Input Space Adaptive Design: A High-level Methodology for Energy and Performance Optimization*

W. Wang†, A. Raghunathan‡, G. Lakshminarayana‡, N. K. Jha†
† Dept. of Electrical Eng., Princeton University, NJ 08544
‡ NEC, C&C Research Labs, Princeton, NJ 08540

Abstract

This paper presents a high-level design methodology, called *input space adaptive design*, and new design automation algorithms for optimizing energy consumption and performance. An input space adaptive design exploits the well-known fact that the quality of hardware circuits and software programs can be significantly optimized by employing algorithms and implementation architectures that adapt to the input statistics. We propose a methodology for such designs which includes identifying parts of the behavior to be optimized, selecting appropriate input sub-spaces, transforming the behavior, and verifying the equivalence of the original and optimized designs. Experimental results indicate that such designs can reduce energy consumption by up to 70.6% (average of 55.4%), and simultaneously improve performance by up to 85.1% (average of 58.1%), leading to a reduction in the energy-delay product by up to 95.6% (average of 80.7%), compared to well-optimized designs that do not employ such techniques.

1. Introduction

In this paper, we present a high-level design methodology and new design automation algorithms for optimizing energy consumption and performance through input space adaptive design. Our techniques can be applied to behavioral descriptions, RTL circuits, or in the context of traditional high-level design methodologies. Such designs suitably adapt to changing input statistics, resulting in significant energy and performance improvements for a wide class of designs.

Starting with a high-level behavioral description of the circuit to be optimized, and typical input traces that are used to profile the behavior and generate various statistics, we present techniques to perform the key steps involved in the design of input space adaptive circuits, which consist of the following steps: (i) identification of parts of the behavior that hold the highest potential for optimization, (ii) selection of input sub-spaces whose occurrence can lead to significant reductions in the implementation complexity, (iii) transformation of the behavior to realize performance and/or energy savings, and (iv) verification of the equivalence of the original and optimized designs.

The concept of input space adaptive design has been exploited in various related areas of research. Custom ICs, such as high-end microprocessors, often use information derived from data statistics to speed up program execution [1]. Branch prediction and control and data speculation are some common examples. Recent work in embedded system design has focused on adaptation of the memory hierarchy based on program execution statistics [2, 3]. Architectures that adaptively exploit input statistics have also been shown to improve performance and power consumption in the context of signal processing applications [4, 5, 6, 7]. Compiler optimizations such as constant propagation and folding, strength reduction, value range based optimizations, *etc.*, exploit information about values of program variables to eliminate or simplify operations in the program [8]. These techniques have also been employed in many high-level synthesis tools prior to scheduling and resource sharing [9]. The success of the above techniques, together with the need to bridge the quality gap between hand-crafted circuits and those generated by automatic tool flows, has fueled interest in exploring the use of data statistics in ASIC designs and design methodologies [10, 11, 12, 13]. The technique presented in [10] performs power optimization at the logic level by adding significantly simpler circuits (called predictor circuits) to compute the output, and disabling the original circuit, for a subset of input conditions. In high level synthesis, the individual components used to construct a circuit's architecture may be locally optimized to be input space adaptive (*e.g.*, through the use of variable-latency components [12]), however, such techniques do not exploit the global, larger scale potential for input space adaptive design.

2. Motivation and Design Issues

In this section, we illustrate the basic ideas, and detail the tradeoffs and issues involved in input space adaptive design, through illustrative examples.

2.1 Fundamentals

Figures 1 (a), (b) and (c) illustrate the concept of input space adaptive design. Figure 1(a) shows a behavioral description, B, with inputs $i_1, i_2, \ldots, i_n$, and outputs $o_1, o_2, \ldots, o_m$. The input-output space of a behavior is defined as a plot of all valid input-output values. In general, for

*This work was supported in part by Alternative System Concepts under an SBIR contract from Army CECOM and in part by DARPA under contract no. DAAB07-00-C-L516.

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

a behavior with n inputs and m outputs, the input-output space is $n+m$ dimensional. However, it is possible to examine sub-spaces of fewer dimensions if necessary. Figure 1(b) depicts a portion of the input-output space of behavior B in three dimensions. The shaded region shown in Figure 1(b) represents the sub-space, S, targeted for further simplification. The equation $i_2 > i_1$ represents this sub-space. Figure 1(c) illustrates how behavior B is modified to perform an optimized computation for the sub-behavior when sub-space S is encountered.

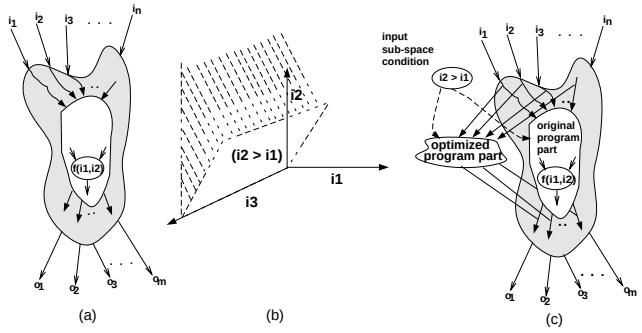

Figure 1: Illustrating input space adaptive design: (a) original behavior, (b) chosen input sub-space, and (c) optimized behavior

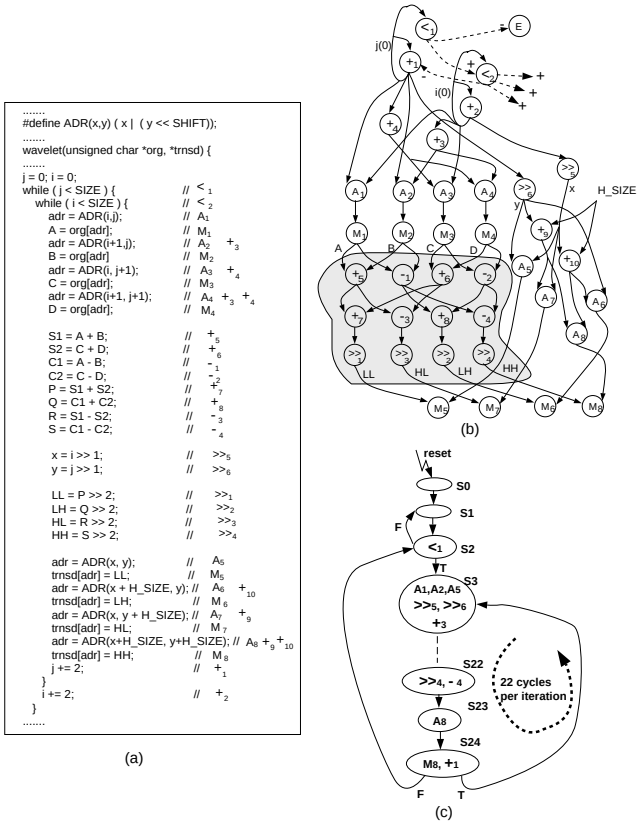

Figure 2: Discrete wavelet transformation: (a) behavioral description, (b) CDFG, and (c) schedule STG

We now illustrate the input space adaptive design using an illustrative example.

Example 1: Consider the behavior shown in Figure 2(a) that implements the discrete wavelet transformation, commonly used for compression of digital images. The control-data flow graph (CDFG) representation of the behavior is shown in Figure 2(b). The behavior reads the pixels of the input image from array *org*[] and outputs the transformed image data into array *transd*[]. Statements in the behavior are annotated with the names of the corresponding operations in the CDFG. For example, statement $S1 = A + B$ corresponds to operation $+5$ in the CDFG. Solid (dotted) lines are used in Figure 2(b) to represent data (control) dependencies.

The first step in input space adaptive design is to identify a sub-behavior for further optimization. For the `Wavelet` example, based on the techniques described later in Section 3, the shaded sub-behavior in Figure 2(b) is chosen for further optimization. The next step is to determine the target sub-space. Based on our analysis procedure, described in detail in Section 3, the target sub-space is described by the following equation

$$A = B = C = D \tag{1}$$

To qualify as an optimization target, the candidate sub-behavior and sub-space should have the following two properties: the behavior should be frequently executed and exhibit significantly reduced complexity when the inputs belonging to the target sub-space are encountered, and the chosen input sub-space should occur with a high frequency. The chosen sub-behavior in this case occurs with a very high frequency of 99.8%, since it is within the inner loop of the behavior. The sub-space described in Equation (1) occurs in the input trace with a frequency of 95.2%. The sub-behavior shaded in Figure 2(b) is now further optimized under the conditions imposed by the optimization condition described in Equation (1), in the following manner:

$$LL = A, LH = 0, HL = 0, HH = 0 \tag{2}$$

Thus, the original sub-behavior consisting of four additions, four subtractions, and four shift operations is replaced by the assignments to 0 or A, as given in the above equation. Since the chosen sub-behavior displays highly reduced complexity under the chosen input sub-space, significant performance enhancements and energy savings can be obtained. Figure 3(a) shows the `Wavelet` behavior, modified by input space adaptation, and Figure 3(b) shows the corresponding CDFG.

The shaded portions of Figure 3(b) indicate (i) the operations that are added to the behavior to test for the occurrence of input vectors belonging to the sub-space, and (ii) the optimized sub-behavior that is executed when the input sub-space condition is satisfied.

Figures 2(c) and 3(c) show the schedules generated for the original and optimized versions of the `Wavelet` example, respectively, as state-transition graphs (STGs). As indicated in Figure 3(c), the schedule for the input space adaptive design requires 11 cycles/iteration of the inner loop when the optimized input sub-space is encountered, and 22 cycles/iteration otherwise. In contrast, the original design always requires 22 cycles/iteration for the inner loop. This results in a performance improvement of 1.95X and a *simultaneous* energy reduction of 1.70X. These savings come at a modest area overhead (increase in grid count from 13624 to 16005), due to an increase in multiplexers, registers, and control logic. ∎

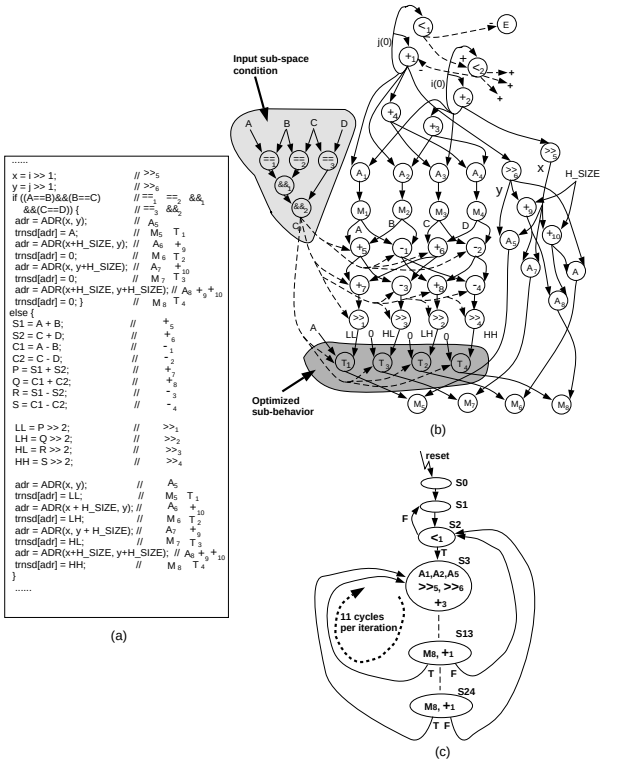

Figure 3: The `Wavelet` example optimized through input space adaptive design: (a) optimized behavioral description, (b) optimized CDFG, and (c) optimized STG

2.2 Issues and tradeoffs involved in input space adaptive design

This sub-section illustrates the tradeoffs involved in selecting the right sub-behaviors and input sub-spaces, underlining the need for a systematic design methodology to identify and exploit input space adaptive design opportunities. In general, the following factors are involved in this process: (i) Sub-behaviors that account for a larger portion of the total execution time and energy consumption of the design are better targets for optimization. (ii) For a given sub-behavior, input sub-spaces that occur with a higher probability may yield larger savings. (iii) Different input sub-spaces lead to different reductions in the complexity of the chosen sub-behavior. Use of the first two metrics alone is incomplete since it does not consider the potential for complexity reduction. (iv) Different input sub-spaces translate into differing hardware requirements for the circuitry that implements the input sub-space condition.

The tradeoffs caused by the above-mentioned factors and their inter-dependencies are illustrated through the next example.

Example 2: Consider again the `Wavelet` example presented in Figure 2. The input space adaptive design shown in Figure 3 was the result of choosing a specific sub-behavior and a specific input sub-space for optimization. In order to illustrate the issues involved in choosing the right sub-behavior and input sub-space, we performed the following experiment. We synthesized input space adaptive designs for the `Wavelet` example for different sub-behaviors ranging in size from 1 operation to 24 operations (in order to expose a sufficient number of operations to study this tradeoff, the inner loop in the behavior was unrolled to the necessary extent). We also considered 10 different input sub-spaces that occur with varying frequency. For each of the $24 * 10 = 240$ combinations of input sub-spaces and sub-behavior sizes, we derived input space adaptive designs. All 240 designs were subject to high-level synthesis and logic synthesis using a commercial design flow [14], mapped to a 0.35 micron cell-based array technology, and evaluated for energy consumption.

The results of our experiment, plotted as a 3-D surface, are shown in Figure 4. In this figure, the x axis denotes the number of operations in the chosen sub-behavior, and the y axis denotes the input sub-space chosen as the optimization condition. The z axis shows the energy consumption of the corresponding input space adaptive design. The curves plotted in the $z = 0$ plane indicate iso-energy contours, *i.e.*, all points on a curve correspond to designs with similar energy consumption. The minimum energy design corresponds to a sub-behavior of size 12, and the condition $A = B = C = D$. The variation in energy between the best design and the worst design represented in Figure 4 is 40.2%. Significant variations are present along both the x and y dimensions. ■

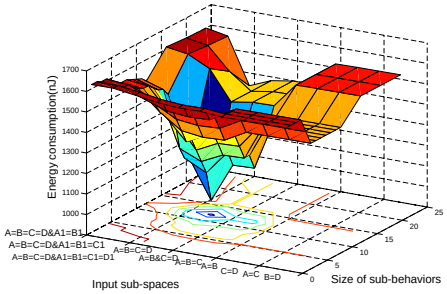

Figure 4: Energy consumption of input space adaptive designs for the `Wavelet` example with different sub-behaviors sizes and input sub-spaces

The following key observations can be made from the above example. (i) We can conclude that the energy consumption of input space adaptive designs strongly depends on the choice of the sub-behavior and input sub-space. (ii) The largest improvement in energy consumption does not necessarily occur at the largest or smallest sub-behavior size. (iii) The energy consumption initially improves with increasingly complex input sub-space conditions, but starts deteriorating after a point.

The above observations point to the need for a sufficiently accurate metric that can be used to identify the most promising candidate sub-behaviors and input sub-spaces. We have developed an entropy-based metric for identifying optimization opportunities. Our metric, which we term *optimization potential*, accounts for the impact of the factors mentioned above on the energy consumption of an input space adaptive design.

The entropy, E, of a variable, which can take one of N values, is described by the following equation

$$E = -\Sigma_{i=1}^{N} p_i \, log(p_i) \qquad (3)$$

In this equation, p_i is the probability that the variable takes the ith value. A random variable that is distributed uniformly in the range $[0, 2^{n-1} - 1]$ will have an entropy of n.

A variable that can take on two values with a probability of 0.5 each would have an entropy of 1, and a variable that has a constant value would have an entropy of 0. The above results suggest that entropy correlates well with a variable's information content.

Entropy is, in fact, used as a measure of information content of communication channels, and a significant body of work is devoted to the study of entropy and its implications on the design of several communication protocols, channel capacities, coding schemes, *etc.* [15]. The rationale behind this body of work derives from the fact that, for a given set of inputs, a lower output entropy implies that the outputs have lower information content, and can hence be realized by a simpler (more energy-efficient) circuit. This assumption has been tested and validated for a wide range of designs [16, 17]. Our work adapts the use of entropy-based metrics to identify optimization opportunities in our context.

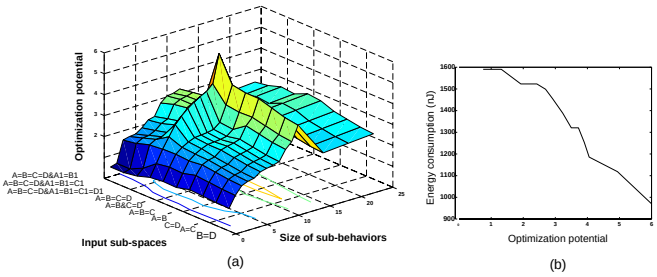

Figure 5: Plots of (a) *optimization potential* for different sub-behaviors and input sub-spaces, and (b) energy consumption *vs.* optimization potential

Example 3: Consider again the graph shown in Figure 4. We employed the entropy-based metric (presented later in Section 3) to evaluate each of the 240 designs represented in Figure 4. Figure 5(a) shows the optimization potential for different choices of sub-behaviors and input sub-spaces. Note that the optimization potential is highest in Figure 5(a) for exactly the same design point where energy consumption is lowest in Figure 4. The correlation between the optimization potential and energy consumption of the synthesized implementation is shown in Figure 5(b). From Figures 4 and 5, it is evident that design points with a larger optimization potential synthesize into lower-energy designs than those with a lower optimization potential. This example illustrates that our entropy-based metric (optimization potential) is able to easily identify promising optimization opportunities that correlates well with energy consumption savings. ∎

3. Methodology and Algorithms

In this section, we present the overall methodology and algorithm details for our input space adaptive techniques. The inputs to the input space adaptive design algorithm are a CDFG representing the behavior to be optimized, typical input traces, and designer-specified values for a set of optimization parameters. Parameters k and m control the number of candidate sub-behaviors, and the complexity of their input sub-space conditions. The output of our algorithm is the optimized input space adaptive behavior, which can be synthesized using conventional high-level and logic synthesis tools.

Figure 6 presents an overview of our algorithm and the steps involved. We first simulate the behavior with the given

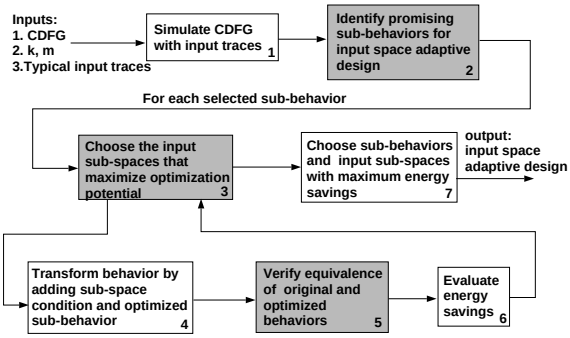

Figure 6: The input space adaptive optimization algorithm

input traces and extract various statistics that are used to drive the algorithm (**Step 1**). In addition to profiling statistics such as the execution counts of each operation, we also compute the entropy for each variable in the behavior, using Equation (3).

Step 2 identifies and ranks the k most promising sub-behaviors as candidates for optimization. We employ a **bottom-up** method. To form a candidate sub-behavior, we start with a single operation, and we continue to incorporate neighboring operations until no further benefit can be achieved. The sub-behaviors are evaluated by the *Gain* function. This function measures the desirability of a sub-behavior, σ, with input I and output O to be optimized by our input space adaptive technique.

$$Gain(\sigma) = \frac{trace\_coverage(\sigma)}{min(avg\_entropy(O), avg\_entropy(\frac{O}{I}))} \quad (4)$$

$$= \frac{C(\sigma)}{min(Q_\sigma(O), Q_\sigma(\frac{O}{I}))} \quad (5)$$

Trace coverage is defined as the fraction of operations from the complete dynamic execution trace of the behavior that belong to sub-behavior σ. Let *Num* be the number of all the operations in the original behavior, and *Num(σ)* be the number of operations in σ. We then have:

$$trace\_coverage(\sigma) = C(\sigma) = \frac{Num(\sigma)}{Num} \quad (6)$$

For a given sub-behavior, σ, we evaluate the average entropy value under the whole input space:

$$avg\_entropy(O) = Q_\sigma(O) = -\frac{1}{N}\Sigma_{i=1}^{N} p_i log(p_i) \quad (7)$$

Each time we group a neighboring operation, we evaluate the *Gain* value. We stop the growing process when *Gain* is maximized. We then start with an operation outside the generated sub-behaviors and repeat the above process until all the operations are in one of the sub-behaviors. The identified candidate sub-behaviors are further explored in **Steps 3-7**. For a given sub-behavior, **Step 3** identifies the input sub-spaces that would lead to maximum energy reductions. We start from the two-term optimization conditions (sub-spaces, in our terminology): $I_i = (\neq, <, >, \geq, \leq)I_j; i, j = 1, 2, \ldots, N$. I_i and I_j are two of the N inputs of the sub-behavior. The relationship can be $=, \neq, <, >, \geq, \leq$. It is computationally too expensive to try all these conditions. Therefore, we employ an entropy-based metric, *optimization potential (OP)*, to evaluate the sub-space, ρ, for the given

sub-behavior, σ.

$$OP(\sigma,\rho) = Gain(\sigma) \times Prob(\rho) \quad (8)$$

$$= \frac{C(\sigma)}{min(Q_\sigma(O), Q_\sigma(\frac{O}{T}))} \times Prob(\rho) \quad (9)$$

The validity of optimization potential as the metric to select sub-spaces under the given sub-behaviors has been verified in Figure 5(b). For each sub-behavior, we select m conditions (two-term) that result in the biggest optimization potential. For the selected m conditions, we try all their combinations and retain those with correct simulation results. For example, in `Wavelet`, the sub-space $A = B = C = D$ is the combination of $A = B$, $B = C$, and $C = D$.

Step 4 applies optimizing transformations to simplify the chosen sub-behavior under the restrictions imposed by the selected input sub-space. Optimizing transformations have been extensively studied in the literature in the context of compilers [8] as well as high-level power optimization [18, 19].

Step 5 verifies the correctness of the optimization by verifying the equivalence of the original and optimized behaviors. We formulate this problem as one of justifying a 1 at the output of a verification circuit shown in Figure 7.

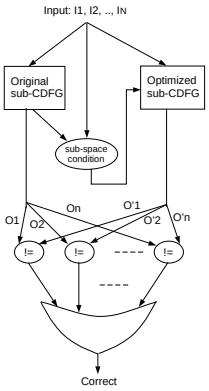

Figure 7: The circuit used to verify equivalence of the original and input space adaptive behaviors

Step 6 evaluates the energy savings obtained. This is performed by adapting behavioral power models that have been proposed in the literature [20, 21]. **Step 7** selects the optimizations that result in the highest energy savings.

4. Experimental Results

We applied our input space adaptive technique to several behavioral benchmarks. Typical input traces were assumed to be available for all the behaviors. The original CDFGs were modified by applying our technique. The original and optimized behaviors were subject to high-level synthesis and logic synthesis using state-of-the-art commercial tools [14], and mapped to NEC's 0.35 micron cell-based array library [14]. The resulting gate-level circuits were compared with respect to the following metrics: *area*, *performance*, and *energy*. These metrics were extracted from the technology-mapped circuits and designer provided testbenches using a commercial design flow [14]. The results obtained are summarized in Tables 1 and 2. Of our benchmarks, `Wavelet`, which implements the discrete wavelet transform, was discussed in Section 2. The `Poly` example performs the multiplication of two polynomials. `Matrix` represents the multiplication of two matrices. `Mergesort` implements the merge-sort algorithm. Finite impulse response (`FIR`) filter and discrete cosine transform (`DCT`) are well-known signal processing benchmarks.

In Table 1, major columns *Circuit*, *Area*, *# cycles*, and *Execution time* represent the name of the behavior, area (cell array grid count), number of clock cycles per input trace, and execution time, respectively. Minor columns *original* and *optimized* represent, respectively, the original and input space adaptive designs. Column *A.O.* represents the area overhead incurred by our technique, and column *P.I.* represents the improvement in performance. The average area overhead is 6.0%, whereas the average performance improvement is 58.1% (the averages were calculated based on comparing the sum of the values in the respective columns).

Table 2 presents the results for energy consumption of the original and optimized designs. If V_{dd}-scaling is performed, the optimized design is assumed to take the same time as the original design. This enables us to use the following equation [22] to scale the supply voltage

$$\frac{V_{dd}initial}{(V_{dd}initial - V_t)^2} \times T_{orig} = \frac{V_{dd}new}{(V_{dd}new - V_t)^2} \times T_{opt}$$

where $V_{dd}initial$ is the initial supply voltage, $V_{dd}new$ is the new supply voltage, and V_t is the threshold voltage of the implementation.

In Table 2, columns *energy* (non-V_{dd}-scaled), and *energy* (V_{dd}-scaled) represent, respectively, the energy consumption, without and with V_{dd}-scaling, over the entire duration of the input trace. The energy savings are shown under minor column *E.S.*. The average energy reduction before supply voltage scaling is 55.4%. The average energy reduction after using supply voltage scaling is 83.5%. The energy-delay product for the non-V_{dd}-scaled designs (product of the *Execution time* column from Table 1 and *energy* (non-V_{dd}-scaled) column from Table 2) is reduced by up to 95.6% (average of 80.7%).

5. Conclusions

In this paper, we presented a design methodology for optimizing performance and energy consumption through input space adaptive design. We presented algorithms to perform the important steps in input space adaptive deign, including selection of sub-behaviors to be optimized and the targeted input sub-spaces. Experimental results on several benchmarks using a commercial design flow demonstrated that input space adaptive designs perform significantly faster and consume significantly lower energy than the original designs, leading to over an order-of-magnitude improvement in the energy-delay product.

References

[1] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*. Morgan Kaufman Publishers, San Mateo, CA, 1989.

[2] D. Chiou, P. Jain, L. Rudolph, and S. Devadas, "Application-specific memory management for embedded systems using software-controlled caches," in *Proc. Design Automation Conf.*, pp. 416–419, June 2000.

[3] R. K. Gupta and A. Chien, "Architectural adaptation in MORPH," in *Proc. SPIE Workshop on Configurable*

Table 1: Area and performance results

Circuit	*Area (grid cnt)*			*# cycles*		*Execution time(μs)*		
	original	*optimized*	*A.O.*(%)	*original*	*optimized*	*original*	*optimized*	*P.I.* (%)
`Wavelet`	13,624	16,005	17.5	22,564	11,762	642.2	329.6	48.7
`Poly`	27,866	28,952	3.9	4195	625	102.6	15.3	85.1
`Matrix`	44,426	49,620	11.7	404	142	4.6	1.9	58.4
`Mergesort`	12,283	13,871	12.9	2,311	1,033	15.3	6.9	55.0
`FIR`	33,169	34,989	5.5	13,116	5,483	341.8	135.8	60.2
`DCT`	50,039	42,357	-15.4	4,840	2,740	136.6	80.3	41.3

Table 2: Energy results

Circuit	*Energy (nJ)* (non-V_{dd}-scaled)			*Energy (nJ)* (V_{dd}-scaled)	
	original	*optimized*	*E.S.* (%)	*optimized*	*E.S.* (%)
`Wavelet`	1641.3	967.0	41.0	437.2	73.4
`Poly`	133.4	39.2	70.6	6.0	95.5
`Matrix`	40.2	16.7	58.3	5.1	87.4
`Mergesort`	170.3	97.8	42.5	37.6	77.9
`FIR`	743.0	349.7	52.9	125.3	83.1
`DCT`	731.4	242.7	66.8	120.2	83.6

Computing, Oct. 1998.

[4] S. R. Park and W. Burleson, "Reconfiguration for power saving in real-time motion estimation," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 3037–3040, May 1998.

[5] S. Kim and M. C. Papaefthymiou, "Reconfigurable low energy multiplier for multimedia system design," in *Proc. IEEE Annual Wkshp. VLSI*, Apr. 2000.

[6] K. Lengwehasatit and A. Ortega, "DCT Computation with minimal average number of operations," in *Proc. Visual Communications and Image Proc. Conf*, Feb. 1997.

[7] P. Fernandez and A. Ortega, "An input dependent algorithm for the inverse discrete wavelet transform," in *Proc. Asilomar Conf. Signals, Systems, and Computers*, Nov. 1998.

[8] A. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Publishing Company, 1985.

[9] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York, NY, 1994.

[10] M. Aldina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou, "Precomputation-based sequential logic optimization for low power," *IEEE Trans. VLSI Systems*, vol. 2, pp. 426–436, Dec. 1994.

[11] L. Benini, E. Macii, M. Poncino, and G. De Micheli, "Telescopic units: A new paradigm for performance optimization of VLSI designs," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 220–232, Mar. 1998.

[12] V. Raghunathan, S. Ravi, and G. Lakshminarayana, "Integrating variable-latency components into high-level synthesis," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 1105–1117, Oct. 2000.

[13] G. Lakshminarayana, A. Raghunathan, K. S. Khouri, N. K. Jha, and S. Dey, "Common case computation: A high-level power optimization technique," in *Proc. Design Automation Conf.*, pp. 56–61, June 1999.

[14] *OpenCAD V 5 Users Manual*. NEC Electronics, Inc., Sept. 1997.

[15] R. B. Wells, *Applied Coding and Information Theory for Engineers*. Prentice Hall, Englewood Cliffs, NJ, 1998.

[16] D. Marculescu, R. Marculescu, and M. Pedram, "Information theoretic measures for energy consumption at the register-transfer level," in *Proc. Int. Symp. Low Power Design*, pp. 81–86, Apr. 1995.

[17] F. N. Najm, "Towards a high-level power estimation capability," in *Proc. Int. Symp. Low Power Design*, pp. 87–92, Apr. 1995.

[18] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 12–31, Jan. 1995.

[19] G. Lakshminarayana and N. K. Jha, "FACT: A framework for applying throughput and power optimizing transformations to control-flow intensive behavioral descriptions," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 1577–1594, Nov. 1999.

[20] R. Mehra and J. Rabaey, "Behavioral level power estimation and exploration," in *Proc. Int. Wkshp. Low Power Design*, pp. 197–202, Apr. 1994.

[21] K. S. Khouri, G. Lakshminarayana, and N. K. Jha, "High-level synthesis of low-power control-flow intensive circuits," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 1715–1729, Dec. 1999.

[22] A. R. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*. Kluwer Academic Publishers, Norwell, MA, 1995.