Post-silicon Resource Binding Customization for Low Power

Mehrdad Majzoobi
Farinaz Koushanfar
Electrical and Computer Engineering Department
Rice University, Houston, Texas

In this paper, we propose the first method for post-silicon customization of resource binding for low power application specific integrated circuits (ASICs) design. We devise and implement a new synthesis framework that generates a diverse set of resource binding candidates where any one of the candidates could be selected post-silicon. Orthogonal arrays are used to construct multiple candidates with diversified resource usage patterns. We tune the resource usage to match the unique power characteristic of each IC by selecting the best binding candidate that minimizes the pertinent chip's power consumption. We show efficient methods for embedding multiple binding candidates inside one design. Experimental evaluations on benchmark circuits demonstrate the effectiveness and low overhead of the proposed methods. For example, our post-silicon tuning achieves an average of 10% power savings on the benchmark circuits for variations present in 45nm technology. The power savings of post-silicon binding customization are expected to increase with scaling and miniaturization of CMOS feature sizes that inherently incur higher variations.

Categories and Subject Descriptors: B.1.4 [**Logic Design**]: Design Aids—*Optimization,Automatic synthesis*; B.1.2 [**Control Structures and Microprogramming**]: Control Structure Performance Analysis and Design Aids—*Automatic synthesis*

Additional Key Words and Phrases: High Level Synthesis, Low power, Post-Silicon Optimization, Resource Binding Customization

1. INTRODUCTION

Continuous scaling of CMOS to nanometer sizes and imperfections of the mask and process technologies in smaller scales have resulted in increased deviation of process characteristics from their nominal values. The intra-chip and inter-chip variations cause fluctuations in dynamic and static power consumption of ICs. The fluctuations are in particular significant for the leakage power that was shown to have an exponential dependence on some of the environmental and process parameters [Srivastava et al. 2005]. Several possible optimization techniques for minimizing the power consumption at all levels of design abstraction are available. Unfortunately, one-size-fits-all solutions that are based on optimizing the power consumption for

Author's address: Rice University, 6100 Main St. MS-380, Houston, Texas 77005. Email address: mehrdad.majzoobi@rice.edu, farinaz@rice.edu

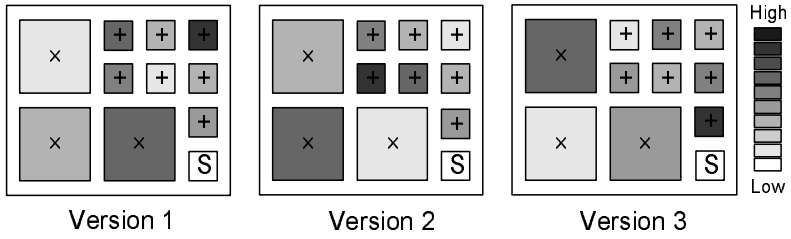


Fig. 1. The same benchmark with three candidates for task distribution.

nominal process values have a limited effectiveness in presence of variations across the chips.

Adaptive post-silicon optimization has been shown to be effective by tuning the design parameters to the specific characteristics of the ICs after fabrication. A standing challenge is applying post-silicon tuning to power control at different levels of abstraction since not all design parameters are adjustable at this stage. In fact, there are rather a few design parameters that have been so far tuned for low power post-silicon, including adaptive body biasing and standby input vector control [Alkabani et al. 2008]. Wang et al. [Wang et al. 2008] proposed pre-silicon module selection that is optimized to be variation-aware with post-silicon tuning by adaptive body biasing. However, module selection is not adjustable post fabrication. Presently available approaches for post-silicon power adjustment and optimization are mostly done at the device, gate or logic level [Mani et al. 2006; Kulkarni et al. 2006; Wang et al. 2008; Liu and Sapatnekar 2007].

Several studies have revealed the effectiveness of power optimization at a higher level [Raghunathan et al. 1998]. In particular, a number of approaches for low power design at the high-level synthesis stage have shown promising results. Early research on high-level synthesis for low power concentrated on dynamic power optimization [Chandrakasan et al. 1992; Raghunathan et al. 1998]. Scaling of CMOS to smaller sizes is increasing the significance of static leakage power compared to dynamic power. As a result, the emphasis of power-efficient methods shifted to leakage minimization in recent years [Khouri and Jha 2002]. We note that design space exploration during synthesis is usually done by deterministic optimization. For example, conventional schedules that are optimized and hardcoded in design are different from (i) the instruction scheduling in heavily pipelined circuit where the instruction priorities may change [Ndai et al. 2008], (ii) schedule selection in reconfigurable environments, or (iii) the application-level scheduling of multiprocessors performed in software [Wang et al. 2008].

In this paper, we introduce a high level synthesis methodology that generates multiple resource binding candidates instead of one. Each candidate assigns different usage levels to functional units. For example, Figure 1 shows three binding candidates for the same circuit where the functional units are being used at different rates in each candidate (the usage level is illustrated by the degree of darkness for each functional unit). The multiple candidates are efficiently embedded into the controller with a low overhead. Using conventional testing methods, the total chip power for binding candidates could be measured post fabrication. For each

IC, we select the binding candidate that minimizes the pertinent chip's power consumption. The best candidate, in effect, is the one that uses the power-inefficient functional units the least. Note that testing can be performed at runtime such that the system automatically picks the version that prolongs the battery life.

Our new methodology can be used for a number of other objectives, including balanced aging and defect tolerance. For instance, in case of failure or malfunctioning of a functional unit, the circuit can switch to an alternate candidate that does not use the defective unit. This can be made feasible by introducing redundancies in the functional units and using the multiple candidate binding method. The binding candidates must be designed such that one of the functional units is not used in each candidate. As another example, in rapidly aging technologies, by switching among the binding candidates one could provide equalized usage of all functional units and thus, amortize the aging effects. Multiple binding candidate method is a realization of N-variant design. The works in [Alkabani and Koushanfar 2008; Alkabani et al. 2009] explore other applications of N-variant design in security and temperature balancing. The contributions of this paper are as follows:

—Introduction of the first multiple candidate resource binding framework for ASICs that allows post-silicon variation-aware selection of the best candidate;
—Efficient formation of a diverse set of resource binding possibilities using orthogonal arrays;
—Creation and evaluation of heuristic algorithms that create multiple diverse binding scenarios for a given set of scheduled operations and number of available functional units;
—Low overhead embedding of multiple binding candidates into the controller;
—Introducing new methods for low overhead post-silicon power measurement for cases where the number of candidates is much larger than the number of resources;
—Studies and evaluations of the impact of number of post-silicon candidates and power savings of the resulting variation aware optimization compared to the best available low-power pre-silicon binding methods.

The remainder of the paper is organized as follows. Section 2 provides the preliminaries and background. Section 3 presents the new synthesis framework and the flow of our approach. Section 4 explains the significance and relevance of process variation and task level assignment. Section 5 introduces the design of multiple binding candidates to achieve diverse usage profiles across different versions. In Sections 6 and 7, we present generation of multiple candidates and efficient integration into the controller state machine respectively. Post-silicon selection of the best candidate is introduced in Section 8. The evaluation results are presented in Section 9. Finally, Section 10 concludes the paper.

2. BACKGROUND AND RELATED WORK

High Level Synthesis. When designing complex circuits, it is no longer practical to start designing from low-level hardware primitives. Designers need high level tools to provide algorithmic and behavioral description of their designs. Given the high level description, the tools must automatically convert it to low level available hardware primitives with minimal input from the designer. Many high-level

hardware description languages were introduced based on C/C++, Java, or functional programming languages. Other examples of high-level hardware description languages include SystemC, Bluespec, ImpulseC, and Handel-C.

In general, conversion of high level description to low level hardware primitives involves (i) compilation and optimization, and (ii) scheduling, assignment, and binding. Figure 2 shows the generic flow of the high-level synthesis process. First, the design is described using a high-level language like C/C++. Then, the description is translated into a control dataflow graph (CDFG). The CDFG passes through initial compiler optimizations to make it ready for hardware implementation. For instance, loops in the description code are unrolled, and dead codes are eliminated. Then, the CDFG along with designer's constraints are used to perform scheduling, allocation, and binding of the CDFG to generate the hardware. The hardware circuit is usually composed of a finite state machine (FSM) and a datapath described in a hardware description language like VHDL or Verilog. In the following, we briefly explain the CDFG, scheduling, allocation, and binding processes.

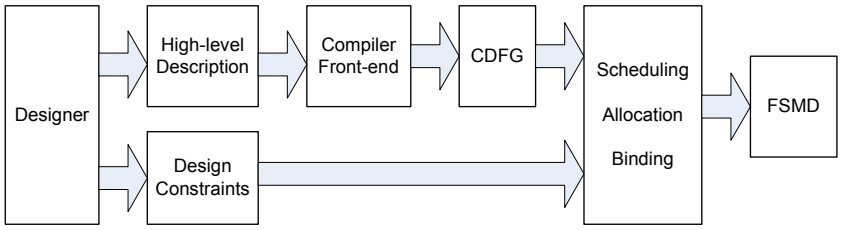


Fig. 2. Block diagram of a general flow for high level synthesis.

In particular a high level description code can be divided into basic code blocks connected by control modules representing branching and looping. These basic blocks can be represented as dataflow graphs (DFG). Figure 3(a) shows a sample DFG. A DFG can be defined as an acyclic graph $G_d = (V_d, E_d)$, composed of nodes denoted by V_d and edges denoted by E_d. Each node V_d represents an operation. Edges E_d represent the data dependencies between the nodes. In particular, the input edges represent operands to operations, and the output edges represent the result. A cyclic DFG can be converted to an acyclic DFG by unfolding and unrolling the cycles one or more times.

A CDFG shows how the DFGs are connected in code. Figure 3(b) shows a sample CDFG. The CDFG is defined as a graph with DFG subgraphs. The control nodes represent conditional switches. Based on the condition, data on the output of DFG_0 is moved to one of the two blocks, DFG_1 and DFG_2.

The scheduling involves analyzing the CDFG to decide in which clock cycle what operations will be executed. The scheduler can minimize the total runtime with a constraint on the number of resources or vise versa. Allocation involves selection of different resources from a library provided by the designer to be used for the hardware implementation. Different resources for the same operation may have a different area, power consumption, or delay. Binding is the process of mapping

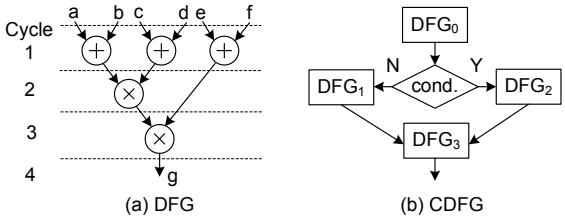


Fig. 3. Example of (a) dataflow graph (DFG) and (b) control flow graph (CDFG).

different data and operations to the functional and memory units as well as specifying the interconnect binding. The three design stages (i.e. scheduling, allocation, and binding) can sometimes be combined or interleaved. The terms 'resource' and 'functional unit' is used interchangeable in the remainder of the paper.

Process variation. Process variation is the fluctuation in the device parameters and characteristics caused by imperfections and uncertainties in the fabrication process. The device variability is caused by multiple factors, primarily threshold voltage variation, thin film thickness variation, line-edge roughness, and energy level quantization [Orshansky et al. 2007]. The total process variation can be viewed as the sum of inter-die and intra-die variations. Inter-die variations refer to differences among the dies and are constant within one die. Intra-die variations, on the other hand, account for the differences among devices on the same die. The intra-die component can be further divided into spatially correlated and uncorrelated random components. The uncorrelated random variations are caused by the fundamental intrinsic atomic-scale randomness of devices and materials, while systematic correlated components stem from unintentional shifts in processing conditions such as mask errors, lithographic off-axis focusing and reticle stepper alignment errors [Orshansky et al. 2007].

Thus, the process variation can be represented by Equation 1, where Φ_{nom} is the nominal parameter value, $\Delta\Phi_{inter}$, is the inter-die variation component, and $\Delta\Phi_{spatial}(x_i, y_i)$ and $\Delta\Phi_{random}(i)$ are the spatially correlated (systematic) and random components of intra-die variation for device i respectively [Srivastava et al. 2005; Koushanfar et al. 2008; Shamsi et al. 2008]:

$$\Phi = \Phi_{nom} + \Delta\Phi_{inter} + \Delta\Phi_{spatial}(x_i, y_i) + \Delta\Phi_{random}(i). \quad (1)$$

Differences in process parameters lead to variations in circuit level electrical properties, such as timing and power. Fluctuations in power consumption can be decomposed into variations in static leakage power and dynamic power. Variations of leakage, in particular, is reportedly much higher than dynamic power due to the exponential dependance of gate leakage current on process parameters (such as length, L, threshold voltage, V_{TH}, and gate oxide thickness, T_{OX}). Assuming a Gaussian distribution for process parameters, leakage variation follows a lognormal distribution. Dynamic power, on the other hand, is a function of node switching activities and associated node capacitances. Since gate capacitances linearly depend on device dimensions (L,W) that are typically assumed to be Gaussian, dynamic power exhibits a Gaussian distribution.

System-level power reduction techniques. For saving the dynamic power, two methods are typically used: Dynamic power management (DPM) and dynamic voltage scaling (DVS). DPM works by shutting-off the system components that are not in use. DVS runs different computations at different clock frequencies and voltages to fill-in the idle time slots in the schedule. Another approach for static power reduction is called input vector control [Abdollahi et al. 2004; Alkabani et al. 2008]. In this method, the input vector to a functional unit is fixed to a pre-defined value when the functional unit goes idle. The input vector is chosen such that it minimizes the leakage current of the corresponding unit. The proposed method in [Abdollahi et al. 2004] uses the built-in scan-chains in a VLSI circuit to drive the chip with the minimum leakage vector when it enters the sleep mode.

In our approach, DPM is used for saving the dynamic power during the schedule steps when a functional unit is not used. DVS is orthogonal to our approach and can be implemented in conjunction with our method but we did not employ it in this paper. For saving the static power, in addition to DVS, input vector control (IVC) can be used. We utilize the approach suggested in [Abdollahi et al. 2004; Alkabani et al. 2008] for applying the minimum leakage inputs to a sequential circuit.

3. FLOW

Figure 4 presents the flow of our approach. Our new methodology for generation of several binding candidates is integrated within the design flow. Assuming the input is in form of a CDFG, the method schedules the operations with a minimum number of resources to achieve a fixed pre-specified timing constraint.

Our new multi-candidate resource binding method creates a number of diverse binding choices (candidates) and simultaneously embeds the candidates into the controller with a very low overhead. The remaining steps of the design flow after binding are performed in the conventional way. The resulting GDS-II of the design is sent to a fabrication house. The chips are noninvasively tested post-silicon and the power profile of each chip is found for each binding candidate. Next, the power profile of the chip is compared against the available candidates and the best candidate for minimizing the chip's power profile based on its characterized properties is selected. To save test time and effort, we present a learning approach that alleviates the need for power measurement of each candidate if the number of candidates is much larger than the number of resources. Lastly, post-silicon selection could be stored on a small nonvolatile memory on the chip. Alternatively, burn-in fuses could be used for one-time programming of binding selection.

4. PROCESS VARIATION AND USAGE LEVEL ASSIGNMENT

A circuit typically consists of functional units and controller that schedules the operations. Each functional unit consumes both dynamic and static power. The dynamic power consumption of a functional unit is a linear function of its usage rate. A static current flows through the functional unit as long as it is on. By using DPM techniques, such as input vector control or power gating, a functional unit can enter leakage power saving mode when not actively used. In this case, leakage would also be linearly dependent on the usage rate. Therefore, the total

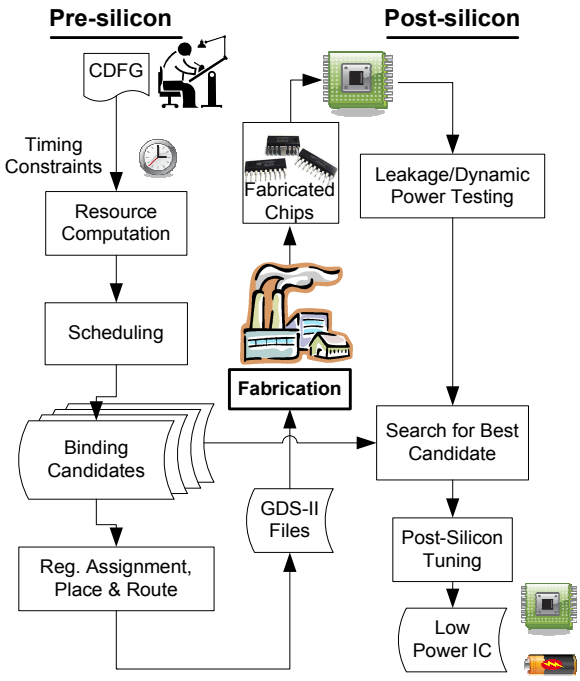


Fig. 4. The flow of the proposed approach.

power consumption of each functional unit can be formally expressed as:

$$P^{fu}_{total} = (1-\alpha) \times \beta \times P^{fu}_{leak} + \alpha \times (P^{fu}_{leak} + P^{fu}_{dyn}), \tag{2}$$

where P^{fu}_{total} is the total power consumption of a given functional unit; P^{fu}_{leak} and P^{fu}_{dyn} are the leakage and dynamic power of the functional unit while fully utilized; α is the usage rate of the functional unit and it is a number between 0 (never being used) and 1 (always being used); β in Equation 2, represents the leakage saving factor when the units are idle; β is a number between 0 and 1, where 0 corresponds to the case where units are completely shut down when being idle and 1 implies there is no leakage saving mechanism. The total chip power consumption can then be written as

$$P_{total} = P_{cnt} + \sum_{i=1}^{N_{FU}} P^{fu}_{total}(i), \tag{3}$$

where N_{FU} is the number of functional units (i.e. ALUs, multipliers, dividers,...), and P_{cnt} is the controller's average power consumption. In presence of manufacturing process variations, power consumption of functional units deviate from the nominal values. Therefore, it is desirable to assign less tasks to a unit that has a higher power consumption. Since, power usage is not known prior to fabrication, our proposed method creates a set of multiple binding candidates that have a diversely different usage profiles and chooses the one that achieves the lowest overall power consumption. The selection phase requires testing a set of candidates and

Run	Factor						
	1	2	3	4	5	6	7
1	+	-	+	-	+	-	+
2	-	+	+	-	-	+	+
3	+	+	-	-	+	+	-
4	-	-	+	+	-	-	+
5	-	-	-	+	+	+	+
6	-	+	+	+	+	-	-
7	+	+	-	+	-	-	+

Fig. 5. An example of a 2-level orthogonal array with 7 factors and strength 2

choosing the best one based on the collected data. In the next section, we explain how to make a group of binding candidates, each imposing a distinct usage level on each functional unit.

5. DESIGN OF BINDING CANDIDATES

We now discuss how to design a diverse set of resource binding candidates to enable post-silicon tuning of usage levels that match the specific underlying power variations of each functional unit. Individual adjustment of resource usage levels is not feasible, because of the following challenges. First, increasing the number of binding candidates would increase the overhead and the complexity of the approach, eventually defeating the savings of post-silicon tuning. Second, the CDFG constraints allow us to only heuristically assign the functional unit usages. Third, in resource constrained environment, all the functional units may always be needed to meet the application demand, leaving very little degree of freedom to change the usage assignment of the heavily used functional units.

To address the challenges of complexity and overhead, we employ orthogonal arrays to efficiently devise orthogonal and distinct task distribution scenarios for each circuit. Assuming that one (or a group of) functional units is controlled by one factor, then the usage rate of the unit can be defined by a factor level (e.g., low and high). By assigning independent factors to each functional unit, the bindings can be performed in a way such that functional units with assigned high usage factor level are used more often than those with assigned low usage factor level. A full factorial design is one that has all the possible combinations of factor level assignments to functional units. Such a design can be prohibitively large. For example, a full factorial design with 7 factors controlling 7 functional units with two usage levels requires 2^7 combinations of factor levels.

A fractional factorial design consist of a portion of full factors such that the factor-level assignment is done by an orthogonal array. An example of orthogonal array (of strength 2) with 7 factors and two levels for each factor is shown in Figure 5. Strength 2 means that for every two columns of the array, all combination of different factor levels (–, +-, -+, ++, -:low, +:high) occur equal number of times. Orthogonal arrays provide the most parsimonious (in the sense of the lowest number of combinations) set of design levels for studying the main effect of combination of a set of factors. Orthogonal arrays find usage across a diverse set of fields, the

theory behind them has been fully developed, and the orthogonal array tables can be readily adopted [Hinkelmann and Kempthorne 2007].

6. GENERATION OF BINDING CANDIDATES

In this section, we describe the methodology to construct the binding candidates that satisfy given task distributions. We present a heuristic method that attempts to achieve the closest task distribution that matches the desired input usage levels for each functional unit. The inputs to the algorithm are the circuit's CDFG, and target task distribution among the available functional units. The input task distributions are in fact the experiment runs (rows) of the orthogonal array. Therefore, the input task distribution specifies different relative usage levels for each functional unit (two levels in this case, i.e., low and high usage).

Alg. 1 Scheduling of operations.

Input: *CDFG*
Output: $Scheduled[type, cycle]$, for $cycle$ = 1,2,...,N_{cycle}

```
1  for type ∈ {mult, alu}
2       Find min N_type using list scheduling;
3       Find the mobility of each operation;
4       Ready ← Operations without predecessors;
5       cycle = 1;
6       Scheduled[type, cycle] = 0;
7       while there exists an unscheduled operation
8                 if the number of type operations in Ready < N_type
9                       Remove the scheduled operations from Ready
10                      Scheduled[type, cycle] = N_removed;
11                if the number of type operations in Ready > N_type
12                      Remove the scheduled operations from Ready giving
                            priority to operations with least mobility;
13                      Scheduled[type, cycle] = N_type;
14                Ready ← Operations whose predecessors are done;
15                cycle = cycle + 1;
16      end
17 end
```

Before performing task assignment to the available functional units, the operations are scheduled. In this work, we assume that operations are single-cycle. Although the method is demonstrated and evaluated on acyclic CDFG benchmarks, it can be easily generalized to CDFGs containing cycles. Cyclic CDFGs can be transformed into a pseudo-acyclic CDFG by unrolling the entire CDFG one or more times [Bhatia and Jha 1998]. The method needs to only determine what operations have to be performed in each cycle so that it can bind operations to resources to achieve a desired work balance and usage profile. In other words, the input to the binding algorithm is merely a list of operations to be performed in each cycle.

The scheduling algorithm outlined in Alg. 1 determines the lower bound on the number of functional units (N_{fu}) for each *type* of available functional units (i.e., $N_{fu} = N_{mult} + N_{alu}$) that satisfy the timing constraints (Line 2). The mobility of each operation is separately determined by performing ASAP (i.e. as soon as possible) and ALAP (i.e. as late as possible) scheduling on the $CDFG$ (Line 3). The scheduling algorithm then initializes the set of operations *ready* to be scheduled based on the dependencies of operations in $CDFG$ (Line 4). Next, it selects the operations with smaller mobility to schedule first. The scheduled operations are removed from the set *Ready* and the number of scheduled operations are stored in array *Scheduled* at each cycle. The number of schedules operations that are removed from the *Ready* set at once is always smaller than the total of number of available functional units. As a result, if the number of ready operations of a given *type* is larger than available functional units of the corresponding type, then N_{type} of them are schedule and the rest are postponed to be scheduled in the next cycles. The algorithm stops when all the operations are scheduled (Lines 7-14).

After scheduling, we would have (i) the total number of operation cycles N_{cycles}, and (ii) the number of operations to be performed by each *type* of functional units at each *cycle*, i.e., $Scheduled[type, cycle]$. As an example, if $Scheduled[mult, 5]$= 3, then it means three multiplications that must be performed in cycle 5.

Alg. 2 Binding the operations to achieve input usage level.

Input: TD and $Scheduled[type, cycle]$
Output: $Bound\{type, cycle\}$

```
1  for each resource type
2       for cycle = 1,2,...,N_cycles
3            Bound{type, cycle} = φ;
4            Free = set of available functional units;
5            while Scheduled[type, cycle] ≠ 0 i.e. an unassigned operation exists
6                 Pick a functional unit randomly; i ∈ Free;
7                 if the functional unit i accepts the operation
8                      Bound{type, cycle} ← i;
9                      Remove i from Free;
10                     Scheduled[type, cycle] = Scheduled[type, cycle] - 1;
11      end
12 end
```

Now by passing the matrix $Scheduled[type, cycle]$ to the binding algorithm, Alg. 2, it assigns the operations to the available functional units to satisfy a given input task distribution (TD). TD is a binary vector whose size is equal to the number of functional units (N_{fu}). In effect, each element of TD corresponds to one functional unit. If an element in TD has a value of one, then its corresponding functional unit is going to be used more than those whose corresponding values in TD are zero. The presented algorithm follows a probabilistic approach in distributing the tasks to the functional units. If a functional unit, as specified by TD, is expected to have a high usage rate, then the algorithm would assign a high accepting probability value (p_h) to it; otherwise, it would give the unit a low accepting probability value

(p_l). The algorithm randomly visits functional units at each cycle. The associated probability states the chance a functional unit accepts an operation at a given visit. For example, a probability 0.2 means that the functional unit would accept to perform the operation in five visits on average. If a functional unit accepts to do an operation, it would not be visited again in that cycle.

Lines 3 and 4 of Alg. 2 initialize an empty set denoted by $Bound\{type, cycle\}$ and a set of all available functional units denoted by $Free$. In Line 6, the algorithm picks a functional unit randomly from the set $Free$. Line 7, based on the assigned probability, checks to see if the functional unit **i** is willing to accept the operation. If so, the functional unit would be moved to the $Bound\{type, cycle\}$ set (Line 8) so it would not be visited in that cycle again (Line 9). The **while** loop (Line 5) continues until there are no operations to be assigned in that cycle. The same is repeated in the the next cycles and unit types (Lines 1 and 2). Note that the functional units with a higher level of willingness (higher probability of accepting the operation) end up having more tasks assigned to them.

Alg. 3 Generation of multiple binding candidates

```
1  Schedule CDFG using Alg. 1;
2  N_factor = N_mult + N_alu;
3  OA ← Design OA ( N_factor, N_run, N_level);
4  for v = 1, 2, ..., N_run
5        for f = 1, 2, ..., N_factor
6             if OA(v, f) = high
7                  Accepting Prob(f) = p_h;
8                  TD(f) = 1;
9             if OA(v, f) = low
10                 Accepting Prob(f) = p_l;
11                 TD(f) = 0;
12       end
13       Bind the operations using Alg. 2;
14 end
```

So far, we described the algorithm that binds the operations to functional units to achieve a pre-specified task distribution. Next, we use this algorithm to make diversely different binding candidates for a given circuit. As discussed earlier, we take advantage of orthogonal arrays to make the diverse candidates. We choose to control the usage level of each functional unit by a separate factor. For larger benchmark circuits, we group functional units and adjust each group's usage level by a single factor. Alg. 3 shows how different binding candidates are created for a given $CDFG$. Alg. 3 calls Alg. 2 to perform binding for the expected task distributions dictated by an orthogonal array. Line 1 schedules the operations and determines how many operations of each type are required at each cycle. Line 2 sets the number of design factors N_{factor}, to the total number of functional units, i.e., the number of multipliers (N_{mult}) and ALUs (N_{alu}). An orthogonal array with a specified number of factors (N_{factor}), rows (N_{run} which is equal to the total number of candidates), levels (N_{level} which equals two in this case for low and high) will be initialized at Line 3. Each row of the orthogonal array represents one binding

candidate. Low and high accepting probabilities are assigned based on the factor levels dictated by orthogonal array values in Lines 4, 5, 6, 7. The operations are assigned to functional units according to the assigned probabilities using Alg. 2. The same steps are repeated for the subsequent rows of the orthogonal array. Each iteration yields a unique binding candidate.

7. HARDWARE INTEGRATION OF BINDING CANDIDATES

In this section, we describe how the different binding candidates can be efficiently integrated and hard coded into one finite state machine (FSM) denoted by F_{tot} that simultaneously implements the multi-candidate binding. As shown in Figure 6 (a), F_{tot} accepts a special input for selecting the candidate to be enabled. We use F_s to refer to the FSM that controls one of the candidates. One of the candidates is chosen at random for this FSM. The FSMs for the different candidates are going to have the same inputs and the same number of control steps. The only difference between the various candidates is the output control signals that dictate which functional unit should be activated. The signal **Output** yields the correct control step for the selected candidate.

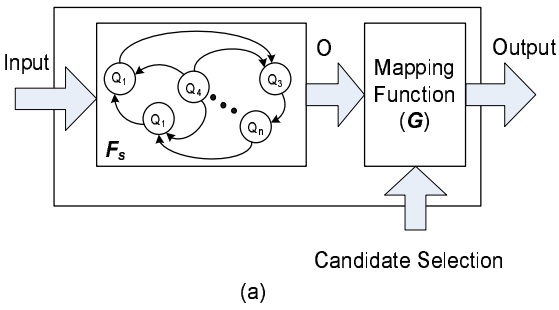
Therefore, a combinational mapping is constructed using a truth table that takes the candidate number and the control signals of F_s as inputs. Note that for example if F_s is built based on the first candidate, then when candidate selection input is equal to '1', the output of mapping logic is going to be the same as its input (i.e. the mapping acts like a wire). The following steps show how we generate F_{tot}: **(i)** Generate the hardware FSM for one of the candidates (F_s). **(ii)** Construct a mapping (truth table) that at each cycle, given the candidate number, maps the output control signal of F_s to the target control signal. **(iii)** Generate and optimize the combinational circuit (G) described by the mapping and connecting it to F_s.

Figure 6 (b) shows a sample truth table. The leftmost column values in the table shows the output of the reference FSM F_s, for each operation cycle. The values on the subsequent columns shows the F_{tot} expected outputs for each target candidate. The binary values indicate which functional unit must be active at a cycle. Notice that connecting the mapping logic to F_s will increase the worst case delay of the controller and introduces extra fan-out effect. In datapath-intensive designs, the speed of operation is dominated by the speed of functional units in the data path and corresponding bus interconnects. In such systems, a slight increase in latency of controller does not significantly affect the overall speed of operation. As a result, our method works best for datapath-intensive designs.

For maximum flexibility, we assume global registers are connected to functional units through on-chip busses. The busses are MUX-ed to enable the functional units to read their inputs from and write their outputs to the registers. The input to the MUXs are controlled by the controller which decides what operations to be done on which variables inside the registers.

8. POST-SILICON BINDING SELECTION

After fabrication, the total power consumption of a chip needs to be measured or estimated for each binding candidate. The best binding candidate that minimizes the power consumption would be then chosen. However, exhaustive testing of



O	Output			
	V_1	V_2	...	V_n
10110	11011	10100	...	01011
11010	11011	00000	...	00011
...	...	...	...	...
00101	11111	01110	...	01111

Truth Table

(b)

Fig. 6. (a) Efficient integration of multiple binding candidates into the controller. (b) Sample mapping truth table.

all candidates would be tedious, incurring a high test time for a large number of candidates. To save the test time and effort, we present a method that learns and characterizes the power consumption of each functional unit by only testing a small subset of candidate bindings. Once each functional unit's power characteristic is estimated, the method finds the best candidate by forming a linear combination of its functional units power consumptions and their usages.

To do the post-silicon power characterization, we select a subset of binding candidates and measure the average total power consumption for each candidate by applying a series of test input vectors to the circuit. Since the total power is a weighted sum of each functional unit's leakage and dynamic power as expressed by Equation 2, one can solve a system of linear equations to estimate each functional units's average (leakage and dynamic) power consumption. The solution to system of linear equations (i.e., P^i_{leak} and P^i_{dyn} for $i = 1,...,N_{fu}$) can be found by solving a linear programming optimization that minimizes l_2 norm of the measurement error e_m (for measurement m), i.e., $min \ \sqrt{\sum_{m \in S} e_m^2}$, subject to Equation 4 for all $m \in \mathcal{S}$, where $\mathcal{S} \subset \mathcal{V}$ and $\mathcal{V}$ is a set of all binding candidates. Assuming linear independency of the equations which is automatically established by the full rank of orthogonal array, $N_B \geqslant 2 \times N_{resouces}$ number of binding candidates will be sufficient to characterize each functional units leakage and dynamic power:

$$P_m = \sum_{i=1}^{N_{fu}} (1 - T^i) \times \beta_i \times P^i_{leak} + \alpha_i \times (P^i_{leak} + P^i_{dyn}) + e_m. \quad (4)$$

After estimating the power of each functional unit, we can evaluate other candidates by linear combination of the functional units' power consumptions/usages. The best candidate is the one achieving the least total power for the chip characteristics. However, in addition to power consumption criteria, we must check whether the interconnect delay affects the closure time for the selected candidate or not. During design and scheduling, we construct our candidates such that their delays (including the interconnects) statistically satisfy the desired circuit timing closure. Post-silicon timing characterization can identify those statistically rare binding candidates that violates the timing constraints due to the significant deviation of interconnect delays from their nominal values caused by process variations. A binding candidate that does not satisfy the timing closure will not be selected.

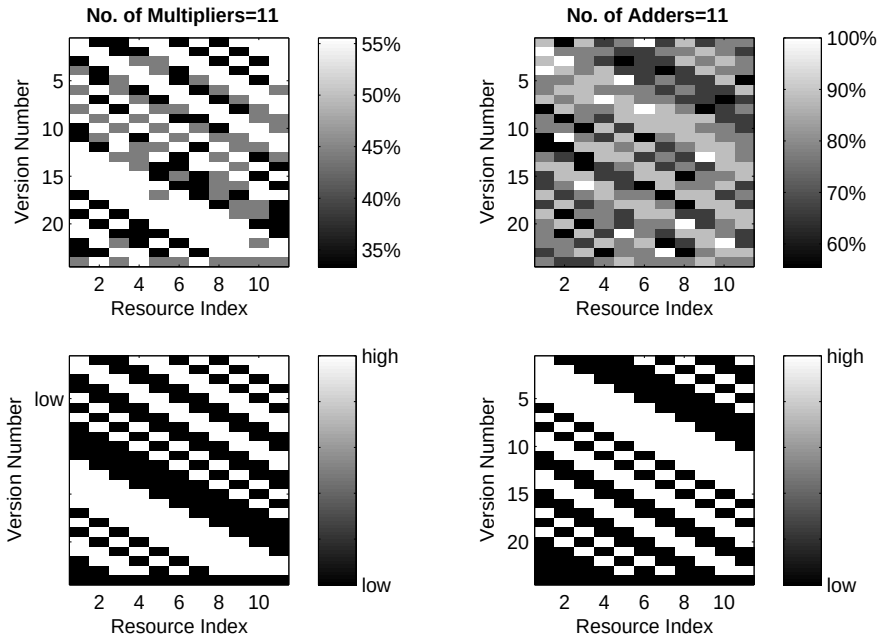


Fig. 7. The plots of expected (bottom) and achieved (top) task distributions for different candidates of $aircraft$ benchmark for the multiplier (left) and ALU (right) modules.

9. EXPERIMENTAL RESULTS

We evaluated the proposed multiple binding method on HYPER benchmarks [Rao and Yi 1990]. The first column of Table I shows the benchmark names. The benchmark circuits *arai*, *pr*, *wang*, *lee* and *dir* perform 8 point fast discrete cosine with different algorithms. *arai* implements Arai-Agui-Nakajima algorithm. *pr* performs DCT based on planar rotation transformation. *wang* benchmark is Suehiro-Hatori's version of the Wang Wang sparse planar rotation-based matrix factorization. Lee's recursive sparse matrix factorization algorithm is implemented in *lee* benchmark. *dir* represents the direct generic definition of DCT-I algorithm. *mcm* is an optimized version of *dir* with reduced number of multiplication. *honda* and $aircraft$ are two mechanical controllers of industrial strength.

We used NOA (Near-Orthogonal Array) tool from Gendex DOE toolkit [DOE Toolkit 6] and S-plus to generate orthogonal arrays of desirable sizes. As mentioned earlier, the number of factors directly depends on the number of functional units. The number of candidates (rows of the orthogonal array) for a complete orthogonal array depends on the strength and number of factors. NOAs can be generated for any given number of runs (candidates). The algorithms presented in Section 6 were evaluated on the benchmark circuits and different candidates were extracted accordingly.

Figure 7 shows the functional unit usage rates for different binding candidates of $aircraft$ benchmark for both multipliers and ALUs. The two lower matrices are the

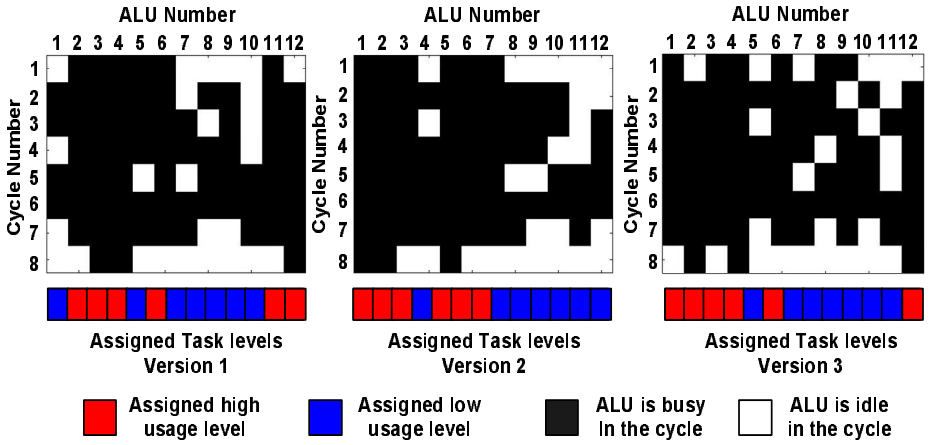


Fig. 8. Three binding candidates for the *honda* benchmark each satisfying a pre-imposed usage level pattern.

corresponding orthogonal arrays. The rows and columns in each array correspond to a binding candidate and a functional unit respectively. In other words, each row of the orthogonal array is one binding candidate. The white (dark) pixels in the orthogonal array implies that the associated functional unit is forced to be used (less) more frequently in the given version. The top plots show the resulting usage rates after enforcing the factor levels on the benchmark. The darkness level indicate the usage level of a functional unit in the resulting binding. It can be observed that the task distributions for different candidates (rows) are closely following the expected behavior dictated by the NOA. The left and right correspond to multipliers and ALUs respectively. The high and low accepting probabilities of Alg. 3 are set at $p_h = 0.8$ and $p_l = 0.1$.

Figure 8 illustrates the different resulting binding candidates for *honda* benchmark. Each row of a binding candidate corresponds to a cycle of operation and each column corresponds to one ALU. The black pixels indicate that the ALU is being used on the specific cycle. The imposed usage levels for each version is depicted by the vector below each binding candidate. The red color in the vector suggest that the associated functional unit is forced to take more tasks in all operation cycles. Notice how the operations are redistributed for each candidate in order to match the imposed usage levels.

Ideally, in an unconstrained system with unlimited number of functional units, the ones with an assigned high level usage are always used and those with assigned low level usage are always skipped and never used. But in presence of structural dependencies and limited number of resources, it could be impossible to hit 0% and 100% usage levels. It is, however, desirable to achieve the maximum usage level separation between the functional units with high and low assigned usage levels. Figure 9 shows for each benchmark and functional unit type, the distribution of

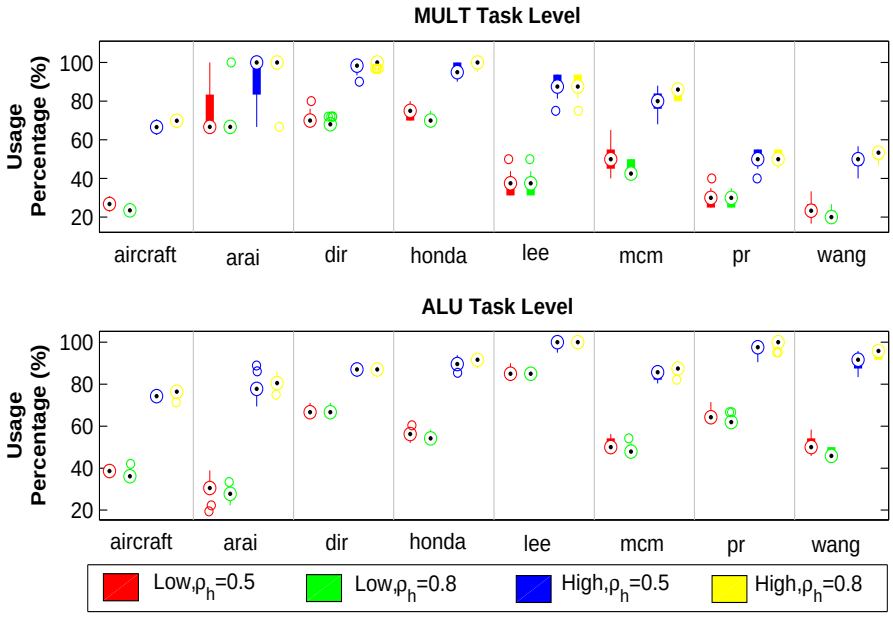


Fig. 9. The low and high usage levels across different versions for ALU and multipliers with ρ_h=0.5 and ρ_h=0.8.

low and high usage levels for different versions, i.e.,

$$U_l = \{u_{i,v} \mid i \in High,\ v \in \mathcal{V}\},$$
$$U_h = \{u_{i,v} \mid i \in Low,\ v \in \mathcal{V}\}, \tag{5}$$

where, $u_{i,v}$ is the usage of the i-th functional unit in the v-th binding candidate. $High$ and Low are the sets of functional units with high and low assigned usage rates respectively, and $\mathcal{V}$ is the set of all binding candidates. The experiment is repeated for accepting probabilities ρ_h=0.5 and ρ_h=0.8 while ρ_l=0.1 as defined in Alg. 3. The distribution of U_l and U_h are illustrated using boxplots. On each box, the central mark (the encircled dot) is the median, the edges of the thicker line are the 25th and 75th percentiles, the thinner lines or whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually by circles. As it can be seen, a large difference between ρ_h and ρ_l causes a larger separation between the resulting low and high usage rates. *arai* benchmark multipliers have the smallest task level separation meaning that all multipliers in this benchmark are being used almost at the same rate. In other words, the multiple candidates act similarly. A closer look at *arai* benchmark reveals that the two multipliers in this small benchmark are used for only three cycles while both are simultaneously used for two of the three cycles, leaving a limited degree of freedom. The amount of usage level separation will directly affect the amount savings achieved by the proposed method.

Benchmark Name	#A	#M	#C	#V	P (mW)	ρ_h=0.5		ρ_h=0.8	
						P^{opt}	**I**(%)	P^{opt}	**I**(%)
lee	4	4	10	12	10.04	8.98	10.54	8.95	10.8
aria	8	2	9	16	4.08	3.81	6.7	3.79	7
pr	6	10	7	20	11.61	10.67	8.11	10.66	8.18
wang	6	10	8	20	13.29	12.15	9.27	11.97	9.92
honda	12	8	8	24	20.97	20.17	3.82	20.14	3.92
mcm	13	9	9	24	17.07	16.10	5.69	16.02	6.12
dir	11	11	9	24	24.83	23.99	3.41	23.96	3.5
aircraft	15	16	17	24	34.73	32.33	6.93	32.04	7.73

Table I. The amount savings achieved for different benchmark using the proposed method.

To demonstrate the efficacy of multiple binding candidates for power reduction, we simulated the basic ALU and multiplier units to capture static and dynamic power component variations in 45nm technology. We selected an 8-Bit ALU (c880) and 16x16 multiplier (c6288) from ISCAS-85 benchmark. The ALU has 60 inputs, 26 outputs and 383 gates while the multiplier has 32 inputs, 32 outputs and 2406 gates. The circuits were synthesized and mapped to OSU FreePDK45 standard cell library [Stine et al. 2005] by the Synopsys Design Compiler. The HSPICE netlist for each unit was extracted by using Cadence Analog Artist. The 45nm predictive technology model (PTM) was used in HSPICE simulations. The circuits were simulated at operational frequency of 250MHz and V_{DD}=1V. We performed 100 Monte Carlo simulations on each circuit by applying 1000 random inputs. Multiplier average dynamic and leakage power were extracted as 2.4 mW and 1.55 mW. The ALU has an average dynamic and leakage power of 235μW and 140μW respectively. We observed around 10% variation (with respect to the nominal value, i.e., $3*\sigma/\mu$) in dynamic power and 20% fluctuations in leakage power.

Next, we investigate the amount of power saving by choosing the best candidate. Functional units are assumed to go into stand-by mode by input vector control when they are not being used. In our HSPICE simulations, we find the input (from 1000 inputs) for which the leakage power is minimum and assume at stand-by mode this vector is being fed to functional units. The average leakage power of the ALUs and multipliers for the chosen input vector is 10% of the average leakage of normal operation mode. We calculate the total power consumption of each benchmark for different candidates according to Equations 2 and 3, and derive the minimum power consumption across the candidates. This step is repeated for 1000 circuit samples.

Table I shows maximum power savings achieved with minimum number of functional units compared to the case where all the functional units have nominal power consumption. The first column of Table I shows the benchmark name. The second and third columns contain the number of ALUs (**#A**) and multipliers (**#M**). Columns 4 and 5 contain the number of operation cycles (**#C**) and binding candidates (**#V**) for each benchmark. Columns 6 shows the nominal total power consumption for the benchmarks. The total power using the best candidate, and the savings in percentage are shown in the remaining columns for two design parameter values ρ_h=0.5 and ρ_h=0.8. In both cases ρ_l=0.1.

lee and *dir* benchmark circuits show the largest (10.8%) and smallest (3.5%) percentage of savings in Table I. Notice that the savings are smaller for ρ_h=0.5.

Technology	Leakage/dynamic power ratio	Leakage power variation (3σ)	Dynamic power variation (3σ)
45nm	40% / 60%	20%	10%
32nm	50% / 50%	30%	15%
25nm	60% / 40%	40%	20%

Table II. The predicted amount of variations in dynamic and leakage power.

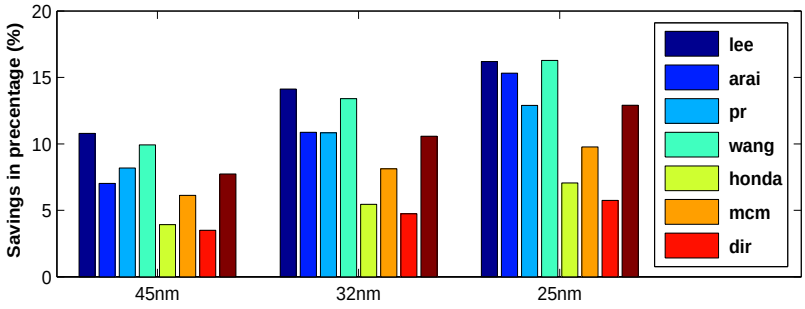


Fig. 10. The predicted amount of savings for 45nm, 32nm and 25nm technology nodes.

The decrease in power savings can be explained by the smaller low and high usage level separation as indicated in Figure 9. The same explanation can be directly applied to benchmark *arai*. The limited degree of freedom in *arai* benchmark which causes the small usage level separation in Figure 9, is responsible for the decrease in the overall power savings for this benchmark.

The amount of power savings primary depends on the following factors: *(i)* The degree of freedom in the scheduled operations and the average usage of functional units in all cycles. For example, if all the units are used in all cycles in a circuit then obviously all the candidates would behave the same way. *(ii)* The number of candidates. The larger the number of candidates, the chance of reaching the optimal scenario would be higher. However, increasing the number of candidates would impose hardware and power overhead on the controller that would eventually overshadow the power savings by post-silicon tuning. *(iii)* The amount of present variation for leakage and dynamic power.

Since the amount of leakage and dynamic power and their variations are constantly increasing in the state-of-the-art and future technologies, we performed a predictive analysis to determine how much savings the proposed method can achieve as the device sized keep shrinking. Table II shows the predicted amount of dynamic to leakage power ratios along with the amount of variations in each technology according to International Technology Roadmap for Semiconductors (ITRS) [rep 2004]. Using the values from Table II, we predicted the amount of total savings in each technology node as shown in Figure 10. The results suggest that the savings can reach above 15% for some benchmarks.

To study the effect of using multiple binding candidates on the hardware area overhead, we used ABC synthesis tool to estimate the area overhead of a single

binding and the multi-candidate binding. The area overhead is shown on Table III. The first column shows the benchmark name. The second, third and fourth columns illustrate the number of ALUs, multipliers and candidates for each benchmark. Column 5 and 6 show the area overhead for the chip using a fixed single binding denoted by *orig* and the area for the new method denoted by *new*. Finally, the last column represents the total area overhead of the new method. The larger benchmarks naturally incur a higher overhead. As the overhead results in Table III suggest, a large number of binding candidates can be simultaneously realized with a relatively low overhead on the controller. Note that the size and power consumption of the controller is significantly smaller than the functional units and the data path [Hennessy and Patterson 1996]. Therefore, the small overhead on the controller has a negligible effect on the total power consumption. In other words, even large overheads in terms of size and power consumption in the controller will be still negligible compared to the savings achieved by the proposed method. This is particularly true since the more complicated circuits in Table III have proportionally larger data paths and number of functional units.

Table III. Area overhead of integrating the binding candidates in Table I in the controller.

Name	A	M	V	orig(lit)	new(lit)	%
lee	4	4	12	83369	83969	0.7
arai	8	2	16	99738	109420	0.3
pr	6	10	20	163421	171592	5
wang	6	10	20	153823	167755	2.7
honda	12	8	24	211627	217537	2.8
mcm	13	9	24	232476	238533	2.6
dir	11	11	24	229201	237967	3.8
aircraft	15	16	24	322173	356728	10.7
chem	20	10	24	320727	337269	5.2

Finally, it is worthwhile to notice that it is extremely difficult to quantify the accuracy of the power saving estimation without comparing the results to power measurements taken off an actual manufactured chip. However, there are certain factors and approximations that could slightly affect the accuracy of our estimation: (1) The spatial correlation in variations of process parameters can theoretically introduce correlations among the power consumption of functional units, which can in turn lower the power savings achieved by the multiple binding candidate method. For example, suppose that variations are highly correlated, such that the power consumption of all functional units on the same chip are similar (100% correlated), while their power consumption across different chips are different. In this extreme case, the savings would drop to zero (all functional units would have similar performance). However, in practice this is not the case. The spatial correlation dies out across a few gates. Work on modeling spatial correlations using grid-based approaches [Agarwal et al. 2003; Chang and Sapatnekar 2007], treat the gates inside the same grids as 100% correlated, cells in neighboring grids are slightly correlated, and the rest are completely uncorrelated. Typically, each grid contains less than ten gates. Since the functional units are composed of hundreds of gates, it is safe

to assume their total power consumptions are spatially uncorrelated; (2) Furthermore, we assume the power overhead of the controller is negligible; because of two reasons: first, the incurred overhead on the controller according to Table III is relatively small; second, the size of the controller is significantly smaller compared to the size of the data path and the functional units; (3) IR drops across the power rails and second order effects such as self-heating can further increase the variations in the power consumption of functional units, which can potentially further increase the savings; and (4) finally, inaccuracy of device models in SPICE simulation can slightly affect the predicted power saving.

10. CONCLUSION

We presented a new method for customizing the resource binding to the unique characteristics of each IC post-silicon. Multiple binding candidates with diversely different resource usages were constructed. The construction method utilized orthogonal arrays to diversify the task distributions across different candidates. We showed how multiple candidate can be embedded into the controller with a low overhead. The evaluation results on benchmark circuits suggest that savings in the order of 10% can be achieved for benchmarks with larger degree of freedom in their scheduled operations.

11. ACKNOWLEDGEMENT

This research is in part supported by the Office of Naval Research (ONR) YIP award under grant No. R16480 and National Scient Foundation CAREER award under grant No. R3A530. The authors would like to thank Ms. Alkabani for minor help with programming [Alkabani et al. 2009].

REFERENCES

2004. International technology roadmap for semiconductors. Tech. rep., Semiconductor Industry Association.

Abdollahi, A., Fallah, F., and Pedram, M. 2004. Leakage current reduction in CMOS VLSI circuits by input vector control. *IEEE Transactions on Very Large Scale Integration (VLSI) 12,* 2, 140–154.

Agarwal, A., Blaauw, D., Zolotov, V., Zhao, S. S. M., Gala, K., and Panda, R. 2003. Statistical delay computation considering spatial correlations. In *ASPDAC: Asia South Pacific Design Automation*. 271–276.

Alkabani, Y. and Koushanfar, F. 2008. N-variant IC design: methodology and applications. In *Design Automation Conference(DAC)*. 546–551.

Alkabani, Y., Koushanfar, F., and Potkonjak, M. 2009. N-version temperature-aware scheduling and binding. In *International symposium on Low power electronics and design (ISLPED)*. 331–334.

Alkabani, Y., Massey, T., Koushanfar, F., and Potkonjak, M. 2008. Input vector control for post-silicon leakage current minimization in the presence of manufacturing variability. In *Design Automation Conference(DAC)*. 606–609.

Bhatia, S. and Jha, N. 1998. Integration of hierarchical test generation with behavioral synthesis of controller and data path circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 6,* 4, 608–619.

Chandrakasan, A., Sheng, S., and Brodersen, R. 1992. Low power CMOS digital design. *IEEE Journal of Solid State Circuits 27*, 473–484.

Chang, H. and Sapatnekar, S. S. 2007. Prediction of leakage power under process uncertainties. *12,* 2, 12.

Hennessy, J. and Patterson, D. 1996. *Computer architecture: a quantitative approach*. Morgan Kaufmann Publishers.

Hinkelmann, K. and Kempthorne, O. 2007. *Design and Analysis of Experiments, Introduction to Experimental Design*. Wiley Series in Probability and Statistics.

Khouri, K. and Jha, N. 2002. Leakage power analysis and reduction during behavioral synthesis. *IEEE Trans. on Very Large Scale Integration (VLSI) 10,* 6, 876–885.

Koushanfar, F., Boufounos, P., and Shamsi, D. 2008. Post-silicon timing characterization by compressed sensing. In *International Conference on Computer-Aided Design (ICCAD)*. 185–189.

Kulkarni, S., Sylvester, D., and Blaauw, D. 2006. A statistical framework for post-silicon tuning through body bias clustering. In *International conference on Computer-aided design (ICCAD)*. 39–46.

Liu, Q. and Sapatnekar, S. S. 2007. Confidence scalable post-silicon statistical delay prediction under process variations. In *Design Automation Conference*. 497–502.

Mani, M., Sing, A., and Orshansky, M. 2006. Joint design-time and post-silicon minimization of parametric yield loss using adjustable robust optimization. In *International conference on Computer-aided design (ICCAD)*. 19–26.

Ndai, P., Bhunia, S., Agarwal, A., and Roy, K. 2008. Within-die variation-aware scheduling in superscalar processors for improved throughput. *IEEE Transactions on Computers 57,* 7, 940–951.

Orshansky, M., Nassif, S. R., and Boning, D. 2007. *Design for Manufacturability and Statistical Design: A Constructive Approach*. Springer.

Raghunathan, A., Jha, N., and Dey, S. 1998. *High-level power analysis and optimization*. Springer.

Rao, K. and Yi, P. 1990. *Discrete Cosine Transform*. Academic Press.

Shamsi, D., Boufounos, P., and Koushanfar, F. 2008. Noninvasive leakage power tomography of integrated circuits by compressive sensing. In *International symposium on Low power electronics and design (ISPLED)*. 341–346.

Srivastava, A., Sylvester, D., and Blaauw, D. 2005. *Statistical Analysis and Optimization for VLSI: Timing and Power*. Springer.

Stine, J. E., Grad, J., Castellanos, I., Blank, J., Dave, V., Prakash, M., Iliev, N., and Jachimiec, N. 2005. A framework for high-level synthesis of system-on-chip designs. In *International Conference on Microelectronic Systems Education*. 11–12.

DOE Toolkit 6, G. http://designcomputing.net/gendex/.

Wang, F., Wu, X., and Xie, Y. 2008. Variability-driven module selection with joint design time optimization and post-silicon tuning. In *Asia and South Pacific design automation conference (ASP-DAC)*. 2–9.