

HYPER-LP: A System for Power Minimization Using Architectural Transformations

Anantha P. Chandrakasan[†] Miodrag Potkonjak^{††} Jan Rabaey[†] Robert W. Brodersen[†]

[†]EECS Department, University of California at Berkeley

^{††}C & C Research Laboratories, NEC USA, Princeton

ABSTRACT

The increasing demand for "portable" computing and communication, has elevated power consumption to be the most critical design parameter. An automated high-level synthesis system, HYPER-LP, is presented for minimizing power consumption in application specific datapath intensive CMOS circuits using a variety of architectural and computational transformations. The sources of power consumption are reviewed and the effects of architectural transformations on the various power components are presented. The synthesis environment consists of high-level estimation of power consumption, a library of transformation primitives (local and global), and heuristic/probabilistic optimization search mechanisms for fast and efficient scanning of the design space. Examples with varying degree of computational complexity and structures are optimized and synthesized using the HYPER-LP system. The results indicate that an order of magnitude reduction in power can be achieved over current-day design methodologies while maintaining the system throughput; in some cases this can be accomplished while preserving or reducing the implementation area.

1. Introduction

The major VLSI design and research efforts until now have been focused on optimizing speed to realize computationally intensive real-time tasks such as video compression and speech recognition. As a result, many systems have successfully integrated various complex signal processing modules meeting users computation and entertainment demands. While these solutions have provided answers to the real-time problem, they have not addressed the rapidly increasing demand for portable operation. This strict limitation on power dissipation which portability imposes, must be met by the designer while still meeting ever higher computational requirements.

An example of a system requiring portability, moving beyond today's portable computers, is a future personal communications terminal described in [1] that will support speech communication and recognition, data transfer, computing services, and high-quality, full-motion video. The intense computational nature of the terminal functions coupled with the requirement of portability will place severe constraints on the total power being consumed. For example, a basic terminal with speech recognition and video decompression units implemented using current day technology will require about 20lbs of battery for 10hrs of operation [1]. Clearly, more power efficient means for implementing these functions need to be developed. One major degree of freedom available in optimizing design for such applications is that once real-time operation is achieved, there is *no* advantage in making the computation any faster. The goal then becomes one of reducing the power

consumption while maintaining the system throughput.

Fortunately, the increasing density of VLSI systems, due to sub-micron feature size scaling and high-density packaging such as multichip modules, has enabled the development of an architectural strategy which can be used to trade-off area and power for a fixed throughput [2].

In this work, we attack the problem of automatically finding computational structures that results in the lowest power consumption for a specified throughput given a high-level algorithmic specification. The basic approach is to scan the design space by utilizing various flowgraph transformations, high-level power estimation, and efficient heuristic/probabilistic search mechanisms. While transformations have been successfully applied recently in high-level synthesis with the goal of optimizing speed and/or area, the problem of power optimization has not been addressed. It will be shown that optimizing for power using transformations requires a different strategy than those used for speed or area optimization.

2. Sources of Power Consumption

In CMOS technology, there are three sources of power dissipation arising from: switching (dynamic) currents, short-circuit currents, and leakage currents. The switching component, however, is the only one which cannot be made negligible if proper design techniques are followed. The power consumption due to the switching of a CMOS gate with a load capacitor, C_L , is given by the following formula [3]:

$$P_{\text{switching}} = p_t (C_L * V_{dd}^2 * f) \quad (\text{EQ 1})$$

where f is the clock frequency, V_{dd} is the supply voltage, and p_t is the probability of a power consuming transition (or the activity factor). Probabilistic approaches have been proposed to estimate the internal node activities of a network given the distribution of the input signals (i.e. the probability the value is a "1" or "0") [4,5].

In our analysis and optimizations, we will refer to the energy per computation of a gate or module (e.g. an adder), which is given by:

$$\text{Energy} = P_{\text{total}} / f_{\text{clk}} = C_{\text{avg}} V_{dd}^2 \quad (\text{EQ 2})$$

where C_{avg} is the average capacitance being switched per clock cycle (i.e. $C_{\text{avg}} = p_t * C_{L\text{total}}$).

The energy consumed by a logic block per computation is therefore a quadratic function of the operating voltage, as verified experimentally for a number of logic functions and logic styles in [2]. It is clear that operating at the lowest possible voltage is most desirable, however, this comes at the cost of increased delays and thus reduced throughput. This is seen from Figure 1 which shows an experimentally derived plot of normalized delay vs. V_{dd} for a typical CMOS gate. Once again, the delay dependence on supply voltage was verified to be relatively independent of various logic

functions and logic styles [2].

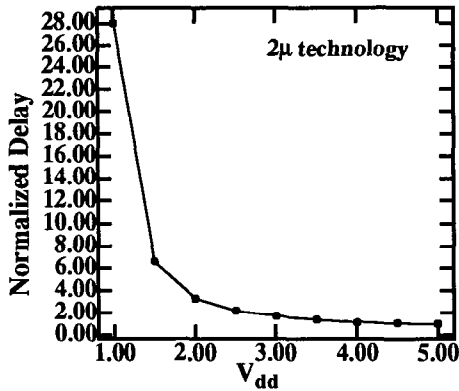


Figure 1: Plot of normalized delay vs. V_{dd}.

By modifying the architecture through a variety of transformations, however, the throughput can be regained, and thus a power savings can be accomplished while retaining the required functionality. It is also possible to reduce the power by choosing an architecture that minimizes the effective capacitance; through reductions in the number of operations, the average transition activity, the interconnect capacitance, and internal bit widths and using operations that require less energy per computation. It is these two strategies which will be pursued to minimize the power dissipation. There is, however, a strong interaction between optimizing capacitance and voltage for a fixed throughput. A lot of transformations will have conflicting effects on these parameters making the optimization a non-trivial task.

3. Transformations for Optimizing Power

Transformations are changes to the computational structure in a manner that the input/output behavior is preserved. The use of transformations makes it possible to explore a number of alternative architectures and to choose those which result in the lowest power. A brief summary of transformations for optimizing power is presented in this section. A detailed discussion of the effects of transformations on power is presented in [6].

3.1 Critical Path Reduction

This is probably the single most important type of transformations for power reduction. It is not only the most common type of transformation, but also often has the strongest impact on power. The basic idea is to reduce the critical path, so that supply voltage can be lowered while keeping the throughput fixed. The reduction of critical path is most often possible due to the exploitation of concurrency. Many transformations profoundly affect the amount of concurrency in the computation including pipelining and loop unrolling.

To illustrate the reduction of power using speedup techniques, consider a module with capacitance C running at a maximum frequency of $f @ 5V \Rightarrow P_{ref} = C (5)^2 f$. By transforming this structure to a parallel architecture with two identical units (unrolling), the clock frequency can be dropped to half the original rate while maintaining the original throughput. Since the modules have twice the available time as the original case, the voltage can be dropped to 2.9V (where the delays increase by a factor of 2, Figure 1). The power of the transformed solution is $P_{par} = 2C (2.9)^2 f / 2$ and a factor of $(5 / 2.9)^2$ reduction in power is achieved without sacrificing performance. The above analysis assumed that there is no overhead for parallelizing. In reality, the overhead due to routing and control must be taken into account. Even so, the quadratic dependence of voltage on power usually more than compensates for the

increase in capacitance resulting in an overall reduction of power. However at very low voltages ($< 1.5V$), the delays (and hence the overhead circuitry) increase very rapidly, causing the power to increase with further reduction of the supply voltage [2].

3.2 Reducing the Number of Operations

The most obvious approach to capacitance reduction, is to reduce the number of operations (and hence the number of switching events) in the data control flow graph. While this almost always has the effect of reducing the effective capacitance, the effect on critical path is case dependent. Transformations which directly reduce the number of operations in a data control flow graph include: common subexpression elimination, manifest expression calculation, loop merging, and distributivity.

3.3 Reducing the Transition Activity

Designs using static CMOS logic can exhibit spurious transitions due to finite propagation delays from one logic block to the next (called critical races or dynamic hazards). i.e. a node can have multiple transitions in a single clock cycle before settling to the correct logic value. The amount of extra transitions is a complex

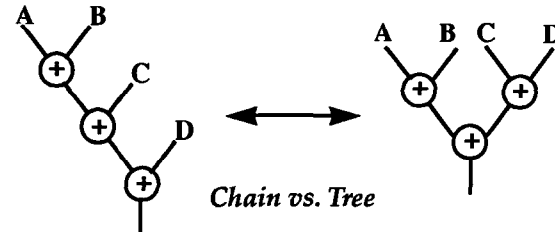


Figure 2: Reducing the glitching activity.

function of logic depth, input patterns, and skew. To minimize the "extra" transitions and power in a design, it is important to balance all signal paths and reduce the logic depth. For example, consider the two implementations for adding four numbers shown in Figure 2 (assuming a non-pipelined implementation). Assume that all primary inputs arrive at the same time. Since there is a finite propagation delay through the first adder for the chained case, the second adder is computing with the new C input and the previous output of $A + B$. When the correct value of $A + B$ finally propagates, the second adder recomputes the sum. Similarly, the third adder computes three times per cycle. In the tree implementation, however, the signal paths are more balanced and the amount of extra transitions is reduced. The capacitance switched for a chained implementation is a factor of 1.5 larger than the tree implementation for a four input addition and 2.5 larger for an eight input addition. The above simulations were done on layouts generated by the LagerIV silicon compiler [7] using the IRSIM [8] switch-level simulator over 1000 random input patterns.

3.4 Reducing the Interconnect Capacitance

It is often possible to reduce the required amount of hardware, while preserving the critical path (or number of control steps) [9]. This is possible because after certain transformations, operations are more uniformly distributed over the available time, and thus a denser scheduling (effective utilization of the hardware) can be achieved. These transformations include retiming for resource utilization, associativity, distributivity and commutativity. Smaller capacitance is achieved because there are fewer interconnects and/or fewer functional elements and registers, which are obstacles during floorplanning and routing, which indirectly influence interconnect area and capacitance.

3.5 Operation Substitution

Certain operations inherently require less energy per computation than other operations. A prime example of this is strength reduction, often used in software compilers, in which multiplications are substituted for additions.

Another powerful transformation in this category is converting multiplications with constants into shift-add operations. Since multiplications with fixed coefficients are quite common in many signal processing applications (DCT, FFT, various types of filters, etc.), this transformation can prove to be beneficial.

3.6 Bit-width Optimization

The number of bits used can strongly affect all the key parameters of a design, including speed, area and power. A smaller bit-width typically results in fewer switching events (and hence lower capacitance), faster circuits (and hence lower supply voltage), and smaller area (and hence lower average interconnect length). Certain transformations (e.g. associativity and distributivity) can have a profound impact on the bit-width.

4. Power Estimation

The goal is to develop an objective function that is highly correlated to the final (and unknown) power dissipation of the circuit. The objective function should be very easy to compute since it has to be evaluated many times during the optimization process. Elaborate power estimation, while being much more accurate, will require the hardware mapping and compilation steps to convert a flowgraph to layout, making it impractical during the optimization process. Hence a model correlated to the power must be developed strictly from the flowgraph level.

The goal of power optimization in this work is to keep the throughput constant by allowing the supply voltage to vary. Given that the sample period is fixed, power optimization is equivalent to minimizing the total energy switched, $C_{total} V^2$, where V is appropriate voltage required to meet the initial throughput rate.

4.1 Capacitance estimate

Estimating the total capacitance being switched involves considering four components:

$$C_{total} = C_{exu} + C_{registers} + C_{control} + C_{interconnect} \quad (EQ 3)$$

The capacitance estimation is built on top of an existing estimation routine in HYPER that determines the bounds and activity of various execution, register and interconnect components as well as the active implementations area [10]. A brief description of the estimation routines is presented below.

4.1.1 Execution Units and Registers

The capacitance contributed due to the execution units is determined by multiplying (over all types of units utilized) the number of times the operation was performed per sample period with the average capacitance of the unit type. The total capacitance is hence given by:

$$C_{exu} = \sum_{i=1}^{numtypes} N_i \cdot C_i \quad (EQ 4)$$

where $numtypes$ is the total number of operation types, N_i the number the times the operation of type i is performed per sample period, and C_i is the average capacitance being switched per operation of type i . The average capacitance for the various modules has been characterized as a function of bit-width (through SPICE and IRSIM simulations) for a uniformly distributed set of inputs.

In general the probabilities are not uniform, however, this assumption is made to simplify the cost evaluation. The effect of inter-module capacitance (between modules inside a datapath) is taken into account by incorporating an average loading capacitance during the characterization of the leaf-cells. It is important to note that the contribution due to the execution units is relatively independent of resource utilization (or the degree of time-sharing) since the required number of operations must be performed within the sample period. However, the amount of parallelism will affect the interconnect capacitance.

Registers are treated the same way as the execution units. The number of register accesses per sample period (read/write) is multiplied with the capacitance per register access to yield a register contribution given by $N_{registers} * C_{register}$. Once again, while the total number of registers is not important in calculating the register switching capacitance, it will affect floorplanning and therefore the interconnect capacitance.

4.1.2 Interconnect

A relatively accurate model for interconnect capacitance is important when performing power trade-offs since often the interconnects start to dominate over the logic capacitance and restricts improvement in power that can be achieved. Determining the interconnect capacitance is a difficult task since we have to in essence emulate the partitioning, place, and route.

The goal of interconnect estimation is to estimate the inter-block (between macro blocks, such as between datapaths) routing capacitance. A statistical model based approach is used to predict the inter-block capacitance from high level parameters such as the number of global interconnects, active area and bit-width. This model for estimating the average interconnect capacitance switched requires the number of interconnects, N , the average activity, α , and the average interconnect length, L . The average interconnect length is obtained from statistical estimates of the final routed chip area and typical interconnect length distributions as a function of area [11]. The interconnect capacitance is then estimated as:

$$C_{interconnect} = \alpha * N * L * B * C_L \quad (EQ 5)$$

where C_L is the capacitance per unit length and B is the bit-width. A more extensive model for the interconnect is currently being developed.

4.1.3 Control Logic

From several circuits, it was observed that there is a strong correlation between the control capacitance switched and the total relevant capacitance (muxes, tri-states, registers, and any other module that requires control signals). Based on this information, a simple model is used to predict the total control capacitance as a function of high-level parameters. Notice that control contribution will be a function of the architecture style used.

4.2 Supply Voltage Estimation

We are interested in computing the power supply voltage at which the transformed flowgraph will meet the timing constraints. The initial flowgraph which meets the timing constraints is typically assumed to be operating at a supply voltage of 5V with a critical path of $T_{initial}$ (the initial voltage will be lower if $T_{initial} < T_{sampling}$). After each move, the critical path is re-estimated, and the new supply voltage at which the transformed flowgraph still meets the time constraint, $T_{sampling}$, is determined. For example, if the initial solution requires 10 control steps running at a supply voltage of 5V, then a transformed solution that requires only 5 con-

trol steps can run at a supply voltage of 2.9V while meeting the throughput constraint. This relationship (of delay- V_{dd}) was modeled using Neville's algorithm for rational function interpolation and extrapolation [12].

5. Optimization Algorithm

The transformation mechanism is based on two types of moves, global and local. While global moves optimize the whole DCFG simultaneously, local moves involve applying a transformation only on one or very few nodes in the DCFG. The most important advantage of global moves is, of course, a higher optimization effect; the advantages of local moves is their simplicity and small computational cost. We used the following global transformations (i) retiming and pipelining for critical path reduction (ii) associativity (iii) constant elimination and (iv) loop unrolling. In the library of local moves we have implemented three algebraic transformations: associativity, distributivity and commutativity.

The computational complexity analysis of the power minimization problem showed that even highly simplified versions of the optimization tasks are NP-complete. Two widely used alternatives for the design of high quality suboptimal optimization algorithms are probabilistic and heuristic algorithms. Both heuristic and probabilistic algorithms have several distinctive advantages over each other. While the most important advantage of heuristic algorithms is a shorter run time, probabilistic algorithms are more robust and have stronger mechanisms for escaping local minimas.

The algorithm for power minimization using transformations has both heuristic and probabilistic components. While the heuristic part uses global transformations, the probabilistic component uses local moves. The heuristic part applies global transformations one at the time in order to provide good starting points for the application of the probabilistic algorithm. The probabilistic algorithm conducts a probabilistic search in a broad vicinity of the solution provided by the heuristic part. The underlying search mechanism of the probabilistic part is simulated annealing.

6. Results

A summary of power improvement after applying transformations relative to an initial solution that met the required throughput constraint at 5V for several representative examples is shown in Table 1. The results indicate that a large reduction in power consumption is possible (at the expense of area) compared to present-day methodologies. Also interesting was the fact that the "optimal" final supply voltage for all the examples was much lower than existing and emerging standards and was around 1.5V.

Example	Power Reduction	Area Increase
RGB -> YUV	8	5
FIR Filter	11	1.1
DCT (8 point)	8	5
Speech Filter	8	6.4
Elliptical Filter	9	2.7
Wavelet Filter	10	2

Table 1: Summary of results.

Example	Power Reduction	Area Increase
Volterra Filter	8.6	1

Table 1: Summary of results.

7. Conclusions

The problem of power minimization is becoming a very important problem with the increasing demand for "portable" computing and communication and we have presented a high-level synthesis system, HYPER-LP, for optimizing power consumption in application specific datapath intensive circuits using a variety of architectural and computational transformations. The synthesis approach consisted of applying transformation primitives (from a library of local and global moves) in a well defined manner in conjunction with efficient high-level estimation of power consumption. The results indicate that an order of magnitude reduction in power is possible over current-day design methodologies while maintaining the system throughput, and it was found that the optimal supply voltage for minimizing power was much lower than existing standards (present-day 5V and emerging 3.3V) and was around 1.5V for most of the examples investigated. While this work has addressed some key problems in the automated design of low-power systems, there are still many open research problems like detailed power estimation, module selection, partitioning, and scheduling for power optimization.

Acknowledgments

This project was funded by DARPA. We would like to thank Shan-Hsi Huang for coding and supporting the library of local transformation primitives.

References

- [1] A. Chandrakasan, S. Sheng, R.W. Brodersen, "Design Considerations for a Future Multimedia Terminal", in Third Generation Wireless Information Network, edited by D. Goodman and S. Nanda, Kluwer Academic Publishers, 1992.
- [2] A. Chandrakasan, S. Sheng, R. Brodersen, "Low-power CMOS Digital Design", IEEE Journal of Solid-state circuit, pp. 473-484, April 1992.
- [3] N. Weste and K. Eshragian, Principles of CMOS VLSI Design: A Systems Perspective, Addison-Wesley, MA, 1988.
- [4] F. Najm, "Transition Density, A Stochastic Measure of Activities in Digital Circuits", DAC, pp. 644-649, 1991.
- [5] A. Ghosh, S. Devadas, K. Keutzer, J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits", DAC, pp. 253-259, 1992.
- [6] A. Chandrakasan, M. Potkonjak, J. Rabaey, R. Brodersen, "An Approach to Power Minimization Using Transformations", IEEE VLSI Signal Processing Workshop, 1992.
- [7] R. W. Brodersen, (ed.), "Anatomy of a Silicon Compiler", Kluwer Academic Publishers, 1992.
- [8] A. Salz, M. Horowitz, "IRSIM: An Incremental MOS Switch-level Simulator", Proceedings of the 26th ACM/IEEE Design Automation Conference, June 1989, pp. 173-178.
- [9] M. Potkonjak and J. Rabaey, "Optimizing the Resource Utilization Using Transformations", Proc. IEEE ICCAD Conference, Santa Clara, pp. 88-91, November 1991.
- [10] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak, "Fast Prototyping of Data Path Intensive Architecture", IEEE Design and Test, Vol. 8, No. 2, pp. 40-51, 1991.
- [11] D. Schultz, "The Influence of Hardware Mapping on High-Level Synthesis", M.S. report, U.C. Berkeley, 1992.
- [12] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling: "Numerical Recipes in C", Cambridge University Press, 1988.