FPGA-Targeted High-Level Binding Algorithm for Power and Area Reduction with Glitch-Estimation*

Scott Cromar, Jaeho Lee, Deming Chen
Department of Electrical and Computer Engineering
University of Illinois, Urbana-Champaign

ABSTRACT

Glitches (i.e. spurious signal transitions) are major sources of dynamic power consumption in modern FPGAs. In this paper, we present an FPGA-targeted, glitch-aware, high-level binding algorithm for power and area reduction, accomplished via dynamic power estimation and multiplexer balancing. Our binding algorithm employs a glitch-aware dynamic power estimation technique derived from the FPGA technology mapper in [6]. High-level binding results are converted to VHDL, and synthesized with Altera's Quartus II software, targeting the Cyclone II FPGA architecture. Power characteristics are evaluated with the Altera PowerPlay Power Analyzer. The binding results of our algorithm are compared to LOPASS, a state-of-the-art low-power high-level synthesis algorithm for FPGAs. Experimental results show that our algorithm, on average, reduces toggle rate by 22% and area by 9%, resulting in a decrease in dynamic power consumption of 19%. To the best of our knowledge this is the first high-level binding algorithm targeting FPGAs that considers glitch power.

Categories and Subject Descriptors

B.6.3 [**Hardware**]: Design Aids—*optimization*

General Terms

Algorithms, Design, Measurement, Performance

Keywords

FPGA, high-level synthesis, glitch power, power reduction

1. INTRODUCTION

FPGAs hold significant promise as a fast-to-market replacement for ASICs in many applications. As the price of single-purpose chip development skyrockets in each successive technology iteration, the relative price of the FPGA architecture becomes more and more attractive. This, coupled with the many other advantages of FPGAs, such as rapid prototyping and field reprogrammability, makes them a more and more viable alternative in many current ASIC applications. Unfortunately, the advantages of FPGAs are offset in many cases by high power consumption and large area. In fact, it has been shown that FPGAs can be up to 40 times larger and consume up to 12 times more dynamic power than the equivalent ASIC implementation [13].

In FPGAs there are two sources of power consumption: *static power* and *dynamic power*. Static power is power consumed when the circuit is either active or idle. Unless power gating or other transistor-level techniques are built in the chip, static power cannot be easily reduced. Dynamic power, on the other hand, is power consumed when a signal transition occurs at gate outputs, and, being a characteristic of the design implemented on the FPGA, is more easily mitigated. Signal transitions make up the switching activity (SA) of a circuit, and can be classified into two types: *functional transitions*, and *glitches*. Functional transitions are the signal transitions necessary to perform the required logic function, while glitches are spurious transitions: unnecessary signal transitions that occur due to unbalanced path delays at the inputs of a gate.

Dynamic power consumption can be estimated as $P_d = 0.5 \times SA \times C \times V_{dd}^2 \times f$, where SA is the switching activity of the circuit, C is the effective capacitance, V_{dd} is the supply voltage, and f is the operating frequency. Reducing any of these factors will reduce the dynamic power of a circuit. In Altera's Stratix II FPGAs in the 90 nm process technology, dynamic power is the dominant type of power consumed. Further, glitches can account for up to 19% of the total power consumed in FPGAs, and if considering only dynamic power the percentage is even higher [16]. In this work, we focus on reducing SA (through a glitch-aware SA estimator and multiplexer balancing) and C (by indirectly reducing multiplexer area), for power minimization.

High-level synthesis—the mapping of a behavioral description of a circuit to RTL—is a well studied topic consisting of three steps: scheduling, allocation, and binding. Scheduling determines when an operation will take place; allocation determines how many of each resource is needed; and binding assigns operations, or variables, to the resources.

Our FPGA-targeted, glitch-aware, high-level binding algorithm for power and area reduction is unique in that it can connect to the gate-level implementation and employ a glitch-aware dynamic power estimation to guide the synthesis process. The dynamic power estimation is accomplished using a low-power FPGA technology mapper [6], which makes use of a switching activity estimation model considering glitches and has been shown to be effective at capturing glitch power. Reduction of switching activity, and thus dynamic power, is futher targeted via the balancing of multiplexers on the two input ports of a functional unit. The Altera Cyclone II FPGA is used as a testbed FPGA architecture in our experiments (due to its support of a large number of I/O pins that can accommodate large benchmarks), and our binding results are verified using Altera's gate-level

*This work is partially supported by NSF CCF 07-02501 and a research grant from Altera. We used machines donated by Intel. Jaeho Lee participated in the project when he was an undergraduate student in ECE at University of Illinois. Email contact: scromar2@illinois.edu or dchen@illinois.edu

DAC'09, July 26-31, 2009, San Francisco, California, USA.

power estimator, Quartus II PowerPlay Analyzer. To the best of our knowledge this is the first FPGA-targeted high-level binding algorithm that considers glitch power.

The rest of this paper is organized as follows: In Section 2 we discuss related work. In Section 3 we present the problem formulation. In Section 4 we describe the technique used for switching activity estimation considering glitches. In Section 5 we describe the binding algorithm, herein referred to as HLPower, in detail. In Section 6 we present experimental results. In Section 7 the paper is concluded.

2. RELATED WORK

Much effort has gone into evaluating and reducing dynamic power in FPGAs. Recent work has included improved SA estimation tools and architectural changes to reduce glitches [15]. Additionally, binding for ASICs is a well-studied topic. The work in [18] proved that the problem of resource binding for multiplexer reduction is NP-complete. Work in the area of low-power binding has included a bipartite graph formulation for multiplexer reduction [11], low-power register binding through a network-flow formulation [1], simultaneous register and resource binding and scheduling algorithms [9], generalized low-power binding formulated as an ILP problem with heuristic speed-ups [10], and early evaluation of DFGs for low-power binding [14], among many others. Reference [10] provides a good overview of the previous work in low-power binding.

Despite this previous work, low-power high-level synthesis and binding for FPGAs is a relatively new area of research. In [20], the switching activity characteristics of the functional units were pre-characterized and used during low-power synthesis targeting FPGAs. In [5], a low-power, simultaneous resource allocation and binding algorithm for FPGAs was presented, and included a high-level power estimator. In [3] and [4], the authors presented a simulated annealing-based algorithm which carried out high-level synthesis subtasks simultaneously, targeting FPGAs for low-power, called LOPASS. Their binding algorithm initially used minimum weight bipartite matching, and then was enhanced using a network flow approach presented in [2] that binds all the resources simultaneously. We compare our own algorithm to LOPASS and show that an iterative approach enables greater power savings.

HLPower, the FPGA targeted, low-power binding algorithm we present here, sets itself apart from previous work in that it considers low-level glitches in its power estimation. This allows HLPower to target a major contribution to dynamic power during high-level synthesis that has not been considered before. We think the main reason this has been missing is because of the difficulty of estimating glitches during high-level synthesis. We present a unique way to address this problem in this paper.

3. PROBLEM FORMATION

The input to our binding algorithm is a scheduled CDFG, a resource constraint, and a resource library. The problem to be solved involves the allocation and assignment of registers to variables, and functional units to operations. Efficient sharing of functional units by operations, and registers by variables, in order to reduce power and area, are the challenges of binding. The binding problem can be formulated as follows:

Given: A scheduled CDFG, a resource constraint, and a resource library.

Tasks: Allocate and bind registers to variables, and allocate and bind functional units to operations.

Objectives: Produce a valid binding solution while meeting the resource constraint and optimizing the solution for power and area on the targeted FPGA.

4. SWITCHING ACTIVITY ESTIMATION

As dynamic power estimation is a central driver of our binding algorithm, the way this is accomplished is described in this section. Dynamic power is estimated in the form of a switching activity model based on probabilistic techniques developed originally in [17], extended in [7], and further developed to target FPGA mapping and include glitches in [6].

In [17] the ideas of *transition density* (also referred to as *toggle rate* or *switching activity*) and *signal probability* are initially developed. The *transition density* of a logic signal is defined as the average number of transitions per unit time, while the *signal probability* is defined as the fraction of the time that the logic signal is in the 1 state (i.e. the average value of the logic signal over all time). An efficient technique is presented that allows for the calculation of the total circuit switching activity by means of propagation of transition densities and signal probabilities from input nodes to output nodes. For a node y with independent fanin nodes $x_1, x_2, \ldots, x_n$, and given the transition density (switching activity) $s(x_i)$ of each fanin node x_i, the transition density of node y, $s(y)$ can be computed using the Boolean difference $(\partial y/\partial x)$ of y with respect to x_i:

$$s(y) = \sum_{i=1}^{n} P(\frac{\partial y}{\partial x_i}) s(x_i) \quad (1)$$

where $P(\partial y/\partial x_i)$ is the signal probability of the Boolean difference.

This technique was extended in [7] to take into account simultaneous switching, something that the technique in [17] lacked. Let y be a Boolean expression, $y(t)$ be its value at time t, $P(y)$ be the signal probability of y, and $s(y)$ now be the normalized switching activity of y. $s(y)$ is the probability of y having different values at time t and $t+T$, where T is a unit time period, and is thus given by $s(y) = P(y(t)\overline{y(t+T)}) + P(\overline{y(t)}y(t+T))$. Additionally, note that $P(y(t)\overline{y(t+T)}) = P(\overline{y(t)}y(t+T))$. Thus, $P(y(t)\overline{y(t+T)}) = P(\overline{y(t)}y(t+T)) = 1/2 \cdot s(y)$. Since $P(y(t)) = P(y(t)y(t+T)) + P(y(t)\overline{y(t+T)})$, we find

$$s(y) = 2(P(y(t)) - P(y(t)y(t+T))) \quad (2)$$

And, as noted in [6], the term $P(y(t)y(t+T))$ can be calculated from the probabilities and switching activities of fanin nodes of y using the procedure in [7].

Finally, the technique for switching activity estimation was applied to FPGA technology mapping in [6]. The algorithm in [6] reads in a netlist, and uses a cut-enumeration technique [8] to select K-input cuts that will be mapped to the FPGA (K-input look-up tables). Primary inputs are assumed to have signal probabilities and switching activities of 0.5. For each node, the signal probability of all of the K-input feasible cuts of that node are computed using the weighted averaging algorithm from [12]. When calculating the switching activities for each cut, the widely accepted unit delay model is assumed for the FPGA look-up tables. This means that signal transitions are assumed to happen

Algorithm 1 HLPower Binding Algorithm

```
1: Input: Scheduled CDFG, library, resource constraint
2: Output: Scheduled and bound CDFG
3: precalc SA values for all FU & MUX combinations
4: bind registers according to [11]
5: traverse CDFG, select nodes for set U
6: put remaining nodes in set V
7: while resource constraint is not met do
8:    initialize bipartite graph G = (U, V, E)
9:    for all edges in E do
10:      calculate input MUX sizes (if nodes were combined)
11:      look up SA value for particular FU & MUXs
12:      calculate edge weight
13:   end for
14:   solve G for maximum weight
15:   combine matched nodes & allocate functional units
16: end while
```

only at discrete time units: $1, 2, \ldots, D(C)$, where $D(C)$ is the depth of the cut. The transition that takes place at time $D(C)$ is considered the functional transition, while the transitions that occur at the other time steps are considered glitches. Switching activities are then calculated and propagated through the cut according to Equation (2). For a given cut, the effective switching activity is a summation of the switching activities at each time step. An example of how this is accomplished can be found in [6].

The best cuts, those with the lowest switching activities, are then chosen for implementation of the node in the FPGA. Summing up the switching activities, sa_i, for all of the selected cuts, $1, 2, \ldots n$, provides the *total estimated switching activity*, SA, for the netlist:

$$SA = \sum_{i=1}^{n} sa_i \tag{3}$$

The *total estimated switching activity*, SA, is used in the binding algorithm. This technique for switching activity estimation has the advantages over previous techniques of being mapping-aware and considering glitches.

5. BINDING ALGORITHM

The HLPower binding algorithm proceeds in two major parts. First, registers are allocated and bound, and second, functional units are allocated and bound. In this paper we focus on the functional unit binding. The functional unit binding proceeds in an iterative fashion until the resource constraint is met, driven by the estimated dynamic power usage and multiplexer sizes of various operation-to-functional unit bindings, as will be explained below. Algorithm 1 provides a summary of the HLPower binding algorithm.

5.1 Register Binding

Register binding is accomplished in a manner similar to that described in [11], where variables are bound by solving a weighted bipartite graph. An allocated set of registers is determined by counting the number of variables present in the control step with the largest number of variables with overlapping lifetimes. This set of registers is allocated, and a cluster of mutually unsharable variables (meaning the lifetimes of these variables are overlapping) is bound at a time, by way of a weighted bipartite graph, sorted in ascending order according to their birth times. Operator ports are randomly bound during this step.

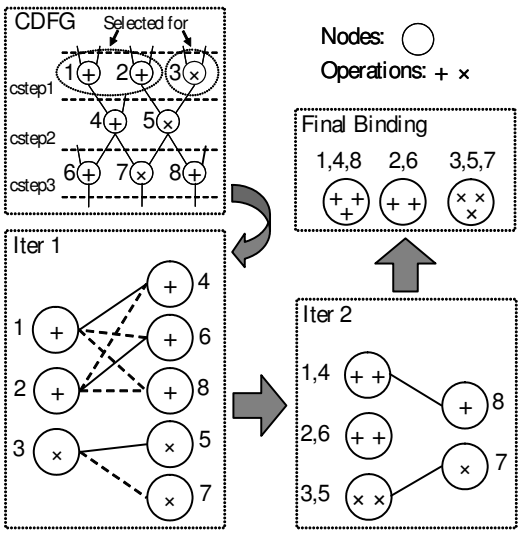

Figure 1: An example of the functional unit binding.

5.2 Functional Unit Binding

5.2.1 Algorithm Overview

Functional unit binding iteratively constructs weighted bipartite graphs, finds a maximum matching, and combines nodes that are matched. Before the first iteration of the functional unit binding, the scheduled CDFG is traversed, and for each operation type, the control step with the largest number of operations of that type is found. This gives a lower bound on the possible resource constraint. These operations are selected to make up one set of vertices (or nodes), U, in the bipartite graph. The second set of vertices, V, includes all of the other nodes. See Figure 1.

During functional unit binding, the nodes of the graph are each considered an allocated functional unit. Initially, as none of the operations have been bound to functional units, every operation is considered to be bound to its own functional unit, and each is represented by an individual node of the graph. On subsequent iterations, each node (functional unit) of the graph may contain more than one operation. Edges (making up the set E) are created between compatible nodes in the graph. Two nodes are compatible if they meet the following two criteria:

1. They perform the same type of operation, e.g. are both multiplications.
2. They do not contain any operations that have overlapping lifetimes in the schedule.

Each edge of the graph represents a possible binding of two sets of operations to the same functional unit. Edge weights are then assigned as described in the next section. This formulation is similar to binding that works with a compatibility graph, but not all nodes will be bound in a single iteration.

Figure 1 illustrates the bipartite graph formulation. In iteration one, add operations 1 and 2, and mult operation 3 are selected for set U, because they come from the control steps of maximum density for their respective types in the scheduled CDFG. (Note that, alternatively, any of the mult operations could have been chosen, or add operations 6 and 8 could have been chosen.) Solid edges represent those selected in the maximum weighted matching. Nodes are combined, and in iteration two nodes are further combined.

In iteration three there is no longer any compatibility between the nodes, and the algorithm is completed. The final allocation is 2 adders and 1 multiplier.

THEOREM 1. *A weighted bipartite graph $G = (U, V, E)$, containing the single-cycle operations of a scheduled CDFG, if iteratively generated and solved, combining matching nodes in each iteration (as previously described), guarantees that the minimum possible resource constraints can be met.*

PROOF. Suppose on the contrary that the minimum resource constraint cannot be met. This would mean that there exists a node in the set V that is incompatible with all nodes in the set U. Since this incompatibility could not be due to compatibility criterion 1 given above—the non-existence of a compatible operation type (if, for example, there was only one operation of a particular type in a CDFG, then it would already lie in set U)—it must be due to criterion 2—operations that have overlapping lifetimes. That would imply that there were more operations in the incompatible operation's control step than were in the control step chosen initially for set U. This could not be the case due to the selection criteria for set U. □

Theorem 1 guarantees that, for a library of single-cycle resources, the minimum resource constraint for the given scheduled CDFG can be met. Although no similar guarantee can be made for multi-cycle resources, our experiments show that the algorithm is nonetheless effective in achieving a minimum resource allocation in most cases. The runtime complexity of the algorithm is $O(|N|^2 * (|E| + |N|log|N|))$, where $|N|$ is the total number of nodes in the CDFG. This is because the number of bipartite graphs solved is, in the worst case, linear with the number of nodes in the CDFG.

5.2.2 Edge Weight Calculation

Edge weights for each graph edge are calculated as follows:

1. The sizes of the input multiplexers to the functional unit to which the operations connected by the edge would be bound (if the matching included the given edge) are found. This is possible because the registers have already been assigned, enabling the calculation of the exact multiplexer sizes. This combination of multiplexers and a functional unit make up a partial datapath.
2. A gate-level netlist of the partial datapath is generated in .blif format [19]. This is accomplished by creating a new .blif file with proper input and output ports, importing existing instantiations of the multiplexers and functional units, and making the necessary connections. See Figure 2.
3. The switching activity is estimated for the gate-level netlist (.blif), based on the technique described in Section 4. This produces an estimate of the dynamic power, including glitch power, which will be used to estimate part of the cost of this particular binding of operations to functional unit.
4. The weight on the edge is computed as follows:

$$w(e_{i,j}) = \alpha \times \frac{1}{SA} + (1-\alpha) \times \frac{1}{(muxDiff + 1) \times \beta} \quad (4)$$

where SA is the *total estimated switching activity* as defined in Equation (3), α is a weighting coefficient, β is a value used to adjust the size of the $muxDiff$ factor relative to SA (based on emperical study $\beta \approx 30$ for add operations, and 1000 for mult), and $muxDiff$ is defined as the absolute difference in the sizes of the two multiplexers that input to the functional unit. ($(muxDiff + 1)$ is used so the equation will be valid if $muxDiff = 0$.)

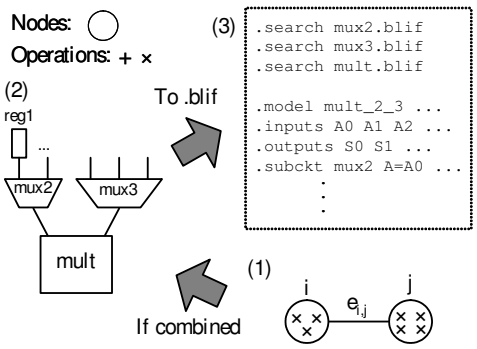

Figure 2: Gate-level partial data-path netlist generation. Based on the register binding, and operations assigned to the edge nodes (1), it is determined that a 2-input and a 3-input MUX are needed (2). The .blif netlist is then generated (3).

Equation (4) uses the weighting coefficient α to balance the contribution to the weight of two important factors for the reduction of glitches and switching activity: the *total estimated switching activity*, or SA, and the difference in size between the two multiplexers, or $muxDiff$.

SA provides a *low-level* consideration of the circuit. It explicitly estimates dynamic power usage (including glitches) at the gate-level, taking into account the multiplexers in the partial datapath. It also implicitly considers area through the number of look-up tables required to implement the partial datapath, because a larger area correlates with a higher SA. $muxDiff$, on the other hand, provides a *high-level* consideration of the circuit. It explicitly considers multiplexer balancing, which would have a direct impact on glitch reduction, even if the SA estimation is not 100% accurate. This combination of *high-level* multiplexer balancing, and *low-level* SA estimation, work together to select the best matches for power and area reduction in each iteration of the binding algorithm.

As dynamic calculation of the switching activities for each edge during the binding iterations can be time consuming, in our experiments we precalculate the switching activities for all combinations of multiplexers and functional units. This is done by generating the gate-level netlists for the partial data-path of each combination of functional unit and multiplexers, and running the SA estimation on each. The calculated SA values are then stored in a text file. A hash table is then generated when HLPower is initially run by reading in the precalculated values from the text file. This allows fast look-up of the estimated SA value for a particular combination of input multiplexer sizes, and functional unit. Experimental results show that this method provided us with the same results as running the algorithm with dynamic SA estimation, but with a much shorter run time.

The iterative approach to functional unit binding allows the multiplexer size to be better controlled than is possible with single iteration approaches, such as with a network flow algorithm. By iteratively building up the numbers of operations assigned to allocated functional units, the multiplexer sizes, balance among multiplexers, and the contributions of

Table 1: Benchmark Profiles.

Benchmarks	No. of PIs	No. of POs	No. of Adds	No. of Mults	Total No. of Edges
chem	20	10	171	176	731
dir	8	8	84	64	314
honda	9	2	45	52	214
mcm	8	8	64	30	252
pr	8	8	26	16	134
steam	5	5	105	115	472
wang	8	8	26	22	134

Table 2: Resource Constraints, Scheduling Length, and Number of Registers used for both LOPASS and HLPower binding. Identical schedules and register bindings were used by both LOPASS and HLPower.

Benchmarks	Add	Mult	Cycle	Reg	HLPower Runtime (s)
chem	9	7	39	70	812
dir	3	2	41	25	56
honda	4	4	18	13	14
mcm	4	2	27	54	16
pr	2	2	16	32	2
steam	7	6	28	39	189
wang	2	2	18	39	2

the multiplexers to dynamic power (including glitch power) consumption can be carefully controlled and evaluated.

6. EXPERIMENTAL RESULTS

6.1 Experimental Setup

Our experiments are carried out on a 2.8 GHz Intel Pentium 4 Linux machine, with 2 GB of memory. A number of data-intensive benchmarks are used. The benchmark CDFGs include several different DCT algorithms including *pr*, *wang*, and *dir*, and several DSP programs including *chem*, *steam*, *mcm* and *honda*. The benchmarks are profiled in Table 1. Each node in the benchmarks is either an addition/subtraction or a multiplication.

We compare our binding algorithm, HLPower, to a state-of-the-art low-power high-level synthesis algorithm for FPGAs, LOPASS [3] [4], which can perform scheduling, allocation, and binding. We use a resource library containing single-cycle resources, including a multiplier, an adder, a register, and multiplexers. The benchmarks are first run through LOPASS, and a binding solution is obtained. Then they are run through HLPower with the same schedule, register allocation, and resource constraints, to obtain the HLPower binding solution. Table 2 summarizes the scheduled benchmark characteristics used for both the LOPASS and the HLPower solutions.

The binding solutions, in CDFG format, are then converted to RTL design in VHDL with a CDFG to VHDL tool. To verify our results using a commercial tool, the designs are put into Quartus II for RTL synthesis, placement and routing, timing analysis, simulation, and power analysis. This is done by first building a project on each benchmark's VHDL, setting the device family to Cyclone II, and selecting the same device for each benchmark. We use the Quartus II vector waveform file (.vwf) editor to generate 1000 random input vectors for each benchmark. We also set the simulator settings *glitch filtering* to *never*, *max balancing dsp blocks* to *0*, *wysiwyg remap* to *on*, *optimization technique* to *speed*, and *synthesis effort* to *fast*. These settings help to ensure that the benchmarks for both LOPASS and HLPower are synthesized in the same way without Quartus II optimizations that would invalidate the power results produced by both algorithms. The same .vwf file is used for both LOPASS and HLPower. Then we run the command *quartus_sh –flow compile* (which runs the synthesis, placement and routing, and timing analysis), *quartus_sim* (which makes use of the .vwf file, and generates a switching activity file, .saf), and *quartus_pow* (which makes use of the .saf file). The command *quartus_pow* runs PowerPlay Power Analyzer, and reports the dynamic power consumption.

Table 4: Mean and variance of $muxDiff$ across all allocated resources for the final binding solutions.

Benchmarks	LOPASS	HLPower $\alpha = 1$	HLPower $\alpha = 0.5$	
	mean/var	mean/var	mean/var	# muxes
chem	7.4/16.1	4.6/9.8	2.4/5.3	16
dir	5.4/12.2	4/11.2	4.2/3.8	5
honda	3.1/11.1	3.9/6.4	3/6.3	8
mcm	1/0.3	1.8/0.5	0.5/0.3	6
pr	0.8/0.2	0.3/0.2	0.8/0.2	4
steam	8.1/56.1	6.8/29.9	5.8/26.7	8
wang	1.3/0.7	0.8/0.2	1.8/0.7	4
average	3.9/13.8	3.2/8.3	2.6/6.2	

6.2 FPGA Area and Power Reduction Results

Table 3 summarizes the synthesis and power analysis results for both LOPASS and HLPower (with $\alpha = 0.5$ from Equation (4)), for each benchmark. There was an average reduction in dynamic power of 19.3%, and area (in the form of LUTs) of 9.1%. These reductions came at the expense of 0.6% of the clock period, on average. For comparison, an $\alpha = 1$ yielded an average power reduction of 6.5%, clock period increase of 3.5%, and an area reduction of 5.1%. (See below for a further discussion of the choice of α.)

Table 3 columns 5, 6, 10, and 11 show the multiplexer reduction results. In the table, *Largest MUX* is the largest multiplexer needed to implement the binding solution, while *MUX length* is a measure of the total number of multiplexers implemented, and is calculated by adding up the total number of multiplexer inputs (sizes).

HLPower reduced the largest multiplexer size by an average of 2.6, and the length an average of 7.2%, over LOPASS. Additionally, Table 4 shows the change in the mean and variance of $muxDiff$ across all allocated resources for the final binding solutions of LOPASS, HLPower with $\alpha = 1$, and HLPower with $\alpha = 0.5$. The variation from $\alpha = 1$ (no $muxDiff$ influence in the weighting equation) to $\alpha = 0.5$ (equal weighting of SA and $muxDiff$) shows a clear reduction in both the mean and variance of $muxDiff$. (Note that the same number of multiplexers were allocated in each of the solutions.) This better balancing of multiplexers on the resource inputs contributes to a power reduction by balancing paths and eliminating extra glitch transitions.

Evidence for the decrease in switching activity, and indirectly glitches, is given in Figure 3. Average toggle rate is defined as number of transitions per second, and is a number reported by Quartus II. HLPower with $\alpha = 1$ reduces the toggle rate for most benchmarks (averaging 8.4%), while $\alpha = 0.5$ produces reductions for all benchmarks, averaging 21.9%. Again we see that SA estimation alone (in Equation (4)) effectively helps reduce toggling, but the combina-

Table 3: Power, Clock Period, Number of LUTs, and Multiplexer Results for LOPASS and HLPower Bindings.

	LOPASS/HLPower					Change				
Bench-marks	Dynamic Power (mW)	Clk Per. (ns)	LUTs	Largest MUX	MUX Length	Dynamic Pow.(%)	Clk Per. (%)	LUTs (%)	Lrgst MUX	MUX Len.(%)
chem	1602.3/1468.6	26.0/27.5	9,806/9,613	26/23	672/637	-8.35	5.67	-1.97	-6	-5.2
dir	709.1/405.8	23.8/24.2	4,527/3,453	18/15	167/157	-42.78	2.04	-23.72	-3	-6.0
honda	658.7/534.1	23.5/23.2	3,352/3,057	15/13	165/162	-18.92	-1.40	-8.80	-2	-1.8
mcm	351.3/208.7	24.1/24.2	3,274/2,548	17/14	159/153	-40.60	0.38	-22.17	-3	-3.8
pr	232.7/192.9	20.9/21.7	1,714/1,732	11/8	70/57	-17.09	3.60	1.05	-3	-18.6
steam	729.6/690.6	24.4/23.6	5,121/4,469	19/22	429/321	-5.35	-3.32	-12.73	3	-25.2
wang	161.5/158.5	20.5/19.9	1,697/1,775	12/8	69/76	-1.85	-2.88	4.60	-4	10.1
Average						-19.28	0.58	-9.11	-2.6	-7.2

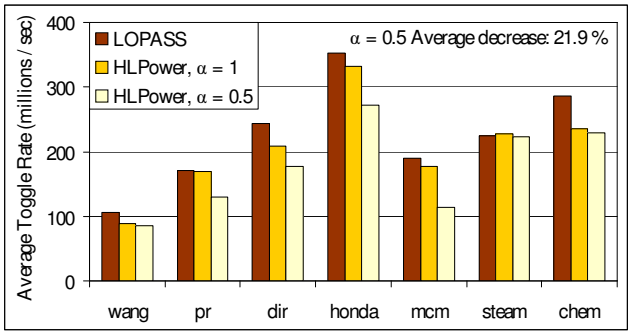

Figure 3: Average Toggle Rate.

tion of SA and $muxDiff$ is more effective. The combination of area savings through multiplexer reduction, and reduced glitching (as evidenced by the significant decrease in toggle rate), produces an aggregate reduction in dynamic power, for a negligible change in clock period.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a new high-level binding algorithm for power and area reduction, targeting FPGAs. The binding algorithm, HLPower, is based on weighted bipartite matching, and makes use of glitch power-aware, dynamic power estimation. HLPower successfully reduces the switching activity and area of a design, producing savings in dynamic power consumption. Future work will include integrating HLPower into a complete high-level synthesis algorithm that includes scheduling and module selection, while providing better support for multi-cycle resources.

8. REFERENCES

[1] J.-M. Chang and M. Pedram. Register allocation and binding for low power. *DAC*, 1995.

[2] D. Chen and J. Cong. Register binding and port assignment for multiplexer optimization. *ASP-DAC*, 2004.

[3] D. Chen, J. Cong, and Y. Fan. Low-power high-level synthesis for FPGA architectures. In *ISLPED*, 2003.

[4] D. Chen, J. Cong, Y. Fan, and L. Wan. LOPASS: A low-power architectural synthesis system for FPGAs with interconnect estimation and optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, to be published.

[5] D. Chen, J. Cong, Y. Fan, and Z. Zhang. High-level power estimation and low-power design space exploration for FPGAs. In *ASP-DAC*, 2007.

[6] L. Cheng, D. Chen, and M. D. F. Wong. GlitchMap: an FPGA technology mapper for low power considering glitches. In *DAC*, 2007.

[7] T.-L. Chou and K. Roy. Estimation of activity for static and domino CMOS circuits considering signal correlations and simultaneous switching. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(10):1257–1265, Oct. 1996.

[8] J. Cong, C. Wu, and Y. Ding. Cut ranking and pruning: enabling a general and efficient FPGA mapping solution. In *FPGA*, 1999.

[9] A. Dasgupta and R. Karri. Simultaneous scheduling and binding for power minimization during microarchitecture synthesis. In *ISLPED*, 1995.

[10] A. Davoodi and A. Srivastava. Effective techniques for the generalized low-power binding problem. *ACM Trans. Des. Autom. Electron. Syst.*, 11(1):52–69, 2006.

[11] C.-Y. Huang, Y.-S. Chen, Y.-L. Lin, and Y.-C. Hsu. Data path allocation based on bipartite weighted matching. *DAC*, 1990.

[12] B. Krishnamurthy and I. Tollis. Improved techniques for estimating signal probabilities. *IEEE Transactions on Computers*, 38(7):1041–1045, Jul. 1989.

[13] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. In *FPGA*, 2006.

[14] E. Kursun, A. Srivastava, S. O. Memik, and M. Sarrafzadeh. Early evaluation techniques for low power binding. In *ISLPED*, 2002.

[15] J. Lamoureux, G. G. Lemieux, and S. J. E. Wilton. GlitchLess: an active glitch minimization technique for FPGAs. In *FPGA*, 2007.

[16] F. Li, Y. Lin, L. He, D. Chen, and J. Cong. Power modeling and characteristics of field programmable gate arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*, 24(11):1712–1724, Nov. 2005.

[17] F. Najm. Transition density: a new measure of activity in digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(2):310–323, Feb. 1993.

[18] B. Pangrle. On the complexity of connectivity binding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(11):1460–1465, Nov. 1991.

[19] E. Sentovich, et al. SIS: A system for sequential circuit synthesis. Technical report, UCB/ERL Memorandum M89/49, Department of EECS, University of California, Berkeley, Nov. 1992.

[20] F. Wolff, M. Knieser, D. Weyer, and C. Papachristou. High-level low power FPGA design methodology. *NAECON*, 2000.