academicJournals

Vol. 9(8), pp. 257-263, 30 April, 2014
DOI: 10.5897/SRE2014.5846
Article Number: 91C18ED44201
ISSN 1992-2248 © 2014
Copyright©2014
Author(s) retain the copyright of this article
http://www.academicjournals.org/SRE

Scientific Research and Essays

Full Length Research Paper

Synthesis of Chebyshev-I filter using folding and retiming

Nongmaithem Lalleima Chanu[1]* and Vimal Kant Pandey[2]

Department of ECE, M-Tech Digital Electronics, DIT Dehradun, India – 248009.

Received 25 February 2014; Accepted 9 April, 2014

In synthesizing Digital signal processing (DSP) architecture, maintaining low silicon area and high performance becomes an important factor which can be achieved by various optimization techniques. To achieve this, we employ two design optimization techniques: folding and retiming, which are applied to 3rd order Chebyshev I high pass digital filter to minimize the functional units (adders, multipliers) and to reduce the number of registers. Folding transformation is used to determine the control circuits in DSP architecture by executing multiple algorithm operation on a single functional unit. Retiming using register minimization is applied after folding, thereby reducing the numbers of multipliers and adders from 7 to 1 and 6 to 1, respectively, without affecting the input and output characteristics of the filter.

Key words: Data flow graph (DFG), Chebyshev filter, folding, retiming, lifetime analysis.

INTRODUCTION

Tremendous growth of digital signal processing (DSP) and its importance promotes advances in certain fields of applications such as telecommunication, military, instrumentation and control, image processing, seismology, speech processing and biomedical signal processing. DSP programs are executed repetitively for an infinite number of times and they are assumed to be non-terminating (Jackson et al., 2003; Salivahanan et al., 2010). This can be exploited by designing more efficient DSP system in terms of speed, area and power. The strategy of designing an efficient filter also needs to concentrate on reducing the number of functional units.

Advancement in technology and emerging trends required DSP architecture with less space and power consumption where the signal processing algorithm are modified to accommodate the circuit. To achieve the goals such as less area, high speed and low power different algorithms are proposed such as pipelining, folding, retiming etc. The transformation in which multiple algorithm operations are time multiplexed to a single functional unit is known as folding. This algorithm provides a technique for designing control circuits for hardware and helps to synthesize DSP architecture that can be operated using single or multiple clocks. Folding reduces the number of functional units; it may also lead to the usage of large number of registers (Keshab, 2012; Rajalakshmi et al., 2013). To avoid this, retiming technique is used to compute the minimum number of registers require to implement a folded DSP architecture and to allocate data to these registers to provide

*Corresponding author. E-mail: lallei.chanu98@gmail.com

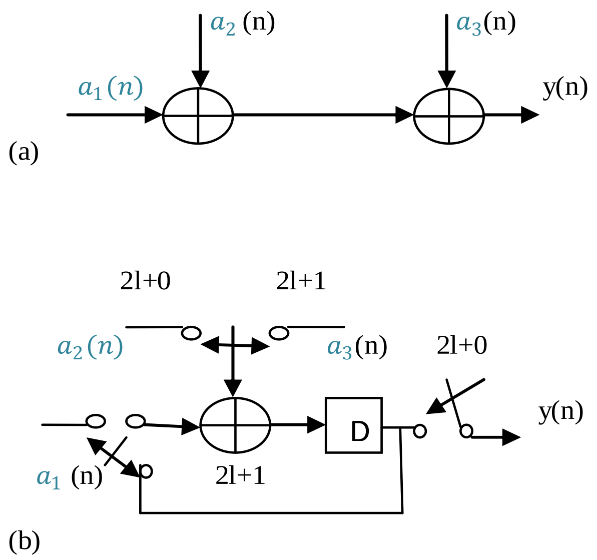


Figure 1. (a) A DSP program with 2 addition operations; (b) A folded architecture where the 2 addition operations are folded to a single pipelined adder.

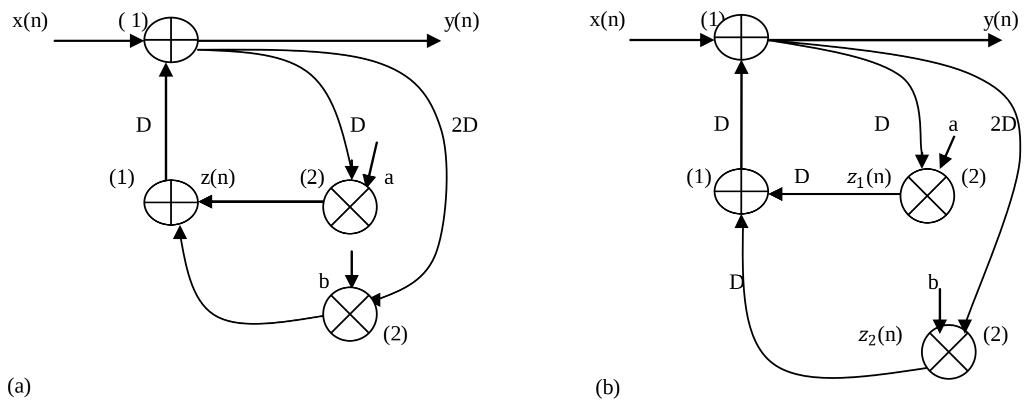


Figure 2. Two versions of an IIR filter and the computation times of the nodes are in parentheses.

architecture with low silicon area (Rajapadhy and Kiaei, 1991). This design optimization platform is designed using MATLAB/Simulink and Xilinx.

DESIGN OPTIMIZATION TECHNIQUE

Folding

This transformation technique helps to determine the control circuits in DSP system in which multiple algorithm operations are time multiplexed to a single functional unit which leads to the reduction of functional units (such as adders, multipliers) resulting with low silicon area. Figure 1a shows an example of a DSP program for adding two samples where the operation computes

$$Y(n) = a_1(n) + a_2(n) + a_3(n)$$

Here, one output sample is produced every 2 clock cycles and hence the input is valid for 2 clock cycles (2l+0 and 2l+1, where l is the iteration). In 0 cycle, a_1(n)+a_2(n) is performed. In cycle 2l+1(second), a_1(n)+a_2(n) is switch to the adder along with a_3(n) and the sum is stored in the unit cycle 2. Folded architecture in which 2 addition operation are folded to a single pipelined adder is shown in Figure 1(b).

Retiming

It is used to change locations of delay elements without changing the input/output characteristic of the system which is illustrated using Figure 2a and b. The filter 2(a) is described by

$$z(n) = ay(n-1)+by(n-2)$$
$$y(n) = z(n-1)+x(n)$$
$$= ay(n-2)+by(n-3)+x(n)$$

And filter 2(b) is described by

z_1(n) = ay(n-1)
z_2(n) = by(n-2)
y(n) = z_1(n-1)+z_2(n-1)+x(n)
= ay(n-2)+by(n-3)+x(n)

These two filters are having the same input-output characteristics and can be derived from one another with the help of retiming, even though these filters are having delays at different locations.

Retiming can be used to increase the clock rate, to decrease the number of registers and to reduce the power consumption of a circuit (Keshab et al., 1992).

CHEBYSHEV FILTER

Type I Chebyshev filters are all-pole filters that exhibit equiripple behavior in the passband and monotonic characteristic in the stopband. By increasing the order N, the Chebyshev response approximates the ideal response. It has the property that they minimize the error between the idealized and the actual filter characteristic over the range of the filter. This type of filter is named after Pafnuty Chebyshev (John and Dimitris, 1996). The magnitude response of Chebyshev type I filter can be expressed as:

$$|H(j\Omega)| = \frac{A}{[1+\varepsilon^2 \; C_N^2(\frac{\Omega}{\Omega_c})]^{0.5}}$$

Where A is the filter gain, ε is the constant, $\Omega_c$is the 3dB cutoff frequency. C_N(x) is the N^{th} order chebyshev polynomial defined as

C_N(x) = cos (N$\cos^{-1} x$), |x|≤ 1 (passband)

C_N(x)= cos(N$\cosh^{-1} x$), |x|> 1 (stopband) and the chebyshev polynomial is defined by the recursive formula:

$C_N(x)$ = 2x $C_{N-1}(x)$ - $C_{N-2}(x)$, N > 1

Where $C_0(x)$ = 1 and $C_1(x)$ = x

DATA FLOW GRAPH (DFG)

The operations of DSP algorithm are assumed to be executed repetitively. The DSP filter blocks needed to be optimized can be represented by DFG due to its easier, efficient and compactness. DFG is a directed graph G with sets of nodes/vertices V and sets of edges E (Edward, 1991; Rakshi et al., 2010). Each node in the DFG represents an algorithm operation and any arc *U → V with* w(e) delays states that the output of the l^{th} iteration of U is used to execute the $(l + w(e))^{th}$ iteration of V. The arc with and without delays represent the inter iteration and intra iteration precedence constraint, respectively (John and Dimitris, 1996).

DFG is used to described hardware architecture which depends on folding factor (N), number of operation folded to a single functional unit. Hu and Hv denote the operators that execute the operation U and V in the hardware DFG. Operations processed by the operators form a folding set S. Each folding sets contains N entries, some of which may be a null operations denoted as $\emptyset$. A delay or register elements in the hardware represents a storage unit.

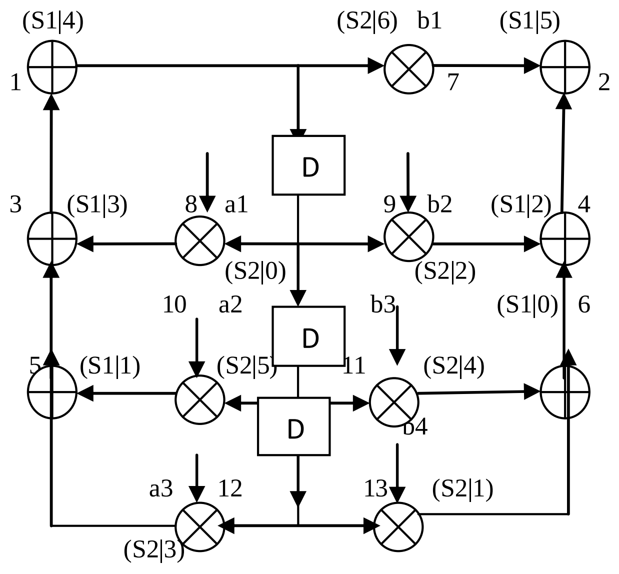


Figure 3. Direct form II, 3rd order Chebyshev I filter.

FOLDING EQUATIONS

Folding is a transformation technique used to reduce the silicon area by time multiplexing many algorithm operations into single functional units, such as adders and multipliers. It provides a systematic process to design control circuit for hardware. Folding is applied to the filter to reduce the chip area (Keshab, 2012). Consider an edge e connecting the nodes U and V with w(e) delays. Let the execution of l^{th} iteration of the nodes U and V be scheduled at the time units Nl+u and Nl+v, respectively, where u and v are the folding orders of nodes U and V that satisfy 0≤u, v≤N-1. Hu and Hv denote the functional unit that executes the nodes U and V. If Hu is pipelined by Pu stages, the l^{th} iteration of node U is available at time unit Nl+u+Pu. The result of l^{th} iteration of node U is used by $(l + w(e))^{th}$ iteration of the node V which is executed at N(l+w(e)) + v. Thus, the result must be stored for:

$$\begin{aligned} D_F(U \rightarrow V) &= [N(l + w(e)) + v] - [Nl + Pu + u] \\ &= Nw(e) - Pu + v - u \end{aligned} \quad (1)$$

Time units, which is independent of l, a folding set is an ordered set of N operations executed by the same functional unit which depends on the folding order. The folding order of a node is the block of time to which the node is scheduled to execute the operation in the hardware. The folding sets of 3rd ordered Chebyshev filter is shown in Figure 3 and are given by

S_A= S1= {1, 2, 3, 4, 5, 6, $\emptyset$} and

S_M= S2= {7, 8, 9, 10, 11, 12, 13}

Using the above folding sets, the filter is folded with folding factor 6 which means that the iteration period of the folding architecture is 6 units of time (u.t). Here, each node of the filter is executed once every 6 u.t in the folded architecture that is the folded hardware executes six operations. The folding set S_A contains one null operation in Position 6 during which no operation is performed by the adder. The folding equations for each edge are given in Table 1

Table 1. Folding equations.

Edge	Folding equation
1→ 7	D_F(1→ 7) = 6(0)-1+6-4 = 1
1→ 8	D_F(1→ 8) = 6(1)-1+0-4 = 1
1→ 9	D_F(1→ 9) = 6(1)-1+2-4 = 3
1→ 10	D_F(1→ 10) = 6(2)-1+5-4 = 12
1→ 11	D_F(1→ 11) = 6(2)-1+4-4 = 11
1→ 12	D_F(1→ 12) = 6(3)-1+3-4 = 16
1→ 13	D_F(1→ 13) = 6(3)-1+1-4 = 14
8→ 3	D_F(8→ 3) = 6(0)-2+3-0 = 1
3→ 1	D_F(3→ 1) = 6(0)-1+4-3 = 0
10→ 5	D_F(10→ 5) = 6(0)-2+1-5 = -6
5→ 3	D_F(5→ 3) = 6(0)-1+3-1 = 1
12→ 5	D_F(12→ 5) = 6(0)-2+1-3 = -4
7→2	D_F(7→ 2) = 6(0)-2+5-6 = -3
9→ 4	D_F(9→ 4) = 6(0)-2+2-2 = -2
4→ 2	D_F(4 → 2) = 6(0)-1+5-2 = 2
11→ 6	D_F(11→ 6) = 6(0)-2+0-4 = -6
6→ 4	D_F(6→ 4) = 6(0)-1+2-0 = 1
13→ 6	D_F(13→ 6) = 6(0)-2+0-1 = -3

using Equation 1. Here, the equations are derived with an assumption that addition and multiplication operations require 1 and 2 units of time, respectively.

RETIMING

Basically, retiming is also a transformation technique used to change the location of the delay elements without affecting the input and output characteristic of the circuit. Retiming has to be performed before folding to forced causality of the system (Leiserson et al., 1986; Monteiro et al., 1993). The negative values of the above folding equations are made positive by using cutest retiming; a special case of retiming which only affects the weights of the edges of the cutest. It consist of adding k delays to each edge from disconnected subgraphs G1 to G2 and removing k delays from G2 to G1. Using retiming the weight of the edge U→ V is computed as:

$w_r(e)$= w(e)+r(V)-r(U) (2)

The retiming folding constraints are obtained using the relation

r(U)-r(V)≤ $\left\lfloor \frac{D_F(U \rightarrow V)}{N} \right\rfloor$, (3)

Where ⌊x⌋ is the floor of x, which is the largest integer less than or equal to x. The retimed folding constraints are:

r(1)-r(7)≤0, r(1)-r(8)≤0, r(1)-r(9)≤0

r(1)-r(10)≤2, r(1)-r(11)≤2, r(1)-r(12)≤3,

r(1)-r(13)≤2, r(8)-r(3)≤0, r(3)-r(1)≤0,

r(10)-r(5)≤-1, r(5)-r(3)≤0, r(12)-r(5)≤-1,

r(7)-r(2)≤0, r(9)-r(4)≤0, r(4)-r(2)≤0,r(11)-r(6)≤-1,
r(6)-r(4)≤0, r(13)-r(6)≤0

From these folding constraints, we can form the constraint graph. The inequalities can be solved using Floyd–Warshall algorithm and the final constraints after applying algorithm are:

r(1)=0, r(2)=0,r(3)=0,r(4)=0,r(5)=0,r(6)=0,

r(7)=0, r(8)=0, r(9)=0, r(10)=-1, r(11)=-1,

r (12)=-1, r(13)=0.

We can find the new retimed value using Equation 2. By applying the folding equations the new delays can be obtained and then cutest retiming is applied to have positive values from which the architecture can be derived.

REGISTER MINIMIZATION TECHNIQUE

The main objective here is to minimize the architectural area by minimizing the number of registers. The folded structure contains a higher number of register because the intermediate results need to be stored (Keshab, 2012; Parhi, 1992; Rajapadhy and Kiaei, 1991). This minimization process follows two steps:

(a) Lifetime analysis table and lifetime chart.
(b) Data allocation using forward and backward register allocation.

Table 2. Lifetime table.

Node	$T_{input} \rightarrow T_{output}$
1	5→ 19
2	–
3	4 → 4
4	3 → 5
5	2→ 3
6	1→ 2
7	8 →11
8	2→ 3
9	4 →8
10	7→ 7
11	6→ 12
12	5 → 7
13	3 → 6

In lifetime analysis, a data sample (variable) is live from the time it is produced through the time it is consumed. It is dead, after the variable is consumed. When the variable is live, it occupies one register (Deepa and Vijaya, 2012). Here, the number of live variables at each time unit is computed and the number of registers needed by the folded architecture is computed. Lifetime table can be constructed by considering the two parameters and their relations:

T_{input} = u+P_u (Table 2).

T_{output} = T_{input} +max{D_F(U→ V)}

The linear lifetime chart is shown in Figure 4, which graphically represents the lifetime of each variable.

Here, the horizontal lines represent the clock cycle and vertical lines represent the lifetime of a variable. With the help of this chart, the resultant minimum number of registers is obtained as the maximum number of live variable at any time step. The maximum number of register is Max{0,0,1,1,2,3,4.4,3,4,4,5,5,5,5,4,5,5,6,6} = 6. After lifetime chart, the minimum number of register required to implement the architecture is found to be 6 and data are allocated to the registers. The registers are named as R1, R2, R3, R4, R5 and R6. This allocation scheme dictates how the variables are assign to registers (Figure 5).

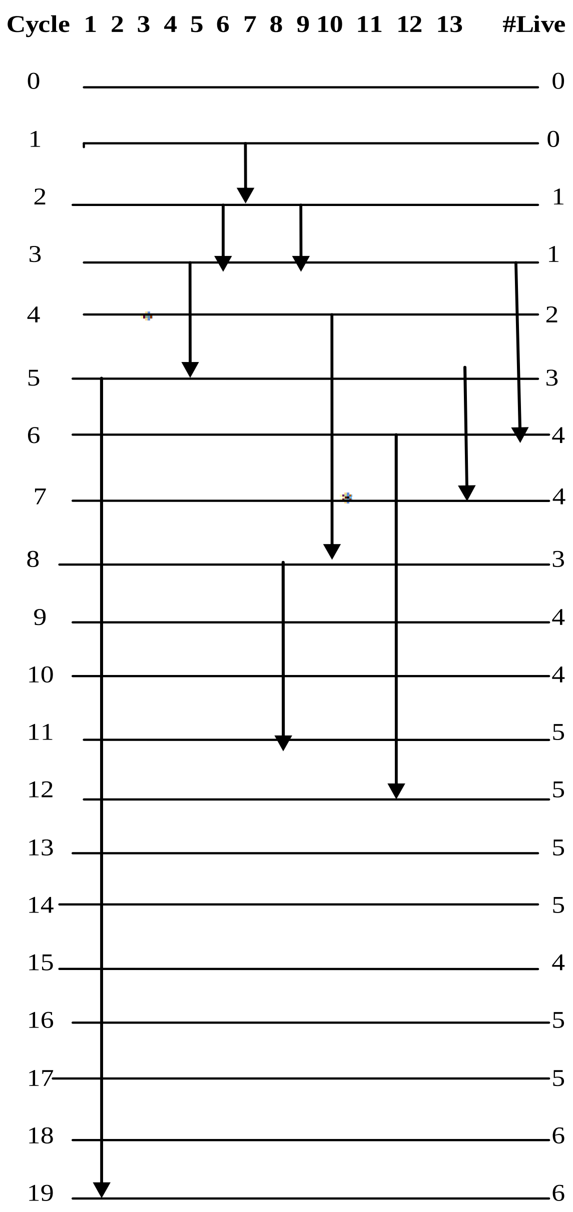


Figure 4. Lifetime analysis chart.

Folded architecture

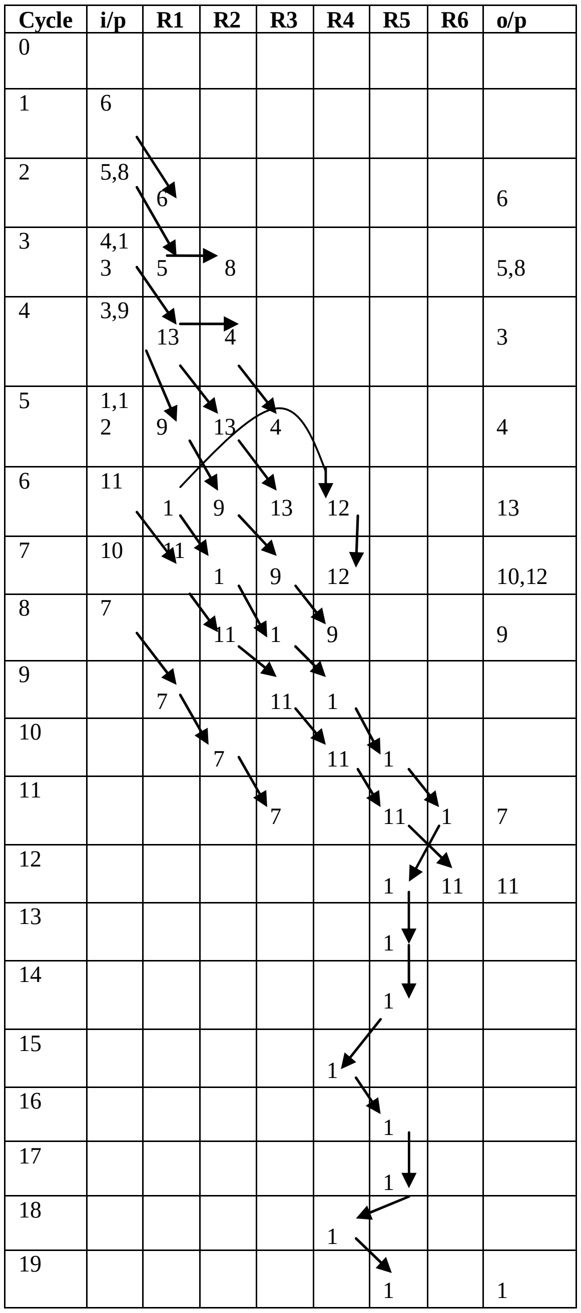
The folded architecture with respect to the folding equations and allocation table with a minimum of 6 registers is derived and shown in Figure 6.

RESULTS

Tables 3 and 4 show the comparison of unfolded and folded filter with respect to adders, multipliers and number of registers, and the device utilization for both the architecture. These architectures (unfolded and folded with register minimization) are synthesized using Spartan 3A/3AN device. Here the number of functional units such as adders and multipliers are reduced to 1 each, with 6 registers. In addition to these, the number of components namely slices, slice flip flop, look up tables (LTU) and input-output blocks (IOs), present in Spartan are reduced due to the reduction in functional units.

Conclusion

This paper addresses the challenges and opportunity of minimizing the filter architecture by the growing trend of VLSI DSP systems. It has been demonstrated that the



Cycle	i/p	R1	R2	R3	R4	R5	R6	o/p
0								
1	6							
2	5,8	6						6
3	4,13	5	8					5,8
4	3,9	13	4					3
5	1,12	9	13	4				4
6	11	1	9	13	12			13
7	10	11	1	9	12			10,12
8	7		11	1	9			9
9		7		11	1			
10			7		11	1		
11				7		11	1	7
12						1	11	11
13						1		
14						1		
15					1			
16						1		
17						1		
18					1			
19						1		1

Figure 5. The allocation table.

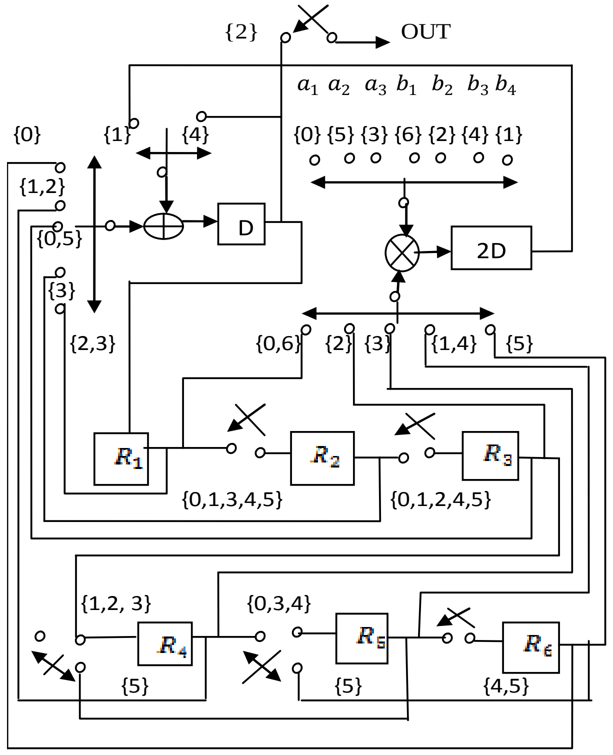


Figure 6. Folded architecture for Chebyshev I filter.

Table 3. Comparison between unfolded and folded filter.

Architecture	Adder	Multiplier	No. of register
Unfolded	6	7	3
Folded	1	1	6

Table 4. Comparison of unfolded and folded architecture.

Architecture	Unfolded	Folded
No. of Slices	112	9
No. of Slice flip flop	24	16
No. of 4 input LTUs	194	8
No. of IOs	76	68

Chebyshev I high pass filter architecture in terms of required number of functional unit such as adder and multiplier is substantially reduced from 6 adders to 1 and 7 multipliers to 1 by the folding and retiming with register minimization techniques. This context describes an effective and efficient heuristic and provides an optimized environment for digital filter. Our experimental results demonstrate that folding and retiming can significantly reduce the silicon area and therefore providing flexibility to the cost and effort of the designers.

Conflict of Interests

The author(s) have not declared any conflict of interests.

REFERENCES

Deepa Y, Vijaya K (2012). High Speed Digital Filter using register minimization retiming and Parallel Prefix Adders. IEEE 3rd International Conference on EAIT. pp. 449-453.

Edward AL (1991). Consistency in Data Flow Graph. IEEE Trans.

Parallel Distrib. Syst. 2:225-235.

Jackson LB, Keiser JF, Donald HS (2003). An approach to implementation of Digital Filters. IEEE Trans. Audio Electroacoust. 16(3):413-421. http://dx.doi.org/10.1109/TAU.1968.1162002

John GP Dimitris GM (1996). Digital Signal Processing Principles, Algorithms and Applications. Pearson Education (3rd ed.), Chapters 7 and 8.

Keshab KP (2012). VLSI Digital Signal Processing Design and Implementation. In Wiley Student (ed.), Chapters 4 and 5.

Keshab PK, Wang CY, Brown AP (1992). Synthesis of Control Circuits in Folded Pipelined Architecture. IEEE J. Solid State Circuit. 27(1):29-43. http://dx.doi.org/10.1109/4.109555

Leiserson C, Rose F, Saxe J (1986). Optimizing Synchronous Circuitry by Retiming. Third Caltech Conf. VLS I:87-116.

Monteiro DS, Ghosh A (1993). Retimimg Sequential Circuit for low power. In Proc. IEEE Int. Conf. Comput. Aided Design. pp. 398-402.

Parhi KK (1992). Synthesis of DSP data format converted using lifetime analysis and forward-backward register allocation. IEEE Trans. Circuit Systems-II. 39(7):423-440. http://dx.doi.org/10.1109/82.160168

Rajalakshmi K, Arumugam K, Priya MS (2013). Folded Architecture for Digital Gammatone Filter used in Speech Processor of Cochlear Implant. ETRI. J. 35(4).

Rajapadhy S, Kiaei S (1991). A folding transformation for VLSI IIR filter array design. Int. Conf. Acoust. Speed Sig. Process. 9:1237-1240.

Rakshi S, Premananda BS, Mahir NM (2010). Synthesis of DSP System using Data Flow Graph for silicon area reduction. IJCSIT. 1(5).337-341.

Salivahanan S, Vallavaraj A, Gnanapriya C (2010). Digital Signal Processing. (2nd ed.), Tata McGraw Hill.