Leakage Power Optimization With Dual-V_{th} Library In High-Level Synthesis*

Xiaoyong Tang
Magma Design Automation, Inc
5460 Bayfront Plaza
Santa Clara, CA 95054 USA
xtang@magma-da.com

Hai Zhou
Northwestern University
2145 Sheridan Road
Evanston, IL 60208 USA
haizhou@ece.northwestern.edu

Prith Banerjee
University of Illinois at Chicago
851 South Morgan Street
Chicago, IL 60607 USA
prith@uic.edu

ABSTRACT

In this paper we address the problem of module selection during high-level synthesis. We present a heuristic algorithm for leakage power optimization based on the maximum weight independent set problem. A dual threshold voltage (V_{th}) technique is used to reduce leakage energy consumption in a data flow graph. Experiments are performed on a data-path dominated test suite of six benchmarks. Our approach achieves an average of 70.9% leakage power reduction, which is very close to the optimal results from an Integer Linear Programming approach.

Categories and Subject Descriptors

B.7.2 [Hardware]: INTEGRATED CIRCUITS − Design Aids; J.6 [Computer Applications]: COMPUTER-AIDED ENGINEERING

General Terms

Algorithms, Measurement, Performance

Keywords

Leakage Power, High-Level Synthesis, Dual-V_{th}, Optimization

1. INTRODUCTION

With the development of very deep sub-micron technology, power consumption is playing a more important role in modern chip design [16]. It is therefore necessary to consider power and energy consumption in the early synthesis stages [8][10][11][13][14][19]. Power dissipation in CMOS circuits consists of dynamic power and leakage power. While dynamic power accounts for a majority of power in current technologies, leakage power will become more dominant in the future with technologies below 90 nm [12].

The module selection step during high-level synthesis chooses which module type is to be used for an implementation. In previous work, researchers assumed that modules differ with each other mainly at the gate-level structures. Current technology enables different types of devices to exist on the same chip. Hence with the same gate-level structures and interconnects, module instances may have different speed, power consumption, area, etc. One example is multiple threshold voltage implementations for the same module type. The advanced technology broadens the synthesis space.

In this paper we address the problem of module selection during high-level synthesis. Our approach selects a set of candidate operations in a data flow graph (DFG) to be changed simultaneously with maximum potential to reduce the energy and still leave exploration space for later synthesis from a global view of the input DFG. In the proposed algorithm, the synthesis problem is formulated as the maximum weight independent set (MWIS) problem. The solution is weighed and the objective is to produce decisions with maximum power reductions in total.

This paper is organized as follows: Section 2 summarizes related works. Section 3 describes basic concepts of leakage power consumption. Section 4 presents basic terminologies and formulation of the problem. In Section 5, analysis of the problem and a heuristic algorithm are presented. In Section 6, experimental results are discussed. Conclusions are presented in Section 7.

2. RELATED WORK

Owing to the importance of leakage power for future process technologies, researchers have developed many leakage power reduction techniques such as those of Roy et al [15]. Khouri et al [10] performs high-level synthesis with a dual-V_{th} library for leakage power reduction using a simple heuristic as a guide. In each iteration, for the function units to be considered, the algorithm identifies a single candidate with most potential for reducing leakage power, and replaces the function unit with high-V_{th} designs. It uses a greedy approach, and validates the choice after replacement. Srivastava et al [19] considers simultaneous V_{dd}/V_{th} assignment to minimize total power at the gate level. Ye et al [24] analyzes the stack effects. Input vector control techniques to reduce leakage power are proposed in Bobba et al [3] and Abdollahi et al [1]. Most of the input vector control techniques apply their methodologies with the assumption that the idle time of the unit is long enough so that the reduction in leakage power can compensate the extra dynamic power consumption overhead.

* This work was done at Northwestern University. This work was supported by DARPA under contract no F33615-01-C-1631.

Using a dual-V_{th} technology library, we present an approach to reduce leakage power consumption based on the maximum weight independent set problem for latency-constrained data-dominated circuits. A similar algorithm proposed by Chen and Sarrafzadeh uses MWIS for gate re-sizing of combinational circuits at the gate level [4]. A major difference between the problem formulated in [4] and our problem is that there is no resource-sharing problem related to the gate resizing formulation [4], since each logic gate has its own implementation. In contrast, the module selection problem during high-level synthesis involves changing all relevant operators when their shared module instances are replaced (from low-V_{th} to high-V_{th}). In [10], a method based on iterative improvement is described where one function unit is replaced at a time; this approach is applicable to control intensive graphs. In our approach, we consider data flow graphs with resource sharing information available. Without violating the performance, our algorithm simultaneous changes module instances with a global view of the whole data flow graph. Additionally, input data patterns are considered to get more accurate estimation of leakage power consumption.

3. BASIC CONCEPTS FOR LEAKAGE POWER CONCUMPTION

In this section, some basic concepts for leakage power consumption are introduced. The leakage power model is based on the early Berkeley short-cannel IGFET model BSIM [17] that provides the basic models for a single MOSFET transistor.

When $V_{gs}\approx 0$, the leakage current for an n-type MOSFET transistor is:

$$I_s = \mu_0 C_{ox} (\frac{W}{L}) V_t e^{1.8} e^{\frac{V_{gs}-V_{th}}{nV_t}} (1 - e^{\frac{-V_{ds}}{V_t}}) \quad (1)$$

where μ_0 is the effective carrier mobility, C_{ox} is the effective capacitance, W/L is the ratio of channel width to channel length of the transistor, V_t is the thermal voltage, and n is the sub-threshold slope coefficient[17][10].

The leakage current of a MOSFET transistor is comprised of several factors including sub-threshold leakage, PN junction reverse bias currents, gate oxide tunneling, hot carrier injection into gate oxide, gate induced drain leakage, and punch-through. The latest BSIM4 [23] model provides an accurate transistor level model for calculating leakage from above components, which can be used by some device-level simulators such as HSPICE (level 54) [20] and Berkeley SPICE3f5 [23]. From (1), the leakage current decreases exponentially with the increase of the threshold voltage. With the technology development, it is possible to implement multiple-threshold voltage gates on the same chip. Thus it is possible to replace low-V_{th} module instances on the non-critical path with high-V_{th} module instances to save leakage power without impacting performance. For a logic gate, the leakage current not only depends on the implementation details, but also on the on/off state of each transistor in stacked transistors (Gu et al [7]).

4. PRELIMINARIES AND PROBLEM FORMULATION

In this section, we give basic terminologies and notations. The formulation of the problem and the constraints are presented.

4.1 Preliminaries

A DFG can be represented as a directed acyclic graph D=(V, E). Each node $v\in V$ is an operation node. Each directed edge (u, v) $\in E$ represents the data dependency within the DFG. In a DFG, each node will be executed by some module instance m. We associate the delay of m to the node v, as $d(v)$. Each node v in the DFG is also labeled with the earliest possible starting time $t_{ASAP}(v)$, the latest possible starting time $t_{ALAP}(v)$, and the slack time $s(v)$. For each node v the slack $s(v)$ is defined : $s(v) = t_{ALAP}(v) - t_{ASAP}(v)$. In the process of module selection, this value indicates the maximum delay increase the node can afford by replacing current module with a slower module instance. For resource-unconstrained DFGs, the ASAP and ALAP times for the node are only related to the delay of the resource, data dependency and the timing constraint. By introducing the resource constraint, they also depend on the available resources. For a synthesized graph, we define module instance usage graphs to represent the resource allocation result.

Module Instance Usage Graph (MIUG(m)) := (V_m, E_m): It represents the usage of a module instance. Each node $v \in V_m$ is a node in the DFG bound to module instance m. There is a directed edge $(u,v)\in E_m$ if during the usage, the instance m executes operation node u immediately preceding the execution of node v.

Utilizing the information from the initial synthesized DFG, we introduce a composite constraint graph to calculate new slacks.

Composite Constraint Graph (CCG): = (V, E_c). Node $v\in V$ is a node in the DFG. There is a directed edge (u, v) $\in E_c$ if and only if there is either an edge (u, v) in the original DFG, or an edge (u, v) in any MIUG. We call the two types of edges as the data dependency edge, and the resource constrained edge, respectively.

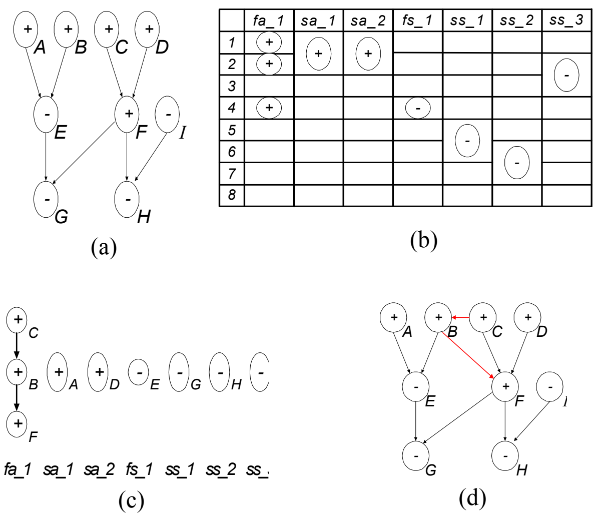

Figure 1. (a) A Sample DFG (b) Example Initial Synthesis Result (c) MIUG (d) Example CCG

Figure 1(a) shows a sample DFG. We assume that there are two slow adders with delay of 2 cycles (sa_1, sa_2), one fast adder with delay of 1 cycle (fa_1), three slow subtractors with delay of 2 cycles (ss_1, ss_2, and ss_3), one fast subtractor with delay of 1 cycle (fs_1). All these instances are implemented in low-V_{th} technology. Assume the latency constraint is 8 cycles, Figure 1(b) shows an initial synthesis result with node B, C, and F shareing one fast adder fa_1. The first column shows the time steps, and the first row shows all module instances. Figure 1(c) shows the module instance usage graphs (MIUG) for each module instance. Figure 1(d) shows the composite constraint graph CCG.

In Figure 1(d), the edges (*C*, *B*) and (*B*, *F*) represent the resource constraint shown in the module instance usage graph of fa_1 in Figure 1(c). The composite constraint graph represents the constraints from data dependency and the resource constraint. It can be easily proved that there is only one such CCG graph for a synthesized data flow graph from the definition of the CCG graph. By running the resource unconstrained ASAP and ALAP algorithms on the CCG graph, we can construct the slack graph (SG) on the composite constraint graph, as shown in Figure 2. The value inside the node is current delay, and the triplet shown next to each node *v* denotes $t_{ASAP}(v)/\,t_{ALAP}(v)/s(v)$.

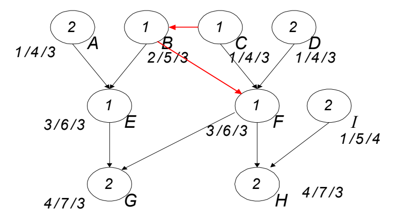

Figure 2. Slack Graph Based On Composite Constraint Graph

From Figure 2, each node has some slack time, which means it can be executed by a slower module instance. Here we only consider the two implementations of each module instance, low-V_{th} and high-V_{th} designs. Low-V_{th} designs usually run faster, but have higher leakage power consumption than high-V_{th} designs.

4.2 Problem Formulation

We define a synthesis result as valid, namely, that it satisfies the latency constraints, if $\forall v$ in the slack graph SG, $s(v)\geq 0$, and it satisfies resource-constraints. Now consider a synthesized DFG implemented in low-V_{th} designs with a given latency constraint and a dual-V_{th} module library that contains low-V_{th} and high-V_{th} designs of the same function unit. Given the leakage power tables as in [21], if we replace a module instance *m* with its high-V_{th} design with lower leakage power consumption, the leakage energy reduction in this replacement would be:

$$\sum_{v\in MIUG(m)} LPWT_{m\_l\_V_{th}}[v]*(t_{idle\_l\_V_{th}}+\alpha_{m\_l\_V_{th}}*D_{m\_l\_V_{th}})$$

$$-\sum_{v\in MIUG(m)} LPWT_{m\_h\_V_{th}}[v]*(t_{idle\_h\_V_{th}}+\alpha_{m\_h\_V_{th}}*D_{m\_h\_V_{th}}) \quad (2)$$

Where MIUG(*m*) is the MIUG of *m*, $LPWT_m[v]$ is the leakage power consumption for the input states of node *v*, t_{idle} is the time when the module instance is in idle states, α_m is the empirical co-efficient, and D_m is the delay for the module instance *m* to finish the computation.

In order to satisfy the latency constraint and resource constraint, not all module instances in the circuits can be replaced with their high-V_{th} correspondents. We only consider a module instance if all nodes in its MIUG have positive slacks. Based on the above discussions, our low leakage power optimization problem is defined as:

Given a synthesized data flow graph, a set of timing constraints, a set of modules from a dual-V_{th} library, replace some (or all) module instances with their corresponding high-V_{th} implementations, such that the data dependency and timing constraints are satisfied, and the total leakage power consumption is minimized.

This problem has some similarities to the multiple voltage-scheduling problems in terms of using different implementations of the module instances to save power. While the authors of [18] assume there are no resource constraints for different implementations, and select one candidate for each iteration, our problem maintains the resource sharing information in the synthesized DFG, and chooses a set of candidates to be changed.

5. A HEURISTIC APPROACH BASED ON SIMULTANEOUS REPLACEMENTS

In this section, we present a heuristic approach to minimize the leakage power consumption from an already scheduled and resource-allocated design. Before describing the details of the algorithm, we discuss the slack calculation under resource constraints, and sensitive edges between nodes and module instances.

5.1 Slack Sensitive Graph and Slack Sensitive Transitive Closure Graph

The slacks in the composite constraint graph give designers the freedom of module replacements. However, the slack change on one node will affect the slack of other nodes due to the data dependency and resource sharing. Following the definition in [4], we define the slack sensitive edge on composite constraint graphs.

On a composite constraint graph, an edge $(u, v)\in E_c$ is called slack sensitive if either $t_{ASAP}(v) - t_{ASAP}(u) = d(u)$ or $t_{ALAP}(v) - t_{ALAP}(u) = d(u)$. A slack sensitive edge implies that the slacks of the two nodes of the edge are sensitive to each other's slack change. We define the slack sensitive graph as follows:

Slack Sensitive Graph (SSG):= (V, E_s). Each node $v\in V$ is a node in the DFG. There is a directed edge $(u, v)\in E_s$ if and only if an edge (u, v) is a slack sensitive edge. Figure 3(a) shows the slack sensitive graph for the composite constraint graph in Figure 1(d).

Slack Sensitive Transitive Closure Graph (SSTG):=(V, E_{st}). Each node $v\in V$ is a node in the DFG. There is a directed edge $(u, v) \in E_{st}$ if there is a directed path from *u* to *v* in SSG. Two nodes $u, v\in V$ are called slack insensitive if there does not exist an edge (u, v) or (v, u) in SSTG. We call a set SI = $\{v_1, v_2, \ldots, v_k\}$ in the DFG as slack insensitive if all nodes in the set are pair-wise slack insensitive [4]. Figure 3(b) shows an example slack sensitive transitive closure graph.

In Figure 3(b), we can find several slack insensitive set, such as $\{E, F, I\}$, $\{A, B, D\}$, $\{G, H\}$, etc.

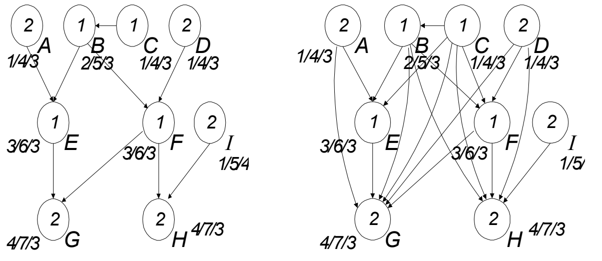

Figure 3. (a) Example Slack Sensitive Graph (SSG); (b) Example Slack Sensitive Transitive Closure Graph (SSTG)

If the nodes are executed exclusively by dedicated resources, our problem becomes a case similar to the problem described in [4]. The authors proved that if there is a complete library, their algorithm can produce optimal power reduction by repeatively finding the maximum weight independent set in the slack sensitive transitive closure graph. In our problem, there is resource sharing between different nodes, and every module instance only has two implementations (low-V_{th} and high-V_{th}) available. The resource sharing accelerates the decrease in slacks. There are no unlimited V_{th} levels for the module instances, and the delay increases for different module instance are discrete. Hence unlike [4], we propose a heuristic solution.

5.2 A Heuristic Approach

We now present a heuristic approach to solve the problem. Instead of using the slack sensitive transitive closure graph, we use a general transitive closure graph. Because the library is not complete, the usage of slack sensitive transitive closure graph may violate the timing constraints. By using the general transitive closure graph, the feasibility can be guaranteed by checking the slack changes for other nodes that have edges with current node.

We assume that an instance replacement is safe in a DFG if after the replacements, the slack for each node is still non-negative. To check the feasibility of each replacement, we assign the nodes with new delay in the module instance graph, and perform resource unconstrained ASAP and ALAP scheduling to calculate the new slacks for all the nodes. If there exists negative slack nodes, the replacement is not safe. We use a flag *RSafe* to indicate the feasibility of replacement. If *RSafe* is TRUE, it means it is safe to replace the module instance with its high-V_{th} design. Otherwise, the module instance cannot be changed. Also we set *RSafe* to FALSE if the module instance is already in its high-V_{th} design.

In order to consider the resource sharing constraints, an undirected module instance sensitive graph (**MISG**) is constructed from the transitive closure graph, with nodes sharing the same module instance being combined into one node. Consider any two combined nodes U and V. Suppose the node U consists of operation $\{u_1, u_2, \dots, u_p\}$, and V consists of $\{v_1, v_2, \dots, v_q\}$. There is an edge (U, V) in the graph if and only if there is an edge (u_i, v_j) in the original transitive graph. In the new graph, each node represents one module instance, and the edges between them imply there are potential slack dependencies between them.

If the replacement is feasible for a module instance m, we assign a leakage power reduction weight to the node in the new graph according to the equation (2); otherwise, the weight on the node is zero.

If the new instance graph MISG is still a transitive closure graph, we can find the maximum-weight-independent-set of the graph as the candidates for replacement in polynomial time. Otherwise, we can use a greedy approach to get a near-maximum weight independent set. After replacements of one set of independent module instances, the slack of operation nodes in original DFG might not become zero, and there might still exist some safe replacements for module instances. The leakage power consumption of the data flow graph will be iteratively reduced. A formal description of our heuristic algorithm is given below:

<u>*Algorithm*</u>: Leakage Power Optimization With Dual-V_{th} Library
<u>*Input*</u>: An initial synthesized data flow graph G, resource allocation table T, timing constraint, a dual-V_{th} module library, leakage power table for each module instance
<u>*Output*</u>: A synthesized data flow graph that satisfies the timing constraint with less leakage power consumption

```
Begin
(1)Construct module instance usage graphs MIUGs from G and T, label each MIUG with RSafe=TRUE to imply the module instance might be feasible to change its implementation.
(2)Construct composite constraint graph CCG
(3)Construct a general transitive closure graph TG from CCG.
(4)Construct a module instance sensitive graph MISG from TG.
(5)Recognition and find a transitive orientation for the MISG [5].
(6)For each module instance with RSafe=TRUE, build slack graph SG from CCG through resource unconstrained ASAP and ALAP algorithms on CCG, and update the safety of the replacement by setting the RSafe value. If there is no safe replacement, return.
(7)For each node U in the MISG with RSafe = TRUE, calculate and assign a weight, according to Equation (2), to node U. For node U with RSafe = FALSE, the weight is set to 0.
(8)Find the maximum weight independent set of MISG if MISG is a transitive graph; otherwise, find a near-maximum weight independent set of MISG using a heuristic approach. The heuristic approach always chooses the independent node with maximum weight first. It constructs three sets in which some special nodes are excluded. They are the maximum weighted node inclusion set, the maximum weighted node exclusion set, and the maximum and second-maximum weighted nodes both exclusion set. The heuristic approach will choose the set with the largest weight to be changed.
(9)Replace the module instances node in the set of (7) with their high-$V_{th}$ designs, and assign their RSafe to be FALSE.
(10)Update the delay of each operation node in G
(11)Go to (6)
End
```

The three sets of step 8 represent the approximate MWIS by using greedy approach while considering multiple candidates to avoid local optimal results in current step.

To illustrate the algorithm, suppose the input is the data flow graph in Figure 1(a), the initial synthesis result shown in Figure 1(b), the latency constraint of 8 cycles, a dual-V_{th} module library, and a leakage power table. First, the module instance usage graphs are constructed, as shown in Figure 1(c), and assigned default

RSafe values of TRUE. Then the composite constraint graph CCG is built, as shown in Figure 1(d). The general transitive closure graph on CCG is built, as shown in Figure 4(a).

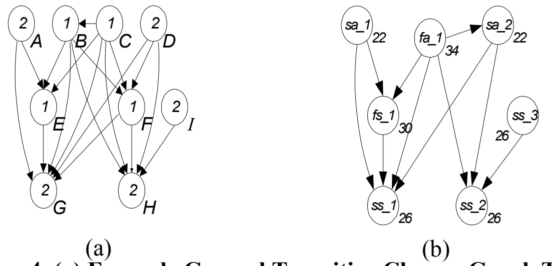

Figure 4. (a) Example General Transitive Closure Graph TG (b) Induced Transitive Graph

The module instance sensitive graph MISG is built next, and a transitive orientation is found, as shown in Figure 4(b). To be illustrative, we assume that the performance penalty for each replacement is 1 cycle. After checking the safety of replacements for each module instance, it shows that every node is safe to be changed. We label the weights for each node in units in Figure 4(b).

The MWIS set of the new transitive graph is {fs_1, sa_2, ss_3}. These module instances are replaced with their high-V_{th} designs. The total leakage power reduction for this pass is 78 units. In the second pass, the *RSafe* of fa_1 is assigned FALSE, because the replacement of this module instance will violate the timing constraint. The module instance sa_1, ss_1 and ss_2 are still feasible for replacements. We can find the MWIS of this pass as {ss_1, ss_2}, and the total weight is 52 units. Finally, the last pass of the algorithm will replace module instance sa_1, and the weight is 22 units. The total leakage power reduction is 152 units. In contrast, the module instances to be replaced by a greedy algorithm are fa_1, fs_1, ss_3, sa_1, sa_2 in that order, and the total leakage power reduction is 134 units.

5.3 Time Complexity Analysis

The time complexity for the heuristic algorithm is $O(|M||V|^3)$, where $|M|$ is the total number of module instances used in the design, and $|V|$ is the number of operation nodes in the DFG. In the algorithm description in section 5.2, the time for the construction of construct MIUG graphs of step 1 is $O(|V|)$. The time for the construction of CCG graph in step 2 is $O(|E|+|V|)$. The time for the construction of transitive graph is $O(|V|(|V|+|E|))$ by performing a breadth first or depth-first search from each node or $O(|V|^3)$ by Warshall's algorithm [22]. MISG is constructed in time $O(|V|+|E|)$. The task of step 5 can be done in time $O(\delta|E|)$[5], where δ is the maximum degree of a node. In the loop from step 6 to step 11, there are $|M|$ iterations in the worst case. In each iteration: the time complexity of step 6 is $O(|M|(|V|+|E|)$, the time complexity of step 7 is $O(|V|)$; the time complexity to find a MWIS is $O(|V|^3)$ if the MWIS is a transitive graph, and $O(|V|)$ for greedy approach; step 8, 9 and 10 can be done in time of $O(|V|)$.

6. EXPERIMENTAL RESULTS

In this section we present experimental results over a suite of six benchmarks written in C. They are Diffeq (Differential Equation), Ellipf (Elliptical Wave Filter), FIR(Finite Impulse Response) filter, Laplace transform, Matrix multiplication, and Sobel transform. The DFGs are generated by a high-level synthesis compiler [9]. The leakage power tables are built for each module instance.

The experiments are performed to optimize leakage power consumption, with and without dual-V_{th} technology libraries, given a set of resource constraints and timing constraints. For comparison, we compare the results from three approaches for different resource binding situations. The first approach is the Integer Linear Programming (ILP) formulations described in [21] with the objective function being the leakage power consumption. The second approach is the greedy algorithm by replacing one instance at a time. The third approach is our heuristic algorithm. We generate the initial synthesis results under three situations using low-V_{th} designs given a set of timing constraints: (1) the minimum area implementation in which case the resources are shared to the maximum degree. These results are generated by the ILP approach with the objective function as the minimum area; (2) the least resource sharing implementation in which each node has its dedicated module instance; (3) the median resource sharing scheme with the number of module instance being the median of case (1) and case (2).

Figure 5, 6, and 7 show results of leakage power consumption using a 0.18-µm 1.8V library from Artisan [2] for three cases: (1) minimum area implementation; (2) minimum resource sharing; (3) median resource sharing, respectively. From the results, we can see that the three algorithms almost get the same optimization results for the minimum area implementation for the benchmarks, as shown in Figure 5. In the minimum area case, the number of instances used is the smallest, which means the instances are highly sensitive to each other in terms of slack change. In the second and third cases, as shown in Figure 6 and Figure 7 respectively, our approach achieves the results very close to the optimal ILP results. For the second and third cases, the ILP approach achieves an average of 75.01% leakage power reduction, and our approach achieves an average of 70.94% leakage power reduction. In comparison, the greedy algorithm only achieves about an average of 58.82% leakage power reduction.

Although ILP approach can guarantee an optimal solution, it is a well-known NP-hard problem and will take exponential time to solve in the worst case. In contrary, our approach runs in polynomial time and is several orders faster than the ILP approach while gives near optimal solutions in all the test cases. For example, in the sobel test bench with median resource sharing, ILP approach using XPRESS on NEOS server [6] costs around 2 hours to find the solution. By using the heuristic approach proposed in this paper on a Sun Ultra-10 Machine with 440MHZ CPU and 256MB physical memory, the average run time is only 3.2 minutes.

Table 1 shows the max, min and average time comparisons on all tests on above machines.

Table 1. Time Comparison for LPILP Approach and MWIS_Heuristic Approach (In minutes)

	Max	Min	Average
LPILP	153.22	7.53	89.62
MWIS_Heuristic	4.11	0.27	1.63

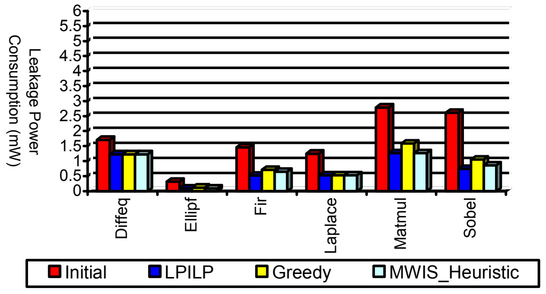

Figure 5. Leakage Power Consumption for Minimum Area Implementation

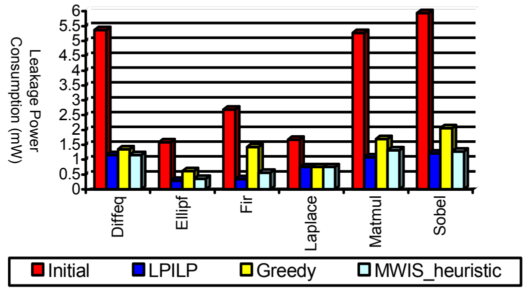

Figure 6. Leakage Power Consumption for Minimum Resource Sharing

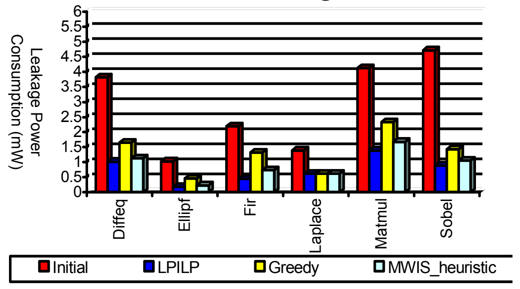

Figure 7. Leakage Power Consumption for Median Resource Sharing

7. CONCLUSIONS

In this paper, we investigated the problem of leakage power optimization using dual-V_{th} technology libraries. We analyzed the time slack dependency among operation nodes and module instances. We proposed a composite constraint graph to explicitly present the resource binding constraints besides data dependency constraints in a data flow graph. We built a module instance sensitive graph to represent the relationship between different module instances. Our approach selects a set of candidates to be changed simultaneously. A maximum or near-maximum weight independent set based heuristic approach is presented to achieve optimized results with maximum or near-maximum potential to reduce the leakage energy and still leaves enough exploration space for later synthesis stages from a global view of the input DFG.

8. REFERENCES

[1] A. Abdollahi, F. Fallah, and M. Pedram, “Leakage Current Reduction in CMOS VLSI Circuits by Input Vector Control,” IEEE TVLSI, pp. 140~154, vol. 12, no. 2, 2004.
[2] Artisan Components Inc. www.artisan.com
[3] S. Bobba, and I. Hajj, “Maximum Leakage Power Estimation for CMOS Circuits,” Proceedings of the IEEE Alessandro Volta Memorial Workshop on Low Power Design, 1999
[4] D. Chen and M. Sarrafzadeh, “An Exact Algorithm for Low Power Library-Specific Gate Re-Sizing,” 25th DAC, 1996.
[5] M. C. Glumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, 1980, New York, USA
[6] Dash Company, XPRESS-MP, www.dashoptimization.com
[7] R.X. Gu and M.I.Elmasry, “Power dissipation analysis and optimization of deep submicron CMOS digital circuits,” IEEE J. Solid-State Circuits, vol.31, pp. 707-713, May 1996
[8] S. Gupta and F. N. Najm, “Power Macromodeling for High Level Power Estimation,” 34th DAC, 1997.
[9] A. Jones, D. Bagchi, S. Pal, X. Tang, A. Choudhary, P. Banerjee, “PACT HDL: A C Compiler with Power and Performance Optimizations,” CASES, France, 2002.
[10] K.S. Khouri, N.K.Jha, “Leakage Power Analysis and Reduction During Behavioral Synthesis,” IEEE TVLSI, 2002
[11] K.S. Khouri, G. Lakshminarayana, and N.K. Jha, “IMPACT: A High-Level Synthesis System for Low Power Control-Flow Intensive Circuits,” DATE, France, 1998.
[12] N.Kim, T.Austin, D.Blauuw, T.Mudge, K.Flautner, J.Hu, M. J.Irwin, M.Kandemir, V.Narayanan, “Leakage Current: Moore’s Law Meets Static Power,” IEEE Comp. v.36, 2003
[13] A. Raghunathan and N.K.Jha, “Behavioral Synthesis for Low Power”, Proceedings of ICCD, 1994.
[14] A. Raghunathan and N.K. Jha, “An ILP formulation for low power based on minimizing switched capacitance during data path allocation,” IEEE Int. Symp. on Cir. & Sys., 1995.
[15] K. Roy, S. Mukhopadhyay, H. M. Meimand, “Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits,” Proc. of the IEEE, vol. 91, No.2, February 2003.
[16] K. Roy, S. Prasad, “Low-Power CMOS VLSI Design,” John Wiley and Sons, Inc., 2000.
[17] B.J.Shen, D.L.Scharfetter, P.K.Ko and M.C.Jeng, “BSIM: Berkeley short-channel IGFET model for MOS transisitors,” IEEE J. Solid-State Circuits, vol.22, pp.558-565, Aug. 1987
[18] W. T. Shiue, C. Chakrabarti, “Low-Power Scheduling with Resources Operating at Multiple Voltages,” IEEE Transactions on Circuits and Systems, vol. 47, No. 6, 2000
[19] A. Srivastava, D. Sylvester, “Minimizing total power by simultaneous Vdd/Vth assignment,” Proc. ASPDAC 2003.
[20] Synopsys Inc. “Star-HSPICE Manual,” www.synopsys.com
[21] X.Tang, “High-Level Synthesis Algorithms for Low Power ASIC Design,” Ph.D. Thesis, 2004.
[22] S. Warshall, A Theorem on Boolean Matrices, Journal of the ACM, 9(1):11-12, 1962
[23] X.Xi, M.Dunga, J.He, W.Liu, Kcao, X.Jin, J. Ou, M.Chan, A.Niknejа, C.Hu, “BSIM4.3.0 MOSFET Model,” http://www-device.eecs.berkeley.edu/~bsim3/bsim4.html
[24] Y. Ye, S. Borkar, and V. De, “A New Technique for Standby Leakage Reduction in High-Performance Circuits,” Symposium on VLSI Circuits, 1998, pp. 40-41