# Dual-Vth Leakage Reduction with Fast Clock Skew Scheduling Enhancement

Meng Tie Haiying Dong Tong Wang Xu Cheng
Micro Processor Research and Development Center
Peking University
Beijing, China

*Abstract* - **Dual-Vth technique is a mature and effective method for reducing leakage power consumption. Previously proposed algorithms assign logic gates with sufficient timing slack to high threshold voltage to reduce leakage power without impact on timing. Meanwhile, clock skew scheduling algorithms are always utilized to optimize period or timing slack. In order to further reduce subthreshold leakage power consumption, in this paper, we ingeniously combine dual voltage assignment technique with intended clock skew scheduling: First, a leakage weight based clock skew scheduling algorithm is proposed to enlarge the leakage power optimization potential. Then we employ a dual-threshold voltage assignment algorithm to minimize leakage power. The experimental results on ISCAS89 benchmark circuits show that, within only several seconds, the leakage power can be further reduced by as much as 41.30% and by 9.87% on average with this new approach, compared to using the traditional method without considering clock skews. Three timing optimized industrial circuit blocks, among which each has around one hundred thousand gates, have also been optimized. It is shown that an average leakage power reduction of 9.95% can be achieved within minutes compared with traditional techniques.**

***Keywords- low power, leakage, dual-threshold, clock skew***

## I. INTRODUCTION

As the technology progress, leakage power has become a significant source of power consumption in integrated circuits. To reduce leakage power, several techniques have been proposed [1]. Among these techniques, dual-threshold voltage (dual-Vth) provides a low cost and effective approach. In dual-Vth designs, high-Vth gates are used on non-critical paths because they have larger delay and less leakage power while low-Vth gates are used on critical paths since they are faster at the cost of more leakage power. Dual-Vth cell assignment techniques have been studied extensively. Several fast and effective algorithms are proposed to reduce static power caused by sub-threshold leakage current [2-7]. Most of the traditional methods implicitly assume that all flip-flops in the circuit have the same clock latency, i.e., the clock skew between flip-flops is always zero, or the clock latencies of flip-flops are constant. In [8], the method of leakage power aware clock skew scheduling is proposed. They integrate clock skew scheduling and threshold voltage assignment into one optimization formulation and use Linear Programming (LP) technique to solve this problem. The introduced clock skews leverage on the borrowed time to achieve leakage power reduction. Compared to other methods, it can save more power dissipation. However, LP method is not practical since it cannot solve large size problem within a reasonable time period.

In order to reduce leakage power using dual-Vth in conjunction with clock skew scheduling in reasonable time, we offer a multi-step approach: First, we compute leakage weight for each adjacent flip-flop pair. Then we propose a leakage weight based clock skew scheduling algorithm to increase optimization potential. At last, under the clock latencies given in last step, dual-Vth assignment algorithm is applied to reduce leakage power.

## II. PROBLEM FORMULATION

The problem is formulated as, under given timing constraints, in a low-Vth gates composed circuit, assign the flip-flops' clock latencies to enlarge optimization potential and then select gates for high-Vth, to save leakage to a feasible extent.

For dual-Vth assignment, a circuit network is modeled as a directed graph $G = (V, E)$. A node $v \in V$ corresponds to a gate in the circuit, more specifically the output pin of a gate when calculating timing information. A directed arc $(u, v) \in E$ represents that node $u$ is an immediate fanin of node $v$, and node $v$ is an immediate fanout of node $u$.

In graph $G$, each arc from u to v is associated with a delay $d(u, v)$ comprised of both cell delay and interconnect delay. The arrival time $AT(v)$ and the required time $RT(v)$ of a combinational gate $v$ are recursively computed by (1) and (2) [3]:

$$AT(v) = \max_{u \in FI(v)} (AT(u) + d(u,v)) \quad (1)$$

$$RT(v) = \min_{u \in FO(v)} (RT(u) - d(u,v)) \quad (2)$$

In these equations, $FI(v)$ ($FO(v)$) denotes the set of all fanins(fanouts), i.e., drivers (loads) of node $v$. The arrival time $AT(V_f)$ and required time $RT(V_f)$ of a flip-flop $v_f$ are given as:

$$AT(v_f) = l(v_f) + t_{clk-Q}(v_f) \quad (3)$$

$$RT(v_f) = T_P + l(v_f) - t_{setup}(v_f) \quad (4)$$

978-3-9810801-6-2/DATE10 © 2010 EDAA

where $Tp$ denotes the clock period, $l(v)$ denotes the clock delay of the node $v$, and $t_{setup}$ denotes the setup time of $v_f$. During timing analysis, $AT(v)$ and $RT(v)$ are updated in topological order beginning from flip-flops. The slack $Slack(v)$ of a combinational gate $v$ in $G$ can be given as the difference between its arrival time and required time:

$$Slack(v) = RT(v) - AT(v) \tag{5}$$

Each node $v \in V$ is also associated with power saving $\triangle Leak(v)$ and cell delay increment $\triangle Delay(v)$ when node $v$ is assigned to high-Vth. A node $v$ can be assigned to high-Vth if $Slack(v) \geq \triangle Delay(v)$.

For clock skew scheduling, a synchronous circuit with edge-triggered storage elements ( flip-flops) is also modeled as a directed graph $G_f(V_f, E_f)$ which is strongly connected, where $E_f = (E_{setup} \cup E_{hold})$. In graph $G_f$, each node represents a flip-flop. Each edge $(u,v)$ in $E_{setup}$ represents the maximum delay path between a pair of flip-flops $u$ and $v$, and each edge in $E_{hold}$ contains a reversed edge $(v, u)$ which represents the minimum delay path between $u$ and $v$. Each flip-flop has its own clock latency, i.e., the propagation delay from the clock source to the clock pin of the flip-flop.

The setup and hold slack, $SL_{setup}(u,v)$ and $SL_{hold}(v,u)$, between a flip-flop pair in $G_f$ are defined as [9]:

$$SL_{setup}(u,v) = Tp - l(u) - D_{max}(u,v) - d_{setup}(v) + l(v) \tag{6}$$

$$SL_{hold}(v,u) = l(u) + D_{min}(u,v) - d_{hold}(v) - l(v) \tag{7}$$

where Dmax(u,v) (Dmin(u,v)) denotes the maximum (minimum) path delay of the edge $(u,v)$ ($(v,u)$), and $d_{setup}(v)$ ($d_{hold}(v)$)denotes the setup (hold) time of node $v$. In programming, the setup and hold time of a flip-flop are included in the path delay. To make sure the circuits do not have setup or hold time violations, both $SL_{setup}(u,v)$ and $SL_{hold}(u,v)$ should be no less than zero. It should be known that assigning a gate to high-Vth will always deteriorate setup timing but seldom deteriorate hold timing since it always increase gate delay.

The basic idea of leakage reduction-aware skew scheduling is to borrow slack from some non-urgent combinational blocks for better leakage power reduction in some other urgent blocks. To illuminate clearly how clock skew affects leakage reduction, we study two examples.

As depicted in Fig. 1(a), combinational gates (shown as smaller circles) are partitioned into two blocks by flip-flop $v_f$. Under zero clock skew, the left combinational block has zero slack, any gates in it cannot be assigned to high-Vth. The right block has slack of 3ns. Assuming that $\triangle Delay$ of each gate is 1ns, normal dual-Vth assignment algorithm will assign two gate in right block to High-Vth. However, it can be easily noticed that, the gate count of left block is quite larger than the number of the right one. If flip-flop $v_f$ is assigned more clock latency by 3ns, the slack of the left block will be 3ns, while the slack of right block will be zero. Thus all five gates ($v_1$, $v_2$, $v_3$, $v_4$ and $v_5$) in the left block can be assigned to High-Vth. In this way, the optimization potential is enlarged by clock skew. We can learn that, sometimes, larger combinational blocks need more slack.

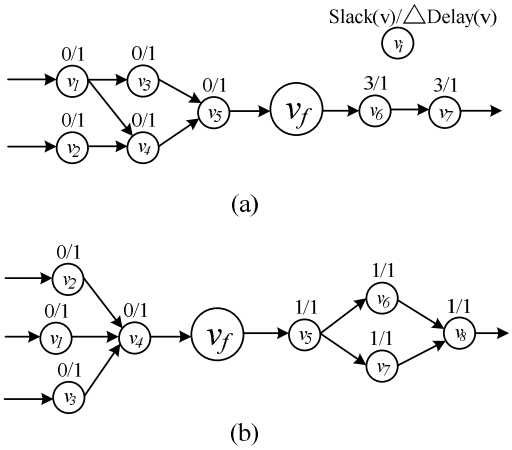


Figure 1. Combinational blocks under clock skew scheduling

The second example is more complicated. As shown in Fig.1(b), the two blocks have the same number of gates. The left block has zero slack and the right has 1ns slack. Here, we assume that all the gates have equal $\triangle Leak$. Thus, two gates in the right block can be assigned to High-Vt. According to the experience learned from the first example, equal slack assignment should be the best choice. However, under 0.5ns slack, no gate can be changed. Instead, by assigning 1ns more clock latency to flip-flop $v_f$, the left block borrows all the slack from right and thus three gates ($v_1$, $v_2$ and $v_3$) can be assigned to high-Vth. This progress relies on the fact that there are more parallelized gates on the left (three: $v_1$, $v_2$ and $v_3$) than on the right (two: $v_6$, and $v_7$).

Studying the two examples, we can know that, the block with more parallelized gates should always be assigned more slack under the assumption of unified gate conditions. In this way, the dual-Vth assignment can save more leakage power. This means the characteristics of the combinational blocks among flip-flops become the bridge between clock skew scheduling and leakage power reduction. In the next section, the details about how this bridge is implemented in our algorithm will be revealed.

## III. ALGORITHMS

In this section, we introduce our multi-step method. First, $Gf$ is got based on graph G, and path delays and path leakage weights are updated. Then, we propose a leakage weight based clock skew scheduling algorithm, Extensive Slack Allocate (ESA), reallocating slack to enlarge leakage optimization potential. The slack reallocation is based on the leakage weights of edges between flip-flops, where edges with larger weights can have more timing slack allocated. This algorithm is inspired by Extensive Slack Balance which has almost linear runtime [9]. Finally, MISA (maximum independent set based slack assignment) based dual-Vth assignment algorithm is performed to reduce leakage power.

**Algorithm** Dual-Vth Leakage Reduction with Fast Clock Skew Scheduling Enhancement
**Input** : Graph $G = (V, E)$
**Output** : $l$:clock latencies of Flip-flops and Set of high-Vth gates, *HList:* the nodes can be assigned high-Vth
1 Get $G_f(V_f, E_f)$
2 Update $AT$ in graph $G$
3 **foreach** $(u,v) \in E_{setup}$
4 Calculate $D_{max}(u,v)$ $D_{min}(u,v)$
5 Calculate $LW(u,v)$ and let $LW(v,u)=0$
6 $l$= ExtensiveSlackAllocate($G_f$, $T$)
7 MISA-DVA($G$, $l$)

**Algorithm** ExtensiveSlackAllocate($G_f$, $T$)
**Input** : Graph $G_f = (V_f, E_f)$, $T$: the initial clock period
**Output:** $l$: clock latencies of Flip-flops
1 **while** (true)
2 $G'$= AllocateWMMC($G_f$, $T$)
3 **if** ($G'=\phi$) **return**
4 **foreach** scc in $G'$: isolate scc from $G_f$

Figure 2. Pseudo code of two-step leakage reduction algorithm

### A. Overview

The pseudo code of our algorithm is shown in Fig. 2. Graph $G$ contains the basic information including gate delay (a multi-input gate may have multiple arc delays), delay increment and gate leakage power. Line 1 gets the graph $G_f$ of flip-flops in $G$. Line 2 updates the arrival time information of each node in $G$, which will be used in the following lines. The updating is carried in a topological order. Lines 3-5 calculate maximum and minimum path delays and leakage weight $LW(u,v)$ of each setup edge $(u,v)$. The definition of $LW(u,v)$ will be discussed later. The leakage weights of hold edges are set to zero. Zero leakage weight tells the following scheduling algorithm not to assign more slack for hold edges. $D_{max}(u,v)$ and $D_{min}(u,v)$ can be computed by accessing nodes in $G$ in a reversed topological order. Line 2 and 4 are fundamental functions for a timing analysis engine. Line 6 calls our proposed ESA algorithm to schedule clock skew based on the delay and leakage weight information got from lines 3-5. Line 7 utilizes MISA algorithm to assign high-Vth based on the clock latencies got from ESA.

Finally, both the scheduled clock latencies of flip-flops and the list of high-Vth gates are obtained.

### B. Leakage Weight Based Clock Skew Scheduling

The path delay can be calculated by updating arrival time of each node in graph $G$ in topological order. To let clock scheduling algorithm have a global view on what combinational blocks have more potential optimization potential, the algorithm computes leakage weight on each setup edge before scheduling. Learned from the two examples in last section, the leakage weight of an edge $e(u_f,v_f)$ in $G_f$ is defined as the sum of the weights of combinational gates between two flip-flops, shown as below:

$$LW(uf,vf) = \sum_{v \in TFO(uf)} \frac{\Delta Leak(v)}{MD(v)} + \sum_{v \in TFI(vf)} \frac{\Delta Leak(v)}{MD(v)} \qquad (8)$$

where $TFO(u_f)$ ($TFI(v_f)$) denotes all transitive combinational fanouts (fanins) of flip-flop $u_f$ ($v_f$). In fact, the summation of $TFO$($TFI$) gates leakage weights are stored in flip-flop nodes to save runtime. $\triangle Leak(v)$ is the power saving of a gate if it is assigned to high-Vth. $MD(v)$ represents the maximum delay of all the paths that go through gate $v$. $MD(v)$ reflects how long a path through $v$ is. And $1/MD(v)$ reflects how urgent node $v$ is to be assigned slack. This is based on the fact learned from the second example. The fact is, with the same gate count, the block having more parallelized gates should be assigned more slack. On the other hand, the paths in that block should be always shorter since more gates are parallelized. So the shorter the paths are, the more urgent to be assigned slack they are.

Our proposed ESA employs the framework of extensive slack balance algorithm (ESB) [9] to solve the leakage weight based skew scheduling problem. To our best knowledge, ESB is one of the fastest practical clock skew scheduling algorithms. It has been used for slack optimization. In ESB, a BalanceMMC [10, 11] algorithm is performed iteratively to solve the slack optimization problem. It works under this theory: when the most critical circle is balanced, it will be isolated out of the graph, and then the second most critical circle will become the most critical one and get balanced. Here, critical cycle is also known as maximum mean cycle (MMC). Since BalanceMMC only balance the slack, it cannot be used for leakage weight based scheduling directly. Thus, we improve BalanceMMC and call the new algorithm AllocateWMMC. The new algorithm reallocates slack based on leakage weight instead of simply balancing in weighted critical cycle (or weighted maximum mean cycle, WMMC), in which the edges are the most urgent for more setup slack assignment.

The pseudo code is shown in Fig.3. Line 1 calculates minimum weighted slack ε in graph $G_f$. It is the criterion for finding weighted critical edges whose $SL_{setup}/LW$ are the smallest. The edges with $SL_{hold}$=zero are not counted since they cannot be assigned more setup slack. Lines 4-8 compute the critical subgraph which contains the setup edges whose weighted slacks $SL_{setup}/LW$ are equal to ε and hold edges whose slacks are zero. Lines 9-10 return to ESA if the slack of WMMC is reallocated. Lines 12-14 compute the cumulative leakage weight from root for each node in critical subgraph. Lines 16-21 calculate the maximum possible slack allocation unit $\theta$ in a conservative way. From last section we know that leakage weight of each hold edges is set to zero and thus these lines are applicable to both setup edges and hold edges. The value of $\theta$ is decided by both setup and hold edges so that no violation will emerge later. Lines 23-24 compute the real latency increment of each node in critical subgraph according to its cumulative leakage weight. And thus slack is reallocated to critical edges from non-critical ones which start from the

ends of critical subgraph. Line 25 increases $\varepsilon$ by slack allocation unit $\theta$, so that in the next iteration AllocateWMMC can find a new critical subgraph. Note that when all the setup edges have equal leakage weights, AllocateWMMC degenerates into traditional MMC balancing algorithm.

After clock skew scheduling, the new slack of an edge $e_f$ becomes:

$$SL'_{setup}(e_f) = \frac{LW(e_f)}{\sum_{e \in WMMC} LW(e_f)} \cdot \sum_{e \in WMMC} SL_{setup}(e_f) \qquad (9)$$

**Algorithm** AllocateWMMC($G_f$, $T$)
**Input** : Graph $G_f = (V_f, E_{setup}, E_{hold})$. $T$: the initial clock period
**Output** : $G'$: the critical subgraph.

1 $\varepsilon = \min_{(u,v) \in E_{setup}} \{ \frac{SL_{setup}(u,v)}{LW(u,v)} \}$
2 **while** (true)
3 //compute critical edges of $G_f$ yielding the subgraph G'
4 $E_{setup}' = \{(u,v) \mid (u,v) \in E_{setup} \wedge SL_{setup}(u,v) = LW(u,v) \cdot \varepsilon \}$
5 $E_{hold}' = \{(u,v) \mid (u,v) \in E_{hold} \wedge SL_{hold}(u,v) = 0 \}$
6 $E' = E_{setup} \cup E_{hold}$
7 $V' = \{v \mid \exists u : (u,v) \in E' \vee (v,u) \in E'\}$
8 $G' = (V', E')$
9 **if** ($G'$ contains a cycle)
10 **return** $G'$ // WMMC is balanced
11 //compute the cumulative weight for each node
12 **foreach** $v \in V'$ $\Delta(v) = 0$
13 **foreach** $u \in V'$ in topological order
14 $\Delta(v) = \max\{\Delta(v), \{\Delta(u) + LW(u,v) \mid (u,v) \in E'\}\}$
15 //compute maximum possible slack allocation unit
16 $\theta = \infty$
17 **foreach** $(u,v) \in E$
18 $\delta = \Delta(u) - \Delta(v) + LW(u,v)$
19 **if** ($\delta > 0$)
20 //$SL$ can be eighter $SL_{setup}$ or $SL_{hold}$
21 $\theta = \min\{\theta, (SL(u,v) - LW(u,v) \cdot \varepsilon) / \delta\}$
22 //compute real latency increment of each node
23 **foreach** $v \in V'$
24 $l(v) = l(v) + \theta \cdot \Delta(v)$
25 $\varepsilon = \varepsilon + \theta$ //update criteria for finding new critical edges

Figure 3. Pseudo code of AllocateWMMC

There will not be setup time violation as long as $LW$ is not negative. It is true according to the definition of leakage weight $LW$.

Fig. 4 illustrates an iteration of the loop body in AllocateWMMC. It shows that, after single iteration, the slacks are reallocated based on leakage weight, and the number of critical edges increases as $\theta$ increases. AllocateWMMC will repeat this process till a weighted critical cycle is found. Hold edges are not drawn in the figure for simplification.

As depicted in Fig. 2, ESA invokes AllocateWMMC iteratively. Since there may be multiple cycles in $G'$ when AllocateWMMC terminates in each iteration, ESA isolates [9] the strong connected cycles (scc) out of current graph $G_f$ and store the clock latencies of the flip-flops in scc. In this way, each time a critical subgraph is found, the scc in $G'$ is packed into a node. Thus AllocateWMMC gradually redistributes slack throughout the whole graph $G_f$. When there is no critical subgraph anymore, ESA terminates and the clock skew scheduling is completed. By restoring the clock latencies stored during ESA algorithm, the leakage optimization oriented clock skews are achieved and thus optimization potential is enlarged.

### C. MISA Based Dual-Vth Assignment Algorithm

We employ MISA based dual-Vth assignment algorithm (MISA DVA) [3] to reduce leakage power after clock skew scheduling. Under a given timing constraint, MISA algorithm will first allocate allowable additional delay to gates such that the total delay assigned to gates is maximum. Here, we use *ad(v)* to denote the allocated delay. Then it assigns the gate with $ad(v) \geq \triangle Delay(v)$ to high-Vth. Equations (1), (2) and (5) are used to compute $AT$, $RT$ and Slack of a gate.

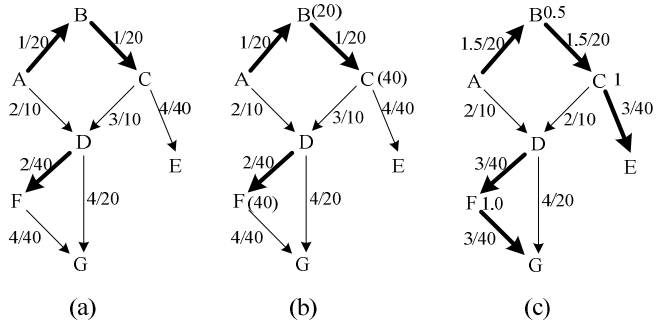


Figure 4. An iteration of AllocateWMMC (a) The initial status of a graph *Gf*, the critical edges are drawn in bold lines and *SLsetup(ef)/LW(ef)* is marked near each edge; $\varepsilon$=1/20 (b) The cumulative leakage weights for slack allocation are marked near the nodes in parentheses. $\theta$=1/40 (c) The allocated clock latencies are calculated and annotated near the nodes; the slacks of edges *SLsetup* are updated and marked. New $\varepsilon$=3/40

The pseudo code of MISA DVA is illustrated in Fig.5. Lines 1-3 construct low-Vth gates list *Llist* by *Slack(v)* in decreasing order, in which each gate v is feasible, i.e., $Slack(v) \geq \triangle Delay(v)$. Gates in *Llist* are the candidates for assignment of high-Vth. Before slack allocation, *ad(v)* of all the gates in *Llist* are set to zero. Line 6 construct a graph *Gmax* based on sensitive transitive closure graph *Gs=(V, Es)*. First Gs is constructed, where there is an edge $(u,v) \in Es$ if there is a sensitive path from $u$ to $v$. A directed path *(u,v)* is called sensitive if it satisfies: when $AT(u)$($RT(v)$) increase (decrease) even only a little, $AT(v)$ ($RT(u)$) will change accordingly. Next, nodes with maximum slack in G are selected as *Vmax* and thus subgraph *Gmax=(Vmax, Emax)*, of Gs, is constructed. Then,

Maximal Weight Independent Set (MWIS) will be found in *Gmax*. Since the maximum independent set problem is NP-complete on general graph, a heuristic method is used to solve it. For node $v$ in MWIS, the additional delay, *ad(v)* is increased by $\varepsilon$. Note that the small constant $\varepsilon$ added in each step should be smaller or equal to $Slack_{max} - Slack_{max-1}$ (line 4). In this case, the minimal amount of nodes in the circuit is affected. Lines 7-8 assign a weight to each node in the graph. The weight *W(v)* is defined to reflect the gain of selecting a gate to work at high-Vth. The first term $\triangle Leak(v)/(1+Adj(v))$ represents the effective power saving, and the second part $ad(v)/\triangle Delay(v)$ reflects how urgent the gate is to be selected. *Adj(v)* denotes the number of adjacent gates to $v$ in $G_{max}$. Lines 10-13 select the gates with largest weight in $G_{max}$ in a heuristic way. Lines 14-19 allocate additional delay $\varepsilon\varepsilon$ to gates in MWIS and if a gate is feasible it will be assigned to high-Vth. Line 20 updates timing information in graph $G$. This process repeats until $\varepsilon = 0$.

In original MISA DVA algorithm, graph $G$ only contains combinational gates. In this work, we refine it by adding flip-flops to graph $G$. We utilize (3) and (4) to make MISA algorithm adapt to non-zero clock skew conditions and assign both combinational gates and flip-flops to high-Vth for leakage reduction.

**Algorithm** MISA DVA($G, l$)
**Input**: $G = (V, E)$, $l$: clock latencies of Flip-flops
**Output**: High-Vth gates list
//Init $Llist$ and $ad(v)$
1 **foreach** $v \in V$
2 **if** $(Slack(v) \geq \Delta Delay(v))$
3 Add $v$ to $Llist$ and $ad(v) = 0$
4 $\varepsilon = Slack_{\max} - Slack_{\max-1}$
5 **while** ($\varepsilon \neq 0$ and $Llist \neq \phi$) {
6 Construct $G_{max}$ of $G_s$
7 **foreach** $v \in V$
8 $W(v) = \frac{\Delta Leak(v)}{1 + Adj(v)} \cdot \frac{ad(v)}{\Delta Delay(v)}$
9 //Find $MWIS$ in $G_{max}$
10 **while** ($G_{max} \neq \phi$) {
11 Move node $v$ with maximum *W(v)* to $MWIS$
12 Remove $v$ and its neighbors in $G_{max}$
13 }
14 **foreach** $v \in MWIS$ {
15 $\varepsilon\varepsilon(v) = \min(\varepsilon, \Delta Delay(v) - ad(v))$
16 $ad(v) = ad(v) + \varepsilon\varepsilon(v)$ // allocate ad
17 **if** $(ad(v) = \Delta Delay(v))$ //assign High-Vth
18 Move $v$ from $Llist$ to $Hlist$
19 }
20 Update $AT/RT/Slack/\varepsilon$ in graph $G$
21 }

Figure 5. Pseudo code of MISA-DVA

## IV. EXPERIMENTAL RESULTS

To evaluate the proposed algorithm and observe its performance on practical designs, we implemented our two step leakage reduction algorithm by C++(for weight calculation, graph $G_f$ extraction and MISA DVA) and Java(for ESA) and choose ISCAS89 benchmark circuits and three industrial circuits as the testcases. The algorithm runs on an AMD 2.0GHz Opteron processor.

First, the circuits are synthesized and timing optimized in Synopsys DesignCompiler with only low-Vth cell library for best performance. The synthesis is based on ARM standard cell libraries on TSMC 65nm technology, characterized under typical process, normal voltage and room temperature. Then we extract the graph $G$ for our algorithm from Synopsys PrimeTime: gates are converted to nodes in graph, timing arcs between gates are converted to edges in $G$, the cell delay, delay increment and power information are stored in the nodes. We replace all the gates to high-Vth so the delay increment can be computed. Then our algorithm extracts *Gf* from graph $G$. To make the timing analysis accurate, we consider both rising and falling delays of each gate in our algorithm since the difference between delay increments of rising and falling propagations at a single gate is comparable with cell delay. In our algorithm, for faster computing, the delay increment on a single gate does not change when its context is altered, like [3], while in fact the cell delay and delay increment could change a little since its input transition and output load capacitance may have changed during dual-Vth assignment. Thus, there may emerge very few violations after power optimization. We utilized a simple script to fix this by changing a high-Vth gate back to low-Vth, one path after another in PrimeTime. The following power data are based on the results after fixing timing.

The experimental results on ISCAS89 are listed in TABLE I. . The first five columns show the basic information about the circuits: circuit name, number of nodes in $G$ (total cells), edges in graph $G$, nodes in $G_f$ (total flip-flops), setup edges in graph $G_f$. The column labeled ORI is the original leakage power after low-Vth synthesis. The column labeled NoSkew and ESA are leakage power after zero-skew MISA DVA leakage reduction and clock skew scheduled leakage reduction respectively. Relative improvement is calculated by (NoSkew – ESA)/NoSkew. Due to the lack of runtime information in [8], we cannot do a direct comparison. So we follow their method to formulate the clock scheduling problem into an LP form and adopt a free LP solver, QSopt [12], to solve it. The LP method results are listed in the columns labeled with LP.

The data show that our proposed algorithm is able to reduce leakage power by as much as 41.3% and 9.87% on average compared to a pure MISA algorithm without considering clock skews. As a heuristic method, its result is not always as good as LP method. But the runtime of our method is much better than LP, which makes our method practical. Clock scheduling runtime comparison is shown in Figure 6. where ESA runtime includes both $G_f$ extraction and scheduling runtime. The effectiveness of our method is further validated on three large industrial circuits, each has about one hundred thousand gates. As TABLE II. shows, the improvement is as high as 16.69% on large circuits and the total runtime including all steps($G_f$ extaction, ESA and MISA DVA) is within an hour for each of the three large circuits.

TABLE I. RESULTS OF OUR ALGORITHM VERSUS TRADITIONAL ALGORITHM

| Circuit | circuit info | | | | Leakage Power (uW) | | | | RelativeImprove | |
|---|---|---|---|---|---|---|---|---|---|---|
| | cell# | edge# | FF# | path# | ORI | No Skew | ESA | LP | ESA | LP |
| s27 | 17 | 26 | 4 | 14 | 1.34 | 1.27 | 1.27 | 1.27 | 0.00% | 0.00% |
| s298 | 97 | 186 | 15 | 85 | 5.59 | 5.22 | 4.14 | 3.82 | 20.69% | 26.82% |
| s344 | 91 | 182 | 16 | 116 | 6.60 | 6.04 | 5.58 | 5.58 | 7.62% | 7.62% |
| s349 | 101 | 193 | 16 | 116 | 6.07 | 5.18 | 5.11 | 4.6 | 1.35% | 11.20% |
| s382 | 127 | 231 | 21 | 174 | 8.73 | 7.98 | 7.98 | 7.48 | 0.00% | 6.27% |
| s386 | 98 | 179 | 7 | 49 | 5.32 | 4.60 | 4.17 | 4.33 | 9.35% | 5.87% |
| s420 | 127 | 222 | 17 | 169 | 11.35 | 9.98 | 8.42 | 8.64 | 15.63% | 13.43% |
| s444 | 148 | 267 | 22 | 174 | 11.52 | 8.82 | 6.62 | 6.62 | 24.94% | 24.94% |
| s510 | 142 | 339 | 7 | 46 | 8.34 | 7.27 | 6.84 | 6.48 | 5.91% | 10.87% |
| s641 | 167 | 248 | 20 | 154 | 16.62 | 10.75 | 10.92 | 10.75 | -1.58% | 0.00% |
| s713 | 176 | 274 | 20 | 154 | 11.44 | 9.24 | 9.54 | 9.24 | -3.25% | 0.00% |
| s820 | 201 | 423 | 6 | 36 | 11.27 | 9.06 | 8.32 | 8.28 | 8.17% | 8.61% |
| s832 | 188 | 412 | 6 | 36 | 8.04 | 6.93 | 6.12 | 6.13 | 11.69% | 11.54% |
| s838 | 277 | 479 | 33 | 593 | 21.05 | 17.96 | 16.16 | 14.95 | 10.02% | 16.76% |
| s953 | 305 | 599 | 30 | 206 | 22.05 | 15.09 | 15.09 | 15.09 | 0.00% | 0.00% |
| s1196 | 325 | 664 | 19 | 57 | 27.90 | 17.80 | 15.60 | 17.1 | 12.36% | 3.93% |
| s1238 | 383 | 742 | 19 | 57 | 29.91 | 18.21 | 18.01 | 18.81 | 1.10% | -3.29% |
| s1423 | 525 | 961 | 75 | 197 | 41.54 | 29.65 | 22.50 | 28.64 | 24.11% | 3.41% |
| s1488 | 380 | 914 | 7 | 49 | 24.23 | 18.44 | 18.44 | 18.42 | 0.00% | 0.11% |
| s5378 | 871 | 1711 | 180 | 1346 | 56.15 | 37.44 | 27.75 | 27.55 | 25.88% | 26.42% |
| s9234 | 1180 | 2435 | 212 | 2692 | 91.51 | 65.39 | 57.05 | 56.11 | 12.75% | 14.19% |
| s13207 | 1917 | 3752 | 639 | 3775 | 124.9 | 53.10 | 31.17 | 35.15 | 41.30% | 33.80% |
| s15850 | 2366 | 4852 | 535 | 12370 | 138.1 | 64.83 | 58.94 | 56.64 | 9.09% | 12.63% |
| s35932 | 6007 | 9834 | 1729 | 6684 | 826.0 | 724.7 | 710.8 | 686.34 | 1.91% | 5.29% |
| s38417 | 6321 | 14043 | 1637 | 34210 | 436.9 | 286.2 | 263.9 | 259.95 | 7.79% | 9.17% |
| s38584 | 6859 | 15501 | 1276 | 14945 | 327.3 | 169.5 | 153.0 | 143.36 | 9.77% | 15.43% |
| Average | | | | | | | | | 9.87% | 10.19% |

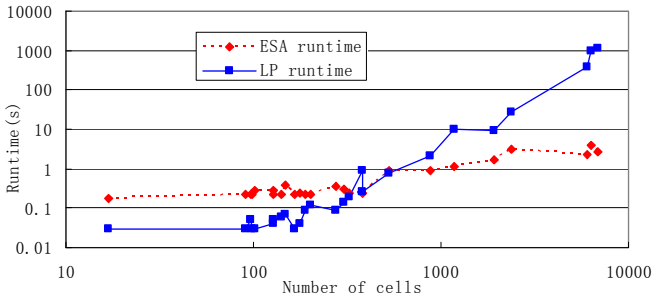


Figure 6. Runtime comparison between ESA and LP

It should be noticed that the proposed leakage weight based clock skew scheduling should be able to cooperate with most of the traditional dual-Vth assignment algorithms, since the separation between clock skew scheduling and dual-Vth assignment.

## V. CONCLUSION

We propose a new algorithm for leakage reduction with clock skew scheduling enhancement. The proposed leakage weight based clock skew scheduling redistributes setup slack between flip-flops for better leakage optimization. Compared with traditional dual threshold voltage assignment algorithm, our method can save additional leakage power up to 41.3% and 9.87% on average within only several seconds for benchmark circuits. For industrial circuits it can save additional leakage power up to 16.69%.

## REFERENCES


[1] S. Narendra and A. Chandrakasan, Leakage in nanometer CMOS technologies: Springer, 2006.

[2] L. Wei, Z. Chen, K. Roy, M. Johnson, Y. Ye, and V. De, "Design and optimization of dual-threshold circuits for low-voltage low-power applications," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 7, pp. 16-24, 1999.

[3] Y. Ho and T. Hwang, "Low power design using dual threshold voltage," Proceedings of the Conference on Asia South Pacific Design Automation, pp. 205-208, 2004.

[4] J. Seomun, J. Kim, and Y. Shin, "Skewed flip-flop transformation for minimizing leakage in sequential circuits," Proceedings of the annual Conference on Design Automation, pp. 103-106, 2007.

[5] A. Srivastava, D. Sylvester, and D. Blaauw, "Concurrent sizing, Vdd and V th assignment for low-power design," Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, 2004.

[6] S. Gupta, J. Singh, and A. Roy, "A Novel Cell-Based Heuristic Method for Leakage Reduction in Multi-Million Gate VLSI Designs," Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on, pp. 526-530, 2008.

[7] A. Davoodi and A. Srivastava, "Probabilistic dual-Vth leakage optimization under variability," Proceedings of the International Symposium on Low Power Electronics and Design, pp. 143-148, 2005.

[8] M. Ni and S. Memik, "Leakage power-aware clock skew scheduling: converting stolen time into leakage power reduction," Proceedings of the annual Conference on Design Automation, pp. 610-613, 2008.

[9] K. Wang, L. Duan, and X. Cheng, "ExtensiveSlackBalance: an approach to make front-end tools aware of clock skew scheduling," Proceedings of the annual Conference on Design Automation, pp. 951-954, 2006.

[10] S. M. Burns, "Performance analysis and optimization of asynchronous circuits," California Institute of Technology, Computer Science Dept, CA, 1991.

[11] K. Ravindran, A. Kuehlmann, and E. Sentovich, "Multi-domain clock skew scheduling," Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 2003.

[12] QSopt Linear Programming Solver, 2008. http://www2.isye.gatech.edu/~wcook/qsopt/


TABLE II. RESULTS ON INDUSTRIAL CIRCUITS

| Circuit | Circuit Information | | | | | Leakage Power (mW) | | | Relative Improve | Total runtime(m) |
|---|---|---|---|---|---|---|---|---|---|---|
| | Function | cell# | edge# | FF# | path# | ORI | NoSkew | ESA | | |
| usb_otg | USB On-The-Go Controller | 77709 | 198766 | 11450 | 2536045 | 3.232 | 1.132 | 1.042 | 7.95% | 12.3 |
| h264e | H264 Encoder | 104667 | 280379 | 18425 | 798086 | 6.655 | 2.481 | 2.067 | 16.69% | 4.5 |
| RISC-II | A 32 bits RISC CPU | 124301 | 279135 | 25913 | 6661098 | 4.890 | 2.395 | 2.270 | 5.22% | 34.0 |
| Average | | | | | | | | | 9.95% | |