Behavioral Synthesis with Activating Unused Flip-Flops for Reducing Glitch Power in FPGA

Cheng-Tao Hsieh[+], Jason Cong[++], Zhiru Zhang[++], Shih-Chieh Chang[+]
[+]Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
[++]Computer Science Department, University of California, Los Angeles, USA
cthsieh@gmail.com, {cong, zhiruz}@cs.ucla.edu, scchang@cs.nthu.edu.tw

ABSTRACT

In this paper we discuss optimizing the interconnect power of designs implemented in FPGA platforms. In particular, we reduce the glitch power on interconnects associated with the output of functional units in a design. The idea is to activate unused flip-flops to block the propagation of glitches, which takes advantage of the abundant flip-flops in modern FPGA structures. Since the activation of additional flip-flops may cause data hazard problems, we develop several effective behavioral synthesis techniques to prevent such data hazards. We also study the optimality of our techniques. The experimental results show that on average, our methods lead to a 28% reduction in dynamic power in the Xilinx Virtex-II platform.

1. Introduction

Power efficiency is becoming a forefront concern of FPGA designs in nanometer-scale technologies. The research in [14][20] has shown that interconnect resources dominate the power consumption in modern FPGA designs. In particular, interconnect could dissipate at least 60% of the total power in the Xilinx Virtex-II family [20]. Therefore, reducing interconnect power is important for FPGA designs to achieve power efficiency.

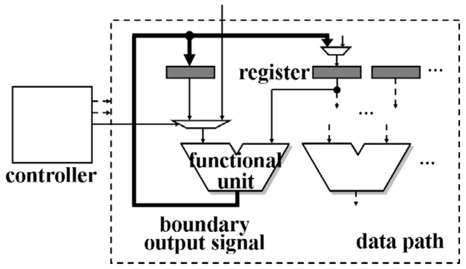

Figure 1: Generic structure of the FSMD model.

A synchronous design can be implemented with the architecture of *finite state machine with data path* (*FSMD*). Figure 1 shows the generic structure of FSMD. The data path contains arithmetic functional units as well as registers which serve to temporally store computation results between functional units. We use the term *boundary output signal* to refer to the interconnect at the boundary of the data path (bold line in Figure 1), which is between the output of functional units and the input of registers.

It is important to note that a boundary output signal may have multiple fanouts; i.e., a functional unit is connected to several registers, as shown in Figure 2(a). This occurs commonly if during resource binding, multiple operations are bound to the same functional unit, and those operations produce results with overlapped lifetimes.

We observed that, when using an FPGA to implement designs, if we insert a *single* register at a multi-fanout boundary output signal, as shown in Figure 2(b), the power consumption on the boundary output signal could significantly decrease. This is because large glitches which originally propagate through the whole boundary output signal now occur only in the interconnect with an extremely small capacitance *C* between the inserted register and the functional unit. We call such an additional register a *firewall register* due to its ability to filter out unwelcome glitches.

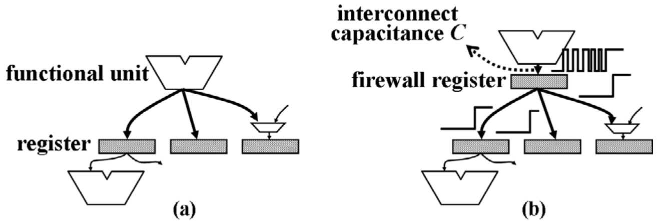

Figure 2: The insertion of a firewall register.

Interconnect capacitance *C* in Figure 2(b) can be much smaller than the capacitance of the whole boundary output signal if we implement it using FPGA. Figure 3 shows a typical FPGA structure. A basic logic element (e.g., a *Logic Element* in Altera Stratix FPGAs or a *Slice* in Xilinx Virtex series FPGAs) contains a LUT as well as a flip-flop, so that the output of a logic gate (implemented in LUTs) can be configured as either an unregistered mode or a registered mode. Precisely, if a logic gate is connected to only one register, it can be implemented in the registered mode. Inserting a firewall register to a functional unit creates such a situation to benefit the registered mode. Therefore, capacitance *C* in Figure 2(b) indicates the interconnect capacitance between the LUT and the local flip-flop inside a basic logic element, which is much smaller than the capacitance of the inter-block programmable interconnect.

The insertion of firewall registers is not trivial because a firewall register delays data propagation from a functional unit to its original registers for one clock cycle, thus possibly causing *data hazard* problems. We observed that the hazard problem can be solved by scheduling and binding operations in a particular way. Therefore, in this paper we propose novel scheduling and binding methods to generate functionally correct, low-power RTL designs with firewall registers.

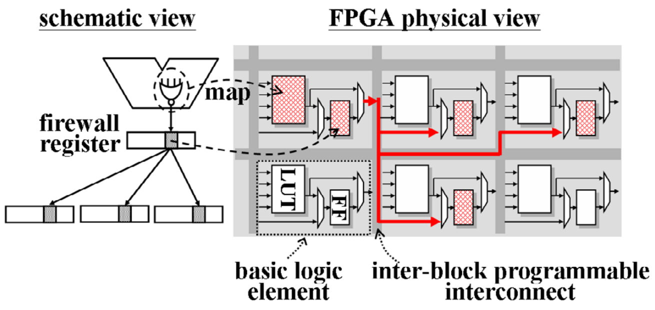

Figure 3: Firewall register implemented in a modern FPGA.

We intend to insert firewall registers to those functional units generating large glitches at the output. This implies that our method needs an accurate glitch estimation to guide the insertion. Considering that the occurrence of glitches is sensitive to component delays, we suggest to use our behavioral synthesis method for *data intensive* designs, which are mainly composed of arithmetic functional units serving as IPs. Those IP blocks have pre-determined circuit structure so the glitch information is predictable in the behavioral synthesis stage. Our problem formulation considers power models [7][12][19][21] for each arithmetic module as the input.

In our experiments, we applied our method to a set of data intensive designs, and obtained, on average, 28% power reduction with 4% area overhead. The small area overhead implies that the additional control circuit for firewall registers is insignificant. Also, the leakage power is not impacted much because leakage is roughly proportional to a design's area.

Similar to our idea, pipelining [15][18][22] and retiming [13][17] also adopt flip-flops to block the propagation of glitches for power minimization. However, all of them assume that an RTL design is given so they cannot change the computation sequence as scheduling does. This makes the previous methods hard to solve the data hazard problem encountered in our problem formulation. They focus on activating unused flip-flops in a local scale, particularly, within a functional unit. Still, one can apply both our method and previous methods in different stages of design flow for low power.

Our major contributions are 1) to expend the solution space of low-power implementation by proposing an additional dimension of using/not using firewall register, and 2) to provide methods to guide the insertion of firewall registers in the behavioral synthesis stage. We have incorporated our techniques into a behavioral synthesis tool, xPilot, introduced in [8].

2. Data Hazard Problems

Inserting firewall registers may impact the correctness of a design's function. In fact, as we will discuss later, there exists a certain scheduling and binding condition where the use of firewall register will cause functional errors. Therefore, if we want to take advantage of firewall register, we must avoid scheduling and binding a design in that way. In this section we discuss the scheduling and binding pattern to be avoided.

A design's function can be represented as a *data-flow graph* (*DFG*). A DFG is a directed acyclic graph (DAG), where every node represents an operation, such as an addition or a multiplication, and every directed edge (u, v) represents a *dataflow* indicating that operation u produces values to be consumed by v. After scheduling, we can derive a *scheduled DFG*, where every operation is scheduled to execute at one or more consecutive *control steps* (*c-steps*).

To maintain a design's functionality after inserting firewall registers, we have to guarantee that for every dataflow (u, v), consuming operation v correctly read results from producing operation u. Figure 4(a) shows a partial scheduled DFG, where operations u and v form a dataflow (u, v) and are bound to functional units p and q, respectively. Note that the use of the firewall register will delay the data transfer from functional unit p to register r by one c-step. In case functional unit q intends to fetch from register r a value that is still stored at the firewall register, a functional error occurs. We need to carefully deal with this condition when using firewall registers.

We can just "forward" the results from the firewall register to functional unit q as shown in Figure 4(b), which is traditionally called *forwarding*. Forwarding can absolutely resolve the functional error problem if the consuming operation v is executed in a single c-step. In this case, the firewall register is required to keep the target results for only one c-step during operation v's reading. However, if the consuming operation v is a multi-cycle operation, the firewall register must keep the target results for several c-steps until operation v finishes the reading. In case the firewall register cannot keep the target results long enough, a functional error still occurs. We elaborate this issue using an example.

Figure 5 shows a partial scheduled DFG, where dataflow (u, v) are bound to functional units p and q, respectively. In addition, operation w is also bound to functional unit p, sharing the same functional unit with the producing operation u. Note that the consuming operation v is a two-cycle operation and will read results through forwarding during c-steps i and i+1, so the firewall register must keep the producing operation u's results during the two c-steps. However, at the end of c-step i, functional unit p will finish the computation of operation w and store its results to the firewall register, which, accidentally, overwrites the result that is still forwarded to functional unit q. Formally speaking, a *write-after-read* (*WAR*) hazard occurs on the firewall register.

Note that we cannot attach two firewall registers to functional unit *p* to store lifetime-overlapping results from operations *u* and *w*, because this way the two firewall registers will not be implemented in local flip-flops, making no power reduction as shown in Figure 3. On the contrary, because we can attach "original" registers, such as register *r* in Figure 4(a), as many as possible to store lifetime-overlapping results, it is impossible for original registers to involve WAR hazards.

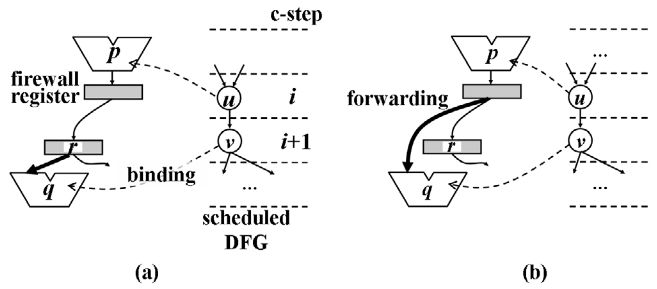

Figure 4: Forwarding.

We formally describe the conditions to induce a WAR. Assume that an operation *v* is a non-pipelined multi-cycle operation and is executed at *k* consecutive c-steps, which are labeled by consecutive integers {*i*, *i*+1, …, *i*+*k*-1}.

Lemma 1: Inserting a firewall register for a dataflow (*u*, *v*) causes a WAR hazard if and only if: (1) operations *u* and *v* are scheduled at consecutive c-steps, and (2) there exists an operation *w* such that *w* and *u* are bound to the same functional unit, and *w* produces results between c-step *i* and *i*+*k*-2.

Proof: Figure 5 shows a simple instance verifying this lemma. **Q.E.D.**

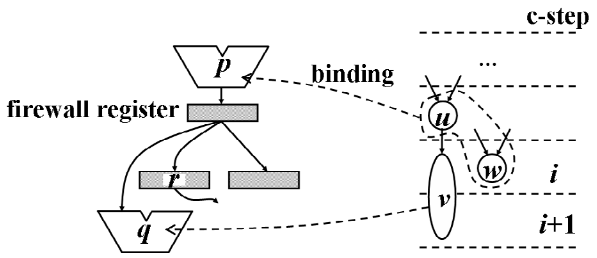

Figure 5: Dataflow (*u*, *v*) with a WAR hazard after firewall register insertion.

To maintain a design's function, we cannot apply firewall registers to dataflows satisfying Lemma 1. However, this will reduce the opportunity of using firewall registers for low power, so we should carefully perform scheduling and binding to avoid such conditions.

3. Binding with Firewall Register Insertion Support

In this section we discuss how to perform resource binding to avoid the conditions in Lemma 1. Our idea can be briefly illustrated with the example in Figure 5. Since operations *u* and *w* are bound to the same functional unit *p*, according to Lemma 1, a hazard on dataflow (*u*, *v*) appears. If we can separately bind operations *u* and *w* to two functional units, Lemma 1 will become unsatisfied.

Traditionally, binding achieves power optimization by minimizing the switching activities of resources [1][5][6][16]. In this research we perform a low-power binding by considering both the switching activity and the insertion of firewall registers simultaneously. The problem formulation is described as follows.

Given: (1) A scheduled DFG $G=(V, E)$; (2) a set of resources R; (3) switching activity s_{uw} on $u \rightarrow w$, where $u, w \in V$; (4) power models for each type of resource.

Goal: Generate a functional unit binding $\{B: v \text{ to } r \mid \text{for all } v, \text{ where } v \in V \text{ and } r \in R\}$, and for every $r \in R$ determine whether it is protected by a firewall register. The objective is to minimize the power of the functional units.

In our problem formulation, the resource number is a constraint. Therefore, although our method seems to increase the usage of resources due to binding operations to separate resources, our method will not use more resources than conventional binding approaches. However, the resource constraint limits how much our binding can avoid Lemma 1. The looser the resource constraint, the more the dataflows to be protected by firewall register.

3.1 Network Flow Formulation

We adopt network flow formulation to solve the binding problem. We will show that, through proper network construction and an optimal min-cost flow algorithm, we can derive an optimal solution for the goal. The outline of our algorithm is shown as follows.

Algorithm:

(1) Build a graph *H* representing the compatible information among operations in *V*.

(2) Assign cost and capacity constraints to the edges in *H*.

(3) Solve the min-cost flow problem on *H* with a set of equal integral flow constraints.

(4) Derive a binding solution with firewall register insertion based on the solution in step (3).

We first introduce several notations in our formulation. In a DFG, $G=(V, E)$, different types of operations (e.g., addition and multiplication) are bound separately. We use V_f to denote the set of operations in type *f*. For two operations *u* and *w* of type *f*, if their corresponding lifetimes do not overlap, we call *u* and *w* *compatible* with each other. Two compatible operations can be bound to a single functional unit. Next we define two operations to be *FR-compatible* as follows.

Definition 1: Two operations *u* and *w* of type *f* is FR-compatible if and only if (1) they are compatible, and (2) when they are bound together to a functional unit, the functional unit can be protected by a firewall register; i.e., the conditions in Lemma 1 do not occur.

For example, in Figure 5 operations *u* and *w* are **not** FR-compatible because binding them to the same functional unit forbids inserting a firewall register to that functional unit.

We intend to build a graph $H = (s, t, V_H, E_H, C, K_l, K_u)$ based on the compatibility information among operations.

First there are source node s and sink node t in H. Next, V_H is the node set of the network. For each operation $v \in V_f$ there are six corresponding nodes in V_H, as shown in Figure 6. We denote the six nodes as $\{v_{FRin}, v_{FRout}, v_{Pin}, v_{Pout}, v_{Xin}, v_{Xout}\}$, where nodes v_{FRin} and v_{FRout} are responsible for the situation of associating a firewall register, v_{Pin} and v_{Pout} are responsible for the situation without firewall registers, and v_{Xin} and v_{Xout} are responsible for the exclusive constraint that operation v is either with a firewall register or not.

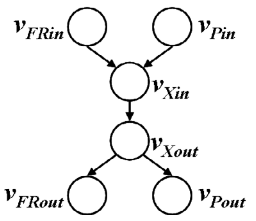

Figure 6: The six corresponding nodes in *H* of operation *v*.

E_H is the edge set of the network. The edges in E_H can be classified into three categories.

(1) The internal edges among the six corresponding nodes of an operation v, as shown in Figure 6.

(2) If two operations u and w are compatible and u comes before w, edge (u_{Pout}, w_{Pin}) is in E_H.

(3) If two operations u and w are FR-compatible and u comes before w, edge (u_{FRout}, w_{FRin}) is in E_H.

C is the cost assigned to the edges in E_H, which is set in the following way.

$C(u_{Pout}, w_{Pin}) = power_{primitive}(s_{uw})$

$C(u_{FRout}, w_{FRin}) = power_{FR}(s_{uw})$

$C(x, y) = 0$ for any other edge (x, y) in E_H.

In this formulation the dynamic power is affected by the switching activity as well as whether a functional unit is protected by a firewall register. There must be two power models for each type of functional unit. Power model $power_{FR}(s_{uw})$ is used to calculate the power for the case of binding operations u and w together with a firewall register; $power_{primitive}(s_{uw})$ is for the case of binding operations without a firewall register. Both the models take switching activity as input. There have been plenty of papers [7][12][19][21] discussing how to derive power models for behavioral synthesis. Especially those methods can take glitches into account when characterizing the power for pre-designed IP blocks. Then we assign the calculated power values to the corresponding edges in H as the cost.

Finally, K_l is the lower bound flow capacity, which is set to 0 for every edge in E_H; K_u is the upper bound flow capacity, which is set to 1.

We use an example to illustrate the construction of H. Figure 7(a) shows a scheduled DFG containing three operations {1, 2, 3} with the same type. The constructed network for those operations is shown in Figure 7(b). Note that operations 1 and 2 are compatible but not FR-compatible so there exists edge $(1_{Pout}, 2_{Pin})$ but no $(1_{FRout}, 2_{FRin})$. In addition, operations 2 and 3 are both compatible and FR-compatible so both edges $(2_{Pout}, 3_{Pin})$ and $(2_{FRout}, 3_{FRin})$ exist.

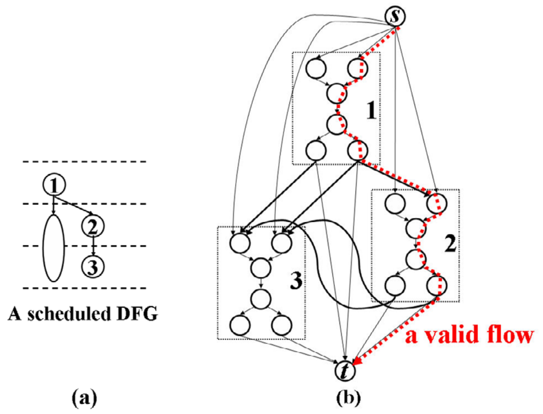

Figure 7: Network construction.

3.2 Obtaining Binding from Network Flow Solution

If the resource constraint of resource type f is N_f, we solve the min-cost N_f-flow problem on the constructed network H. Then we use the solution (network flows) to perform binding and firewall register insertion simultaneously, which will be discussed in this section.

All the operations visited by a flow will be bound to a single functional unit with or without a firewall register. Precisely, if a flow goes through edge (u_{FRout}, w_{FRin}), operations u and w will be bound together with a firewall register. If a flow goes through edge (u_{Pout}, w_{Pin}), operations u and w will be bound without a firewall register. However, if the condition occurs that a flow go through edge (u_{FRout}, x_{FRin}) and then (x_{Pout}, w_{Pin}), i.e., operations u and x are bound with a firewall register but operations x and w not, we cannot decide whether the functional unit associated with operations u, x, and w is protected by a firewall register. To avoid such a situation, for each operation u we require that edges (u_{FRin}, u_{Xin}) and (u_{Xout}, u_{FRout}) have the same flow and also edges (u_{Pin}, u_{Xin}) and (u_{Xout}, u_{Pout}) have the same flow. With these constraints, we guarantee that a flow always stays at either the primitive part or the firewall register part of edges. We call a unit flow satisfying the above constraints a valid flow. For example, in Figure 7(b) the highlighted path indicates a valid flow.

After applying a min-cost N_f-flow algorithm to the network, we can derive a set of valid flows, and then construct the corresponding binding result. Since we use the power as the cost in the network, a min-cost algorithm leads to a binding solution with the lowest power.

3.3 Solving Network Flow Problem with Equal Integral Flow Constraints

As research [6] mentioned, the min-cost flow can be solved by the shortest path based algorithm [1]. However, different from the general characteristics of networks, network H in our formulation requires that the flows on certain edges are equal. The min-cost flow problem with equal integral flow constraints is a difficult problem (NP-hard) [2]. To trade off solution quality with runtime, we can use heuristic algorithms, such as the one presented in [3], where the authors used a Lagrangian relaxation technique to speed up the min-cost equal-flow problem.

4. Scheduling with Firewall Register Insertion Support

In this section we discuss how to perform scheduling to avoid the conditions in Lemma 1. Our idea can be briefly illustrated with the example in Figure 5, which shows a data hazard on dataflow (*u*, *v*). If we can schedule the two operations *u* and *v* in a nonconsecutive way, like in Figure 8, Lemma 1 will never be satisfied. We define the *slack* of an edge (*u*, *v*) as the distance between operations *u* and *v* in terms of c-step. If the slack is zero, operations *u* and *v* are executed at consecutive c-steps; if the slack is a positive value, the two operations are separated by at least one c-step. Our goal is to assign positive slacks to many edges to avoid the situation in Lemma 1.

The problem formulation is as follows.

Given: (1) A DFG *G*; (2) A latency constraint *T* in number of c-steps and a set of optional scheduling constraints, including data dependency, throughput, and relative timing [9].

Goal: Generate a scheduled DFG *G'* without violating *T* and all the given scheduling constraints; in the meantime, the number of dataflows (or edges in *G'*) with hazards is minimized.

Since the assignments of slacks are constrained by the overall latency constraint, we have to intelligently budget and distribute time slacks to the non-critical edges of the given DFG. This problem is traditionally called the timing budgeting problem. The previous research [10] has well studied this problem and provided an optimal solution.

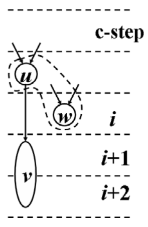

Figure 8: The use of slack for avoiding a hazard.

5. Experimental Results

We incorporated our scheduling and binding techniques into the behavioral synthesis tool, xPilot, introduced in [8]. In this section we will compare the power efficiency of the RTL designs generated by the conventional behavioral synthesis [8] and by our firewall-register-supporting (FR-supporting) behavioral synthesis.

The experimental flow is as follows. We first performed both the behavioral synthesis methods under the same resource and timing constraints. Next, we implemented each RTL design into a real FPGA device using Xilinx ISE in version 8.1.03i. The target FPGA is mainly device XC2V500 in Xilinx's Virtex-II family while we use XC2V1500 for benchmark CHEM due to its large size. All multiplications are implemented using the dedicated multiplier blocks of an FPGA device, and the target clock period is 15ns. After deriving the post-place-and-route implementations, we randomly simulated them to obtain switching activities and then used xPower [23] to compute a circuit's dynamic power. We have to emphasize that the power and area reported in the experimental results are extracted from the post-place-and-route implementations in order to reflect the real situations.

We used a set of data intensive benchmarks to test our methods. The experimental results are shown in Table 1. Column 1 presents the name of a benchmark. Columns 2 and 3 show the resource constraints of adders/subtractors (ADD/SUB) and multipliers (MUL), respectively. Here the resource constraints are 20% of the total number of the operations in a DFG. Columns 4 to 6 show the results from the conventional synthesis flow presented in [8]. Columns 7 to 9 present the results from the conventional flow with firewall register insertion; i.e., we still use the conventional scheduling and binding algorithms but additionally insert firewall registers. Finally Columns 10 to 12 show the results from our FR-supporting flow.

Let us consider benchmark DIF as an example. The RTL from the conventional flow physically needs 788 flip-flops and 987 slices. Note that the number of flip-flops includes those in slices and dedicated multiplier blocks. The dynamic power is 174mW. After the insertion of firewall registers at the RTL, the flip-flop usage increases to 852 and the slice usage increases to 988, but the power decreases to 155. In other words, with an 8% increase of flip-flops and a 0.1% increase of slices, the power can be decreased by 11%. Furthermore, if we apply the FR-supporting flow to generate an RTL, with a 16% increase of flip-flops and a 4% increase of slices, the power can be reduced by 28%.

On average, the conventional flow with firewall registers achieves a 16% reduction of dynamic power while introducing a 1% increase of slices (area overhead). This shows that for those designs, the insertion of firewall registers can effectively reduce the dynamic power. In addition, on average the FR-supporting flow achieves a 28% reduction of dynamic power while introducing a 4% increase of slices. This shows that our FR-supporting scheduling and binding algorithms can further enhance the insertion of firewall registers, thus leading to larger power reduction.

Our method is based on the assumption that a firewall register must be implemented by local flip-flops; otherwise, no power can be saved when glitches still propagate through programmable interconnect with large capacitance. Note that we can do nothing in high level synthesis to control the placement of firewall registers while this is controlled by FPGA placer. Fortunately, using Xilinx ISE placer in the experiments, we checked the layouts of some designs and found that all firewall registers are implemented as local flip-flops by this tool. Secondly, according to the experimental results, the use of firewall registers does not increase the usage of slices much, suggesting that the firewall registers are implemented in local flip-flops rather than occupying spare slices. The slice increase is due to additional control circuit for firewall registers. We believe that other synthesis tools should produce the same results since the use of local flip-flops is good for delay, power, and routing congestion.

Table 1: Experimental results.

Design	ADD /SUB	MUL	Conventional flow [8]			Conventional flow [8] with firewall register insertion			FR-supporting flow		
			FF	Slice	Power (mW)	FF	Slice	Power (mW)	FF	Slice	Power (mW)
ARAI	6	1	676	833	146	772	828	128	788	919	119
DIF	6	2	788	987	174	852	988	155	916	1029	126
DIT	7	3	932	1214	199	1012	1173	202	1076	1296	140
LEE	6	4	1028	1126	174	1092	1132	158	1172	1156	115
MCM	13	6	1652	2085	241	1956	2261	243	1940	2225	220
WANG	5	4	836	998	204	916	1025	144	980	1065	133
CHEM	33	33	5076	6063	719	5892	6126	553	5716	5330	556
DIR	11	12	1732	2091	275	2084	2156	195	2100	2152	160
HONDA	9	10	1364	1768	223	1668	1741	175	1668	1764	166
PR	5	3	548	859	137	612	872	101	676	850	86
Avg.			1	1	1	1.13	1.01	0.84	1.18	1.04	0.72

Note that the use of firewall registers would not adversely impact the timing of a design. Firstly, after inserting a firewall register to a functional unit, because the firewall register is implemented in local flip-flops with tiny interconnect capacitance, this causes shorter critical paths within the functional unit than those in the original design. Therefore, no setup time violation can occur under the use of firewall registers. Secondly, because there is no combinational logic between the firewall register and the original registers, hold time violation may occur in this place. This issue can be automatically handled by the synthesis tools, which will route wires or add buffers to increase the propagation delay.

We do not show leakage power here because the FPGA chip we used has no "turn-off" mechanism to shut down the leakage of unused components. Therefore, the leakage power is a constant in every result. However, we think the leakage overhead should be small in our method considering that the leakage is roughly proportional to the area.

6. Conclusions

In this paper we propose the concept of firewall registers to block the propagation of glitches on boundary output signals. To resolve the WAR hazard problem caused by the insertion of firewall registers, we also propose an FR-supporting behavioral synthesis flow. The experimental results show that the reduction in dynamic power is around 28%.

Acknowledgements

This work is partially supported by the National Science Foundation grant CCR 0096383 and the SRC GRC contract 2006-TJ-1400 0306682.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood, Cliffs, 1993.

[2] R. K. Ahuja, J. B. Orlin, G. M. Sechi, and P. Zuddas, "Algorithms for the simple equal flow problem," in *Management Science*, pp. 1440-1455, 1999.

[3] A. I. Ali, J. Kennington, and B. Shetty, "The equal flow problem," in *European Journal of Operational Research*, pp. 107-115, 1988.

[4] J. M. Chang and M. Pedram, "Register allocation and binding for low power," in *Proc. of Design Automation Conf.*, pp. 29-35, 1995.

[5] J. M. Chang and M. Pedram, "Module assignment for low power," in *Proc. of Conf. on European Design Automation*, pp. 376-381, 1996.

[6] D. Chen, J. Cong, and J. Xu, "Optimal simultaneous module and multivoltage assignment for low power," in *ACM Trans. on Design Automation of Electronic Systems*, vol. 11, Issue 2, pp. 362-386, April 2006..

[7] J.A. Clarke, A.A. Gaffar, and G.A. Constantinides, "Parameterized logic power consumption models for FPGA-based arithmetic," in *Proc. of International Conference on Field Programmable Logic and Applications*, pp. 626-629, Aug. 2005.

[8] J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang, "Platform-based behavior-level and system-level synthesis," in *Proc. of IEEE International SOC Conference*, pp. 199-202, 2006.

[9] J. Cong and Z. Zhang, "An efficient and versatile scheduling algorithm based on SDC formulation," in *Proc. of Design Automation Conference*, pp. 433-438, July, 2006.

[10] J. Cong, W. Jiang, and Z. Zhang, "Scheduling with integer time budgeting for low-power optimization," in *Proc. of ASP-DAC*, Jan. 21-24, 2008.

[11] C. Y. Huang, Y. S. Chen, Y. L. Lin, and Y. C. Hsu, "Data path allocation based on bipartite weighted matching," in *Proc of Design Automation Conference*, pp. 499-504, June 24-27, 1990.

[12] P.E. Landman and J.M. Rabaey, "Architectural power analysis: the dual bit type method," in *IEEE Trans. on VLSI Systems*, pp. 173-187, 1995.

[13] J. Leijten, J. van Meerbergen, and J. Jess, "Analysis and reduction of glitches in synchronous networks," in *Proc. of European conference on Design and Test*, pp. 398-403, 1995.

[14] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient FPGAs," in *Proc. of ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, California, pp. 175-184, February 2003.

[15] H. Lim, K. Lee, Y. Cho, and N. Chang, "Flip-flop insertion with shifted-phase clocks for FPGA power reduction," in *Proc. of the IEEE/ACM International Conference on Computer-aided Design*, San Jose, CA, pp. 335-342, 2005.

[16] C. G. Lyuh and K. Taewhan, "High-level synthesis for low-power based on network flow method," in *IEEE Trans. on VLSI Systems*, pp. 364-375, 2003.

[17] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proc. of the IEEE/ACM International Conference on Computer-aided Design*, pp. 398-402, 1993.

[18] N. Rollins and M. J. Wirthlin, "Reducing energy in FPGA multipliers through glitch reduction," in *Proc. of 7th Annual International Conference on Military Applications of Programmable Logic Devices (MAPLD '05)*, Washington, DC, USA, September 2005.

[19] L. Shang and N.K. Jha, "High-level power modeling of CPLDs and FPGAs," in *Proc. of ICCD*, pp. 46-51, 2001.

[20] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex-II FPGA family," in *Proc. of the International Symposium on Field Programmable Gate Arrays*, pp. 157-164, February 2002.

[21] S.A. Wadekar and A.C. Parker, "Interconnect-based system-level energy and power prediction to guide architecture exploration," in *IEEE Trans. on VLSI Systems*, pp. 373-380, 2004.

[22] S. J. E. Wilton, S-S. Ang, and W. Luk, "The impact of pipelining on energy per operation in Field-Programmable Gate Arrays," in *Proc. of International Conference on Field-Programmable Logic and its Applications*, Antwerp, Belgium, pp. 719-728, August 2004.

[23] Xilinx Website, http://www.xilinx.com.