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

Mani Srivastava
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Miodrag Potkonjak
Computer Science Department
University of California
Los Angeles, CA 90095-1596

Abstract - Linear computations form an important type of computation that is widely used in DSP and communications. 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 2.2 and 8 respectively. To accomplish this we exploit a newly identified property of unfolding whereby as a linear system is unfolded, the number of operations per sample at first decreases to reach a minimum and then begins to rise. 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 power is minimized.

1.0 Throughput and Power in Linear Systems

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. This paper explores the relationship of throughput with increasingly important design metric - power. 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.

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 are the preferred medium of implementation as opposed to custom datapaths.

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

In CMOS technology there are three sources of power consumption: switching current, 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. The average power consumption of a CMOS gate due to the switching component is given by:

$$P = \alpha C_L V_{dd}^2 \qquad \textbf{(EQ 1)}$$

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→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 function of voltage

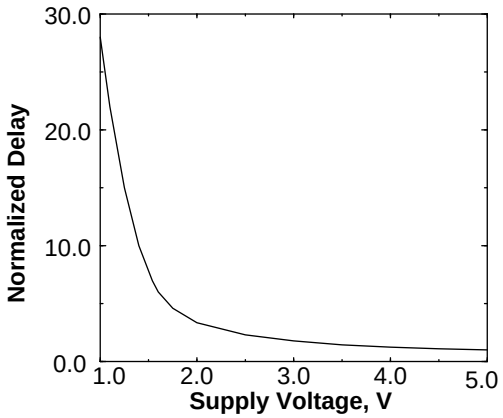

Figure 1: Normalized Gate Delay (delay @ 5V = 1) vs. Voltage

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

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

while compensating for the resulting loss in circuit speed by techniques that increase the throughput.

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). 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. 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. When voltage reduction is not possible, one can trade-off the extra throughput obtained with lower clock frequency or with shutdown, both of which will result in linear reduction.

1.2 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 for $n \in \{0, 1, 2, 3, \ldots\}$:

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

where $X[n] \in \Re^{P\times 1}$, $Y[n] \in \Re^{Q\times 1}$, and $S[n] \in \Re^{R\times 1}$ 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:

$$\begin{aligned} &\#\text{ of muls } = (R+P)(R+Q) \\ &\#\text{ of adds } = (R+P-1)(R+Q) \\ &\text{feedback critical path } = m + \lceil \log_2(1+R) \rceil \end{aligned}$$

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 $m \geq 1$. 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 fully balanced binary tree.

1.3 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 design parameters during high level synthesis [Wal89]. While power optimization was rarely addressed in the past (e.g. [Gla84]), it has recently attracted attention due to the growing demands from portable computing and communication products. 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] also proposed methods for power minimization which explore trade-offs between voltage scaling, throughput, and power.

2.0 Unfolding-Driven Voltage-Throughput Trade-Off

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 [Rob87] 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:

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

Note that these equations process $i+1$ data samples for each execution. For the i times unfolded system we get the following characteristics, where $\#(+,i)$ is the number of additions, $\#(*,i)$ is the number of multiplications, $\text{CP}(i)$ is the feedback critical path, and $\text{MaxThroughput}(i)$ is the maximum throughput:

$$\begin{aligned} \#(*,i) &= R^2 + (i+1)PR + (i+1)QR + \frac{(i+1)(i+2)}{2}PQ \\ \#(+,i) &= R^2 + (i+1)PR + (i+1)QR + \frac{(i+1)(i+2)}{2}PQ - R - (i+1)Q \\ \text{CP}(i) &= m + \lceil \log_2(1+R) \rceil \\ \text{MaxThroughput}(i) &= \frac{(i+1)}{m + \lceil \log_2(1+R) \rceil} \end{aligned}$$

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, the increase in number of multiplication operations per sample due to i times unfolding is:

$$\frac{\#(*,i)}{i+1} - \#(*,0) = -\frac{i}{i+1}R^2 + \frac{i}{2}PQ < 0 \quad \text{for } i < \left(\frac{2R^2}{PQ} - 1\right) \qquad \textbf{(EQ 4)}$$

and, the increase in number of addition operations is:

$$\frac{\#(+,i)}{i+1} - \#(*,0) = -\frac{i}{i+1}R(R-1) + \frac{i}{2}PQ < 0 \quad \text{for } i < \left(\frac{2R(R-1)}{PQ} - 1\right) \qquad \textbf{(EQ 5)}$$

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 observations lead to the following strategies for low power implementations.

3.0 Implementation on a Single Processor

In the case of a single programmable processor the throughput that is achieved is solely decided by the number of operations. 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: $i_{opt} = \left\lfloor \sqrt{\frac{(2R-1)R}{PQ}} - 1 \right\rfloor$ or $\left\lceil \sqrt{\frac{(2R-1)R}{PQ}} - 1 \right\rceil$ 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, 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:

$$S_{\max} = (i_{opt} + 1)\frac{\#(*,0) + \#(+,0)}{\#(*, i_{opt}) + \#(+, i_{opt})}$$

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 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. 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 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 using binary search as long as the number of operations continues to decline. Since the run times are low, the preceding linear search strategy is quite acceptable. In any case, more sophisticated search techniques such as binary search could be employed if desired.

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

Name	Description	P	Q	R
ellip	4-state 1-input linear controller	1	1	4
iir5 (wdf5)	5th order elliptic wave digital filter	1	1	5
iir6	6th order low-pass elliptic cascade IIR filter	1	1	6
iir10	10th order band-stop Butterworth IIR filter	1	1	10
iir12	12th order band-pass Chebyshev 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

Table 1: Description of the Example Suite

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 (*ellip, steam*) had dense coefficient matrices so that the power reduction obtained is the same as for the dense coefficient matrices case. For one examples

P	Q	R	Dense Coefficient Matrices						Real Examples						
			Initial	After Optimal Unfolding					Name	Initial	After Heuristic Unfolding				
			# Ops	i	# Ops	V	Frq Red	Pwr Red		# Opa	i	# Ops	V	Frq Red	Pwr Red
1	1	4	45	4	26.6	2.2	1.7	2.3	ellip	45	4	26.6	2.2	1.7	**2.3**
1	1	5	66	6	33.4	2.0	2.0	4.4	wdf5	32	3	27.3	2.7	1.2	**1.4**
									steam	66	6	33.4	2.0	2.0	**4.4**
1	1	6	91	7	40.3	1.9	2.3	5.6	iir6	40	5	36	2.8	1.1	**1.3**
1	1	10	231	13	67.6	1.6	3.4	12.0	iir10	85	9	59.8	2.4	1.4	**2.2**
1	1	12	325	16	81.2	1.5	4.0	16.0	iir12	114	11	71.8	2.2	1.6	**3.0**
2	1	5	78	4	50	2.3	1.6	2.7	dist	48	3	47.3	3.0	1.0	**1**
2	1	4	55	3	40	2.4	1.4	2.2	chem.	41	2	33	2.6	1.2	**1.6**

Table 2: Power Reduction in a Single Processor using Unfolding-Driven Voltage-Throughput Trade-off (Initial = 3 V)

(*dist*) no power reduction is obtained. The average processor power reduction for the example suite is x2.2 when the initial voltage is 3.0V, and an even larger x3 when the initial voltage is 5.0V (not shown), 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 x1.6 reduction in the number of operations. Therefore, the clock frequency can be reduced by x1.6, resulting in a power reduction by x1.6 (or, 37%) while the processor voltage remains unchanged. This strategy gives an average power reduction of x1.4 (29%) over all our examples.

4.0 Implementation on Multiple Processors

Potentially more savings can be obtained 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.

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 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 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 assumptions are appropriate when one also considers empirical results, reported by researchers such as [Tiw94], that indicate a strong correlation between power and number of operations in programmable general purpose and DSP computation.

The first step 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 the speed-up due to multiple processors is linear for $N \le R$, i.e., $S_{\max}(N, i) = NS_{\max}(1, i)$ for $(N \le R)$. This 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. 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.

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
ellip	1	1	4	3.0	2.2	1.69	**2.3**	4	1.3	6.76	**9.0**
iir5/wdf5	1	1	5	3.0	2.7	1.17	**1.4**	5	1.4	5.85	**5.4**
steam					2.0	1.97	**4.4**	5	1.2	9.85	**12.3**
iir6	1	1	6	3.0	2.8	1.11	**1.3**	6	1.3	6.66	**5.6**
iir10	1	1	10	3.0	2.4	1.42	**2.2**	10	1.1	14.2	**10.6**
iir12	1	1	12	3.0	2.2	1.59	**3.0**	9	1.1	14.3	**11.8**
dist	2	1	5	3.0	3.0	1.02	**1**	5	1.4	5.1	**4.4**
chemical	2	1	4	3.0	2.6	1.24	**1.6**	4	1.5	4.96	**4.9**

Table 3: Power Reduction with Unfolding and Multiple Processors

5.0 Implementation on Custom Datapath ASICs

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

Constant multiplication is a transformation which replaces a constant multiplication by shifts 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, this transformations has been widely used in computer architecture, compilers [Mag88], and VLSI signal processing [Rab91].

Recently, it has been realized that a common computational structure in many ASIC application domains is multiple constant multiplication with same variable [Pot94]. More complex structures give a significantly higher potential for design optimization which is related to a complex combinatorial problem [Pot94]. 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 $y_2 = x << 7 + x << 6 + x << 5 + x << 3 + x << 1 + x$. The direct computations of two product using the constant multiplication transformations requires nine shifts and nine additions. However, using common subexpression 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 ($y_3 = x << 7 + x << 5 + x << 3 + x$) computed, the products y_1 and y_2 can be computed as $y_1 = y_3 + x << 4$ and $y_2 = y_3 + x << 6 + x << 2$ only six additions are required.

As the number of constant multiplied by the same variable increases, the MCM optimization process becomes more involved and effective [Pot94], as indicated by the following theorem.

Asymptotic Effectiveness Theorem [Pot94]: *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. Formally, linear computations can be defined as those which can be described by:

$$X[n+1] = A*X[n] + B*U[n]$$
$$Y[n] = C*X[n] + D*U[n]$$

X[n] denotes feedback states (algorithmic delays), U[n] denotes primary inputs, and Y[n] denotes 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. Horner's rule rearranges an n-th degree polynomial $u(x) = u_n x^n + u_{n-1} x^{n-1} + ... + u_1 x + u_0, \quad u_n \neq 0$ to the following form: $u(x) = (...(u_n x + u_{n-1})x + ...)x + u_0$. Therefore, an arbitrary polynomial can be computed using at most n additions and n multiplications. An excellent exposition of Horner's rule and a number of its generalization is given in [Knu81].

$$X_{n+1} = A^n X_1 + A^{n-1} BU_1 + A^{n-2} BU_2 + ... + ABU_{n-1} + BU_n$$
$$Y_1 = CX_1 + DU_1$$
$$Y_2 = CAX_1 + CBU_1 + DU_2$$
$$\vdots$$
$$Y_n = CA^{n-1} X_1 + CA^{n-2} BU_1 + CA^{n-2} BU_2 + ... + CBU_{n-1} + DU_n$$

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

$$X_{n+1} = A^n X_1 + A^{n-1} BU_1 + A^{n-2} BU_2 + ... + ABU_{n-1} + BU_n$$
$$Y_1 = CX_1 + DU_1$$
$$Y_2 = CAX_1 + C(BU_1) + DU_2$$
$$Y_3 = CA^2 X_1 + C(A(BU_1) + BU_2) + DU_3$$
$$\vdots$$
$$Y_n = CA^{n-1} X_1 + C(A(\quad ... \quad)) + DU_{n-1}$$

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

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. 3 using the functional dependency form. Note that we can add to the nonrecursive 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^nX_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)*

5.1 Experimental Results

We evaluate the effectiveness of the method by conservatively assuming that voltage can not be lowered bellow 1 V. 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 30.2 and 31.8 respectively.

6.0 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.

Design Name	Initial Energy [nJ / sample]	Energy after Optimization [nJ / sample]	Improvement Factor
ellip	222	6.69	33.2
iir5 (wdf5)	118	4.70	25.1
iir6	40.2	1.77	22.7
iir10	96.7	3.35	28.9
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

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

7.0 References

[Cha92] A.P. Chandrakasan, M. Potkonjak, J. Rabaey, R. Brodersen, "An Approach for Power Minimization Using Transformations", *VLSI Signal Processing*, pp. 500-509, 1992.

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

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

[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.

[Pot94] 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*, pp. 318-322, 1994.

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

[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.

[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.