# Power Optimization using Minimization of the Number of Operations


## Abstract

Design of modern portable application specific systems is often defined by their demand for low power, flexibility, cost sensitivity, and by their DSP nature. Therefore, for economic realization of portable applications effective low power compilation and architectural methodologies are of prime importance.

In this paper we introduce an approach for power minimization using a set of compilation and architectural techniques. The key technical innovation is a compilation technique for minimization of the number of operations. We have developed a new divide and conquer technique which combines and coordinates optimization and enabling effects of several transformations so that the number of operations in each logical partition is minimized. On a set of benchmark examples the number of operations is reduced by average of 42.3%, which results in reduction in power consumption by factor of 3.48 on single processor platform with no cost overhead. Even further power reduction, more than an order of magnitude, is achieved by using multiprocessors platforms.

On the architectural dimension, we investigate coordinated impact of compilation techniques and new ultra low power technologies on the number of processors which provide optimal trade-off of between cost and power. We demonstrate that proper compilation technique can significantly reduce power consumption, with bounded hardware cost penalty. The effectiveness of all techniques and algorithms is documented on a number of real-life designs.

# 1 Introduction

## 1.1 Motivation

Recently, a new class of portable applications has been forming new market at exceptionally high rate. The applications and products of portable wireless market are defined by their intrinsic demand for portability, flexibility, cost sensitivity and by their high DSP computation content [12]. Portability translates in the crucial importance of low power design, flexibility results in a need for programmable platforms implementation, and cost sensitivity narrows architectural alternatives to a uniprocessor or an architecture with a limited number of off-the-shelf standard processors. The key optimization degree of freedom for relaxing and satisfying this set of requirements comes from properties of typical portable computations. The computations are mainly linear, but rarely 100% linear due to either need for adaptive algorithms or nonlinear quantization elements. This type of computations is well suited for static compilation and intensive quantitative optimization.

Two main recent relevant technological trends are reduced minimal feature size and therefore reduced voltages of deep submicron technologies, and introduction of ultra low power technologies. In widely dominating digital CMOS technologies the power consumption is proportional to square of supply voltage($V_{dd}$). The most effective techniques try to reduce $V_{dd}$ while compensating for speed reduction using a variety of architectural and compilation techniques [14].

The main limitation of the conventional technologies with respect to power minimization is also related to $V_{dd}$ and threshold voltage($V_t$). In traditional bulk Si technologies both voltages are commonly limited to range above 0.7V. However, in the last few years ultra low power silicon on insulator(SOI) technologies, such as SIMOX(SOI using separation of oxygen), bond and etchback SOI(BESOI) and silicon-on-insulator-with-active-substrate(SOIAS), both $V_{dd}$ and $V_t$ can be reduced to well below 0.7V [3]. There are a number of reported ICs which have values for $V_{dd}$ and $V_t$ in a range as low as 0.05V-0.1V [3].

Our goal in this paper is to develop a system of synthesis and compilation methods and tools for realization of portable applications. Technically restated, the primary goal is to develop techniques which efficiently and effectively compile typical DSP wireless applications on single and multiple programmable processors assuming both traditional bulk silicon and SOI technologies. Furthermore, we study achievable power-cost trade-offs when parallelism is traded for power reduction on programmable platforms.

The main technical innovation of the research presented in this paper is the first approach for the minimization of the number of operations in general computations. The approach does not treat just significantly wide set of computations than the other previously published techniques [15], but also

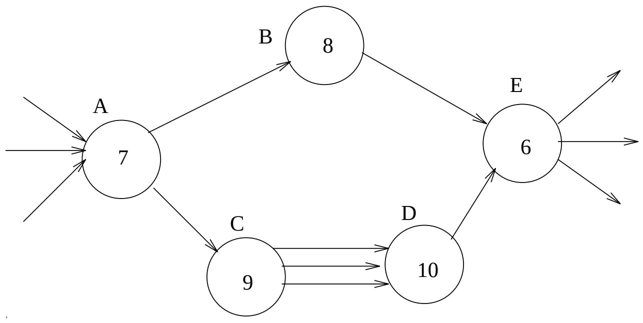


Figure 1: A motivational example

outperforms or performs at least as well as other techniques on all examples.

## 1.2 Introductory Example

Among the three main sources of power consumption in digital CMOS circuits which are switching power, short circuit power and leakage power, switching power dominates in CMOS technology [2]. Since switching power has quadratic dependence on supply voltage, it is desirable to operate at the lowest possible power supply voltage for lower power consumption in CMOS circuits. However, lower supply voltage results in reduction of circuit speed. To keep design speed constant, techniques to increase throughput are required. Once the throughput is improved, supply voltage can be reduced to an extent that the throughput reverts back to the original. Since the throughput of a computation on a single programmable processor is entirely determined by the number of operations, the total number of operations should be minimized to optimize throughput and in turn power consumption.

$S[n] = AS[n-1] + BX[n]$
$Y[n] = CS[n-1] + DX[n]$
where $X$, $Y$, and $S$ are the input, output, and state vectors respectively and $A, B, C,$ and $D$ are constant coefficient matrices.

Figure 2: State-space equations for linear computations

We illustrate the key ideas of our approach for minimizing the number of operations by considering the computation of Figure 1. Each node represents a subpart of the computation. We make the following assumptions only specifically for clarifying the presentation of this example and simplifying the example. We stress here that the assumptions are not necessary for our approach. We assume that each subpart is linear which can be represented by state-space equations in Figure 2. We also assume that every subpart is dense, which means that every output and state in a subpart are linear

$$
\begin{array}{|l|}
\hline
S_1[n] = 2 * S_1[n-1] + S_2[n-1] + 4 * X[n-1] \\
S_2[n] = 3 * S_1[n-1] + 5 * S_2[n-1] + X[n-1] \\
Y[n] = 2 * S_1[n-1] + 3 * S_2[n-1] \\
\#\ \text{addition} = 5,\ \#\ \text{multiplication} = 6 \\
\#\ \text{operations} = \#\ \text{addition} + \#\ \text{multiplication} = 11 \\
\hline
\end{array}
$$

Figure 3: A simple example of # operations calculation

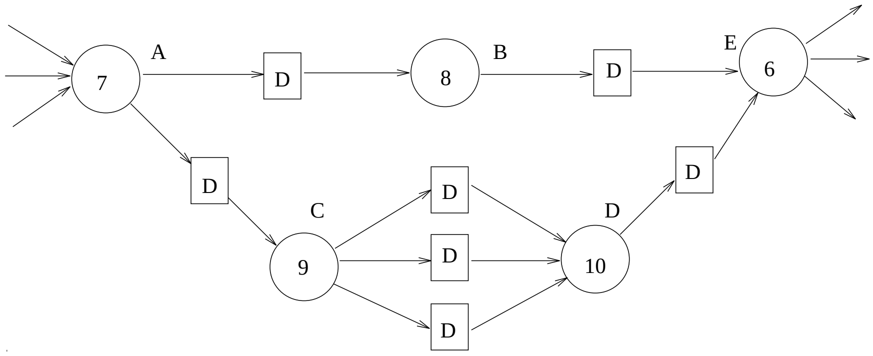


Figure 4: A motivational example after the isolation step

combinations of all inputs and states in the subpart with no 0, 1, or -1 coefficients. The number inside a node is the number of delays or states in the subpart. We assume that when there is an arc from a subpart X to a subpart Y, every output and state of Y depend on all inputs and states of X.

The number of operations per input sample is initially 2081(We explain how the number of operations is calculated in a maximally fast approach [9] in Figure 3). Using the technique of [15] which unfolds the entire computation, the number can be reduced to 725 with an unfolding factor of 12. Our approach optimizes each subpart separately. This separate optimization is enabled from isolating the subparts using pipeline delays. The Figure 4 shows the resulting computation after the isolation step. Separate optimization step results in 522.27 operations. We perform subparts merging to further optimize. If the subparts C and D are merged and optimized together, the number of operations is further reduced to 399.4. The details of our approach on this example are presented in Section 4.3. The approach has reduced the number of operations by a factor of 1.82 from the previous technique of [15], while it has achieved the reduction by a factor of 5.2 from the initial number of operations.

For single processor implementation, since both the technique of [15] and our new method yield higher throughput than the original, the supply voltage can be lowered up to the extent that the extra throughput is compensated by the loss in circuit speed due to reduced voltage. If the initial voltage is 3.3V, then our technique reduces power consumption by a factor of 26.0 with the supply voltage of 1.48V while the technique of [15] reduces it by a factor of 10.0 with the supply voltage of 1.77V.

### 1.3 Design Methodology: What is New?

Our design methodology can be briefly described as follows. Given throughput, power consumption and cost requirements for a computation, our goal is to find cost-effective, power-efficient solutions on single or multiple programmable processor platforms. The first step is to find power-efficient solutions for single processor implementation by applying the new technique described in Section 4. The second step is to continue to add processors until the addition does not reduce power consumption much enough to justify the cost of additional processor. This step generates cost-effective and power efficient solutions. This straightforward design methodology produces implementations with low cost and low power consumption for given design requirements.

The first technical highlight of the paper is a new divide and conquer compilation procedure for the minimization of the number of operations. We have developed a new divide and conquer technique which combines and coordinates power and enabling effects of several transformations so that the number of operations in each logical partition is minimized using a well organized ordering of transformations. To the best of our knowledge this is the first approach for minimization of the number of operations which in optimization intensive way treats an arbitrary computation.

The second technical highlight is the quantitative analysis of trade-off of cost-power on multiple programmable processor implementation platforms. We derive condition under which optimization of the cost-power product using parallelization is beneficial.

### 1.4 Paper Organization

The rest of the paper is organized in the following way. First, in the next section, we summarize the relevant background material. In Section 3 we review the related work on low power estimation and optimization as well as on program optimization using transformations, and in particular the minimization on the number of operations. Sections 4 and 5 are technical core of the paper and present new approaches for minimization of the number of operations and exploration of compiler and technology impact on power-cost tradeoffs of multiple processors-based low power application specific systems. Before conclusion in Section 7, we present comprehensive experimental results and their analysis in Section 6.

## 2 Preliminaries

Before we delve into technical details of the new approach, we outline the relevant preliminaries in this section. In particular, we describe application and computation abstractions, selected implementation platform at the technology and architectural level, and power estimation related background material.

We selected as our computational model homogeneous synchronous data flow(HSDF) [7]. The HSDF model is well suited for specification of single task computations in numerous application domains such as digital signal processing, video and image processing, broadband and wireless communications, control, information and coding theory, and multimedia.

The syntax of a targeted computation is defined as a hierarchical data-control flow graph(CDFG) [11]. The CDFG represents the computation as a flow graph, with nodes, data edges, and control edges. The semantics underlying the syntax of the CDFG format, as we already stated, is one of synchronous data flow model of computation. The semantics of the HSDF model assume a semi-infinite or a very long input stream of data arriving and being processed at periodic intervals, imposed by the nature of the specified applications.

The only relevant speed metrics is throughput, the rate at which the implementation is capable of accepting the input samples from two consecutive iterations. We opted for throughput as the selected speed metrics since in essentially all DSP and communication wireless computations latency is not a limiting factor.

The basic atomic building block of targeted hardware platform is a single programmable processor. We assume that all types of operations take one clock cycle for their execution, as it is the case in many modern DSP processors. The adaptations of the software and algorithms to other hardware timing models is straightforward. In the case of multiprocessor, we make the following additional simplifying assumptions: (i) all processors are homogeneous and (ii) inter-processor communication does not cost any time and hardware. This assumption is reasonable because multiple processors can be placed on single integrated circuit due to increased integration.

It is well known that there are three principal components of power consumption in CMOS integrated circuits: switching power, short-circuit power, and leakage power. The switching power is given by $P_{switching} = \alpha C_L V_{dd}^2 f_{clock}$, where $\alpha$ is the probability that the power consuming switching activity, i.e. transition from 0 to 1, occurs, $C_L$ is the loading capacitance, $V_{dd}$ is the supply voltage, and $f_{clock}$ is the system clock frequency. $\alpha C_L$ is defined to be effective switched capacitance. In CMOS technology, switching power dominates power consumption. We make a couple of assumptions that effective switched capacitance increases linearly with the number of processors and supply voltage can not be lowered below threshold voltage $V_t$, for which we use several different values between 0.1V and 0.7V for both conventional and ultra low power technology.

It is also known that reduced voltage operation comes at the cost of reduced throughput [2]. The clock speed $T$ follows the following formula: $T = C\frac{V_{dd}}{(V_{dd}-V_t)^2}$ where $C$ is a constant [2]. The maximum rate at which a circuit is clocked monotonically decreases as the voltage is reduced. As the supply voltage is reduced close to $V_t$, the rate of clock speed reduction becomes higher.

The power model used in this research is built on three statistically validated and experimentally established facts. The first fact is that the number of operations at the machine code-level is proportional to the number of operations at high-level language [5]. The second fact is that the power consumption of modern programmable processor is directly proportional to the number of operations, regardless of what the mix of operations being executed is [17]. The final empirical observation is related to power consumption and timing models in digital CMOS circuits presented in the previous subsections.

Based on these three observations, we conclude that if the targeted implementation platform is a single programmable CMOS processor, the key for power minimization is the reduction of the number of operations. When the initial number of operations is $N_{init}$, the optimized number of operations is $N_{opt}$, the initial voltage is $V_{init}$ and the scaled voltage is $V_{opt}$, the optimized power consumption relative to the initial power consumption is $(\frac{V_{opt}}{V_{init}})^2\frac{N_{opt}}{N_{init}}$. For multiprocessors, based on the assumption of no communication overhead the optimized power consumption for $n$ processors relative to the optimized power consumption for single processor is $(\frac{V_n}{V_1})^2$, where $V_i$ is the optimized voltage for $i$ processors.

## 3 Related Work

In the last five years power minimization has been arguably the most popular optimization goal. Numerous power modeling and optimization techniques have been proposed at all levels of the abstraction in the synthesis process. Power minimization effort across all levels of design abstraction process is surveyed in [14] and in particular system-level compilation techniques for power optimization are reviewed in [4]. Power minimization techniques, when programmable platforms are targeted, have been proposed [17].

Transformations have been widely used at all levels of abstraction in the synthesis process for optimization of a variety of design and program metrics, such as throughput, latency, area, power, permanent and temporal fault-tolerance, and testability [10]. Interestingly, the power of transformations is most often focused on secondary metrics such as parallelism, instead on the primary metrics such as the number of operations.

Potkonjak and Rabaey [9] addressed the minimization of the number of multiplications and additions in linear computations in their maximally fast form so that the throughput is preserved. Potkonjak et al. [10] presented a set of techniques for minimization of the number of shifts and additions in linear computations. Sheliga and Sha [13] presented an approach for minimization of the number of multiplications and additions in linear computations. Srivastava and Potkonjak [15] developed an approach for the minimization of the number of operations in linear computations using unfolding and the application of the maximally fast procedure.

# 4 Single Programmable Processor Implementation

The global flow of the approach is presented in Section 4.1. The strategy is based on divide-and-conquer optimization followed by post optimization step, merging of divided subparts which is explained in Section 4.2. Finally, Section 4.3 provides a comprehensive example to illustrate the strategy.

## 4.1 Global Flow Of the Approach

The core of the approach is presented in the pseudo-code of Figure 5. The rest of this sub-section explains the global flow of the approach in more detail.

```
Decompose a computation into strongly connected
components(SCCs);
Use pipelining to isolate the SCCs;
Minimize the number of delays using retiming;
For each of SCCs
  If(the SCC is linear)
    Apply unfolding;
  Else
    Apply unfolding with isolation of nonlinear operations;
Merge linear SCCs to further optimize;
Schedule subparts to minimize memory usage;
```

Figure 5: The core of the approach

The first step of the approach is to identify the computation's strongly connected components(SCCs), using the standard depth-first search-based algorithm [16]. For any pair of operations $A$ and $B$ within a SCC, there exist both a path from $A$ to $B$ and one from $B$ to $A$. The SCCs are isolated from each other using pipeline delays, which enables us to optimize each subpart separately. The inserted pipeline delays are treated as input or output of the subpart. As a result, every output and state in a subpart depend only on the inputs and states of the subpart. None of the other states in the computation and none of the primary inputs to the other subparts affect the states and outputs of the subpart. Thus, in this sense, the subpart is isolated from the rest of the computation and it can be optimized separately. Note that this isolation is not affected by unfolding.

In the next step, the number of delays in the computation is minimized using retiming by the Leiserson-Saxe algorithm [8]. Note that smaller number of delays will require smaller number of operations since both the next states and outputs depend on the previous states.

| |
|---|
| $i_{opt} = \lfloor\sqrt{\frac{(2R-1)R}{PQ}} - 1\rfloor, or \lceil\sqrt{\frac{(2R-1)R}{PQ}} - 1\rceil$, which gives smaller value of $i_{opt}(PQ - \frac{R(2R-1)}{i_{opt}+1})$. |
| $N(*, i) = \#$ of multiplications for $i$ times unfolded system |
| $= R^2 + (i+1)PR + (i+1)QR + \frac{(i+1)(i+2)}{2}PQ$ |
| $N(+, i) = \#$ of additions for $i$ times unfolded system |
| $= R^2 + (i+1)PR + (i+1)QR + \frac{(i+1)(i+2)}{2}PQ - R - (i+1)Q$ |

Figure 6: Formula of optimal unfolding for dense linear system with $P$ inputs, $Q$ outputs, and $R$ states.

The SCCs are further classified as either linear or nonlinear. Linear computations can be represented using the state-space equations in Figure 2. We have used an approach of [15] for optimization of linear SCCs, which uses unfolding and the maximally fast procedure [9]. We note that instead of maximally fast procedure the ratio analysis by [13] can be used. [15] has provided the closed-form formula for the optimal unfolding factor with the assumption of dense linear computations. We provide the formula in Figure 6. For sparse linear computations, they have proposed a heuristic which continues to unfold until there is no improvement. We have made the simple heuristic more efficient with binary search, based on the unimodality property of the number of operations on unfolding factor [15].

When a SCC is classified as nonlinear, we apply unfolding with isolation of nonlinear operations. All nonlinear operations are isolated from the SCC so that the remaining linear subparts can be optimized by maximally fast procedure. All arcs from nonlinear operations to the linear subparts are considered as inputs to the linear subparts, and all arcs from linear subparts to the nonlinear operations are considered as outputs from the linear subparts. The process is illustrated in Figure 7. All arcs denoted as i are considered to be inputs and all arcs denoted as o are considered to be outputs for unfolded linear subpart. We state our observation without proof for the sake of brevity that if every output and state of the nonlinear SCC depend on nonlinear operations, then unfolding with separation of nonlinear operations is ineffective in reducing the number of operations.

Sometimes it is beneficial to decompose a computation into larger subparts than SCCs. We con-

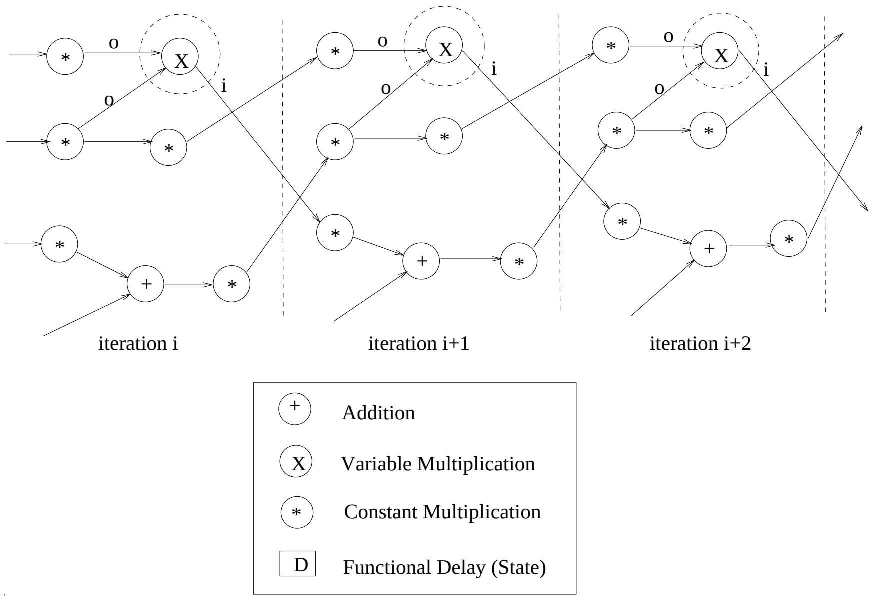


Figure 7: An example of isolating nonlinear operations from 2 times unfolded nonlinear SCC

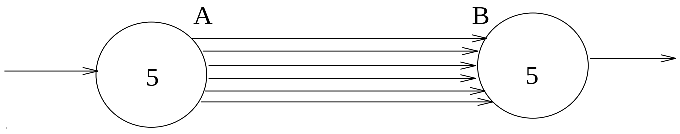


Figure 8: A motivational example for SCC merging

sider an example given in Figure 8. We use the same assumptions made for the motivational example in Section 1.2. Separately optimizing SCCs A and B costs 211 operations from the formula in Figure 6 while optimizing the entire computation produces 63.67 operations. The reason why separate optimization does not perform well in this example is because there are too many intermediate outputs from SCC A to SCC B. The observation leads us to develop an approach of merging SCCs for further reduction of the number of operations. Since it is worthwhile to explain the SCC merging problem in detail, the next sub-section is devoted for the explanation of the problem and our heuristic approaches.

Since the subparts of a computation are unfolded separately by different unfolding factors, we need to address the problem of scheduling the subparts. They should be scheduled so that memory requirements for code and data of a schedule are minimized. We observe that the unfolded subparts can be represented by multi-rate synchronous dataflow graph [7] and the works of [1] can be directly used.

$$
\boxed{\begin{array}{l}
S[n+i] = A^{i+1}S[n-1] + A^{i}BX[n]+ \\
A^{i-1}BX[n+1] + ... + BX[n+i] \\
Y[n] = CS[n-1] + DX[n] \\
Y[n+1] = CAS[n-1] + CBX[n] + DX[n+1] \\
\qquad ... \\
Y[n+i] = CA^{i}S[n-1] + CA^{i-1}BX[n]+ \\
CA^{i-2}BX[n+1] + ... + CBX[n+i-1] + DX[n+i]
\end{array}}
$$

Figure 9: $i$ times unfolded state-space equations

### 4.2 SCC Merging

Initially, we only consider merging of SCCs. When two SCCs are merged, however, the merged SCCs does not form a SCC. Thus, in general, we must consider merging of any adjacent arbitrary subparts. Suppose we consider merging of subparts $i$ and $j$. The gain $GAIN(i,j)$ of merging subparts $i$ and $j$ can be computed as follows: $GAIN(i,j) = COST(i) + COST(j) - COST(i,j)$, where $COST(i)$ is the number of operations for subpart $i$ and $COST(i,j)$ is the number of operations for the merged subpart of $i$ and $j$. To compute the gain, $COST(i,j)$ must be computed, which requires constant coefficient matrices $A, B, C$, and $D$ for only the merged subpart of $i$ and $j$. It is easy to construct the matrices using the depth-first search [16].

The $i$ times unfolded system can be represented by the state-space equations in Figure 9. From the equations, the total number of operations can be computed for $i$ times unfolded subpart. Let $N(*,i)$ and $N(+,i)$ denote the number of multiplications and the number of additions for $i$ times unfolded system respectively. The resulting number of operations is $\frac{N(*,i)+N(+,i)}{i+1}$ because $i$ times unfolded system uses a batch of $i+1$ input samples to generate a batch of $i+1$ output samples. We continue to unfold until no improvement is achieved. If there are no coefficients of 1 or $-1$ in the matrices A, B, C, and D, then closed-form formula of the optimal unfolding factor $i_{opt}$ and of the number of operations for $i$ times unfolded system can be obtained. The formula are provided in Figure 10.

Now, we can evaluate possible merging candidates. We propose two heuristic algorithms for SCC merging. The first heuristic is based on greedy optimization approach. The algorithm is simple. Until there is no improvement, merge the pair of subparts which produces the highest gain.

The other heuristic algorithm is based on a general combinatorial optimization technique known as simulated annealing [6]. The details of the implementation is provided in our journal version.

Both heuristics perform equally well on all the examples and the run times for both are very small because the examples have a few SCCs. We have used both greedy and simulated annealing based

$$i_{opt} = \lfloor\sqrt{\frac{\sum_{j=1}^{n} S_j(2SS_j-1)}{\sum_{j=1}^{m} O_jIO_j}} - 1\rfloor, or$$

$\lceil\sqrt{\frac{\sum_{j=1}^{n} S_j(2SS_j-1)}{\sum_{j=1}^{m} O_jIO_j}} - 1\rceil$, which gives smaller value of

$$i_{opt}(\sum_{j=1}^{m} O_jIO_j - \frac{\sum_{j=1}^{n} S_j(2SS_j-1)}{i_{opt}+1}).$$

$$N(*, i) = \sum_{j=1}^{n}(S_jSS_j + (i+1)S_jIS_j) + \sum_{j=1}^{m}((i+1)O_jSO_j + \frac{(i+1)(i+2)}{2}O_jIO_j)$$

$$N(+, i) = \sum_{j=1}^{n}(S_j(SS_j - 1) + (i+1)S_jIS_j) + \sum_{j=1}^{m}((i+1)O_j(SO_j - 1) + \frac{(i+1)(i+2)}{2}O_jIO_j)$$

, where $m$ = # output groups, $n$ = # state groups

$S_j$ = # states in state group j

$O_j$ = # outputs in output group j

$IO_j$ = # inputs that output group j depends on

$IS_j$ = # inputs that state group j depends on

$SO_j$ = # states that output group j depends on

$SS_j$ = # states that state group j depends on

Figure 10: Closed-form formula for the number of operations after unfolding; If two outputs depend on the same set of inputs and states, they are in the same group, and the same is true for states.

heuristics for generating experimental results and they produced exactly the same results.

### 4.3 Explanatory Example: Putting It All Together

We illustrate the steps of the approach using an example. Consider the motivational example in Figure 1. Each node in the computation forms a SCC. Since every SCC is linear, unfolding is performed to optimize the number of operations for each SCC. The SCC A, B, C, D and E cost 120.75, 53.91, 114.86, 129.75, and 103.0 operations per input sample with unfolding factors 3, 10, 6, 7, and 2, respectively. The total number of operations per input sample for the entire computation is 522.27.

We now apply SCC merging to further reduce the number of operations. We first consider the greedy heuristic. The heuristic considers merging of adjacent subparts. Initially, the possible merging candidate pairs are AB, AC, BE, CD, and DE, which produce the gains of -51.48, -112.06, -52.38, 122.87, and -114.92, respectively. SCC C and SCC D are merged with a unfolding factor of 22. In the next iteration, there are now 4 subparts and 4 possible merging candidate pairs which produce minus gains. So, the heuristic stops at this point. The total number of operations per input sample becomes 399.4. Simulated annealing heuristic produced the same solution for this example.

The scheduling of the unfolded subparts is performed to generate the minimum code and data memory schedule. The schedule period is the least common multiple of (the unfolding factor+1)'s which is 3036. Let W denote the merged subpart of SCC C and D. While a simple minded schedule (759A, 276B, 132W, 1012E) to minimize the code size ignoring loop overheads generates 9108 units of data memory requirement, a schedule (759A, 4(69B, 33W, 253E)) which minimizes the data memory requirement among the schedules minimizing the code size generates 4554 units of data memory requirement.

# 5 Multiple Programmable Processors Implementation

When multiple programmable processors are used, potentially more savings in power consumption can be obtained. We summarize the assumptions made in the Section 2: (i) processors are homogeneous, (ii) inter-processor communication does not cost any time and hardware, (iii) effective switched capacitance increases linearly with the number of processors, (iv) both addition and multiplication take one clock cycle, and (v) supply voltage can not be lowered below threshold voltage $V_t$, for which we use several different values between 0.06V and 1.1V. Based on the assumptions, using $k$ processors increases the throughput $k$ times when there are enough parallelism in the computation, while the effective switched capacitance increases $k$ times as well. In all the real-life examples considered, there actually exists sufficient parallelism for the numbers of processors that we used.

$$\boxed{\begin{array}{l} \frac{k}{2}\log(1+\frac{k}{R}) - \frac{R}{2}(\frac{1}{2})^{\lceil \log(1+\frac{k}{R})\rceil}, \text{ if } k \le R^2 \\ \frac{R^2}{2}\log(1+R) - \frac{R}{2}(\frac{1}{2})^{\lceil \log(1+R)\rceil} + R(1+\lceil \log(1+R)\rceil)(k-R^2), \text{ if } k > R^2 \end{array}}$$

Figure 11: Closed-form condition for sufficient parallelism when using $k$ processors for dense linear computation with $R$ states

We observe that the next states, i.e., the feedback loops $S[n] = AS[n-1]$ can be computed in parallel. Since all next states are computed in a maximally fast way, on the bottom of the tree there exists more parallelism. All other operations not in the feedback loops can be computed in parallel because they can be separated by pipeline delays. As the number of processors becomes larger, the number of operations outside the feedback loops gets larger to result in more parallelism. For dense linear computations, we provide the closed-form condition for sufficient parallelism when using $k$ processors in Figure 11. We note that though the formula are derived under the worst case scenario, the required number of operations outside feedback loops is small for the range of the number of processors used in the experiment. There usually exist more operations outside feedback loops than required for full parallelism.

Now one can reduce the voltage so that the clock frequency of all the $k$ processors is reduced by a factor of $k$. The power consumption of $k$ processors is reduced from that of a single processor by a factor of $(\frac{V_1}{V_k})^2$ where $V_k$ is a scaled supply voltage for $k$ processor implementation, and $V_k$ satisfies the equation $\frac{V_k}{(V_k - V_t)^2} = k * \frac{V_1}{(V_1 - V_t)^2}$ [2]. The observation leads that it is always beneficial to use more processors in terms of power consumption with a couple of limitations: (i) the amount of parallelism available limits the improvement in the throughput and the critical path of the computation is the maximum achievable throughput and (ii) when supply voltage approaches close to threshold voltage, the improvement in power consumption becomes very small which does not justify the cost of adding one more processor. With this in mind, we want to find the number of processors which minimizes power consumption cost-effectively in both standard CMOS technology and ultra low power technology.

| | | | number of processors | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $V_t$ | $V_{init}$ | $V_1$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0.7 | 3.3 | 3.3 | 0.89 | 0.91 | 0.95 | 1.00 | 1.06 | 1.12 | 1.19 | 1.25 | 1.31 |
| | | 2.0 | 1.12 | 1.28 | 1.45 | 1.62 | 1.79 | 1.96 | 2.12 | 2.28 | 2.45 |
| | | 1.0 | 1.63 | 2.22 | 2.80 | 3.38 | 3.94 | 4.50 | 5.05 | 5.60 | 6.15 |
| 0.3 | 1.3 | 1.3 | 0.92 | 0.96 | 1.02 | 1.08 | 1.16 | 1.23 | 1.30 | 1.38 | 1.45 |
| | | 1.0 | 1.04 | 1.14 | 1.27 | 1.39 | 1.52 | 1.64 | 1.76 | 1.89 | 2.01 |
| | | 0.7 | 1.24 | 1.49 | 1.75 | 2.00 | 2.24 | 2.48 | 2.72 | 2.95 | 3.19 |
| 0.1 | 0.5 | 0.5 | 0.87 | 0.87 | 0.91 | 0.95 | 1.00 | 1.06 | 1.11 | 1.17 | 1.22 |
| | | 0.4 | 0.95 | 1.01 | 1.08 | 1.17 | 1.25 | 1.34 | 1.43 | 1.51 | 1.60 |
| | | 0.3 | 1.09 | 1.24 | 1.39 | 1.54 | 1.70 | 1.85 | 2.00 | 2.15 | 2.30 |

Table 1: The values of PN products with respect to the number of processors for various combinations of the initial voltage $V_{init}$, the scaled voltage for single processor $V_1$, and the threshold voltage $V_t$

Since the cost of programmable processors and especially processors on ultra low power platforms such as SOI is high, the guidances for cost-effective design are important. We need a measure to differentiate between cost-effective and cost-ineffective solutions. We propose a $PN$ product, where P is the power consumption normalized to that of optimized single processor implementation and N is the number of processors used. The smaller the $PN$ product is the more cost-effective the solution is. If PN is smaller than 1.0, using N processors has decreased the power consumption by a factor of more than N. It depends on the power consumption requirement and the cost budget for the implementation how many processors the implementation should use. Table 1 provides the values of PN products with respect to the number of processors used for various combinations of the initial voltage $V_{init}$, the scaled voltage for single processor $V_1$, and the threshold voltage $V_t$. $V_{init}$ is the initial voltage for the implementation before optimization. We note that PN products monotonically increase with respect to the number of processors. From the Table 1, we observe that cost effective solutions usually use a few processors in all the cases considered on both the standard CMOS and ultra low power platforms.

Based on those observations, we have developed our strategy for the multiple processor implementation. The first step is to minimize power consumption for single processor implementation using the proposed technique in Section 4. The second step is to increase the number of processors until the PN product is below the given maximum value. The maximum value is determined based on the power consumption requirement and the cost budget for the implementation. The strategy produces solutions with only a few processors, in many cases single processor for all the real-life examples because our method for the minimization of the number of operations significantly reduces the number of operations and in turn the supply voltage for the single processor implementation, adding more processors does not usually reduce power consumption cost-effectively. Our method achieves cost-effective solutions with very low power penalty compared to the solutions which only optimize power consumption without considering hardware cost.

## 6 Experimental Results

| Design | Init Ops | [15] | New Method | Improvement Factor | Reduction % |
|---|---|---|---|---|---|
| DAC | 2098 | 2098 | 1327.83 | 1.58 | 36.7 |
| modem | 213 | 213 | 148.83 | 1.43 | 30.1 |
| GE controller | 180 | 180 | 105.26 | 1.71 | 41.5 |
| Audio Filter | 154 | N/A | 76.0 | 2.03 | 50.7 |
| Video Filter | 296 | N/A | 157.14 | 1.88 | 46.8 |
| VSTOL | 1686 | N/A | 876 | 1.92 | 47.9 |

Table 2: Results of minimizing the number of operations

Our set of benchmark designs include the following typical portable DSP, video, communication, and control applications: DAC - 4 stage NEC digital to analog converter(DAC) for audio signals; modem - 2 stage NEC modem; GE controller - 5-state GE linear controller; Audio Filter - analog to digital converter(ADC) followed by 14 order cascade IIR filter; Video Filter - two ADCs followed by 10-order two dimensional IIR filter; and VSTOL - VSTOL robust observer structure aircraft speed controller. DAC, modem, and GE controller are linear computations and the rest are nonlinear computations.

Table 2 presents the experimental results for minimizing the number of operations of real-life examples. The fifth column of Table 2 provides only the improvement factor of our method from the initial number of operations since [15] is either ineffective or inapplicable for all examples. Our method

| Design | $V_{init}$ | $V_t$ | New Volt | Pow Red |
|---|---|---|---|---|
| DAC | 3.3 | 0.7 | 2.50 | 2.75 |
| | 1.3 | 0.3 | 1.00 | 2.69 |
| | 0.5 | 0.1 | 0.38 | 2.80 |
| modem | 3.3 | 0.7 | 2.65 | 2.23 |
| | 1.3 | 0.3 | 1.05 | 2.19 |
| | 0.5 | 0.1 | 0.40 | 2.25 |
| GE controller | 3.3 | 0.7 | 2.39 | 3.25 |
| | 1.3 | 0.3 | 0.96 | 3.16 |
| | 0.5 | 0.1 | 0.36 | 3.31 |
| Audio Filter | 3.3 | 0.7 | 2.03 | 4.43 |
| | 1.3 | 0.3 | 0.88 | 4.45 |
| | 0.5 | 0.1 | 0.33 | 4.73 |
| Video Filter | 3.3 | 0.7 | 2.26 | 4.03 |
| | 1.3 | 0.3 | 0.91 | 3.90 |
| | 0.5 | 0.1 | 0.34 | 4.12 |
| VSTOL | 3.3 | 0.7 | 2.25 | 4.15 |
| | 1.3 | 0.3 | 0.90 | 4.02 |
| | 0.5 | 0.1 | 0.34 | 4.25 |

Table 3: Power minimization results on single programmable processor

has reduced the number of operations by an average factor of 1.76(average 42.3%) for the examples that previous techniques are either ineffective or inapplicable.

Table 3 presents the experimental results for minimizing power consumption on single programmable processor of real-life examples on various technologies. Our method results in power consumption reduction by an average factor of 3.48. For multiple processor implementations, Table 4 summarizes the experimental results for minimizing power consumption. We define threshold PN product $PN_T$ to be the value of PN product, where we should stop increasing the number of processors. When $PN_T = 1$, i.e., the power reduction by the addition of an processor must be greater than 2 to be cost effective, in almost all cases single processor solution is optimum. When $PN_T$ gets larger, the number of processors used increases, but the solutions still use a few processors which result in an order of magnitude reduction in power consumption. All the results clearly indicate the effectiveness of our new method.

# 7 Conclusion

We introduced an approach for power minimization using a set of compilation and architectural techniques. The key technical innovation is a compilation technique for minimization of the number of operations which synergistically uses several transformations within a divide and conquer optimization framework. The new approach not only deals with an arbitrary computation, but also outperforms

| Design | $V_{init}$ | $V_t$ | N = 1 | | $PN_T = 1$ | | $PN_T = 1.25$ | | $PN_T = 1.5$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | New Volt | Pow Red | N | Pow Red | N | Pow Red | N | Pow Red |
| DAC | 3.3 | 0.7 | 2.50 | 2.75 | 1 | 2.75 | 4 | 9.22 | 6 | 11.71 |
| | 1.3 | 0.3 | 1.00 | 2.69 | 1 | 2.69 | 3 | 7.05 | 5 | 9.67 |
| | 0.5 | 0.1 | 0.38 | 2.80 | 2 | 5.73 | 5 | 11.43 | 7 | 13.83 |
| modem | 3.3 | 0.7 | 2.65 | 2.23 | 2 | 4.56 | 5 | 9.07 | 7 | 10.97 |
| | 1.3 | 0.3 | 1.05 | 2.19 | 1 | 2.19 | 4 | 7.22 | 6 | 9.13 |
| | 0.5 | 0.1 | 0.40 | 2.25 | 2 | 4.72 | 5 | 9.64 | 8 | 12.62 |
| GE controller | 3.3 | 0.7 | 2.39 | 3.25 | 1 | 3.25 | 4 | 10.49 | 6 | 13.20 |
| | 1.3 | 0.3 | 0.96 | 3.16 | 1 | 3.16 | 3 | 8.05 | 5 | 10.92 |
| | 0.5 | 0.1 | 0.36 | 3.31 | 2 | 6.62 | 4 | 11.18 | 6 | 14.21 |
| Audio Filter | 3.3 | 0.7 | 2.03 | 4.43 | 1 | 4.43 | 2 | 7.99 | 4 | 12.38 |
| | 1.3 | 0.3 | 0.88 | 4.45 | 1 | 4.45 | 2 | 8.07 | 4 | 12.56 |
| | 0.5 | 0.1 | 0.33 | 4.73 | 1 | 4.73 | 3 | 12.31 | 5 | 16.84 |
| Video Filter | 3.3 | 0.7 | 2.26 | 4.03 | 1 | 4.03 | 3 | 10.32 | 5 | 14.04 |
| | 1.3 | 0.3 | 0.91 | 3.90 | 1 | 3.90 | 3 | 9.55 | 4 | 11.34 |
| | 0.5 | 0.1 | 0.34 | 4.12 | 1 | 4.12 | 4 | 13.25 | 6 | 16.66 |
| VSTOL | 3.3 | 0.7 | 2.25 | 4.15 | 1 | 4.15 | 3 | 10.60 | 5 | 14.40 |
| | 1.3 | 0.3 | 0.90 | 4.02 | 1 | 4.02 | 3 | 9.76 | 4 | 11.58 |
| | 0.5 | 0.1 | 0.34 | 4.25 | 1 | 4.25 | 4 | 13.66 | 6 | 17.18 |

Table 4: Power minimization results on multiple programmable processors; $PN_T$ - threshold PN product; N - # of processors

previous techniques for limited computation types. Furthermore, we investigated coordinated impact of compilation techniques and ultra low power technologies on the number of processors which provide optimal cost-power trade-off. The experimental results on a set of real-life designs clearly indicate the effectiveness of all proposed techniques and algorithms.

# References


[1] S. S. Bhattacharyya et al,"A scheduling framework for minimizing memory requirements of multirate signal processing algorithms expressed as dataflow graphs,", VLSI Signal Processing VI, pp. 188-196, 1993.

[2] A. P. Chandrakasan, S. Sheng, R. W. Broderson, "Low-power CMOS digital design", IEEE Journal of Solid-State Circuits, Vol. 27, No. 4, pp. 473-484, 1992.

[3] A. P. Chandrakasan et al, "Design considerations and tools for low-voltage digital system design", Design Automation Conference, pp. 113-118, 1996.

[4] G. Goossens et al., "Design of heterogeneous ICs for mobile and personal communication systems", International Conference on Computer-Aided Design, pp. 524-531, 1994.

[5] P.D. Hoang, J.M. Rabaey, "Scheduling of DSP programs onto multiprocessors for maximum throughput" IEEE Trans. on Signal Processing, Vol.41, No.6, pp. 2225-2235, 1993.

[6] S. Kirkpatrick, C. Gelatt, M. Vecchi, "Optimization by Simulated Annealing," Science, Vol. 220, No. 4598, pp. 671-680, 1983.

[7] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow" , Proceedings of the IEEE, Vol. 75, No. 9, pp. 1235-1245, 1987.

[8] C. E. Leiserson, J. B. Saxe, "Retiming synchronous circuitry," Algorithmica, Vol. 6, No. 1, pp. 5-35, 1991.

[9] M. Potkonjak, J. Rabaey, "Maximally fast and arbitrarily fast implementation of linear computations," International Conference on Computer-Aided Design, pp. 304-308, 1992.

[10] M. Potkonjak et al, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination", IEEE Trans. on CAD, Vol. 15, No. 2, pp. 151-165, 1996.

[11] J. Rabaey, et al., "Fast prototyping of data path intensive architecture", IEEE Design and Test, Vol. 8, No. 2, pp. 40-51, 1991.

[12] R. Schneiderman, "Wireless Personal Communications", IEEE Press, New York, NY, 1994.

[13] M. Sheliga, E.H.-M. Sha, "Global node reduction of linear systems using ratio analysis", International Symposium on High-Level Synthesis, pp. 140-145, 1994.

[14] D. Singh et al,"Power conscious CAD tools and methodologies: a perspective", Proc. of the IEEE, Vol. 83, No. 4, pp. 570-594, 1995.

[15] M. Srivastava, M. Potkonjak, "Power optimization in programmable processors and ASIC implementations of linear systems: transformation-based approach," Design Automation Conference, pp. 343-348, 1996.

[16] R. E. Tarjan, "Depth first search and linear graph algorithms," SIAM Journal on Computing, Vol. 1, No. 2, pp. 146-160, 1972.

[17] V. Tiwari, S. Malik, A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization", IEEE Trans. on VLSI Systems, Vol.2, No.4, pp. 437-445, 1994.