Low-Power Resource Binding by Postsilicon Customization

MEHRDAD MAJZOOBI, JOONHO KONG, and FARINAZ KOUSHANFAR, Rice University

This article proposes the first postsilicon customization method for resource binding to achieve power reduction application specific integrated circuits (ASICs) design. Instead of committing to one configuration of resource binding during synthesis, our new synthesis method produces a diverse set of candidate bindings for the design. To ensure diversity of the resource usage patterns, we introduce a binding candidate formation method based on the orthogonal arrays. Additional control components are added to enable post manufacturing selection of one of the binding candidates. The resource binding candidate that minimizes the power consumption is selected by considering the specific power characteristics of each chip. An efficient methodology for embedding several binding candidates in one design is developed. Evaluations on benchmark designs show the low overhead and the effectiveness of the proposed methods. As an example, applying our method results in an average of 14.2% (up to 24.0%) power savings on benchmark circuits for a variation model in 45nm CMOS technology. The power efficiency of our customized postsilicon binding is expected to improve with scaling of the technology and the likely resulting higher process variations.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Optimization, Automatic synthesis*

General Terms: Performance

Additional Key Words and Phrases: High level synthesis, low power, postsilicon optimization, resource binding customization

ACM Reference Format:
Majzoobi, M., Kong, J., and Koushanfar, F. 2013. Low-power resource binding by postsilicon customization. ACM Trans. Des. Autom. Electron. Syst. 18, 2, Article 26 (March 2013), 22 pages.
DOI: http://dx.doi.org/10.1145/2442087.2442097

1. INTRODUCTION

Miniaturization of CMOS to nanometer technologies and inaccuracies of the mask and lithography for the smaller size devices cause an increased uncertainty of the process parameters around their nominal values. As a result of the process characteristics variations, there are interchip and intrachip fluctuations of static and dynamic power consumption. Especially, the leakage power variability is known to have an exponential dependence on a number of operational and process parameters [Srivastava et al. 2005]. Because of the importance of power minimization, presilicon optimization techniques for lessening the power consumption at different levels of design flow have been introduced. Unfortunately, one-size-fits-all solutions that are based on optimizing the power consumption for nominal process values have a limited effectiveness in presence of variations across the chips.

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.
Authors' address: Rice University, 6100 Main St. MS-380, Houston, TX 77005; email: mehrdad.majzoobi@rice.edu, jk18@rice.edu, farinaz@rice.edu.

DOI: http://dx.doi.org/10.1145/2442087.2442097

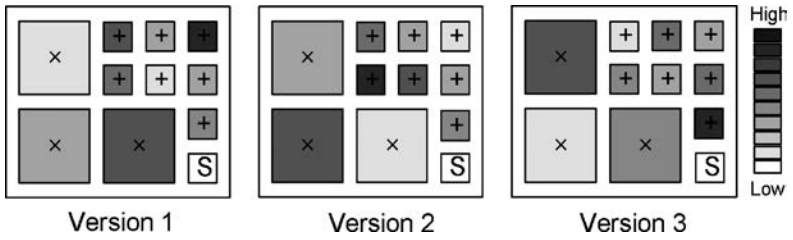


Fig. 1. The same benchmark with three candidates for task distribution.

Adaptive postsilicon optimization has been shown to be effective by tuning design parameters to the specific characteristics of each IC after fabrication. A standing challenge is applying postsilicon 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 postsilicon, including adaptive body biasing and standby input vector control [Alkabani et al. 2008]. Wang et al. [2008] proposed presilicon module selection that is optimized to be variation-aware with postsilicon 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 [Raghunathan et al. 1998] at a higher level. 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 the design, are different from (i) instruction scheduling in heavily pipelined circuit where the instruction priorities may change [Ndai et al. 2008], (ii) schedule selection in reconfigurable environments, or (iii) application-level scheduling of multiprocessors performed in software [Wang et al. 2008].

In this article, we introduce a high-level synthesis methodology that during the design phase generates multiple resource binding candidates instead of one. Each candidate binding assigns different usage levels to the functional units. For example, Figure 1 shows three resource allocation and 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 option, 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 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. Note that other methods for embedding multiple options during high-level synthesis have been applied for security and temperature balancing purposes [Alkabani and Koushanfar 2008; Alkabani et al. 2009]. The multiple binding approach introduced in this article is new and orthogonal to the earlier introduced methods. The contributions of this article are as follows:

—Introduction of the first multiple candidate resource allocation and binding framework for ASICs that allows postsilicon variation-aware selection of the best candidate;
—Efficient creation of a diverse set of resource binding possibilities using orthogonal arrays;
—Formation and evaluation of heuristic algorithms for embedding multiple diverse resource allocation and 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;
—Creation of new methods for low overhead postsilicon 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 postsilicon candidates and power savings of the resulting variation-aware optimization compared to the best available low-power presilicon binding methods.

The remainder of the article 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. Postsilicon selection of the best candidate is introduced in Section 8. The evaluation results are presented in Section 9. Finally, Section 10 concludes the article.

2. BACKGROUND AND RELATED WORK

High-Level Synthesis. Automated high-level synthesis (HLS) methods and tools are necessary for coping with complexity and scale of contemporary large designs. The algorithmic and functional behavioral description of the design are inputs to the HLS tools. High-level descriptions are automatically synthesized and compiled by the HLS tools to lower level hardware components and primitives with a minimal involvement and input from the designer. Several high-level hardware description languages (HDL) were created based on higher level functional programming languages such as C/C++. Examples of HLS tools include Xilinx AutoESL[1], Mentor Graphics CatapultC[2], SystemC[3], ImpulseC[4], Bluespec[5], and Handel-C[6].

[1]Xilinx. Autoesl. http://www.xilinx.com/tools/autoesl.htm.
[2]CatapultC. Mentor graphics. http://www.mentor.com/esl/catapult/overview.
[3]SystemC. Open system initiative. http://www.systemc.org.
[4]ImpulseC. Impulse accelerated technologies. http://www.impulseaccelerated.com.
[5]Bluespec. http://www.bluespec.com.
[6]Handel-C. Mentor graphics. http://www.mentor.com/products/fpga/handel-c/.

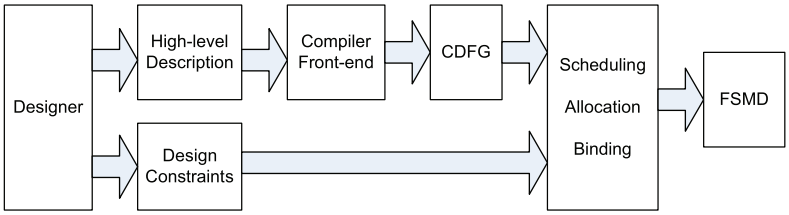


Fig. 2. General flow for high-level synthesis.

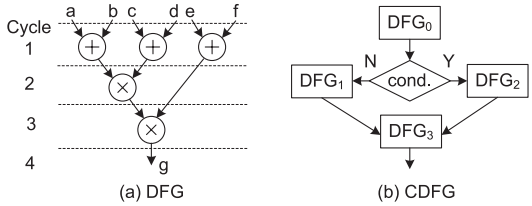


Fig. 3. An example of (a) dataflow graph (DFG) and (b) control flow graph (CDFG).

Conversion of HDL to the hardware level is done by (i) compiling and optimizing the code for hardware implementation, and (ii) scheduling, resource allocation, and binding. Figure 2 shows a generic flow for HLS. First, the design is described using a high level HDL. This high-level description is then transformed to the control data flow graph (CDFG) which would then go through initial compiler optimizations to prepare for hardware implementation. For example, loops in the high-level HDL code are unrolled, or the dead codes are removed. The designer's additional constraints and the CDFG are input to subsequent phases of HLS including scheduling, resource allocation, and binding. The resulting hardware implementation typically constitutes a finite state machine (FSM) and a datapath. In what follows, we briefly describe CDFG generation, scheduling, allocation, and binding phases.

A high-level description code can be divided into basic code blocks connected by control modules which correspond to branching and looping. These basic blocks can be represented as dataflow graphs (DFG). Figure 3(a) shows a sample DFG. A DFG can be shown by 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 data dependencies between the adjacent nodes. In particular, input edges represent operands to operations, and 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.

Figure 3(b) shows a sample CDFG. The CDFG is defined as a graph with DFG subgraphs. A CDFG shows how the DFGs are connected in code. 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 may 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 hardware implementation. Different resources for the same operation may incur a different area, power consumption, or delay. Binding is the process of mapping different data and operations to the functional and memory units as well as specifying the interconnects. The three design stages (i.e., scheduling, allocation, and binding) can sometimes be

combined or interleaved. The terms *resource* and *functional unit* are used interchangeably in the remainder of the article.

Though several techniques [Chon and Kim 2009; Jung and Kim 2009; Lucas et al. 2009; Lucas and Chen 2010] have been proposed for variation-aware HLS, these techniques mainly focus on the performance/timing optimization. In this work, we propose a variation-aware HLS technique focusing on low power.

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 may be 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 a sum of interdie and intradie variation. Interdie variation refers to differences among the dies and is constant within one die. Intradie variation, on the other hand, accounts for the differences among devices on the same die. The intradie 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 interdie variation component, and $\Delta\Phi_{spatial}(x_i, y_i)$ and $\Delta\Phi_{random}(i)$ are the spatially correlated (systematic) and random components of intradie 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 variation 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. Two methods are typically used for saving the dynamic power: 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.

A common 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 predefined 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 scheduling 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 article.

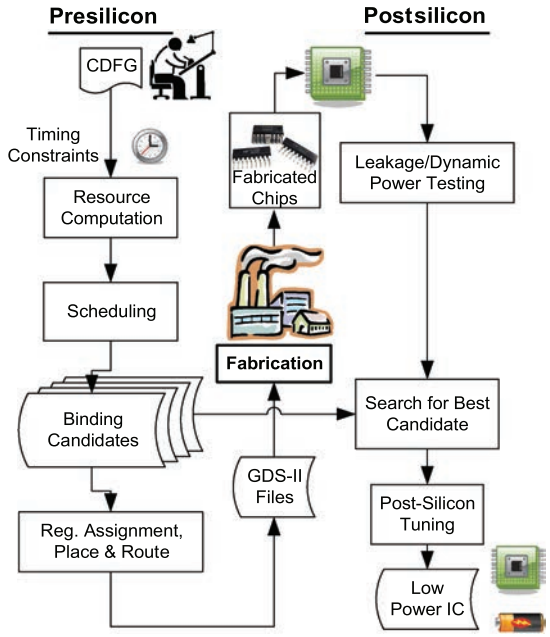


Fig. 4. The flow of the proposed approach.

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] and 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 prespecified 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.

During the postsilicon stage, chips are noninvasively tested postsilicon and the power profile of each chip is found for each binding candidate. Next, power profile of each 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. Note that postsilicon 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.

Note that in our algorithm, the quantification of the process variation is done postsilicon and not in binding candidate generation process. Although one can devise presilicon multiple binding methods which consider process variations, such methods would naturally be conservative since they will be optimized for the statistically average case.

Such presilicon methods will likely incur a high overhead in terms of power and performance since they are not tuned to individual chip characteristics. This overhead would likely be much larger as process technology evolves (due to larger fluctuations). Thus, we opted to do the actual binding process of functional units during the postsilicon phase. Our algorithm works by characterizing the actual power fluctuations of each chip.

4. PROCESS VARIATION AND USAGE LEVEL ASSIGNMENT

A circuit typically consists of functional units and controller that schedules the operations performed by the functional units. 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 power consumption of each functional unit can be formally expressed as:

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

where P_{total}^{fu} is the total power consumption of a given functional unit; P_{leak}^{fu} and P_{dyn}^{fu} 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 used) and 1 (always 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 idle and 1 implies there is no leakage saving mechanism. The total chip power consumption can then be written as:

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

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 variation, 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 diversely different usage profiles before fabrication. Our method chooses the one that achieves the lowest overall power consumption. Our new selection phase requires testing a set of candidates and then 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 postsilicon 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 complexity of the approach, eventually defeating the savings of postsilicon tuning. Second, the CDFG constraints allow us to only heuristically assign the functional unit usages. Third, in resource constrained cases, all the functional units may always be needed to meet the application demand, leaving very

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.

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. In general, an orthogonal array is a matrix which contains "factor" and "run" information. The factors and runs correspond to the matrix columns and rows, respectively [Hinkelmann and Kempthorne 2007]. In our algorithm, a factor represents the functional unit(s) and run represents the number of binding candidates. 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, 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. Therefore, 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 our 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. First, using the CDFG of the target circuit, one can obtain a schedule of the circuit by using a conventional HLS method (tool) such as AutoESL [Xilinx] and CatapultC [CatapultC]. The inputs to our algorithm are the schedule of the circuit generated by the conventional HLS methods and target task distribution (TD) among the 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, that is, low and high usage).

With the generated schedule, we would have (i) the total number of cycles Ncycles, and (ii) the number of operations to be performed by each type of functional units at each cycle, that is, *Scheduled[type, cycle]*. As an example, if $Scheduled[mult,5] = 3$, then it means three multiplications must be performed in the fifth cycle.

Now, by applying our binding algorithm, Algorithm 1, on the matrix *Scheduled[type, cycle]* the operations are assigned 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 often 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 ρ_h to it; otherwise, it would give the unit a low accepting probability value ρ_l.

In practice, proper ρ_h and ρ_l can be set by detailed simulations. For example, one can model the process variation using parametric values (e.g., mean and standard deviation of Gaussian distribution) and perform Monte Carlo simulations to get optimal ρ_h and ρ_l values for power saving. Such parametric distributions do not need the exact characteristics of the target technology node and can be done across a few values which is fine with the high-level synthesis flow. By performing the simulations, one can obtain the required statistical distribution of power saving results according to each ρ_h and ρ_l value. This way, one would be able to optimize the ρ_h and ρ_l values for the pertinent circuit.

After the assignment of the accepting probability (either ρ_h or ρ_l) to each functional unit, 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 and this functional unit meets the predefined timing constraint, it would not be visited again in that cycle. To consider the timing issue in our algorithm, we set a bound on the target timing closure, which can be statistically determined during the design-time. In the binding process of our algorithm, only the functional units that can meet the timing bound are selected and bound.

Lines 3 and 4 of Algorithm 1 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 randomly picks a functional unit that meets the timing constraint 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 9) 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 process is repeated in the next cycles and unit types (Lines 1 and 2). Note that the functional units with a higher level of willingness to accept the operation are likely to end up with having more tasks assigned to them.

So far, we described the algorithm that binds the operations to functional units for achieving a prespecified 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. Algorithm 2 shows how different binding candidates are created for a given schedule. Algorithm 2 calls Algorithm 1 to perform binding for the expected task distributions

ALGORITHM 1: 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                 && the functional unit i meets the bound on timing closure)
9                      Bound{type, cycle} ← i;
10                     Remove i from Free;
11                     Scheduled[type, cycle] = Scheduled[type, cycle] − 1;
12      end
13 end
```

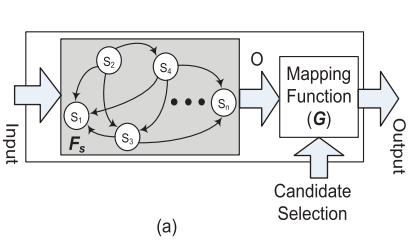
ALGORITHM 2: Generation of multiple binding candidates

```
1  Schedule CDFG using conventional HLS methods;
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) = ρ_h;
8                 TD(f) = 1;
9            if OA(v, f) = low
10                Accepting Prob(f) = ρ_l;
11                TD(f) = 0;
12      end
13      Bind the operations using Algorithm 1;
14 end
```

dictated by an orthogonal array. Line 1 schedules the operations and determines how many operations of each type are required at each cycle. As we explained above, this step can be done by conventional HLS scheduling methods. Line 2 sets the number of design factors N_{factor}, to the total number of functional units, that is, 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 as we explained in Section 5. Low and high accepting probabilities are assigned based on the factor levels dictated by orthogonal array values in Lines 4, 5, 6, and 7. The operations are assigned to functional units according to the assigned probabilities using Algorithm 1. 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



Truth Table

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

(b)

Fig. 6. (a) Integration of multiple binding candidates into the controller. (b) Example mapping truth table.

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 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 show 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 on 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 operations. Therefore, our method works best for datapath-intensive designs.

To support the maximum flexibility of functional unit mapping, 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 implementation details are described in Figure 7. Additional MUXs and wires are needed to connect the registers (flip-flops) to the functional units (ALUs and multipliers). Since it is not known which binding candidate is the lowest power consuming candidate in the presilicon stage, one should make a connection between every register and every functional unit. To support this, MUXs can forward the data from the register to any functional unit. The selection signals generated from the controller designate which functional units are used for each operation. In the controller's mapping function, the selection signals for MUXs and enable signals are generated and forwarded to each MUX and functional unit by referring to the controller's truth table. The total cost of the mapping function is linearly increased as the number of binding candidates increases. However, the hardware cost per one candidate is only a column in the lookup truth table, which is not significant. Moreover, our

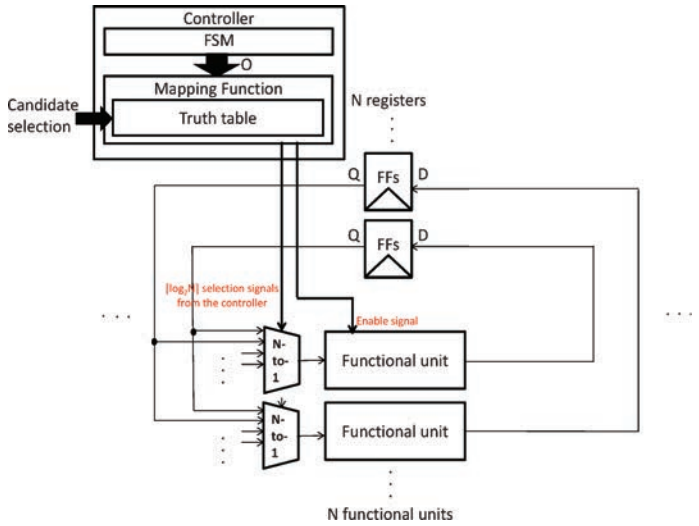


Fig. 7. Additional MUXs and wires to realize our postsilicon tuning (for flexible forwarding of data from FFs) and their interaction with the controller.

method typically suffices a small number of binding candidates (as shown in Table III). Thus, the controller overhead is negligible.

8. POSTSILICON 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 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 postsilicon 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 $i = 1, \ldots, N_{fu}$) can be found by solving a linear programming optimization that minimizes l_2 norm of the measurement error e_m (for measurement m), that is, $min \ \sqrt{\sum_{m \in S} e_m^2}$, 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:

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

Table I. The Description of Benchmark Circuits Used for Our Evaluation

Name	#A	#M	#V	Description
honda	12	8	20	A mechanical controller
aircraft	15	16	24	A mechanical controller
rsa	8	2	16	Public-key cryptographic algorithm
sha1	12	0	16	Cryptographic hash function
adpcm	11	0	16	Adaptive differential pulse-code modulation
g721	18	3	20	G.721 voice compression algorithm
reedsolomon	10	3	16	Reed Solomon encoding function
dft	8	10	20	Discrete Fourier Transform function
dct	4	4	12	Discrete Cosine Transform function
fir	16	8	20	Finite impulse response filter
iir	8	12	20	Infinite impulse response filter
kalman	16	16	24	Kalman filter

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. As we already explained in Section 6, during design and scheduling, we construct our candidates such that their delays (including the interconnect) statistically satisfy the desired circuit timing closure by setting a bound on the timing closure in our binding algorithm. Postsilicon 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.

9. EXPERIMENTAL RESULTS

We evaluated the proposed multiple binding method on various benchmarks. The first column of Table I shows the benchmark name. *honda* and $aircraft$ are mechanical controllers of industrial strength. *rsa* and *sha*1 perform cryptographic functions. *adpcm* and *g*721 are from Mediabench [Lee et al. 1997], which is designed for evaluate media processing operations. The others are Reed Solomon encoder (*reedsolomon*), discrete Fourier transformer (dft), discrete cosine transformer (*dct*), FIR filter (fir), IIR filter (iir), and Kalman filter (*kalman*). These benchmark circuits are representative examples of ASIC designs. The second, third, and fourth columns contain the number of ALUs (**#A**), multipliers (**#M**), and binding candidates (**#V**), respectively. In the fifth column, there is a short description of these benchmark circuits.

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 8 shows the functional unit usage rates for different binding candidates of $aircraft$ benchmark for both multipliers and ALUs. The two lower matrices are the 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 more (less) frequently

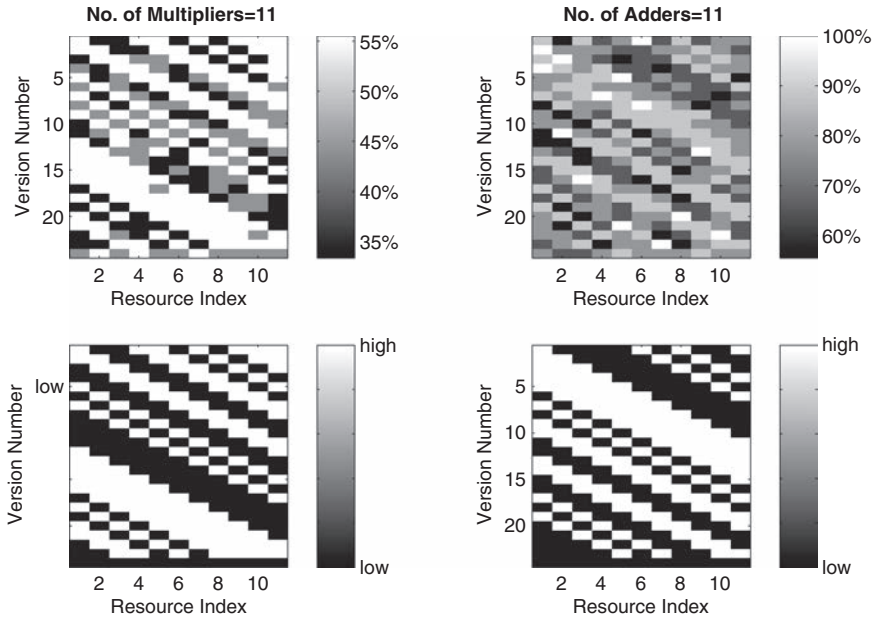


Fig. 8. The plots of expected (bottom) and achieved (top) task distributions for different candidates of *aircraft* benchmark for the multiplier (left) and ALU (right) modules.

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 Algorithm 2 are set at $\rho_h = 0.8$ and $\rho_l = 0.1$.

Figure 9 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.

As shown in Figure 9, our method makes this separation well. Between the FUs that have high and low usage levels, we can see a clear difference of actual usage rates. Consider for example, the case of *honda* in Figure 9. In this case, and most other cases, the functional units (FUs) with high usage level are likely used while FUs with low usage level are rarely used. Through this usage level separation, we can get more power saving by using the low-power consuming FUs (which have a high usage level) far more frequently than the high-power consuming FUs (which have a low usage

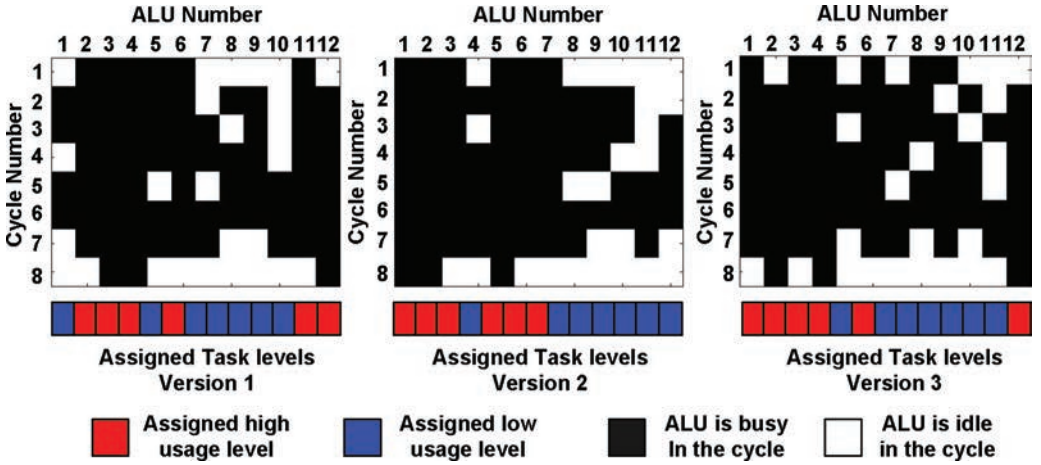


Fig. 9. Three binding candidates for the *honda* benchmark each satisfying a preimposed usage level pattern.

level). Figure 10 shows for each benchmark and functional unit type, the distribution of low and high usage levels for different versions, that is,

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

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 $\rho_h = 0.5$ and $\rho_h = 0.8$ while $\rho_l = 0.1$ as defined in Algorithm 2. 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. In addition, in most benchmark instances, a clear usage level separation is shown, which implies a high potential for power reduction by using our 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 16×16 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 $V_{DD} = 1$V. To obtain the average power consumption of the functional units, we applied 1000 random inputs to the multiplier and ALU circuits. 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, that is, $3 \times \sigma/\mu$) in dynamic power and about 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

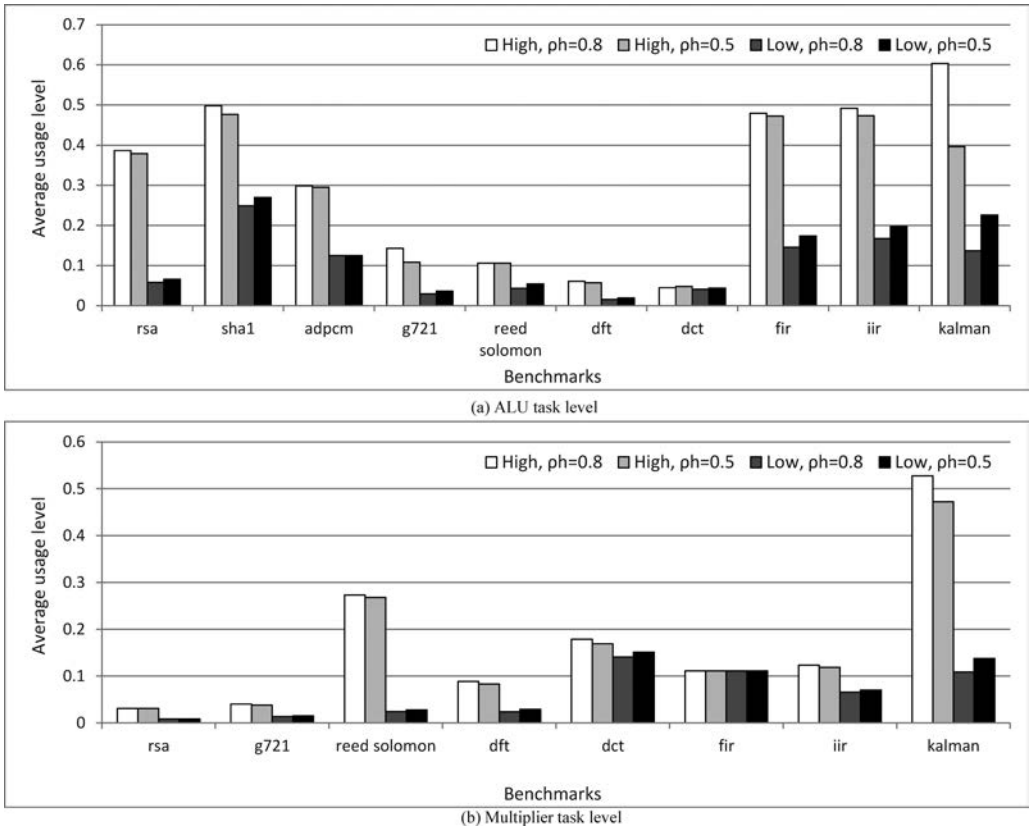


Fig. 10. The low and high usage levels across different versions for ALU and multipliers with $\rho_h = 0.5$ and $\rho_h = 0.8$.

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. In our Monte Carlo simulations, this step is repeated for 1000 circuit samples.

As we explained in Section 7, our method incurs an additional overhead due to MUXs and wires for connecting the registers (flip-flops) to every functional unit (ALUs and multipliers). These logics also consume power, which may reduce the power efficiency of our method. In our evaluation, we also reflect the impact of power overhead of these additional logics.

Figure 11 shows normalized power consumption results compared to the single binding case (1.0 in the results) with two design parameter values $\rho_h = 0.5$ and $\rho_h = 0.8$. In both cases, ρ_l is set to 0.1. Overall, our method reduces power consumption by 14.8% (on average) in case of $\rho_h = 0.8$, compared to the single binding case. In case of *sha*1 and *adpcm*, our method consumes additional power compared to the single binding case. The reason is that these circuits have only ALUs. Since an ALU circuit consumes much less power than a multiplier circuit, there is also smaller power variation (due to process variation) in case of ALUs. In this case, the difference of power consumption between the power hungry and power efficient resources would be small. It implies there remains only small room for power saving using our method. Thus, if the power

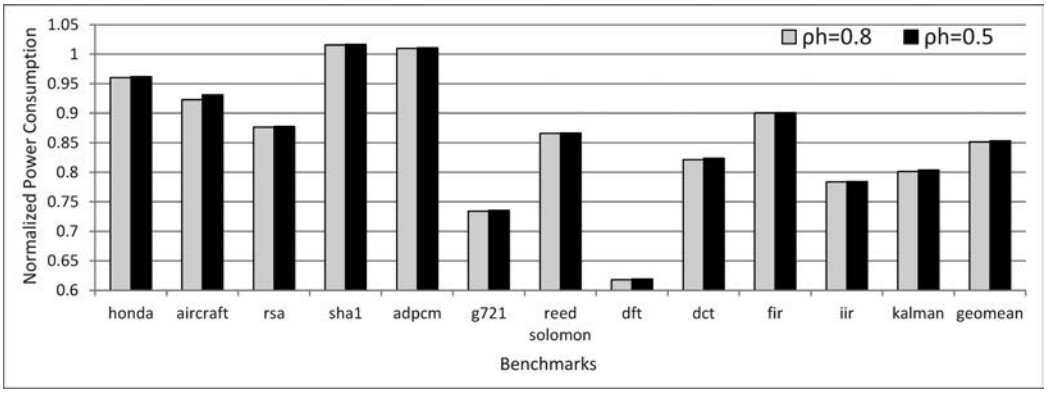


Fig. 11. Normalized power consumption results across various process variation severities compared to the single binding case.

overhead incurred by our method (additional MUXs, wires, and controller overhead) is larger than the power saving of our method, it becomes inefficient.

For most practical cases, including the other 10 benchmark circuits, our method shows significant power savings. In case of dft, it shows 38.2% of power saving, which is significant. This huge power saving comes from the input vector control of unused functional units, which can save up to 90% of leakage power consumed by unused functional units. In case of dft, we have a low functional unit usage rate (i.e., there are many idle functional units), compared to the other benchmark circuits. Thus, it has much more room to save leakage power consumption via the input vector control. In case of $\rho_h = 0.5$, we can reduce power consumption almost as much as in case of $\rho_h = 0.8$, though power saving is slightly smaller than in case of $\rho_h = 0.8$. The decrease in power saving can be explained by the smaller low and high usage level separation due to the smaller difference between ρ_h and ρ_l.

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 postsilicon tuning. *(iii)* The amount of present variation for leakage and dynamic power.

Since the amount of leakage and dynamic power variations is continually increasing in the state-of-the-art and future technologies, we performed our simulations across four different process variation scenarios: low, medium, high, and extreme. Table II presents the variation severity of each scenario. The low, medium, and high variation scenarios are realistic while the extreme scenario is to show the effectiveness of our method when there is abundant process variation.

As shown in Figure 12, compared to the single binding case, in case of high process variation scenario where $3 \times \sigma/\mu$ of dynamic power=20% and leakage power=40%, our method reduces power consumption by an average of 17.1% (2.3% more reduction compared to the low variation scenario where $3 \times \sigma/\mu$ of dynamic power=10% and leakage power=20%). In the extreme scenario where $3 \times \sigma/\mu$ of dynamic power=50% and leakage power=100%, power saving is up to 19.2%, which means 4.4% more saving compared to the case where $3 \times \sigma/\mu$ of dynamic power=10% and leakage power=20%

Table II. The Amount of Variations in Dynamic and Leakage Power Across Four Process Variation Scenarios

Process variation severity	Leakage power variation (3σ)	Dynamic power variation (3σ)
Low (low)	20%	10%
Medium (med)	30%	15%
High (hig)	40%	20%
Extreme (ext)	100%	50%

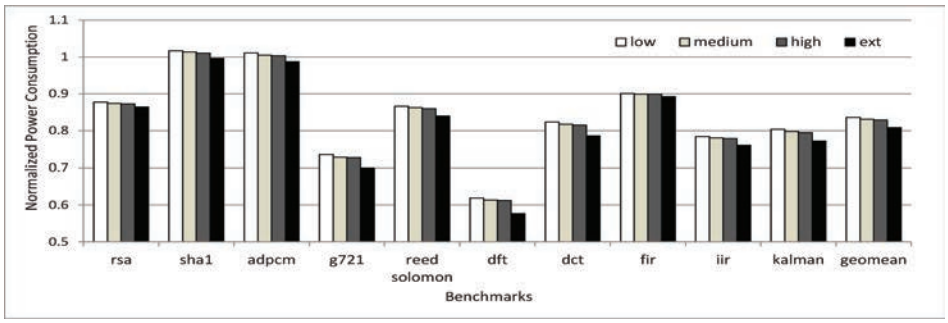


Fig. 12. Normalized power consumption results across various process variation severities compared to the single binding case.

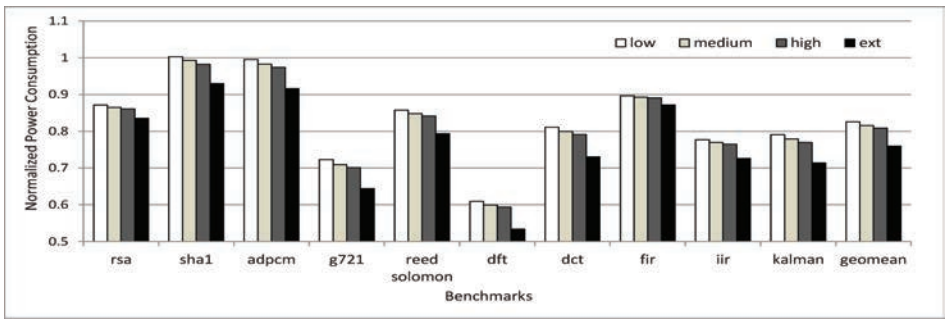


Fig. 13. Normalized power consumption results across various process variation severities compared to the worst-case binding case.

(i.e., low variation scenario). We also compare our method with the worst-case binding scenario. The worst-case binding scenario means the power consumption of the most power-consuming chip among 1000 chip instances when using the single binding method, while the single binding case means that of the average power consumption across the 1000 chip instances. Compared to the worst-case binding (Figure 13), our method reduces power consumption by an average of 18.4% and 19.1%, in cases of the medium (where $3 \times \sigma/\mu$ of dynamic power=15% and leakage power=30%) and high (where $3 \times \sigma/\mu$ of dynamic power=20% and leakage power=40%) process variation scenario, respectively. In the process variation scenario where $3 \times \sigma/\mu$ of dynamic power=50% and leakage power=100% (i.e., the extreme scenario), power saving on our benchmark circuits is up to an average of 24.0%. As process technology scales to lower dimensions, much more severe process variation is expected. Therefore, we expect much more power savings using our proposed method for the future process technologies.

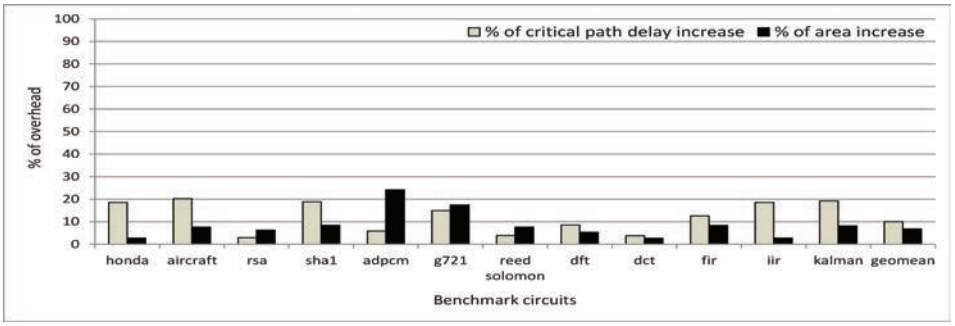


Fig. 14. Timing and area overhead caused by the MUXs and interconnect.

As we explained in Section 7, our method requires implementing some additional logics to realize the postsilicon binding of manufactured chips. To set appropriate timing bounds, we also carefully examined how much the delay/area really increases. To quantify this, we generated Verilog codes that implement these MUXs and interconnect in our benchmark circuits. We synthesized these codes with Synopsys Design Compiler with 45nm Nangate Opencell library[7] and performed place & route with Cadence SoC Encounter.

Figure 14 shows the timing and area overhead incurred by additional MUXs and interconnects. Timing overhead incurred by our method is under 10%, on average. The bounds we set on timing closure (10% margin in our experiments) take care of controlling this timing overhead. We also give an enough timing margin to the critical path (10%) to consider the corner-case of process variations. Finally, our timing closure is set to be 80% of the critical path delay. If tighter time bounds are required, one may need to tradeoff area and timing which can be set by iteratively synthesizing for a multiple binding approach in our binding algorithm. Through postsilicon timing characterization, our method picks up the candidate that meets the predefined timing constraints which also consumes the lowest power. In terms of area overhead, we need only 6.8% of additional area compared to the case when there is no postsilicon binding support. In case of *adpcm* and *g*721, it shows a little higher area overhead compared to the other benchmark circuits. Since these circuits do not have multipliers, which occupy much larger area than ALUs, area overhead incurred by our method may be seen relatively higher than the other circuits. However, in case of the other circuits, the area overhead of our method is small.

We also present the controller area overhead. To quantify the effect of using multiple binding candidates on the controller 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 in 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, the area overhead of the controller is shown to be less than 3.3% (on average), which is significantly low. By using our method, 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

[7]Nangate. 45nm nangate open cell library v1.3.

Table III. Area Overhead of Integrating the Binding Candidates in Table I in the Controller

Name	A	M	V	Literals (orig)	Literals (new)	Overhead
honda	12	8	20	211634	217545	2.8
aircraft	15	16	24	322201	356811	10.7
rsa	8	2	16	66639	68709	3.1
sha1	12	0	16	50202	52685	4.9
adpcm	11	0	16	46206	48484	4.9
g721	18	3	20	124695	130202	4.4
reedsolomon	10	3	16	91276	93968	2.9
dft	8	10	20	212787	217506	2.2
dct	4	4	12	90221	91435	1.3
fir	16	8	20	197242	203534	3.2
iir	8	12	20	231939	237183	2.3
kalman	16	16	24	331508	341664	3.1

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. In case of area-constrained environment, we can reduce the controller area overhead by reducing the number of binding candidates, though power consumption of the chip may be slightly increased. By exploiting an efficient trade-off (by adjusting the number of binding candidates) between power and area, one can find the optimal point between power and area of the chip.

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;

(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 in the postsilicon stage. 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 various benchmark circuits show that our method reduces power consumption by 14.2% in the low variation scenario and up to 24.0% in the extreme variation scenario.

REFERENCES

ABDOLLAHI, A., FALLAH, F., AND PEDRAM, M. 2004. Leakage current reduction in CMOS VLSI circuits by input vector control. *IEEE Trans.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 *Proceedings of the Asia South Pacific Design Automation Conference*. 271–276.

ALKABANI, Y. AND KOUSHANFAR, F. 2008. N-variant IC design: methodology and applications. In *Proceedings of the Design Automation Conference*. 546–551.

ALKABANI, Y., KOUSHANFAR, F., AND POTKONJAK, M. 2009. N-version temperature-aware scheduling and binding. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 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 *Proceedings of the Design Automation Conference*. 606–609.

CHANDRAKASAN, A., SHENG, S., AND BRODERSEN, R. 1992. Low power CMOS digital design. *IEEE J. Solid State Circuits 27*, 473–484.

CHANG, H. AND SAPATNEKAR, S. S. 2007. Prediction of leakage power under process uncertainties. *ACM Trans. Des. Autom. Electron. Syst. 12*, 2, 12.

CHON, H. AND KIM, T. 2009. Timing variation-aware task scheduling and binding for mpsoc. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 137–142.

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.

JUNG, J. AND KIM, T. 2009. Timing variation-aware high-level synthesis considering accurate yield computation. In *Proceedings of the IEEE International Conference on Computer Design*. 207–212.

KHOURI, K. AND JHA, N. 2002. Leakage power analysis and reduction during behavioral synthesis. *IEEE Trans. VLSI Syst. 10*, 6, 876–885.

KOUSHANFAR, F., BOUFOUNOS, P., AND SHAMSI, D. 2008. Post-silicon timing characterization by compressed sensing. In *Proceedings of the International Conference on Computer-Aided Design*. 185–189.

KULKARNI, S., SYLVESTER, D., AND BLAAUW, D. 2006. A statistical framework for post-silicon tuning through body bias clustering. In *Proceedings of the International Conference on Computer-Aided Design*. 39–46.

LEE, C., POTKONJAK, M., AND MANGIONE-SMITH, W. H. 1997. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture*. 330–335.

LIU, Q. AND SAPATNEKAR, S. S. 2007. Confidence scalable post-silicon statistical delay prediction under process variations. In *Proceedings of the Design Automation Conference*. 497–502.

LUCAS, G. AND CHEN, D. 2010. Variation-aware layout-driven scheduling for performance yield optimization. In *Proceedings of the International Conference on Computer-Aided Design*. 17–24.

LUCAS, G., CROMAR, S., AND CHEN, D. 2009. Fastyield: variation-aware, layout-driven simultaneous binding and module selection for performance yield optimization. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 61–66.

MANI, M., SING, A., AND ORSHANSKY, M. 2006. Joint design-time and post-silicon minimization of parametric yield loss using adjustable robust optimization. In *Proceedings of the International Conference on Computer-Aided Design*. 19–26.

NDAI, P., BHUNIA, S., AGARWAL, A., AND ROY, K. 2008. Within-die variation-aware scheduling in superscalar processors for improved throughput. *IEEE Trans. Comput. 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.

SHAMSI, D., BOUFOUNOS, P., AND KOUSHANFAR, F. 2008. Noninvasive leakage power tomography of integrated circuits by compressive sensing. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 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 *Proceedings of the International Conference on Microelectronic Systems Education*. 11–12.

WANG, F., WU, X., AND XIE, Y. 2008. Variability-driven module selection with joint design time optimization and post-silicon tuning. In *Proceedings of the Asia and South Pacific Design Automation Conferenc*e. 2–9.

Received December 2011; revised September 2012; accepted October 2012