Activity Packing in FPGAs for Leakage Power Reduction

Hassan Hassan, Mohab Anis, Antoine El Daher, Mohamed Elmasry

▶ **To cite this version:**

Hassan Hassan, Mohab Anis, Antoine El Daher, Mohamed Elmasry. Activity Packing in FPGAs for Leakage Power Reduction. DATE'05, Mar 2005, Munich, Germany. pp.212-217. hal-00181517

HAL Id: hal-00181517
https://hal.science/hal-00181517

Submitted on 24 Oct 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Activity Packing in FPGAs for Leakage Power Reduction

Hassan Hassan Mohab Anis Antoine El Daher Mohamed Elmasry
VLSI Research Group
University of Waterloo, Waterloo, ON N2L 3G1, Canada

Abstract

In this paper, two packing algorithms for the detection of activity profiles in MTCMOS-based FPGA structures are proposed for leakage power mitigation. The first algorithm is a connection-based packing technique by which the proximity of the logic blocks is accounted for, and the second algorithm is a logic-based packing approach by which the weighted Hamming distance between the blocks activities is considered. After both algorithms are analyzed, they are applied to a number of FGPA benchmarks for verification. Once the activity profiles are realized, sleep transistors are carefully positioned to contain the clustered blocks that share similar activity profiles. Finally, the percentage of the leakage power savings for each of the two algorithms is evaluated.

1. Introduction and Related Work

The scaling of the CMOS technology has precipitated an exponential increase in subthreshold leakage currents. Therefore, it is not surprising that leakage power now constitutes a high percentage of the total chip power. In FPGA applications, the management of leakage power has been overshadowed by performance improvement and dynamic power minimization techniques. As modern FPGAs are getting implemented in 90nm CMOS technology, solving the leakage power problem is pivotal to devising power-aware FPGAs. For the FPGA industry to continue competing with high-performance custom VLSI designs in the semiconductor market, or to explore new territories such as wireless personal communication systems (PCSs), the industry must invest in novel techniques to control leakage power dissipation. Although, FPGAs provide flexibility in design, they are not fully exploited. In fact, the logic and switching resources utilization are approximately only 60% and 50%, respectively, of the total FPGA resources. As a result, unutilized parts of FPGAs cost designers a large amount of inactive leakage power without providing any gainful output. Thus, these unutilized resources should be investigated to achieve minimal leakage power. In addition, it is helpful to recall that even the utilized blocks dissipate inactive leakage power during their standby modes.

One technique that has become increasingly popular for mitigating inactive leakage power is employing high-V_{th} (HVT) sleep transistors (STs) to cut off a low-V_{th} circuit from the power rails during the standby mode. When the ST is turned off, the circuit leakage is limited to that of the ST. In this technique, the sizing of the ST also impacts the amount of speed loss in the active mode because of the added resistance to ground. STs should be able to support the peak current requirements of the logic clusters that the STs control so that the speed penalty does not exceed 5%. Therefore, by selecting the appropriate ST size, the speed penalty, leakage power, and area overhead can be minimized for the entire circuit. Since the peak currents of the clustered logic blocks control the circuit speed, a method by which the kind and number of blocks are chosen to be clustered to share one ST is crucial. Ideally, any number of gates can be grouped with one ST, as long as their switching periods are mutually exclusive. However, for gates with simultaneous switching, the number of clustered gates must be limited to ensure that the speed penalty does not exceed 5%.

Over the past few years, a number of ST sizing methodologies have been reported in the literature. In [1] and [2], a single ST was proposed to support the whole circuit. The ST was sized according to the mutual exclusive discharge patterns in [1]. Also, a distributed ST network methodology was suggested in [3] to minimize the total ST area. None of the techniques in the literature offer automated methodologies to cluster the logic blocks [1, 2, 3]. Either they were all based on intuition, or on supporting the entire circuit with a global ST. In addition, none of these techniques accounted for the routing overhead for clustering purposes, which is a critical issue in nanometer designs. In addition, [4] and [5] discussed techniques that employed STs, but did not address how and which gates should be chosen to share an ST. From the FPGA perspective, a leakage control technique which employed region-constrained placement of STs was developed in [6]. Again, there was no description of how clusters were created.

Consequently, in this paper two new packing algorithms for detecting which blocks in FPGA structures exhibit switching correlations, i.e., can be clustered in one activity profile. A flow chart for the new design flow is shown in Figure 1. The first algorithm is a connection-based packing technique, where the proximity of FPGA blocks is considered, whereas the second algorithm is a logic-based packing approach where the Hamming Distance between the activity of the different blocks is utilized. Both algorithms are analyzed and applied to a number of FGPA benchmarks for validity. Once the activity profiles are known, STs are connected to contain these clustered blocks sharing similar activity profiles. This connection is performed in the configuration stage of the FPGA. The leakage power saving is finally evaluated for each of the two algorithms. It is important to note that this clustering (packing) stage occurs before the placement stage in FPGAs. Consequently, by applying the two algorithms and knowing which

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

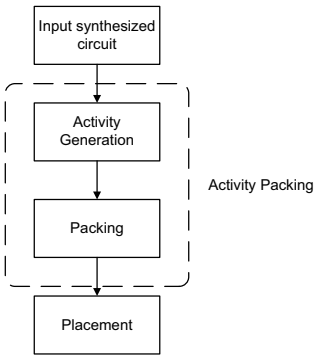

Figure 1. Proposed design flow.

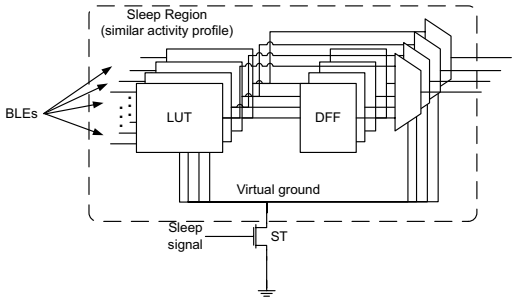

Figure 2. Sleep region architecture.

gates should be clustered, the placer ensures that they would be placed close to each other to minimize the routing overhead. The FPGA fabric is thus divided into regions of similar activity, each of which is independently controlled though a local ST. Furthermore, latches are inserted between one MTCMOS circuit and another to ensure that the logic is retained during the standby mode (when the virtual ground rails float). These latches are inherently located in the FPGA BLEs, and are adopted as the interface between every two MTCMOS blocks. Thus, no extra logic is required to perform this interface to guarantee data retention. By applying these techniques, the blocks which are unutilized are connected to a common ST and permanently turned *off* during the configuration of the FPGA. On the other hand, utilized blocks that display similar switching activities (i.e., the same activity profile) will be grouped together to dynamically and collectively turn them *on* or *off*. In order to limit the overhead of the reconfiguration control circuitry, the ST should not change state so frequently.

2. Targeted FPGA Architecture

The targeted FPGA architecture is shown in Figure 2. Each BLE consists of a 4-input LUT, flip-flop and 2:1 multiplexer. Several BLEs are grouped together to form CLBs. Inside the CLBs, the BLEs are connected together using the local switching resources. In addition, the CLBs are connected using the global routing resources of the FPGA. Every n CLBs are connected to the ground via a High-V_T (HVT) NMOS transistor to reduce leakage current and force the n CLBs into low-power modes during their inactive periods. The HVT sleep transistor is controlled using a SLEEP signal at its gate. Moreover, in each CLB, the latches are used to retain the value of the BLEs outputs when they enter they sleep mode. The several CLBs served by one sleep transistor are called the sleep region. The size of the sleep region, i.e., n is controlled by many factors; maximum allowable size for the sleep transistor, hence the maximum peak current this transistor will hold, the maximum performance deterioration due to the sleep transistor allowed, as well as the maximum permitted ground bounce in the virtual supply lines.

3. Connection-Based Activity Packing Algorithm (CAP)

The CAP algorithm involves assigning similar activity labels to the BLEs that are expected to have similar activity profiles. Afterwards, the BLEs are clustered to minimize the delays along the critical paths by applying simulated annealing. The CAP algorithm consists of two phases: activity generation and clustering.

3.1 Activity Generation

The aim for the activity generation phase of CAP is to give BLEs that share nets, similar activity labels. The main reasoning for this approach is that BLEs that share inputs are expected to be active at the same time. Moreover, cascaded BLEs are more likely to have similar activity labels because as the output of the driving BLE change, the driven gate is expected to change state. Hence, this approach assumes a 100% probability of change in the output of a BLE when one of its inputs change state, thus giving the pessimistic results for the activity labeling.

The algorithm begins with the circuit primary inputs and greedily allocates activity regions as it traverses the circuit netlist by means of simple depth-first graph search algorithm. The result is a fast and computationally efficient algorithm. While traversing the circuit netlist, whenever a new BLE is encountered, it is necessary to determine whether to add this BLE to the current activity region, or to place it in a new activity region. There are two principal driving costs that need to be considered at each node: the total number of activity regions and the size of the activity region.

The number of activity regions corresponds physically to the total number of sleep signals employed in the design. Increasing the number of activity regions results in increasing the number of sleep signals used, thus causing a power-inefficient implementation, as well as complicating the control circuitry for generating these signals. Hence, the total number of activity regions needs to be minimized as much as possible.

The other driving cost function for the activity generation algorithm is the size of the activity region. Reducing the size of the activity region provides the clustering algorithm with more flexibility to pack only those BLEs that manifest the same activity, not those that have close activity profiles. Although this leads to a greater leakage savings, the disadvantages of a large number of activity regions once again become an issue. Furthermore, the algorithm must be expansive while each BLE is processed. The addition of any BLE to the current activity region, implies the addition of all of its fan-in and fan-out BLEs, because the algorithm is connection-based. As a result of this, the number of fan-ins and fan-outs of any BLE, should be considered during the process. Consequently, the cost of adding the current BLE to the current activity region is

IEEE COMPUTER SOCIETY

expressed as

$$cost_1 = \frac{currCap + \alpha \times lev_b + (1-\alpha) \times lev_a - maxCap}{maxCap} , \quad (1)$$

where $maxCap$ is the predefined maximum capacity for the activity region, $currCap$ is the current capacity of the activity region, lev_b and lev_a are the minimum number of unlabeled logic levels from the BLE to the primary inputs and outputs, respectively, and α is a weighting constant that is used to signify the logic levels either before or after the BLE, respectively, hence, improve the quality of the final solution. The use of lev_a and lev_b provides the cost function with the ability to look around the current BLE to examine what other BLEs are expected to be attracted to the current activity region when the BLE under investigation is placed in it.

By running the algorithm on several benchmarks, it is found that a value for $maxCap$ of 1.5 times the longest path from input to output in the circuit provides the best results in terms of power savings. Giving a constant value for $maxCap$, irrespective of the circuit size, results in impractical results. Moreover, increasing $maxCap$ than 1.5 times the longest path in the circuit results in having excessively large activity regions that are usually not fully filled up by the algorithm. On the other hand, decreasing $maxCap$ increases the number of activity regions in the final design.

Another cost function is maintained to represent the attraction between the BLE and the activity region under consideration, and is expressed as

$$cost_2 = m , \quad (2)$$

where m is the number of nets that connect the current sleep region to the BLE under consideration. Hence, the decision of whether or not a certain BLE should be placed in the current activity region is given by:

$cost_1 + \delta \times cost_2 \leqslant 0 \Rightarrow$ add to the current activity region

$cost_1 + \delta \times cost_2 \geqslant 0 \Rightarrow$ start a new current activity region

where δ is a normalization factor. The values of the α and δ are determined by exhaustively trying several values and checking the quality of the solution. In our experiments, a value of 0.5 is selected for α. Again, the value of δ controls which of the cost functions, $cost_1$ or $cost_2$, should be given higher priority. A value of -0.1 is adopted for δ in this work, and it proved to produce good results. The reason for choosing a negative value is that the value of $cost_1$ is negative, unless the activity region size constraint is violated.

The activity generation phase consists of two stages: exploration and labeling. In the exploration stage, the netlist is converted into a directed graph and traversed by the depth-first search algorithm. While each node is traversed, two labels are added to it; the number of levels and paths from this node to any of the primary inputs and the primary outputs. In the second stage, the graph is traversed by the depth-first search approach and the cost of adding each of the nodes connected to it to the current activity region ($cost_1$) is computed. Then the cost of not adding them to the activity region ($cost_2$) is computed. The node with the minimum $cost_1$ is selected as the candidate node, and then compared to its $cost_2$, and a decision is made. This continues until each node in the graph is labeled with its activity region. The pseudocode for the activity generation phase of CAP is given in Figure 3.

```
Create a directed graph from the netlist
Traverse the graph using DFS
for each node
   Calculate levels_before
   Calculate levels_after
end for
Traverse the graph using DFS
for each node i
   for each node j connected to i
      calculate cost_1
      calculate cost_2
      if cost_1 ⩽ min_cost_1
         min_cost_1 = cost_1
         min_cost_2 = cost_2
         min_node = j
      end if
   end for
   if min_cost_1 = δ × min_cost_2
      add to current activity region
   else
      start a new region
end for
```

Figure 3. CAP pseudocode.

Figure 4 depicts an example of the activity generation phase of CAP for a maximum activity region size of four. Figure 4 indicates that the algorithm begins with node A and then studies its child D, and adds it to the activity region. Following that, the logic blocks connected to D; B and E are examined. B is added to the activity region because it has the minimum C_2. Afterwards, E is added to the activity region. A new activity region is started from F because the sum of C_1 and δC_2 is positive. Lastly, C is added to the second activity region.

3.2 Packing Phase

The packing is performed by employing simulated annealing technique on the resulting circuit netlist which consists of the input netlist plus an activity number for each CLB. The packing algorithm follows these hard constraints while solving the optimization problem: (i) the number of BLEs must be less than the cluster size, (ii) the number of inputs needed by the BLEs inside the cluster must be less than the number of cluster inputs, (iii) all the BLEs inside a cluster must have the same activity profile, and (iv) the peak current in the cluster does not surpass the maximum allowable current of the ST for a speed penalty of 5%. In addition, the packing algorithm follows the same objective function as that of T-VPack [7]. The objective is to minimize the number of inter-cluster connections that lie on the critical path of the circuit. In addition, simulated annealing is applied to the optimization problem, unlike [7]. The reason for using simulated annealing is to speed up the solution of the packing problem since the problem here is more complex than the one in [7] due to the addition of the new hard constraints (iii) and (iv).

3.3 Experimental Results

The CAP algorithm is implemented and tested on several benchmarks to assess its capability of using connectivity to generate the activity of the circuit, as well as the power savings due to the use of STs. The experiments are performed on a 900MHz Ultra Sparc III machine with 8Gbytes RAM, and the results are summarized in Table 1. The third column in Table 1 lists the number of resulting clusters and the minimum FPGA array that can be used to map the circuit.

IEEE COMPUTER SOCIETY

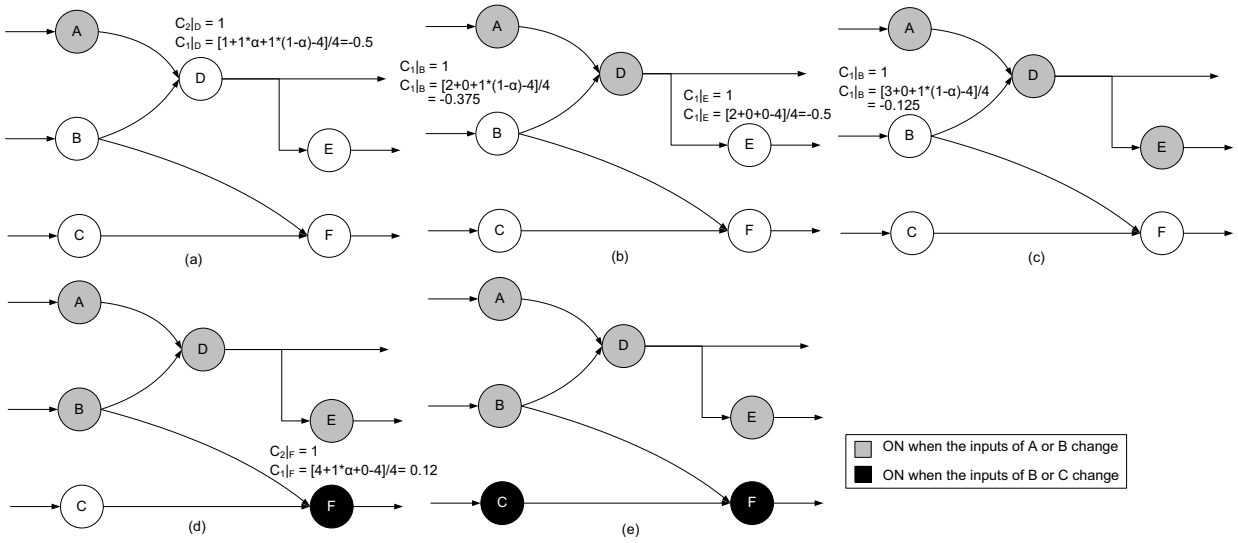

Figure 4. CAP activity generation flow for $\alpha = 0.5$ and $\delta = -0.1$

The power dissipated by each design is calculated using the power model developed in [8]. In each benchmark, the power savings consist of two parts; savings from permanently turning *off* all the unused cluster and savings from dynamically turning *on* and *off* the different used clusters in the design during operation depending on their activity profile. When the unused clusters are turned *off*, their standby leakage power dissipation is reduced significantly because of the presence of the sleep transistor in the leakage path. Thus this part of power saving is calculated by merely subtracting the standby leakage for each cluster with and without a sleep transistor and multiplying it by the total number of unused clusters in the design. Moreover, to calculate the savings due to the dynamic switching of the used clusters, the logic power dissipation per cluster P_d is calculated by

$$P_d = t_{on} \times P_{dyn} + t_{off} \times P_{leak} \ , \qquad (3)$$

where t_{on} and t_{off} are the percentage of times the cluster is either *on* or *off*, respectively, and P_{dyn} and P_{leak} are the dynamic and standby leakage power dissipation of the cluster. The active leakage dissipation of each cluster is ignored in the case when sleep transistors are used with respect to the dynamic power dissipation. The power savings is thus the difference between P_d and the logic power dissipation without using sleep transistors.

From the results in Table 1, it can be deduced that the CAP algorithm can be used to achieve an average power saving of 22.5%. Moreover, the minimum power saving that is attained is more than 10%, except for the s1269 benchmark which has high switching characteristics, thus resolving the power savings to merely that of turning *off* the unused part of the FPGA. In addition, the results for cm150a and cm163a denotes that the power savings in these two cases results only from turning *off* the used parts of the FPGA during their idle state, as there are no utilized parts in the FPGA. Furthermore, the execution time of the packing algorithm is almost linear with the circuit size, except for sequential circuits that have long cycles which complicates their processing, due to the use of simple depth-first search algorithms in traversing the netlist graph.

Table 1. CAP algorithm results with 5% delay penalty.

Circuit	# of BLEs	# of Clusters	% Saving in Leakage	Execution Time (s)	% of Unused CLBs
cm82a	6	2 (2×2)	42.84	0.01	50
cm151a	9	3 (2×2)	37.31	0.005	25
cm150a	16	4 (2×2)	50.07	0.01	0
cm163a	16	4 (2×2)	22.38	0.02	0
cm162a	19	5 (3×3)	23.14	0.02	44.4
cm85a	24	6 (3×3)	21.98	0.01	33.3
s400	162	41 (7×7)	13.61	30.3	16.32
s991	519	130 (12×12)	11.27	1336.18	9.72
s1269	569	145 (13×13)	4.63	12.84	14.2
s1494	647	162 (13×13)	10.13	875.43	4.14
s1488	653	164 (13×13)	10.27	456.49	2.96

This grouping is similar to the worst-case grouping because the algorithm does not incorporates the actual logic function of the circuit and assumes that when the inputs to the BLE change, its output will change, which is not true in all cases. Incorporating the logic function of the BLE can actually result in a better grouping, but can be computationally expensive. That's why the Logic-based Activity Packing (LAP) algorithm is proposed.

4. Logic-Based Activity Packing (LAP)

The LAP algorithm depends on the representation of the activities as binary sequences. The packing is then performed by grouping those BLEs that have similar activity sequences, i.e., minimum Hamming distance between the activity vectors. For LAP, the circuit topology for the activity-based packing is ignored and instead the circuit logic function is used to find the optimum clustering that prolongs the *off* periods of each CLB. This is achieved by exhaustively simulating all the input combinations of the circuit to generate the activity vectors. In order to properly explain this algorithm, several definitions and notations will be first explained.

4.1 Activity Vector

Definition 1: Activity Vector
Given a net x in a circuit netlist, the *activity vector* A_x of x is

IEEE COMPUTER SOCIETY

defined as follows:

$$A_x = [\ a_1 \quad a_2 \quad a_3 \quad .... \quad a_{2^n-1} \quad a_{2^n}]^T \quad , \qquad (4)$$

where n is the total number of inputs to the circuit, a_i is a binary variable that is '1' if any of the outputs of the circuit depend on net x for evaluation when the inputs to the circuit are given by the i^{th} input vector, and T represents the transpose of the vector.

In FPGAs, each BLE has only one output; thus, the activity vector of each net resolves to be the activity vector of the BLE driving that net. Hence, A_x is the activity vector of net x, as well as the BLE called X, where x is the output of X.

Example 1: For the circuit in Figure 5, blocks F and G must be *on* to generate the outputs of the circuit f and g, respectively. Consequently, the activity vectors A_f and A_g for blocks F and G, respectively, are given by

$$A_f = [\ 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1\ ]^T \ ,$$
$$A_g = [\ 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1\ ]^T \ . \qquad (5)$$

On the other hand, for computing the activity vector at the inputs

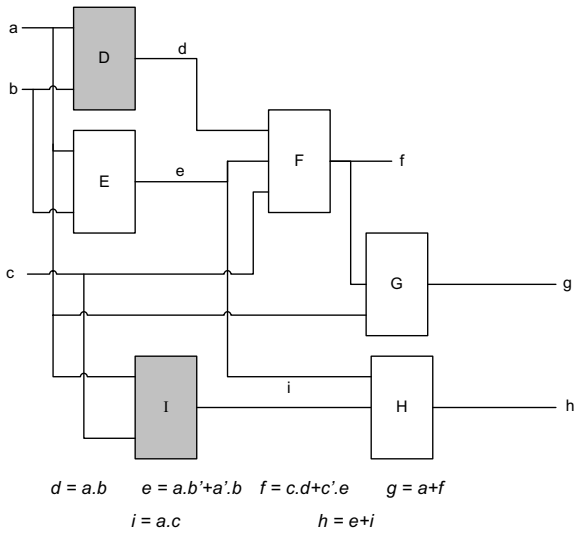

Figure 5. An example of a circuit.

of block F, it is noteworthy that block D will be only used to generate the output signal f if the input c is '1'. Similarly, block E is only used when c is '0'. Hence, the activity vectors for D and E, when f is evaluated, are represented by

$$A_d = [\ 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1\ ]^T \ ,$$
$$A_e|_f = [\ 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0\ ]^T \ . \qquad (6)$$

However, to evaluate h, E will have the following activity vector:

$$A_e|_h = [\ 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0\ ]^T \ , \qquad (7)$$

which differs from the one given in (6). Hence, the resulting A_e is given by

$$A_e = A_e|_1 + A_e|_2 = [\ 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0\ ]^T \ .$$

Furthermore, the activity vector for i will be given by

$$A_i = [\ 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1\ ]^T \ . \qquad (8)$$

From this discussion, it can be deduced that if F, G, and H are active for all the input combinations, packing them together will result in improved results. Moreover, E will be active for almost all of the input combinations except for only one, thus it can also be packed with F, G, and H in the same cluster. Therefore, the cluster containing E, F, G, and H will be always *on*. On the other hand, D and I have similar activity profiles for half of the input combinations, thus it will be a good strategy to group them together and turn *off* this cluster for half of the circuit operational time.

4.2 Hamming Distance and Weighted Hamming Distance

Definition 2: Hamming Distance
Given two binary sequences of length n; A_n and B_n, the *Hamming distance* $d_{(a,b)}$ between these two sequences is defined as

$$d_{(a,b)} = \sum_{k=0}^{n-1} |a_k - b_k| \ , \qquad (9)$$

where a_k and b_k are the k^{th} elements of A_n and B_n, respectively.

From (9), the Hamming distances between the activity vectors given in (5) to (8) are written as

$$\begin{array}{lll} d_{(f,g)} = 0 & d_{(f,d)} = 4 & d_{(f,e)} = 1 \\ d_{(f,i)} = 4 & d_{(f,h)} = 0 & d_{(g,d)} = 4 \\ d_{(g,e)} = 1 & d_{(g,i)} = 4 & d_{(g,h)} = 0 \\ d_{(e,d)} = 5 & d_{(e,i)} = 5 & d_{(e,h)} = 1 \\ d_{(d,i)} = 4 & d_{(d,h)} = 4 & d_{(i,h)} = 4 \end{array} \qquad (10)$$

From (10), it is seen that the Hamming distance between the activity vectors of any two CLBs is a measure of the correlation between the activity profiles of the CLBs. A Hamming distance close to the absolute minimum of zero, indicates that the two blocks will exhibit the same activity profile, thus when positioned together in the same cluster will result in maximum power saving. On the other hand, a Hamming distance close to the absolute maximum of n, denotes that the two clusters have distant activity profiles, and will be power-inefficient if these two BLEs are grouped together. This is verified by examining the values in (10) and the results stated in the previous sub-section.

The Hamming distance between the activity vectors of two CLBs does not take into consideration the probability of occurrence of the different input combinations. As a result, the quality of the results can be notably affected, especially for large circuits.

Definition 3: Weighted Hamming Distance
Given two binary sequences of length n; A_n and B_n, and a weighting vector W_n, the *weighted Hamming distance* $dw_{(a,b)}$ between these two sequences is defined as

$$dw_{(a,b)} = \sum_{k=0}^{n-1} w_k \times |a_k - b_k| \ , \qquad (11)$$

where a_k, b_k, and w_k are the k^{th} elements of A_n, B_n, and W_n, respectively.

IEEE COMPUTER SOCIETY

The weighting Hamming distance is an efficient way to incorporate the various probabilities of the input combinations into the algorithm. Hence, in this work, the weighted Hamming distance is used to group the different BLEs into activity regions.

4.3 The LAP Algorithm Operation

The LAP algorithm consists of two main phases: activity vector generation and packing. The activity generation phase exhaustively simulates the circuit by iterating all the input vectors and finding the values of all the circuit nets resulting from that input vector. Moreover, for each input vector iteration, each signal (or block) is tested to investigate whether or not the output will be affected. This is achieved by complementing the value of the signal under consideration and then proceeding from that point to the circuit outputs. If any of the circuit outputs change their state due to changing the signal value, this means that the net (or block) under consideration is needed in order to generate the output and a '1' is placed in the corresponding row of the activity vector. Otherwise, a '0' is inserted in the activity vector.

After exhaustively generating all the vectors as well as activity vectors for all the circuit nets, the static probability of each net is calculated. This is calculated directly from the exhaustive simulation performed in the first stage.

The clustering phase has the same hard-constraints as those of the CAP algorithm. The clustering phase starts with any BLE and then inserts it into a new cluster. In addition, BLEs are added to the current cluster in a greedy manner based on their minimum weighted Hamming distance to the blocks currently in the cluster. Thus, the block with the minimum weighted Hamming distance is added to the cluster, until the cluster is full. Afterwards, a new cluster is created and a seed BLE is selected and added and the same procedure is repeated until all the BLEs are clustered. A pseudocode of the algorithm is listed in Figure 6.

```
for all the input combinations
   for all the nets in the circuit
      find the value of the net
   end for
   for each net in the circuit
      toggle the value of the net
      Activity[input_vector][net] = 0
      proceed with the new value of the net
      if the value of any output changes
         Activity[input_vector][net] = 1
      end if
   end for
end for
while there are unpacked BLEs
   if current cluster has space
      find the BLE with minimum HD to
      the current cluster
      add this BLE to the current cluster
   else
      start a new cluster
   end if
end while
```

Figure 6. Pseudocode of LAP.

4.4 Experimental Results

Although the LAP algorithm produces very accurate results, yet it is computationally expensive. In order to test its efficiency, LAP is allowed to pack several FPGA benchmarks, and the results are listed in Table 2. The algorithm is run on an 900MHz Ultra Sparc III processor with 5Gbyte RAM. From the results in Table 2, it can be deduced that LAP can achieve an average *off* time of 33% for the benchmarks investigated. Moreover, the LAP algorithm has an exponential complexity with the number of primary inputs to the circuit.

Table 2. LAP algorithm results with 5% delay penalty.

Circuit	# of BLEs	# of inputs	# of clusters	Avg OFF time %	Execution time (s)	% Savings in Power
cm82a	6	5	3	32.3	0.01	45.4
sec	10	5	4	24.2	0.02	12.9
x2	12	10	4	17.75	0.81	9.3
alu2	59	10	16	43.9	50.7	15.3
cm85a	24	11	7	13.4	1.74	23.1
cm151a	9	12	3	45.8	0.74	38.9
cm162a	19	14	6	20.5	13.49	23.7
cu	23	14	7	27.7	26.71	18.5
pm1	48	15	9	29.2	157.82	12.4
cmb	14	16	5	57.72	38.1	27.3
cm163a	16	16	5	40	42.35	24.1
cm150a	16	21	5	48.75	1277.86	51.6

5. Conclusion

This work presented two packing techniques for MTCMOS-based FPGAs architectures for leakage power minimization. The two algorithms suggest average standby leakage reduction of 15% for the FPGA benchmarks tested. The LAP algorithm, although is more accurate in predicting the activity, and hence, the packing of the circuit, than the CAP algorithm, yet its computational time is exponential with respect to the circuit inputs.

References

[1] M. Khellah *et al.*, "Power Minimization of High-Performance Submicron CMOS Circuits using a Dual-V_{dd} Dual-V_{th} (DVDV) Approach," *in Proc. ISLPED*, pp. 106–108, 1999.

[2] S. Mutoh *et al.*, "1-V Power Supply High-Speed Digital Circuit Technology With Multithreshold-Voltage CMOS," *IEEE J. Solid-State Circ.*, vol. 30, no. 8, pp. 847–854, Aug. 1995.

[3] C. Long and L. He, "Distributed Sleep Transistor Network for Power Reduction," *in Proc. DAC*, pp. 181–186, 2003.

[4] J. Tschanz *et al.*, "Dynamic Sleep Transistor and Body Bias for Active Leakage Power Control of Microprocessors," *IEEE J. Solid-State Circ.*, vol. 38, no. 11, pp. 1838–1845, Nov. 2003.

[5] A. Abddollahi *et al.*, "Precomputation-Based Guarding for Dynamic and Leakage Power Reduction," *in Proc. ICCD*, pp. 90–97, 2003.

[6] A. Gayasen *et al.*, "Reducing leakage energy in fpgas using region-constrained placement," *in Proc. International Symposium on FPGAs*, pp. 51–58, 2004.

[7] A. Marquardt *et al.*, "Using cluster-based logic blocks and timing-driven packing to improve fpga speed and density," *in Proc. International Symposium on FPGAs*, pp. 37–46, 1999.

[8] K. W. Poon *et al.*, "A Flexible Power Model for FPGAs," *in Proc. international conference on Field-Programmable Logic and Applications*, pp. 312–321, 2002.

IEEE COMPUTER SOCIETY