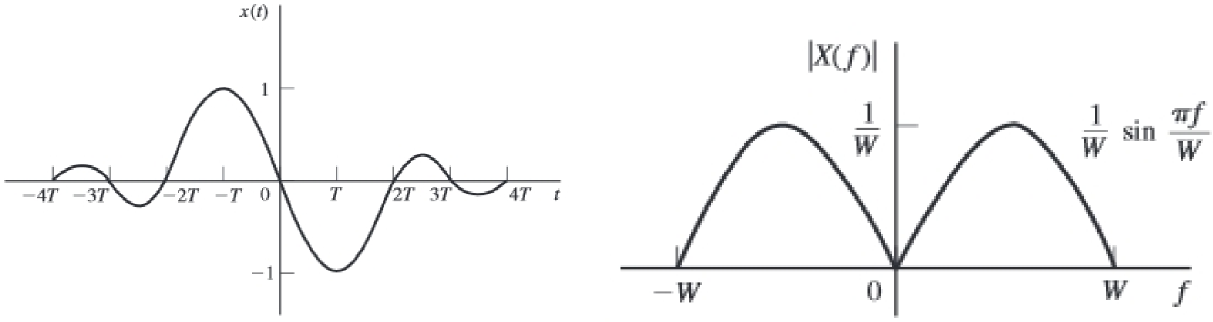


**Figure 2.4: Modified Duobinary Signal Pulse in the time and frequency domain**

which are known as the duobinary signal pulse and the modified duobinary signal pulse, respectively. Both are (approximately) physically realizable, since filters can be designed that follow their smooth spectrum very closely. We will note that the modified duobinary's spectrum has a zero for f=0, making it applicable to channels that do not propagate signals with frequency equal to zero (static signals).

In practice a more generalized partial response signal,

$$\mathrm{X(f)} = \frac{1}{2\mathrm{W}}\sum \mathrm{x}(\frac{\mathrm{n}}{2\mathrm{W}})\mathrm{e}^{-\mathrm{j}\mathrm{n}\pi\mathrm{f}/\mathrm{W}} \text{ for } |\mathrm{f}| \leq \mathrm{W} \tag{2.10}$$

can be utilized. However, as we select more nonzero elements, the problem of unraveling the controlled ISI becomes more cumbersome and impractical.

### 2.2.3 Data detection for controlled ISI:

The easiest method to implement is **symbol by symbol detection**. A direct implementation e.g. for the duobinary signal pulse

$$\mathrm{y_m} = \mathrm{b_m} + \mathrm{v_m} = \mathrm{a_m} + \mathrm{a_{m-1}} + \mathrm{v_m} \tag{2.11}$$

, where let us ignore the noise for the moment, would plainly be to subtract the last detected symbol from the received sum.

However, errors arising from the additive noise would propagate. Luckily there is an easy fix. Precoding of the original data at the transmitter instead of subtracting the estimate at the receiver can eliminate error propagation**. In more detail:**

Let the M-level data sequence $\{d_n\}$, with possible values 0,1,2,…,M-1 and the precoding rule

$$\mathrm{p_m} = \mathrm{d_m} - \mathrm{p_{m-1}} \text{ (mod M)} \tag{2.12}$$

, followed by the renormalization

$$\mathrm{a_m} = 2\mathrm{p_m} - (\mathrm{M}-1) \tag{2.13}$$

Which will of course translate them into the M-ary PAM levels.

This will give rise to the received (noise -free) sequence

$$\mathrm{b_m} = \mathrm{a_m} + \mathrm{a_{m-1}} = 2[\mathrm{p_m} + \mathrm{p_{m-1}} - (\mathrm{M}-1)\,] \tag{2.14}$$

Now we only need to notice that

$$\mathrm{d_m} = \mathrm{p_m} + \mathrm{p_{m-1}}(\text{mod M}) = \frac{\mathrm{b_m}}{2} + (\mathrm{M}-1)(\text{modM}) \tag{2.15}$$

In the presence of noise, the received sequence is quantized to the nearest possible signal level before applying the aforementioned rule.

A similar procedure can now be derived for the modified duobinary pulse by noticing that $\mathrm{b_m} = \mathrm{a_m} - \mathrm{a_{m-2}}$. Therefore, the following precoding rule $\mathrm{p_m} = \mathrm{d_m} + \mathrm{p_{m-2}}$ (mod M) would serve to evaluate $\mathrm{p_m} - \mathrm{p_{m-2}}$ in the modified duobinary counterpart. And as a result $\mathrm{d_m} = \frac{\mathrm{b_m}}{2}(\text{modM})$.

However, the symbol by symbol detection ignores the inherent memory contained in the signal (because of intersymbol interference) and therefore should be expected to be suboptimal. The optimal method would of course be the **maximum- likelihood sequence detection**. Which would lead to none other than a Viterbi search on the corresponding trellis. Of course, since the noise samples are no longer uncorrelated, the metrics will no longer be Euclidean in nature. Even though sequential trellis search algorithms for correlated noise have been developed, the suboptimal procedure of ignoring the correlation altogether is not uncommon.

Next we will raise the bar and incorporate the effects of channel distortion into our design.

## 2.4 System Design in the Presence of Channel Distortion

A system, here a channel, causes distortion when its amplitude is not constant in its respective bandwidth and/or its phase is nonlinear. A common practice is instead of phase distortion to use the envelope delay, $\tau(f) = -\frac{1}{2\pi}\frac{d\theta_c(f)}{df}$, to define delay distortion as the distortion suffered whenever $\tau$ is not constant.

Channel characteristics might be known, unknown but stable over a period of interest, like a telephone line whose path changes with every connection but is the same until the end of that connection, or even time varying altogether, like a wireless connection. In the following section we will consider system design for known and unknown channel characteristics.

### 2.4.1 Known channel characteristics :

With the knowledge we have accumulated up to this point we can design transmitting and receiving filter functions that maximize SNR and result in zero ISI. Obviously in our pursuit for zero ISI we aim for $G_T(f)C(f)G_R(f) = X_{rc}(f)e^{-j2\pi ft_0}, |f| \leq W$. Then, the noise spectrum will also be transformed into $S_v(f) = S_n(f)|G_R(f)|^2$.

For M-PAM with distance 2d between levels, the probability of error is

$P_e = \frac{2(M-1)}{M}Q(\sqrt{\frac{d^2}{\sigma_v^2}})$, which is minimized by maximizing the $\mathrm{SNR} = d^2/\sigma_v^2$. As we can see

$$P_{av} = \frac{E(a_m^2)}{T}\int_{-\infty}^{+\infty} g_T^2(t)dt = \frac{(M^2-1)d^2}{3T}\int_{-\infty}^{+\infty} g_T^2(t)dt \tag{2.16}$$

or $\frac{1}{d^2} = \frac{(M^2-1)}{P_{av}3T}\int_{-\infty}^{+\infty}\left|G_T^2(f)\right|df$. Hence,

$$\frac{\sigma_v^2}{d^2}=\frac{(M^2-1)}{P_{av}3T}\int_{-W}^{W}S_n(f)|G_R(f)|^2df\int_{-\infty}^{+\infty}\frac{|X_{rc}(f)|^2}{|C(f)|^2|G_R(f)|^2}df \qquad (2.17)$$

and by applying the Cauchy-Schwartz inequality we find that the filters that minimizes the noise-to-signal ratio are

$|G_R(f)|=K\frac{|X_{rc}(f)|^{1/2}}{[S_n(f)]^{1/4}|C(f)|^{1/2}}$ , $|f|\le W$ and $|G_T(f)|=\frac{1}{K}\frac{|X_{rc}(f)|^{1/2}[S_n(f)]^{1/4}}{|C(f)|^{1/2}}$ , $|f|\le W$

,with $\theta_T(f)+\theta_c(f)+\theta_R(f)=2\pi f t_o$ (phase distortion is perfectly compensated).

### 2.4.2 Unknown channel characteristics and Equalization:

When dealing with an unknown, yet time invariant, channel, methods of lower complexity can be developed, by measuring its response to an input sequence of finite length, in comparison to the adaptive equalization techniques that are employed for time varying systems.

---

#### Maximum-likelihood sequence detection and noise whitening

Whilst following the usual procedure for designing a maximum-likelihood receiver for AWGN, we reach the metric

$$\mathrm{CM}(\mathbf{a})=2\mathrm{Re}\left(\sum_n a_n^*\int_{-\infty}^{+\infty}r(t)\,h^*(t-nT)dt\right)-\sum_n\sum_m a_n^*a_m\int_{-\infty}^{+\infty}h^*(t-nT)h(t-mT)\,dt=$$

$$2\mathrm{Re}\left(\sum_n a_n^*y_n\right)-\sum_n\sum_m a_n^*a_m x_{n-m} \qquad (2.18)$$

, which as expected demands the use of a matched (to the combined transmitter and channel filter) filter at the receiver. All of the equations' constituents are of course real, but keeping a generalized notation will pay off shortly.

If we now make the reasonable assumption that in any practical system ISI is affected only by a finite number of symbols, let's say 2L, the system can be described as a discrete-time transversal filter that spans a time interval of 2LT seconds.

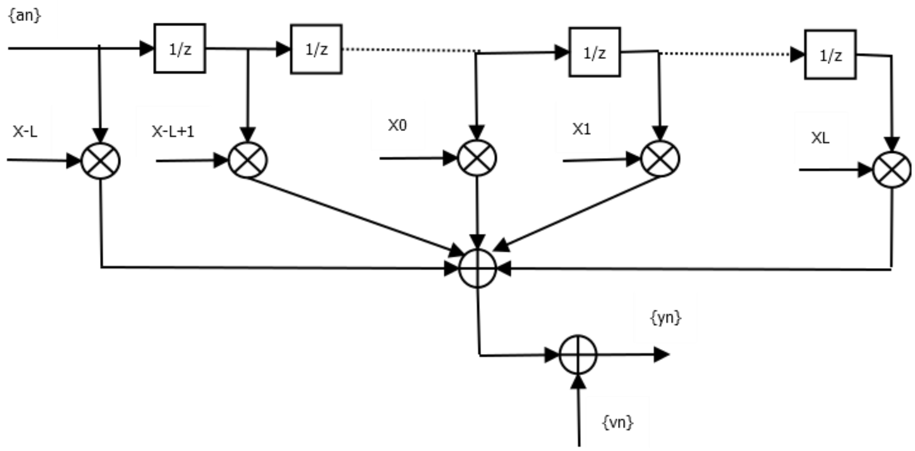


**Figure 2.5: Equivalent discrete-time system model**

Of special interest to us would then be the two-sided z transform $X(z) = \sum_{-L}^{L} x_k z^{-k}$,for which we should notice $x_k = x^*_{-k}$. Therefore, $X(z) = X^*(z^{-1})$, from which we can easily infer that if $\rho$ is a root ,$1/_{\rho^*}$ is also a root. Hence, $X(z) = F(z)F^*(z^{-1})$ , from where we'll proceed to uniquely define $F^*(z^{-1})$ as the minimum phase polynomial (the polynomial having its roots inside the unit circle), thereby making $1/F^*(z^{-1})$ a physically realizable, stable, recursive discrete-time filter. With minimal effort it can also be shown that $E(v_k^* v_j) = \frac{N_o}{2} x_{k-j}$. Therefore, following the whitening transformation $\mathbf{H} = (\mathrm{Cov}(\mathbf{V}))^{-1/2}\mathbf{V}$ **,** it turns out this is the noise whitening filter we need to cascade the transmitting filter, channel, matched filter and sampler combination with, to receive the equivalent discrete-time white noise filter model, i.e. the L root polynomial $F(z)$.

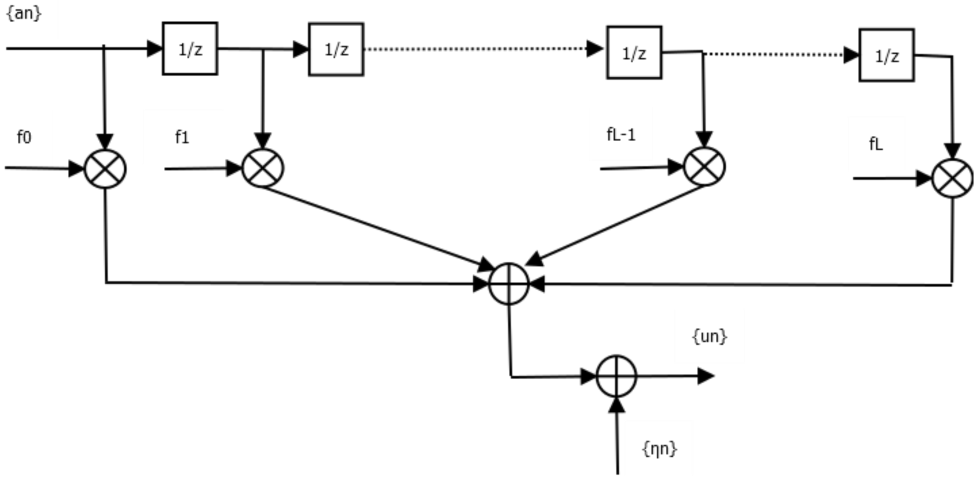


**Figure 2.6: Equivalent discrete-time white noise system model**

From here on,the Viterbi algorithm can be used to determine the most probable path through the $M^L$-state trellis, as usual. However, the computational complexity grows exponentially with the length of the channel time dispersion ($M^{L+1}$ metrics have to be computed for each new symbol) and in most channels of practical interest the complexity is too expensive to implement. Instead, suboptimum methods are usually employed, such as the linear transversal filters that we'll describe next. These structures have a computational complexity that is only a linear function of L.

---

## 1 Linear Equalizers:

Linear equalizers are linear transversal filters with adjustable coefficients $c_n$. The law of electing the most appropriate tap coefficients is of course determined by the chosen performance criterion. And even though the probability of error is the most meaningful measure of performance, the corresponding function is highly nonlinear. Other criterions are usually employed in its stead.

a. **Peak Distortion Criterion** is simply defined as the minimization of the worst–case inter-symbol interference at the output of the equalizer .

   We observe that the cascade of the linear F(z) and the equalizer having an impulse response $\{c_n\}$ is $q_n = \sum c_j f_{n-j} = c_n * f_n$. Its output, if for convenience we normalize $q_0$ to unity (after all its nothing more than a universal constant gain factor), will be $$\widehat{a_k} = a_k + \sum\nolimits_n' a_n q_{k-n} + \sum c_j \eta_{k-j} \qquad (2.19)$$

   The peak distortion metric is defined as $D(\mathbf{c}) = \sum_n' |q_n|$ and with an equalizer having an infinite number of tap weights it can be minimized to zero. That is,

   $$q_n = \sum c_j f_{n-j} = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} \qquad (2.20)$$

   can be uniquely solved through the z-transform $Q(z) = C(z)F(z) = 1$ or $C(z) = {}^{1}/_{F(z)}$. This is called a **zero-forcing filter**. An expected, since the cascade of the whitening and equalizer filters gives us $C'(z) = \frac{1}{F(z)F^*(z^{-1})} = \frac{1}{X(z)}$ (the filter 's tap coefficients can be calculated from the inverse z-transform by complex integration), and pretty much indifferent result except for its usefulness in calculating the signal to noise ratio.

   For a finite-length equalizer, with 2K+1 taps, the peak distortion has been shown to be a convex function of **c** and its minimization can be carried out numerically. However, when the inter-symbol interference is not severe enough to close the eye the peak distortion is minimized by the zero-forcing solution in the range $1 \leq |n| \leq K$.

Before we carry on with our analysis we should **note** that the channel characteristics are generally unknown before the design process begins and a common practice is to design the transmitting and receiving filters to follow $|G_T(f)||G_R(f)| = |X_{rc}(f)|$. This allows for the simplification of the system to the discrete-time **channel only** filter which describes the ISI suffered due to the channel distortion, which in turn is the only source of ISI in this design. This leads to suboptimal but also more elementary results. For the zero-forcing equalizer, if x(t) is the channel distorted pulse $q(mT) = \sum_{-K}^{K} c_n x(mT - n\tau)$ and $\mathbf{c_{opt}} = \mathbf{X^{-1}q}$.

b. **Mean Square Error Criterion** A serious drawback of the zero-forcing equalizer is that it completely ignores the presence of additive noise and this might result in significant noise enhancement if the system presents a serious dip within some frequency range as it will try to compensate for it in order to level the frequency response.

   If we relax the zero-ISI condition, a filter can be designed around the minimization of the combined ISI and noise power at the output of the equalizer. This will result in the $\mathrm{MSE} = E[\widehat{a_k} - a_k]^2$ cost function. The minimization is obviously achieved when the error is orthogonal to the signal sequence. Equivalently $\sum c_j E\left(u_{k-j} u_{k-l}^*\right) = E(a_k u_{k-l}^*)$ (2.21)
   where $u_n$ the output of the whitening filter. For the infinite tap case and iid data sequence $\{a_n\}$ this will yield

$$C(z) = \frac{F^*(z^{-1})}{F(z)F^*(z^{-1}) + N_o} \text{ or } C'(z) = \frac{1}{F(z)F^*(z^{-1}) + N_o} = \frac{1}{X(z) + N_o}$$

   But once more, beside the approximation of the SNR (here of the residual ISI plus noise energy as well), the finite tap solution is the one that interests us the most. For this case $\mathbf{c_{opt}} = \boldsymbol{\Gamma}^{-1}\boldsymbol{\xi}$ where $\Gamma_{lj} = R_U[l - j] = \text{for iid} \begin{cases} x_{l-j} + N_0 \delta_{ij}, & |l - j| \leq L \\ 0 & , \text{otherwise} \end{cases}$ and $\xi_l = R_{AU}[l] = \text{for iid} \begin{cases} f_{-l}^*, & -L \leq l \leq 0 \\ 0 & , \text{otherwise} \end{cases}$. Also $J(K) = 1 - \boldsymbol{\xi}^{\mathbf{T}*}\boldsymbol{\Gamma}^{-1}\boldsymbol{\xi}$ (2.22)

We will once again **note** that if we had used the discrete time **channel** filter instead, the result would plainly be $\sum_{-K}^{K} c_n R_Y[n - k] = R_{AY}[k]$. We will also note that in practice the statistics are unknown but the correlation functions can be adequately estimated by transmitting and measuring a test signal.

$$\widehat{R_Y}[n] = \frac{1}{N}\sum_{1}^{N} y(kT - n\tau)y(kT) \text{ and } \widehat{R_{AY}}[n] = \frac{1}{N}\sum_{1}^{N} y(kT - n\tau)a_k \qquad (2.23)$$

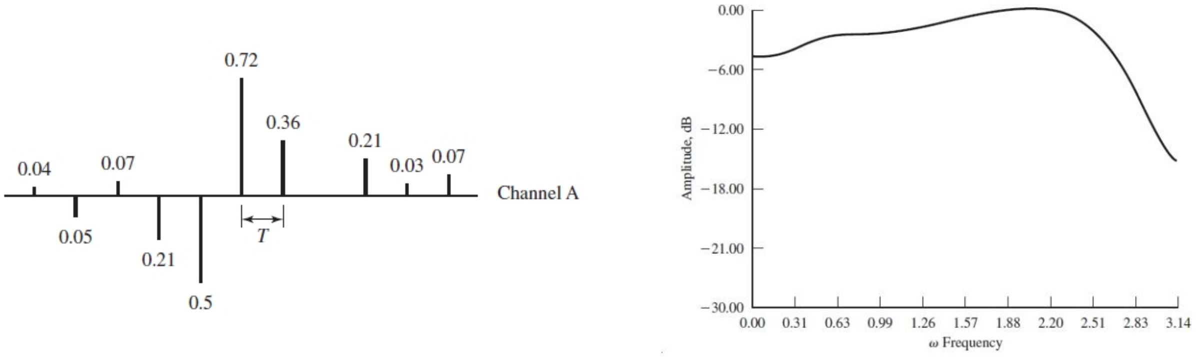
Lastly the iid hypothesis is unnecessarily restrictive. An uncorrelated symbol sequence would be just as good. But more importantly, if the unrealistic expectation of uncorrelated symbols were to strike the reader as a rather odd choice, we would have to remind them that that *theoretical* design concerns the equalization of a known channel and that the statistics of the signal should, ideally, be irrelevant to the design of a flat spectrum for the system at that point.

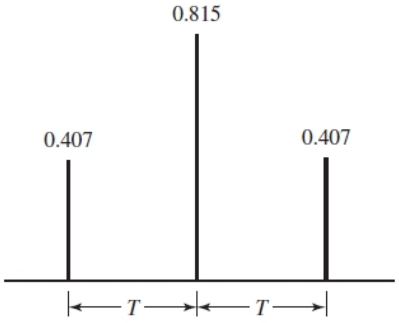
## Symbol and Fractionally-spaced Equalizers

The keen reader might have noticed that on occasion we have selectively used a different sampling period $\tau$than the symbol period T. When the channel characteristics are known and the receiver can be matched to the cascade of the transmitter and channel filter, the signaling frequency is the optimal tap spacing. However, in the suboptimal case where the receiver is only matched to the transmitter, the channel distorted signal is still aliased by the sub Nyquist symbol rate used for the transmitter-receiver zero ISI signal design. If the input to the equalizer is sampled on kT+$\tau_0$ its spectrum would be $Y_T(f) = \frac{1}{T}\sum X(f - \frac{n}{T})e^{j2\pi(f-{}^{n}/_{T})\tau_0}$ and the equalizer would compensate for the frequency characteristics of the aliased signal and not directly for the channel distortion characteristics inherent in $X(f)e^{j2\pi f\tau_0}$.

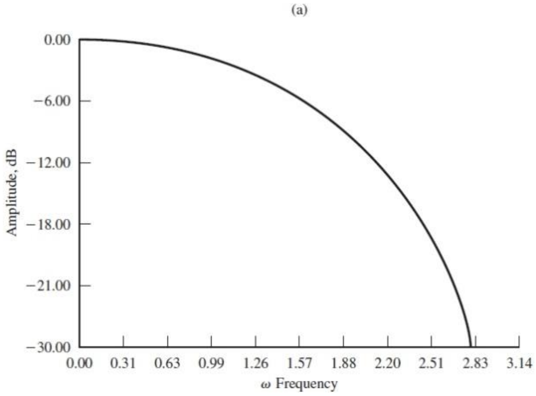
On the other hand a fractionally spaced equalizer, i.e. an equalizer sampling at a fraction of the symbol period $\frac{1}{\tau} = 2F_{max} = \frac{1+\beta}{T}$, at least as fast as the Nyquist rate, will prevent the folding of the spectrum and the equalizer will compensate for channel distortion before the aliasing effects.

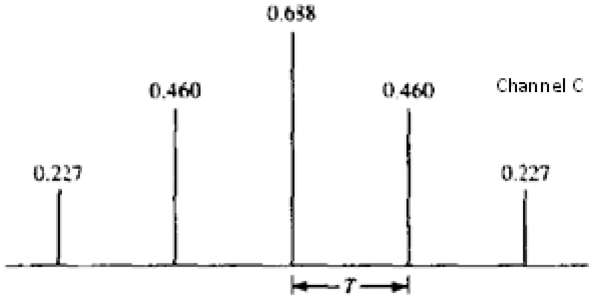
**Figure 2.7: Dicrete-time channel models A,B,C and their amplitude spectra**



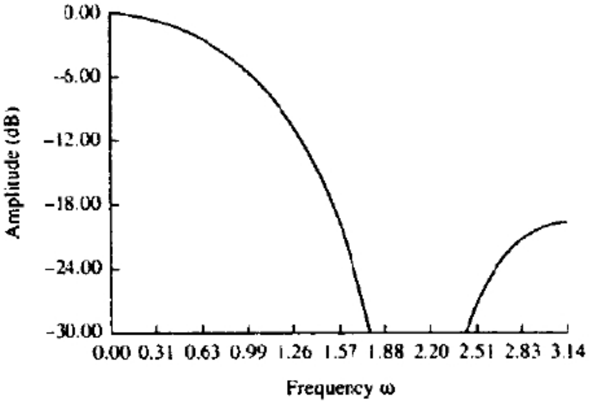

0.815
0.407
0.407
T
T

Channel B




0.688
0.460
0.460
0.227
0.227
T

Channel C



## 2 Decision Feedback Equalizers:

As we have already mentioned, the main reason linear equalizers are suboptimal compared to the trellis search is that they ignore the inherent memory in the data sequence. Decision feedback equalizers try to mend that by enlisting the last few detected symbols to try and remove their ISI from the signal. Their structure involves passing the sampled signal through a feed forward filter, which is a simple linear equalizer, and subtracting a normalized sum of the last $K_2$ detected symbols from its output before attempting to detect the next symbol.

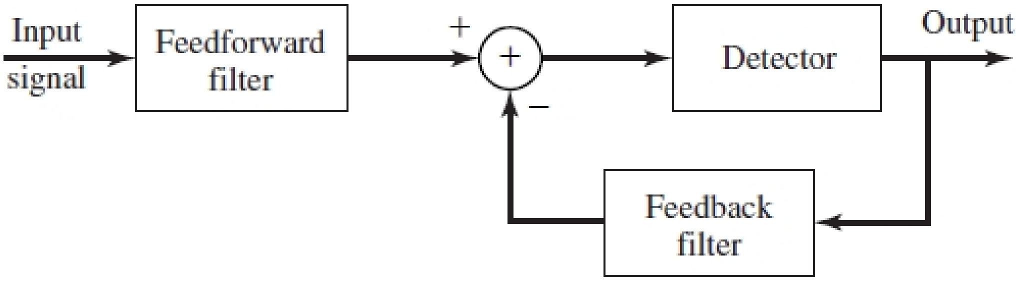

Input
signal
Feedforward
filter
+
+
−
Detector
Output
Feedback
filter

**Figure 2.8: Decision Feedback Equalizer block diagram**

$$\widehat{a_k} = \sum\nolimits_{-K_1}^{0} c_j u_{k-j} + \sum\nolimits_{0}^{K_2} c_j \widetilde{a_{k-j}} \qquad (2.24)$$

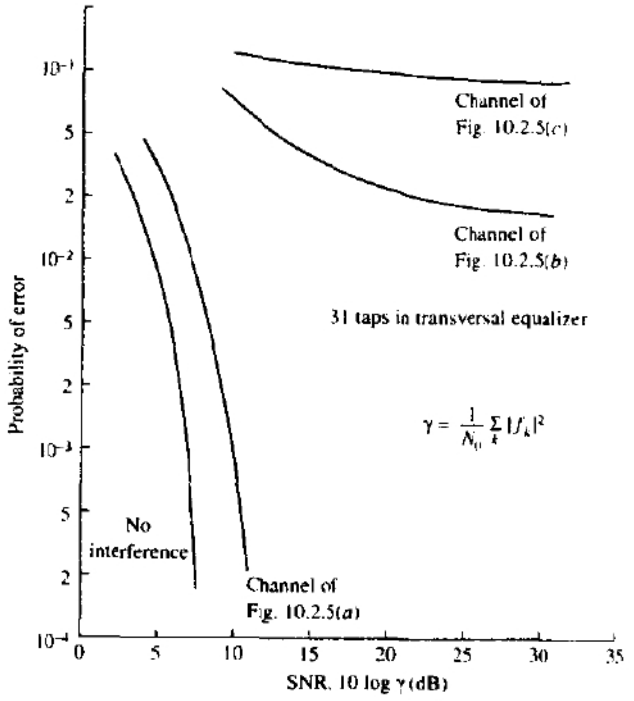
Both the peak distortion and the MSE criterion lead to mathematically tractable solutions for the tap coefficients. But the MSE is more prevalent in practice for the aforementioned reasons. Based on the assumption that the previously detected symbols are correct, it yields the same solution for the feed forward filter and, at least for uncorrelated sequences, the feedback filter coefficients are given in terms of the coefficients of the feed forward section by $c_k = -\sum_{-K_1}^{0} c_j f_{k-j}\, , k = 1,2, \dots , K_2$.

Decision feedback equalizers severely outperform linear equalizers, and are really the only option, for channels that present a big dip, or even a spectral null, within their bandwidth. This will manifest in the time domain as severe, not necessarily spanning many symbols, ISI. Figure 2.9 through 2.11 will compare the performance of a linear MSE and a decision feedback equalizer of similar complexity on channels B,C, which can be seen in figure 2.7. And will also quantify the DFE’s slightly suboptimal performance compared to the ML sequence estimator’s performance. The linear equalizer will be shown to be adequate for channel A.

# 3 Adaptive equalizers:

When the channel is time-varying we have no choice but use equalizers with on-line adaptable tap coefficients .Even when dealing with time invariant channels, matrix inversion operations are avoided and recursive algorithms, such as the method of the steepest decent, are used to train the tap coefficients. Therefore the first step in the training of the adaptive equalizer will be shared by all of the aforementioned filters, too.

Adaptive models for both the peak distortion and MSE criterion are available.



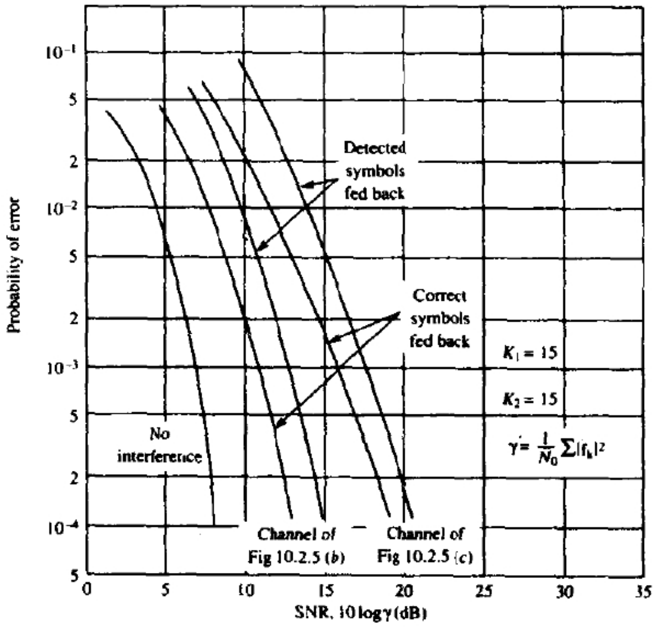


**Figure 2.9: Error rate performance of linear 31 tap MSE equalizer**

**Figure 2.10: Error rate performance of DFE with $K_1$=15, $K_2$ =15 taps**

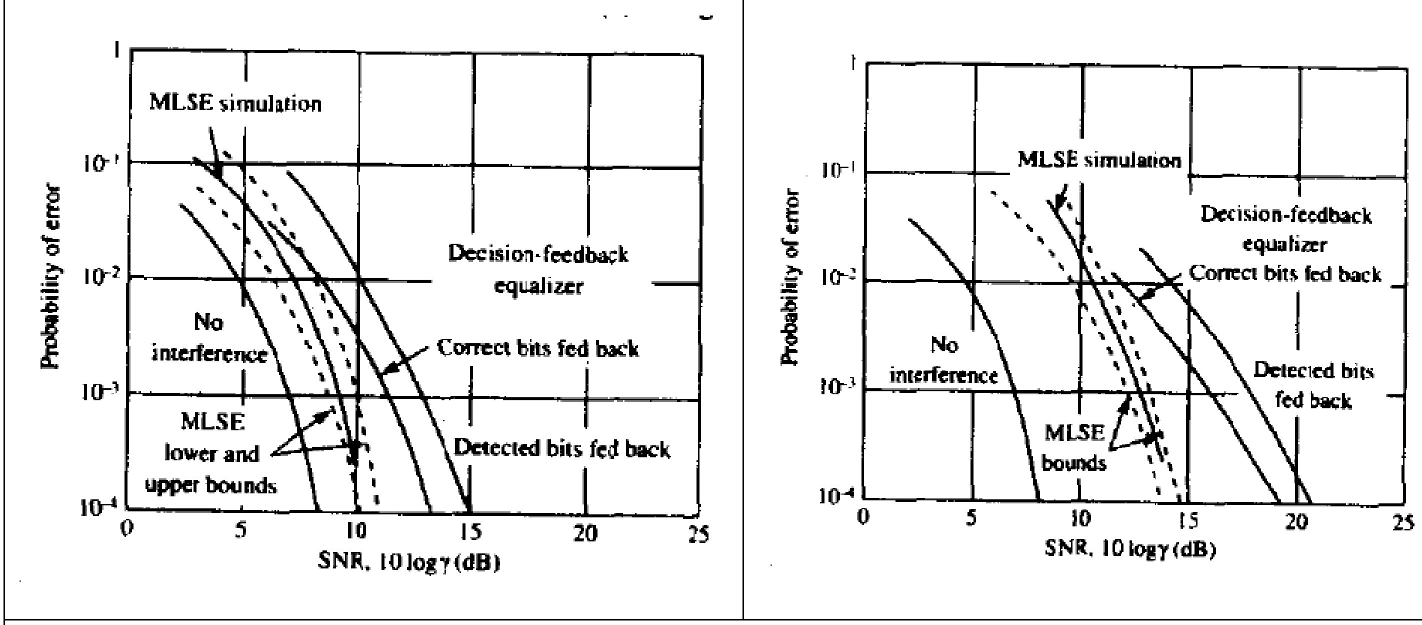


**Figure 2.11: Performance comparison between MLSE simulation and DFE, both for the correct and estimated symbols feedback**

**a. Zero-forcing Algorithm** The idea is, whenever the ISI is not too severe, to zero-force the cross-correlation between the error sequence and the data sequence. $\mathrm{E}(\varepsilon_k a_{k-j}^*) = 0$ , $\varepsilon_k = a_k - \widehat{a_k}$. If the symbols are uncorrelated and uncorrelated with the additive noise sequence $\mathrm{E}(\varepsilon_k a_{k-j}^*) = \delta_{j0} - q_j$ , $j = -K, \dots, K$ -equivalent to the known criterion for peak distortion minimization. The first step would be to train the network using a predetermined sequence and the simple recursive algorithm $c_j^{(k+1)} = c_j^{(k)} + \Delta\varepsilon_k a_{k-j}^*$ (2.25) where $\Delta$ a scale factor that controls the rate of adjustment and $\varepsilon_k a_{k-j}^*$ a stand-in estimate for the unknown cross-correlation.

Once the training period has ended the decisions $\widetilde{a_k}$ (the output of the detector, not to be confused with the estimates who are the output of the equalizer ) are sufficiently reliable and will be used instead of the predetermined input sequence. This is called the **decision-directed mode** of adaptation .

For a high enough ISI the zero forcing condition is no longer the solution and a universal optimization technique will be required. At which point we might as well make use of the superior and mathematically elegant MSE criterion .

**b. The LMS Algorithm** is the online or recursive version of the MSE algorithm, which is a batch processing method that requires matrix inversions. The method of steepest decent will be used instead to reach the minimum of the quadratic, therefore convex, cost function J.

The method of steepest decent changes the point **c<sub>k</sub>** in the direction opposite to the direction of maximum increase of the cost function at that point, i.e. the gradient $\mathbf{g_k} = \frac{1}{2}\frac{\partial J}{\partial \mathbf{c_k}}$. For the MSE the gradient is of course $\mathbf{g_k} = -\mathrm{E}[\varepsilon_k \mathbf{U_k^*}]$ and will, once more, be approximated by $\widehat{\mathbf{g_k}} = -\varepsilon_k \mathbf{U_k^*}$. Therefore $\mathbf{c_{k+1}} = \mathbf{c_k} + \Delta\varepsilon_k \mathbf{U_k^*}$. After the training period the decided symbols will be trusted to estimate the error $\widetilde{\varepsilon_k} = \widetilde{a_k} - \widehat{a_k}$ .

Several other variations of the LMS are obtained by different gradient estimates. Popular choices are the average of the last N, $-\varepsilon_k \mathbf{U_k^*}$vectors, $\overline{\widehat{\mathbf{g_k}}}$ and a weighted average $\overline{\widehat{\mathbf{g_k}}} = w\overline{\widehat{\mathbf{g_{k-1}}}} + (1-w)\widehat{\mathbf{g_k}}$, that will act as a lowpass filter for the noisy gradient.

**c. Decision-Feedback Equalizer** The steepest decent algorithm will be applied unchanged $\mathbf{c_{k+1}} = \mathbf{c_k} + \Delta\mathrm{E}[\varepsilon_k \mathbf{U_k^*}]$, however, in accordance with the assumption that the previously detected symbols are correct, the latter half of $\mathbf{U_k^*}$ will be composed of the relevant training sequence or detected symbols, for the training and decision- directed mode respectively.

Many more adaptive filter algorithms are applicable to the problem of channel equalization which, in essence, isn't any different from the problems pertaining to other disciplines that develop implementations for, or employ the services of, recursive algorithms such as the discipline of pattern recognition, neural networks and learning machines.

## 2.5 Summary

Since the subject of this thesis will revolve around DFE's this chapter serves a detailed introduction to the theory of equalizers in general. Specifically, we have presented the reasons and situations that make the need for channel equalization imperative. We have discussed popular methodologies for linear and linear adaptive equalizers. And we have also explained the need for decision feedback equalizers (and their adaptive counterparts), by comparing their performance with that of linear equalizers of the same size for various channels (most importantly for channels with a spectral null). For a deeper exposition, [1] and [2] by Proakis and Salehi are an indispensable read for anyone with an interest in communications.

# Fast Adder and Multiplexer Implementations 3

## Speed and Area Estimates

Digital filters are composed of adders, multipliers and delay elements (flip-flops). In the next part of the thesis, we will transform multipliers or even a whole part of the transverse filter into a composite multiplexer. Hence, any exposition on multiplier design can be avoided, while speed and area estimations of adder and multiplexer designs will be instrumental to our reformulation strategies. In this chapter we will aim to produce the necessary speed and area estimates for our application.

## 3.1 Fast Adder Implementations

### 3.1.1 Ripple Carry Adder

The throughput speed-up of adder units covers an extremely diverse array of topics. From the low-level redesign of full adder cells to carry prediction schemas.

The **HA** equations every engineer should be familiar with are

$$s = a \oplus b \text{ and } c = ab \qquad (3.1)$$

Two such HA cells are needed for the full a,b,$c_{in}$ addition. The simplest adder design, **the ripple-carry adder**, is simply a cascade of **FA** cells which means that the delay is bounded by the carry propagating through the whole chain. Even for such a simple design, the FA cell equations can be reformulated in a way that logic is shared by sum and carry, the schematic slightly rerouted from the complementary p,n-mos network design and the layout optimized, for a faster path from input to carry. As an added bonus, the inverter that follows the output of every cmos function design can be removed if we notice that complementary inputs produce complementary outputs. Hence, inversions can be shifted away from the critical path as can be seen in figure 3.1.

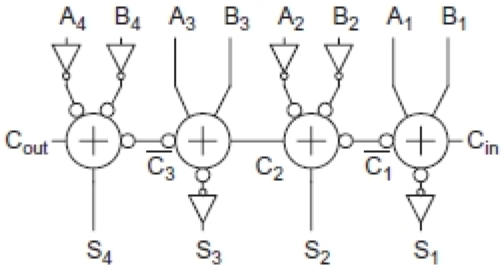


**Figure 3.1: Ripple carry adder with the inverters shifted away from the critical path**

Technologies faster than cmos can also be implemented for the physical standard-cell library. But in the end, long ripple carry chains, cannot be fast enough to rival the higher level designs that use the same technology for their constituents. The development of a common descriptive language would benefit the quantification of such comparisons.

### Generate, Propagate Equations and Canonical Design

In order to develop a common platform that will allow us to evaluate the performance of every adder design, the **generate and propagate equations** will be introduced.

$$G_{i:j} = G_{i:k} + P_{i:k}G_{k-1:j} \tag{3.2}$$

$$P_{i:j} = P_{i:k}P_{k-1:j} \tag{3.3}$$

where $G_{i:i} = G_i = A_iB_i$ and $P_{i:i} = P_i = A_i \oplus B_i$ the base case sand the specially treated $G_{0:0} = C_{in}, P_{0:0} = 0$.

These simply tell us that a carry generated from j-to-i must have either been generated from j-to-(k-1) and then propagated to-i or generated from k-to-i and that propagation from j-to-i is simply the cascade of propagation from j-to the intermediate-(k-1) bit plus propagation from k-to-i. As a final note these equations define a valency-2 group PG logic. Higher valency groups can be introduced by replacing $G_{k-1:j}$,$P_{k-1:j}$with their respective equations etc., for fewer levels but slower more complex gates.

Every design can thus be divided into three parts, the bitwise **PG Logic** where the base signals are generated, the **Carry Generator** where the $G_{i:0} = C_i$ are computed and finally the **Sum Logic**.

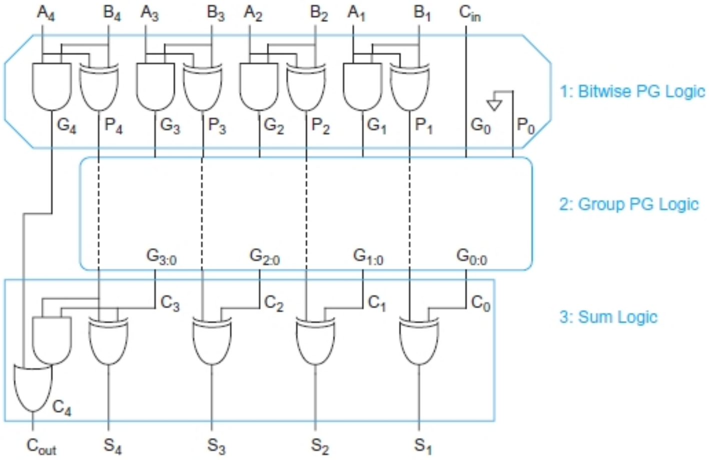

$A_4$
$B_4$
$A_3$
$B_3$
$A_2$
$B_2$
$A_1$
$B_1$
$C_{in}$
1: Bitwise PG Logic
$G_4$
$P_4$
$G_3$
$P_3$
$G_2$
$P_2$
$G_1$
$P_1$
$G_0$
$P_0$
2: Group PG Logic
$G_{3:0}$
$G_{2:0}$
$G_{1:0}$
$G_{0:0}$
$C_3$
$C_2$
$C_1$
$C_0$
3: Sum Logic
$C_4$
$C_{out}$
$S_4$
$S_3$
$S_2$
$S_1$

**Figure 3.2: Canonical Design division**

---

For a ripple carry adder, the carry generator will simply be the chain of the figure below,

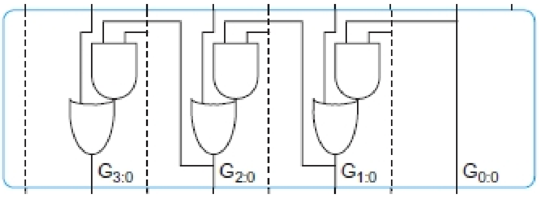

$G_{3:0}$
$G_{2:0}$
$G_{1:0}$
$G_{0:0}$

**Figure 3.3: Ripple carry generator**

which will yield

$$t_{ripple} = t_{pg} + (W - 1)t_{AO} + t_{xor} \tag{3.4}$$

All of the subsequent design attempts will focus around either breaking up the chain or trading area for levels of the carry generator, by computing intermediate generate and propagate signals.

---

For simplicity's sake, since more complex gates will not be attempted, we will obscure low level gates in our schematics by introducing **black** and **grey cells** representing the group PG gates (figure 3.4).

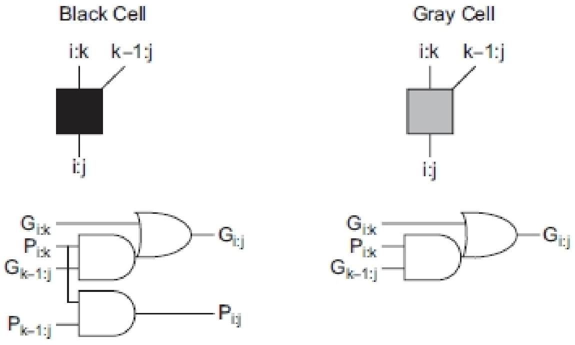

Black Cell
i:k
k–1:j
i:j
Gray Cell
i:k
k–1:j
i:j
$G_{i:k}$
$P_{i:k}$
$G_{k-1:j}$
$P_{k-1:j}$
$G_{i:j}$
$P_{i:j}$
$G_{i:k}$
$P_{i:k}$
$G_{k-1:j}$
$G_{i:j}$

**Figure 3.4: Group PG cells**

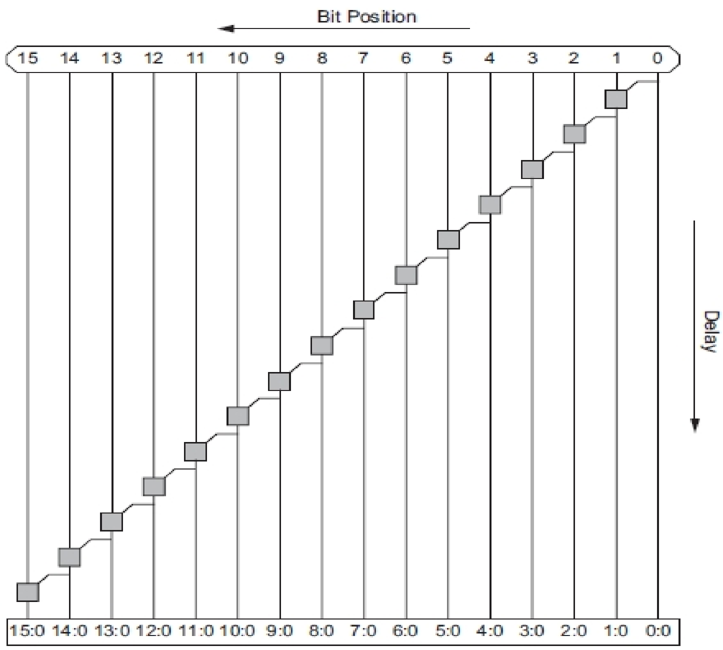

Bit Position
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Delay
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

**Figure 3.5: Carry-ripple adder group PG network**

The only exception to that rule will be a **Manchester carry-chain** of short length where a faster domino gate can be utilized to compute the carries with great speed.

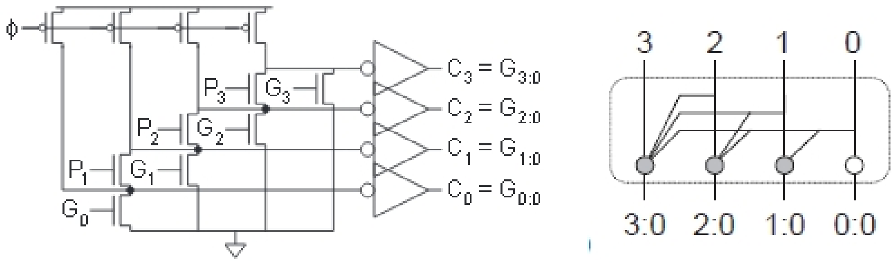

ϕ
$P_3$
$G_3$
$P_2$
$G_2$
$P_1$
$G_1$
$G_0$
$C_3 = G_{3:0}$
$C_2 = G_{2:0}$
$C_1 = G_{1:0}$
$C_0 = G_{0:0}$
3 2 1 0
3:0 2:0 1:0 0:0

**Figure 3.6: Manchester carry-chain and its schematic representation**

### 3.1.2 Carry Skip Adder and Linear Carry Look-Ahead Adder

An elementary, and historically first, attempt to statistically shorten the delay, calls for breaking the word into shorter parts and using a carry propagation predictor, which is simply the and gate of all the $P_i$, to, sometimes, propagate the input rather than the output carry of the previous stage. This design is called a **carry-skip adder** and was proposed by Charles Babbage himself. A Manchester chain can improve upon the objective speed of this scheme.

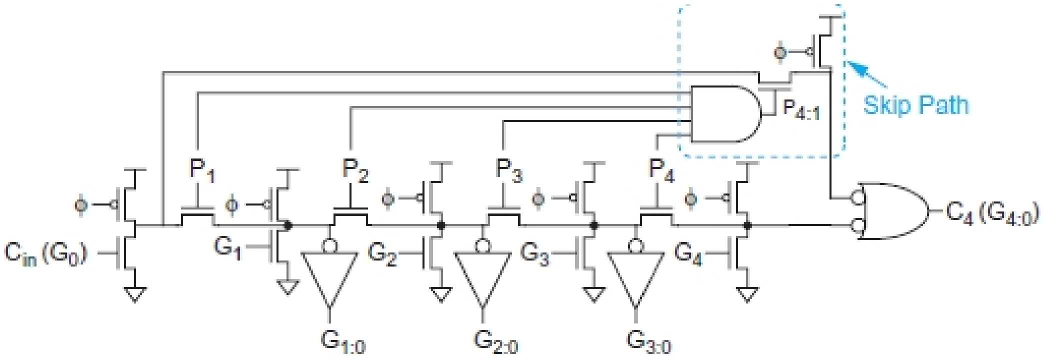

ϕ
Skip Path
P4:1
P1
P2
P3
P4
Cin (G0)
G1
G2
G3
G4
C4 (G4:0)
G1:0
G2:0
G3:0

**Figure 3.7:Propagation predictor with Manchester chain generator**

In the same vein **a linear carry look-ahead** adder utilizes a short and fast PG predictor similar to the Manchester chain by breaking up the word in k groups of n bits each.

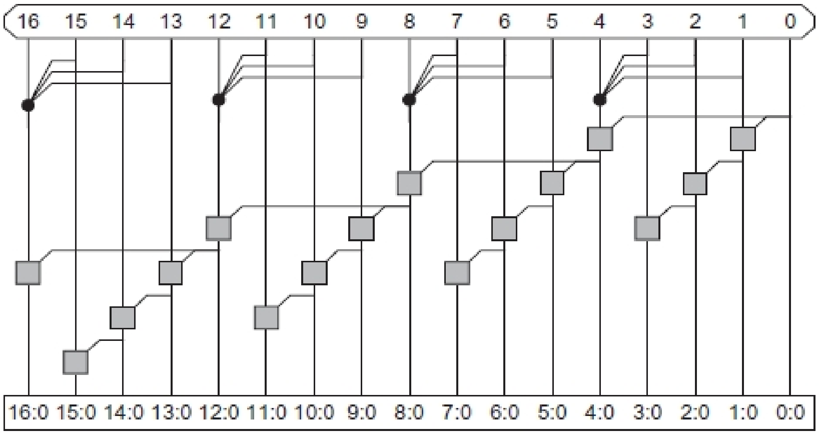

16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
16:0
15:0
14:0
13:0
12:0
11:0
10:0
9:0
8:0
7:0
6:0
5:0
4:0
3:0
2:0
1:0
0:0

**Figure 3.8: Linear carry look-ahead adder**

This will yield a delay of $t_{cla} = t_{pg} + t_{pg(n)} + [(n-1) + (k-1)]t_{AO} + t_{xor}$ (3.5)

Of course, this is *statistically* slower than the carry skip adder of a similar design.

The problem with the aforementioned designs is however, that non cmos logic is being utilized and a HDL cannot account for that unless custom cell libraries are created and the components recognized as such by the optimizer.

### 3.1.3 Carry Select Adder

Alternatively each stage, apart from the first one, can be calculated for both a zero and unit carry in and the result passed to the output when the last stage's carry out is ready. This is known a **linear carry-select or carry-increment** (if the common logic is factored out and the output multiplexer is simplified to a grey cell) **adder**. Its delay can be viewed either as

$$t_{select} = t_{pg} + [n + (k-2)]t_{AO} + t_{mux} \text{ or}$$

$$t_{increment} = t_{pg} + [(n-1) + (k-1)]t_{AO} + t_{xor} \qquad (3.6)$$

(a future note will also prove that 2-to-1 and xor delays can be used interchangeably). Or even as $t_{increment} = t_{pg} + t_{pg(n)} + [(k-1)]t_{AO} + t_{xor}$ if a faster than ripple-carry n-group PG logic is used (not necessarily of the variety introduced above as will be explained shortly).

### 3.1.4 Square-root Carry Select Adder

We will now notice that the carry chains for the more significant bits complete early. Therefore it would pay–off if instead of the **linear** designs we utilized short chains of variable group size so that the path of the carry through the previous stages and the computation of that group would be closer together. A notable choice would be for each group to include one bit more than the previous one. Then if the first group is of M bit and for a total of k groups, $W = M + (M+1) + (M+2) + \cdots + (M+k-1) = Mk + \frac{k(k-1)}{2} = {k^2}/{2} + k\left(M - {1}/{2}\right)$ or if $M \ll W$, $k \approx \sqrt{2W}$. Therefore,

$$t_{sqrt} \approx t_{pg} + t_{pg(M)} + \sqrt{2W}t_{AO} + t_{xor} \tag{3.7}$$

whence the name **square-root carry-select adder**.

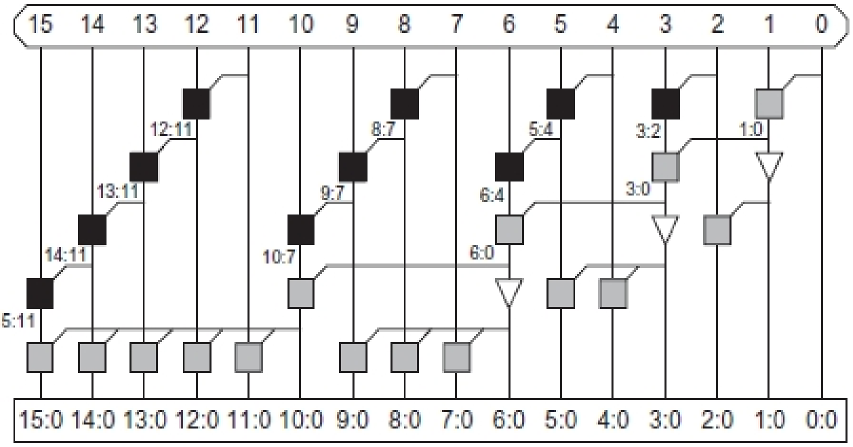


**Figure 3.9: Square-root carry-select adder**

In all of the above, the delay equations do not account for the fan-out that each stage must drive. Especially in the latter stages of the variable-group length adders the fan-out becomes large enough for buffering to be a requirement. Also when the subgroup adders become wide enough we can recursively apply the designs in the latter stages. By taking this to the limit, we obtain the **conditional-sum adder** that performs carry select starting with groups of 1 bit and recursively doubling to W/2 bits.

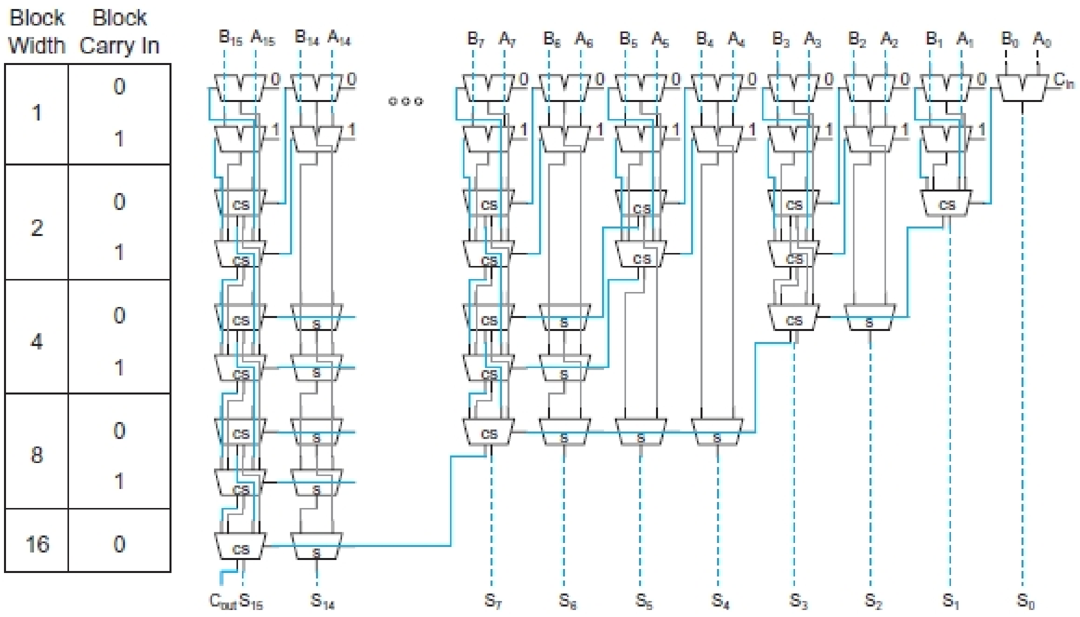

Block Width
Block Carry In
1
2
4
8
16
0
1
$B_{15}$ $A_{15}$ $B_{14}$ $A_{14}$
$B_7$ $A_7$ $B_6$ $A_6$ $B_5$ $A_5$ $B_4$ $A_4$ $B_3$ $A_3$ $B_2$ $A_2$ $B_1$ $A_1$ $B_0$ $A_0$
$C_{in}$
cs
s
$C_{out}$ $S_{15}$ $S_{14}$ $S_7$ $S_6$ $S_5$ $S_4$ $S_3$ $S_2$ $S_1$ $S_0$

**Figure 3.10: Conditional-sum adder**

Factoring out the common logic and using AO gates interchangeably (as will be shortly shown) with 2-to-1 multiplexers results in the Sklansky parallel prefix adder that will be discussed shortly.

### 3.1.5 Parallel Prefix Adders

The last but not least category of wide adders to be discussed is that of **tree, logarithmic, parallel-prefix or multilevel look-ahead adders**. The characterization will become apparent in just a moment.

For wide adders, the delay is dominated by the delay of passing the carry through the look-ahead stages. This can be reduced by looking ahead across the look-ahead blocks or as we've previously stated trade area for computational levels of the carry generator, by computing intermediate generate and propagate signals. And for wide enough adders even the O($\sqrt{W}$) delay of the square-root carry-select adder cannot match the O($\log_2 W$) of this family.

The many look-ahead trees present different tradeoffs amongst them with the **Brent-Kung tree** having a constant fan-out of 2 at each stage and minimal wiring but requiring $2\log_2 W - 1$ stages,

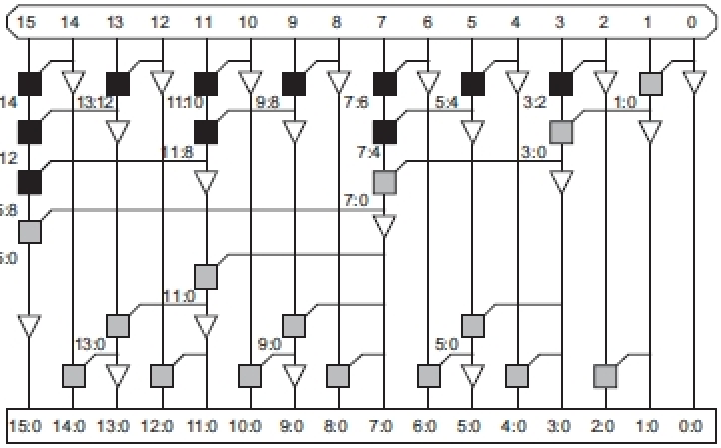

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14 13:12 11:10 9:8 7:6 5:4 3:2 1:0
15:12 11:8 7:4 3:0
15:8 7:0
15:0
11:0
13:0 9:0 5:0
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

**Figure 3.11: Brent-Kung tree**

the **Sklansky tree** with a delay of only $\log_2 W$ stages, but plagued by doubling fan-outs at each level, hence requiring either transistor sizing that will cut into the regularity of the layout or sufficient buffering of the critical signals,

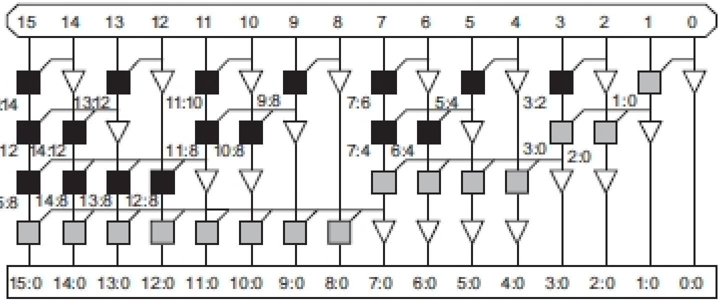

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14 13:12 11:10 9:8 7:6 5:4 3:2 1:0
15:12 14:12 11:8 10:8 7:4 6:4 3:0 2:0
15:8 14:8 13:8 12:8
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

**Figure 3.12: Sklansky tree**

and the **Kogge-Stone tree** with only $\log_2 W$ stages as well as a constant fan-out of 2, but a lot of long wiring. Also, in the order presented, the cell count is increased dramatically, thereby increasing the cost both in area and power consumption.

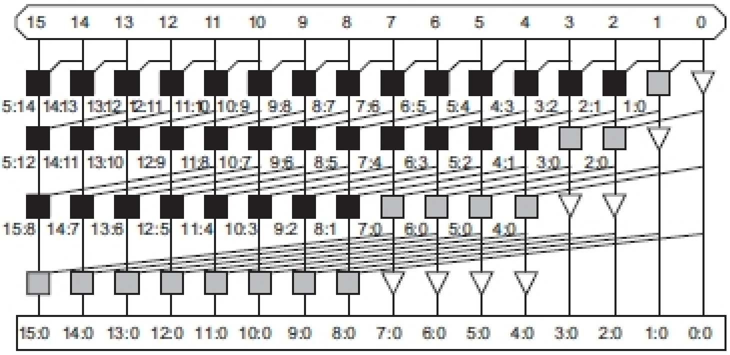

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14 14:13 13:12 12:11 11:10 10:9 9:8 8:7 7:6 6:5 5:4 4:3 3:2 2:1 1:0
15:12 14:11 13:10 12:9 11:8 10:7 9:6 8:5 7:4 6:3 5:2 4:1 3:0 2:0
15:8 14:7 13:6 12:5 11:4 10:3 9:2 8:1 7:0 6:0 5:0 4:0
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

**Figure 3.13: Kogge-Stone tree**

All three trees represent deviations from the ideal of $\log_2 W$ delay, constant fan-out of 2 and a single wiring track between each row in one respect. Between these extremes different compromises exist such as the **Han-Carlson**, **Knowles** and **Ladner-Fisher** trees. Higher valency tree adders can also be constructed utilizing for instance Manchester carry chains. These may have fewer stages but each stage is slower.

### 3.1.6 Hybrid Adders

Finally, as have hinted before **hybrid tree/select adders** can be constructed by using the tree to compute the carry into each stage, such as the **spanning tree adder** where the last level of the valency 3 Brent-Kung tree is saved by the carry select adder, or even utilize such a tree for each stage if the stages are long enough to warrant the hardware overhead. The group length should be balanced such that the carry-in and the pre-computed sums become available at the same time. This structure can be used for instance to battle the large fan-out of the Sklansky tree with the **sparse-tree adder** computing the carry-in of only every fourth bit.

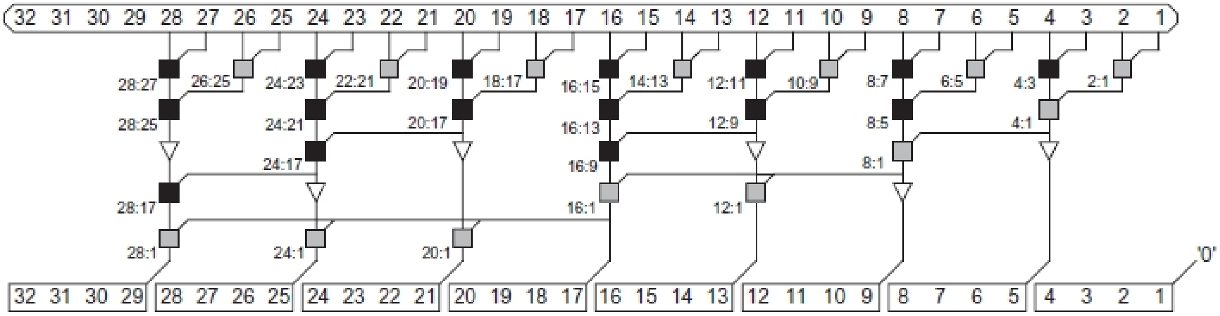

32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
28:27
26:25
24:23
22:21
20:19
18:17
16:15
14:13
12:11
10:9
8:7
6:5
4:3
2:1
28:25
24:21
20:17
16:13
12:9
8:5
4:1
24:17
16:9
8:1
28:17
16:1
12:1
28:1
24:1
20:1
'0'
32 31 30 29
28 27 26 25
24 23 22 21
20 19 18 17
16 15 14 13
12 11 10 9
8 7 6 5
4 3 2 1

**Figure 3.14: Sparse-tree adder**

Emphasis should be given to the ease with which the tree adder can be **pipelined** for a larger throughput rate. Something that might be of use when we discuss retiming at the end of this part.

---

### Multiplexer based adders. Mux AO and XOR.

If we were to notice that propagation and generation signals cannot both be true at once, the generation equation could be rewritten as $G_{i:j} = \overline{P_{i:k}} G_{i:k} + P_{i:k} G_{k-1:j}$. Clearly a 2-to-1 multiplexer unit. In the same vein a xor gate can be reworked into a multiplexer unit. The practical use of all that is of course provisional on a mux implementation, faster than the and-or or xor cmos implementations, respectively. A multiplexer based tree adder can then achieve a delay of $(\log_2 W + 2) T_{mux}$ (3.8) and just about half that with a singular pipelining element.

---

### 3.1.7 Carry Save Adder and Multiple Operand Addition

When three operands need to be added, an alternative strategy to cascading two adders can be employed. Noting that a disjoint FA can add 3 bits, if we provide as inputs bits of the same weight and save both carry and sum (**CSA**), we then need only add the two words (obviously the carry word needs to be shifted left by one space) with a singular adder to produce the same outcome. If we approximate the delay of the FA with the two cascading xor gates of the sum the delay could be about as low as

$$(\log_2 \mathrm{W} + 2 + 2)\mathrm{T}_{\mathrm{mux}} \tag{3.9}$$

,as we have already mentioned.

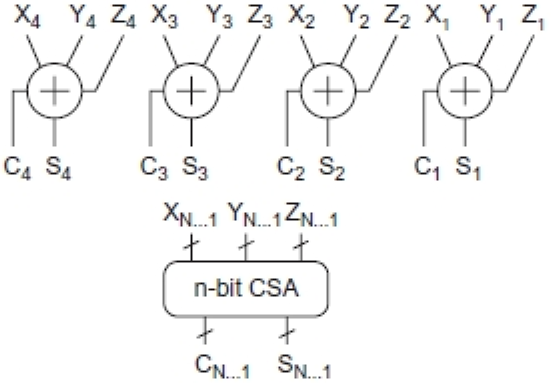

$X_4$ $Y_4$ $Z_4$ $X_3$ $Y_3$ $Z_3$ $X_2$ $Y_2$ $Z_2$ $X_1$ $Y_1$ $Z_1$
$C_4$ $S_4$ $C_3$ $S_3$ $C_2$ $S_2$ $C_1$ $S_1$
$X_{N...1}$ $Y_{N...1}$ $Z_{N...1}$
n-bit CSA
$C_{N...1}$ $S_{N...1}$

**Figure 3.15: Carry save adder**

## 3.2 Fast Multiplexer Implementations

To begin with, we cannot stress enough that for a relatively small fan-in, cmos components should not be composed of their logic constituents but should be developed as composite cmos gates. For instance if the 2-to-1 multiplexer were constructed as a 2 level and-or gate, the capacitive load for the previous stage would have been about 2/3 that of the static cmos mux, but the propagation delay of each stage, let alone the inverter that follows positive logic cmos gates, would vastly overshadow that small advantage.

Even faster multiplexers can, probably, be designed by transmission gate logic, tri-state gates and, as expected, inverted cmos multiplexers have close to half the delay of non-inverted ones, just by removing the cascading inverter. For the recursive design at hand, where the output is eventually fed back to the selector, we only need to change the enumeration of the inputs to its 1's complement for the multiplexer to make the correct selection. Unless it passes through an even number of inverted mux stages, in which case the output is the intended one.

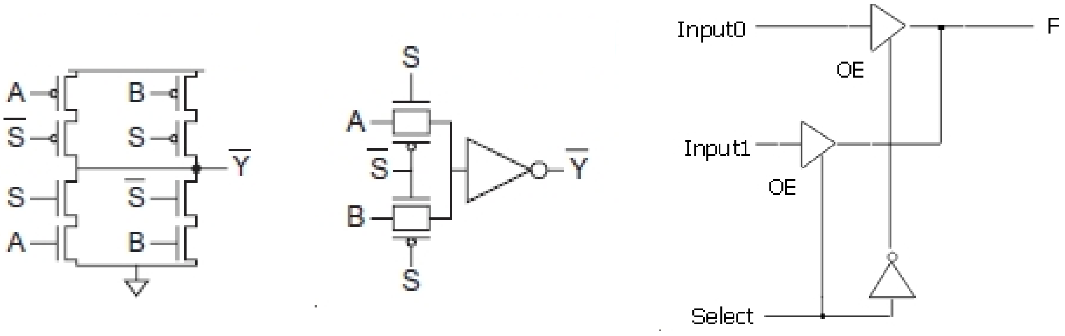

A
$\overline{S}$
S
A
B
S
$\overline{S}$
B
$\overline{Y}$
S
A
$\overline{S}$
B
S
$\overline{Y}$
Input0
OE
F
Input1
OE
Select

**Figure 3.16: In order, static cmos, transmission gate and tri-state multiplexers**

Yet, eventually, both the effort and parasitic delay of a large composite gate become large enough to make the multiple stages design a more appealing solution. In more detail, the linear delay model used for first order calculations defines the propagation delay of the gate as d=f+p, where f the effort or stage delay, that is the delay emerging from driving h identical gates or having fan-out h- f=gh, where g the logical effort representing the gate capacitance -, and p the parasitic delay emerging from the gate driving its internal capacitances. The delay is given in units of an inverter's, which can deliver the same output current, delay. And that of course means that the transistors should be sized accordingly and that the logical effort and parasitic delay represent the gate capacitance and the inherent gate load, respectively, in units of the inverter's gate (or load) capacitance.

It's worth noting that the over-encumbrance sometimes comes from the increased logical effort while others from the parasitic delay, or even both. For example the tri-state multiplexer has a constant logical effort of 2 while the parasitic delay grows linearly, as 2n, with the fan-in. On the other hand the cmos inverted mux's logical effort grows as ($\log_2 n+1$) and the parasitic delay even worse, as $n(\log_2 n+1)$.Therefore, the optimal number of inputs and stages is highly dependable on the design. For our purposes all multiplexers will be designed as a tree structure of 2-to-1 multiplexers. This will make depth and area calculations trivial and will also define a common unit time for the adders' delay. Besides, this is the logical choice for cmos multiplexers and for fan-out larger than one.

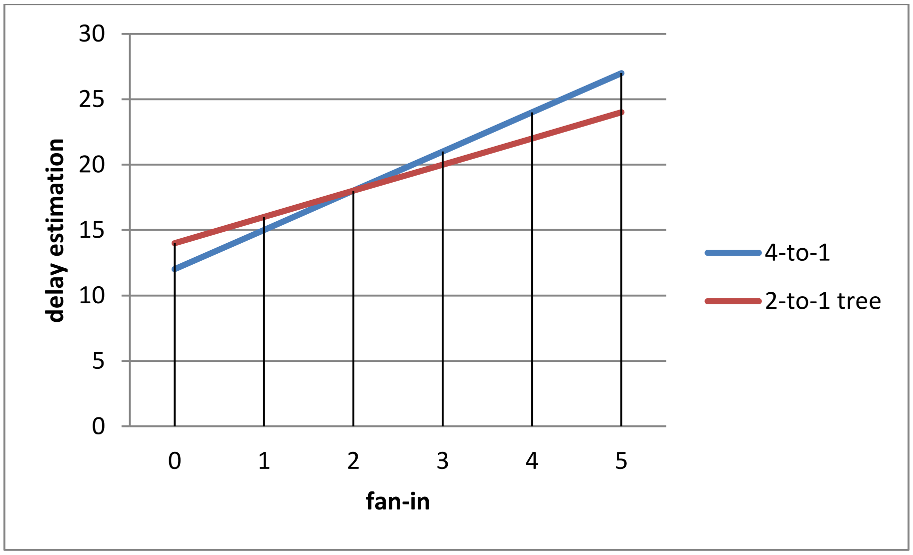


**Figure 3.17: Delay estimation/ fan-in of 4-to-1 cmos multiplexer and 4-to-1 tree of 2-to-1 cmos multiplexers**

An M-to-1 multiplexer tree, where $M=2^k$, composed of 2-to-1 multiplexer units would have one such unit in the final level, two in the one before and 4 in the one before that …up to $2^{k-1}$ in the first level, since pairs are grouped together. Hence, the depth of the unit would be $\log_2 M$ levels and the sum of the geometric series will yield (M-1) 2-to-1 multiplexer units. An $M^L$-to-1 multiplexer unit can now be seen either as L levels of M-to-1 units and summed in a similar manner, or we could directly apply the formulas we have just developed to yield a depth of $L\log_2 M$ and a total of $(M^L-1)$2-to-1 multiplexer units. Here, depth is equated to delay, since we decided to universally measure delay in units of 2-to-1 mux delay, $T_{mux}$.

## 3.3 Summary

In this chapter we talked about the pros and cons of various adder units, from the simple RCA to logarithmic adders, and we calculated their delay. We have also provided a similar exposition of multiplexer units and sufficient justification for the choice of a tree architecture. This has also allowed us to develop an area estimate as well as provide a universal time unit that will allow us to write all delay estimates, for both adder and multiplexer units, in a normalized form. The reader can review the basics of the methodology used in 3.2 as well as a more detailed account of adders in [3] by Weste and Harris. Some complementary notes can also be found in [4] by Rabaey et al.

# Iteration Bound 4

## DFGs, Iteration Period, Unfolding and Retiming

While for most circuitry the critical path is easy to determine, for a recursive algorithm, i.e. a block diagram with loops, the generalization of these techniques is not straightforward. Different well known techniques such as pipelining and parallel processing can be applied in order to improve upon the critical delay for a block diagram represented as a feed forward graph.

Furthermore, before we can commit to a particular realization of said algorithm and try to optimize it, we need to know its limits. And of course, be able to tell of its superiority amongst all other implementation or the trade-offs that make it an appealing solution. In this chapter we shall provide the minimum necessary training that will, in combination with chapter 3, answer questions of this sort for our application.

### 4.1 Data Flow Graphs

The data flow graph, or DFG for short, of a circuit is a directed graph whose nodes represent the various subunits (computations or functions) and the edges represent the intended flow of the signals. With each node we associate a normalized computation time and with each edge a non-negative number of delay elements. The DFG doesn't aim to capture the hardware architecture but rather the data flow among the various subtasks.

Each node fires when all the input data needed are available, therefore, only after its precedent nodes have produced an output. Hence, the edges describe a precedence constraint, which we shall call an intra-iteration precedence constraint whenever the edge has zero delays and inter-iteration precedence constraint otherwise. Of course many nodes can fire simultaneously and this concurrency is something that we may want to exploit.

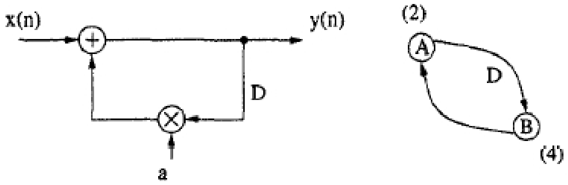


**Figure 4.1: (a) Block diagram and (b) its respective DFG**

Various DFGs describing one algorithm can be derived from one another through high level transformations.

## 4.2 Iteration Period Bound

The feedback loops in recursive algorithms impose an inherent bound on the achievable iteration period. This is referred to as the iteration period bound, or even iteration bound for short, and it is linked to that particular representation of the algorithm. That is, it is the best case scenario for the sample period, considering any and all transformations of that DFG (unfolding, retiming) without that untransformed DFG necessarily reaching it.

The IPB (iteration period bound) can be formally defined as the iteration bound of the critical loop of the DFG. The time needed to execute a loop is determined by the precedence constraints. That is to say, the precedence constraints tell us how many samples-not input samples but time delayed signals and specifically how many different time instances -are present in the loop at any given time and the nodes tell us the computation time of that loop. The loop can be transformed so that the sample period will be $T_i/D_i$ ,where $T_i$ the loop computation time and $D_i$ the number of delays in the loop. This is known as the loop bound and the critical loop is the one with the largest bound. Therefore, the iteration period bound is $T_\infty = \max_{\mathrm{ALL\ Loops}}\{\frac{T_i}{D_i}\}$. For large systems the longest path matrix algorithm or the minimum cycle mean algorithm can be used to compute the IPB.

## 4.3 Pipelining and Cut Set Retiming

The idea behind the retime approach is to move some delay elements to other edges of the data flow graph so that the transfer function is preserved. This could, like all akin approaches, accommodate faster clock speeds, reduced number of registers, reduced power consumption through the effect of switching frequencies on it or even logic synthesis. A formal algorithmic approach to retiming can be devised, however the version that lends itself the most to back of the envelope calculations is that of cut set retiming. This is a generalization of pipelining.

The pipelining transform adds a register on each edge of some cut set of a feed forward DFG (or a feed forward cut set, for more generality). This shortens the path between the neighboring registers and can therefore be used to shorten the critical path. Of course for each level of pipelining the latency, i.e. the time from input sample to the respective output, is increased by one. Pipelining can also be used to "cut across" units that are fired concurrently, assuming their architecture can be conveniently split in two time dependent parts and they belong to a common feed forward cut set. This is referred to as fine grain pipelining.

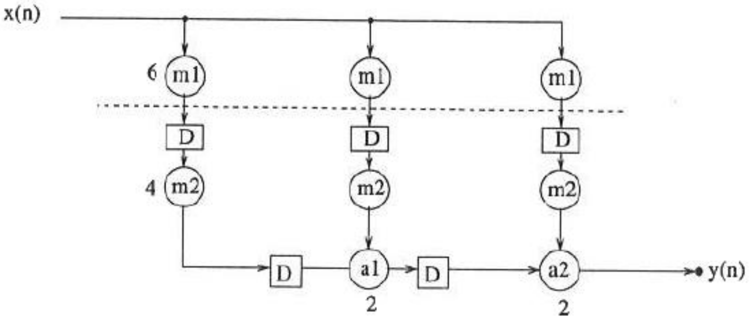


**Figure 4.2: Fine grain pipelining where the multipliers where conveniently split in two subunits**

For the cut set retime approach the steps to follow after producing a cut set is to simply remove a set number of delay elements from all the edges that belong to the cut set and flow into the one half of the graph and add the same number of delay element to the ones that flow into the other half.

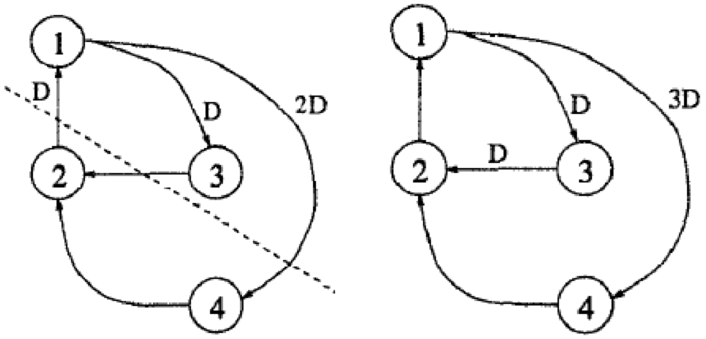


**Figure 4.3: An application of cut set retiming**

## 4.4 Unfolding

The unfolding transformation is the equivalent of parallel processing for recursive filters and as such is used for high-speed and low-power VLSI architectures. To apply the unfolding transformation we need only write the filter equations for J consecutive samples/iterations, for unfolding factor J, and factor the terms in the abscissas so that they correspond to output signals or retarded versions of them. Each delay element is now obviously J-slow. As is also the case for parallel processing, since in both cases they get a "fresh" sample every J$^{th}$ sample period.

**Example:**

$$y[n]=ay[n-9]+x[n] \qquad (4.1)$$

will be rewritten as $y[2k]=ay[2k-9]+x[2k]$ and $y[2k+1]=ay[2k-8]+x[2k+1]$ (4.2) for 2 unfolding factor, and they, in turn, will be factored as

$$y[2k]=ay[2(k-5)+1]+x[2k] \text{ and } y[2k+1]=ay[2(k-4)]+x[2k+1] \qquad (4.3)$$

The DFG transformation can be seen figure 4.4 below.

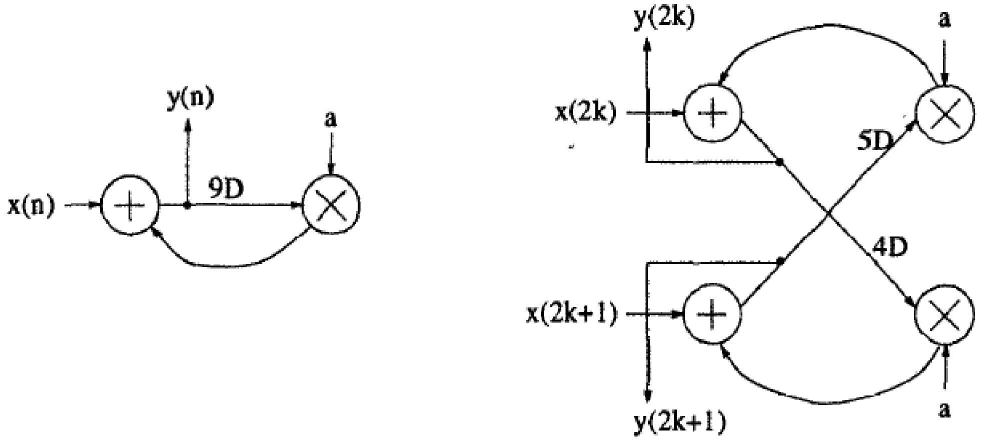


**Figure 4.4: The 2-Unfolding of the DFG in (a) into (b)**

The aforementioned method of tackling the problem will suffice for our needs, however, it is often tedious and automated algorithmic techniques have been developed. Of the formal theory, pertaining to sample period reduction, we could make use of theorems and corollaries referring to path transformations. From them we can infer that the original DFG can be retimed in a way such that the J-unfolded version of the retimed DFG will meet a specified achievable critical path bound. Or for low complexity circuitry we could retime the J-unfolded DFG of the original architecture to fit our needs. In general the following lemma can be derived.

**Lemma:** Any feasible clock cycle period that can be obtained by retiming the J-unfolded DFG, can be achieved by retiming the original DFG directly and then unfolding by unfolding factor J.

But before all that, the most important thing to understand is, when is unfolding necessary. In a lot of cases, as we have already mentioned, a DSP program cannot reach its iteration bound, even with retiming. The case might well be that the computation time of a node of the DFG exceeds the iteration bound. Therefore, the critical path cannot be retimed any shorter than that node. In that case $\left\lceil \frac{t_{node}}{T_\infty} \right\rceil$ unfolding factor should be used. It might also happen that the iteration bound is not an integer. The denominator of its irreducible form could then be used as the unfolding factor. If we encounter both at the same time then the minimum value of J is that for which $JT_\infty$ is an integer, greater or equal to the longest computation node. Creating a perfect rate DFG, i.e. a DFG with 1 delay in each loop will also achieve the iteration bound, but the unfolding factor required will usually be much larger than that computed by the previous rules.

The combination of unfolding and retiming is an advanced theoretical topic, usually excluded from most text books. By retiming the DFG, an acceptable performance can be achieved for much smaller unfolding factors and by following somewhat more advanced theoretical results, the critical path can be predicted and minimized before the unfolding transformation. This and the computation of the smaller unfolding factor required are beyond the scope of this thesis and a subject unto their own.

## 4.5 Summary

In this chapter we set the theoretical foundations for the transformations that we will apply on our circuit in the next part. We introduced the retime and unfolding transformations and we also tried to justify our expectations of their performance, by ways of the iteration period bound and by setting the rules by which an acceptable unfolding factor, that can achieve the iteration bound, can be derived. We purposely avoided referring to single and multiple rate DFGs as the concept of single rate DFG is the more intuitive of the two, and it is the only one we will have need of. For the interested reader [5] by Parhi can serve as a great introduction to the subject matter.

# PART II:

## IMPLEMENTATION AND ANALYSIS

# FBF Transformations 5

## Low Cost Techniques for High-Speed Implementation of Decision Feedback Equalizers

In modern digital communication systems, throughput rates in the order of gigabits per second are not uncommon. Hence, the need for faster implementations of equalizing filters. A well-known fact is that the feedback loop limits the achievable speed bound for decision feedback equalizers, since the feed-forward part can always be pipelined. And in sight of that particular problem the subject of this thesis will revolve around the reformulation of said FBF for increased clock and throughput rates. The complexity overhead of the implementation will also be a major concern.

### 5.1 Faster Adders and Multiplier Transformation

From the start we shouldn't fail to notice that the critical path of the FBF is composed of one multiplier, one slicer and two adders, as noted in bold line in the diagram below.

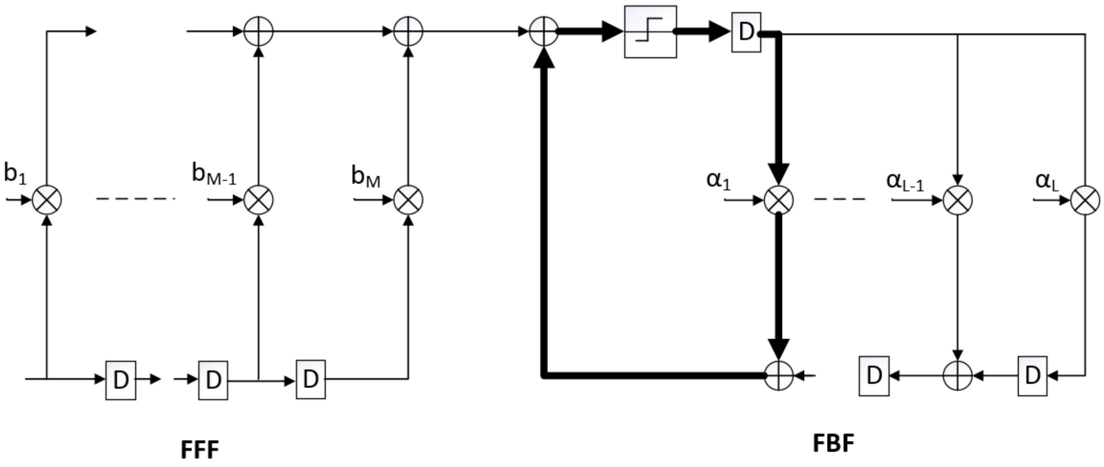


Figure 5.1: FBF's critical path

The critical path delay is also the iteration bound for this design. For recursive filters in general, a retime approach could shorten the critical path, but it wouldn't shorten the iteration bound. Also, more often than not, it cannot even achieve the iteration bound. An unfolding approach, as we'll discuss later, possibly in combination with retiming, could achieve the iteration bound, yet it too wouldn't improve upon it. As a result, the FBF has to be reworked into a new design with faster iteration bound before we attempt to reach it.

The first thing to notice is that even though the multiplier can easily be reworked, for an M-PAM system, as an M-to-1 multiplexer ($b_0a_j,b_1a_j,\dots,b_{M-1}a_j$) the problem of designing faster adder units remains. By introducing tree adders (see 3.1.5) the iteration bound even if we were to replace the multipliers with multiplexers would still be $T_\infty = (\log_2 W + 4) + \log_2 M$ u.t. (see 3.2), where u.t the 2 to 1 mux delay. And even though a better unit design has significantly reduced the critical path and iteration bound, the results are still lackluster.

## 5.2 Complete circuit overhaul

In this chapter we will redesign the filter in order to replace the multipliers with multiplexers. By observing the diagram we can derive the following equation:

$$\hat{y}[n] = y[n] - \sum_1^L a_1 \tilde{y}[n-k] \tag{5.1}$$

can be reformulated as $M^L$-to-1 multiplexer with the output selected from all of the possible sums

$$\begin{aligned}
&b_{M-1}a_1+b_{M-1}a_2+\dots+b_{M-1}a_L\\
&b_{M-1}a_1+b_{M-1}a_2+\dots+b_{M-2}a_L\\
&\dots\\
&b_{M-1}a_1+b_{M-1}a_2+\dots+b_0a_L\\
&\dots\\
&b_0a_1+b_{M-1}a_2+\dots+b_{M-1}a_L\\
&\dots\\
&b_0a_1+b_0a_2+\dots+b_0a_L
\end{aligned} \tag{5.2}$$

where $b_j$ the j-th M-PAM amplitude, with select vector $[\tilde{y}[n-k]]_{1xL}$, plus an adder. Or even $M^L$ adders, if we move them away from the critical path, each one preceding an input to the multiplexer.

The authors of [6] have claimed that according to [7],[8] it could also be reformulated as an$(\log_2 M)^L$ to 1 multiplexer, however, closer examination of the source material has deemed such claims to be untrue. The authors of [7],[8] have only formulated the transformation that we have devised.

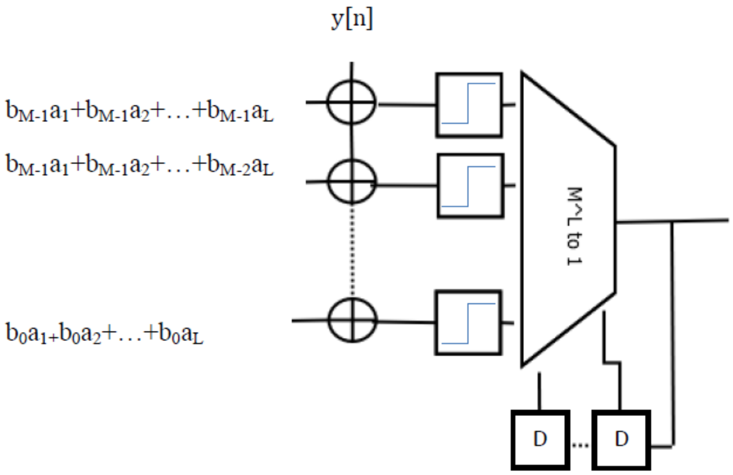

y[n]
$b_{M-1}a_1+b_{M-1}a_2+\ldots+b_{M-1}a_L$
$b_{M-1}a_1+b_{M-1}a_2+\ldots+b_{M-2}a_L$
$b_0a_1+b_0a_2+\ldots+b_0a_L$
M^L to 1
D
D

**Figure 5.2: FBF transformation to a single multiplexer**

As an added bonus ,we could probably, if we were willing to use 1's complement encoding, halve the size of the multiplexer since the amplitudes are symmetrically placed on either side of zero, hence the multitude of sums is actually half that plus their complements. It would, therefore, be much beneficial if only $M^L/2$ sums and their bitwise complements were passed through the multiplexer.

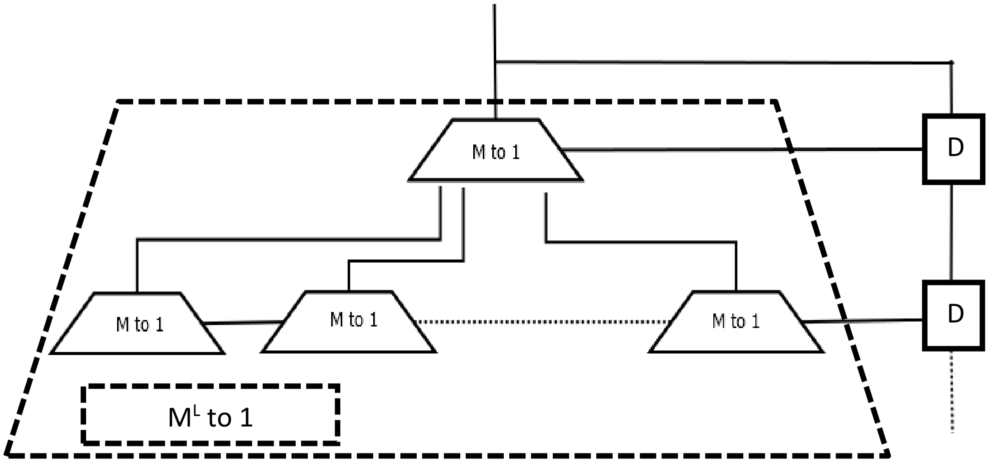


**Figure 5.3: The tree structure of the $M^L$-to-1 multiplexer. M multiplexers are the input to one multiplexer in the next stage**

However, the best multiplexer designs are probably not fast enough to provide enough margins for gigabit systems .A good case scenario for the iteration bound, since each of the L stages of the multiplexer is of depth $\log_2 M$, would be $T_\infty = \log_2 M \text{ u.t.}$ (see 3.2). We can achieve this iteration bound by retiming the multiplexer as can be seen for the special case of 2-Pam and L=2, below. That reformulation alone might provide enough margins in some technologies however the overhead would be extremely large- $M^L$ overhead for the adders alone- and as we have mentioned in the introduction, that will also be a concern of ours in this thesis. Furthermore it certainly would not be nearly as fast, if the increase in the delay caused by the ($M^L$ -1) fan-out of the output stage multiplexer and the wiring were taken into account. And as we will see in the next chapter its performance will also be limited by the transmission delay of the registers. Unfolding the circuit might let us sufficiently relax clock period, but then the overhead would be J times that of the previous transformation.

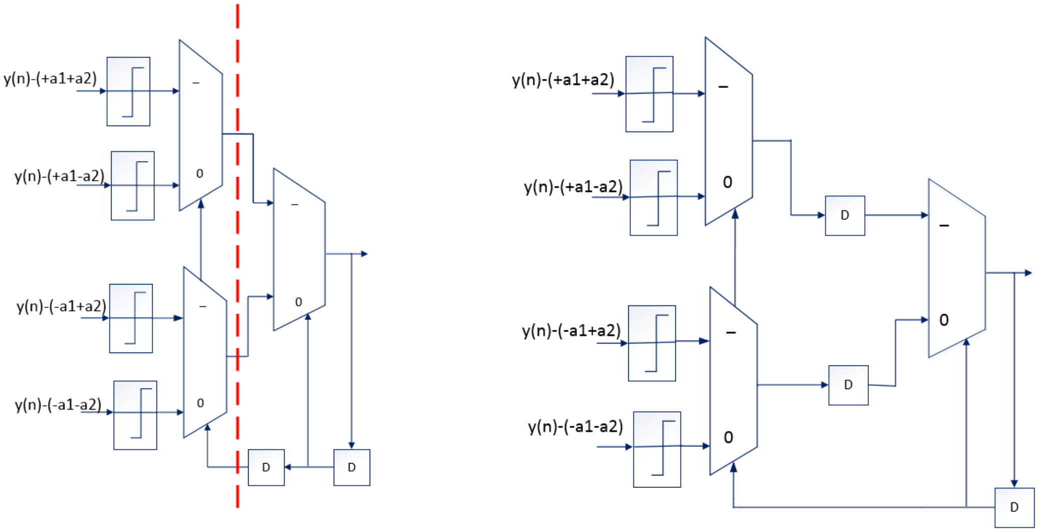


**Figure 5.4:Retiming of the binary PAM 2-Tap FBF**

By taking into consideration the exponential cost dependence on L an alternate design strategy will be introduced, to, at first, improve upon the hardware overhead but as it turns out, will also yield a low iteration bound.

---

## Functional decomposition

Transformations of this short are a special case (or in this case a slight generalization) of functional decomposition, which is derived from Shannon's expansion theorem. The theorem expresses the simple truth that a function with one or more variables with values in a countable finite set can be partially pre-computed for all the possible values of said variable and the true result can be decided when the variable's true value is available. A simple case for a Boolean type variable can be seen in the example below.

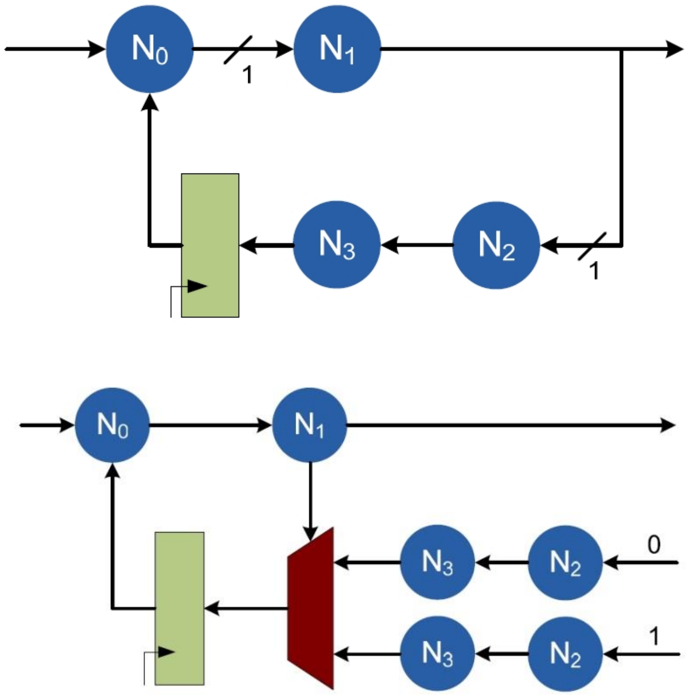


**Figure 5.5: Application of Shannon's decomposition theorem for iteration bound lowering.**

This technique is commonly used in order to improve upon the iteration bound, just as the example above or indeed our case. And will be reused time and time again in our thesis.

---

## 5.3 Partial Pre-Computation

The first step will be to only partially employ the pre-computation of all the possible sums strategy. A 2-term pre-computation for 2-Pam can be seen in figure 5.5. The critical path of this design is about $(\log_2 W + 4) + N\log_2 M$ u.t. (see Chapter 3), where N the number of pre-computation terms. The computation of the iteration bound seems a little more involved at first and very much dependent not only on the number of taps L and the number of pre-computation terms N but also on the word length and the multitude of the encoding levels. However, after some careful algebraic manipulation we can say that the iteration will always be bounded by the inner loop to

$$T_\infty = \log_2 M + \frac{2}{N+1}(\log_2 W + 3)\mathrm{u.t.} \qquad (5.3)$$

The N delay elements in the feedback loop can be used to retime the design, breaking up the critical path and for large enough N the performance will eventually be bounded by the adders in the rest of the FBF. An expected application of the retime approach for the 2-term pre-computation design would be to break the critical path within the adders and the multiplexer. The retiming cut sets can be viewed in the following schematic, too. The registers to be moved are crossed.

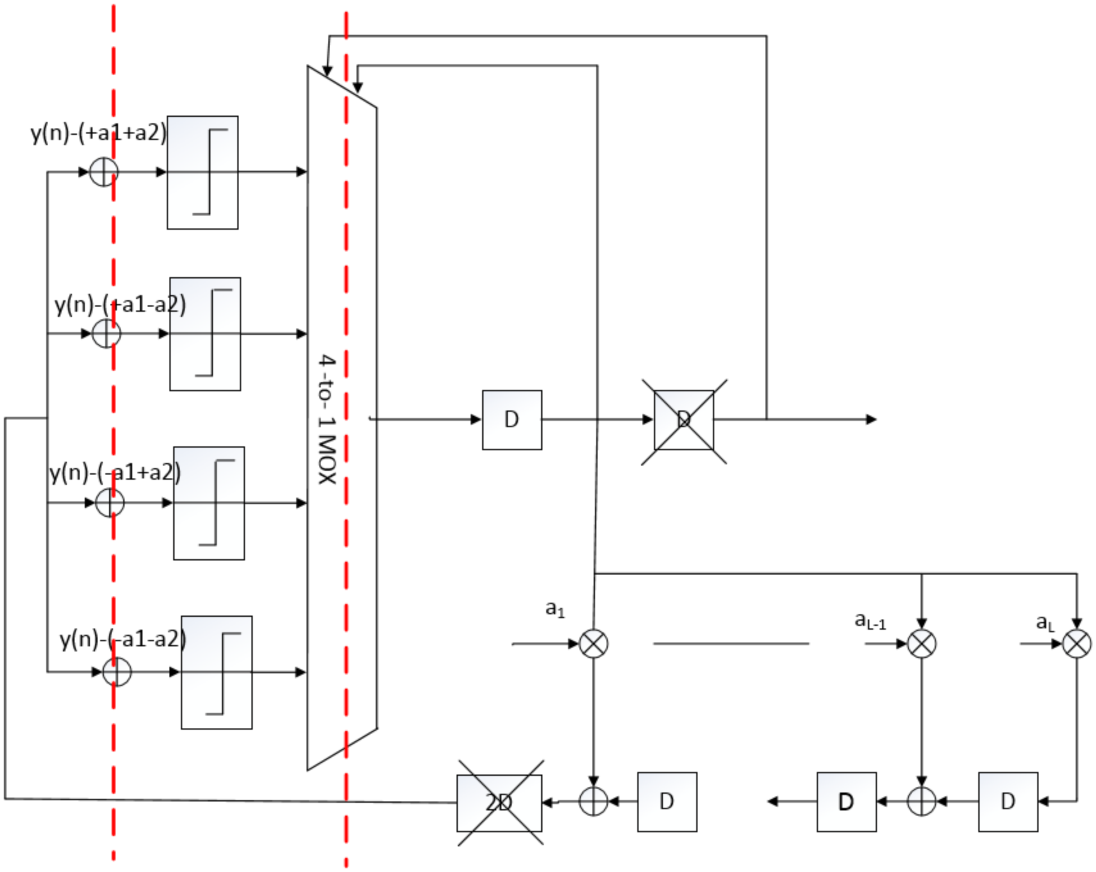


Figure 5.6: 2-term partial pre-computation and retiming

In general N, will now be the exponent in L's stead when it comes to hardware overhead, and a balance has to be struck between a smaller critical path and a large overhead. In the next section it will become obvious that N should be about L/2.

## 5.4 Two-Stage Pre-Computation

Naturally the obvious next step would be to, separately, pre-compute the remaining terms.

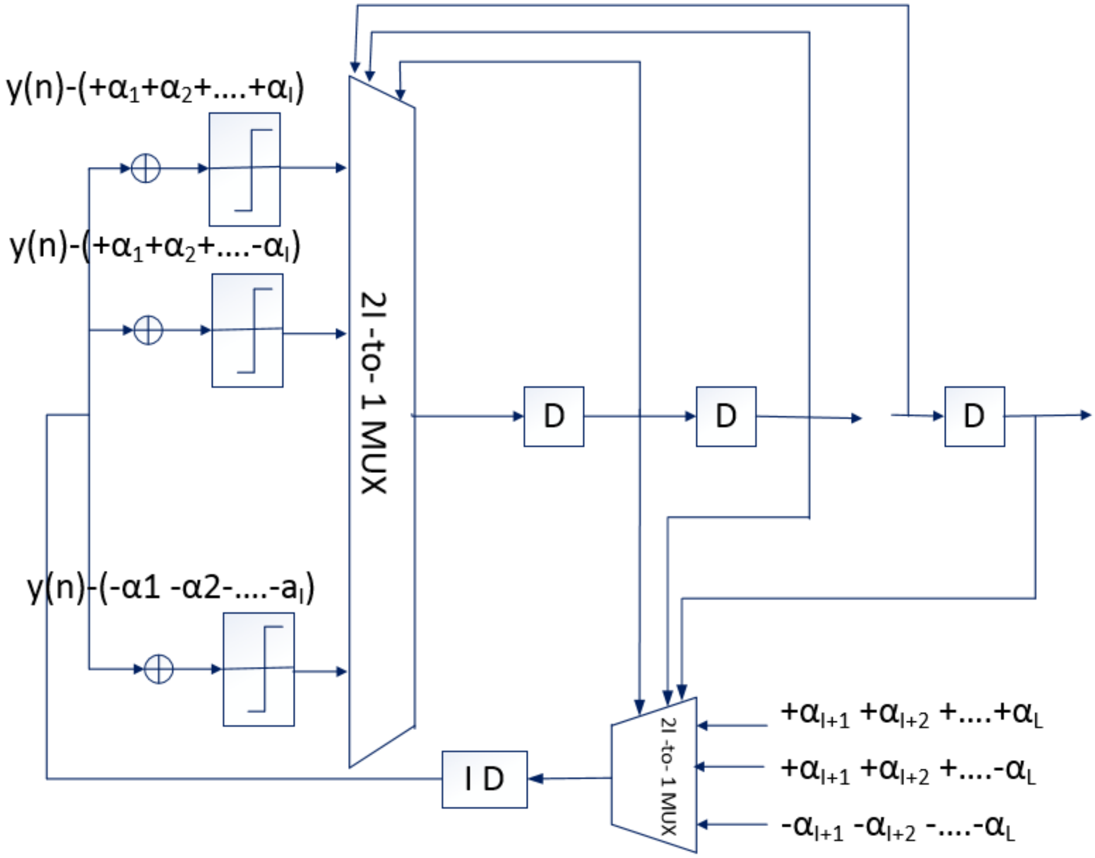


**Figure 5.7: Two stage pre-computation of the FBF for binary PAM**

Worthy of special note is the partition scheme. The different ways of partitioning the tap coefficients between the two groups lead to different iteration bounds. According to [6] the best grouping is to put the first terms in one group, and the rest in the other. This time a simple overview of this grouping will yield an iteration bound of

$$T_\infty = \log_2 M + \frac{1}{I+1}(\log_2 W + 4)\text{u.t.} \tag{5.3}$$

Also the complexity of the proposed architecture presents only about half the exponent of the original multiplexer design. Specifically, ($M^I$ -1) ,$\log_2 M$ width and ($M^{(L-I)}$ -1),W width 2-to-1 multiplexers and $M^I$ adders (CSA parallel prefix adders but the hardware overhead compared to the parallel-prefix adder on its own is insignificant) will be needed for the two-stage design. Whereas ($M^L$ -1),$\log_2 M$ width 2-to-1 multiplexers and $M^L$ adders would be needed for the original multiplexer design. The single multiplexer architecture can use slower, much smaller, adders, however, the path between the feed forward filter of the DFE and the FBF would become considerably slower. A cost function incorporating delay and hardware overhead should be invented and minimized in order to determine the best **I** per occasion. For our purpose **I** will be elected about L/2 in order to limit the complexity of the multiplexers and definitely not larger than L/2 to limit the overhead in adders while keeping the iteration bound fairly low, since the depth of the "feed forward" multiplexer impacts the iteration bound, while that of the "feedback" multiplexer does not.

Finally the extremely large fan-out of the original multiplexer design has also degraded exponentially, since the output of the "feedback" multiplexer is only carried to $M^I$ adders. To achieve the iteration bound of the last design we will next need to unroll (and possibly retime) the loop. The minimum unfolding factor is dependent both on the iteration bound and the unit design. That is the number of taps, the word length, the number of PAM levels, adder and multiplexer architecture etc. All of the above will be varied and experimented upon in the next chapter. An effort will also be made to experimentally approximate the iteration bound and to, if needed, retime the unfolded architecture in order to reach it.

## 5.5 Summary

In this chapter we reformulated the FBF in both a single and two stage pre-computation architectures. The area and iteration bound of both designs were defined. And their area-speed trade-offs were explained in detail. This has also led to a rule of thumb for the number of tap coefficients included in each stage of the two stage pre-computation architecture. However, all these are very ASIC centric and their validity for FPGA implementations will be discussed in the next chapter.

# FPGA Specific Implementations 6

## Approximations, Estimations and IPB achieving methodologies

A lot of the theoretical work that has been done thus far is inevitably tied to the specific unit design. That by itself is not specific to either an ASIC or for instance an FPGA specific design. It is not even specific to any VLSI technology. The fan-out and the capacitance of the wiring has not been taken into consideration and neither has the propagation delay of the registers themselves.

Yet, the optimal unfolding factor, for high speed and low area, is so intricately dependent on all possible factors, and its fractional form, that even our best estimates will be crude approximations which cannot possibly account for all real world phenomena. The most we can hope for is a small dispersion among all of the theoretical and experimentally approximated results, which will give us a small enough span to implement and compare.

### 6.1 Theoretical Estimation of the Unfolding Factor J

As we've already mentioned the iteration bound is very much dependent on the number of PAM levels, word length and the number of tap coefficients through the number of pre-computed constants (keeping in mind the $\mathrm{I} = \lfloor {}^{\mathrm{L}}/_{2} \rfloor$ rule). Following the rules stated towards the end of paragraph 4.2 the unfolding factor is also dependent on these, through the form of the iteration bound and the delay of the adders (the slowest node, regardless of the word length). The iteration bound we will tinker with is that of the last design $\mathrm{T}_{\infty} = \log_2\mathrm{M} + \frac{1}{\mathrm{I}+1}(\log_2\mathrm{W} + 4)\mathrm{u.t.}$

To simplify the design of the detector unit, a reasonable 2 level PAM will be used. With 1/-1 PAM levels it will be simplified to a sign check. This will also make its delay insignificant, just the way it was treated during the derivation of the iteration bound. The combined csa adder and data slicer unit will henceforth be noted as the vector merge unit.

Moreover, we will only experiment with even tap number multitudes since the added odd numbered coefficient, according to the $I = \lfloor L/2 \rfloor$ rule, will be inserted through the "feedback" multiplexer, whose delay contribution is expected to be a constant of $\log_2 M$ u.t.

The iteration bounds of the units to be composed are:

| $T_\infty$ | 4 bit | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|---|
| **6 tap** | 5/2 | 11/4 | 3 | 13/4 | 7/2 |
| **8 tap** | 11/5 | 12/5 | 13/5 | 14/5 | 3 |
| **10 tap** | 2 | 13/6 | 7/3 | 5/2 | 8/3 |
| **Slowest Node** | 6 u.t | 7 u.t | 8 u.t | 9 u.t | 10 u.t |

Table 6.1: Iteration bound for the ASIC Design

Hence, they should be unfolded according to:

| J | 4 bit | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|---|
| **6 tap** | 4 | 4 | 3 | 4 | 4 |
| **8 tap** | 5 | 5 | 5 | 5 | 4 |
| **10 tap** | 3 | 6 | 6 | 4 | 6 |

Table 6.2: Appropriate Unfolding factor for the ASIC Design

## 6.2 Constituent Units for the FPGA Design

However, all of the preceding designs have been built around ASIC methodology. The iteration bound is very much conditional on the use of fast parallel-prefix adders and the linear growth of the multiplexer's delay with **I**. Since Xilinx, which will translate the HDL code to an FPGA design, will be used for measurements, it would be wise to check how the proposed component architecture transfers over. The Virtex 5 family , XC5VLX20T module, set to speed -2, was simulated for the area and delay estimates.

### 6.2.1 Adders for FPGA Designs

To begin with, five different adder architectures will be pitted against the mapping the synthesizer produces for the HDL unsigned std_logic_vector addition. One bit's sign extension was used for the "integrated" design to account for the availability of $c_{out}$. The delay is the one that the synthesizer gives.

| **delay(ns)** | 4 bit | 8 bit | 16 bit | 32 bit | 64 bit | 128 bit | 256 bit |
|---|---|---|---|---|---|---|---|
| Ripple carry | 4.972 | 6.136 | 8.464 | 13.120 | 22.431 | 41.053 | 78.298 |
| Carry select-4 | ** | 6.104 | 8.526 | 13.372 | 23.062 | 42.444 | 81.207 |
| Carry select-8 | ** | ** | 7.525 | 9.983 | 14.899 | 24.732 | 44.398 |
| Square root | ** | 6.550 | 8.166 | 11.236 | 15.940 | 23.217 | ** |
| Sklansky | 4.972 | 6.136 | 8.061 | 11.332 | 16.001 | 19.897 | 23.719 |
| integrated | 4.616 | 4.460 | 4.642 | 5.006 | 5.734 | 7.190 | 10.102 |
| **area(LUTs)** | 4 bit | 8 bit | 16 bit | 32 bit | 64 bit | 128 bit | 256 bit |
| Ripple carry | 7 | 13 | 25 | 49 | 97 | 193 | 385 |
| Carry select-4 | ** | 16 | 32 | 64 | 128 | 256 | 512 |
| Carry select-8 | ** | ** | 38 | 76 | 152 | 304 | 608 |
| Square root | ** | 13 | 32 | 70 | 150 | 313 | ** |
| Sklansky | 7 | 13 | 28 | 70 | 176 | 416 | 908 |
| integrated | 6 | 9 | 17 | 33 | 65 | 129 | 257 |

Table 6.3: Delay and Area measurements of various Adder Implementations

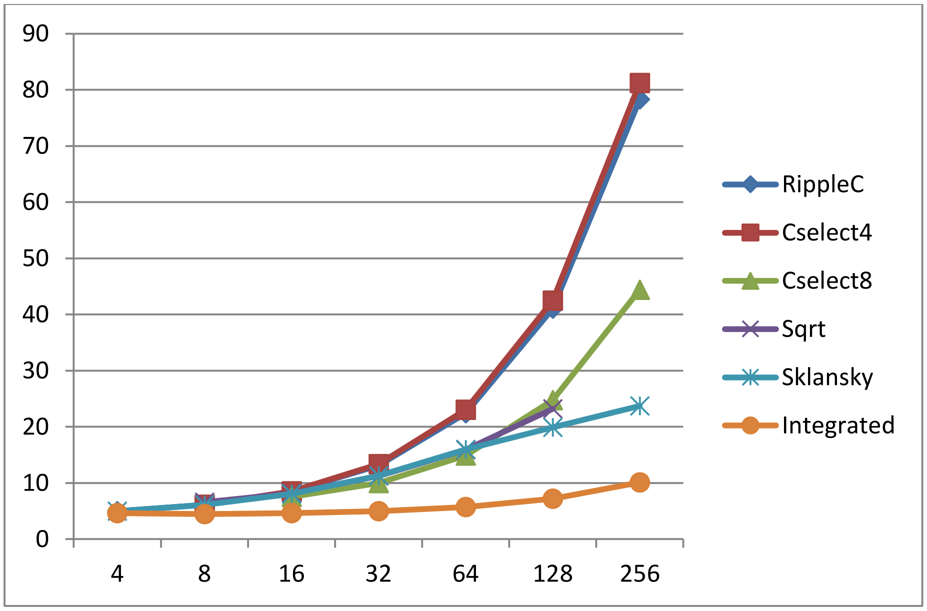


**Figure 6.1: Adder delay growth with word length: x-axis(bits) y-axis(ns)**

The results seem pretty disheartening at first sight. However, a closer examination of the logic delay alone will prove that these bard results are probably due to a routing problem. The synthesizer seems to use application specific software cores to develop a compact adder when one is clearly described as such, but has serious trouble routing the added wiring of more complex designs and certainly sufficiently buffer the signal to account for the ever increasing fan-out of the Sklansky carry gen.

| **Delay (NSec)** | 4 bit | 8 bit | 16 bit | 32 bit | 64 bit | 128 bit | 256 bit |
|---|---|---|---|---|---|---|---|
| Sklansky | 3.096 | 3.268 | 3.526 | 4.128 | 4.730 | 5.246 | 5.676 |
| integrated | 3.010 | 3.688 | 3.870 | 4.234 | 4.962 | 6.418 | 9.330 |

Table 6.4: Logic Delay of Sklansky's tree adder vs. the synthesizer's adder mapping

In reality the tree adder outperforms the "integrated" adder in every step of the way. In fact its performance is closer to that of the ripple carry adder. The assumption that it actually is a ripple carry adder is reinforced by its hardware cost (one LUT per bit) and confirmed by the critical path report of the synthesizer.

The results will further deteriorate for a carry save adder. It is our estimate that this is due to routing issues, but it is beyond the scope of this thesis to further explore this. Probably a CSA/RCA is constructed. The hardware of the 3 Op. integrated adder still doubles as expected, yet it remains lower than the tree adder design’s overhead.

| **Delay (Nsec)** | 4 bit | 8 bit | 16 bit | 32 bit | 64 bit | 128 bit | 256 bit |
|---|---|---|---|---|---|---|---|
| CarrySave/Sklansky | 5.198 | 6.279 | 8.606 | 13.381 | 17.989 | 23.198 | 27.172 |
| 3 Op. integrated | 5.579 | 4.838 | 5.020 | 5.384 | 6.112 | 7.568 | 10.480 |

Table 6.5: Logic Delay of CSA/Sklansky's tree adder vs. the synthesizer's adder mapping for 3 operand addition

For the aforementioned reasons the adder solution that is automatically produced by the Xilinx tool (when writing behavioral code) will be used and its contribution to the iteration bound will be approximated by (W-1)+4 u.t. (see equation 3.4). A drawback of this choice is of course that an elementary solution for pipelining the component, if the need arises, is not readily available (input and output and intermediate carry wires will be included in the cut set), and an effective solution for FPGAs is based on a deeper understanding of FPGA specific design, according to [10], and is beyond the scope of this thesis. If pipelining is needed towards the end, the simple RCA pipelining scheme will be used.

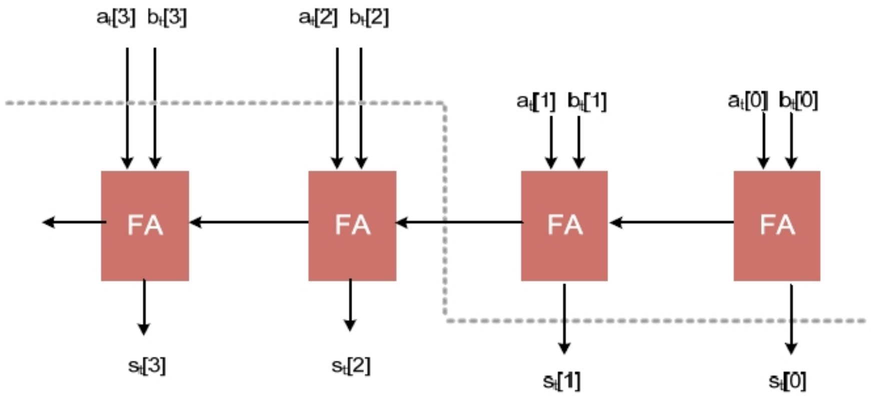


**Figure 6.2: Pipelining cut set for RCAs**

### 6.2.2 Multiplexers for FPGA Designs

One thing we noticed when we tried to implement AO and XOR gates with multiplexers is the inability of the synthesizer to successfully map 2-to-1 multiplexers. The suspected result can become explicit by viewing the technology schematic of a simple 2-to-1 multiplexer, wherefrom we notice that it is implemented by means of and/or gates. Meanwhile, in the schematic of the 8-to-1we can notice the existence of a muxf7 component which is a dedicated 2-to-1 multiplexer component. However, as it turns out this is by design. The muxf7 unit is more than twice as slow as a LUT AO gate or even a 4-to-1 LUT mux, which is the largest LUT constructed mux and has exactly the same delay as the AO gate. And it is only used as the output stage of larger multiplexers, which we suspect means it has better driving capabilities. The synthesizer can justify the trade-off between parasitic delay and driving capabilities when a large fan-out is expected. But all of this is only speculation on our part.

Therefore, multiplexers will not be implemented by explicitly coding mux trees but by a simple **case…is** structure, which the synthesizer recognizes as a N-to-1 multiplexer. This will give us fewer stages, since 4-to-1 LUT multiplexers of exactly the same speed will be incorporated. Moreover, Xilinx manuals tell of faster implementations for larger multiplexers by taking advantage of the structure of the FPGA, and we actually stumbled upon a slightly faster implementation for a 16-to-1 multiplexer, but such designs, once more, are probably specific to the FPGA used and require an understanding of FPGA specific design that is beyond the scope of this thesis.

| **Mux x-to-1** | 2-to-1 | 4-to-1 | 8-to-1 | 16-to-1 | 32-to-1 |
|---|---|---|---|---|---|
| **Delay(ns)** | 3.823 | 4.124 | 4.328 | 4.910 | 5.261 |

Table 6.6: Multiplexer delay

The mux delay results are severely skewed and masked by the IO buffer delays but the delay increase is sufficiently linearly depended on the length of the select vector to warrant the use of the $\log_2 M$ u.t. or even I $\log_2 M$ u.t. delay, as before. However since the output stage of 8-to-1 and larger multiplexers, which is the case with our design, dominates the delay, while previous stages are of smaller than expected depth and delay, using a full (L-I)$\log_2 M$ u.t. delay for the "feedback" multiplexer might be closer to the truth. The experimental approximation of the iteration bound will shortly prove this to be quite true.

## 6.3 Estimation of the Unfolding Factor J for FPGAs

After all these, the iteration bound and minimum unfolding factor will be computed for both the $T_\infty = \log_2 M + \frac{1}{I+1}\left((W-1)+4\right)\text{u.t.}$ (6.1)

| $T_\infty$ | **4 bit** | **8 bit** | **16 bit** | **32 bit** | **64 bit** |
|---|---|---|---|---|---|
| **6 tap** | 11/4 | 15/4 | 23/4 | 39/4 | 71/4 |
| **8 tap** | 13/5 | 16/5 | 24/5 | 8 | 72/5 |
| **10 tap** | 14/6 | 17/6 | 25/6 | 41/6 | 73/6 |
| **Slowest Node** | 7 u.t | 11 u.t | 19 u.t | 35 u.t | 67 u.t |

Table 6.7: Iteration bound for the (6.1) approximation

| **J** | **4 bit** | **8 bit** | **16 bit** | **32 bit** | **64 bit** |
|---|---|---|---|---|---|
| **6 tap** | 4 | 4 | 4 | 4 | 4 |
| **8 tap** | 5 | 5 | 5 | 5 | 5 |
| **10 tap** | 6 | 6 | 6 | 6 | 6 |

Table 6.8: Appropriate Unfolding factor for the (6.1) approximation

and the $T_\infty = \frac{1}{I+1}(L\log_2 M + (W-1)+4)\text{u.t.}$ (6.2) approximations.

| $T_\infty$ | 4 bit | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|---|
| **6 tap** | 13/4 | 17/4 | 25/4 | 41/4 | 73/4 |
| **8 tap** | 3 | 19/5 | 27/5 | 43/5 | 15 |
| **10 tap** | 17/6 | 21/6 | 29/6 | 45/6 | 77/6 |
| **Slowest Node** | 7 u.t | 11 u.t | 19 u.t | 35 u.t | 67 u.t |

Table 6.9: Iteration bound for the (6.2) approximation

| J | 4 bit | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|---|
| **6 tap** | 4 | 4 | 4 | 4 | 4 |
| **8 tap** | 3 | 5 | 5 | 5 | 5 |
| **10 tap** | 6 | 6 | 6 | 6 | 6 |

Table 6.10: Appropriate Unfolding factor for the (6.2) approximation

In any case, all our theoretical approximations do not yield vastly different results and all unfolding factors from 3 to 6 will be explored.

The multiplexers' select signals can be extrapolated from the tables in appendix A. These express the unfolded equation y's indices and their factored form. The J=2 example can be seen in the table below.

| UF 2 | | 2k indices | | 2k+1 indices | |
|---|---|---|---|---|---|
| 1 | $a_1$ | 2k-1 | 2(k-1)+1 | 2k | 2k |
| 2 | $a_2$ | 2k-2 | 2(k-1) | 2k-1 | 2(k-1)+1 |
| 3 | $a_3$ | 2k-3 | 2(k-2)+1 | 2k-2 | 2(k-1) |
| 4 | $a_4$ | 2k-4 | 2(k-2) | 2k-3 | 2(k-2)+1 |
| 5 | $a_5$ | 2k-5 | 2(k-3)+1 | 2k-4 | 2(k-2) |
| 6 | $a_6$ | 2k-6 | 2(k-3) | 2k-5 | 2(k-3)+1 |
| 7 | $a_7$ | 2k-7 | 2(k-4)+1 | 2k-6 | 2(k-3) |
| 8 | $a_8$ | 2k-8 | 2(k-4) | 2k-7 | 2(k-4)+1 |
| 9 | $a_9$ | 2k-9 | 2(k-5)+1 | 2k-8 | 2(k-4) |
| 10 | $a_{10}$ | 2k-10 | 2(k-5) | 2k-9 | 2(k-5)+1 |

Table 6.11: Factored form for the indices of y, for unfolding factor J=2

## 6.4 Manual Retiming Approach

Finally we will attempt to retime the circuit for the purpose of shortening its critical path. This won't be done by retiming the DFG, but by hand in order to allow for fine grain pipelining of the assorted units. And even though a few ways of pre-dividing the units into smaller parts in order to use an automated retiming algorithm  for the modified DFG come to mind, trade-offs between them and ways of accounting for experimental data would be the subject of much research. In this quest of ours [6] will be used as a guide. For this purpose we will guess and dissect their procedure of reaching the retimed design, study it step by step and then reevaluate its success for an FPGA design.

The first thing to notice is that the retiming methodology to follow is, naturally, very much affected by the unfolding factor and the number of tap coefficients, but also, and to a lesser extent, by the word length. Their design is for an 8bit 4unfolded FBF with 6Tap coefficients. And as much of our initial research, it utilizes CSA/parallel-prefix adders and a mux tree structure.

The critical path follows the dotted line and it has a delay of $(7_{(VM)}+8)$mux delays. The **first step** in retiming the design is to move delays $D_o$ through $D_3$ to all of the branches of their respective edges. This will make clearer the availability of delay elements around each unit.

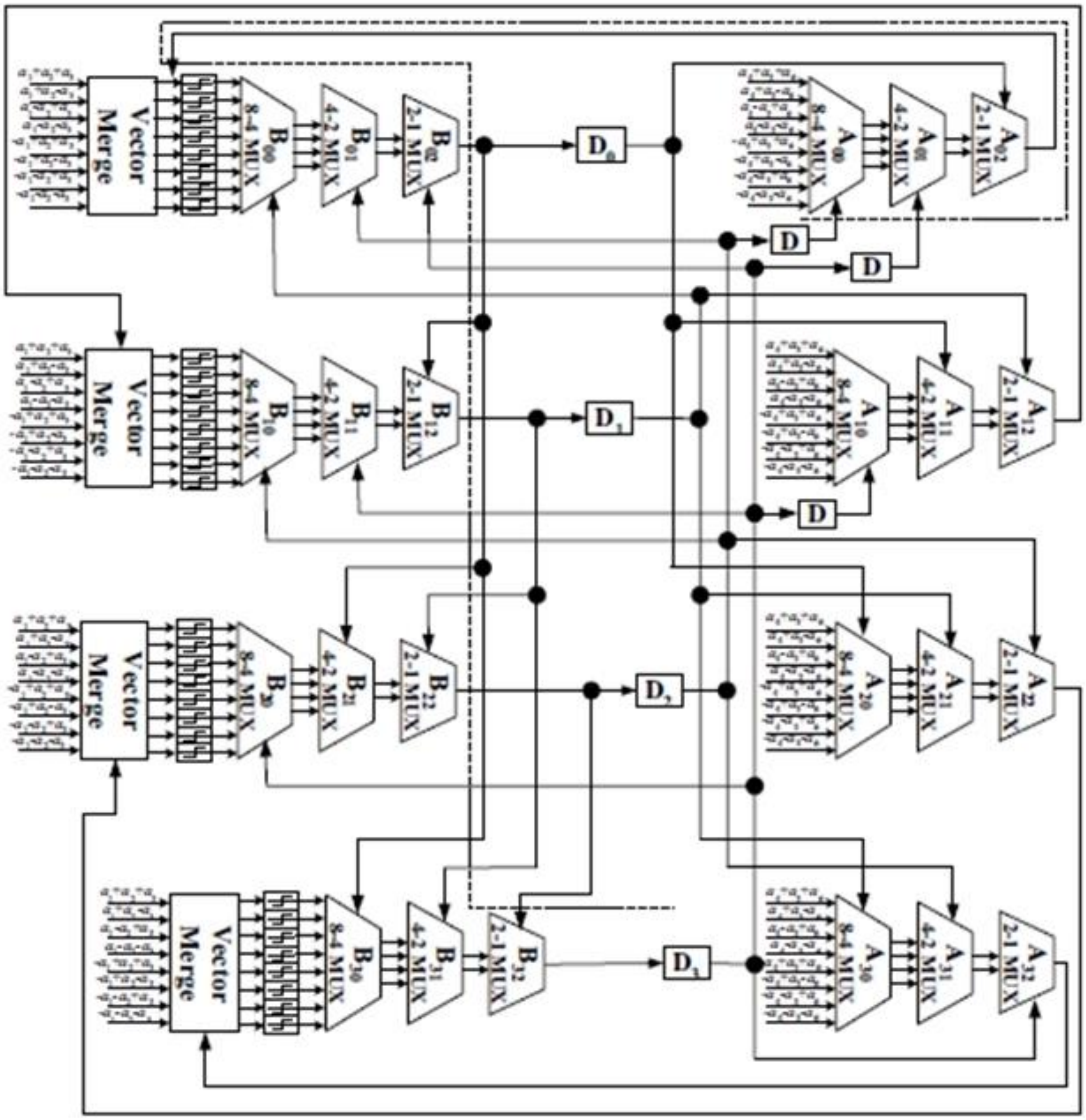

Vector Merge
B00 8-4 MUX
B01 4-2 MUX
B02 2-1 MUX
D0
A00 8-4 MUX
A01 4-2 MUX
A02 2-1 MUX
D
D
Vector Merge
B10 8-4 MUX
B11 4-2 MUX
B12 2-1 MUX
D1
A10 8-4 MUX
A11 4-2 MUX
A12 2-1 MUX
D
Vector Merge
B20 8-4 MUX
B21 4-2 MUX
B22 2-1 MUX
D2
A20 8-4 MUX
A21 4-2 MUX
A22 2-1 MUX
Vector Merge
B30 8-4 MUX
B31 4-2 MUX
B32 2-1 MUX
D3
A30 8-4 MUX
A31 4-2 MUX
A32 2-1 MUX

**Figure 6.3: 8bit 4Unfolded FBF with 6Tap coefficients**

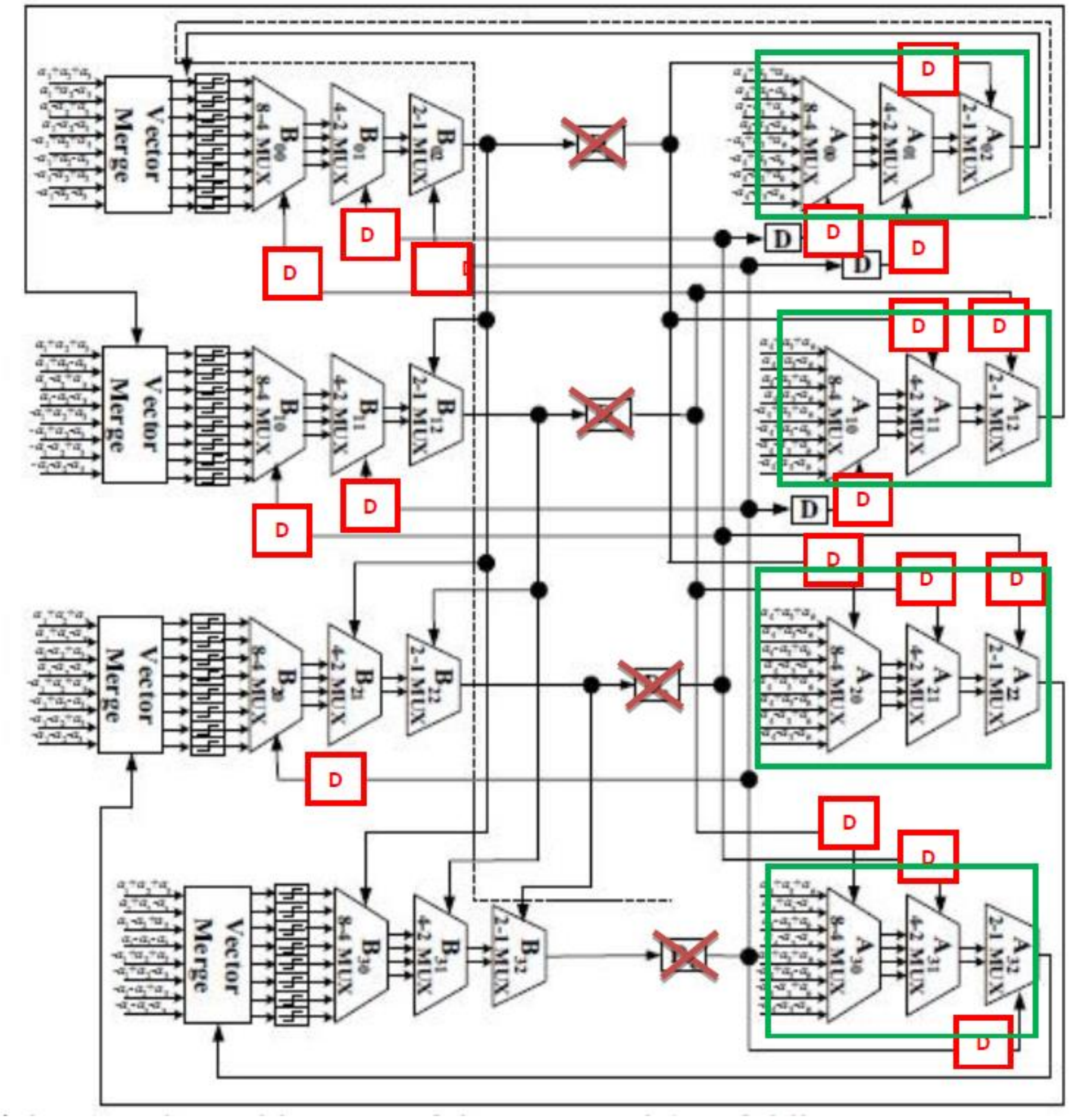

Vector Merge
B00 8-4 MUX
B01 4-2 MUX
B02 2-1 MUX
B10 8-4 MUX
B11 4-2 MUX
B12 2-1 MUX
B20 8-4 MUX
B21 4-2 MUX
B22 2-1 MUX
B30 8-4 MUX
B31 4-2 MUX
B32 2-1 MUX
A00 8-4 MUX
A01 4-2 MUX
A02 2-1 MUX
A10 8-4 MUX
A11 4-2 MUX
A12 2-1 MUX
A20 8-4 MUX
A21 4-2 MUX
A22 2-1 MUX
A30 8-4 MUX
A31 4-2 MUX
A32 2-1 MUX
D

**Figure 6.4: Step1. Move the available registers to all of their branches and Step2. retime the**

The **second step** will be to retime all of the "feedback" multiplexers by moving the delays from their select inputs to their outputs. This step will move the topmost delay along the critical path. However, a new critical path of only slightly shorter length has now been created.

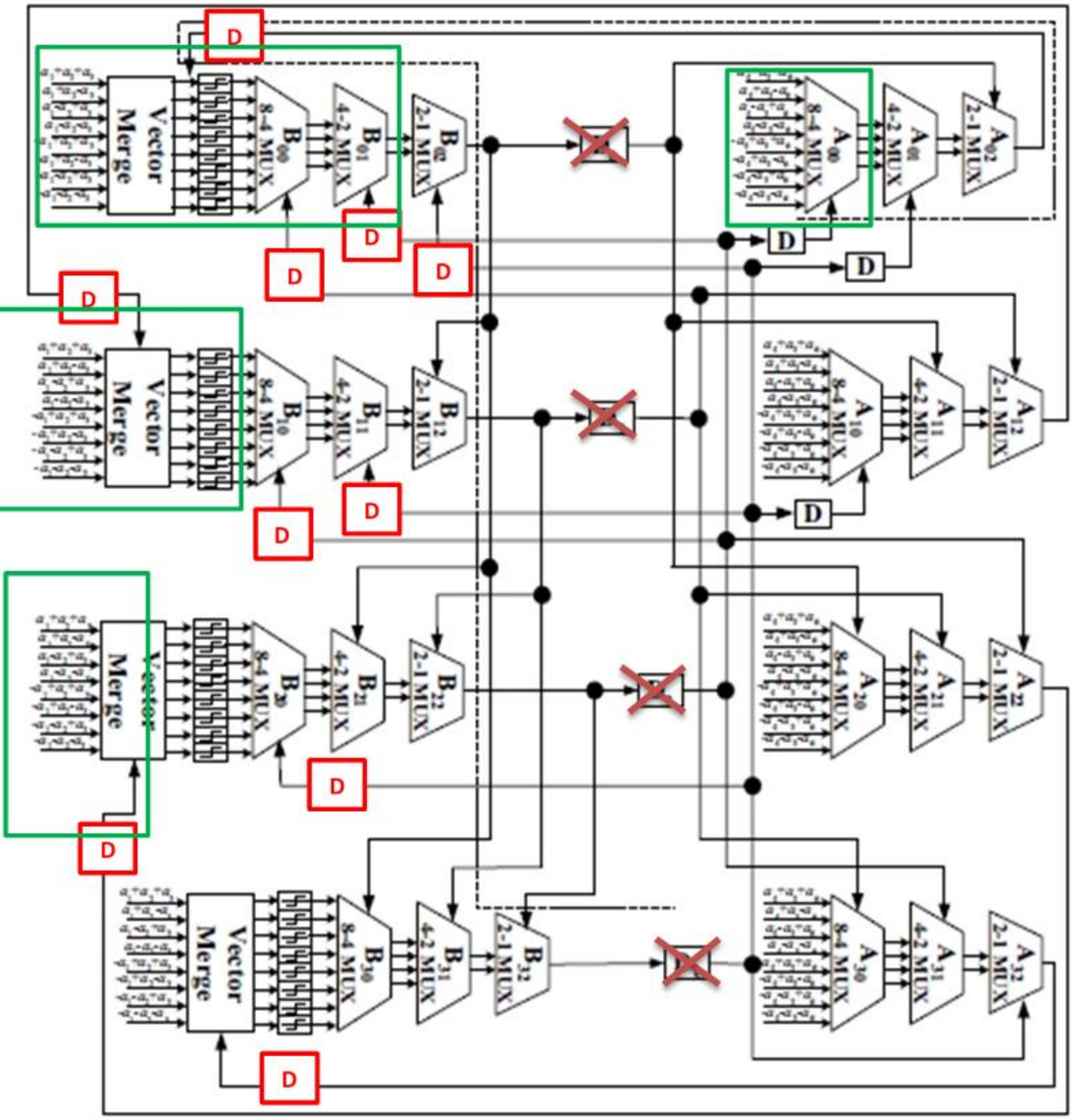

D
Vector Merge
B00 8-4 MUX
B01 4-2 MUX
B02 2-1 MUX
A00 8-4 MUX
A01 4-2 MUX
A02 2-1 MUX
B10 8-4 MUX
B11 4-2 MUX
B12 2-1 MUX
A10 8-4 MUX
A11 4-2 MUX
A12 2-1 MUX
B20 8-4 MUX
B21 4-2 MUX
B22 2-1 MUX
A20 8-4 MUX
A21 4-2 MUX
A22 2-1 MUX
B30 8-4 MUX
B31 4-2 MUX
B32 2-1 MUX
A30 8-4 MUX
A31 4-2 MUX
A32 2-1 MUX

**Figure 6.5: Step3. The ad-hoc chosen cut sets**

**Finally**, retiming will be applied to the cut-sets that can be viewed in figure 6.6. These were carefully chosen to limit the critical path to ($7_{(VM)}$+4) mux delays as can be seen by following the dotted line in the design below. Specifically the third row vector merge was pipelined in order to truncate the path followed by the red line in figure 6.7, to just under the length of the critical path. A couple of minor errors were detected and corrected in the design but the validity of [6]'s results was largely unaffected. Moreover, since the design is ASIC centered a few more improvements were made (inverted multiplexers- see previous chapter 3- and driving/buffer elements were used).

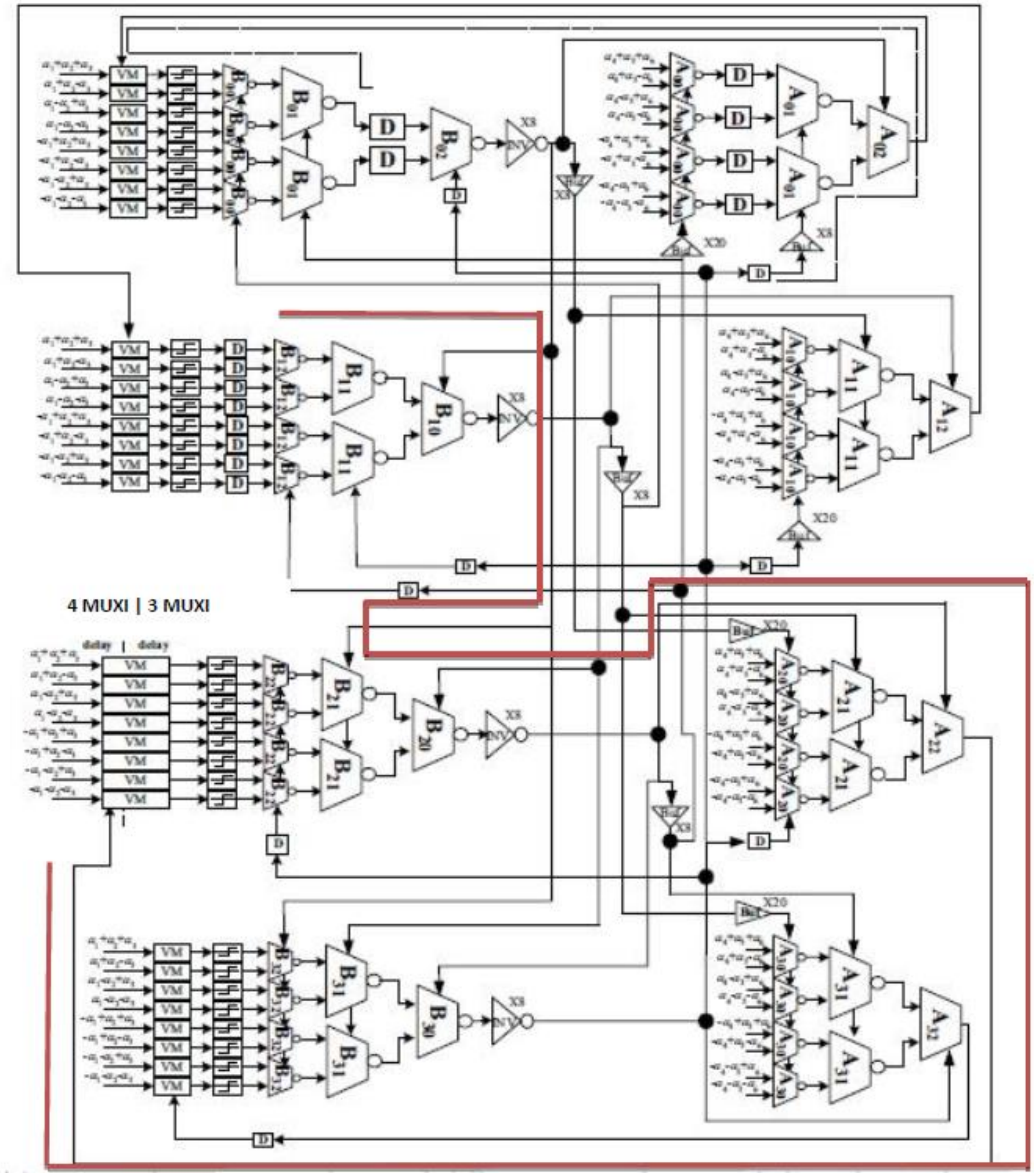

VM
D
B01
B02
INV
X8
A01
A02
X20
B11
B10
A11
A12
4 MUXI | 3 MUXI
delay | delay
B21
B20
A21
A22
B31
B30
A31
A32
Buf

Figure 6.6: Retimed ASIC design for 8bit 4Unfolded FBF with 6Tap coefficients

For an FPGA centric design, if we were to combine the derived knowledge of this thesis up to this point, the initial critical path would have a delay of about 3, 8-to-1 multiplexers plus that of a CSA/RCA. Pipelining of the CSA/RCA mapping that the synthesizer has produced wouldn't be straightforward at all, since we need access to the CSA carry signals in order to produce an appropriate cut set. We would be forced to retime the third row cut set just like the second row cut set. Besides, the critical path would be the one following the red line with 3, 8-to-1 multiplexers plus the VM delay. Exactly that of the unfolded design on its own.

Yet, if we assume that a more careful unit redesign for the multiplexer would allow us to count the delays as an 8-to-1, a 4-to-1 and a 2-to-1 multiplexer an improvement of about .5 to .7 (if we didn't retime the 3rd row cut set at all) ns can be expected. And while this is not insignificant this is not promising either. The aforementioned circuit will be tested and its improvement (or lack thereof) quantified in the next chapter.

However, absence of significant improvement is not proof of unattainability. It only means that the retiming scheme we have devised have failed. A rigorous algorithmic approach could, possibly, vastly improve upon the sample frequency. But as we have already mentioned, developing the specifics for the application of the method would be a big undertaking unto its own.

## 6.5 Summary

In this chapter our unit design decisions were reevaluated to better fit an FPGA centric design. This also gave rise to two new estimates for the iteration bound. The latter was experimentally approximated as well. Based on these and in accordance to the rule of thumb 4.4 the unfolding factored required was estimated between 4 and 6, depending mainly on the number of tap coefficients L.

# Simulation Results 7

## Area/Speed Metrics and Validation

At the final stage of our thesis the constructed architectures will be simulated by the Xilinx Ise synthesizer for a Virtex 5 FPGA. Faster FPGAs are available (Virtex 6 for instance), however we find that the results are less informative. Route delay becomes even more prevalent and there is a large dispersion between custom architectures and automated ones. Both facts would hinder speed comparisons. Of course, both area and frequency comparisons will be made. We will also, briefly at least, talk about our test benching methodology.

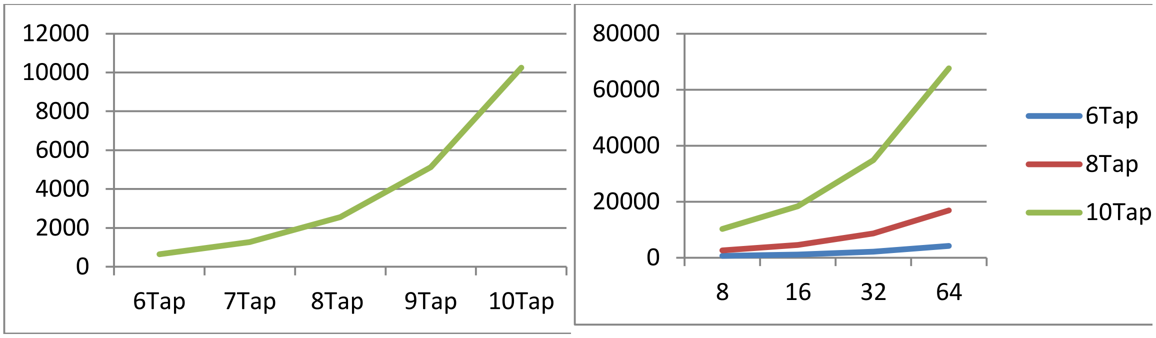
### 7.1 Retimed Multiplexer Architecture

Firstly, we will test the complete circuit to mux transformation. As we have already shown in chapter 5 the circuit can easily be retimed to reach its iteration bound. And this, of course, is the version that we will use.

| **Area(LUTs)** | **8 bit** | **16 bit** | **32 bit** | **64 bit** |
|---|---:|---:|---:|---:|
| **MuxFBF6Tap** | 639 | 1151 | 2175 | 4223 |
| **MuxFBF7Tap** | 1279 | 2303 | 4351 | 8447 |
| **MuxFBF8Tap** | 2559 | 4607 | 8703 | 16895 |
| **MuxFBF9Tap** | 5119 | 8210 | 17407 | 33791 |
| **MuxFBF10Tap** | 10239 | 18431 | 34815 | 67583 |

Table 7.1: Area measurements for the Multiplexer FBF Design

As expected, since both the size of the multiplexer and the number of RCAs double, the size seems to double with L, therefore follow the respective exponential law. Also it seems to grow linearly with the word length, also as expected, since the size of the RCAs grows linearly with it. In figure 7.1 (b) the respective growth may seem exponential, but we should notice that the x-axis is logarithmic.



(a) Exponential area growth with the number of tap coefficients

(b) Growth with the word length

**Figure7.1: Area growth laws for the Multiplexer FBF design**

The maximum clock frequency, in accordance with the theoretical results, is constant for all word lengths, since the part they affect has been shifted away from the critical path, and number of tab coefficients, since the $2^L$-to-1 multiplexer has been pipelined with all paths equal to any of its L stages. The maximum frequency is 924.556MHz and as has been foretold the design is bottlenecked by the delay of the register (82% of the logic delay). And even though the iteration bound has clearly been reached we can still unroll the loop, expecting the critical path to pass through J 2-to-1 multiplexers. The register delay, however, will still contribute only once, therefore limiting its effect to only one $J^{th}$ . Keeping in mind the already large hardware overhead, we will delay discussion of that possibility for a later paragraph.

## 7.2 Two Stage Pre-Computation FBF

The 2 stage pre-computation FBF is not very interesting by itself, but the speed and area analysis of this lesser component will significantly decrease the complexity of its unfolded version's analysis.

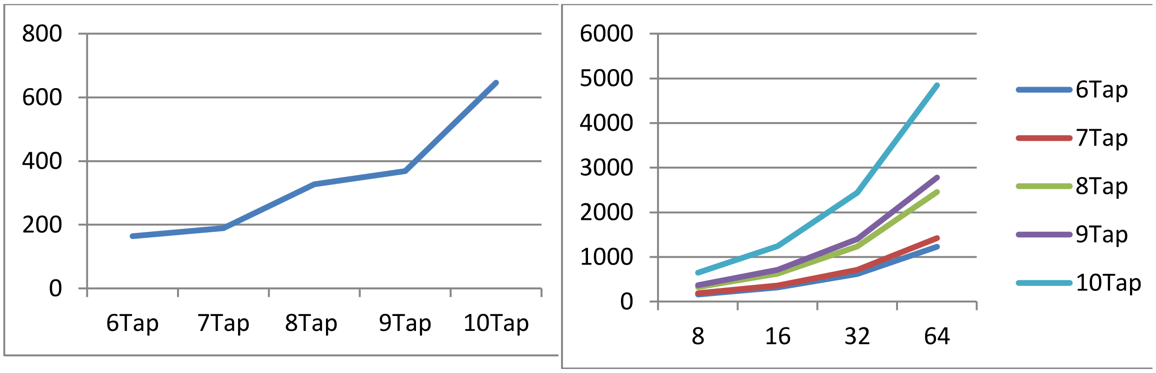
| **Area(LUTs)** | **8 bit** | **16 bit** | **32 bit** | **64 bit** |
|---|---|---|---|---|
| **FBF6Tap** | 164 | 316 | 620 | 1228 |
| **FBF7Tap** | 189 | 365 | 717 | 1421 |
| **FBF8Tap** | 327 | 631 | 1239 | 2456 |
| **FBF9Tap** | 369 | 713 | 1401 | 2777 |
| **FBF10Tap** | 646 | 1246 | 2446 | 4846 |

Table 7.2: Area measurements of the 2 stage pre-computation FBF

| MaxFreq(MHz) | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **FBF6Tap** | 360.308 | 338.135 | 301.078 | 246.950 |
| **FBF7Tap** | 360.308 | 338.135 | 301.078 | 246.950 |
| **FBF8Tap** | 301.566 | 285.876 | 258.932 | 217.864 |
| **FBF9Tap** | 301.566 | 285.876 | 258.932 | 217.864 |
| **FBF10Tap** | 282.679 | 268.847 | 244.883 | 207.832 |

Table 7.3: max Frequency estimation for the 2 stage pre-computation FBF

From the area measurements we can infer two major things. One, since the area grows significantly only on even multitudes of the number of tap coefficients- that is, when the "feed forward" part grows - the area is dominated by the adders. The "feed forward" multiplexer (width 1) doesn't contribute much either , since the contribution of the "feedback" multiplexer (width W>>1) is insignificant as well. And two, the area once more grows linearly with the word length. This can also be directly extrapolated from the dominance of the adders' area, but is also corroborated by the linear grow in the dominant "feedback" multiplexer.



(a) Significant area growth only on even multitudes of the number of tap coefficients

(b) Linear growth with the word length

**Figure7.2: Area growth laws for the 2 stage FBF design**

With regards to the frequency, the difference in the critical path's delay doubles when the difference in word lengths doubles, therefore revealing the linear dependence on the word length, through the adders. But the small drop per bit in frequency also reveals the, once more, significant contribution of the register delay (here of the CSA as well). This can also be seen in the synthesizer's critical path report. Also, the frequency, naturally, only drops when the "feed forward" part is affected, since only that enters in the critical path. And the severe drop in frequency is telling of the impact of growth of the "feed forward" multiplexer. But not so much the increased fan-out, since the drop is very much anomalously dependent on the I, just as the construction of multiplexer units by the synthesizer is (remember, for instance, that 2-to-1 and 4-to-1 multiplexers are both mapped to a single LUT, which affords them about the same delay, while for an 8-to-1 multiplexer the output driving element dominates the delay).

## 7.3 Unfolded Circuits, J= 2 through 7

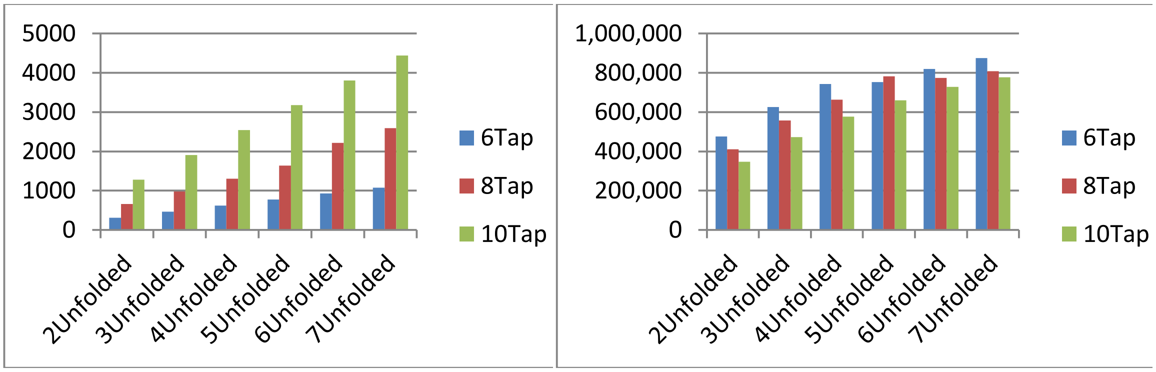
For the 2 stage pre-computation architecture, unfolded versions with unfolding factors J=2 through 7 were developed. The theoretical results called for unfolding factors 4,5 and 6, however, we considered measuring a wider range would be the wiser choice. All of the measurements can be found in appendix B. Only the bare essentials will be displayed here due to the overwhelming volume.

| Area(LUTs) | 6Tap | 8Tap | 10Tap |
|---|---|---|---|
| 2Unfolded | 312 | 656 | 1276 |
| 3Unfolded | 465 | 980 | 1909 |
| 4Unfolded | 618 | 1306 | 2542 |
| 5Unfolded | 771 | 1634 | 3175 |
| 6Unfolded | 925 | 2213 | 3808 |
| 7Unfolded | 1079 | 2589 | 4442 |

Table 7.4: Area measurements for unfolding factors 2 through 7 and 8 bit words

| MasterClk(MHz) | 6Tap | 8Tap | 10Tap |
|---|---|---|---|
| 2Unfolded | 475.324 | 411.050 | 347.164 |
| 3Unfolded | 625.920 | 558.150 | 472.719 |
| 4Unfolded | 742.444 | 663.720 | 576.292 |
| 5Unfolded | 752.655 | 782.715 | 660.245 |
| 6Unfolded | 819.918 | 774.804 | 728.244 |
| 7Unfolded | 875.567 | 808.269 | 777.672 |

Table 7.5: max Frequency estimation for unfolding factors 2 through 7 and 8 bit words

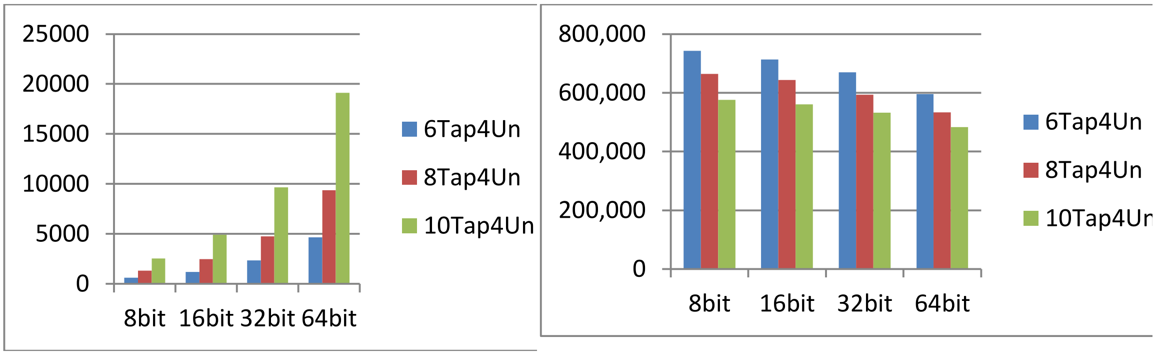


(a) Area growth proportionate to J (b) Speed-up with J approaching saturation

**Figure 7.3: Area and frequency measurements for unfolding factors 2 through 7 and W=8**

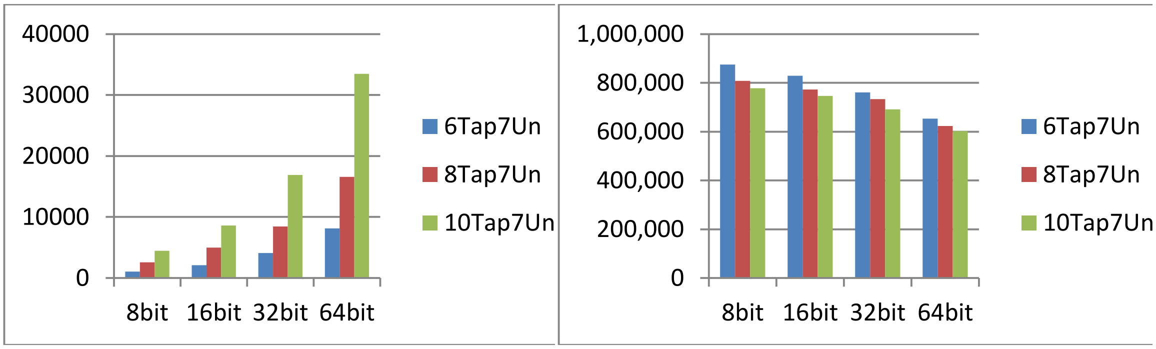
Keeping in mind the observations in section 7.2 the area seems to grow proportionally with J, exactly as expected. The frequency however contrary to the theoretical expectations steadily grows beyond the expected unfolding factors independently of the number of tap coefficients L. The growth however seems to be reaching a certain saturation point, with every hit in hardware overhead yielding diminishing returns. Unfolding factors in excess of 7 can be explored however as will be discussed in the next section that would be of limited use. Furthermore, the effect the register delay has already been reduced to 1/7 of its potency and is therefore, in comparison with the architecture in 7.1, already inconsequential.

We can also notice a slight anomaly in the maximum achievable frequency of the 5 unfolded FBF. However, since none of the designs is proven optimal, a slightly shorter or longer critical path here and there, within the margin of error, is nothing that some elementary retiming cannot invert. Moreover contrary to the results in tables 6.8, 6.10 and 6.13, with the rise in L, even though theoretically higher achievable rates are allowed, they don't conclusively seem to be achievable in quite as small a rise in J. Unless we disregard the "tainted" results of J=5. Then the expected performance is more than likely.



(a) Area growth with W (b) Drop in frequency with W

**Figure 7.4: Effects of the word length W on the area and frequency for J=4**



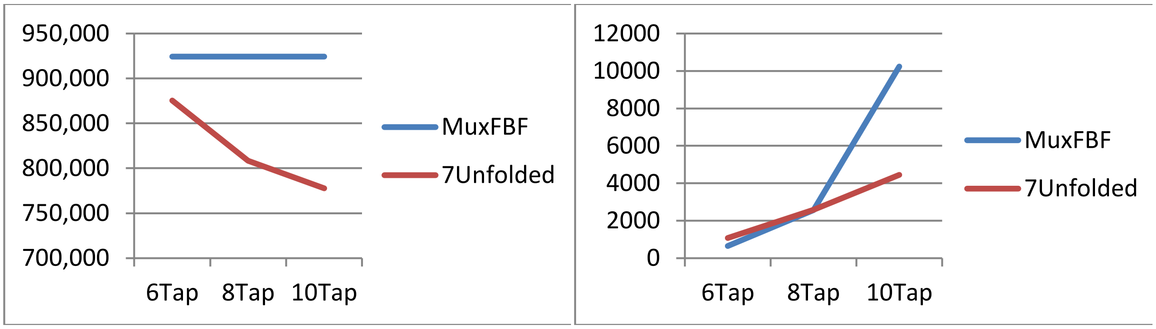
(a) Area growth with W (b) Drop in frequency with W

**Figure 7.5: Effects of the word length W on the area and frequency for J=7**

The changes in area and frequency with W seem to follow the laws extrapolated from the 2 stage FBF in 7.2, as expected.

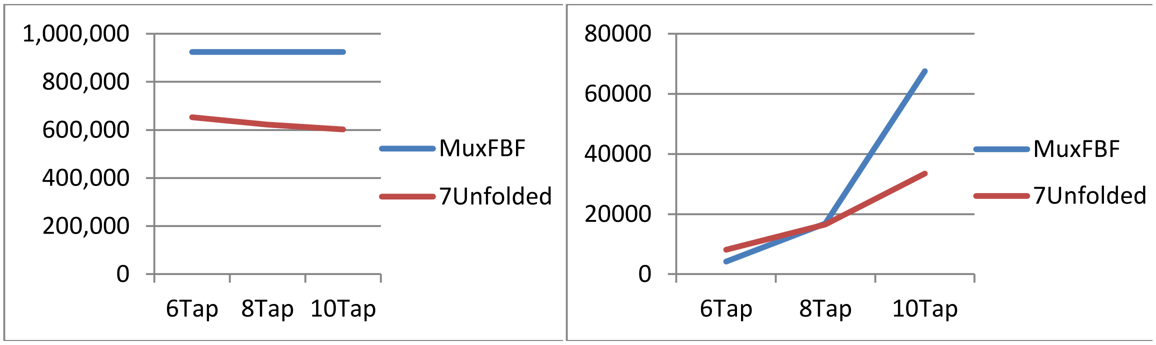
## 7.4 Discussion

While the area estimations were on point the performance of the circuit was largely underwhelming. The required unfolding factor was vastly underestimated, from the expected J=4,5 and 6 to larger than 7 for L=8,9 and 10. This, however, is not that notable, since the iteration bound estimates were very coarse and the rules for electing J too dependent on the fractional form of the IPB.



(a) Frequency comparison (b) Area comparison

**Figure 7.6: Frequency and area comparison for the Multiplexer architecture and 7 Unfolded FBF, W=8**



(a) Frequency comparison (b) Area comparison

**Figure 7.7: Frequency and area comparison for the Multiplexer architecture and 7 Unfolded FBF, W=64**

As can be seen in figures 7.6 and 7.7 our best results, for J=7, yield acceptable performance for smaller word lengths. And since a word length of 8 is much more likely than 32 or 64 for binary PAM, that is not insignificant at all. Especially for L larger than 8 the exponential growth of the single multiplexer design cannot be contained by the hardware of even large FPGAs like Virtex 5. Then the unfolded FBF is the only option. However, for L up to 8 the single multiplexer architecture is by far the superior design. And even though the performance is expected, according to table 6.13, to improve further for even larger J and eventually outperform the single multiplexer design, this will also move the L for which its limited size outweighs its suboptimal performance even further, or make it altogether too large to fit on the FPGA.

Furthermore, as we have already mentioned in 7.2 the performance of the unfolded FBF seems to be slowly reaching saturation. So to recapitulate, in the next unfolding factor, or two, the unfolded FBF will probably outperform the previous design, but it will not reach its full potential until much later. However, its use will be severely limited or nonexistent. On the other hand, the exponential area growth of the initial design is already prohibitive of the possibility of unfolding in hope of reducing the effect of the taxing register delay. And its use is already impossible for L larger than 8.

The lesson to take away from this paragraph is that instead of aiming for optimal results by unfolding the loop even more, it would be preferable to be content with adequate performance which satisfies the hardware constraints. We have confidence that a systematic and well stated retiming of the J=7 (or even 6) can yield results comparable to that of the first design while keeping the hardware overhead low enough to be of use for larger Ls.

For a more detailed exposition the reader is referred to chapter 8.

## 7.5 Retimed circuit

Below we can see the simulation results of the retimed circuit seen in Figure 6.7.

| **4Unfolded6Tap** | **8 bit** | **16 bit** | **32 bit** | **64 bit** |
|---|---|---|---|---|
| **MasterClk(MHz)** | 756.660 | 728.624 | 678.352 | 596.100 |

Table 7.6: Simulation results for the retimed 4 unfolded FBF of figure 6.7

We can compare these results to the results of the non-retimed circuit:

| **4Unfolded6Tap** | **8 bit** | **16 bit** | **32 bit** | **64 bit** |
|---|---|---|---|---|
| **MasterClk(MHz)** | 742.444 | 713.252 | 669.216 | 595.664 |

We can see the the performance only improves by around 2%. We can conlude that this specific retiming transformation on the given circuit does not manage to achieve significant improvement in performance.

## 7.6 Validation

The proper function of the various designs was tested with the use of the behavioral simulator module of Xilinx Ise. Each unit's input/output was tested separately comparing it with that of its logical function.

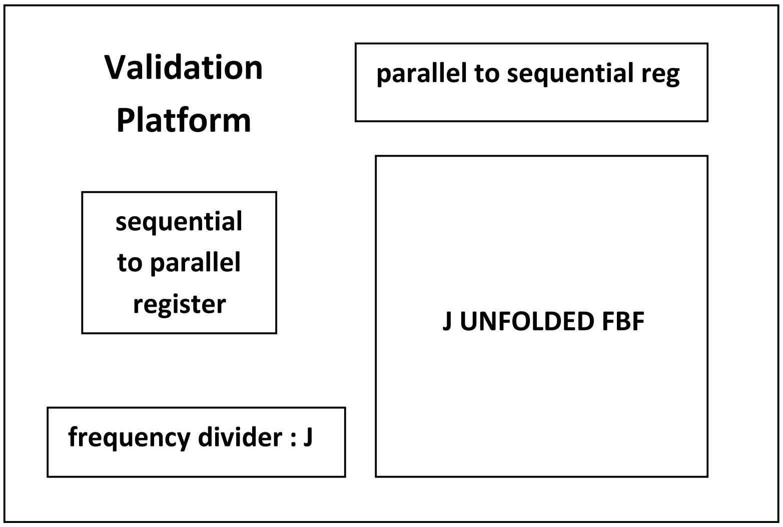


**Figure 7.8: Testing Board for the J Unfolded FBF**

Another algorithm of just the FBF was then developed, with both its pseudorandom input and output saved for comparison with the hardware implementation. The testing was straightforward for most architectures, that is, apart from the unfolded designs. For those a special validation platform had to be scripted as seen in figure 7.8. The initialization of the various units had to be done by hand and differs vastly among the various unfolding factors. Good hardware designs practices were employed, considering the edge on which we pass data in the input and the output of the J Unfolded FBF, so that there is no conflict off loading and passing times.

The keen reader who might closely examine the accompanying code supplement may notice a slight instability of the elementary coding choice of the states in the frequency divider. This is important in real world application but as we have already mentioned the simulation is behavioral and does not take into consideration any propagation delays.

# Conclusions and Recommendations 8

## 8.1 Conclusions

At this thesis we optimized the performance of a Decision Feedback Equalizer circuit. We focused on the optimization of the Feed Backward Filter, since this is the one that poses the upper limit to the frequency. We applied the unfolding and retiming techniques that are well known techniques, invented several years ago. We used FPGAs in order to simulate the performance of these circuits for several unfolding factors. Then we experimented with a retimed version of the FBF.

The speed of the FBF was increasing significantly with the increase of the unfolding factor. However, the area was increasing as well. According to the theory developed by dr Parhi, there should be an optimal unfolding factor, with which the iteration bound could be achieved. We computed this optimal unfolding factor to be between 4,5,6 for our circuits. We then performed measurements for unfolding factors ranging from 3 to 7. The results were that the frequency continued to increase significantly for unfolding factors bigger than the optimal one. That might be because the optimal unfolding factor was computed mathematically for ideal cases and for asic designs. However, for fpga designs, things are not so simple, since the output of the synthesizer is not totally controllable.

Moreover, the sub-optimally retimed 4 unfolded FBF according to table 7.6 yields a minor performance improvement, especially for shorter word lengths. The improvement is much less than the improvement achieved with the unfolding transformation, for this specific design. With a better optimized automated retiming algorithm, we have confidence that a vast performance improvement is more than possible. Which naturally brings us to the recommendations section.

Finally, after optimizing the FBF in order to reach the desired frequency, we will need to take care of the feed forward filter as well. It probably will have to be pipelined (making it N-slow) and parallelized to a level close to J to achieve similar performance metrics. Moreover, the linking adder between the two subunits will probably bottleneck the design. Additionally, we aimed for throughput rates more appropriate for ASIC centric design choices. If the volume of the circuits that are going to be produced justifies an ASIC design cost, then an ASIC implementation would be preferable.

## 8.2 Recommendations for Future Research

As we've already mentioned in chapter 4, the combination of unfolding and retiming is an advanced theoretical subject. For the researcher willing to invest the time and energy the performance of the resulting architecture can be estimated a priori to the implementation and the right retime approach applied to the initial recursive implementation, before unrolling it, which will significantly decrease the complexity.

However, nothing stops us from blindly applying a critical path minimization retiming algorithm on a larger collection of already unfolded DFGs, whose implementation satisfies the area constraint, and then picking the results that best fit our needs. But as we've seen in chapter 6 the performance and implementation of the multiplexer is not linearly dependent on its size and the CSA/RCA is hard to pipeline. So the first step would be to solve the second problem. And then for step number 2 develop a database for the various nonlinearities and general "quirkiness" of the synthesizer's mapping for various basic units (adders, multipliers, shifters, multiplexers...) and different FPGAs with widespread use. The database has a more general appeal and in our case for step 3 it can be used by a proprietarily modified critical path minimization algorithm to automatically optimize our search space. As of the date of completion of this thesis the author is not familiar with the existence of such a module.

## Bibliography

[1]. *Communication Systems Engineering , John G.Proakis ,Masoud Salehi*

[2]. *Digital Communications , John G.Proakis*

[3]. *N. H. E. Weste and D. Harris, CMOS VLSI Design, 4th edition, Pearson–Addison-Wesley, 2011.*

[4]. *Jan M. Rabaey, AnanthaChandrakasan and BorivojeNicolic, Digital Integrated Circuits, 2nd edition, Pearson Education,Inc- Prentice Hall 2003*

[5]. *KeshabK.Parhi, VLSI Digital Signal Processing Systems, Design and Implementation, Wiley Interscience- John Wiley & Sons Inc., 1999*

[6]. *Chih-Hsiu Lin and An-Yeu(Andy) Wu "Low cost decision feedback equalizer (DFE) design for giga-bit systems",Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on, vol.3,2005*

[7]. *K.K. Parhi, "Pipelining in algorithms with quantizer loops", IEEE Trans. on Circuits and Systems ,vol.37, no.7, pp. 745-754, July 1991.*

[8]. *S. Kasturia and J.H. Winters, "Techniques for high-speed implementation of nonlinear cancellation", IEEE J. Select. Areas Commun, vol.9, no.5, pp. 711-717, June 1991.*

[9]. *M.VenkataDurgaramaraju and A.Deepthi, "Design and implementation of Parallel Prefix Adders using FPGAs", IOSR Journal of Electronics and Communication Engineering, vol.6, no.5, pp. 41-48, July-August 2013*

[10]. *Florent De Dinechin, Hong Diep Nguyen, BogdanPasca. "Pipelined FPGA Adders" . 2010.*

[11]. *K. K. Parhi and D. G. Messerschmitt*, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding"

[12]. *K. K. Parhi and D. G. Messerschmitt*, "Pipeline interleaving and parallelism in recursive digital filter- Part I and II"

# PART III:

## Appendices

# ***Appendix A***

# Unfolded Circuits' Indices

The unfolding factor is displayed in the upper left corner of the table. The factored indices are on the right side of their respective sub-circuits column with their unfactored form on the left.

| UF 2 | | 2k indices | | 2k+1 indices | |
|---|---|---|---|---|---|
| 1 | $a_1$ | 2k-1 | 2(k-1)+1 | 2k | 2k |
| 2 | $a_2$ | 2k-2 | 2(k-1) | 2k-1 | 2(k-1)+1 |
| 3 | $a_3$ | 2k-3 | 2(k-2)+1 | 2k-2 | 2(k-1) |
| 4 | $a_4$ | 2k-4 | 2(k-2) | 2k-3 | 2(k-2)+1 |
| 5 | $a_5$ | 2k-5 | 2(k-3)+1 | 2k-4 | 2(k-2) |
| 6 | $a_6$ | 2k-6 | 2(k-3) | 2k-5 | 2(k-3)+1 |
| 7 | $a_7$ | 2k-7 | 2(k-4)+1 | 2k-6 | 2(k-3) |
| 8 | $a_8$ | 2k-8 | 2(k-4) | 2k-7 | 2(k-4)+1 |
| 9 | $a_9$ | 2k-9 | 2(k-5)+1 | 2k-8 | 2(k-4) |
| 10 | $a_{10}$ | 2k-10 | 2(k-5) | 2k-9 | 2(k-5)+1 |

| UF 3 | | 3k indices | | 3k+1 indices | | 3k+2 indices | |
|---|---|---|---|---|---|---|---|
| 1 | $a_1$ | 3k-1 | 3(k-1)+2 | 3k | 3k | 3k+1 | 3k+1 |
| 2 | $a_2$ | 3k-2 | 3(k-1)+1 | 3k-1 | 3(k-1)+2 | 3k | 3k |
| 3 | $a_3$ | 3k-3 | 3(k-1) | 3k-2 | 3(k-1)+1 | 3k-1 | 3(k-1)+2 |
| 4 | $a_4$ | 3k-4 | 3(k-2)+2 | 3k-3 | 3(k-1) | 3k-2 | 3(k-1)+1 |
| 5 | $a_5$ | 3k-5 | 3(k-2)+1 | 3k-4 | 3(k-2)+2 | 3k-3 | 3(k-1) |
| 6 | $a_6$ | 3k-6 | 3(k-2) | 3k-5 | 3(k-2)+1 | 3k-4 | 3(k-2)+2 |
| 7 | $a_7$ | 3k-7 | 3(k-3)+2 | 3k-6 | 3(k-2) | 3k-5 | 3(k-2)+1 |
| 8 | $a_8$ | 3k-8 | 3(k-3)+1 | 3k-7 | 3(k-3)+2 | 3k-6 | 3(k-2) |
| 9 | $a_9$ | 3k-9 | 3(k-3) | 3k-8 | 3(k-3)+1 | 3k-7 | 3(k-3)+2 |
| 10 | $a_{10}$ | 3k-10 | 3(k-4)+2 | 3k-9 | 3(k-3) | 3k-8 | 3(k-3)+1 |

| UF 4 | | 4k indices | | 4k+1 indices | | 4k+2 indices | | 4k+3 indices | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | $a_1$ | 4k-1 | 4(k-1)+3 | 4k | 4k | 4k+1 | 4k+1 | 4k+2 | 4k+2 |
| 2 | $a_2$ | 4k-2 | 4(k-1)+2 | 4k-1 | 4(k-1)+3 | 4k | 4k | 4k+1 | 4k+1 |
| 3 | $a_3$ | 4k-3 | 4(k-1)+1 | 4k-2 | 4(k-1)+2 | 4k-1 | 4(k-1)+3 | 4k | 4k |
| 4 | $a_4$ | 4k-4 | 4(k-1) | 4k-3 | 4(k-1)+1 | 4k-2 | 4(k-1)+2 | 4k-1 | 4(k-1)+3 |
| 5 | $a_5$ | 4k-5 | 4(k-2)+3 | 4k-4 | 4(k-1) | 4k-3 | 4(k-1)+1 | 4k-2 | 4(k-1)+2 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 6 | $a_6$ | 4k-6 | 4(k-2)+2 | 4k-5 | 4(k-2)+3 | 4k-4 | 4(k-1) | 4k-3 | 4(k-1)+1 |
| 7 | $a_7$ | 4k-7 | 4(k-2)+1 | 4k-6 | 4(k-2)+2 | 4k-5 | 4(k-2)+3 | 4k-4 | 4(k-1) |
| 8 | $a_8$ | 4k-8 | 4(k-2) | 4k-7 | 4(k-2)+1 | 4k-6 | 4(k-2)+2 | 4k-5 | 4(k-2)+3 |
| 9 | $a_9$ | 4k-9 | 4(k-3)+3 | 4k-8 | 4(k-2) | 4k-7 | 4(k-2)+1 | 4k-6 | 4(k-2)+2 |
| 10 | $a_{10}$ | 4k-10 | 4(k-3)+2 | 4k-9 | 4(k-3)+3 | 4k-8 | 4(k-2) | 4k-7 | 4(k-2)+1 |

| UF 5 | | 5k indices | | 5k+1 indices | | 5k+2 indices | | 5k+3 indices | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | $a_1$ | 5k-1 | 5(k-1)+4 | 5k | 5k | 5k+1 | 5k+1 | 5k+2 | 5k+2 |
| 2 | $a_2$ | 5k-2 | 5(k-1)+3 | 5k-1 | 5(k-1)+4 | 5k | 5k | 5k+1 | 5k+1 |
| 3 | $a_3$ | 5k-3 | 5(k-1)+2 | 5k-2 | 5(k-1)+3 | 5k-1 | 5(k-1)+4 | 5k | 5k |
| 4 | $a_4$ | 5k-4 | 5(k-1)+1 | 5k-3 | 5(k-1)+2 | 5k-2 | 5(k-1)+3 | 5k-1 | 5(k-1)+4 |
| 5 | $a_5$ | 5k-5 | 5(k-1) | 5k-4 | 5(k-1)+1 | 5k-3 | 5(k-1)+2 | 5k-2 | 5(k-1)+3 |
| 6 | $a_6$ | 5k-6 | 5(k-2)+4 | 5k-5 | 5(k-1) | 5k-4 | 5(k-1)+1 | 5k-3 | 5(k-1)+2 |
| 7 | $a_7$ | 5k-7 | 5(k-2)+3 | 5k-6 | 5(k-2)+4 | 5k-5 | 5(k-1) | 5k-4 | 5(k-1)+1 |
| 8 | $a_8$ | 5k-8 | 5(k-2)+2 | 5k-7 | 5(k-2)+3 | 5k-6 | 5(k-2)+4 | 5k-5 | 5(k-1) |
| 9 | $a_9$ | 5k-9 | 5(k-2)+1 | 5k-8 | 5(k-2)+2 | 5k-7 | 5(k-2)+3 | 5k-6 | 5(k-2)+4 |
| 10 | $a_{10}$ | 5k-10 | 5(k-2) | 5k-9 | 5(k-2)+1 | 5k-8 | 5(k-2)+2 | 5k-7 | 5(k-2)+3 |

| UF 5 | | 5k+4 indices | |
|---|---|---|---|
| 1 | $a_1$ | 5k+3 | 5k+3 |
| 2 | $a_2$ | 5k+2 | 5k+2 |
| 3 | $a_3$ | 5k+1 | 5k+1 |
| 4 | $a_4$ | 5k | 5k |
| 5 | $a_5$ | 5k-1 | 5(k-1)+4 |
| 6 | $a_6$ | 5k-2 | 5(k-1)+3 |
| 7 | $a_7$ | 5k-3 | 5(k-1)+2 |
| 8 | $a_8$ | 5k-4 | 5(k-1)+1 |
| 9 | $a_9$ | 5k-5 | 5(k-1) |
| 10 | $a_{10}$ | 5k-6 | 5(k-2)+4 |

| UF 6 | | 6k indices | | 6k+1 indices | | 6k+2 indices | | 6k+3 indices | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | $a_1$ | 6k-1 | 6(k-1)+5 | 6k | 6k | 6k+1 | 6k+1 | 6k+2 | 6k+2 |
| 2 | $a_2$ | 6k-2 | 6(k-1)+4 | 6k-1 | 6(k-1)+5 | 6k | 6k | 6k+1 | 6k+1 |
| 3 | $a_3$ | 6k-3 | 6(k-1)+3 | 6k-2 | 6(k-1)+4 | 6k-1 | 6(k-1)+5 | 6k | 6k |
| 4 | $a_4$ | 6k-4 | 6(k-1)+2 | 6k-3 | 6(k-1)+3 | 6k-2 | 6(k-1)+4 | 6k-1 | 6(k-1)+5 |
| 5 | $a_5$ | 6k-5 | 6(k-1)+1 | 6k-4 | 6(k-1)+2 | 6k-3 | 6(k-1)+3 | 6k-2 | 6(k-1)+4 |
| 6 | $a_6$ | 6k-6 | 6(k-1) | 6k-5 | 6(k-1)+1 | 6k-4 | 6(k-1)+2 | 6k-3 | 6(k-1)+3 |
| 7 | $a_7$ | 6k-7 | 6(k-2)+5 | 6k-6 | 6(k-1) | 6k-5 | 6(k-1)+1 | 6k-4 | 6(k-1)+2 |
| 8 | $a_8$ | 6k-8 | 6(k-2)+4 | 6k-7 | 6(k-2)+5 | 6k-6 | 6(k-1) | 6k-5 | 6(k-1)+1 |
| 9 | $a_9$ | 6k-9 | 6(k-2)+3 | 6k-8 | 6(k-2)+4 | 6k-7 | 6(k-2)+5 | 6k-6 | 6(k-1) |
| 10 | $a_{10}$ | 6k-10 | 6(k-2)+2 | 6k-9 | 6(k-2)+3 | 6k-8 | 6(k-2)+4 | 6k-7 | 6(k-2)+5 |

| UF 6 | | 6k+4 indices | | 6k+5 indices | |
|---|---|---|---|---|---|
| 1 | $a_1$ | 6k+3 | 6k+3 | 6k+4 | 6k+4 |
| 2 | $a_2$ | 6k+2 | 6k+2 | 6k+3 | 6k+3 |

| | | | | | |
|---|---|---|---|---|---|
| 3 | $a_3$ | 6k+1 | 6k+1 | 6k+2 | 6k+2 |
| 4 | $a_4$ | 6k | 6k | 6k+1 | 6k+1 |
| 5 | $a_5$ | 6k-1 | 6(k-1)+5 | 6k | 6k |
| 6 | $a_6$ | 6k-2 | 6(k-1)+4 | 6k-1 | 6(k-1)+5 |
| 7 | $a_7$ | 6k-3 | 6(k-1)+3 | 6k-2 | 6(k-1)+4 |
| 8 | $a_8$ | 6k-4 | 6(k-1)+2 | 6k-3 | 6(k-1)+3 |
| 9 | $a_9$ | 6k-5 | 6(k-1)+1 | 6k-4 | 6(k-1)+2 |
| 10 | $a_{10}$ | 6k-6 | 6(k-1) | 6k-5 | 6(k-1)+1 |

| UF 7 | | 7k indices | | 7k+1 indices | | 7k+2 indices | | 7k+3 indices | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | $a_1$ | 7k-1 | 7(k-1)+6 | 7k | 7k | 7k+1 | 7k+1 | 7k+2 | 7k+2 |
| 2 | $a_2$ | 7k-2 | 7(k-1)+5 | 7k-1 | 7(k-1)+6 | 7k | 7k | 7k+1 | 7k+1 |
| 3 | $a_3$ | 7k-3 | 7(k-1)+4 | 7k-2 | 7(k-1)+5 | 7k-1 | 7(k-1)+6 | 7k | 7k |
| 4 | $a_4$ | 7k-4 | 7(k-1)+3 | 7k-3 | 7(k-1)+4 | 7k-2 | 7(k-1)+5 | 7k-1 | 7(k-1)+6 |
| 5 | $a_5$ | 7k-5 | 7(k-1)+2 | 7k-4 | 7(k-1)+3 | 7k-3 | 7(k-1)+4 | 7k-2 | 7(k-1)+5 |
| 6 | $a_6$ | 7k-6 | 7(k-1)+1 | 7k-5 | 7(k-1)+2 | 7k-4 | 7(k-1)+3 | 7k-3 | 7(k-1)+4 |
| 7 | $a_7$ | 7k-7 | 7(k-1) | 7k-6 | 7(k-1)+1 | 7k-5 | 7(k-1)+2 | 7k-4 | 7(k-1)+3 |
| 8 | $a_8$ | 7k-8 | 7(k-2)+6 | 7k-7 | 7(k-1) | 7k-6 | 7(k-1)+1 | 7k-5 | 7(k-1)+2 |
| 9 | $a_9$ | 7k-9 | 7(k-2)+5 | 7k-8 | 7(k-2)+6 | 7k-7 | 7(k-1) | 7k-6 | 7(k-1)+1 |
| 10 | $a_{10}$ | 7k-10 | 7(k-2)+4 | 7k-9 | 7(k-2)+5 | 7k-8 | 7(k-2)+6 | 7k-7 | 7(k-1) |

| UF 7 | | 7k+4 indices | | 7k+5 indices | | 7k+6 indices | |
|---|---|---|---|---|---|---|---|
| 1 | $a_1$ | 7k+3 | 7k+3 | 7k+4 | 7k+4 | 7k+5 | 7k+5 |
| 2 | $a_2$ | 7k+2 | 7k+2 | 7k+3 | 7k+3 | 7k+4 | 7k+4 |
| 3 | $a_3$ | 7k+1 | 7k+1 | 7k+2 | 7k+2 | 7k+3 | 7k+3 |
| 4 | $a_4$ | 7k | 7k | 7k+1 | 7k+1 | 7k+2 | 7k+2 |
| 5 | $a_5$ | 7k-1 | 7(k-1)+6 | 7k | 7k | 7k+1 | 7k+1 |
| 6 | $a_6$ | 7k-2 | 7(k-1)+5 | 7k-1 | 7(k-1)+6 | 7k | 7k |
| 7 | $a_7$ | 7k-3 | 7(k-1)+4 | 7k-2 | 7(k-1)+5 | 7k-1 | 7(k-1)+6 |
| 8 | $a_8$ | 7k-4 | 7(k-1)+3 | 7k-3 | 7(k-1)+4 | 7k-2 | 7(k-1)+5 |
| 9 | $a_9$ | 7k-5 | 7(k-1)+2 | 7k-4 | 7(k-1)+3 | 7k-3 | 7(k-1)+4 |
| 10 | $a_{10}$ | 7k-6 | 7(k-1)+1 | 7k-5 | 7(k-1)+2 | 7k-4 | 7(k-1)+3 |

# *Appendix B*

# Area and Speed Measurements

## B1. Retimed Multiplexer Architecture

| MuxFBF6Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 639 | 1151 | 2175 | 4223 |
| **MaxFreq(MHz)** | 924.556 | | | |

| MuxFBF7Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 1279 | 2303 | 4351 | 8447 |
| **MaxFreq(MHz)** | 924.556 | | | |

| MuxFBF8Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 2559 | 4607 | 8703 | 16895 |
| **MaxFreq(MHz)** | 924.556 | | | |

| MuxFBF9Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 5119 | 8210 | 17407 | 33791 |
| **MaxFreq(MHz)** | 924.556 | | | |

| MuxFBF10Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 10239 | 18431 | 34815 | 67583 |
| **MaxFreq(MHz)** | 924.556 | | | |

## B2. Two Stage Pre-Computation FBF

| FBF6Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 164 | 316 | 620 | 1228 |
| **MaxFreq(MHz)** | 360.308 | 338.135 | 301.078 | 246.950 |

| FBF7Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 189 | 365 | 717 | 1421 |
| **MaxFreq(MHz)** | 360.308 | 338.135 | 301.078 | 246.950 |

| FBF8Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 327 | 631 | 1239 | 2456 |
| **MaxFreq(MHz)** | 301.566 | 285.876 | 258.932 | 217.864 |

| FBF9Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 369 | 713 | 1401 | 2777 |
| **MaxFreq(MHz)** | 301.566 | 285.876 | 258.932 | 217.864 |

| FBF10Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 646 | 1246 | 2446 | 4846 |
| **MaxFreq(MHz)** | 282.679 | 268.847 | 244.883 | 207.832 |

## B3. Unfolded Circuits

For the Unfolded circuits the master clock will obviously operate at J times the measured frequency.

| 2Unfolded6Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 312 | 600 | 1176 | 2328 |
| **MaxFreq(MHz)** | 237.662 | 225.828 | 208.455 | 180.660 |
| **MasterClk(MHz)** | 475.324 | 451.656 | 416.910 | 361.320 |

| 2Unfolded8Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 656 | 1232 | 2384 | 4688 |
| **MaxFreq(MHz)** | 205.525 | 197.818 | 184.018 | 161.486 |
| **MasterClk(MHz)** | 411.050 | 395.636 | 368.036 | 322.972 |

| 2Unfolded10Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 1276 | 2460 | 4828 | 9564 |
| **MaxFreq(MHz)** | 173.582 | 167.981 | 157.798 | 140.773 |
| **MasterClk(MHz)** | 347.164 | 335.962 | 315.596 | 281.546 |

| 3Unfolded6Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 465 | 897 | 1761 | 3489 |
| **MaxFreq(MHz)** | 208.640 | 199.463 | 185.788 | 163.384 |
| **MasterClk(MHz)** | 625.920 | 598.389 | 557.364 | 490.152 |

| 3Unfolded8Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 980 | 1843 | 3571 | 7033 |
| **MaxFreq(MHz)** | 186.050 | 179.631 | 168.035 | 149.737 |
| **MasterClk(MHz)** | 558.150 | 538.893 | 504.105 | 449.211 |

| 3Unfolded10Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 1909 | 3685 | 7237 | 14341 |
| **MaxFreq(MHz)** | 157.573 | 152.944 | 144.456 | 130.057 |
| **MasterClk(MHz)** | 472.719 | 458.832 | 433.368 | 390.171 |

| 4Unfolded6Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 618 | 1194 | 2346 | 4650 |
| **MaxFreq(MHz)** | 185.611 | 178.313 | 167.304 | 148.916 |
| **MasterClk(MHz)** | 742.444 | 713.252 | 669.216 | 595.664 |

| 4Unfolded8Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 1306 | 2458 | 4760 | 9368 |
| **MaxFreq(MHz)** | 165.930 | 160.935 | 148.263 | 133.459 |
| **MasterClk(MHz)** | 663.720 | 643.740 | 593.052 | 533.836 |

| 4Unfolded10Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 2542 | 4910 | 9646 | 19118 |
| **MaxFreq(MHz)** | 144.073 | 140.193 | 133.029 | 120.721 |
| **MasterClk(MHz)** | 576.292 | 560.772 | 532.116 | 482.884 |

| 5Unfolded6Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 771 | 1492 | 2933 | 5813 |
| **MaxFreq(MHz)** | 150.531 | 141.250 | 127.836 | 107.600 |
| **MasterClk(MHz)** | 752.655 | 706.250 | 639.180 | 538.000 |

| 5Unfolded8Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 1634 | 3074 | 5954 | 11714 |
| **MaxFreq(MHz)** | 156.543 | 152.056 | 143.664 | 128.638 |
| **MasterClk(MHz)** | 782.715 | 760.280 | 718.320 | 643.190 |

| 5Unfolded10Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 3175 | 6135 | 12055 | 23899 |
| **MaxFreq(MHz)** | 132.049 | 128.782 | 122.711 | 112.163 |
| **MasterClk(MHz)** | 660.245 | 643.910 | 613.555 | 560.815 |

| 6Unfolded6Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 925 | 1789 | 3517 | 6974 |
| **MaxFreq(MHz)** | 136.653 | 128.898 | 117.529 | 100.052 |
| **MasterClk(MHz)** | 819.918 | 773.388 | 705.174 | 600.312 |

| 6Unfolded8Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 2213 | 4333 | 7958 | 15225 |
| **MaxFreq(MHz)** | 129.134 | 122.936 | 112.531 | 96.718 |
| **MasterClk(MHz)** | 774.804 | 737.616 | 675.186 | 580.308 |

| 6Unfolded10Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 3808 | 7360 | 14468 | 28672 |
| **MaxFreq(MHz)** | 121.374 | 118.608 | 113.440 | 104.366 |
| **MasterClk(MHz)** | 728.244 | 711.648 | 680.640 | 626.196 |

| 7Unfolded6Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 1079 | 2087 | 4104 | 8136 |
| **MaxFreq(MHz)** | 125.081 | 118.464 | 108.643 | 93.320 |
| **MasterClk(MHz)** | 875.567 | 829.248 | 760.501 | 653.240 |

| 7Unfolded8Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 2589 | 5014 | 8445 | 16588 |
| **MaxFreq(MHz)** | 115.467 | 110.453 | 104.838 | 88.946 |
| **MasterClk(MHz)** | 808.269 | 773.171 | 733.866 | 622.622 |

| 7Unfolded10Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **Area(LUTs)** | 4442 | 8586 | 16875 | 33451 |
| **MaxFreq(MHz)** | 111.096 | 106.649 | 98.743 | 86.009 |
| **MasterClk(MHz)** | 777.672 | 746.543 | 691.201 | 602.063 |

# B4 Retimed Circuit

| 4Unfolded6Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **MasterClk(MHz)** | 756.660 | 728.624 | 678.352 | 596.100 |

Comparison with non-retimed circuit:

| 4Unfolded6Tap | 8 bit | 16 bit | 32 bit | 64 bit |
|---|---|---|---|---|
| **MasterClk(MHz)** | 742.444 | 713.252 | 669.216 | 595.664 |