

Power Management for FPGAs: Power- Driven Design Partitioning

Rajarshi Mukherjee and Seda Ogrenci Memik

Department of Electrical and Computer Engineering, Northwestern University
{rajarshi, seda}@ece.northwestern.edu

1. INTRODUCTION

Power optimization is becoming an increasingly important design aspect for FPGAs. In order to enable efficient integration of FPGAs into cost effective and reliable high-performance systems as well potentially into low power mobile systems, their power efficiency needs to be improved. In this paper, we propose a power management scheme for FPGAs centered on a power-driven partitioning technique. Our power-driven partitioner creates clusters within a design such that within individual clusters, power consumption can be improved via voltage scaling. Our aim is to identify subgraphs/partitions in a design, such that the total power consumption is minimized while resource constraints associated with the partitioning problem are satisfied. The delay on the critical path of the design is unchanged during this process. Hence, performance metrics such as latency and throughput for the overall design are not degraded.

We present algorithms to perform power-driven partitioning and formulate two applications within this domain: chip-level power management for multi-FPGA systems and single chip power optimization using localized voltage scaling. We tested the effectiveness of our approach on a set of LUT-level benchmark netlists. Savings in power consumption with our approach are 19.18% (as high as 32.28) and 14.78 % on average (as high as 25.79%) when partitioning is performed under resource constraints. If the number of partitions is not restricted, power improvements as high as 54 % are achievable.

1.1 Modeling of Potential Timing Relaxation

Given a DAG representation of a design, the longest path from any of the primary inputs to any of the primary outputs defines the longest combinational path, i.e., the critical path. Logic blocks that reside on the critical path are called *critical nodes*. Any increase of the delay of those nodes would result in an increase of the critical path length. Taking the length of the critical path as our timing constraint and assuming that all input signal arrive at the same time, we can assign *arrival* and *required* times for each node. The difference between the required time and the arrival time of a node is called *time slack*. Formally, time slack of a node k is defined as $time\ slack(k) = required(k) - arrival(k)$. This entity will be equal to 0 for critical nodes, while it takes a positive value for non-critical nodes. The time slack available on a node tells us by how much a node can be made slower which in turn dictates by what percentage we can scale down the voltage supply of that node.

2. POWER MANAGEMENT USING VOLTAGE SCALING

It is well known that the dynamic power reduces by the square of the supply voltage ($P = C_{load} V_{dd}^2 f_{switch}$), and the delay increases linearly ($D = C_{load} / V_{dd}$) as the supply voltage is decreased. Hence, it is possible to perform a tradeoff between power consumption and performance by changing the supply voltage.

2.1 Our Power-driven Partitioning Algorithm

Our algorithm takes in a LUT – level netlist. We assume the netlist is a *DAG*. Further we assume each LUT has a delay of 1 unit when

operating at full supply voltage level. Our algorithm tries to identify clusters of nodes along a path, which can share the time slack available along that path. We first find the slack values on the nodes and form an initial set of clusters assuming the availability of arbitrary scaling factors. Then, in a refinement phase the voltage scaling factors of the clusters are adjusted according to the available supply voltage levels in the target hardware.

First, the nodes of the netlist are topologically sorted and the critical path in a netlist is calculated starting from the input nodes. Next, we compute the slack of each node based on (ALAP-ASAP) values. Initially the number of partitions is zero and none of the nodes belong to any partition. The algorithm takes as input the feasible scale factor SC_FAC – the minimum amount by which voltage can be scaled. *CreateCluster* procedure selects the nodes based on ascending *ASAP* schedule thereby selecting nodes from the input level and then going towards the output. If the node has zero slack it is added to the non-scaled partition. After choosing a non-zero slack node v , which is not already added to any partition, it is checked if

$$SC_FAC > \frac{longest_path_in_partition}{longest_path_in_partition + slack}$$

If this condition is satisfied, a new cluster is created and the algorithm tries to grow the cluster. Iteratively minimum slack fanout nodes are selected (not already in another cluster) and added to the cluster if it is feasible to add a new node and the slack of the path is updated as the minimum of the slacks of the nodes in the path. Let us call this entity $slack^{Path}$. Similarly, the length of the longest path in the partition is updated.

Once we stop adding any more nodes to a cluster c_i , we will have the following information about this cluster:

M_i (number of nodes along the longest path in c_i), L_i (length of the longest path in c_i), $slack^{Path}$ (available slack along the longest path in c_i . Without violating any overall timing constraints of the circuit, we can slow down the longest path in this cluster by $slack^{Path}$ time units.), $node_delay$ (amount of time by which each individual node in c_i can be delayed). We can formally express $node_delay$ as follows:

$$node_delay = \frac{slack^{Path}}{M_i}$$

Since the same voltage scaling factor will be applied to all nodes within a partition, the slowdown of each node will be same. $node_delay$ represents the amount of this slowdown.

Given the parameters described above, the voltage supply for this cluster could be scaled down by the factor of

$$scale^{cluster_i} = \frac{L_i}{L_i + slack^{Path}}$$

As a certain amount of slack is now dedicated to increase delay of nodes in the formed cluster, the latest start time as well as the available time slack of their transitive¹ fanin nodes must be updated.

¹ Node v is the transitive fanin of node q if there is a direct path from v to q . Similarly; node v is the transitive fanout of a node q , if there is a direct path from q to v .

Similarly all the transitive fanout nodes associated with the nodes in the cluster are updated. The *CreateCluster* procedure is repeated until all nodes in the input circuit are assigned to a cluster.

Now we have clusters each having a particular slack^{Path} value and corresponding voltage scaling factor $scale^{cluster}$. Depending on the particular constraints of our target system, we might need to further optimize this initial partition. The two foremost constraints are related to resource availability and connectivity. For a multi-FPGA system we might be limited with a pre-determined number of FPGA devices. In the case of the single FPGA system, due to the overhead of creating voltage islands on a chip, we would need to further optimize the initial partition in order to reduce the total number of different voltage islands as well as the interconnection cost between clusters. We devised post-processing techniques to comply with such resource constraints and/or to optimize the cut cost of the final partition. We refer to these schemes as Delay Based Merging and Connectivity Based Merging. Delay Based Merging tries to satisfy the resource constraint while searching for the best possible merging in terms of power gain. Connectivity Based Merging puts higher emphasis on reducing the cut cost of the final partitioning result.

After completing the assignment of clusters to FPGAs, the total power consumption now becomes P_{scale} . If the total power consumption without any voltage scaling was P , the ratio is given by

$$\frac{P_{scale}}{P} = \frac{(V_{dd_scale})^2 \times k + (n - k)}{n}$$

where, n is the total number of nodes in the circuit and k is the number of nodes in the voltage scaled partition.

For an r -way partition into r FPGA devices, we take the same approach. Assume that the partition sizes are $k_1, k_2, k_3, \dots, k_r$. The ratio of the scaled power consumption to the non-scaled power consumption is given by

$$\frac{P_{scale}}{P} = \frac{(V_{scale1})^2 \times k_1 + (V_{scale2})^2 \times k_2 + \dots + (V_{scaler})^2 \times k_r}{n}$$

3. EXPERIMENTS

The experiments are formed on a set of eight combinational benchmarks from the ISCAS 85 benchmark suite [1]. We used Synplify Pro tool from Synplicity to synthesize the Verilog netlists of the ISCAS85 benchmarks onto netlists of LUTs. The power-driven partitioner reads in an EDIF file and produces a partition for N FPGAs in a multi-FPGA system or N voltage islands on a single FPGA. The initial delay of each LUT is assumed to be 1ns. We further assumed that voltage scaling is done in increments of 0.08 Volts. Finally, we assume that for the multi-FPGA scenario, all FPGA devices have the same capacity. Our techniques are independent of the actual values of these parameters.

As described in Section 2.1, after the *CreateCluster* procedure we perform an initial refining, where we round off voltage scaling values according to the smallest scaling step. Next we perform our post processing heuristics. Table 3 presents our results using the Delay Based Merging strategy. We tested three scenarios, where the number of clusters is restricted to be 2, 4, or 8. We report the cut cost, i.e., the number of inter partition connections as well as the percentage power improvement obtained. We report similar results for the Connectivity Based Merging strategy in Table 4. We observe that while we obtain better power improvement using Delay Based Merging the cut cost is consistently inferior to the Connectivity Based Merging. Symmetrically, Connectivity Based Merging always yields better cut cost at the expense of reduced power improvement. Hence, the possible trade-off between the cut cost objective and power

improvement is evident. The average cut cost obtained using Delay Based Merging is 569.11 for partitioning into 8 partitions, while it is 482.44 after using Connectivity Based Merging. On the other hand, the percentage power improvement is 19.18 % for Delay Based Merging whereas it has dropped to 14.78 % for the Connectivity Based Merging.

Table 3. Results after Delay Based Merging heuristic.

Circuit Name	Delay Based Merging of Clusters					
	2 way		4 way		8 way	
	Cost	% Imp	Cost	% Imp	Cost	% Imp
c1355	144	10.60	148	10.60	168	12.90
c1908	194	23.73	206	24.54	260	26.48
c2670	414	13.32	462	16.88	488	19.08
c3540	652	8.46	664	8.97	712	11.72
c432	226	17.24	236	21.35	262	32.28
c499	196	12.08	214	13.36	302	17.69
c5315	984	4.26	1120	9.26	1156	10.24
c6288	1154	4.65	1190	5.52	1384	12.46
c880	268	11.40	342	22.80	390	29.80
Maximum	1154	23.73	1190	24.54	1384	32.28
Average	470.2	11.75	509.11	14.81	569.11	19.18

Table 4. Results after applying the Connectivity Based Merging heuristic.

Circuit Name	Connectivity based Merging of Clusters					
	2 way		4 way		8 way	
	Cost	% Imp	Cost	% Imp	Cost	% Imp
c1355	144	10.61	148	10.73	154	11.09
c1908	194	23.74	198	24.48	206	25.44
c2670	414	13.32	424	13.88	432	18.98
c3540	652	8.46	656	8.96	664	9.88
c432	14	64.51	200	23.59	210	25.79
c499	196	12.08	204	12.36	220	13.05
c5315	984	4.26	992	5.65	1010	7.18
c6288	1154	4.65	1158	4.99	1166	5.45
c880	268	11.42	274	12.99	280	16.17
Maximum	1154	64.51	1158	24.48	1158	25.79
Average	447	17.01	472.67	13.07	482.44	14.78

4. CONCLUSIONS

In this paper, we presented a power optimization technique for FPGA-based systems. Our proposed approach aims to create opportunities for voltage scaling by grouping nodes in a LUT-level netlist into partitions. Our power management approach yields significant improvements in power consumption ranging from 38.22% to 14.78% on average for different resource constraints.

5. REFERENCES

- [1] I. Brglez, F., D. Bryan, and K. Kozminski. *Combinational Profiles of Sequential Benchmark Circuits*. in *International Symposium on Circuits and Systems*. 1989.