Peak Dynamic Power Estimation of FPGA-mapped Digital Designs

Abstract*— The *Peak Dynamic Power Estimation* ($PDPE$) problem involves finding input vector pairs that cause maximum power dissipation (maximum toggles) in circuits. The $PDPE$ problem is essential for analyzing the reliability and performance of digital circuits prior to fabrication. This paper proposes a methodology for solving the PDPE problem on circuits mapped onto Field Programmable Gate Arrays (FPGAs). An FPGA-mapped circuit comprises of a collection of Look Up Tables (LUTs) connected by interconnects. Hence, the input to the proposed algorithm is an LUT-level netlist (similar to gate-level netlists that are generated in the ASIC design flow). To the best of our knowledge, this is the first such technique reported in the literature for the $PDPE$ on LUT-level netists. The proposed methodology was experimented on the LUT-level netlists of ISCAS'85 combinational benchmark circuits. A maximum toggle estimate improvement of 32.05% is observed when compared to a random estimation method on the same. The paper also presents interesting observations on the non-correlation between optimizations at the gate level and the LUT level netlists. These suggest that *low-power design techniques applied at higher levels of design abstractions need not necessarily result in a design that is power aware at the LUT level.

I. Introduction

With the advent of portable and high-density microelectronic devices, excessive power dissipation is a problem of extreme propensity. The continuing decrease in feature size, increase in chip density and clock frequency in recent years have invigorated concerns about excessive power dissipation in modern VLSI chips. High power dissipation may lead to drops in performance or in extreme cases cause burnouts and damage to circuits. Peak power dissipation of a circuit determines the thermal and electrical limits of components and system packaging requirements [2]. Faster Times-To-Market (TTM) and expensive redesign cycles necessitate accurate and efficient power estimation at an early design phase. The peak power consumption corresponds to the highest switching activity generated in the circuit during one clock cycle. The energy per clock cycle (peak power) in the combinational portion of a circuit can be computed as:

$$P_R = \frac{V_{dd}^2}{2 \times frequency} \times \sum_{\forall\ g} [toggles(g) \times C(g)] \quad (1)$$

where, the summation is performed over all gates g, $toggles(g)$ is the number of times gate g has switched from 0 to 1 or vice versa within a given clock cycle, $C(g)$ is the output capacitance of gate g and V_{dd} is the supply voltage. This work assumes that the output capacitance for each gate is equal to the number of fanouts. Therefore, the total switching activity (toggles) is the parameter that needs to be maximized for maximum P_R. Accurate estimations involve finding a pair of input vectors which when applied successively to the circuit maximize the value of P_R among all possible input vector pairs. Given that the circuit has n primary inputs, there are 4^n possible input vector pairs to be considered for an exhaustive search. Thus, the search space for the vector pairs is huge even for reasonably large values of n. Similar to what is attempted in the case of gate level netlists [3], this paper also assumes the following abridged definition of the $PDPE$ problem: *Given a LUT-level description L of the input circuit and an initial input vector V_0, find an input vector V_1 such that, the vector pair V_0 and V_1 when applied in sequence, lead to maximum switching activity in L.* This reduces the search space to 2^n.

II. Previous Work

The $PDPE$ problem is well studied and reported for gate-level netlists. In [4], the problem of worst-case power computation was transformed to a weighted max-satisfiability problem. Its limitations were constrained scalability and no provision for delay incorporation. In [6] and [7], switching time windows were used with partial input enumerations for correlation resolution. Symbolic transition counts was introduced in [8]. Automatic Test Generation (ATG) based techniques have also been proposed in the literature [9]. Inherently, they are limited by their lack of adaptation to handle delay parameters. In [4] and [5] a test generation strategy was devised for finding test patterns that would produce the maximum power. The technique proposed in [4], takes exponential times with respect to the number of levels in the circuit and hence lacks scalability. Static timing analysis was used in [5] to find the time instants at which the gates can switch and this information was used to maximize energy dissipation in a clock cycle. However, this approach requires complete and specific information about the circuit and has complexity proportional to the number of gates and fan-in. Hence, it would take a large computation time to create a valid sequence. A controllability based approach is reported for the $PDPE$ problem in [3]. A detailed survey of the methods reported in the literature for the $PDPE$ problem is presented in [3]. However, all the methods reported above are for the gate-level netlists. This paper proposes a methodology for solving the PDPE problem on circuits mapped onto Field Programmable Gate Arrays (FPGAs). An FPGA-mapped circuit comprises of a collection of Look Up Tables (LUTs) connected by interconnects. An FPGA-mapped circuit comprises of a collection of LUTs connected by interconnects which are generated based on a gate level or behavioral level description of the circuit. The $PDPE$ estimates on the circuit can hence be an early gate level (description) estimate or a late LUT level (post technology mapping) estimate. The LUT level estimate is important when a design is realized on the FPGA, as it reflects the dynamic power dissipated by the circuit on the field. The input to the proposed algorithm is an LUT-level netlist (similar to gate-level netlists that are generated in the ASIC design flow). A k-input LUT is a $2^k \times 1$ memory with 2^k bits capable of storing the truth-table of any k-input Boolean function. Modern FPGAs have several such LUTs which are programmed to realize different k-input functions as required. To the best of our knowledge, this is the first such technique reported in the literature for the $PDPE$ on LUT-level netists. The proposed methodology was experimented on the LUT-level netlists of ISCAS'85 combinational benchmark circuits. A maximum toggle estimate improvement of 32.05% is observed when compared to a random estimation method on the same. The paper also presents interesting observations on the non-correlation between optimizations at the gate level and the LUT level netlists. These suggest that *low-power design techniques applied at higher levels of design abstractions need not necessarily result in a design that is power aware at the LUT level*.

The rest of this paper is organized as follows. Section III details the proposed algorithm. Section IV presents the experimental results. Section V concludes the paper.

III. $PDPE$ PROBLEM ON LUT-LEVEL NETLISTS

This section presents the proposed methodology for solving the $PDPE$ problem on LUT-level netlists. The solution is based on the traditional D-algorithm for test vector generation [1]. The proposed algorithm is similar to what is presented in [3], but deals with LUTs instead of gate-level netlists. The crucial step is the calculation of controllability values for the LUTs in the given input netlist. The controllability values thus calculated guides the D-algorithm. As mentioned earlier, the methodology presented in this paper identifies an initial vector V_0 and computes another vector V_1, such that applying V_0 and V_1 are in order on the given LUT-level netlist L shall cause maximum toggles on the interconnects of L. Following are the important steps in the proposed algorithm.

Algorithm (*LUT_Power_Virus*)
begin

1) *Initial Input Vector (V_0) Identification and Implication*: The selection of the initial input vector and the values assigned to the primary outputs for justification play a role in affecting the performance (both speed and quality) of the technique. In the proposed method, random vector pairs are generated till every pair of the primary inputs are assigned all possible sixteen combinations of binary values. This is done to account for signal correlations among at least every two primary inputs. Empirically, the number of generated vector pairs needed to satisfy the above property was close to n^2 , where, n is the number of primary inputs of the circuit. Among these n^2 pairs, the pair (P, Q) which yields the maximum switching activity is selected. The first input vector P in this pair is used as the initial vector V_0. The initial vector is applied to the circuit (*implication*) that results in each of the wires in the circuit being assigned a 0 or a 1. This value is denoted as the *old-value* of the wires.
2) *Primary Output Assignment:* The vector Q computed above is applied to the circuit and the values of the primary outputs got by the application of the same are computed. The primary outputs are *assigned* the computed values, while all the other wires are *unassigned*.
3) *Calculation of the Second Input Vector (V_1):* The vector V_1 is calculated using the modified D-algorithm stated in this section, such that on application of the same to the input circuit shall result in
    a) the primary outputs to take the values as calculated in step (2) above; and,
    b) maximize the number of wires that are assigned a value different from the *old-value* computed in step (1) above.

end. (*LUT_Power_Virus*)

It is straightforward to see that the above steps attempt to generate a vector pair (V_0, V_1) such that applying V_0 followed by V_1 in order to the input circuit shall attempt in maximizing the number of toggles in the circuit and hence increase its dynamic power dissipation. The remaining part of this section shall present the modified D-algorithm used by step (3) above.

A. Modified D-Algorithm for LUT Netlists

The algorithm works on the LUT level representation of the circuit and involves many of the D-Algorithm modules discussed in [1]. The algorithm starts with all the LUTs which drive primary outputs as the J-Frontier (*Justification Frontier*). Note that the primary outputs are assigned binary values by the step (2) of Algorithm *LUT_Power_Virus*. During the execution of the modified D-algorithm the J-frontier has all the LUTs whose outputs are *assigned* and one or more of its inputs are *unassigned*, as defined in step (2) of Algorithm *LUT_Power_Virus*.

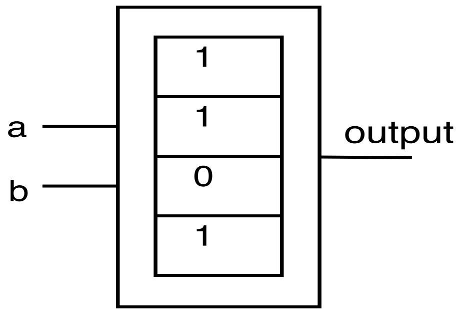

Fig. 1. 2-input LUT

The main step in the modified D-algorithm is to *justify* the LUTs in the J-Frontier. The justification step as follows: To justify (set the value of) the output of the 2-input LUT in the Figure 1 to 0, the inputs (a, b) are set to $(1, 0)$. To justify a value of 1 at the output of the LUT, the inputs may be set to either $(0, 0)$, $(0, 1)$ or $(1, 1)$. Justifying one LUT L may assign values to the outputs of other LUTs driving the inputs of L, hence adding them to the J-Frontier. The above step is repeated till the J-Frontier has no more LUTs. The *justification* step may cause inconsistencies, specifically in the case of *reconvergent fanouts* [1], as illustrated in Figure . In this case, the LUT B is justified for 0 at its output, by assignment of $(1, 1)$ to its input. This results in LUT A entering the J-Frontier with a requirement to justify its output as 1. The algorithm choses $(0, 0)$ as inputs to LUT A. Note that one of the inputs of A is already assigned 1 leading to an inconsistency. This consistency check is performed by the *imply_and_check()* procedure after very justification step. If this procedure returns a consistency check failure, the last decision is *backtracked*. The input assigned to the LUT is marked as *forbidden*. In the Figure 2 the next decision would be to assign $(1, 1)$ to the inputs of LUT A to justify its output with the value 1. This shall not lead to an *imply_and_check* violation. The variable *Back_Threshold* is used to control the number of backtracks at an LUT. If the number of backtracks exceeds this limit on a particular LUT, then of all the inputs tried at the LUT, that *forbidden* input that caused the maximum toggles in the given netlist is assigned as input. This shall cause inconsistency on some of the nets which were earlier assigned to some binary values. The new values are forced on them to resolve the inconsistency. This is done by the procedure *imply_and_force()*.

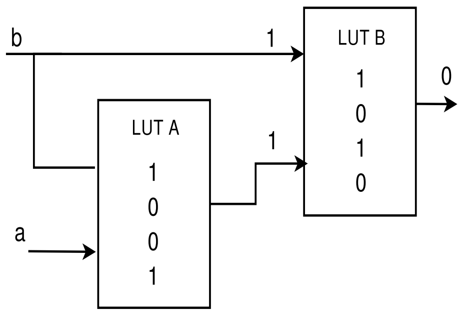

Fig. 2. Inconsistency due to Reconvergent Fanouts

The next important step in the modified D-algorithm is to select the LUTs for justification and justify the same. This is performed by the function *justify_best()*. The following steps explain the working of the function *justify_best()*.

1) Of several LUTs in the J-Frontier, select the next one for justification. The order does matter as justifying one LUT may influence the others due to presence of reconvergent fanouts that may commit some of the inputs of the LUTs in the J-Frontier. A controllability measure introduced later in this paper is used for this purpose. There are two controllability measures defined later, namely, the 0-controllability and 1-controllability. The measures are defined in such a way that if the output of an LUT has a high value of 0- (1)-controllability then it is *easy* to justify 0 (1) on it. Therefore, of all the LUTs L with a need for justifying t on its output, the one with the highest t-controllability value is selected for justification.
2) As seen in Figure 1, there are more than one way for justifying 1 on the output of the LUT. For a given LUT, of all the possible inputs that shall justify the required binary value on its output, the one that is not *forbidden* (already tried) and that which causes the maximum toggle at its input shall be selected. In other words, let the inputs of LUT B in Figure 2 were assigned $(1, 1)$ by the step (1) of Algorithm *LUT_Power_Virus*, that is their *old_value* is $(1, 1)$. Now, the output of LUT B is to be justified with 1. For this there are two choices of inputs, namely, $(0, 0)$ and $(1, 0)$ as seen from Figure 2. The function will choose $(0, 0)$ in this case, as that shall cause two toggles at the inputs of LUT B, while choosing $(1, 0)$ shall cause only one toggle at the inputs of LUT B.

The Algorithm 1 presents the modified D-algorithm. The Algorithm *LUT_Power_Virus* calls Algorithm 1 in Step (3) after initializing the J-Frontier with the LUTs that drive primary outputs.

B. Calculation of Controllability Values

This section defines the controllability values for the interconnects in a given LUT-level netlist. As mentioned earlier, the controllability values guide the modified D-algorithm for selecting the *next LUT* in the J-Frontier for justification. Two controllability values are defined for every interconnect I, namely, the 0- and 1- controllability values, denoted by C_0^I ad C_1^I respectively. The C_0^I (C_1^I) is a measure of the ease with which a line I shall take a value 0(1). The controllability values are in the range $[0..1]$. Higher t-controllability value on an interconnect I implies that it is *more easy* to justify the value t on I. The 0- and 1- controllability values of primary inputs are 1. For an m-input LUT, driven by at least one primary input, the controllability values for its output I is given as follows:

$$C_0^I = \text{number of zeroes stored in the LUT}/2^m \qquad (2)$$

$$C_1^I = \text{number of ones stored in the LUT}/2^m \qquad (3)$$

Note from the above equations that $C_0^I + C_1^I = 1$ for the outputs I of all LUTs that are driven by at least one primary input. For the other LUTs the controllability values are calculated as explained through the following example. Consider a 3-input LUT with wires $N1$, $N2$ and $N3$ as its inputs (in that order) and none of them are primary inputs. Let $N0$ be its output. Assume that the 8-bits stored in the LUT are 01010100. Let C_1^{N1},C_1^{N2} and C_1^{N3} be the one controllability

Algorithm 1 MODIFIEDLUTDALGORITHM

```
1:  MODIFIED LUT D_ALGORITHM()
2:  if Imply_and_check() $\neq$ TRUE then
3:      return false;
4:  end if
5:  if $J$-Frontier = $\phi$ then
6:      return true;
7:  end if
8:  while $J$-Frontier $\neq \phi$ do
9:      justify_best();
10:     Let it return LUT $L$ and inputs $F$ to $L$.
11:     Assign F to $L$ and modify $J$-Frontier;
12:     Let $R$ be the set of new LUTs added to $J$-Frontier;
13:     if Modified LUT D_Algorithm() = true then
14:         return true;
15:     else
16:         if backtrack_counter($L$) = Back_Threshold then
17:             Imply_force()
18:             backtrack_counter($L$) = 0; // reset counter
19:         else
20:             Mark input $F$ as forbidden to $L$;
21:             Remove all LUTS in $R$ from $J$-Frontier
                and Add $L$ to the $J$-Frontier;
22:             Make all inputs of $L$ assigned by the func-
                tion justify_best() as unassigned;
23:         end if
24:     end if
25: end while
```

values and C_0^{N1},C_0^{N2} and C_0^{N3} be the zero controllability values of input wires $N1$, $N2$ and $N3$ respectively. The one/zero controllabilies C_0^{N0}/C_1^{N0} of output $N0$, is calculated as follows:

- **One Controllability:** A one can occur on the output wire $N0$ due to three of the eight possible input combinations - the cases are when $\{N1 = 0, N2 = 0, N3 = 1\}$, $\{N1 = 0, N2 = 1, N3 = 1\}$, or $\{N1 = 1, N2 = 0, N3 = 1\}$. Hence

$$C_1^{N0} = C_0^{N1}C_0^{N2}C_1^{N3} + C_0^{N1}C_1^{N2}C_1^{N3} + C_1^{N1}C_0^{N2}C_1^{N3}$$

- **Zero Controllability:** The zero controllability of $N0$ can be calculated using a similar approach as the that for the one controllability. However, an easier method would be

$$C_0^{N0} = 1 - C_1^{N0}$$

  This follows from the fact that $C_0^{N1} + C_1^{N1} = 1$, and similarly for $N2$ and $N3$. The above equality can be inferred easily using a simple induction with the base case as the controllability values of outputs of LUTs that have at least one input driven by primary inputs.

The same can be easily extended to calculate the controllability values of the output of any general m-input LUT storing any arbitrary 2^m bits.

IV. EXPERIMENTAL RESULTS

The proposed technique was employed on ISCAS'85 combinational benchmark circuits. These circuits were synthesized using the Xilinx ISE synthesis tool to generate a LUT-level netlist. Table I compares the number of toggles estimated by the proposed technique with that obtained by simulating the LUT-level netlist with 10^4 random pairs and selecting the one that yielded the maximum number of toggles. From Table I it is clear that in all the cases, the proposed technique has generated vector pairs that cause more toggles than the ones generated by the random method. In the case of the c2670 circuit the proposed method has computed a vector pair that produces 32.05% more toggles than the ones produced by the random vector pair. The *Back_Threshold* was set to 50 in Algorithm 1. All results reported in this section are for *weighted* toggles which account for the fanout of the switching gates as defined in Equation (1) in Section I.

TABLE I
PDPE USING RANDOM AND CONTROLLABILITY BASED ESTIMATES

Circuit	LUT Ctrl Est LUT Sim	LUT Rand Est Gate Sim	Toggle Count % Improvement
C432	236	184	28.26
C499	147	118	24.58
C880	242	191	26.70
C1355	183	157	16.56
C1908	219	181	20.99
C2670	581	440	32.05
C3540	749	699	07.15
C5315	934	780	19.74
C6288	1352	1344	00.59
C7552	1075	997	07.82

A. Correlation with Gate Level Estimates

The results presented in Table II lead to some interesting observations that are discussed further in the rest of this section. The column 2 of Table II presents the peak toggles on gate-level netlists as reported in [3]. These were got by generating random vector pairs and using that pair that generated the maximum number of toggles. The toggles in column 3 are again reported in [3] that used a controllability-driven method on gate-level netlists. Columns 4 and 5 are the results got by employing the proposed technique on LUT-level netlists and are same as reported in Table I. The vector pair that yielded the maximum number of toggles in the LUT-level netlist as reported in column 4 (random simulation) was input to the corresponding gate-level netlist; the toggles were calculated and the results are reported in column 6 of Table II. Similarly, the vector pair that yielded the maximum number of toggles in the LUT-level netlist as reported in column 5 (method proposed in this paper) was input to the corresponding gate-level netlist; the toggles were calculated and the results are reported in column 7 of Table II. Here are the interesting observations based on the results reported in Table II.

1) *There is no correlation between the peak toggles at the Gate-level and the corresponding LUT-level netlists*:

TABLE II

COMPARISON OF POWER RESULTS FOR ISCAS'85 COMBINATIONAL BENCHMARK CIRCUITS

Circuit	Number of Weighted Toggles					
	Gate Rand Est Gate Sim [3]	Gate Ctrl Est Gate Sim [3]	LUT Rand Est LUT Sim	Lut Ctrl Est LUT Sim	LUT Rand Est Gate Sim	LUT Ctrl Est Gate Sim
C432	201	270	184	236	227	160
C499	272	303	118	147	306	276
C880	437	582	191	242	516	394
C1355	530	610	157	183	583	601
C1908	858	973	181	219	1001	1113
C2670	1332	1516	440	581	1427	1351
C3540	1531	1727	699	749	1947	1579
C5315	2570	3007	780	934	2603	2678
C6288	2558	2684	1344	1352	3411	3352
C7552	3591	3670	997	1075	4426	4240

For example, consider the case of C432 as reported in Table II. The vector pair that generated 184 toggles in the LUT-level netlist (Column 4 of Table II) generated 227 toggles on the corresponding gate-level netlist (Column 6 of Table II), while the vector pair that generated 236 toggles on the LUT-level netlist (Column 5 of Table II) generated only 160 toggles on the gate-level netlist (Column 7 of Table II). Similar is the case for the other circuits too as seen from Table II. This lack of correlation between the gate-level and LUT-level peak power estimates are also apparent from Figure 3. The plots were made by generating 10^3 random vector pairs; applying each of them on both the LUT-level and Gate-level netlists; and counting the toggles at the gate and the LUT-level. In the figure, the sorted LUT values are plotted on the X-axis and the corresponding gate level estimates are on the Y-axis for five benchmark circuits - c432, c880, c1908, c3540, c5315 and c7552. If there existed a correlation between the the two estimates, a regular variation relationship among these would have been observed in the plot. The variation should have been along a straight line. However, there is a rugged dependancy proving the lack of correlation. The results presented in Table III show some interesting anomalies for two ISCAS'85 benchmark circuits, namely, the c1908 and c1355. For the c1908 circuit there are two vector pairs that produce 145 toggles when applied to the LUT-level netlist, but the same vector pairs produce 570 and 1523 toggles respectively when applied on the corresponding gate-level netlist. For the c1355 circuit, there is a vector pair that produces lesser number of toggles on the gate-level netlist than the LUT-level netlist.

2) *LUT level estimates lead to better gate level estimates*: As seen in Table II, for the c6288 and c7552 benchmark circuits the vector pair generated for the LUT-level netlists yielded better number of toggles when applied to the corresponding gate-level netlists than what was reported in [3]. Compare columns 2 and 3 with 6 and 7 respectively in Table II. An intuitive reasoning for the same could be that the LUT level netlists are lesser in size than the corresponding Gate level netlists. This may lead to a better exploration of the search space resulting in better estimates. However, the above reasoning is not valid in general, due to the lack of correlation in power estimation between the gate-level and the corresponding LUT-level netlists.

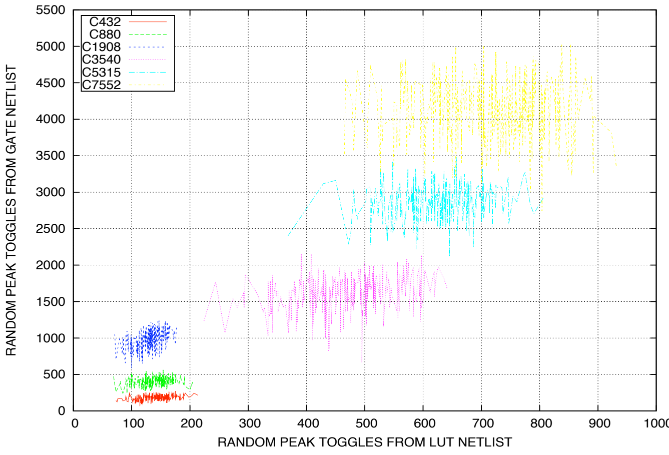

Fig. 3. LUT and gate net toggles for identical random vector inputs.

V. CONCLUSION

In this paper, we have looked at the $PDPE$ problem at the LUT level. For this, we make use of a modified version of a D-Algorithm to guide the toggle maximization, similar to that proposed in [3]. To the best of our knowledge, this work is the first of its kind which addresses the $PDPE$ problem at the LUT level. From our experiments using the proposed controllability based methods, we have shown a maximum improvement of 32.05% in the $PDPE$ toggle estimate over the random approach. Besides the new methodology for $PDPE$, a very interesting conclusion which can be drawn from these experiments is that the $PDPE$ relationship between LUT and Gate level estimates are uncorrelated. An optimized D-Algorithm estimation at the LUT level maximizing the

TABLE III

EXEMPLARY ANOMALIES IN LUT VS GATE LEVEL POWER ESTIMATES

c1908	LUT level	Gate level
V_1=000011000110001011001101011110100 V_2=110111000001111010000001101000100	128	615
V_1=011111101000111111011111010011110 V_2=011011100011110011101100111011000	145	570
V_1=101110000101001010000001111011111 V_2=101000100100111001101000010011100	128	555
V_1=101101101100010110100000011100101 V_2=010010000011011000010010011010110	145	1523
c1355	LUT level	Gate level
V_1=01101110000000000000000000000000000000000 V_2=11101000000000000000000000000000000000000	132	193
V_1=10001110000000000000000000000000000000000 V_2=00111011000000000000000000000000000000000	141	224
V_1=10000110000000000000000000000000000000000 V_2=00001110000000000000000000000000000000000	129	90
V_1=10001111000100000000000000000000000000000 V_2=00111011010011010000000000000000000000000	103	277

wire toggles does not necessarily reflect in a corresponding maximization at the gate level. A possible explanation for this would be that there is hardware abstraction in LUTs which hide gate level details and collapse logic blocks (after optimizations sometimes) to lead to lower number of LUT nets. Hence, an $PDPE$ optimization at the LUT level may not necessarily mean an optimization at the gate level and vice versa. This point is furthered by the illustrations shown in Figure 3. They clearly show an irregular trend and hence a skewed relationship between the gate-level and LUT-level estimates for peak power in digital circuits. From this it can be inferred that *it is not enough to address the power-aware issues for FPGA-targetted designs at the higher levels of abstraction (that includes gate-level), and is necessary to address the same at the technology-mapping stage of the FPGA design flow*.

REFERENCES

[1] M. Abromovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design,* Jaico Publishing House, Mumbai, 2002.

[2] W. Wenzel and K. Hamacher, *Stochastic Tunneling Approach for Global Minimization of Complex Potential Energy Landscapes* Physical Review Letters 82, Issue 15, pp.3003-3007, April 1999.

[3] K. Najeeb, K.Gururaj, V. Kamakoti and V.M Vedula *Controllability-Driven Power Virus Generation for Digital Circuits* Journal of Low Power Electronics, Vol. 3., pp.280-292, 2007.

[4] S. Devadas, K. Keutzer and J. White, *Estimation of Power Dissipation in CMOS Combinational Circuits Using Boolean Function Manipulation* IEEE Trans. on CAD,, Vol.11, No.3, pp.373-383, March 1992.

[5] C.Y Wang and K. Roy, *Maximization of Power Dissipation in Large CMOS Circuits Considering Spurious Transitions* IEEE Tran. on Circuits and Systems, Vol.47, No.4, pp.483-490, Apr 2000.

[6] H Kriplani, *Worst Case Voltage Drops in Power and Ground Buses of CMOS VLSI Circuits* PhD. Thesis, University of Illinois, 1992.

[7] H Kriplani, F. Najm and I. Hajj, *Resolving Signal Correlations for Estimating Maximum Currents in CMOS Combinational Circuits* Proc. of the Design Automation Conference, pp.384-383, 1993.

[8] S. Manne, et.al, *Computing the Maximum Power Cycles of a Sequential Circuit* Proc. of the Design Automation Conference, pp.23-28, 1995.

[9] C.Y Wang, K. Roy and T. Chou, *Maximum Power Estimation for Sequential Circuits Using a Test Generation Based Technique* Proc. of Custom Integrated Circuits Conf. 1996.