LEAST-SQUARES APPROXIMATION AND POLYPHASE DECOMPOSITION FOR PIPELINING RECURSIVE FILTERS

BY

ADITYA GUPTA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Adviser:

Professor Andrew Singer

ABSTRACT

Current techniques used in pipelining recursive filters require high hardware complexity. These techniques attempt to preserve the exact frequency response of the original circuit while seeking to construct a pipelined architecture. We present a technique that relaxes the need to preserve the exact frequency response and instead considers a least-squares formulation in conjunction with the pipelined architecture. The benefit of this design is that it reduces the complexity of the pipelined circuit immensely, while enabling a simple pipelined architecture based on a polyphase decomposition of the original filter.

ACKNOWLEDGMENTS

I would like to thank Prof. Andrew Singer and Prof. Naresh Shanbhag for their constant guidance and help in finishing this project.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Moore's law has provided speed and power benefits at the device level for the last few decades. We approach physical limitations that make developing smaller devices increasingly difficult. It is imperative to probe and develop additional techniques to pursue higher throughput and achieve lower power in digital circuits. One such technique is pipelining. By pipelining an architecture, we enable the architecture to operate at higher clock rates and, if it is designed to run slower than the achievable throughput, enable reduced power consumption. In this work, we will focus on one particular kind of circuit, namely, a recursive filter. Unlike feed forward digital filters with finite length impulse response (FIR), which can be easily pipelined using feed forward cut-sets (therefore translating into higher clock rates), recursive sections are more difficult to pipeline and typically require more hardware to do so. Pipelining recursive circuits by delay scaling or introducing latches in the feedback loop is not always effective. Delay scaling (look-ahead transformations) [1]-[5] can be useful for applications that need moderate sampling rates or that have multiple independent time series that need processing. If such time series are available, then they could be efficiently filtered using time interleaving techniques. In most real systems this is not the case. A variety of techniques are described below that are primarily used for pipelining recursive circuits. In this work, we introduce a new technique that can be used to pipeline recursive filters. This approach does not attempt to preserve the exact frequency response of the original filter to be pipelined. Rather we trade off exact functionality in favor of an ease in pipelining.

The method proposed in this thesis strives to preserve the characteristics of the original filter (pass band, transition band, stop band attenuation, and phase response) while providing stability and yet significantly reducing the complexity in hardware as compared with other related methods for pipelining recursive filters.

CHAPTER 2

CLUSTERED LOOK-AHEAD

The clustered look-ahead method uses pole zero cancellation to preserve the original filter characteristics as described in [6]. One of the problems with this method is that one or more of the cancelling poles can be outside of the unit circle and, therefore, can make the system unstable. It makes use of additional overhead in the feed forward section, which utilizes additional adders and multipliers. Clustered look-ahead (CLA) realization is a result of back subsitution in the original difference equation. Back subsitution $P-1$ times results in a P-fold pipelined filter. The additional hardware complexity needed for CLA is P multiply and accumulates (MACs), where P is the pipelining level. The complexity added due to latches is ignored since it is negligible when compared to the complexity added by MACs.

Given an equation,

$$x(n+1) = ax(n) + bu(n) \tag{2.1}$$

it can be pipelined by M levels using

$$x(n+P) = a^P x(n) \sum_{i=0}^{P-1} a^i b u(n+M-1-i) \tag{2.2}$$

and depicted by the architecture shown in Figure 2.1.

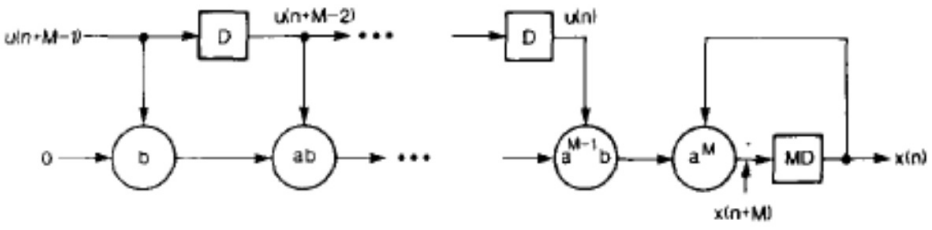


Figure 2.1: Pipelined transformation using clustered look-ahead scheme.

Given a system with unput $u[n]$ and output $y[n]$,

$$y(n) = \sum_{i=1}^{N} a_i y(n-i) + \sum_{i=0}^{N} b_i u(n-i) \tag{2.3}$$

let

$$z(n) = \sum_{i=0}^{N} b_i u(n-i). \tag{2.4}$$

Then the time domain pipelined filter is given by

$$y(n) = \sum_{j=0}^{N-1} \left[\sum_{k=j+1}^{N} a_k r_{j+P-k} \right] y(n-j-P) + \sum_{j=0}^{P-1} r_j z(n-j) \tag{2.5}$$

where
$r_i = 0, \ for \ \ i < 0,$
$r_i = 1, \ for \ \ i = 0,$ and
$r_i = \sum_{k=1}^{N} a_k r_{i-k}, \ for \ \ i > 0.$

The z-transform of the above time domain equation yields,

$$H(z) = \frac{\sum_{i=0}^{P-1} r_i \sum_{j=0}^{N} b_j z^{-(i+j)}}{1 - \sum_{i=0}^{N-1} \left[\sum_{j=i+1}^{N} a_j r_{i+P-j} \right] z^{-(i+P)}}. \tag{2.6}$$

It can be readily seen that the coefficients of $z^{-1}, ..., z^{-P+1}$ in the denominator are zero. Therefore, this can be a P level pipelined realization.

Let us see an example where we will use this technique. To implement a level 2 pipelining, but the resulting filter will be unstable and, therefore, display a weakness of this technique.

Let us consider an all-pole filter with poles at z = 1/2 and z = 3/4. Therefore,

$$H(z) = \frac{1}{1 - \frac{5}{4} z^{-1} + \frac{3}{8} z^{-2}}, \tag{2.7}$$

which has the following difference equation

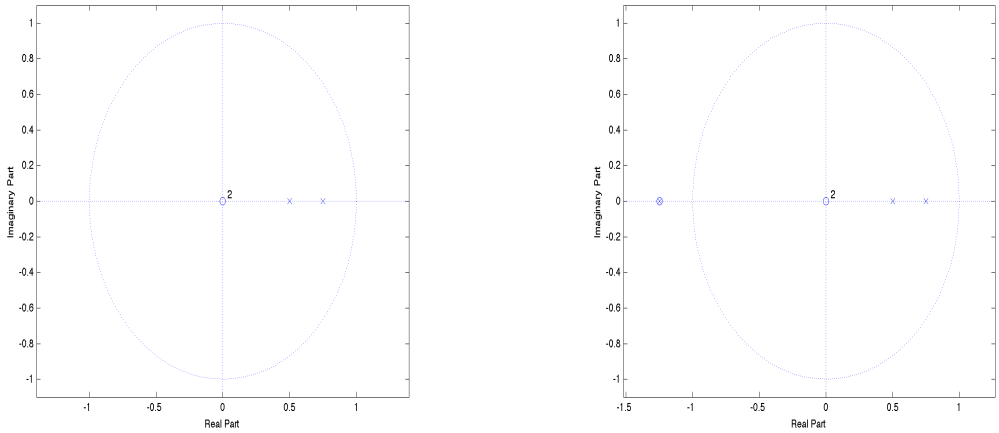
$$y(n) = x(n) + \frac{5}{4} y(n-1) - \frac{3}{8} y(n-2) \tag{2.8}$$

Using back substitution, at $P = 2$,

$$y(n) = x(n) + \frac{5}{4}x(n-1) + \frac{19}{16}y(n-2) - \frac{15}{32}y(n-3), \tag{2.9}$$

or,

$$H(z) = \frac{1 + \frac{5}{4}z^{-1}}{1 - \frac{19}{16}z^{-2} + \frac{15}{32}z^{-3}}. \tag{2.10}$$



(a) original pole-zero diagram

(b) pipelined pole-zero diagram

Figure 2.2: Clustered look-ahead transformation.

It can be seen in Figure 2.2 (b) that a cancelling pole and zero were introduced at z = -5/4 to achieve this pipelining. Since the resulting pole is outside of the unit circle, this filter becomes unstable. Similarly it can be shown that even at P = 3, the resulting filter is unstable. It can be shown that there is no guarantee of getting a stable pipelined filter using this technique. The additional hardware complexity needed for CLA is P, where P is the pipelining level.

CHAPTER 3

SCATTERED LOOK-AHEAD

Scattered look-ahead (SLA) technique is similar to CLA in the sense that it too introduces cancelling poles and zeros in the transfer function [7]. For each pole of the original system, $P-1$ cancelling poles and zeros are introduced. These cancelling poles and zeros are introduced in such a way that they are evenly spread out around the origin and are equidistant (therefore symmetric). Unlike CLA, SLA always results in a stable pipelined filter. Additional hardware complexity needed is $N(M-1)$ multiplys and accumulates (MACs). The resulting system function is

$$H(z) = \frac{N(z)}{D(z)} = \frac{N(z)\prod_{k=1}^{P-1} D(ze^{j2\pi k/P})}{\prod_{k=0}^{P-1} D(ze^{j2\pi k/P})} = \frac{N'(z)}{D'(z)}, \tag{3.1}$$

where

$$D(z) = \left[1 - \sum_{i=1}^{N} a_i z^{-i}\right]\left[1 + \sum_{j=1}^{P-1} r_j z^{-j}\right]. \tag{3.2}$$

Let us look at the example shown in Chapter 2, where the poles of the original transfer function are at 1/2 and 3/4. We add poles and zeros at $\frac{1}{2}e^{\pm j2\pi/3}$ and at $\frac{3}{4}e^{\pm j2\pi/3}$ in order to pipeline this recursive filter by P = 3. The pole zero plot is shown in Figure 3.1.

The overhead of this technique is much larger than CLA as it is depicted by the architecture shown in Figure 3.2.

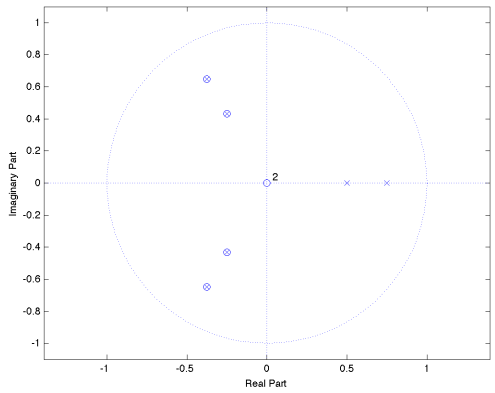


Figure 3.1: Pipelined pole-zero plot.

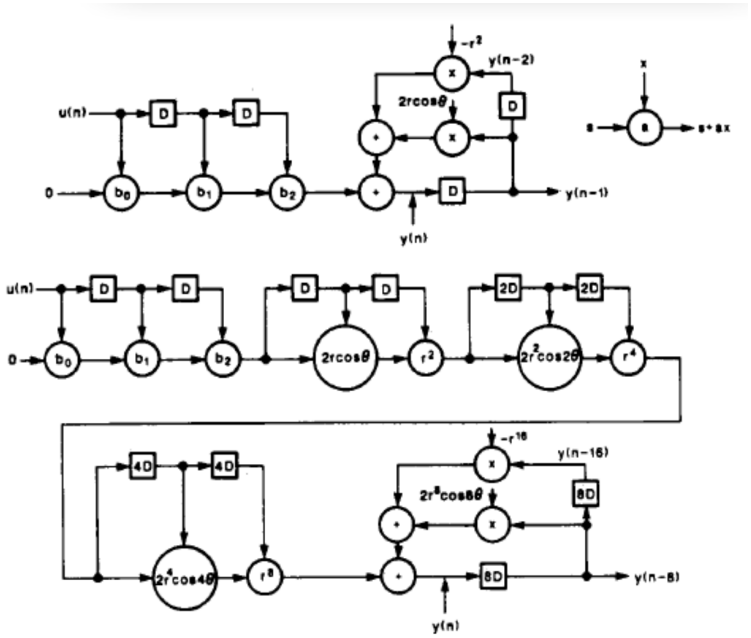


Figure 3.2: Pipelined transformation using the scattered look-ahead scheme.

CHAPTER 4

POLYPHASE DECOMPOSITION TECHNIQUE

Infinite-length impulse response (IIR) filters can also be pipelined by working with the polyphase decomposition [8]. Using constrained design in a polyphase decomposition technique, the resulting filter has intrinsic pipelining delay scaling. The complexity of implementing this technique is comparable to scattered look-ahead.

This method consists of four steps.

Step 1: Given a rational transfer function, using the constrained filter design method of [8], restrict the denominator polynomial to be a function of z^P instead of z. Therefore, given

$$H(z) = \frac{B(z)}{A(z)} \tag{4.1}$$

make the transformation, such that

$$H(z) = \frac{B(z)}{A(z^P)}. \tag{4.2}$$

Step 2: Using FIR polyphase decomposition, decompose $A(z)$ into polyphase filters, such that

$$H(z) = \sum_{i=0}^{P-1} H^{(i)}(z^P) z^{-i}, \tag{4.3}$$

where

$$H^{(i)}(z^P) = \frac{B^{(i)}(z^P)}{A(z^P)}. \tag{4.4}$$

Step 3: Evaluate $H^{(i)}(z^P)$ for each i by the method shown above.

Step 4: Set up the individual filters in a polyphase architecture to pipeline the resulting system directly exploiting the the z^P structure.

CHAPTER 5

GENERAL LOOK-AHEAD SCHEME AND OPTIMAL APPROXIMATION

In the general look-ahead scheme, Shaw and Ahmed [9] decouple the numerator and denominator coefficients to find a set of coefficients that would be a close approximation to the given filter and yet be a pipelined implementation. As shown in [10], an alternating non-linear minimization procedure is used to approximate the resulting filter.

This minimization is performed through an iterative algorithm, which is terminated when each coefficent $|a_k^l - a_k^{l-1}| \leq \delta$, where δ is an arbitrary number. Unfortunately, we were unable to reproduce the performance of this algorithm. There were no examples presented in [9]. We tried simulating the example presented for this thesis research, but the algorithm presented in [9] resulted in an unstable pipelined architecture.

CHAPTER 6

LEAST-SQUARES APPROXIMATION USING PRONY'S METHOD AND POLYPHASE DECOMPOSITION

In this section we describe a new method that we have developed for this thesis research for pipelining recursive filters based on Prony's method and polyphase architectures.

6.1 Time Domain Polyphase Decomposition

Prony's method based polyphase decomposition is similar to the polyphase decomposition method described in Chapter 4. We use the idea of polyphase decomposition, but only to seek an approximation of $H(z)$. Also, we focus on the IIR section. Our aim idea is to make the following approximation:

$$H(z) = \frac{B(z)}{A(z)} = B(z)\frac{1}{A(z)} \approx B(z)\sum_{i=0}^{P-1}\frac{c_i}{A^{(i)}(z^P)}. \tag{6.1}$$

Therefore, some key differences between our method and the polyphase decomposition method presented above are

(a) The desired impulse response is an approximation.

(b) The polyphase decomposition is of the IIR section only.

(c) Unlike the non-linear approximation performed in [9], we perform a linear least-sqaures approximation that is easy to implement.

6.2 Prony's Method

We use Prony's method to perform the approximation as an established IIR filter design and an easy optimization problem. Given an all-pole IIR filter, our aim is to recover the recursive coefficients a_ks to fit the desired h_d by minimizing the squared prediction error,

$$\varepsilon_p^2 = \sum_{n=0}^{\infty}(h_d[n] - \sum_{k=1}^{N} a_k h_n[n-k])^2, \tag{6.2}$$

where h_d is the desired impulse response. In the matrix form, equation (6.2) can be written as

$$\begin{pmatrix} h_d(0) & 0 & \dots & 0 \\ h_d(1) & h_d(0) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_d(N-1) & h_d(N-2) & \dots & h_d(N-M) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_M \end{pmatrix} \approx - \begin{pmatrix} h_d(1) \\ h_d(2) \\ \vdots \\ h_d(N) \end{pmatrix}$$

or,

$$H_d a \approx -h_d. \tag{6.3}$$

The least-squares solution of the system of linear equations presented by equation (6.3) is given by

$$a_{opt} = -((H_d^T H_d)^{-1} H_d^T h_d. \tag{6.4}$$

We split h_d into P independent impulse responses in the time domain. Then through application of Prony's method, we derive pipelined coefficients for each of the P filters. Note that each of these filters is of the form $H(z^P)$. Therefore, there are $P-1$ coefficients that are equal to zero before each non-zero coefficient of the denominator.

The approximation using Prony's method could be made more accurate by adding an FIR section to each of the decomposed filters. The FIR coefficients for each filter can be calculated by equation (6.5), which is also called Pade's method:

$$b_k = \sum_{k=1}^{N} a_k h(n-k). \tag{6.5}$$

CHAPTER 7

SIMULATION RESULTS

The results of this thesis research can be summarized as follows:

1. For a particular pipelining level P, the approximation gets worse as the filter order is increased.

2. The approximation gets better as $P > N$ for a filter of order N.

3. The approximation gets significantly more accurate as the pass band of the filter that has to be approximated gets larger.

We performed the Prony-based polyphase decomposed method by restricting each decomposed filter to two non-zero coefficients. Each of these filters is stable. Through more simulations, we recognized that as the depth in pipelining level increased with respect to the order of the original filter, the second coefficient of each filter became ineffective, so much so, that it could be removed entirely from the approximation without changing the impulse response in any significant way. As the depth of pipelining increases, the second coefficient used for approximation tends to zero. This implies that a filter of an arbitrary order can be pipelined in an approximate sense to an arbitrarily high pipelining level P through approximation with P (one coefficient) filters by using the Prony-based polyphase decomposition method. Therefore, added complexity is $3P - N$ MACs when two coefficients are used for each polyphase filter, but $2P - N$ MACs when only one coefficient is used per decomposed filter.

This pattern holds when a majority of the energy of a recursive filter sits in its first few taps. Therefore, as the pipelining level increases, there are more filters available, which are interleaved between each other; as a result, they provide more control on how we choose the first few taps and, therefore, most of the energy of the original filter is retained.

To evaluate results, the following methods were used:

1. Evaluate the integral,

$$\frac{1}{2\pi}\int W(w)(H(w) - H_d(w))^2 dw, \tag{7.1}$$

where $W(w)$ is a window function and can be used to evaluate error in the pass, transition, and stop bands. Unfortunately, this integral is not always analytically tractable; therefore, alternative methods that are proposed below were used.

2. Evaluate a large FFT of both the desired and the approximated frequency response, and then evaluate

$$\frac{1}{N}\sum_{k=1}^{N} W(k)(H(k) - H_d(k))^2, \tag{7.2}$$

which is a numerical approximation of the integral above. After evaluating the quantity above, it was normalized by

$$\frac{1}{N}\sum_{k=1}^{N} W(k)H_d(k)^2. \tag{7.3}$$

We let the normalized error in the pass, transition, stop, and total bands be PE, QE, SE, and TE respectively using this metric.

3. Using Parseval's relation,

$$\varepsilon = \frac{1}{2\pi}\int_0^{2\pi} (H(w) - H_d(w))^2 dw = \sum_{k=0}^{\infty}(h(k) - h_d(k))^2 \approx \sum_{k=0}^{N}(h(k) - h_d(k))^2. \tag{7.4}$$

For N sufficiently large, the above integral could be evaluated directly from the impulse response of the two filters. The quantity above is normalized by

$$\sum_{k=0}^{N} h_d(k)^2. \tag{7.5}$$

We let the total normalized error calculated using this method be PTE.

Tables 7.1 and 7.2 show error in dB for filters of pass band of 0.2π and 0.4π, respectively. Table 7.3 shows the addition in hardware complexity caused due to pipelining and compares various methods to the Prony-based polyphase decompostion method presented in Chapter 6.

Table 7.1: Error in approximation in dB in filters with pass band of 0.2π

N	**P**	**PE**	**QE**	**SE**	**TE**	**PTE**
1	3	-32	32	31	32	33
2	3	-1.7	-1.4	-0.9	-1.5	-1.5
2	4	-2.6	-2.3	-1.7	-2.3	-2.3
3	4	-1.7	-0.9	-0.2	-1.2	-1.2
3	5	-2.8	-2.2	-0.7	-2.3	-2.3
3	6	-4.1	-3.8	-2.8	-3.9	-3.9
4	6	-3.6	-2.4	-0.5	-2.6	-2.6
4	8	-2.5	-2.2	-0.3	-2.3	-2.3
5	7	-2.6	-1.8	-0.3	-2.3	-2.3
5	20	-4.9	-4.1	-1.7	-4.5	-4.5
7	20	-3.2	-2.4	-0.9	-2.9	-2.9
8	16	-2.0	-1.1	-2.7	-1.7	-1.7
8	20	-2.9	-2.1	-1.3	-2.7	-2.7
10	15	-1.9	-0.7	-4.1	-1.5	-1.5
10	40	-4.5	-3.4	-2.1	-3.9	-3.9

Table 7.2: Error in approximation in dB in filters with pass band of 0.4π

N	**P**	**PE**	**QE**	**SE**	**TE**	**PTE**
1	3	-34	34	34	34	36
2	3	-6.3	-6.0	-6.1	-6.2	-6.2
2	4	-5.5	-5.2	-5.3	-5.4	-5.4
3	4	-3.3	-2.9	-2.9	-3.1	-3.1
3	5	-6.9	-6.7	-6.1	-6.7	-6.7
3	6	-5.3	-5.0	-4.6	-5.1	-5.1
4	6	-3.5	-3.1	-2.6	-3.2	-3.2
4	8	-6.0	-5.6	-4.7	-5.6	-5.6
5	7	-2.9	-2.8	-1.6	-2.6	-2.6
5	20	-8.3	-8.1	-6.9	-8.0	-8.0
7	20	-7.9	-8.0	-6.7	-7.9	-7.9
8	16	-4.3	-4.0	-2.1	-4.0	-4.0
8	20	-6.8	-6.9	-5.4	-6.8	-6.8
10	15	-3.6	-4.4	-1.6	-3.7	-3.7
10	40	-9.9	-10.8	-7.6	-9.9	-9.9

Table 7.3: Hardware complexity added (additional MACs) by using CLA, SLA, polyphase decomposition and Prony-based polyphase decomposition

N	**P**	**CLA**	**SLA**	**PolyD**	**Prony(2nd order)**	**Prony (1st order)**
1	3	3	3	6	9	5
2	3	3	5	12	9	4
2	4	4	7	16	12	6
3	4	4	10	24	12	5
3	5	5	13	30	15	7
3	6	6	16	36	18	9
4	6	6	21	48	18	8
4	8	8	29	64	24	12
5	7	7	31	70	21	9
5	20	20	96	200	60	35
7	20	20	134	280	60	33
8	16	16	121	256	48	24
8	20	20	153	320	60	32
10	15	15	141	300	45	20
10	40	40	391	800	120	70

CHAPTER 8

COMPARISON BETWEEN PRONY-BASED POLYPHASE DECOMPOSITION AND OTHER APPROXIMATION TECHNIQUES

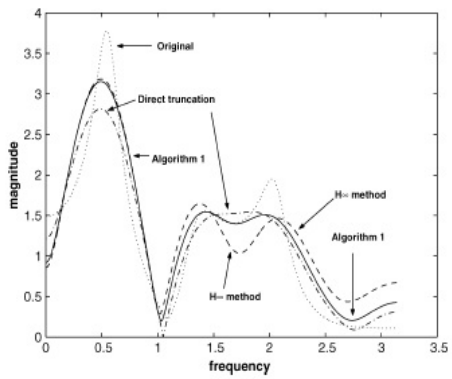
We have looked at FIR approximation of IIR filters (FIR filters are readily pipelinable, with almost no additional complexity except latches), and also at IIR approximations of IIR filters. There is a lot of work that has been done on the former and, therefore, we will compare the approximation performance of other methods to the Prony-based polyphase decomposition method.

Since there are various methods given for FIR approximation of IIR filters, we will compare our results to that of the most recent work in [11] that claims to have the best FIR approximation of IIR filters.

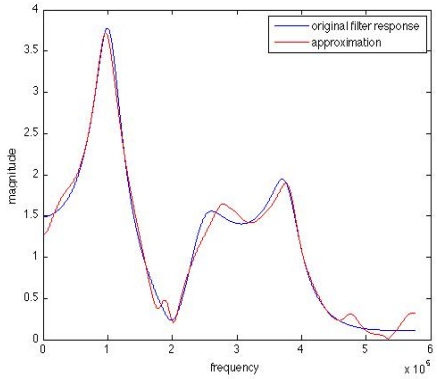
Example: This example is taken from [11], where $G(z)$ is a model IIR filter with many corner cases as shown in Figure 8.1 (a).

$$G(z) = \frac{-0.1242z^5 + 0.1581z^4 + 0.5273z^3 + 0.2154z^2 - 0.0647z^1 + 0.6889}{z^6 - 1.095z^5 + 1.299z^4 - 1.113z^3 + 1.028z^2 - 0.6043z^1 + 0.426}. \tag{8.1}$$

Figures 8.1 and 8.2 compare the approximation quality of the various methods discussed in [11] with our method with pipelining level of 12 and 24, respectively. Figure 8.3 compares the l^2 error of various approximation techniques to that of our method.

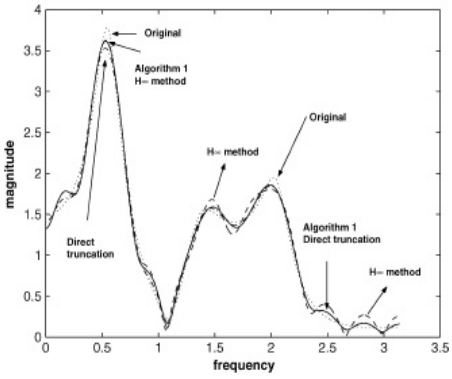


(a) FIR appeoximations

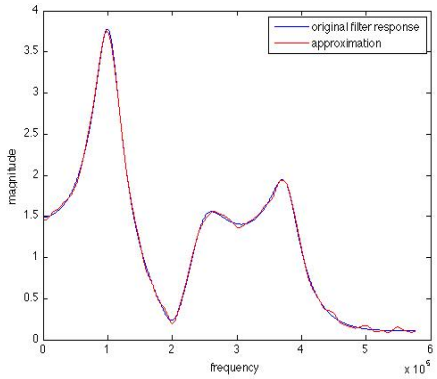


(b) Prony-based polyphase decompostion

Figure 8.1: Approximations with a hardware complexity of 12 MACS. Approximation quality of methods shown in [11] (left), the approximation quality of Prony-based polyphase decomposition method (right).

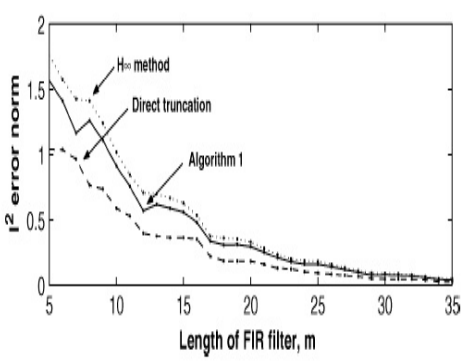


(a) FIR approximations

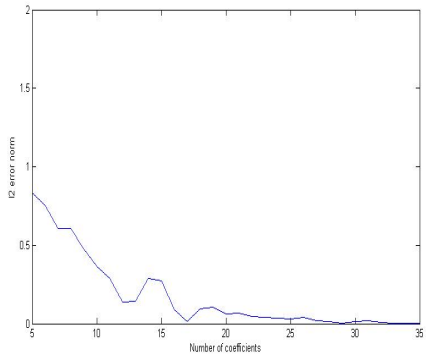


(b) Prony-based polyphase decompostion method

Figure 8.2: Approximations with a hardware complexity of 24 MACS. Approximation quality of methods shown in [11] (left), the approximation quality of Prony-based polyphase decomposition method (right).



(a) FIR approximations



(b) Prony-based polyphase decompostion method

Figure 8.3: Plot of l^2 error at different hardware complexities.

CHAPTER 9

CONCLUSION

We have proposed a method that optimizes the IIR filter coefficients under restrictions such that the resulting filter is pipelined, is stable, and at the same time reduces complexity dramatically. The tradeoff in this process is that we lose the accuracy in approximation for hardware complexity. This method works well for $P >> N$, where P is the depth in pipelining and N is the filter order. More importantly, for any filter order, if the pipelining level is large enough, it can be decomposed into P (with only one coefficient, order P) filters by using our method. There is immense room for approximation techniques in order to reduce the complexity of circuits.

APPENDIX

SIMULATION RESULTS

The results of various simulations that were performed are shown in Figures A.1 - A.11. The filters used in the following simulations are all Butterworth filters with a passband of 0.2π. Note that similar results were observed with other kinds of filters, such as Chebychev.

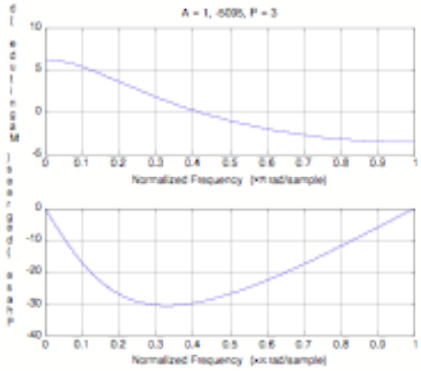
Case 1: A 1st order Butterworth filter

a_k = [1 -0.5095], Pipelining Factor P = 2,

Filter 1 : a_k = [1 0 -0.2596]

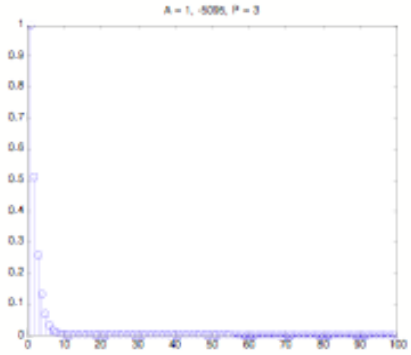
Filter 2 : a_k = [1 0 -0.2596]

The simulation results are shown in Figure A.1

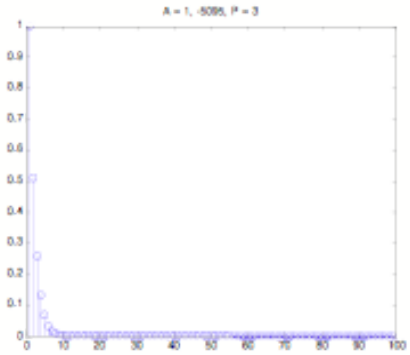


:

(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.1: Comparison between original and pipelined filter frequency response and impulse response.

Case 2: A 1st order Butterworth filter

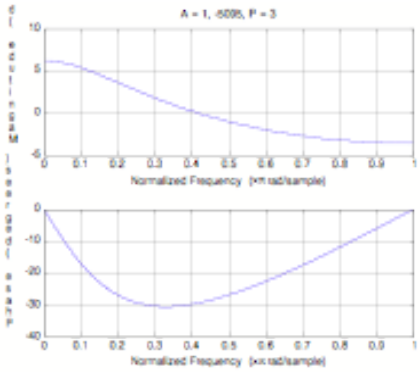
a_{k} = [1 -0.5095], Pipelining Factor P = 3

Filter 1 : a_k = [1 0 0 -0.1323]

Filter 2 : a_k = [1 0 0 -0.1323]

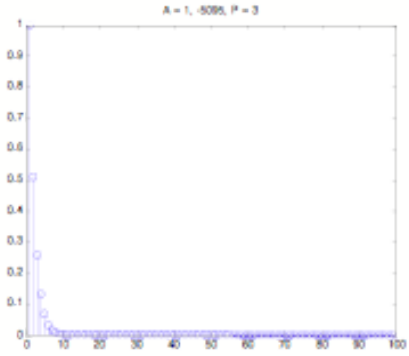
Filter 3 : a_k = [1 0 0 -0.1323]

The simulation results are shown in Figure A.2

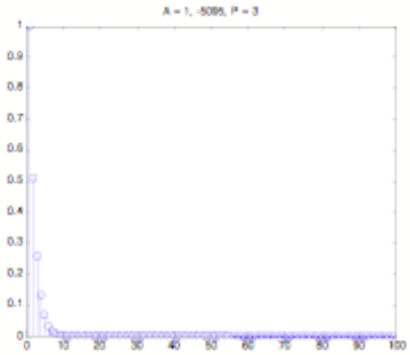


:

(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.2: Comparison between original and pipelined filter frequency response and impulse response.

Case 3: A 2nd order Butterworth filter

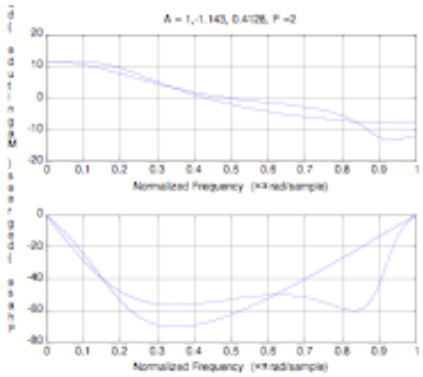
a_{k} = [1 -1.143 0.4128], Pipelining Factor P = 2

Filter 1 : a_k = [1 0 -0.6132 0 0.1177]

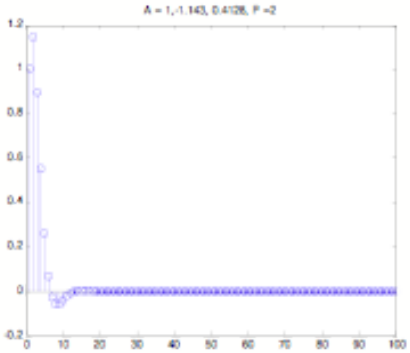
Filter 2 : a_k = [1 0 -0.4126 0 0.0682]

The simulation results are shown in Figure A.3

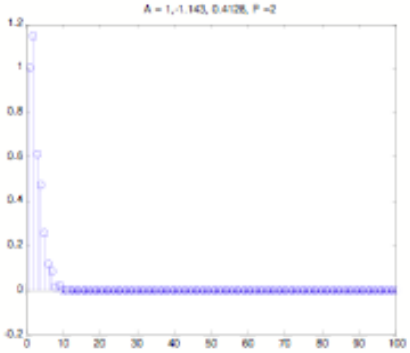
:



(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.3: Comparison between original and pipelined filter frequency response and impulse response.

Case 4: A 2nd order Butterworth filter

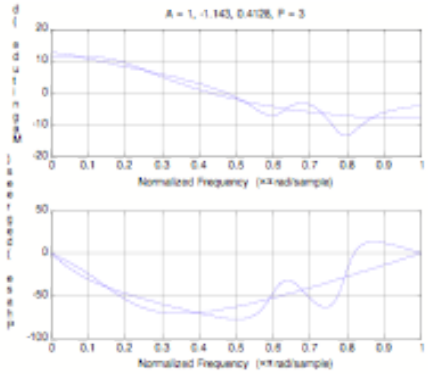
a_{k} = [1 -1.143 0.4128], Pipelining Factor P = 3

Filter 1 : a_k = [1 0 0 -0.4110 0 0 -0.0133]

Filter 2 : a_k = [1 0 0 -0.2048 0 0 -0.0062]

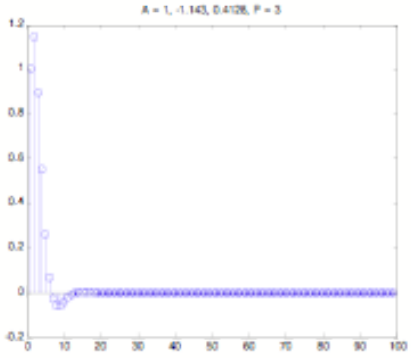
Filter 3 : a_k = [1 0 0 -0.0727 0 0 -0.0045]

The simulation results are shown in Figure A.4

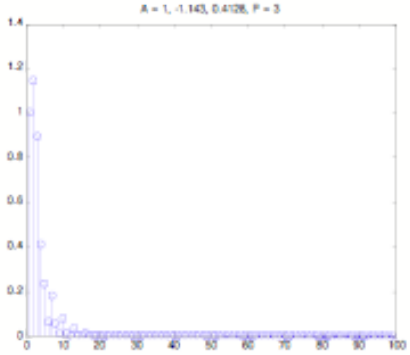


:

(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.4: Comparison between original and pipelined filter frequency response and impulse response.

Case 5: A 2nd order Butterworth filter

a_k = [1 -1.143 0.4128], Pipelining Factor P = 4

Filter 1 : a_k = [1 0 0 0 -0.4110 0 0 0 -0.0133]

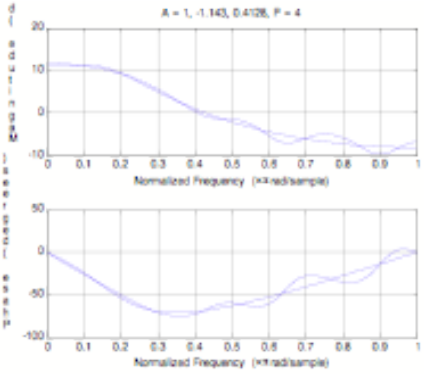
Filter 2 : a_k = [1 0 0 0 -0.2048 0 0 0 -0.0062]

Filter 3 : a_k = [1 0 0 0 -0.0727 0 0 0 -0.0045]

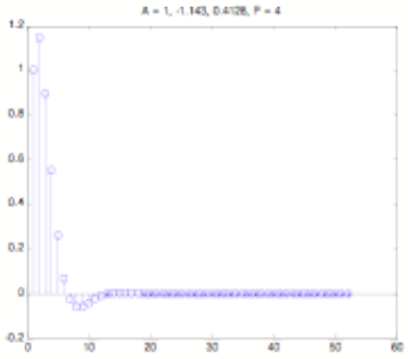
Filter 4 : a_k = [1 0 0 0 -0.0727 0 0 0 -0.0045]

The simulation results are shown in Figure A.5

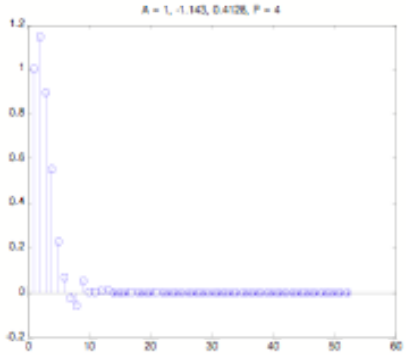
:



(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.5: Comparison between original and pipelined filter frequency response and impulse response.

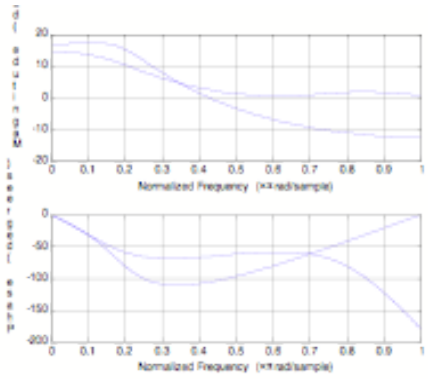
Case 6:A 3rd order Butterworth filter

a_{k} = [1 -1.76 1.1829 -0.2781], Pipelining Factor P = 2

Filter 1 : a_k = [1 0 -0.6911 0 0.1984]

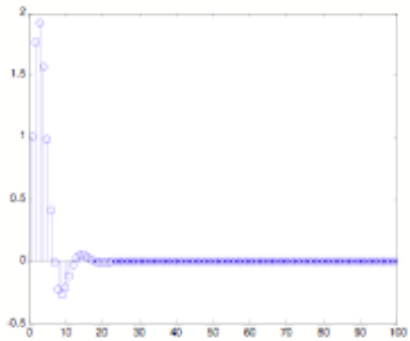
Filter 2 : a_k = [1 0 -0.5844 0 0.1584]

The simulation results are shown in Figure A.6

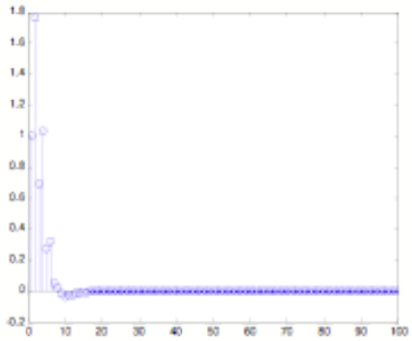


:

(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.6: Comparison between original and pipelined filter frequency response and impulse response.

Case 7: A 3rd order Butterworth filter

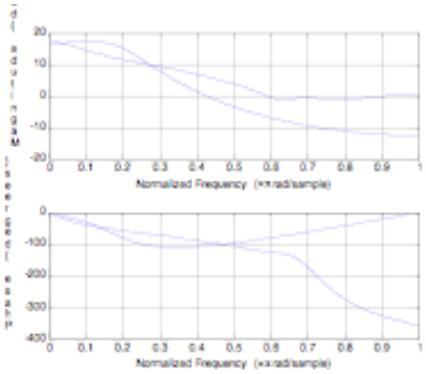
a_{k} = [1 -1.76 1.1829 -0.2781], Pipelining Factor P = 3

Filter 1 : a_k = [1 0 0 -0.4444 0 0 -0.0445]

Filter 2 : a_k = [1 0 0 -0.3716 0 0 -0.0401]

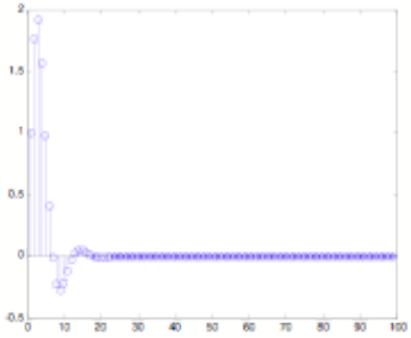
Filter 3 : a_k = [1 0 0 -0.1729 0 0 -0.0261]

The simulation results are shown in Figure A.7

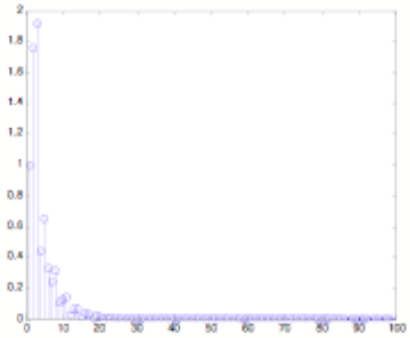


:

(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.7: Comparison between original and pipelined filter frequency response and impulse response.

Case 8: A 3rd order Butterworth filter

a_k = [1 -1.76 1.1829 -0.2781], Pipelining Factor P = 4

Filter 1 : a_k = [1 0 0 0 -0.3475 0 0 0 -0.0043]

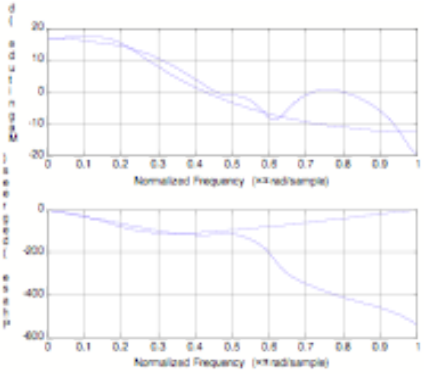
Filter 2 : a_k = [1 0 0 0 -0.1865 0 0 0 -0.0022]

Filter 3 : a_k = [1 0 0 0 0.0063 0 0 0 -0.0008]

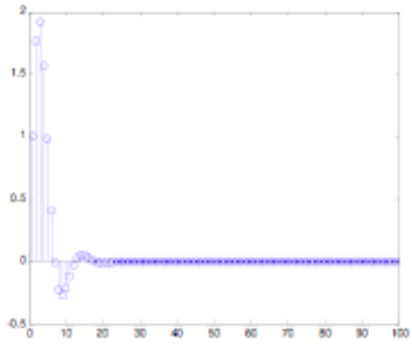
Filter 4 : a_k = [1 0 0 0 0.1383 0 0 0 -0.0005]

The simulation results are shown in Figure A.8

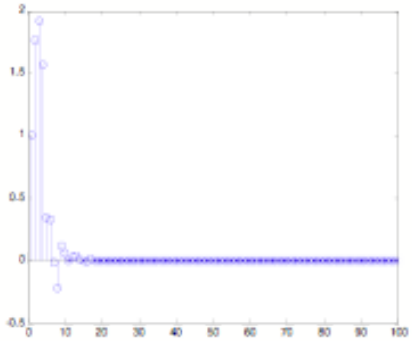
:



(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.8: Comparison between original and pipelined filter frequency response and impulse response.

Case 9: A 3rd order Butterworth filter

a_{k} = [1 -1.76 1.1829 -0.2781], Pipelining Factor P = 5

Filter 1 : a_k = [1 0 0 0 0 -0.2999 0 0 0 0 0.0006]

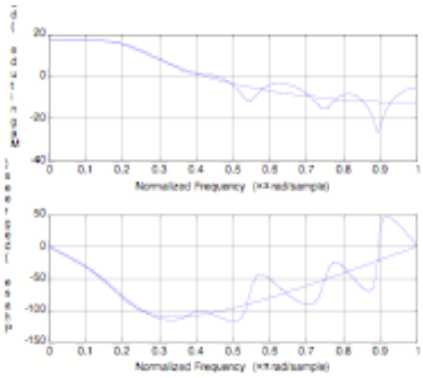
Filter 2 : a_k = [1 0 0 0 0 0.0055 0 0 0 0 0.0002]

Filter 3 : a_k = [1 0 0 0 0 0.1174 0 0 0 0 0.0001]

Filter 4 : a_k = [1 0 0 0 0 0.1736 0 0 0 0 0.0000]

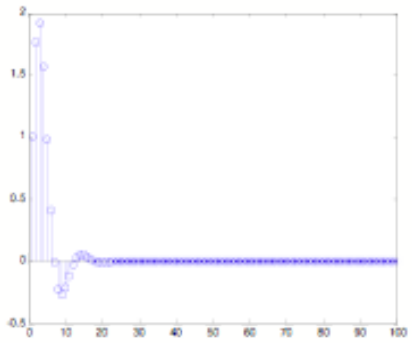
Filter 5 : a_k = [1 0 0 0 0 0.1383 0 0 0 0 0.0000]

The simulation results are shown in Figure A.9

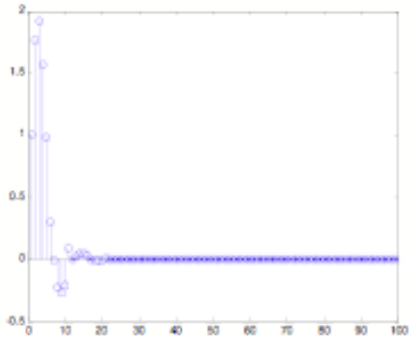


:

(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.9: Comparison between original and pipelined filter frequency response and impulse response.

Case 10: A 4th order Butterworth filter

a_k = [1 -2.3695 2.3140 -1.0547 0.1874], Pipelining Factor P = 6

Filter 1 : a_k = [1 0 0 0 0 0 -0.4140 0 0 0 0 0 0.0007]

Filter 2 : a_k = [1 0 0 0 0 0 0.1387 0 0 0 0 0 0.0000]

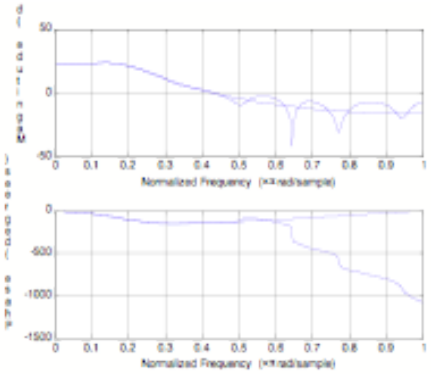
Filter 3 : a_k = [1 0 0 0 0 0 0.2192 0 0 0 0 0 0.0000]

Filter 4 : a_k = [1 0 0 0 0 0 0.2176 0 0 0 0 0 0.0000]

Filter 5 : a_k = [1 0 0 0 0 0 0.1915 0 0 0 0 0 0.0000]

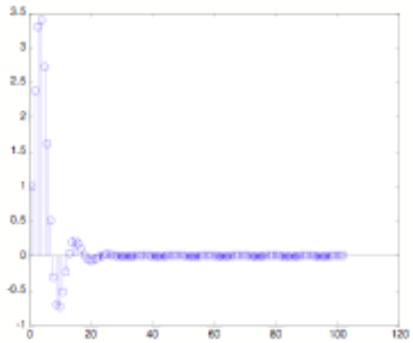
Filter 6 : a_k = [1 0 0 0 0 0 0.1370 0 0 0 0 0 0.0000]

The simulation results are shown in Figure A.10

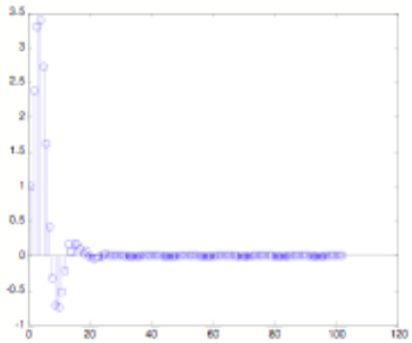


:

(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.10: Comparison between original and pipelined filter frequency response and impulse response.

Case 11: A 5th order Butterworth filter

a_k = [1 -2.9754 3.8060 -2.5453 0.8811 -0.1254], Pipelining Factor P = 7

Filter 1 : a_k = [1 0 0 0 0 0 0 -0.3020 0 0 0 0 0 0 -0.0003]

Filter 2 : a_k = [1 0 0 0 0 0 0 0.3935 0 0 0 0 0 0 0.0000]

Filter 3 : a_k = [1 0 0 0 0 0 0 0.3398 0 0 0 0 0 0 0.0000]

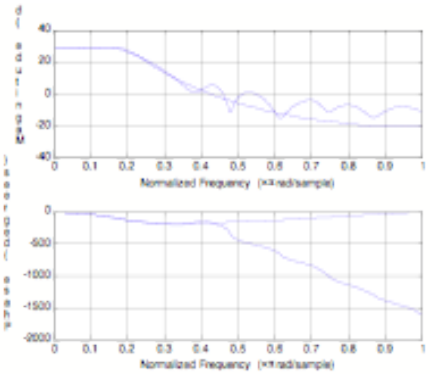
Filter 4 : a_k = [1 0 0 0 0 0 0 0.2555 0 0 0 0 0 0 0.0000]

Filter 5 : a_k = [1 0 0 0 0 0 0 0.1674 0 0 0 0 0 0 0.0000]

Filter 6 : a_k = [1 0 0 0 0 0 0 0.0629 0 0 0 0 0 0 0.0000]

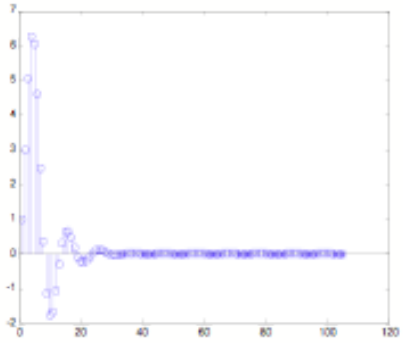
Filter 7 : a_k = [1 0 0 0 0 0 0 -0.1088 0 0 0 0 0 0 0.0001]

The simulation results are shown in Figure A.11

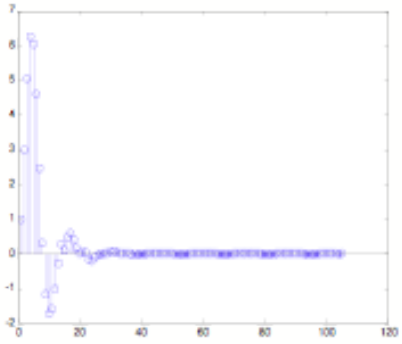


:

(a) Original and pipelined frequency response comparison



(b) Original impulse response



(c) Pipelined impulse response

Figure A.11: Comparison between original and pipelined filter frequency response and impulse response.

REFERENCES

[1] K. Parhi and D. Messerschmitt, "A bit-parallel bit level recursive filter architecture," in *IEEE International Conference on Computer Design*, 1986.

[2] K. Parhi and D. Messerschmitt, "Look-ahead computation: Improving iteration bound in linear recursions," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 12, April 1987. pp. 1855-1858.

[3] K. Pahi and D. Messerschmitt, "Area-efficient high speed VLSl adaptive filter architectures," in *IEEE International Conference on Communications*, 1987.

[4] K. Parhi and D. Messerschmitt, "Concurrent cellular VLSl adaptive filter architectures," *IEEE Transactions on Circuits and Systems*, vol. 34, no. 10, pp. 1141-1151, October 1987.

[5] G. R. Gao, "Maximum pipelining linear recurrence on static data flow computers," *International Journal on Parallel Programming*, vol. 15, no. 2, pp. 127 - 149, 1986.

[6] H. H. Loomis and B. Sinha, "High speed recursive digital filter realization," *Circuits Systems, Signal Processing*, vol. 3 , no. 3, pp. 267- 294, 1984.

[7] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters-Part 1: Pipelining using scattered look-ahead and decomposition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol 37, no. 7, pp. 1099-1117, 1989.

[8] Jun Ma and K. K. Parhi, "Pipelined CORDIC-based cascade orthogonal IIR digital filters," *IEEE Transactions on Circuits and Systems*, vol. 47, no. 7, pp. 2102-2119, 2004.

[9] A. Shaw and M. I. Ahmed, "Pipelined recursive digital filters: A general look-ahead scheme and optimal approximation," *IEEE Transactions on Circuits and Systems*, vol. 46, no. 11, pp. 1415-1420, 1999.

[10] A. K. Shaw, "Optimal identification of discrete-time systems from impulse response data," *IEEE Transactions on Signal Processing*, vol. 42, no. 1, pp. 113120, 1994.

[11] L. Chai, J. Zhang, C. Zhang and E. Mosca, "Hankel-norm approximation of IIR by FIR models: A constructive method," *IEEE Transactions on Circuits and Systems*, vol. 55, no. 2, pp. 586-598, 2008.