Power Optimization in Programmable Processors and ASIC Implementations of Linear Systems: Transformation-based Approach

ABSTRACT

Linear computations form an important type of computation that is widely used in video and image processing, DSP, control, communications, and many other applications. With the ongoing rapid proliferation of portable computation and communication, power minimization has been gaining importance as a crucial design metric. However, while approaches for optimal optimization of throughput and joint optimization of latency and throughput in linear computations are available, until now no approach has been proposed which efficiently optimizes power.

We introduce two approaches for power minimization in linear computations using transformations. First we show how unfolding combined with the procedure for maximally fast implementation of linear computations reduces power in single processor and multiprocessor implementations by factors 2.7 and 15.6 respectively, the former with no hardware penalty.

For the custom ASIC implementation even higher improvements are achievable using the second transformational approach, which builds upon the unfolding based strategy of the first approach. We developed a method that combines the multiple constant multiplication (MCM) technique with the generalized Horner's scheme and unfolding in such a way that an arbitrary linear computation can be computed under an arbitrary throughput constraint with a very low power. The approach can be also used to efficiently and simultaneously optimize any combination of four important design metrics (throughput, latency, power, and area) of linear computations.

The effectiveness of our techniques and algorithms is demonstrated on numerous real-life examples from DSP, control, and communications for both programmable platforms and custom ASIC implementation style.

1.0 Throughput and Power in Linear Systems

1.1 Motivation

Linear computations form an important type of computation that is widely used in video and image processing, DSP, control, communications, and many other applications. A large fraction of systems in these application domains are either linear, or have subsystems that are linear over field F, an additive operator $\oplus$, and a multiplicative operator $\otimes$. Typically, F is the field over real numbers $\Re$, $\oplus$ is arithmetic +, and $\otimes$ is arithmetic *, as is the case in linear filters in DSP, linear process controllers etc. It is important to note that linearity is not restricted to arithmetic + and *. For example, many useful systems, such as in fuzzy control, exhibit linearity over *min* or *max* as the additive operator, and arithmetic addition + as the multiplicative operator.

In recent years two fundamental properties of implementations of linear computations have been identified. The first is that an arbitrary linear computation can be implemented at an arbitrarily high throughput [Pot91]. The second is that there is a fundamental limit on the joint optimization of throughput and latency, the two metrics of speed. Srivastava and Potkonjak [Sri94] showed that it is impossible to reduce latency and increase throughput to an arbitrary level at the same time, and provided analytical formulas for the best feasible combinations of latency and throughput.

This paper explores the relationship of throughput with the third, and increasingly important, design metric - power consumption. In particular, we seek to find the extent to which power consumption of linear systems can be reduced, both independently and in conjunction with throughput improvement, and to develop techniques for doing so. Chandrakasan et. al. [Cha92b] made an attempt to utilize the approach of achieving an arbitrarily high throughput implementation of linear computation described in [Pot91] to lower power by trading-off increased throughput with reduced voltage at the cost of additional hardware. Unfortunately, the hardware overhead was so high that it overshadowed the savings due to voltage-throughput trade-off. In several linear filter examples the overall power consumption started to increase after relatively small throughput increase [Cha92b].

To explore the throughput and power relationship in linear systems we take a more thorough and systematic approach. First, we consider analytically as well as empirically the effect of several algorithm transformations that can be considered as the building-blocks for exploring the power-throughput space. Specifically, we consider unfolding, which is the underlying transformation behind arbitrary throughput improvement, both separately and in combination with decomposition of multiplication-by-constants into primitive sequences of shifts and additions, factorization, common sub-expression elimination. Second, we consider implementations not just in the form of ASICs with application-specific datapaths as is usually the case, but also implementations based on single programmable processor and multiple programmable processors. This is important because increasingly programmable processors such as DSP-cores have emerged as the preferred medium of implementation as opposed to custom datapaths.

1.2 Where does the Power Go, and How to Reduce It?

Prediction of power consumption of a computation at the behavioral level is a difficult task. In CMOS technology there are three sources of power consumption: switching current (dynamic power), short-circuit current, and leakage currents. The switching component not only dominates in most designs, but is also the only one which cannot be made negligible even when proper circuit design techniques are used - architectural techniques therefore become important in reducing the switching component of power consumption. The average power consumption of a CMOS gate due to the switching component is given by:

$$P = \alpha C_L V_{dd}^2 f \tag{EQ 1}$$

where f is the system clock frequency, V_{dd} is the supply voltage, C_L is the load capacitance, and α is the switching activity (the probability of a $0 \rightarrow 1$ transition during a clock cycle). The term αC_L is often lumped into a single parameter called effective switched capacitance. Further, it is well known that the delay through a gate has a monotonically inverse relationship to the supply voltage. Fig. 1 plots the gate delay as a

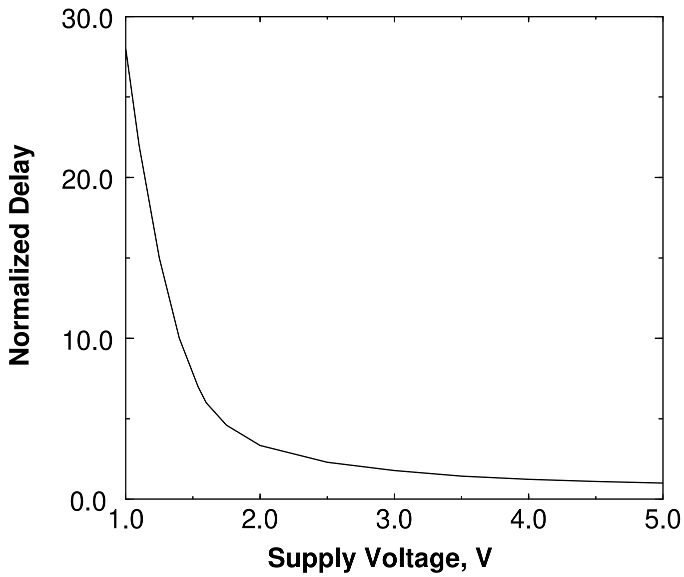

Figure 1: Normalized Gate Delay (delay @ 5V = 1 unit) vs. Supply Voltage

function of voltage normalized to gate delay at 5.0 Volts. Therefore, the maximum rate at which a circuit can be clocked will monotonically decrease as the voltage is reduced.

The above expression suggests several behavior level approaches to reduce power consumption of a computation. First is to shut the system down during periods of inactivity either by shutting off the clock ($f = 0$) or by shutting off the power supply ($V_{dd} = 0$). Second is to reduce the effective switched capacitance αC_L by restructuring the computation, communication, and/or the memory hierarchy, and by changing data encoding and a data formats. The third is to exploit the quadratic dependence of power consumption upon the supply voltage V_{dd} and operate at a reduced voltage while compensating for the resulting loss in circuit speed by techniques that increase the throughput.

While the first technique is not directly relevant to us because we are dealing with computation that is inherently "continuous" in nature as opposed to being event driven, the latter two are of direct relevance. Noting that throughput is the sole metric of speed that is important to us, one can combine the latter two approaches to minimize power at the behavioral level in the following fashion: First, apply a behavior transformation to reduce the effective switched capacitance (by reducing the number of operations) or to increase the throughput (by reducing the critical path). Note that these two goals are often interdependent, for example, improvement in throughput often comes at the cost of worsening of the effective switched capacitance. Next, in the case of increased throughput, we lower the supply voltage just the right amount so as to decrease the clock speed to an extent that the throughput reverts back to what it was before. Of course, the supply voltage reduction is limited by technology considerations. The net power consumption is reduced if either the effective switched capacitance is reduced at a constant voltage, or if the reduction due to reduced voltage and frequency overshadows any increase effective capacitance penalty paid for increase in throughput. Finally, it must be noted that at some point further voltage reduction will not be technically feasible, for example because the supply voltage must be greater than threshold voltage in CMOS. When such a point is reached, one can trade-off extra throughput obtained via behavior level transformations with lower clock frequency or with shutdown, both of which will result in linear reduction.

1.3 Linear Systems

In high-level synthesis terminology, a system is linear if it can be realized by a control-dataflow graph (CDFG) such that at any time instant n, the output samples and the next state values are computed by linear combinations of input samples and previous state values. Equivalently, all the operators in the CDFG are either addition of two variables, or addition of a variable and a constant, or multiplication of a variable and a constant. For analysis in this paper we shall use an equivalent representation of the linear computation as a set of discrete-time finite-dimensional state-space equations. Specifically, a P input, Q output, and R state real-valued linear computation can be expressed by the following matrix equations:

$$\begin{aligned} S[n] &= AS[n-1] + BX[n] \\ Y[n] &= CS[n-1] + DX[n] \end{aligned} \quad \text{(EQ 2)}$$

$$n \in \{0, 1, 2, 3, \ldots\} \qquad X[n] \in \Re^{P \times 1} \qquad S[n] \in \Re^{R \times 1} \qquad Y[n] \in \Re^{Q \times 1}$$

$$A \in \Re^{R \times R} \qquad B \in \Re^{R \times P} \qquad C \in \Re^{Q \times R} \qquad D \in \Re^{Q \times P} \qquad S[-1] \in \Re^{Rx1}$$

where X, Y, and S are the input, output, and state vectors respectively, and A, B, C, and D are constant coefficient matrices. Note that the throughput of the system, or the maximum rate at which it can process incoming samples, is decided solely by the critical path of the feedback section corresponding to the term $S[n] = AS[n-1]$. The remaining terms are not in the feedback loop and therefore can be pipelined away.

This is the base or reference case for our analysis, and has following characteristics:

$$\#\text{ of muls} = (R+P)(R+Q)$$

$$\#\text{ of adds} = (R+P-1)(R+Q)$$

$$\text{feedback critical path} = m + \lceil \log_2(1+R) \rceil$$

and,

$$\text{max throughput} = \frac{1}{\text{feedback critical path}} = \frac{1}{m + \lceil \log_2(1+R) \rceil}$$

For the critical path we assumed that the time for an addition is 1, and the time for a multiplication is $m \geq 1$ which is a reasonable assumption in almost all cases, whether on an ASIC or on a programmable processor. The expressions are obtained by considering the worst case where the coefficient matrices A, B, C, and D are dense matrices with arbitrary non-trivial coefficients (not 0, or 1, or -1). The maximum throughput expression is obtained by organizing the required linear combinations in a maximally fast fashion by first doing the constant-variable multiplications in parallel, and then the additions in a maximally balanced binary tree.

1.4 Paper Organization

The remainder of the paper is organized in the following way. In the next section the related work is surveyed. The generic approach for power optimization in linear systems is introduced in Section 3. In Sections 4 and 5 the effect of unfolding and other transformations on power-throughput trade-off in programmable implementations is discussed. In Section 6 we introduce a transformation technique for power minimization in ASIC implementations of linear computations. After showing in Section 7 how single input single output linear computations can be even further optimized, we summarizes the contributions in the last section.

2.0 Related Work

The related work can be traced along two lines of research: transformations and low power CAD and compilation techniques. Transformations are widely used technique to improve variety of design parameters during high level synthesis [Wal89, Pot91, Ku92].

While power optimization was rarely addressed in the past (e.g. [Gla84]), with rapidly growing market of portable computation and communication products, recently it attracted a great deal of attention in design and CAD communities. Although the methods for power optimization at all level of design process has been proposed [Cha92a], the most numerous research efforts has been reported in logic synthesis. Power minimization effort across all level of design abstraction process are surveyed in [Sin95].

Interestingly, although it is apparent that the highest potential for power reduction is at higher levels of design process and although the first CAD approach for power minimization was proposed in high level synthesis [Cha92c], few efforts for power optimization at the behavioral level has been reported.

Chandrakasan et al. [Cha92] demonstrated the effectiveness of transformations by showing an order of magnitude reduction in several DSP computationally intensive examples using simulated annealing-based transformational script. More recently Raghunathan and Jha [Rag94] and Goodby et al. [Goo94] also proposed methods for power minimization which explore trade-offs between voltage scaling, throughput, and power. Chatterjee and Roy [Cha94b] targeted the power reduction in fully hardwired design by minimization switching activity. Duncan et al. used retiming to reduce power dissipation in DSP designs [Dun93]. Recently, several research groups used trade-offs related to background memory design to achieve substantial power reduction [Bun94, Wuy94].

Chandrakasan et. al [Cha94a] and Tiwari et al [Tiw94] did pioneering work in power minimization and optimization when programmable platforms are targeted.

3.0 Effect of Unfolding and Other Transformations on Power-Throughput Trade-Of

As mentioned earlier, reduction in number of operations (and thereby in effective switched capacitance) and the trading-off of reduced voltage with increased throughput obtained via behavior-level techniques are two basic approaches to behavior level power reduction. Unfolding, decomposition of multiplication-by-constants into primitive sequences of shifts and additions, factoring, and common sub-expression elimination are all behavior level transformations that can increase throughput or reduce number of operations in linear computations, and are therefore crucial to power reduction of linear systems.

3.0.1 Unfolding-Driven Voltage-Throughput Trade-Off for Power Reduction

Unfolding together with block processing or with look-ahead has been shown by various researchers [Par89] to be effective in obtaining arbitrary improvement in throughput. Essentially, several consecutive input samples are processed together as a batch to produce a batch of output samples together with the input state value for the next batch. In the original non-unfolded computation $X[n]$ and $S[n-1]$ are used to compute $Y[n]$ and $S[n]$. In a system that has been unfolded i times, a batch of $i+1$ input samples $X[n]\ldots X[n+i]$ together with $S[n-1]$ is used to compute a batch of output samples $Y[n]\ldots Y[n+i]$ and the next state $S[n+i]$. The batch processing itself can be done in various ways: in a block processing [Rob21] fashion where the batch processing is begun only after all the input samples have been collected in a buffer, or in an on-arrival processing [Sri94] fashion where batch processing is begun as soon as the first relevant data is available. Independent of how the processing is organized, the basic computation executed by a linear system that has been unfolded i times can be represented by the following state-space equations:

$$\begin{aligned} S[n+i] &= A^{i+1}S[n-1] + A^iBX[n] + A^{i-1}BX[n+1] + \ldots + BX[n+i] \\ Y[n] &= CS[n-1] + DX[n] \\ Y[n+1] &= CAS[n-1] + CBX[n] + DX[n+1] \\ &\ldots \\ Y[n+i] &= CA^iS[n-1] + CA^{i-1}BX[n] + CA^{i-2}BX[n+1] \ldots + CBX[n+i-1] + DX[n+i] \end{aligned} \quad \textbf{(EQ 3)}$$

Note that these equations process $i+1$ data samples for each execution. For the i times unfolded system we get the following characteristics:

$$\#(*, i) = \text{\# of muls for } i \text{ times unfolded system} = R^2 + (i+1)PR + (i+1)QR + \frac{(i+1)(i+2)}{2}PQ$$

$$\#(+, i) = \text{\# of adds for } i \text{ times unfolded system} = R^2 + (i+1)PR + (i+1)QR + \frac{(i+1)(i+2)}{2}PQ - R - (i+1)Q$$

$$\mathrm{CP}(i) = \text{feedback critiical path for } i \text{ times unfolded system} = m + \lceil \log_2(1+R) \rceil$$

$$\mathrm{MaxThroughput}(i) = \text{max throughput for } i \text{ times unfolded system} = \frac{(i+1)}{m + \lceil \log_2(1+R) \rceil}$$

As expected, when $i = 0$ the above equations reduce to those for the unfolded case presented earlier. Further, the maximum achievable throughput is arbitrarily increased as the amount of unfolding increases because the feedback critical path remains the same while more samples are processed.

An interesting observation is that the effective number of operations per input sample is lower in the unfolded case when the amount of unfolding i is less than a certain threshold. In particular,

$$\begin{aligned} &\frac{\#(*, i)}{i+1} - \#(*, 0) = \text{\# of muls per sample with unfolding } - \text{\# of muls per sample without unfolding} \\ &= -\frac{i}{i+1}R^2 + \frac{i}{2}PQ \\ &< 0 \qquad \text{for } i < \left(\frac{2R^2}{PQ} - 1\right) \end{aligned} \quad \textbf{(EQ 4)}$$

and,

$$\frac{\#(+,i)}{i+1} - \#(*,0) = \text{\# of adds per sample with unfolding } - \text{\# of adds per sample without unfolding}$$

$$= -\frac{i}{i+1}R(R-1) + \frac{i}{2}PQ \qquad \textbf{(EQ 5)}$$

$$< 0 \qquad \text{for } i < \left(\frac{2R(R-1)}{PQ} - 1\right)$$

It should also be noted that above expressions for differences in numbers of * and + per sample achieve minimum at certain i below the shown thresholds - in other words, as one unfolds, the number of operations per sample at first decreases to reach a minimum and then begins to rise.

The above lead to the following strategies for low power implementations.

4.0 Implementation on a Single Programmable Processor

In the case of a single programmable processor the throughput that is achieved is solely decided by the number of operations (length of schedule = # of instructions in case of one processor). It follows that the throughput is maximized by using the value of i that minimizes the total number of instructions. If one assumes that + and * are the basic processor instructions (they need not take the same number of cycles), it can be shown that the optimum value of unfolding i_{opt} is one of the following two values:

$$i_{opt} = \left\lfloor \sqrt{\frac{(2R-1)R}{PQ}} - 1 \right\rfloor \qquad \text{or} \qquad \left\lceil \sqrt{\frac{(2R-1)R}{PQ}} - 1 \right\rceil$$

depending on whichever leads to a smaller value of $i_{opt}\left(PQ - \frac{R(2R-1)}{i_{opt}+1}\right)$. If both lead to same value of $i_{opt}\left(PQ - \frac{R(2R-1)}{i_{opt}+1}\right)$, we pick the smaller i_{opt} so as to save on coefficient memory because larger unfolding leads to more constant coefficients.

From this one can obtain the following expression for maximum improvement in throughput for the single processor case:

$$S_{\max} = \text{max improvement in single processor throughput by unfolding}$$
$$= (i_{opt}+1)\frac{\#(*,0) + \#(+,0)}{\#(*,i_{opt}) + \#(+,i_{opt})}$$

Finally, the processor voltage can be reduced just the right amount so that the clock frequency f is reduced by a factor of $S_{\max}$. This leads to a reduction in processor power because in the expression for power $P = \alpha C_L V_{dd}^2 f$ the terms V_{dd}^2 and f are reduced whereas the other two terms remain constant. It must be mentioned that in we are implicitly assuming that the processor power consumption is dominant compared to coefficient and data memory power consumption, an assumption that is true in most CPU-memory systems as found in DSP and control processing systems.

As an example, consider a hypothetical linear computation with $P = 1$ input, $Q = 1$ output, $R = 12$ states. Then, from the approach above one can show that $i_{opt} = 16$ which leads to $S_{\max} = 4.075$. One can therefore reduce the voltage such that the clock frequency is reduced by a factor of 4.0007 so that the throughput reverts back to the original throughput. If the initial voltage is 3.0V, then from Fig. 1 it follows that the voltage reduction to 1.5V will result in the desired clock slowdown. The processor operating at 1.5V and computing the equations that have been unfolded 16 times will have the same throughput as the processor operating at 3.0V and computing the initial non-unfolded equations. However, in the unfolded case one obtains a power reduction of $\left(\frac{3.0}{1.5}\right)^2 \times \frac{1}{(1/4)}$, or a factor of 16 over the initial power consumption. If the initial voltage was 5.0V, then our technique will result in a processor operating at 1.9V, with an even larger power reduction of $\left(\frac{5.0}{1.9}\right)^2 \times \frac{1}{1/4} = 27.7$.

As mentioned earlier, the above result is based on analysis that assumed that the coefficient matrices A , B , C , and D are dense matrices with arbitrary non-trivial coefficients (not 0, or 1, or -1). While this is certainly true of linear systems that are found in process controllers, it is often not the case with filters found in DSP applications where these matrices often tend to be sparse and have coefficients that are trivial (for example, coefficients of 1 or -1 do not need multiplication). Unfortunately, it is not possible to come with meaningful analytical expressions for the non-dense case.

However, we have empirically found that unfolding helps in reducing the number of operations and the power even in such cases - although by smaller factors. The only problem is that the optimum level of unfolding and the number of operations can no longer be found by merely evaluating closed-form formulas. We therefore use the following heuristic to find the desired level of unfolding in the non-dense cases: first pick the best performing level of unfolding form amongst all values of i from 0 through i_{opt}, the optimum value analytically predicted for the dense case. If the best level turns out to be i_{opt}, then we continue to unfold further as long as the number of operations continues to decline.

In case unfolding results in such a large increase in throughput (reduction in number of operations) that even after reducing the voltage to the minimum feasible (about 1V in the technology that we used) the new system has higher throughput than the original, then one can obtain a further reduction in power by operating the processor at an even lower frequency (or, equivalently, by shutting the processor for part of the time). This, however, did not happen for any of our examples.

The following results summarize the power reduction obtained for several real-life examples listed in Table 1. In Table 2 we give the empirically obtained numbers for the actual coefficients from the examples, as well as analytically predicted numbers for dense coefficient matrices with same values of P , Q , and R as in the real-life examples. Some of the examples (*mat*, *ellip*, *lin4*, *lin5*, *and steam*) had dense coefficient matrices so that the power reduction obtained is the same as for the dense coefficient matrices case. For three examples (*iir*, *dist*, and *aircraft*) no power reduction is obtained. The average processor power reduction for the example suite is x2.3 when the initial voltage is 3.0V, and x3 when the initial voltage is 5.0V, thus showing that our approach of power reduction using unfolding-driven throughput-voltage trade-off is effective for single programmable processor implementations. It must be emphasized that this power reduction for single-processor implementation is being obtained with no processor area penalty. While the coefficient memory requirements do increase, memories come in sizes that are powers of 2 - therefore in most cases there will be no impact on memory size either.

Interestingly, note that even if voltage reduction is not an option due to hardware and technology constraints, the increased throughput yielded by optimal unfolding can be traded-off against reduced clock frequency for a linear reduction in power consumption. This linear reduction is substantial, though smaller than what unfolding combined with voltage reduction offers. Instead of clock frequency reduction, one may also use shutdown by stopping the clock or making supply voltage zero for part of the sample period. For example, optimal level of unfolding in the *iir12* design in the table above yields a x1.59 reduction in the number of

Design Name	Description	P	Q	R
mat	3-state 1-input linear controller	1	1	3
ellip	4-state 1-input linear controller	1	1	4
lin4	5-state 1-input linear controller	1	1	5
lin5	5-state 1-input linear controller	1	1	5
iir5 (wdf5)	5th order low pass elliptic wave digital IIR filter	1	1	5
iir6	6th order lowpass elliptic cascade IIR filter	1	1	6
iir8	8th order bandpass direct form IIR filter	1	1	8
iir10	10th order bandstop Butterworth cascade IIR filter	1	1	10
iir11	11th order low pass Chebyshev cascade IIR filter	1	1	11
iir12	12th order bandpass Chebyshev cascade IIR filter	1	1	12
steam	power plant controller	1	1	5
dist	distillation plant linear controller	2	1	5
chemical	chemical plant controller	2	1	4
aircraft	6 knot linear plant controller for VSTOL aircraft	3	3	14

Table 1: Description of the Example Suite

operations. Therefore, the clock frequency can be reduced by x1.59, resulting in a power reduction by x1.59 (or, 37%) while the processor voltage remains unchanged. This strategy gives an average power reduction of x1.44 (31%) over all our examples.

5.0 Implementation on Multiple Programmable Processor

Potentially more savings can be obtained using our approach if one considers implementations that are not restricted to a single processor. By using more than one processors the throughput achieved by the implementation can be reduced compared to the single processor case, and by using enough processors the maximum possible throughput (decided by the critical path through the feedback portion of the linear computation) can be achieved. The extra throughput thus obtained can be used for further throughput-voltage trade-off as long as the power reduction from this compensates for the power increase due to more processors.

As an example, consider the same hypothetical linear computation with $P = 1$ input, $Q = 1$ output, $R = 12$ states, and dense coefficient matrices that we considered for the single processor case. Previously we had shown that the number of operations per sample is minimized when the linear computation is unfolded for $i_{opt} = 16$ times, and that the maximum throughput achieved by a single processor relative to original non-unfolded implementation is $S_{\max}(1) = (i_{opt}+1)\frac{\#(*,0)+\#(+,0)}{\#(*,i_{opt})+\#(+,i_{opt})} = 4$. Now, if a second processor is added, the throughput will increase by x2 (ignoring communication costs), and at the same time power consumption will increase by x2 due to the addition of the second processor. Now, one can reduce the voltage such that the clock frequency of both the processors is reduced by $S_{\max}(1) = 2x4 = 8$. If the initial voltage was 3.0V, then this reduced voltage (from Fig. 1) is given by 1.27V. Therefore, the 16-unfolded two-processor 1.27V implementation will have a power reduction of $\left(\frac{3.0}{1.27}\right)^2 \times \frac{1}{1/8} \times \frac{1}{2} = 22.3$ relative to an non-unfolded 3.0V single-processor implementation. This improvement of about x22 is much greater than the x16 obtained if one were to restrict to a single processor only.

In general the situation is more complex when adding processors. First, addition of processors causes a linear increase in switched capacitance, and hence power, for a given voltage and clock frequency. In fact, the

P	Q	R	For Dense Coefficient Matrices							Using Coefficient Matrices from Real Examples							
			Initial		After Optimal Unfolding					De-sign	Initial		After Heuristic Unfolding				
			# Op	V	i	# Ops	V	Frq Red	Pwr Red		# Op	V	i	# Ops	V	Frq Red	Pwr Red
1	1	3	28	3.0 5.0	3	19.75	2.4 3.6	1.42	2.2 2.7	mat	28	3.0 5.0	3	19.75	2.4 3.6	1.42	**2.2** **2.7**
1	1	4	45	3.0 5.0	4	26.6	2.2 3.1	1.69	2.3 2.7	ellip	45	3.0 5.0	4	26.6	2.2 3.1	1.69	**2.3** **2.7**
1	1	5	66	3.0 5.0	6	33.43	2.0 2.8	1.97	4.4 6.3	lin4	66	3.0 5.0	6	33.43	2.0 2.8	1.97	**4.4** **6.3**
										lin5	66	3.0 5.0	6	33.43	2.0 2.8	1.97	**4.4** **6.3**
										wdf5	32	3.0 5.0	3	27.25	2.7 4.2	1.17	**1.4** **1.7**
										steam	66	3.0 5.0	6	33.43	2.0 2.8	1.97	**4.4** **6.3**
1	1	6	91	3.0 5.0	7	40.25	1.9 2.5	2.26	5.6 9.0	iir6	40	3.0 5.0	5	36	2.8 4.5	1.11	**1.3** **1.4**
1	1	8	153	3.0 5.0	10	53.91	1.7 2.2	2.84	8.8 14.7	iir8	34	3.0 5.0	0	34	3.0 5.0	1	**1** **1**
1	1	10	231	3.0 5.0	13	67.57	1.6 2.0	3.42	12.0 21.4	iir10	85	3.0 5.0	9	59.8	2.4 3.6	1.42	**2.2** **2.7**
1	1	11	276	3.0 5.0	14	74.4	1.6 1.9	3.71	13.0 25.7	iir11	102	3.0 5.0	10	66.36	2.3 3.4	1.54	**2.6** **3.3**
1	1	12	325	3.0 5.0	16	81.23	1.5 1.9	4.00	16.0 27.7	iir12	114	3.0 5.0	11	71.83	2.2 3.3	1.59	**3.0** **3.7**
2	1	5	78	3.0 5.0	4	50	2.3 3.3	1.56	2.7 3.6	dist	48	3.0 5.0	3	47.25	3.0 5.0	1.02	**1** **1**
2	1	4	55	3.0 5.0	3	40	2.4 3.7	1.38	2.2 2.5	che-mical	41	3.0 5.0	2	33	2.6 4.0	1.24	**1.6** **1.9**
3	3	14	561	3.0 5.0	5	291	2.0 2.9	1.93	4.3 5.7	air-craft	136	3.0 5.0	0	136	3.0 5.0	1	**1** **1**

Table 2: Power Reduction in a Single Programmable Processor using Unfolding-Driven Voltage-Throughput Trade-off

increase in switched capacitance may be super-linear due to inter-processor communication hardware. Second, the speed-up due to an additional processor is not linear, and begins to saturate due to inter-processor communication overhead. Even if the inter-processor communication cost is ignored, the computation cannot be speeded up more than that allowed by the critical path of the feedback section. Finally, the voltage cannot be reduced below a certain point.

The following analytic approach, developed under certain simplifying assumptions, explores the unfolding-driven power-throughput trade-off in implementations using multiple processors. The simplifying assumptions are (i) inter-processor communication does not cost any time, (ii) effective switched capacitance αC_L increases linearly with the number of processor N, (iii) voltage cannot be reduced below 1V, and (iv) both addition and multiplication instructions take one clock cycle (i.e. $m = 1$).

The first step, as in the single processor case, is to unfold the linear computation to the optimum level $i = i_{opt}$ where the number of operations (instructions) per sample is minimized. The second step, is to increase the number of processors to N. Let $S_{\max}(N, i)$ be the maximum improvement in throughput achieved by N processors on an i times unfolded linear computation compared to a single processor on the original non-unfolded computation. The third step is then to slow-down each of the N processor by a factor of $S_{\max}(N, i)$ - this is done by reducing the voltage just the right amount (but limited by the technology-imposed lower bound) so as to decrease the clock frequency (increase the gate delay) by $S_{\max}(N, i)$. Let $V(d)$ be the voltage at which the value of gate delay relative to the initial implementation is d, with $V(1)$ typically being 3.0V or 5.0V. Then, the power of the new N processor implementation relative to the original non-unfolded implementation is:

$$Power(N) = \left(\frac{V(1)}{V(S_{\max}(N, i_{opt}))}\right)^2 \times \frac{S_{\max}(N, i_{opt})}{N}$$

The task is to find the optimum value $N = N_{opr}$ where the above expression is minimized. The crucial missing piece of the puzzle is an estimate of $S_{\max}(N, i)$, the maximum improvement in throughput achieved by N processors on an i times unfolded linear computation. It can be shown via some intricate algebraic manipulation that under our simplifying assumptions,

$$S_{\max}(N, i) = \frac{(i+1)\ (\#(*, 0) + \#(+, 0))}{\left(\frac{\#(*, i) + \#(+, i)}{N}\right)} = NS_{\max}(1, i) \qquad \text{for } (N \le R)$$

$$S_{\max}(N, i) = \frac{(i+1)\ (\#(*, 0) + \#(+, 0))}{\max\left(\left(\left\lceil \log_2(1 + \frac{N}{R})\right\rceil + \left\lceil \frac{R + 1 - 2^{\lceil \log_2(1 + N/R)\rceil}}{N/R}\right\rceil\right), \frac{\#(*, i) + \#(+, i)}{N}\right)} \qquad \text{for } (R < N < R(R+1))$$

and,

$$S_{\max}(N, i) = \frac{(i+1)\ (\#(*, 0) + \#(+, 0))}{\max\left((1 + \lceil \log_2(1 + R)\rceil), \frac{\#(*, i) + \#(+, i)}{N}\right)} \qquad \text{for } (N \ge R(R+1))$$

As is obvious, the speed-up due to multiple processors is linear for $N \le R$, which will allow a linear decrease in frequency, and therefore power, and thus offset the linear increase in power due to increase in number of processors. Therefore, one can always add up to R processors and get a reduction in power due to the reduction in the voltage term. In other words, the optimum number of processors is at least R.

While the above expressions are true only for dense coefficient matrices, the observation that the speed-up is linear for $N \le R$ is valid even for real-life non-dense coefficient matrices in a slightly modified form: the speed-up will be *at-least* linear (under our assumption of zero communication cost). We exploit this fact, and conservatively use $N = R$ processors to get at least a linear increase in throughput (on top of what i_{opt} level unfolding alone gives) and trade this increased throughput with a voltage reduction to slow down the clock by an equivalent amount. Table 2 shows the resulting power reduction for our suite of examples.

6.0 Implementation on Custom Datapath ASICs

The case of implementation of linear computation as ASICs with custom datapaths with multiple function units is in some ways similar to the multiple programmable processor case, except that meaningful analysis is

Design	P	Q	R	Init. V	Unfolding + Single Processor (from Table 2)			Unfolding + Multiple Processors			
					V	Frq Red	Pwr Red	# of Procs	V	Frq Red	Pwr Red
mat	1	1	3	3.0 5.0	2.4 3.6	1.42	**2.2** **2.7**	3	1.5 1.8	4.26	**5.7** **10.9**
ellip	1	1	4	3.0 5.0	2.2 3.1	1.69	**2.3** **2.7**	4	1.3 1.6	6.76	**9.0** **16.5**
lin4	1	1	5	3.0 5.0	2.0 2.8	1.97	**4.4** **6.3**	5	1.2 1.4	9.85	**12.3** **25.1**
lin5					2.0 2.8	1.97	**4.4** **6.3**	5	1.2 1.4	9.85	**12.3** **25.1**
iir5/wdf5					2.7 4.2	1.17	**1.4** **1.7**	5	1.4 1.6	5.85	**5.4** **11.4**
steam					2.0 2.8	1.97	**4.4** **6.3**	5	1.2 1.4	9.85	**12.3** **25.1**
iir6	1	1	6	3.0 5.0	2.8 4.5	1.11	**1.3** **1.4**	6	1.3 1.6	6.66	**5.6** **10.8**
iir8	1	1	8	3.0 5.0	3.0 5.0	1	**1** **1**	8	1.3 1.5	8	**5.3** **11.1**
iir10	1	1	10	3.0 5.0	2.4 3.6	1.42	**2.2** **2.7**	10	1.1 1.3	14.2	**10.6** **21.0**
iir11	1	1	11	3.0 5.0	2.3 3.4	1.54	**2.6** **3.3**	10 11	1.0 1.2	15.4 16.9	**13.9** **26.7**
iir12	1	1	12	3.0 5.0	2.2 3.3	1.59	**3.0** **3.7**	9 12	1.1 1.2	14.3 19.1	**11.8** **27.6**
dist	2	1	5	3.0 5.0	3.0 5.0	1.02	**1** **1**	5	1.4 1.7	5.1	**4.4** **8.8**
chemical	2	1	4	3.0 5.0	2.6 4.0	1.24	**1.6** **1.9**	4	1.5 1.7	4.96	**4.9** **10.7**
aircraft	3	3	14	3.0 5.0	3.0 5.0	1	**1** **1**	14	1.1 1.3	14	**7.4** **14.8**

Table 3: Power Reduction with Unfolding and Multiple Programmable Processors

impossible because of the difficulty of estimating hardware costs and schedule lengths. However, intuitively one would expect the unfolding-based power reduction approach for linear computation to work here too. To verify the conjecture empirically, we used the HYPER behavioral level synthesis systems to get power numbers estimates for non-unfolded and unfolded versions of our examples, with the appropriate level of unfolding being chosen greedily by increasing the level of unfolding until the energy consumed to process one data sample stops decreasing. We revisit ASIC implementation later in the paper for a more thorough presentation of the empirical results.

Since reduction in number of operations has a direct effect on power consumption, any transformation that will reduce the number of operations needed to compute the unfolded state-space equations will provide additional power and throughput benefits. Factorization and common sub-expression transformations accomplish precisely this, and we in particular note that applying these transformations to the non-feedback parts of the unfolded state-space equations will not effect the maximum throughput because that depends solely on the feedback component, and the non-feedback component can always be arbitrarily pipelined to sustain any desired throughput.

We start this subsection by summarizing the key background information about the MCM transformation which is used as an building block in the new transformation technique for power minimization.

Constant multiplication is a transformation which replaces a constant multiplication by shift and additions. For example, the product y = 175 * x, can be computed in the following way: y = x <<7 + x << 5 + x << 4 + x << 3 + x. Since the shifts and additions are significantly more area, time, and power efficient, the constant multiplication transformations has been widely used in computer architecture [Bur47, Mag88], compilers [Ber86], and VLSI signal processing[Rab91]. The problem has also rich combinatorial optimization aspects, which are described in [Knu81].

Recently, it has been realized that a common computational structure in many ASIC application domains are multiple constant multiplication with same variable [Pot94b]. More complex structure, gives a significantly higher potential for design optimization which is related to a complex combinatorial problem [Pot94b]. The crux of technique can be illustrated using the following example which involves only two constant multiplications with the same variable x: $y_1 = 175 * x$ and $y_2 = 235 * x$. The second product y_2 can be expressed as $y_2 = x \ll 7 + x \ll 6 + x \ll 5 + x \ll 3 + x \ll 1 + x$. The direct computations of two product using the constant multiplication transformations requires nine shifts and nine additions. However, using common subexpression calculation the number of shifts and additions can be reduced. The first observation is that shifts can be shared between two products, therefore only six shifts are required. Moreover, if first is the product y_3 ($y_3 = x \ll 7 + x \ll 5 + x \ll 3 + x$) computed, the products y_1 and y_2 can be computed as $y_1 = y_3 + x \ll 4$ and $y_2 = y_3 + x \ll 6 + x \ll 2$ only six additions are required.

As the number of constant multiplied by the same variable in the MCM transformation increases, the optimization process becomes more and more involved. Potkonjak et al. [Pot94b] developed the iterative pairwise matching algorithm which tackles the problem in optimization intensive manner. They also characterized the effectiveness of the approach by the following theorem.

Asymptotic Effectiveness Theorem [Pot94b]: *An arbitrarily large instance of the multiple constant multiplication problem can always be implemented using the iterative pairwise matching algorithm with a bounded finite constant number of shifts and additions irrespective of the problem size*.

We are now ready to develop an approach for power reduction in linear designs which combines the power of unfolding, the MCM transformation, and generalized Horner scheme.

Intuitively, the class of linear computations encompasses all those computations which have only additions, subtractions, and multiplications (an divisions) with constants. More formally, linear computations can be defined as those which can be described using the following set of equations.

$$X[n+1] = A*X[\mathrm{n}] + B*U[n]$$
$$Y[n] = C*X[\mathrm{n}] + D*U[n]$$

X[n] is a vector which denotes feedback states (algorithmic delays), U[n] is a vector of primary inputs, and Y[n] is a vector representation of primary outputs. Matrices A, B, C, and D have as entries constants.

We now present the procedure which transforms an arbitrary linear computation in a form which can be implemented so that power is an arbitrary low level. The procedure combines the novel use of Horner's rule for polynomial evaluation with the MCM for power optimization. The Horner's rule rearranges the computation of an n-th degree polynomial

$$u(x) = u_n x^n + u_{n-1} x^{n-1} + \dots + u_1 x + u_0, \qquad u_n \neq 0$$

to the following form:

$$u(x) = (\dots(u_n x + u_{n-1})x + \dots)x + u_0$$

Therefore, an arbitrary polynomial can be computed using at most n additions and n multiplications. The excellent exposure of Horner's rule and the number of its generalization is given in [Knu81].

The first step of the new transformational script starts by unfolding n-times set of equations, as shown in Fig. 2. It is easy to see that the resulting equation have form shown in Fig. 3. At this moment a simple analysis (counting) indicates what is required in order to achieve low power implementation. We have to find an efficient implementation for the (n+1) sets of products with the vector X, and the efficient implementation for products of inputs vectors with a variety of coefficient. The first task can be properly solved using the MCM for power transformations, which indicates that eventually regardless of the number of additional unfolding, the number of operations will stay constant for this part of computation, as indicated by the asymptotic effectiveness theorem. For the remainder of task we apply the key idea from Horner's scheme, on the part of the computation used to compute the influence of primary inputs shown in Fig. 2, so that this overhead is reduced to linear increase. For each new unfolding, only three matrix multiplications (by B, A, and C) are required and one matrix addition. Furthermore the computational structure computes significant part of X_{n+1}, after one level of additional unfolding we need only one more matrix addition.

The resulting computational structure is shown in Fig. 4 using the functional dependency form. The part of the computation which depends only on the primary inputs is shown graphically in Fig. 5. Note that we can add to this part of the computational structure an arbitrary number of pipeline delays and therefore increase throughput and reduce voltage to an arbitrary low level. The only part of computation which is cycle is $A^n X_1$ during computation of X_{n+1}. The length of this path does not increase with unfolding, since the constant A^n can be precomputed during synthesis.

So, the approach for achieving arbitrarily low power implementation of linear systems can be described using the following pseudo-code.

Transformation order for low power implementation of linear computations:

(1) *Unfold the computation n times;*

(2) *Rearrange computation using the generalized Horner's scheme;*

(3) *Apply k times the MCM transformation for each component due to constant multiplications with state variables (k is the number of states)*

$$X_{n+1} = A^n X_1 + A^{n-1} B U_1 + A^{n-2} B U_2 + \ldots + A B U_{n-1} + B U_n$$
$$Y_1 = C X_1 + D U_1$$
$$Y_2 = C A X_1 + C B U_1 + D U_2$$
$$\vdots$$
$$Y_n = C A^{n-1} X_1 + C A^{n-2} B U_1 + C A^{n-2} B U_2 + \ldots + C B U_{n-1} + D U_n$$

Figure 2: Arbitrarily Fast Implementation of Linear Computation: Necessary and Sufficient Set of Computations.

$$X_{n+1} = A^n X_1 + A^{n-1} B U_1 + A^{n-2} B U_2 + \ldots + A B U_{n-1} + B U_n$$
$$Y_1 = C X_1 + D U_1$$
$$Y_2 = C A X_1 + C (B U_1) + D U_2$$
$$Y_3 = C A^2 X_1 + C (A (B U_1) + B U_2) + D U_3$$
$$Y_3 = C A^3 X_1 + C (A (A (B U_1) + B U_2) + B U_3) + D U_4$$
$$\vdots$$
$$Y_n = C A^{n-1} X_1 + C (A (\quad \ldots \quad)) + D U_{n-1}$$

Figure 3: Application of generalized Horner scheme on the arbitrarily fast implementation of an arbitrary linear computation after n unfoldings.

$$X_{n+2} = A^{n+1} X_1 + A^n B U_1 + A^{n-1} B U_2 + \ldots + A B U_{n-1} + B U_n$$
$$Y_1 = C X_1 + D U_1$$
$$Y_2 = C A X_1 + C (B U_1) + D U_2$$
$$Y_3 = C A^2 X_1 + C (A (B U_1) + B U_2) + D U_3$$
$$\vdots$$
$$Y_n = C A^{n-1} X_1 + C (A (\quad \ldots \quad)) + D U_{n-1}$$
$$Y_{n+1} = C A^n X_1 + C (A (\quad \ldots \quad)) + D U_n$$

Figure 4: Application of generalized Horner scheme on the arbitrarily fast implementation of an arbitrary linear computation, after n+1 unfoldings

6.1 Experimental Results

We will evaluate the effectiveness of the method by assuming that voltage can not be lowered bellow 1 V. This is conservative assumption, since a number of designs have been demonstrate which operate on voltage bellow 1 V [Sin95].

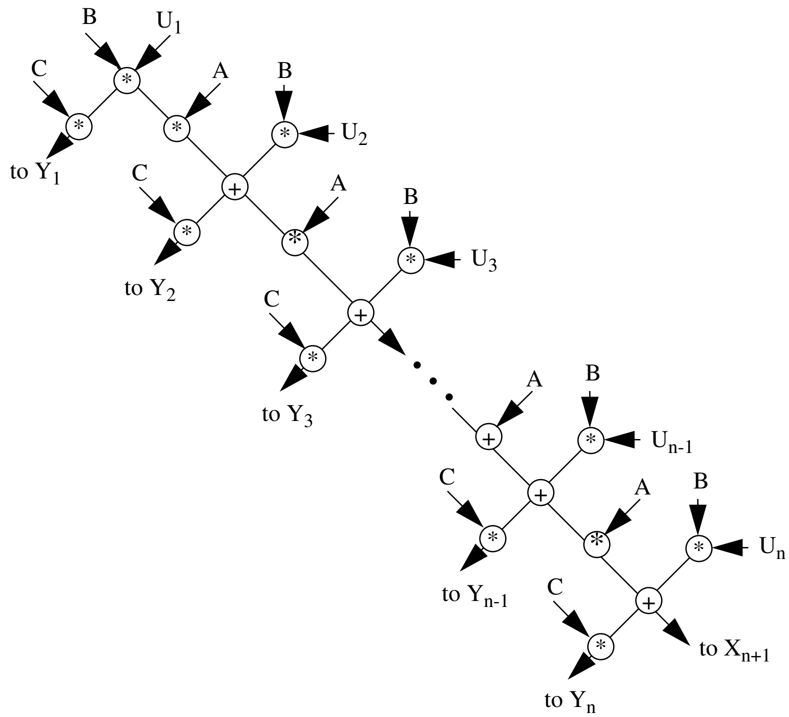

Figure 5: The part of linear CDFG on which generalized Horner scheme is applied. The structure can be pipelined to an arbitrary level.

For the evaluation we used set of commonly used examples from DSP, control and communication applications. Table 4 shows the initial power dissipation and power consumption after the application of the new ordering of transformations, as well as the power reduction factors. The average and median power reduction were by factors 30.6 and 31.8 respectively.

7.0 Low Power Implementation of High Throughput, Low Latency, Low Power and Low Area Designs of Linear Computations

7.1 Motivation

Real-time embedded control systems, real-time DSP and video servers, and ASIC for support of broadband communications are among the fastest growing segments of ASIC market. Those domains are dominated by tasks which have "closed-loop" nature. It is not sufficient just to keep up with the rate of incoming data, but it is also required to produce corresponding output within a limited period of time. Therefore, both throughput and latency are simultaneously relevant metrics of speed.

Recently, Srivastava and Potkonjak [Sri94] developed an approach which produces provably high throughput and low latency implementation of linear computations. Their approach has two components: optimum and heuristic. While the optimum approach always results in design with optimum combination of latency and throughput, the heuristic method produce area efficient solutions, highly competitive designs with respect to latency and throughput.

Design Name	Initial Energy [nJ / sample]	Energy after Optimization [nJ / sample]	Improvement Factor
mat	59.2	1.81	32.7
ellip	222	6.69	33.2
lin4	288	8.57	33.6
lin5	756	22.6	33.5
iir5 (wdf5)	118	4.70	25.1
iir6	40.2	1.77	22.7
iir8	119	3.16	37.7
iir10	96.7	3.35	28.9
iir11	81.3	2.66	30.6
iir12	89.2	2.71	32.9
steam	377	9.13	41.3
dist	157	5.57	28.2
chemical	123	4.18	29.4
aircraft	117	6.33	18.5

Table 4: Improvements in energy per sample over the initial design using the new transformation ordering

In this section, we demonstrate how the knowledge-based approach for transformation ordering combines the method proposed in [Sri94] and the MCM approach to additionally significantly reduce power consumptions in linear designs while satisfying strict throughput and latency constraints.

7.2 Ordering Transformations for Simultaneous Optimization of Latency, Throughput, Power, and Area of Linear Computations

The new approach for simultaneous optimization of latency, throughput, power, and area can be introduced using the following pseudocode:

Transformation ordering for simultaneous optimization of latency, throughput, power, and area

(1) *Apply set of transformations for the optimum (heuristic) optimization of latency and throughput;*

(2) *Apply the set of transformations for maximally fast implementation*

(3) *Apply the MCM-for-power-minimization;*

The idea behind the new script is simple. The first step produces the high throughput and low latency design. The second step rearranges the computation into the form which is suitable for the application of the MCM transformation, while preserving the achieved latency and throughput. The final step minimizes the power by exploring potential for common subexpression elimination exposed by the first two steps.

An intuition behind the new approach can be obtained by observing the first two steps of the transformation order in graphical form (Fig. 6a-c). Triangles denote multiplication of a variable by a constant. The Fig. 6a shows the popular high level synthesis benchmark 5th order elliptical wave digital filter. Using the technique from [Sri94] the computational structure shown in Fig. 6c is obtained. The Fig 6b show the key intermediate step during this process. The computational structure shown in Fig 6c is exceptionally well suited for the

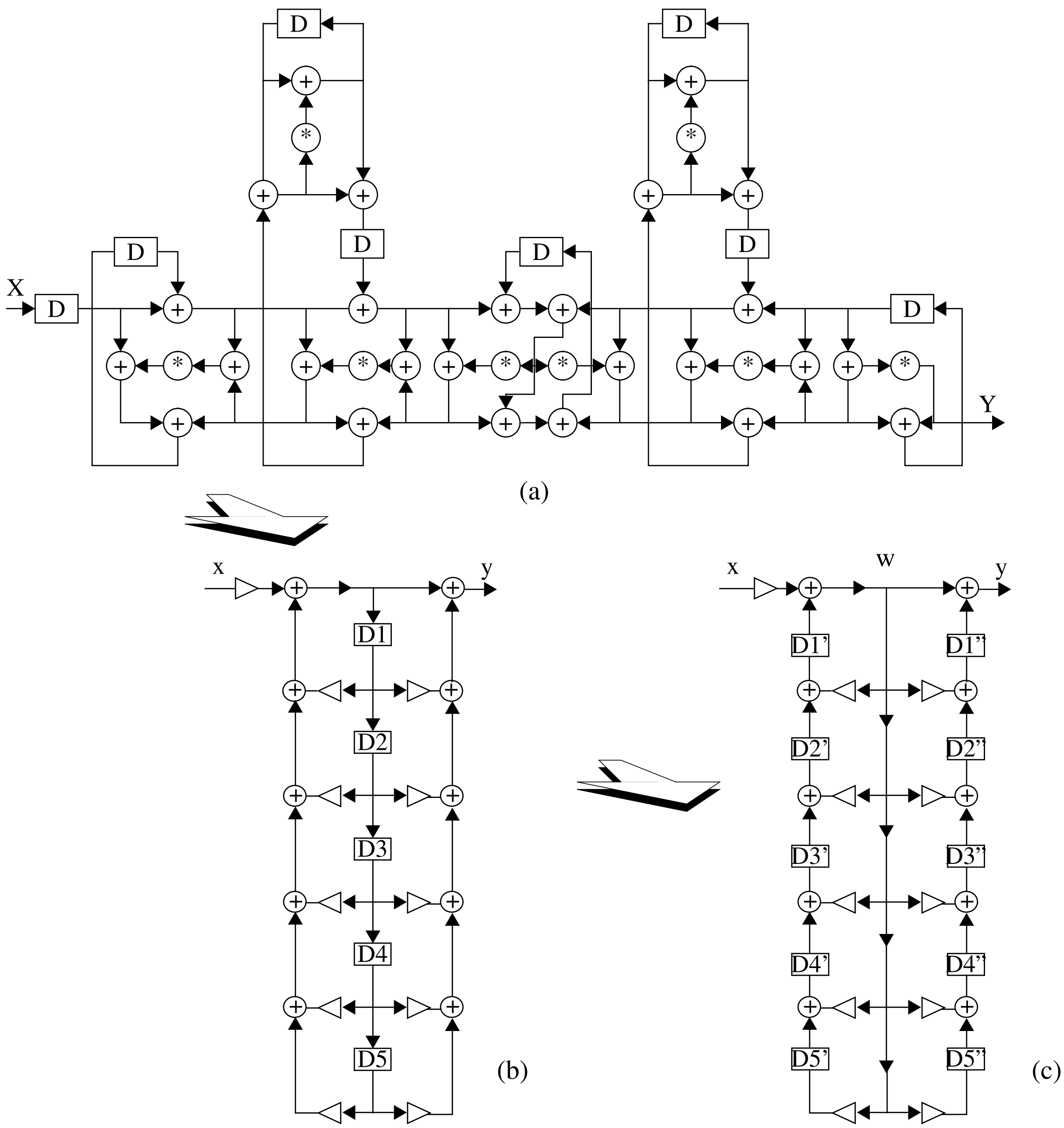

Figure 6: Illustration of the application of transformation ordering for simultaneous optimization of latency, throughput, power, and area: 5th order elliptical wave digital filter: (a) the initial structure, (b) key intermediate structure; (c) structure amenable for application of the MCM transformation.

application of the MCM transformation. In particular we see that many intermediate variables and the output are formed by multiplying the variable w with various constants.

7.3 Experimental Results

This approach presented in this section has been applied on 12 single input single output designs from our example set. Design specification and the description of designs are shown in Table 1. We conducted three sets of experimentations using heuristic technique for simultaneous optimization of throughput, latency, power, and area.

Design Name	Initial Energy [nJ / sample]	Energy After Optimization [nJ / sample]	Improvement Factor
mat	59.2	8.35	7.09
ellip	222	21.9	10.1
lin4	288	14.1	20.4
lin5	756	21.1	35.8
iir5 (wdf5)	118	18.2	6.48
iir6	40.2	12.2	3.30
iir8	119	9.87	12.06
iir10	96.7	19.0	5.09
iir11	81.3	14.6	5.57
iir12	89.2	14.8	6.03
steam	377	21.5	17.5

Table 5: Improvements in energy per sample over the initial design using the new transformation approach

Table 5 contains results of the power reduction when both throughput and latency are kept constants The average and median improvements are by factors 11.8 and 7.08 respectively. Table 6 contains results of the area reduction when both throughput and latency are kept constants. The average and median improvements are by factors 6.27 and 5.75 respectively. Finally, Table 7 contains the change in area when throughput and latency constraints are changed to new values.

Design Name	Initial area [mm^2]	Area After Optimization [mm^2]	Improvement Factor
mat	14.07	2.29	6.14
ellip	52.58	6.71	7.84
lin4	67.59	8.92	7.58
lin5	223.31	12.92	17.3
iir5 (wdf5)	28.92	8.23	3.51
iir6	10.90	4.41	2.47
iir8	29.25	5.09	5.75
iir10	22.69	8.99	2.52
iir11	20.58	7.18	2.87
iir12	25.73	9.95	2.59
steam	82.34	7.88	10.45

Table 6: Improvements in area over the initial design using the new transformation approach

Design Name	Initial Area [mm^2]	Area after Optimization [mm^2]	Latency initial/new [cycles]	Throughput initial/new [cycles]
mat	14.1	7.09	4/2	4/2
ellip	52.6	11.7	5/2	5/2
lin4	67.6	23.9	6/2	6/2
lin5	223.3	45.5	6/2	962
iir5 (wdf5)	28.9	28.8	7/2	9/2
iir6	10.9	12.9	8/2	172
iir8	29.3	20.8	11/2	10/2
iir10	22.7	49.9	17/2	15/2
iir11	20.6	40.9	19/2	19/2
iir12	25.7	41.3	20/2	18/2
steam	82.3	19.2	6/2	6/2

Table 7: Improvements in la6tency and throughput with low area overhead (or imporvement) over the initial design using the new transformation approach

8.0 Conclusion

We introduced a new approach for power minimization in linear computations using transformations. The generic approach was augmented to produce very high power reductions when either programmable of custom ASIC implementations are targeted, often with no or minimal hardware overhead.

9.0 References

[Ber86] R. Bernstein, “Multiplication by Integer Constants”, *Software - Practice and Experience,* Vol. 16, No. 7, pp. 641-652, 1986.

[Bun94] Bunda, D. Fussel, W. Athas, “Evaluating Power Implications of CMOS Microprocessors”, Proceedings *International Workshop on Low Power Design*, Napa Valley, April 1994.

[Bur47] A.W. Burks, H. Goldstine, J. von Neumann, “Preliminary Discussion of the Logical Design of an Electronic Computing Instrument”, Princeton, NJ, June 1947.

[Cha92a] A.P. Chandrakasan, S. Sheng, R.W. Brodersen, “Low-Power CMOS Digital Circuits”. *IEEE Journal of Solid-State Circuits*, pp. 473-484, Vol. 27, No. 4, 1992.

[Cha92b] A.P. Chandrakasan, M. Potkonjak, J. Rabaey, R. Brodersen, “An Approach for Power Minimization Using Transformations”, *1992 IEEE Workshop on VLSI Signal Processing,* Napa, CA, pp. 500-509, 1992.

[Cha92c] A.P. Chandrakasan, M. Potkonjak, J. Rabaey, R. Brodersen, “Hyper-LP: A Design System for Power Minimization using Architectural Transformations”, *ICCAD-92*, pp. 300-303, 1992.

[Cha94a] A.P. Chandrakasan, M.B. Srivastava, R.W. Brodersen, "Energy Efficient Programmable Computation", 7th International Conference on VLSI Design, pp. 261-264, 1994.

[Cha94b] A. Chatterjee, R.K. Roy, “Synthesis of Low Power DSP Circuits Using Activity Metrics”, Proc. of *VLSI Design Conference*, pp. 265-270, 1994.

[Dun93] P. Duncan, S. Swamy, R. Jain,, “Low-Power DSP Circuit Design Using Retimed Maximally Parallel Architectures”, *Proceedings of the 1st Symposium on Integrated Systems*, pp. 266-275, March 1993.

[Gla84] M. Glasser, “Delay and Power Optimization in VLSI Circuits”, *21st Design Automation Conference*, pp. 529-535, 1984.

[Goo94] L. Goodby, A. Orailoglu, P.M. Chau, “Microarchitectural Synthesis of Performance-Constrained, Low Power VLSI Designs”, *International Conference of Computer Design*, paper 2.3.1.1, 1994.

[Knu81] D.E. Knuth, *The Art of Computer Programming: Volume 2: Seminumerical Algorithms*, 2nd edition, Addison-Wesley, Reading, MA, 1981.

[Ku92] D.C. Ku and G.D. Micheli, *High Level Synthesis of ASICs under Timing and Synchronization Constraints*. Kluwer Academic Publishers, 1992.

[Mag88] D.J. Magenheimer, L. Peters, K. Pettis, D. Zuras, "Integer Multiplication and Division on the HP precision Architecture", *Trans. on Computers*, Vol. 37, No. 8, pp. 980-990, 1988.

[Par89] K. K. Parhi, D. Messerschmitt, "Pipeline interleaving and parallelism in recursive Filters, Parts 1 and 2", *IEEE Trans. on ASSP*, Vol. 37, pp. 1099-1117 and 1117-1134, 1989.

[Pot91] M. Potkonjak, J. Rabaey, "Maximally Fast and Arbitrarily Fast Implementation of Linear Computations", Technical Report, NEC USA, Princetion, NJ, 1991.

[Pot94a] M. Potkonjak, M.B. Srivastava, "Behavioral Synthesis of High Performance, Low Cost, and Low Power Application Specific Processors for Linear Computations", Proceedings of the *International Conference on Application Specific Array Processors*, 1994.

[Pot94b] M. Potkonjak, M.B. Srivastava, A. Chandrakasan, "Efficient Substitution of Multiple Constant Multiplications by Shifts and Additions using Iterative Pairwise Matching", *31st DAC*, pp. 189-194, 1994.

[Rab91] J. Rabaey, et al., "Fast Prototyping of Data Path Intensive Architecture", *IEEE Design and Test*, Vol. 8, No. 2, pp. 40-51, 1991.

[Rag94] A. Raghunathan, N.K. Jha, "Behavioral Synthesis for Low Power", *International Conference of Computer Design, ICCD-94*, paper 2.3.1.2, 1994.

[Rob87] R. A. Roberts and C. T. Mullis, *Digital Signal Processing*, Reading, MA: Addison-Wesley, 1987.

[Roy94] K. Roy and S. C. Prasad. Circuit activity based logic synthesis for low power reliable operations. *IEEE Transactions on VLSI Systems*, Vol. 1, No. 4, pp. 503–513, December 1993.

[Sri94] M. B. Srivastava, M. Potkonjak: "Transforming Linear Systems for Joint Latency and Throughput Optimization", *EDAC-94 European Design Automation Conference*, pp. 267-271, 1994.

[Sin95] D. Singh, J. Rabaey, M. Pedram, F. Cathoor, S. Rajgopal, N. Seghai, T. Mozdzen, "Power Conscious CAD Tools and Methodologies: A Perspective", *Proc. of the IEEE*, Vol. 83, No. 4, pp. 570-594, 1995.

[Tiw94] V. Tiwari, S. Malik, A. Wolfe, "Power Analysis of Embedded Software: A first Step towards Software Power minimization, *ICCAD-94*, pp. 384-390, 1994.

[Wal89] R.A. Walker, D.E. Thomas, "Behavioral transformations for algorithmic level IC design", *IEEE Trans. on CAD*, Vol. 8, No. 10, pp. 1115-1128, 1991.

[Wuy94] S. Wuytack, F. Catthoor, F. Franssen, L. Nachtergaele, H. De Man, "Global communication and memory optimizing transformations for low power systems", *Intnl. Workshop on Power Design*, Napa Valley, 1994.