SYNTHESIS OF LOW POWER FOLDED PROGRAMMABLE COEFFICIENT FIR DIGITAL FILTERS *

Vijay Sundararajan *Keshab K. Parhi*

Dept. of ECE, University of Minnesota
Minneapolis, MN 55455
E-mail: vijay@ece.umn.edu, parhi@ece.umn.edu

ABSTRACT

A novel low-power synthesis technique is presented for the design of folded or time-multiplexed programmable-coefficient FIR filters where storage area is traded-off for lowering power consumption. A systematic technique is developed for low power mapping of FIR filters to architectures with arbitrary number of multipliers and adders. Power consumed in multipliers is reduced by reducing switching activity at both the data-input as well as the coefficient input. Common input operands can be exposed by unfolding, which, however leads to a memory increase. Simulation results are obtained for folding 65 and 129 tap bandpass FIR filters. The average power consumed in a multiplier for a fixed number of hardware multipliers with varying unfolding factors is compared. It is observed that most of the gains due to unfolding are obtained for relatively small unfolding factors and therefore for relatively small memory area overhead. Depending on the unfolding factor employed the average power consumed in a multiplier is seen to reduce anywhere from 54.75% to 81.73% when transpose FIR filters are synthesized as opposed to synthesizing direct-form FIR filters with no unfolding.

1. INTRODUCTION

Folding [1], [2], [3], [4], [5] or time-multiplexing is a technique for efficient resource sharing for area-constrained behavioral synthesis from a data-flow graph (DFG). The throughput requirement in folded architectures is met by pipelining the hardware functional units to a relevant number of levels. In this way folded architectures are able to meet both throughput and area constraints for a target application. Unfortunately, maximal resource sharing while minimizing area can lead to a severe increase in power consumption [6],[7]. This surge in power consumption in folded architectures is due to:
1) The increased switching activity in various data-path resources as a result of the successive application of uncorrelated inputs.
2) Increase in the interconnect overhead (muxes and busses) due largely to the inability of the synthesis methodology to capture the regularity in the original data-flow graph (DFG).

Exploitation of locality and regularity were recommended in [7], [8] for reducing power consumption in interconnects and busses. On the other hand increasing the correlation between successive inputs to a functional unit, e.g., a multiplier has been demonstrated as a power saving strategy in [9], [10], [11].

FIR filtering is one of the basic building blocks in DSP applications. Low power FIR filter synthesis, therefore, assumes paramount importance for the reduction of power consumption in DSP applications. Many times the throughput and resource constraints dictate the use of folded architectures. No systematic technique has so far been available which can reduce the power consumed in folded architectures even while allowing for a possible increase in area.

In this paper we present a novel systematic approach for generating folded architectures for FIR filters that reduce power consumption by minimizing the switching activity in the time-multiplexed multipliers of the architecture. With this technique, for large filters, we observe a trade-off between storage area and the power consumed in the multiplier units of the hardware architecture. In the rest of the paper a multiplication (addition) node or operation represents a filter DFG operation, a multiplier (adder) unit represents a hardware functional unit.

2. CAD METHODOLOGY FOR LOW POWER

The basic strategy employed consists of maximizing the correlation of successive inputs to shared computational resources. The primary computational resource targeted here are multipliers as the power consumed in multipliers is significantly higher than in adders which constitutes the other basic computational resource in folded filters. Also, the assignment of DFG operations (nodes) to resources is always done in such a manner that the interconnect overhead is minimized. Finally, as the input data-memory to filters in DSP applications can be very large in size a data-value is accessed exactly once from the data-memory. Once a data-value is accessed from memory all operations utilizing this value are scheduled successively one after the other.

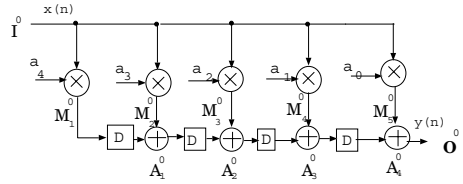

Figure 1. The transpose DFG of a 5-tap FIR filter. The data-input to the all the multiplier nodes is $x(n)$. Also, the filter connectivity described at the beginning of section 2.1 can be verified here.

There are two-inputs for a multiplier in a FIR filter, the coefficient input and the data input. The coefficient input to the multiplication nodes in the DFG of an FIR filter are fixed. Substructure sharing [12] can be used to reduce power consumption when a N-tap FIR filter is mapped to a N multiplier architecture. Substructure sharing for time-multiplexed or folded architectures may not always be straightforward. Due to the constraints of balancing the load on all hardware multipliers (the number of multiply nodes mapped to any multiplier are nearly equal) substructure sharing might be difficult. Also, substructure sharing is not applicable for programmable-coefficient filter implementations.

It is possible to find a scheduling order for the DFG multiplication nodes mapped to a functional unit so that the net switching activity at the coefficient input is minimized.

The data-input, however, is variable and can at best be modeled as a stochastic input which may or may not exhibit temporal correlation. On the other hand successive iterations of an FIR filter share a large number of data-inputs albeit in different multiplication nodes, i.e., in multiplication nodes with different coefficients. By unfolding [13] the DFG it is possible to unearth all the multiplication nodes which share a common data-input. In an N-tap FIR filter there are N multiplication nodes distributed across N successive iterations which share the same data-input.

However, the transpose DFG of an FIR filter shown in Fig. 1 is able to expose all these N multiplication nodes simultaneously without unfolding. We therefore use the transpose FIR filter DFG as a starting point for our synthesis methodology. Also, a k-unfolded version of the transpose FIR filter DFG exhibits k sets of N nodes where each set of N nodes has a common data-input, see section 2.1. This property becomes useful when we have k multiplier functional units at our disposal to implement the N tap FIR filter.

We next discuss k unfolding of a N-tap transpose FIR filter and then discuss low power folding of the unfolded FIR filter.

2.1. Unfolding the transpose FIR filter

The DFG of the N-tap transpose FIR filter in Fig. 1 has 1 input node, I^0, 1 output node, O^0, N multiplication operation nodes,

*This research has been supported by the Army Research Office under grant number DA/DAAG55-98-1-0315.

Proceedings of the Asia and South Pacific Design Automation Conference 2000 (ASP-DAC'00)
0-7803-5973-9/00 $ 10.00 © 2000 IEEE

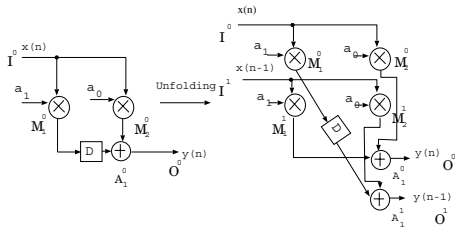

Figure 2. A 2-unfolded version of a 2-tap transpose FIR filter.

M_i^0 $i \in \{1, \cdots, N\}$ and $N-1$ addition operation nodes, A_i^0 $i \in \{1, \cdots, N-1\}$. Input node I^0 acts as a source to each of the N multiplication nodes with 0 delay, multiplication node M_1^0 is a source to adder A_1^0 with 1 delay. Also, multiplication node M_i^0 is a source to adder A_{i-1}^0 with 0 delay, $i \in \{2, \cdots, N\}$. Every adder A_i^0 feeds an input to adder A_{i+1}^0 with 1 delay for $i \in \{1, \cdots, N-2\}$, adder A_N^0 computes the filter output at node O^0. We illustrate our above observations with the 5-tap transpose FIR filter shown in Fig. 1. Unfolding a filter DFG, [13], by a factor of k will result in a new k-parallel filter topology which computes k filter outputs simultaneously. We now outline a strategy to unfold the N-tap transpose FIR filter by a factor k.

1) Create $k-1$ copies, I^j, O^j, M_i^j, A_i^j $j \in \{1, \cdots, k-1\}$ for the input I^0, output O^0, multipliers M_i^0 and adders A_i^0 respectively. As a result there will be k input nodes, $k \times N$ multiplier nodes and $k \times (N-1)$ addition nodes.

2) Connect the input I^j to multiplier M_i^j by an arc with 0 delays. Connect the multiplier M_1^0 to the adder A_{k-1}^1 by an arc with 1 delay. Connect the multiplier $M_1^j, j \in \{1, \cdots, k-1\}$, to adder A_1^{j-1} with 0 delays. Connect every other multiplier M_i^j $i \in \{2, \cdots, N\}$ to adder A_{i+1}^j by an arc with 0 delays. Connect the adder A_i^0 to adder A_{i+1}^{k-1} with 1 delay for $i \in \{1, \cdots, N-2\}$. Connect the adder A_i^j $j \in \{1, \cdots, k-1\}$ $i \in \{1, \cdots, N-2\}$ to the adder A_{i+1}^{j-1} by an arc with 0 delays. Connect each adder A_{N-1}^{k-1} to output O^j.

Fig. 2 shows the 2-unfolded version of a 2-tap transpose FIR filter. It can be proved [13] that the k-unfolded DFG of the FIR filter computes k iterations of the original filter simultaneously.

2.2. Low Power Assignment & Scheduling for Filter Nodes

When we have k hardware multiplier units, H_M^j, with p_M pipeline stages and k hardware adder units, H_A^j, with p_A pipeline stages, $j \in \{0, \cdots, k-1\}$, then we can perform scheduling and assignment as follows:

1) Unfold the transpose FIR filter by k.

i) Assign multiplication operations M_i^j to hardware multiplier unit H_M^j.

ii) Assign addition operations A_i^j to hardware adder unit H_A^j.

The above assignment of operation nodes in the unfolded filter to hardware units has the property that all multiplication operations assigned to a multiplier unit share a common input. In addition every multiplier unit is connected to exactly 1 input and to exactly 2 adder units. Also, every adder unit is connected to exactly 2 multiplier units and exactly 2 adder units. In other words the regularity of the original filter topology is almost completely maintained in the hardware mapping which leads to localized communication, minimum interconnect overhead, e.g., only 2-1 multiplexors are needed, and extremely uniform and compact layout.

2) Order the N-coefficients of the original FIR filter in such a manner that successive scheduling of the coefficients in this order results in minimum total switching activity. We approximate this step in the following manner:

i) Set up a graph $G = (V, E, D(E))$, where vertices $v \in V$ have 1-1 correspondence to coefficients of the original FIR filter. Arcs $e \in E$ exist between every pair of vertices with an associated distance $d(e) \in D(E)$. This distance between vertices, say vertex u and vertex v, is proportional to the average power dissipated when the coefficient corresponding to v is scheduled immediately after the coefficient corresponding to u. This distance measure can be obtained by simulation. Denoting, the coefficient corresponding to a vertex u by $c(u)$, we made a coarse approximation by taking $d(e) = d(u \rightarrow v) \approx Hamming\ Distance\ between\ c(u)\ and\ c(v)$.

ii) Find a cost-optimal Hamiltonian tour for the graph constructed in i). This tour gives the scheduling order for the various multiplication nodes. As might be evident the above problem can be solved as an instance of the Hamming Distance Traveling Salesperson Problem (TSP), [10]. Many efficient heuristics are available for approximately solving the Hamming Distance TSP. We used the C++ TSP-heuristics provided by Chad Hurwitz via e-mail: (churritz@cts.com).

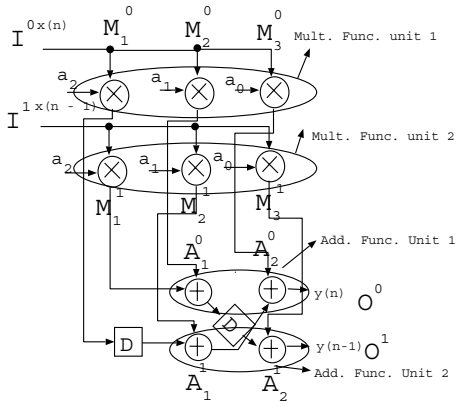

Figure 3. A 3-tap FIR filter is unfolded for mapping on to an architecture with 2 multipliers and 2 adders. While the 2 multipliers exhibit 100% hardware utilization the 2 adders exhibit 67% hardware utilization. As an aside we observe that unfolding the DFG and then folding it can lead to higher hardware utilization.

The optimal Hamiltonian-tour identified in ii) defines the periodic scheduling orders for the various multiplication operations in the FIR filter, i.e., the multiplication operations are scheduled one after the other on the functional unit according to this order and this ordered schedule is repeated every N clock cycles. In the context of folding the scheduling order is known as folding order and these two terms, from here onwards, will be used interchangeably.

There is significant freedom in choosing the folding order for the addition nodes. We experimented with several folding orders and found the empirical folding order given next to give the best results in minimizing storage. The folding orders for the adders is selected as follows: addition operation A_i^j has the folding order $(i + p_A)MOD(N)$. The reason for selecting this folding order for the addition operations, as mentioned earlier, is because this order gave good results in simulation for minimization of storage. Fig. 3 illustrates 2-unfolding of a 3-tap FIR filter for mapping to 2 multipliers and 2 adders.

2.3. Folding the unfolded transpose Filter

As mentioned before the Hamiltonian order from step 2) in the last section specifies a scheduling order (folding order) for the multiplication operations of the unfolded filter. Therefore, corresponding to each multiplication node M_i^j in the unfolded filter there corresponds a folding order m_i. Also, as explained in the last section every addition operation node A_i^j has the scheduling order $a_i = (i + p_A)MOD(N)$. Note that the scheduling orders for both multiplication and addition operations are independent of j; this is done to simplify the control circuitry. Recall that j is an index which identifies 1 among k functional units to which the DFG

Proceedings of the Asia and South Pacific Design Automation Conference 2000 (ASP-DAC'00)
0-7803-5973-9/00 $ 10.00 © 2000 IEEE

node is mapped. Hence, with this j-independent folding orders the same controller can be used for scheduling mapped DFG nodes in all the k functional units.

Due to the periodic nature of filtering operations we are able to choose a time-static (not changing with time) periodic schedule. Every multiplication operation M_i^j or addition operation A_i^j is scheduled exactly once in N clock cycles. The clock period is chosen as the worst case critical path of any pipeline stage among all the functional units. Due to the scheduling order chosen previously the k^{th} iteration of multiplier M_i^j is scheduled on multiplier unit H_M^j in clock cycle $Nk + m_i$. Due to p_M pipelining levels in multiplier H_M^j the output of the k^{th} iteration is available only in clock cycle $Nk + m_i + p_M$. This output is consumed by the k^{th} iteration of adder A_i^j which is scheduled at clock cycle $Nk + a_i$. The chosen schedule is feasible if and only if:

$$\begin{aligned} Nk + a_i - (Nk + m_i + p_m) \geq 0, \\ a_i - m_i - p_m \geq 0. \end{aligned} \quad (1)$$

Similar constraints can be derived for adder to adder arcs in the unfolded filter. For every arc in the unfolded filter such a corresponding constraint will be generated. Unfortunately, some of these constraints may not be satisfied in the original schedule. In order to facilitate the above schedule we resort to retiming, [14], [1], [2], [3]. Retiming alters the iteration structure of the original unfolded filter by moving delays around while maintaining the functionality of the original filter. In retiming an integer variable is associated with every operation node in the unfolded filter, e.g., $r(I^j)$ with input node I^j, $r(O^j)$ with output node O^j, $r(M_i^j)$ with multiply node M_i^j and $r(A_i^j)$ with addition node A_i^j. After retiming, the number of delays in a communication arc between two operation nodes, say from M_i^j to A_i^j, is changed by an amount $r(A_i^j) - r(M_i^j)$. It can be shown that this alteration of the DFG while changing the iteration structure maintains the functionality of the original algorithm, [14]. With this modification the $(k - (r(A_i^j) - r(M_i^j)))^{th}$ iteration of node A_i^j now consumes the output of the k^{th} iteration of node M_i^j. The constraint in (1) therefore gets altered to:

$$\begin{aligned} N(k - (r(A_i^j) - r(M_i^j))) + a_i - (Nk + m_i + p_m) \geq 0, \\ r(M_i^j) - r(A_i^j) \leq \frac{a_i - m_i + p_m}{N}, \\ r(M_i^j) - r(A_i^j) \leq \left\lfloor \frac{a_i - m_i + p_m}{N} \right\rfloor. \end{aligned} \quad (2)$$

The above inequality is referred to as the *retiming for folding* constraint for the arc from M_i^j to A_i^j. Similar constraints can be derived for all the other arcs in the unfolded filter. The storage required in the folded architecture can be modeled approximately as a linear function of the retiming variables. Since multiplication operations of the unfolded filter have exactly one outgoing arc the storage required for these can be modeled approximately as follows, [3],

$$Storage(M_i^j) = r(A_i^j) - r(M_i^j) + \frac{a_i - mi - p_m}{N}. \quad (3)$$

The terms independent of the retiming variables, $r(A_i^j)$ and $r(M_i^j)$ can be ignored as these are constant. Similarly we can derive linear functions for the storage required for the input node and for adders, [3].

In order to ensure that retiming maintains the common shared input of all the multiplication nodes mapped to the same functional unit, we require that for any such pair of multiplier nodes say M_l^j and M_i^j,

$$r(M_l^j) = r(M_i^j). \quad (4)$$

This constraint is referred to as the *low power folding* constraint.

The collection of the *retiming for folding* constraints and the *low power folding* constraints can be used to solve for minimizing the total storage required for folding the DFG using a *linear programming* (LP) formulation. We used a standard LP package GAMS for this purpose.

2.4. Low Power Folding with limited Unfolding

Unfortunately, unfolding by a factor of k leads to a k-fold increase in memory. Hence, it is not always practical to unfold a filter DFG by a factor of k for folding onto k multipliers, especially, when k is large. Thus, we need to develop a strategy for folding a transpose FIR filter DFG on to k multipliers without first unfolding the DFG. A compromise between the two techniques namely folding with and without k-unfolding can then obtained by performing limited unfolding as described later.

Let us assume that we have an N-th order transpose FIR filter to be folded onto hardware consisting of k multipliers. Assume for simplicity that N is divisible by k exactly.

1. As before set up the graph G = (V, E, D(E)), where vertices v $\in$ V have a 1-1 correspondence with the original FIR filter. Arcs $e \in E$ exist between every pair of vertices with an associated distance $d(e) \in D(E)$. As before these distances are a measure of the power dissipated when the coefficients corresponding to the two nodes are scheduled one after the other.

2. Now solve the minimum cost cycle partitioning problem [15] on this graph that breaks the N nodes into k clusters and orders the nodes within clusters in such a manner that the sum of distances in traversing the k cycles is minimized. The k clusters can be individually mapped to k different multipliers. The order of nodes within each cluster gives the minimum power schedule to be employed.

Note that in a 2-unfolded version of a N-tap FIR filter we can identify larger clusters of size $2 * N/k$ which have common inputs and can be mapped to the same multipliers. This will lead to a reduction in swicthing activity and hence power for the multipliers as compared to folding the original DFG, however, the memory requirements will roughly increase 2-fold.

Hence, it is desirable to find an optimal unfolding factor $1 \leq r \leq k$ for folding N multiplication operations on to k multipliers in such a way that overall power consumption is minimized. In order to do this the power savings obtained by reducing switching activity at the multiplier inputs must be weighed against the power expenditure due to the extra memory. This will help uncover an optimal unfolding factor.

2.5. Memory Assignment

We can use standard life-time analysis techniques [16] to compute minimal storage requirements for the folded DFG. Any number of memory allocation strategies [16] can be used for the actual allocation of memory resources. Due to the regular and repetitive nature of FIR filter computations it is possible to have near-static memory allocation in folded architectures. A near-static allocation involves minimal data-movement, i.e., once a variable is assigned to a memory location it requires minimal memory transfers for the duration of its lifetime. Reducing unnecessary data-movement in this way can lead to significant savings in power. As mentioned before, a near-static memory allocation is possible due to the repetitive nature of FIR filter computations and the use of counter based modulo addressing for the memory modules.

In the memory allocation scheme we follow there is a distinct memory module corresponding to each functional unit to store its intermediate output. Hence, only 1 functional unit performs a WRITE to any memory module. Due to the regularity of assignment of DFG operations to functional unit, exactly 2 functional units perform a READ from each memory module. This is because as mentioned in section 2 every hardware unit is connected at most with two other hardware units. Due to the regular nature of the connectivity it turns out that 2-port (1 READ and 1 WRITE) register files will always suffice for the memory modules.

3. SIMULATION RESULTS

We used our synthesis methodology to synthesize $N = 65$ and $N = 129$ tap filters on to architectures with a varying number of multipliers. The bandpass filters were obtained by using *fir1(N,*

Proceedings of the Asia and South Pacific Design Automation Conference 2000 (ASP-DAC'00)
0-7803-5973-9/00 $ 10.00 © 2000 IEEE

Table 1. Synthesis results comparing average power consumed in a multiplier in folding 65-tap, 129-tap bandpass FIR filters on to an architecture with a given number of multipliers and adders. Several scenarios are compared which include folding begining from a direct-form filter DFG and folding from a transpose filter DFG with varying unfolding factors to uncover common data-operands. Coefficient reordering is also resorted to in all the cases in an attempt to further reduce power consumption in the multipliers.

Filter DFG Type	Number mult. Taps	Number of multipliers available	Number of adders available	Unfolding factor	Storage Memory required	Avg. Power Consumed by a mult. (μW)	% power saved Saved in mult. over direct-form
Direct-form	129	15	15	1	134	11050	0%
Transpose	129	15	15	1	135	3000	72.85%
Transpose	129	15	15	3	401	2333	78.89%
Transpose	129	15	15	5	668	2200	80.09%
Transpose	129	15	15	15	1960	2069	81.28%
Direct-form	129	8	8	1	132	11100	0%
Transpose	129	8	8	1	134	2529	77.22%
Transpose	129	8	8	2	266	2264	79.60%
Transpose	129	8	8	4	540	2132	80.79%
Transpose	129	8	8	8	1040	2069	81.73%
Direct-form	65	10	10	1	67	11050	0%
Transpose	65	10	10	1	68	3285	70.27%
Transpose	65	10	10	2	139	2642	76.09%
Transpose	65	10	10	5	332	2257	79.57%
Transpose	65	10	10	10	662	2138	80.65%
Direct-form	65	22	22	1	68	11050	0%
Transpose	65	22	22	1	69	5000	54.75%
Transpose	65	22	22	2	135	3500	68.32%
Transpose	65	22	22	11	723	2272	79.44%
Transpose	65	22	22	22	1436	2138	80.65%

[0.1, 0.4]) in MATLAB.The windowing method used to obtain these filters was the Hamming window approach [17]. We obtained power consumption values for the multipliers in the architecture using the HEAT tool [18]. The data-input to the filter was taken as coming from an *independent identically distributed* process, with a switching probability for each bit of 0.5, i.e., white noise. We believe that due to the regularity of the architecture interconnect power will be minimal, and as long as the size of the storage units is small the contribution of the on-chip memory to power consumption will also be relatively insignificant. Reducing the power consumed in the multipliers may therefore lead to a significant reduction in power dissipation for the entire architecture. However, the memory required for low power folding grows linearly with the unfolding factor if unfolding is resorted to before folding. At some point the power consumed in the memory units may start to dominate. The multiplier used in our simulations was a 16×16 Booth-recoded Wallace-Tree multiplier. The results of our simulation are tabulated in Table 1. Table 1 presents a comparison of average power consumed in a multiplier for folding from a transpose FIR filter DFG with various unfolding factors for a given number of hardware multipliers and adders. Also, tabulated is the average power consumed in a multiplier for folding from a direct form FIR filter DFG with an unfolding factor of 1 (no unfolding).

The gains of low-power folding may reduce when the data-input to the filter has high correlation, e.g., when the data is from a highly correlated image.

4. CONCLUSIONS

Low power folding as presented in this paper can be used to reduce the power consumption of multipliers for folded FIR filter architectures. The technique presented here formalizes an intuitive notion that fixing one of the inputs of a multiplier and increasing the correlation of successive operands in the other input will bring down the power consumed in the multiplier. In future we will try to apply some of the ideas developed here for scheduling the filter multiplication operations on to the MACs of Programmable DSPs in an attempt to lower power consumption. Other possible applications for low power folding like IIR filters and FIR-LMS adaptive filters will also be explored.

REFERENCES

[1] K. K. Parhi, C. Y. Wang, and A. P. Brown, "Synthesis of Control Circuits in Folded Pipelined DSP architectures," *IEEE Journal of Solid State Circuits*, vol. 27, no. 1, pp. 29–43, 1992.

[2] T. C. Denk and K. K. Parhi, "Synthesis of Folded Pipelined Architectures for Multirate DSP Algorithms," *IEEE Transactions on VLSI Systems*, vol. 6, pp. 595–607, December 1998.

[3] V. Sundararajan and K. K. Parhi, "Synthesis of Folded Multidimensional DSP Systems," in *Proceedings of ISCAS-98*, (Monterey, CA, USA), June 1998.

[4] G. Goossens, J. Rabaey, J. Vandwalle, and H. De Man, "An efficient Microcode Compiler for Application Specific DSP Processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 9, pp. 925–937, September 1990.

[5] K. K. Parhi, *VLSI Digital Signal Processing, Design and Implementation*, ch. 6. New York: Wiley & Sons, 1999.

[6] A. P. Chandrakasan and R. W. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits," *Proceedings of the IEEE*, vol. 83, pp. 498–523, April 1995.

[7] R. Mehra, L. M. Guerra, and J. Rabaey, "Low Power Architectural Synthesis and the Impact of Exploiting Locality," *Journal of VLSI Signal Processing*, vol. 13, no. 2-3, pp. 239–258, 1996.

[8] R. Mehra and J. Rabaey, "Exploiting Regularity for Low-Power Design," in *Proc. IEEE Custom Integrated Circuits Conference*, pp. 401–404, May 1996.

[9] T. Arslan and A. T. Erdogan, "Data Block Processing for Low Power Implementation of Direct Form FIR Filters On Single Multiplier CMOS DSPs," in *Proceedings of ISCAS-98*, (Monterey, CA, USA), June 1998.

[10] K. Masselos *et al.*, "A Movel Methodology for Power Consumption Reduction in a Class of DSP Algorithms," in *Proceedings of ISCAS-98*, (Monterey, CA, USA), June 1998.

[11] A. Raghunathan and N. K. Jha, "Behavioral Synthesis for Low Power," *IEEE International Conference on Computer Design*, pp. 318–322, October 1994.

[12] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination," *IEEE Transactions on Computer-Aided Design of Integrated Circ uits and Systems*, vol. 15, pp. 151–165, February 1996.

[13] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data flow programs via optimum unfolding," *IEEE Trans. Computers*, vol. 40, pp. 178–195, Feb. 1991.

[14] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Systems," *Journal of VLSI and Computer Systems*, vol. 1, no. 1, pp. 11–67, 1983.

[15] M. X. Goemans and D. P. Williamson, "A General Approximation Technique for Constrained Forest Problems," *SIAM Journal on Computing*, vol. 24, pp. 296–317, April 1995.

[16] K. K. Parhi, "Systematic synthesis of DSP data format converters using life-time analysis and forward-backward register allocation," *IEEE Trans. Circuits And Systems II Analog and Digital Signal Processing*, vol. 39, pp. 423–440, July 1992.

[17] A. V. Oppenheim and R. W. Schafer, *Discrete-time signal processing*. Prentice Hall, 1992.

[18] J. H. Satyanarayana and K. K. Parhi, "HEAT: Hierarchical Energy Analysis Tool.," *33rd Design Automation Conference*, pp. 9–14, June 1996.

Proceedings of the Asia and South Pacific Design Automation Conference 2000 (ASP-DAC'00)
0-7803-5973-9/00 $ 10.00 © 2000 IEEE