POWER ANALYSIS AND LOW POWER REALIZATION OF DIGITAL FILTER STRUCTURES

by

Okan Zafer Batur

B.S., in Electrical and Electronics Engineering, Doğu Akdeniz University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2006

POWER ANALYSIS AND LOW POWER REALIZATION
OF
DIGITAL FILTER STRUCTURES

APPROVED BY:

Prof. Günhan Dündar .....
(Thesis Supervisor)

Prof. Ömer Cerid .....

Assoc. Prof. Arda Yurdakul .....

DATE OF APPROVAL: .....

ACKNOWLEDGEMENTS

I am very grateful to my thesis supervisor Prof. Günhan Dündar for giving me much support and guidance during my education and during my thesis. I would like to mention his patience, giving me inspiration and hope when I was stuck at dead-ends.

I would like to thank to Prof. Ömer Cerid and Assoc. Prof. Arda Yurdakul for interesting and evaluating my thesis and being a jury member. Many thanks to Mustafa Aktan for giving support during my thesis.

Finally, I would like to thank to my family and to all my friends because of their support in all my life and for becoming what I am.

ABSTRACT

POWER ANALYSIS AND LOW POWER REALIZATION OF DIGITAL FILTER STRUCTURES

Digital Filters are important for Digital Signal Processing (DSP) systems. They are widely used in image, speech processing, data transmission. Digital Filters can be used to reduce noise in the system, get information from the signal, etc. However, these filters consume high power even if they are made in a full custom fashion. The reason is the multiplications or divisions in the filters.

The filters include large number of multipliers. However, if constant coefficients are used in the filter, multiplier operations can be represented by adders, subtractors and shift operations. By using Canonic Signed Digit (CSD) representation and making subexpression sharing on the coefficients, the number of adders in the multiplier which leads to less area and less power consumption.

In this thesis, Finite Impulse Response (FIR) filter structures are studied. A different subexpression sharing algorithm is created to select appropriate subexpressions. Subexpressions are selected such a way that the adder depth is minimized. Minimizing adder depth caused few more adders compared to the other subexpression sharing algorithms. However, minimizing the adder depth minimizes the delay and glitches. In the algorithm, the minimum length subexpressions are selected first. Therefore, the full adders (FAs) used in the adders are reduced.

ÖZET

DİJİTAL FİLTRE YAPILARININ GÜÇ ANALİZİ VE DÜŞÜK GÜÇ TÜKETİMİ ELDE EDİLMESİ

Sayısal süzgeçler, Sayısal Sinyal İşleme uygulamaları içinde önemli bir yer tutmaktadır. İmge işleme, ses işleme, bilgi aktarımı vb. konularda geniş bir kullanım alanı vardır. Sayısal süzgeçler sistemdeki parazitleri azaltmak, sinyallerden bilgi etmek gibi alanlarda kullanılır. Bu süzgeçler tamamen optimize edilip yapılsalar dahi yüksek güç tüketirler ve fazla yer kaplarlar. Bunun nedeni süzgeçlerde kullanılan çarpma ve bölme mimarileridir.

Bu süzgeçler fazla miktarda çarpma işlemi içerir. Ama eğer sabit katsayı kullanılırsa çarpma işlemi toplama, çıkarma ve kaydırma işlemleriyle ifade edilebilir. Katsayılarda işaretlenmiş sayı gösterimi ve alt ifade paylaşımı kullanarak çarpmadaki toplama işlemlerinin sayısını azaltabiliriz. Buda daha az güç tüketimi olmasını sağlar.

Tezde, Sonlu Dürtü Yanıtlı süzgeç mimarisi üzerinde çalışıldı. Değişik bir alt ifade paylasım algoritması geliştirildi. Alt ifadeler seçilirken toplama derinliğinin en azda tutulması sağlandı. Toplama derinliğini en düşük seviyede tutmak diğer alt ifade algoritmaları ile karşılaştırıldığında fazladan birkaç tane toplama işlemine neden oldu ama toplama derinliğini düşük tutmak gecikmeyi ve istenmeyen geçişleri düşürdü. Algoritmada en az uzunlukdaki alt ifadelerin seçimine öncelik verildi. Bu da toplama işlemlerinin içinde kullanılan tam toplayıcıların sayılarını düşürdü.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES vii
LIST OF TABLES viii
LIST OF SYMBOLS/ABBREVIATIONS xi
1. INTRODUCTION 1
2. BACKGROUND OF THE PROBLEM AND DESIGN METHODOLOGY . . . 4
2.1. Background of the Problem 4
2.2. Methodology. 6
3. SUBEXPRESSION SHARING ALGORITHM 8
3.1. Selection of Subexpressions 8
3.2. Adder Depth 10
3.3. Adder Length 11
3.4. Minimum Full Adders. 12
3.5. N01 and 10N Expressions 13
4. GLITCH 15
5. SOFTWARE 17
5.1 Algorithm Code 17
5.2 Vhd Code Creator 20
6. RESULTS 23
7. CONCLUSION & FUTURE WORK 26
REFERENCES 27
REFERENCES NOT CITED 28

LIST OF FIGURES

FIGURE 2.1 FIR Filter Diagram 5

FIGURE 3.1 Subexpression Sharing Algorithm Example 9

FIGURE 3.2 Subexpression Sharing Algorithm Example (cont.) 10

FIGURE 3.3 Adder Length Calculation 11

FIGURE 3.4 Subexpression Selection Difference: (a)101 selected (b)100001 selected 12

FIGURE 3.5 Subexpression 10000-1 Example 13

FIGURE 3.6 Sunexpression -100001 Example 14

FIGURE 5.1 Example Output of the Algorithm Code 16

FIGURE 5.2 Example Coefficient File 17

FIGURE 5.3 Algorithm Code Flowchart 18

FIGURE 5.4 Vhd Code Creator Flowchart 20

LIST OF TABLES

TABLE 3.1 Subexpression Dictionary 9

TABLE 3.2 Comparision of full adders 13

TABLE 6.1 Component Comparisions of the Algorithms for 120 tap filter 23

TABLE 6.2 Component Comparisions of the Algorithms for 124 tap filter 23

TABLE 6.3 Power comparisions of algorithms for 120 tap filter 24

TABLE 6.4 Power comparisions of algorithms for 124 tap filter 24

LIST OF SYMBOLS/ABBREVIATIONS

g0,g1,..	number of nonzero terms between two nonzero terms.
h0,h1,...	added spaces.
a0,a1,...	First adder input in ripple carry adders
b0,b1,...	Second adder input in ripple carry adders
clen	length of coefficient.
s1	shift of the input first vector.
s*2*	shift of the inputvector 2.
v1	value of the first input vector.
v2	value of the secont input vector.
xin	input vector.
zd	zero delay
ed	enable delay
a	subexpression
o	filter output
d	delay expression
t	filter taps
z^{-1}	delay element
k	shifts
x(n-k)	input vector shifted by k
y(n)	output of the filter
a(k)	coefficient of the filter
a_i	represents twos complement bits.
b_i	rrepresents bits in CSD form
N	represents '-1'

CSD	Canonic Signed Digit
MSD	Minimum Signed Digit
DSP	Digital Signal Processing
HA	Half Adder

FA	Full Adder
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
FFT	Fast Fourier Transform
RCA	Ripple Carry Adder

1. INTRODUCTION

Electronic devices and instruments are widely used in our daily life. Personal computers, music players, handphones, all other electronic devices and even the electric bulbs use electrical power to operate.

An ordinary electric bulb uses 40 watts to 100 watts of power to operate. The variation is with respect to the light needed and this power is retrieved from the power line that comes into the building. However, portable electronic devices like handphones need batteries to operate. The battery power changes with respect to the type of the phones. If the phone has extra futures, then more power is needed. And if the phone is to be small in size, then battery must be smaller, which means its power is reduced. Electronic circuits inside the device must be designed to be area and power efficient. They have to be small and consume low power.

High power consumption also brings heat. If the power of the unit area is high, a cooling problem is encounted. For example, in computers, the Central Processor Unit (CPU) needs a huge cooler with fan to lower the temperature to within operation bounds.

Electronic world goes in two parallel directions; analog and digital systems. Both may exist in the same platform or separately. Digital Signal Processing (DSP) has an increasing use in many key areas of technology; telecommunication, digital television and media, biomedicine, digital audio and instrumentation. DSP is now a core subject in most electronic, computer, communication areas [4].

Digital filters play a great role in DSP systems. A filter can be thought of as a system that changes the waveshape, amplitude, frequency, phase of the input signal. The common reasons on filtering are to improve quality of the signal, to extract information from the signals or to separate two or more signals that have been already combined before [4].

In the digital filter, a mathematical algorithm is to be implemented in hardware that takes an incoming signal, operates on it and produces an output signal that matches the filter objectives.

There are many filter structures in the literature. Some digital filter structures are Finite Impulse Response (FIR), Infinite Impulse Response (IIR), Lattice, Multirate, Turbocoders, Viterbi, FFT, Cascade.

Digital filters can be broadly divided into two classes as FIR and IIR. The choice between FIR and IIR depends on the application and relative advantages. For example, when sharp edges are required, one can use an IIR filter. However, FIR filters have exactly linear phase responses. The details are not the subject of this thesis. However, while choosing the structure, power should be taken into account. If both filters can do the same job, one with less power consumption must be chosen.

In the thesis, Finite Impulse Response (FIR) filters are studied. Whatever the structure is, hardware reduction will reduce the power. There are many methodologies for reducing the power consumption. Representing the filter coefficients in a diffrerent number representation can reduce hardware complexity and power dramatically.

In this thesis, Canonic Signed Digit (CSD) representaion is used. CSD represents the given number in the sum powers of two. Some properties of CSD are; no two nonzero terms can stand next to each other and there is an extra number -1. This method guarantees that the number of nonzero terms in the coefficient can not be greater than half of the coeficient length. Research shows that there is 33% of decrease in power with CSD number representation.

Power consumption can be lowered further by applying subexpression sharing algorithms to the CSD represented coefficients. In this thesis, a different subexpression sharing algorithm is presented, which concentrates in minimizing adder depth and lowering the number of Full Adders (FA) used in the multiplier.

There are also glitches to be accounted for, since sigificant power is dissipated due to glitches. Glitches are caused because of unwanted changes at the outputs of gates. Since these outputs are connected to others, reaching the steady state takes time and unwanted signal changes occur. This transition period makes filters suffer from glitches. To reduce glitches, the adder depth must be minimized to have small adder trees and critical paths must be redesigned. In this thesis, glitches are are dealth with changing the selection of the subexpressions. However, changing the selection of subexpressions after finding them brings new subexpressions and this leads to more adders. Therefore, power dissipation increases. However, in some cases where no new subexpressions are produced this method works.

Organization of the chapters are as follows; Chapter 2 is the background of the problem and methodology. It describes the problem; why we need to reduce the power and it explains what has been done throughout the thesis. In Chapter 3, the subexpresion sharing algorithm is analysed and explained in detail. The methods used and key points are explained. In Chapter 4, glitch work is explained; how it is found and what is done. In Chapter 5, the developed software is explained in detail with flowcharts. In Chapter 6, the results from the simulations and analysis are given and discussed. In Chapter 7, conclusions and future can be found.

2. BACKGROUND OF THE PROBLEM AND DESIGN METHODOLOGY

2.1 Background of the Problem

In digital filters, constraints are small area and low power consumption. In the thesis, FIR filter structures are studied. Constant coefficient FIR filters are used for simulations. An FIR filter can be described by the following equation.

$$y(n) = \sum_{k=0}^{N-1} a(k)x(n-k) \tag{2.1}$$

It is obvious from the above equation that FIR filter has a finite duration, since a(k) for the FIR has only N values. y(n) is the output, x(n-k) is the input where k represents the shifts and a(k) is the corresponding coefficient of the filter. The main problem is in these coefficients and power minimization is performed on these coeficients. They are defined in two's complement form as follows.

$$\sum_{i=0}^{N-1} a_i 2^{-i} \tag{2.2}$$

where $a_i = 0$ or 1 for $i > 0$ and $a_0 = 0$ or -1. This is the the two's complement form. However, the CSD is used to reduce the number of nonzero terms. As a short definition, $b_0.b_1b_2 \ldots b_{N-1}$ is said to be in CSD format, if each b_i is equal to 0, +1 or -1 and two consecutive b_i are nonzero. In writing them, -1 can be denoted by N [1].

Figure 2.1 gives the FIR filter diagram used in the thesis. Equation 2.1 can be observed in this figure. The small right handed triangles represent the multiplication, z^{-1} is the delay element, and circles with '+' sign are adders.

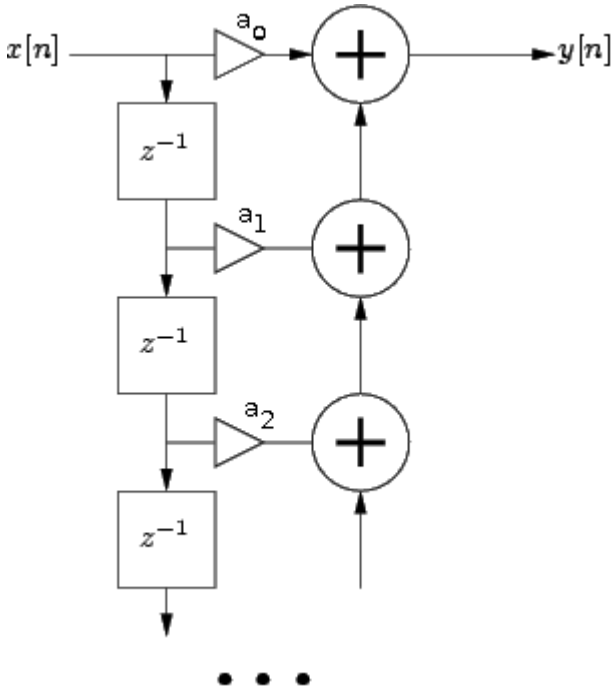


Figure 2.1. FIR Filter Diagram

In our case, coefficients of the filter are constant. In constant coefficient filters, multiplications can be expressed as additions and shift elements. The shifts can be made hardwired such that no components like registers are needed to handle the shifts. Thus they come at no cost to the system.

Realization of multiplication operations with adders and shifts brings the idea of finding common subexpressions for sharing. Therefore, subexpression sharing must be increased to lower the number of adders hence the power. Subexpression sharing can be utilized in FIR filters with up to 50% of the total number of operators. Even for sharing only two most common terms, reductions up to 33% [1].

To benefit from subexpression sharing, CSD number representation should be used. The first investigation of CSD multiplication was done in [3] with showing 33% saving with respect to binary coefficient representation.

In the literature, Minimum Signed Digit (MSD) representation can be found. The MSD number representation is obtained from CSD. MSD differs from CSD only in positioning of the nonzero terms. In MSD, two nonzero terms can come next to each other.

However, this brings subexpression sharing problem. MSD has an advantage over power consumption in filters which have high number of taps. However, for subexpression sharing the CSD form is more suitable. For minimazation of full adders MSD can be more suitable but since the subexpression sharing is the main purpose, CSD is preferred here.

Throughout the literature survey, different subexpression sharing algorithms and different methods are found. Hartley's algorithm [1] and Yurdakul's algorithm [2] are used for comparison with the proposed algorithm in the next chapter.

In both algorithms, it has been seen that adder depth is not dealth with at all. The main concentration in both cases and many others in the literature was sharing as much as possible. It has been seen that Yurdakul's algorithm saves the most expressions compared to Hartleys'.

Glitches are also very important in power saving since almost 50% of the power goes to the glitching in the gates. Glitches occur when there is transition from 0 to 1 or 1 to 0. While the system stabilizes, glitches occur and power is lost. In the following chapters, this will be explained.

2.2 Methodology

In this thesis, lowering the power dissipation was the main design constraint and eveything done here is for reducing the power disipation. Power anlysis is performed to see the results. Literature survey on number systems and subexpression sharing algorithms are done. Then, thesis went through increasing the subexpression sharing and minimizing the adder depth together with the FAs.

To minimize the adder depth, greedy style subexpression selection is proposed where zero terms between each two nonzero terms are counted and obtained results are added with their neighbours to obtain a weight system. Selections are done by finding the minimum length subexpressions. It has been seen and proved that selecting minimum

length subexpressions incresed the subexpression sharing and leads to less number of FAs. Another thing found in lowering the FAs used was selecting or using N01 terms instead of 10N. If any 10N term is encountered, it is reversed to the N01. This way, there is saving in the number of FAs. This saving corresponds exactly to the length of the subexpresion.

For glitch power reduction, various inputs are supplied to localize most common transitions. Then, in that branch subexpression selection is changed to reduce glitch power. Glitch power discussed in the chapter 4. The following chapter gives detailed information about the proposed algorithm.

3. SUBEXPRESSION SHARING ALGORITHM

The algorithm is simple. It counts spaces between nonzero terms for selection of subexpressions. The main purpose of the algorithm is to find minimum length subexpressions and to keep adder length minimum. The algorithm and how the selection is done are explained in the next topic.

3.1 Selection of Subexpressions

As explained before, adder depth and number of full adders are minimized in this algorithm. The subexpression sharing selection algorithm can be described as follows:

Step1: Count the number of zeros between each nonzero term. Call them g0,g1,g2,. . .

Step2: Add the spaces found in step 1 with coresponding neighbours as follows: h0=g0+g1, h1=g0+g1+g2, h2=g1+g2+g3, h3=g2+g3+g4 . . .

Step3: Find the minimum result in step 2, and if there is only one minimum point take the corresponding subexpression. Put zeros to the subexpression location for rerun. Then advance to step 5.

Step4: Since there is more than one minimum point, compare step 1 for results of the minimum points. Take the minimum one as the subexpression. If there is still an equality, any one can be taken. Put zeros to the subexpression location for rerun. Advance to step 5.

Step5: If the subexpression found is not discovered perivously add new subexpression to the subexpression dictionary as done in Table 3.1. Then goto the step1 to run process again until no subexpressions exist.

Figure 3.1 shows an example of proposed subexpression selection algorithm.

Table 3.1. Subexpression Dictionary

Subexpression Found	Exists in Dictionary?	Subexpression Dictionary
N01	no	2
N01	yes	2
1001	no	3
2002	no	4
4003	no	5

The table is filled in by using the example in Figure 3.1. As can be seen in Figure 3.1, the spaces described in step 2 aove are added and minimum of 3 is found which correspond to the first subexpression N01. Then, the subexpression is called 2 and added to the Table 3.1. After that, it is erased to find another subexpression. In this order, all subexpressions can be found. As can be seen from the figure below, the minimum length subexpressions are always selected first.

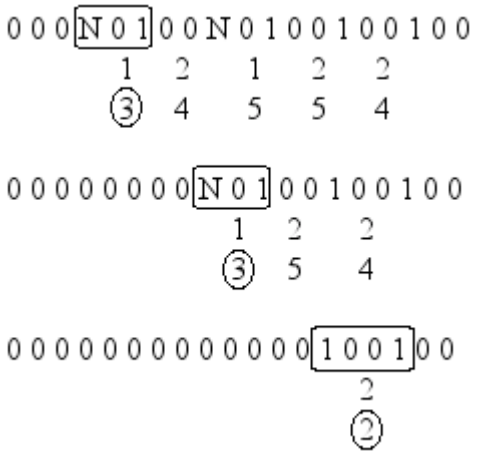


Figure 3.1. Subexpression Sharing Algorithm Example

Since selection is done by pairs and selected subexpression is erased, minimum depth is achived. Algorithm finds a subexpression and looks at the dictionary for

discovered subexpressions. If it is found before, the dictionary value will be used. Otherwise the new values will be added to the subexpression dictionary. Figure 3.2 depicts the final phases of finding subexpressions.

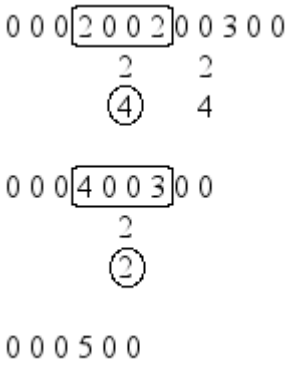


Figure 3.2. Subexpression Sharing Algorithm Example (continued)

3.2 Adder Depth

It was mentioned above that adder depth is lowered. The adder length of each coefficient can be found as follows;

$$\text{Adder Depth} = \text{ceiling}\ (\ \log2(\#\text{nonzero terms})\ ) \qquad (3.1)$$

Therefore if there are 6 nonzero terms in a coefficient like in the example in Figure 3.1, the adder depth becomes 3 from equation 3.1. According to the equation 3.1, maximum depth possibble for a coefficient is;

$$\text{Max Adder Depth} = \text{ceiling}(\ \log2(\text{clen}/2 + 1)\ ) \qquad (3.2)$$

Maximum depth in a filter can be found by finding max number of nonzero terms in the coefficient's adder depth.

3.3 Adder Length

Adder length can be calculated by looking at the input vector length. There is an algorithmic calculation of the adder lengths as described in Figure 3.3. The calculation is as follows.

$$\text{Adder length} = \text{length(xin)} + \text{ceil}(\log(\text{abs}|v1*2^{s1}+v2*2^{s2}|)) \quad (3.3)$$

where v1 and v2 is to be 1 for main input and to be calculated for the others. In Figure 3.3 input bit vector length is 18 bit. That means length(xin)=18 must be put into Equation 3.3.

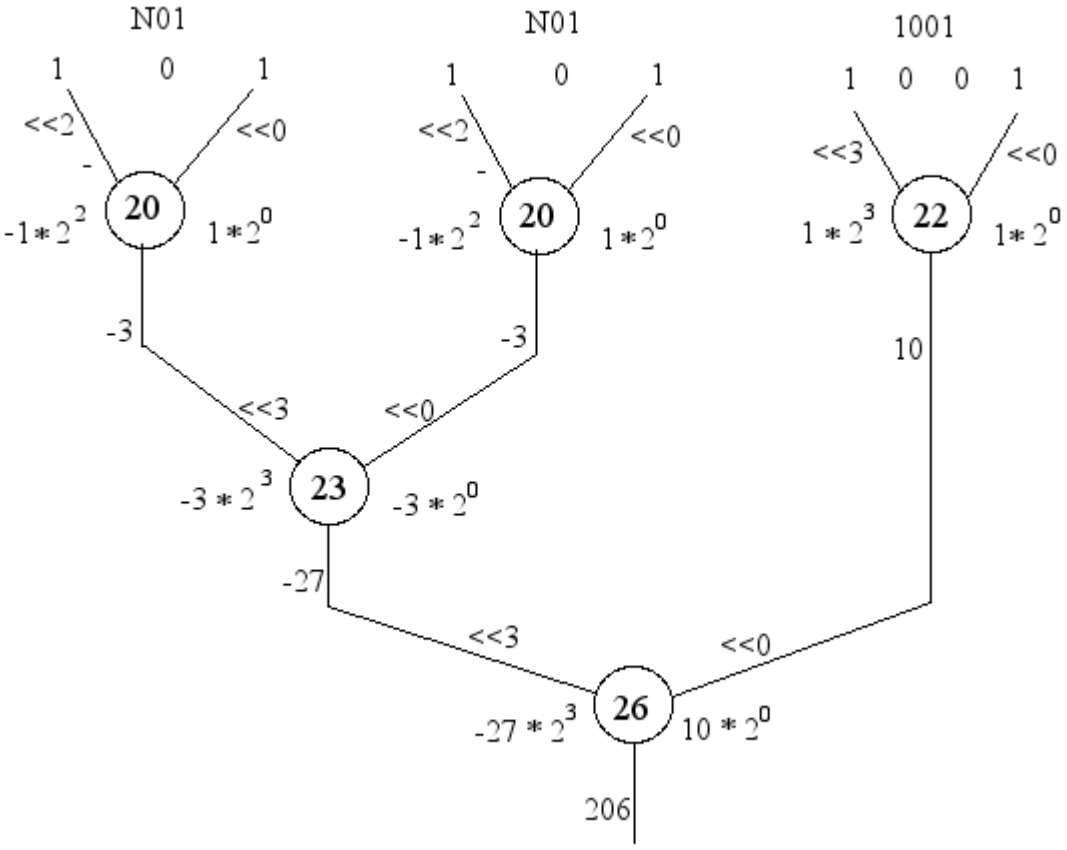


Figure 3.3. Adder Length Calculation

3.3 Minimum Full Adders

Minimum number of full adders is achieved by taking minimum length subexpressions first as shown on the following example. If the coefficient is 00010000101, taking first 10001 or 101 seems to be equivalent at first sight. However there is a difference as depicted in Figure 3.4 where the adder lengths are found.

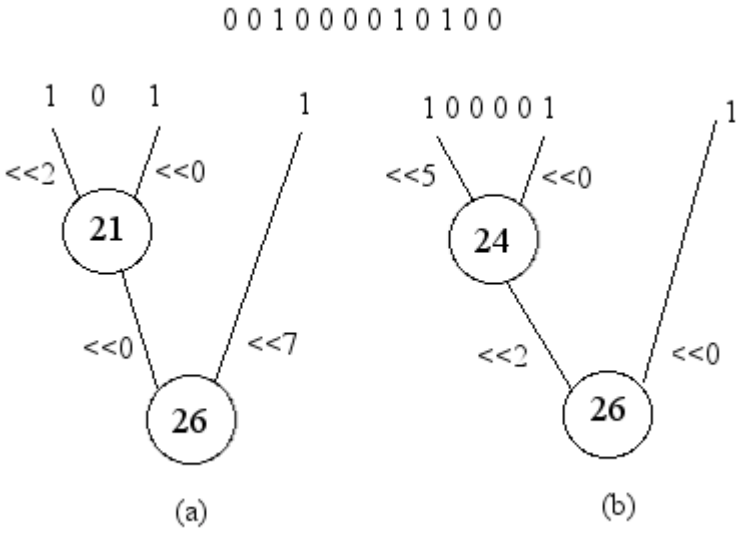


Figure 3.4. Subexpression Selection Difference: (a)101 selected (b)100001 selected

In Figure 3.4(a) the first 101 term is selected . Therefore, the adder length is 21 but in Figure3.4(b) 100001 term is selected. This caused adder length to be 24. Although in the first step, adder length of Figure 3.4(b) is higher, the number of FAs is equal at both cases at this step as can be seen in Table 3.2 . However, when the second step comes, Figure 3.4(a) shows better results in terms of both FAs.

Table 3.2. Comparision of full adders

	FIRST ADD	SECOND ADD	TOTAL
FIGURE2.4 (a)	19FA	29FA	38FA
FIGURE2.4 (b)	19FA	25FA	44FA

As can be seen in Table 3.2 above, there are 6 fewer full adders, if minimum length subexpression is selected first. Number of saved FAs is dependent on the coefficient length.

3.4 N01 and 10N Expressions

While selecting subexpressions, one should differentiate between 10...N and N0...1 terms. Because multiplying a number by 10...N or N0...1 is different with respect to the number of adders used in the multiplier. This can be explained in Figures 3.5 & 3.6.

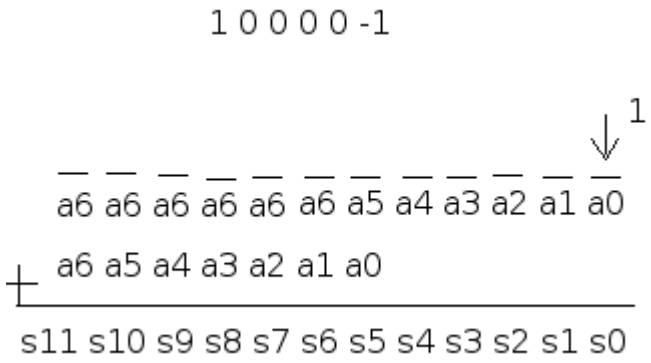


Figure 3.5. Subexpression 10000-1 example

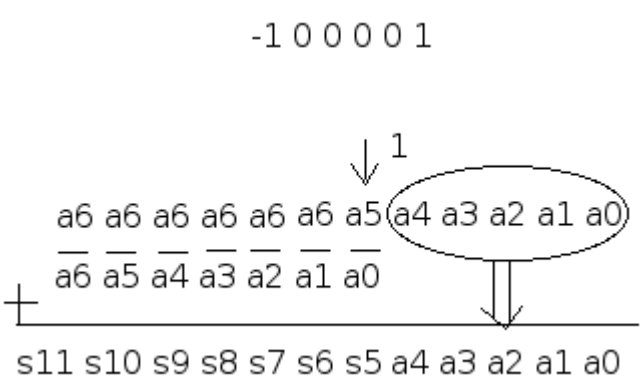


Figure 3.6 Subexpression -100001 example

As can be seen from the Figure 3.6, N0...1 subexpression is more advantageous over the 10...N subexpression. The gain is proportional to the length of the subexpression. In the example, there are 5 shifts. Therefore 5 full adders are gained from this convention. The drawback is there is always a carry of '-' sign in the algorithm. Since the 10...N is reversed as N0...1 the sign of the result is changed to '-' and followed to the next level. This may have a drawback in the following stages like two '-' signs. Therefore, there may be an additional unexpected subexpression.

4. GLITCH

Signal switching in combinational circuits occours for diferent reasons. Input signal transitions take place at different times in a clock cycle. Different logic gates are sensitive to different types of transitions in the input. Also propagation delays of logic gates may differ.

Gate delays are often assumed to be zero to simplify estimation. In this way, an important aspect which is glitch power is always ignored. In a static logic gate, the output or internal nodes can switch before the correct logical value reaches a stable point. For example, consider an AND gate with two inputs of different delays and consider the transition 01 to 10. For a zero delay gate, the output would be stable and would be logic zero. However in the example above, if the first input has a lower delay, the output becomes a temporary logic one. After that, it settles to zero again. The power lost during this unwanted switching activity is called glitching power loss.

Glitches on an internal or output node are dependent on its logic depth. Generally, nodes that are logically deeper are more prone to signal glitches. Therefore in this thesis adder depths are minimized. Another drawback of these signal changes is delay. It will decrease the response time of the filter and filters will operate at lower frequencies. Also, a logically deep node is typically affected by more input switching and therefore more open to the glitches. To reduce glitches, depth of combinational logic can be shortened by adding pipeline registers.

For low power design, glitching should be minimized because it causes power dissipation. It was reported [5] that in a combinational circuit the power dissipaton can be as high as %20 of the total power dissipation, and can be much more in some circuits such as combinational adders.

Glitches are very important in filters, because nearly half of the power dissipated goes to glitches. Reducing glitch can be done by making a depth analysis of the filter. By giving series of inputs to the filter, the adder tree can be analysed and most common

branches of the 1 to 0 and 0 to 1 transitions can be found. By this way, glitch reduction is possible.

In this thesis, subtrees are found that occupy most of the transitions and subexpression selection change is made in that branch. In some cases, it was observed to bring advantages in the number of transitions. Overall, glitches do not decrease significantly. The use of ripple carry adders throughout the study may be the reason for high glitching.

5. SOFTWARE

Software codes were written in C programming language. C codes are divided into two sections; the algorithm code and the vhd code creator. Algorithm c code is explainded in detail in the next section. For both algorithm code and vhdl code creator, there will be a flowchart given for better understanding of how the codes work. Algorithm code was developed as explained in previous chapters. However, most of the programming time was taken by the vhd code creator file. There are also small c codes are written for manuel entry of adders, substractors and other components. In the following section, the algorithm code is explained and then the vhd code creator is explained in detail.

5.1 Algorithm Code

Algorithm finds the subexpressions as explained in previous chapters. Afterwards it produces a text file, where the FIR filter is described by subexpressions. Then, this text file is used by the vhdl code creator. In this section, the algorithm is explained via a flowchart. The algorithm code output is in the same format that Yurdakul and Hartley use. The format of the text file is illustrated in Figure 5.1.

```
a2 = ( a1 << 0 ) - ( a1 << 2 )
a3 = ( a1 << 2 ) + ( a1 << 0 )
a4 = ( a2 << 4 ) - ( a3 << 0 )
...
d0 = ( a5 << 0 )
t1 = d0 + ( a7 << 2 )
d1 = t1
t2 = d1 - ( a5 << 0 )
d2 = t2
...
o119 = t119
```

Figure 5.1. Example Output of the Algorithm Code

From Figure 5.1, the description of a 120 tap FIR filtercan be seen. Symbols start with ‘a’ are the coefficients. ‘d’ symbolizes the delay element output. It doesn’t have effect on the calculations. It just buffers its input. Symbols starting with ‘t’ are used for adder taps. Finally the symbol ‘o’ represents the output of the filter. It seems that the symbol for input is missing. However, for simplicity in the vhd code creator, ‘a1’ is the input. Therefore, filter coefficients start from ‘a2’.

Coefficients are held in a text file also. They are taken one by one and entered into the program. Each coefficient is processed sequentially and subexpressions are found. While subexpressions are found they are written to a dictionary array. Newly discovered subexpressions are written to the output file as in Figure 5.1. When all subexpresions are found, the taps and then output is written to the output file. The format of the input that is given to the program can be found in Figure 5.2.

```
Tapno= 120
Wordlength= 18
000000000N010N0N01
000000000100N00N00
00000000010N01010N
00000000010N000001
000000000100000N0
000000000000001001
. . .
```

Figure 5.2. Example Coefficient File

The algorithm code flowchart can be found in the next page in Figure 5.3. Algorithm code is not too complicated as it drawn in the flowchart. First of all, tap no and wordlength is read from the coefficient text file and neccesary arrays are created. Then, the first coefficient is read from the file. Number of nonzero terms are counted to check if there is any term to be taken as a subexpression. After that, There is a if statement; if the counted number of nonzero terms is 1 or none, then there is no need to find subexpressions since the result is already known to be 0. If there are more then 1 nonzero terms, then the subexpression must be found. In the next step, the number of zero terms between each nonzero termis found. Then, there is one more operation that the counted zeros are added together to create a weighted result. This will be used to select the subexpression.

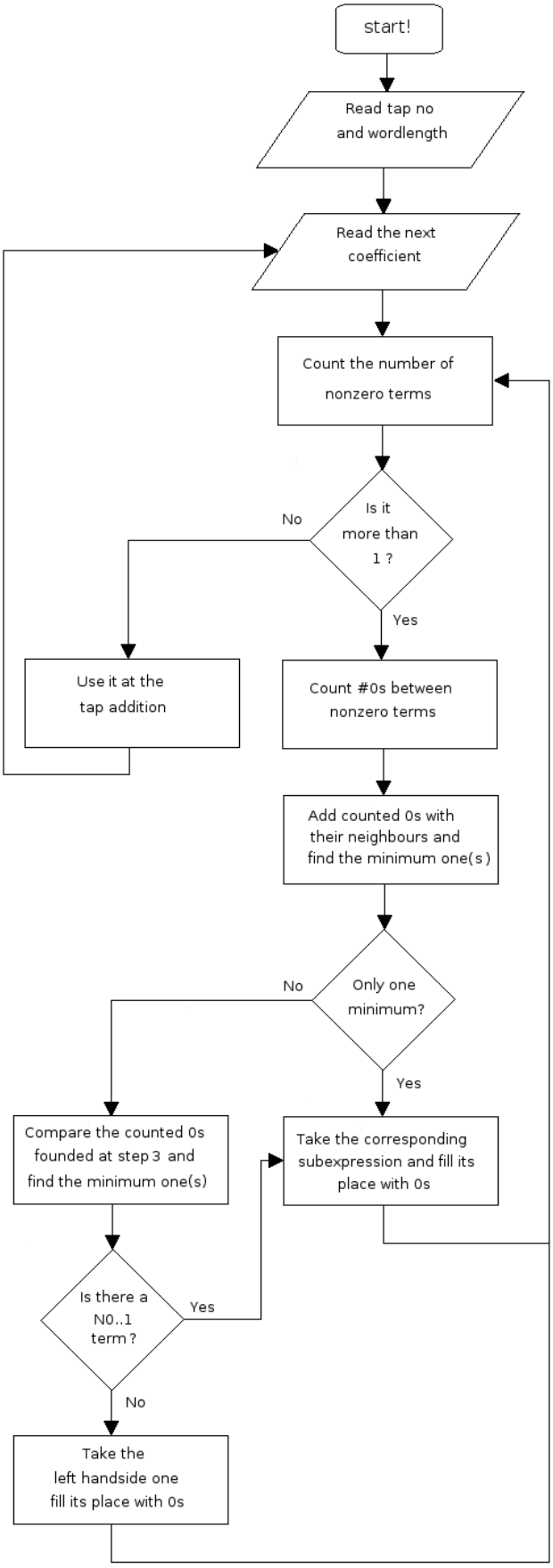


Figure 5.3 Algorithm Code Flowchart

After finding weighted sums, there is an if condition. The software checks if there is only one minimum point found. If the answer is yes, then the subexpression is taken and written to a dictionary and for minimum adder depth condition it is erased from the coefficient to find other subexpressions. This way two term subexpressions taken at a time. If the answer is no, the counted zeros are compared and minimum one is taken to be the subexpression and its place filled by zeros. If both are the same again, then the 10N and N01 comparision is made. If an N01 term exists, then it is taken. Otherwise any subexpression may be used. The left hand sided expression is taken by default. After finding a subexpression, the loop refreshes itself until there is no coefficient left. In the next section, the vhd code creator software is explained in detail with a flowchart.

5.2 VHD Code Creator

Vhd code creator is written to make power simulations avaliable. The code creator output many vhd files and they are coded at the gate level. Therefore, it will be very easy to download this code to an FPGA and use it. The code uses ripple carry adders for its adders and subtractors. Shifts are made hardwired which means that there is no need for registers or buffers to make shifts. Entire hardwired shifts are taken by the code by appopriate placement of the inputs and outputs. The VHD code creator also take care of adderlengths as explained in the previous chapters.

The flowchart of the software code can be found in Figure 5.4. It is a very simplified flowchart, since the code is over 1000 lines it is not possible to fit it in one page as a flowchart. There are many of if commands in the code because of the ripple carry adder operations, shifts variations and adder signs. Since the vhd code is written in gate level there are many fprintf commands. All of these different possibilities and file processing makes the code huge.

On the other hand, flowchart is easy to understand. First, the input file that is taken from the algorithm code is read. The first character is compared with ‘o’, ’d’, ’t’ in order. If there is o recognized, it means the filter is at its last line and there is nothing but output is

written to the main vhd code and code exists. If ‘d’ is recognized, it means there is a need of a register. The length of the register is looked up from the previous tap output. Since they are all written to arrays, there is no problem in finding this data. After that, the register is created at gate level in a separate file. In the main code, it is present as a component. The program goes back to the input file and reads another input. If this input is ‘t’, then the tap addition is to be done. For tap addition, there are two inputs; one of them comes from the previous stages’ register output and the other is the this stages multiplication output. These two inputs must be added to get the tap addition result. Therefore, the code computes the adderlength to be used in this addition from the Equation 3.3.

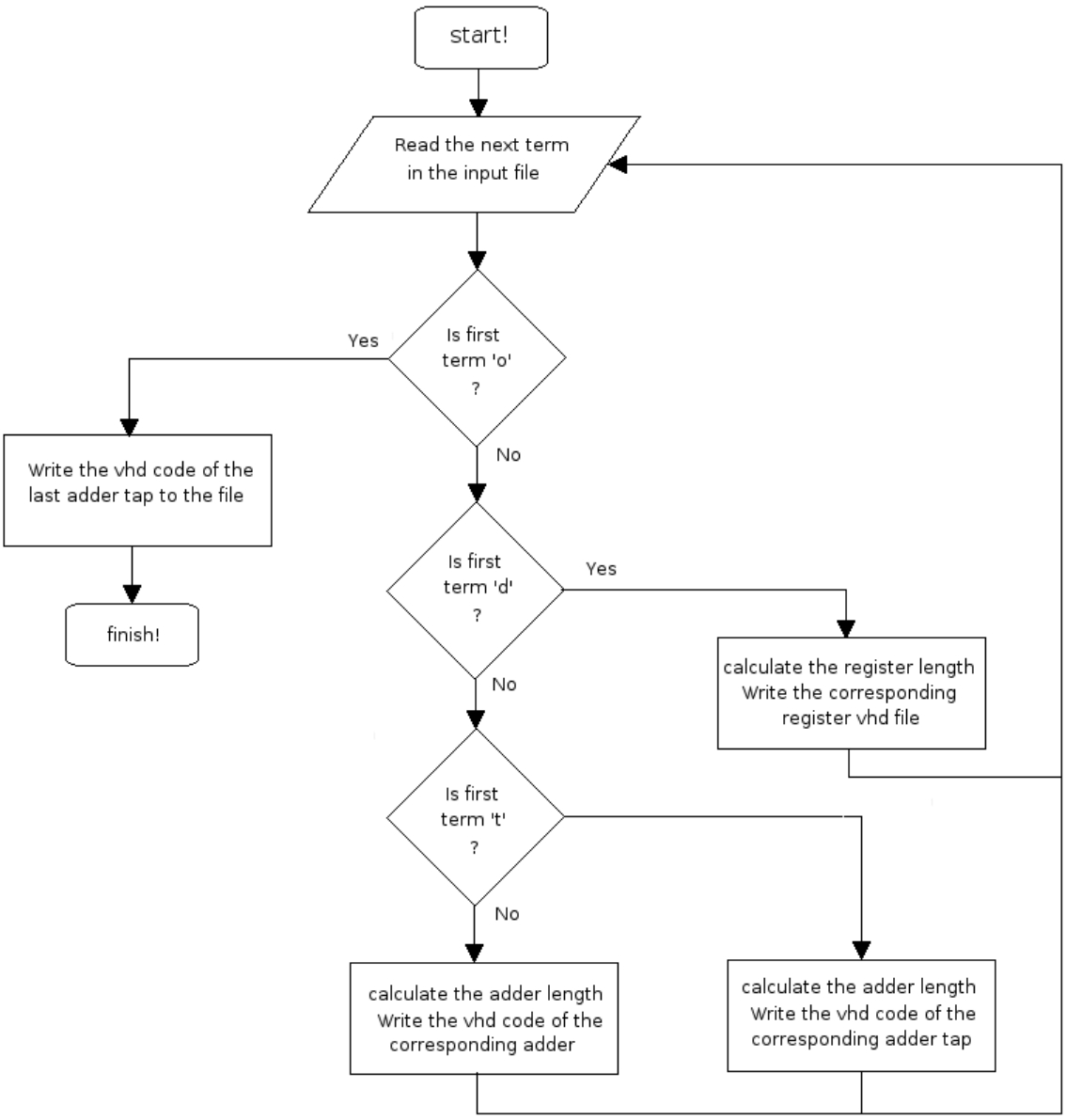


Figure 5.4. Vhd Code Creator Flowchart

The adder vhd code is created in a seperate file and the tap adder subcomponent is written to the main vhd file. Then, the code reads another input from the file. If the input starts with ’a’, that means there is a subexpression. There is an addition, therefore the adder lengths are calculated as explained before and the adder is created as a seperate vhd file. If there is a smilar adder created before, there is no need to create the same adder again. Both must have same length, shift and carryout. The main vhd code is updated with this adder component.

The creation of seperate subcomponents and adding them into the main vhd file is done as in this order. While taps are created, all these components in the main vhd file are connected through signals. This way, the vhd files are created.

6. RESULTS

The Subexpression finder and vhd code producer software has been written to carry out simulations. 120 tap 18 bit and 124 tap 14 bit FIR filters are used for these simulations The comparisons of components for 120 tap filter is in Table 6.1.

Table 6.1. Component Comparisons of the Algorithms for 120 tap filter

	Hartley's Algorithm	**Yurdakul's Algorithm**	**Proposed Algorithm**
INVERTERS	340	634	302
FLIP-FLOPS	2819	2819	2819
FAs	3543	3409	3634
ADDERS	70	69	76
A. DEPTH	3	4	3

Proposed algorithm found 6 more adders compared to Hartley's algorithm in Table 6.1. Maximum adder depth is smaller in Hartley's and proposed algorithm compared to Yurdakul's algorithm. From the Table 6.1 and Table 6.2, number of inverters are minimum in proposed algorithm. Component and adder depth comparisons of 120 tap 14 bit filter can be found in Table 6.2. Maximum adder depth is same in all algorithms for this filter.

Table 6.2. Component Comparisons of the Algorithms for 124 tap filter

	Hartley's Algorithm	**Yurdakul's Algorithm**	**Proposed Algorithm**
INVERTERS	177	329	165
FLIP-FLOPS	2414	2414	2414
FAs	2658	2603	2667
ADDERS	33	33	34
A. DEPTH	3	3	3

The wordlength in second filter is 14. Filter is not complex. Therefore, subexpression sharing is high and number of adders are close. Propesed algoithm found 1 more adder compared to other algorithms.

Table 6.3. Power comparisons of algorithms for 120tap filter

Inputs	Simulation Type	Hartley's Algorithm(µW)	Yurdakul's Algorithm(µW)	Proposed Algorithm(µW)
66000 Sample	zd	8623	8456	8760
	ed	13400	13334	13729
16384 Sample	zd	11772	11560	12010
	ed	21311	21427	21887
4096 Sample	zd	11768	11555	12007
	ed	21216	21333	21592

Power simulations for 120 tap filter is in Table 6.3. Zero delay and enable delay simulations are performed for filters. 3 different random input files are used. Power consumption of proposed algorithm 2% above compared to Hartley's algorithm.

Table 6.4. Power comparisons of algorithms for 124 tap filter

Inputs	Simulation Type	Hartley's Algorithm(µW)	Yurdakul's Algorithm(µW)	Proposed Algorithm(µW)
66000 Sample	zd	6501	6441	6521
	ed	9343	9219	9467
16384 Sample	zd	9041	8968	9069
	ed	14051	14536	14573
4096 Sample	zd	9126	9053	9157
	ed	14560	14581	14615

Power simulations for 124 tap filter is in Table 6.4. Filter has 14 wordlength and filter coefficients are short. Therefore, hardware complexity of the filter is low. Power simulation results are close.

In zero delay simulations, gates are static and there is no gate delays. In enable delay simulations, gates are not static. Therefore, glitch is also present in the simulations. It can be seen from Table 6.3 & 6.4 that glitch increases the power dramatically. In all cases, nearly 30% of the power consumed by glitches.

7. CONCULUSION & FUTURE WORK

FIR filter based power analysis was done. A new subexpression algorithm was proposed which optimizes the adder depth that leads to minimum glitch and delay. The proposed algorithm also decreases the number of FAs. The number of FAs is dependent on the coefficient length. The number of total adders was seen to increase slightly compared to Hartley's algorithm.

As future work, the subexpression sharing algorithm can be developed to have less adders than before. This can be done by adding a search mechanism on frequently used subexpressions. If frequently used subexpressions are taken first, as well as taking into account the adder length, there will be few adders. Minimizing the appearence of 1xN terms will lead to less FAs. These will be the motivation to continue analysing other Filter structures.

REFERENCES

1. Hartley, R. I., ‘‘Subexpression Sahring in Filters Using Cannonic, Signed Digit Multipliers’’, *Analog and Digital Signal processing,* Vol. 43, No. 10, pp. 677-688, 1996.

2. Yurdakul, A. and G. Dündar, “Multiplierless Realization of FIR-based Multirate Systems by Using Common Two-Term Expressions”, *J. VLSI Signal Processing*, Vol. 22, pp. 163-172, 1999.

3. Reitwiesner, R. W., ‘‘Binary Arithmetic’’, *in Advances in Computers. NewYork, Academic*, Vol. 1, pp. 231-308, 1966.

4. Ifeachor, E. C., B. W. Jervis, ‘‘Digital Signal Processing: A Practical Approach’’, Second Edition, 2002

5. Ghosh, A. A., S. Devadas, K. Kentzer, and J. White, ‘‘Estimation of avarage switching activity in combinational and sequential circuits’’, *Proceedings of Design Conference,* pp. 68-73, 1992

REFERENCES NOT CITED

Hartley, R. I., ‘‘Optimization of Canonic Signed Digit Multipliers for Filter *Design’’, IEEE International Sympoisum on Circuits and Systems*, Vol. 4, pp. 1992-1995, 1991

Park, I. and H. Kang, ‘‘Digital Filter Synthesis Based on an Algorithm to Generate All Minimal Signed Digit Representations’’, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and systems*, Vol. 21, No. 12, pp. 1525-1529, 2002.

Pasko, R., P. Schaumont, V. Derudder and S. Vernalde, Member, IEEE,and D. Durackova, ‘‘A New Algorithm for Elimination of Common Subexpressions’’, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and systems*, Vol. 18, No 1, pp. 58-68, 1999.

Hewlitt, R. M., E. S. Swertzlander, ‘‘Cannonical Signed Digit Representation for FIR Digital Filters’’, *Signal Processing Systems*, *IEEE Workshop on Volume , Issue* , pp. 416 – 426, 2000.

Hashemian R., ‘‘A New Method for Conversion of a 2’s Complement to Canonic Signed Digit Number System and its Representation’’, *Signals, Systems and Computers, Conference Record of the Thirtieth Asilomar Conference on Volume Issue* , Vol. 2, pp 904-907, 1996.

Dempster, A. G., ‘‘Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters’’, *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing,*V. 42, pp. 569-577, 1995

Dempster, A. G., M. D. Macleod, ‘‘Constant Integer Multiplication Using Minimum Adders’’, *IEEE proc.-Circuit Devices Syst.*, Vol.141, No. 5, 1994.

Gustafsson, O., H. Ohlsson and L. Wanhammar, ‘‘Minimum-Adder Integer Multipliers Using Carry-Save Adders’’, *Circuits and Systems, IEEE International Symposium on Volume 2*, *Issue,* pp. 709 – 712, 2001.

Gustafsson, O., L. Wanhammar, ‘‘ILP Modelling of the Common subexpression Sharing Problem’’, *9th International Conference on Electronics, Circuits and Systems,* Vol. 3, pp. 1171-1174, 2002

Martínez-Peiró, M., E. Boemo, L. Wanhammar, ‘‘Design of High-Speed Multiplierless Filters Using a Nonrecursive Signed Common Subexpression Algorithm’’, *IEEE transactions on Circuit and systems,* Vol. 49, No. 3, pp.196-203, 2002.

DeBrunner, L. S., V. E. DeBrunner, D. Bhogaraju, ‘‘Defining Canonical-Signed-Digit Number Systems as Arithmetic Codes’’, *Signals, Systems and Computers, Conference Record of the Thirty-Sixth Asilomar Conference on Volume 2, Issue*, Vol. 2, pp. 1593 – 1597, 2002.

Mehendale, M, S. D. Sherlekar, G. Venkatesh, ‘‘Synthesis of Multiplier-less FIR Filters with Minimum Number of Additions’’, *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design,* pp.668-671, 1995

Matsuura, A., M. Yukishita, A. Nagoya, ‘‘An Efficient Hierarchical Clustering Method for the Multiple Constant Multiplication Problem ‘’, *Design Automation Conference, Proceedings of the ASP-DAC*, Volume Issue, 28-31, pp. 83-88, 1997.

Vinod, A. P., E. M-K. Lai, A. B. Premkumar and C. T. Lau, ‘‘FIR filter implementation by efficient sharing of horizontal and vertical common subexpressions’’, *Electronic letters*, Vol. 39, No.2, 2003.

Xu, F., C-H. Chang, and C-C. Jong, ‘‘Contention Resolution Algorithm for Common Subexpression Elimination in Digital Filter Design’’, C*ircuits and Systems II: Express Briefs, IEEE Transactions on Analog and Digital Signal Processing,*Vol. 52, Issue 10, pp. 695 – 700, 2005.

Chung, K-S., T. Kim, C.L Lin, ‘‘G-vector: a new model for glitch analysis’’, Twelfth Annual IEEE International ASIC/SOC Conference, 1999. Proceedings, pp 159-162, 1999.

Chung, K-S., T. Kim, C.L Lin, ‘‘A Non-Zero Delay Model for Glitch Analysis in Logic Circuits’’, *IEEE Midwest Symp. on Circuits and Systems,* Lansing MI Proc 43rd, pp. 1244-1247, 2000.