Circuit power optimization using pipelining and dual-supply voltage assignment

Teng Xu *, Miodrag Potkonjak

Computer Science Department, University of California, Los Angeles, 3750 Boelter Hall, CA 90095, USA

ARTICLE INFO

Keywords:
Pipelines
Circuit optimization
Dual-supply voltage
Low power design
Power modeling

ABSTRACT

Power is one of the most important metrics in the modern integrated circuit design. We optimize the circuit power using two major approaches, pipelining and dual-supply voltage (dual-V_{dd}) assignment. To improve power efficiency, we have designed a new pipelining to reduce the number of gates need to be assigned to the high supply voltage when combined with the dual-V_{dd} assignment. Our overall design is tested on a set of standard ISCAS-85 benchmark circuits using an industrial cell library. An average power saving of more than 10% under a specified target delay is observed.

1. Introduction

Low power has always been among the most important metrics in the modern circuit design. Systems such as mobile devices and wireless sensor networks have imposed new requirements to energy efficiency and utilization. For example, battery technology improvements have not kept pace with the rapid device scaling that has continued to follow Moore’s Law. The platforms such as mobile phones and tablets are highly restricted by the amount of energy that can be stored in the battery and, therefore, energy efficiency has become a premier design requirement. On the other hand, energy consumption directly impacts the circuit temperature, which leads to the creation of hotspots. Frequent and significant fluctuations in temperature can have ramifications on the circuit reliability. To summarize, energy minimization is of great importance to the modern circuit design.

Pipelining is a frequently used circuit optimization technique to speed up the execution of computations in a circuit by dividing the original circuit into n consecutive sub-computational units and overlapping their executions. Additional flip-flops are placed between the sub-computation units to temporarily store the intermediate results such that the n sub-computational units can be executed in parallel.

Dual-V_{dd} is another circuit design technique to optimize circuit power. Two supply voltages (high V_{dd} and low V_{dd}) are applied to a circuit to save power compared to only applying a single supply voltage (medium V_{dd}). Note that in this paper, we always set circuit delay as a constraint, and use power as an indicator of performance comparison.

It is a natural idea to apply dual-V_{dd} on the top of pipelining to jointly optimize circuit performance. However, the optimal way to combine the two techniques is not yet a solved problem, which is due to the following two facts. The first is that there exist multiple pipeline designs with the same power/delay performance when applied under a single supply voltage. For example, assume that a circuit is to be transformed into a 2-stage pipelining. Since the delay of a pipeline circuit is decided by the maximum delay of each single stage of the circuit, then it is important to guarantee that the delay of the original critical path is evenly divided in such a way that each stage has delay d_{c}/2. To maintain such delay, the non-critical path logic of the circuit can be assigned to either stage as long as they do not create a new critical path longer than d_{c}/2. When combined with the dual-V_{dd} assignment, the situation becomes more complex since now the non-critical path logic needs to be assigned to either low-V_{dd} or high-V_{dd}. However, it is not yet a solved problem regarding the best way of assigning such logic to optimize circuit power without compromising delay.

The second fact is due to a technical constraint of dual-V_{dd} assignment. Only the gates on high V_{dd} can directly drive gates on low V_{dd} to prevent DC current due to incomplete PMOS cut-off [1]. One of the exceptions is that a level converting flip-flop allows the logic after low V_{dd} gates to switch back to high V_{dd}. The advantage of the level converting flip-flops is that they allow more freedom when assigning gates to high or low V_{dd}, while the disadvantage is that the flip-flops themselves consume a significant amount of power, thus only a limited number of them should be placed around the circuit. The discussion of how
to optimally place level converting flip-flops is beyond the scope of this paper, we only consider to optimize the circuit by placing level converting flip-flops between pipeline stages.

Combining the above two observed facts, we claim that different pipeline designs even though with the same delay, can have very different power consumptions after dual-V_{dd} assignment. Fig. 1 shows a motivational example of assigning dual-V_{dd} on a circuit with different pipelining designs. The original circuit contains 18 gates, from G1 to G18. Assume that all gates have the same delay d_{G} on the original V_{dd}. We denote the delay of a flip-flop as d_{FF} on the original voltage. Then for both pipeline designs in (a) and (b) before dual-V_{dd}, the circuit has the same delay 5 = d_{G} + d_{FF}. Now we apply dual-V_{dd} optimization on both pipeline designs. The gates and flip-flops on high V_{dd} are colored red (solid line) and their delays are denoted as d_{G}^{*} and d_{FF}^{*}. Correspondingly, the gates and flip-flops on low V_{dd} are colored eyes (dashed line) and the delays are denoted as d_{G} and d_{FF}. For both pipelining in (a) and (b) after dual-V_{dd} assignment, the critical path delay equals to 2*d_{G}^{*} + 3*d_{G} + d_{FF}^{*}. The key observation is that the number of gates assigned to high V_{dd} in (a) is 14 while the number of high V_{dd} gates in (b) equals to only 6. It suggests that the circuit power consumption after dual-V_{dd} assignment in (b) is less than the power consumption in (a) although both circuit designs enable the same delay.

Motivated by the above example, we propose a pipeline design with the objective to minimize the number of gates on high V_{dd} under the constraint of keeping the delay of the standard optimal pipelining. We only introduce level converting flip-flops between the stages of pipelining, which means that within each pipeline stage only high V_{dd} gates can drive low V_{dd} gates. Therefore, it is important that the width of the circuit is small at the positions immediately after the flip-flops so that fewer gates need to be placed at high V_{dd} to satisfy the delay constraints. To distinguish our paper from the previous related work of pipelining and dual voltage assignment, we summarize our contributions as following:

- Modification to the state of art pipelining algorithm such that when combined with pipelining, fewer gates need to be assigned to high-V_{dd} such that the overall circuit power is reduced.
- Optimize the process of dual-V_{dd} assignment under the scenario of multiple pipelining stages. The optimization includes finding the best pair of dual-V_{dd} as well as deciding each gate should be assigned to which voltage to minimize circuit power under a delay constraint.

The rest of the paper is organized in the following way. We first introduce the related work in Section 2. Section 3 offers an overview of our optimization flow. Section 4 depicts the power and delay model we have applied. Then we separately demonstrate our design for pipelining and dual-V_{dd} optimization in Section 5 and Section 6. Lastly, we present our simulation results on ISCAS-85 benchmark circuits in Section 7.

2. Related work

We review the previous literature on pipelining and dual-V_{dd} optimization in this section.

2.1. Pipelining

The task of pipelining is described in standard textbooks such as [2] and [3]. Many research works focus on the automatic generation of high efficient pipelining designs. Kroening et al. proposed automated pipelining which transforms a sequential machine into a pipelined machine by adding forwarding and interlock logic [4]. Cong et al. presented an architecture-level synthesis solution to support automatic pipelining of on-chip interconnections [5]. The work from Calceran-Oms et al. discussed an automatic pipelining method for micro-architectural systems with loops [6]. Most of the conventional pipelining methods target to maximize the processing performance by reducing clock period [7,8].

2.2. Dual-V_{dd} optimization

Usami et al. first proposed to use multiple supply voltages as a way to reduce energy [9,10]. Salil and Sarrafzadeh applied multiple supply voltages at the behavior level for energy minimization [11]. Dynamic programming techniques for solving the dual-V_{dd} scheduling problem in both non-pipelined and functionally pipelined data-paths were proposed by Chang et al. [12]. A low power implementation on the FPGA platform using dual-V_{dd}/dual-V_{th} fabrics was proposed by He [13,14]. Ishihara proposed the level converter required for dual-V_{dd} systems [15]. Srivastava introduced technology that minimizes both switching and static power using simultaneous V_{dd} assignment [16]. Lee et al. studied the use of dual voltages by considering the requirement for power-network planning [17]. More recently, we proposed to combine dual-voltage assignment with retiming to jointly optimize circuit energy [18]. The idea of refining the state of art retiming is to move around flip-flops in circuit such that fewer gates need to be assigned to high-V_{dd} in dual-V_{dd} assignment. The work presented in this paper shares a similar objective, but has switched the circuit topology change technique from retiming to pipelining.

To the best of our knowledge, this work is by far the first to combine pipelining with dual voltage assignment to jointly optimize circuit power. The focus of our work is to modify both techniques to best accommodate with each other.

For pipelining, besides the standard goal of minimizing clock period, we add the objective of placing flip-flops in such a way to reduce the number of gates need to be assigned to high-V_{dd} when dual-V_{dd} is applied after pipelining.

As for dual-V_{dd} assignment, our optimization differs from the standard approach in a way that the level-converting flip-flops between pipelining stages have been pre-placed, which should be considered as extra constraints when optimizing. Note that our dual-V_{dd} assignment method specifically does not require voltage converters inside each pipeline stage. Under this assumption, it still manages to reduce power.

A preliminary version of this work was published in international workshop on power and timing modeling, optimization and simulation (PATMOS), 2016. When comparing to the preliminary version, this paper has extended in the following aspects.

First of all, an improvement for the pipelining splitting algorithm has been proposed. The position of cuts on critical paths is now further optimized to acquire for an even more remarkable energy saving under the same delay constraint. According to the simulation, our new pipelining design can lead to an average 3.3% power saving compared to the pipelining proposed in the preliminary version.

The second extension is to formally introduce our approach of identifying and assigning a dual-voltage pair on multiple pipelining stages. The goal is to find out the combination of high voltage and low voltage that covers the circuit with least energy overhead given the delay constraint.

3. Optimization flow

Our flow of optimization is shown in Fig. 2. In the first stage, we apply gate level simulation to the benchmark circuits based on Markovic’s delay and power model [19]. Based on the circuit model, we then apply pipelining to split circuit into stages. The last step in the flow is to assign dual-V_{dd} to the circuit for power optimization.

4. Power and delay model

We simulate our circuit by using the delay and power models from Markovic et al. [19]. The delay model is shown in Equation (1), where \( k_{fp} \) is the delay-fitting parameter, \( C_L \) is load capacitance, \( V_{dd} \) is supply voltage, \( n \) is subthreshold slope, \( \mu \) is mobility, \( C_{ox} \) is oxide capacitance, \( W \) is gate width, \( L \) is effective channel length, \( kT/q \) is thermal...
Fig. 1. An example of dual-$V_{dd}$ assignment on (a) pipelining 1, (b) pipelining 2. The flip-flops in the circuit are represented with rectangle, and the gates are represented with circle. The gates/flip-flops that are on original $V_{dd}$ are in grey (middle $V_{dd}$), the ones on low $V_{dd}$ are in blue (dashed line), and the ones on high $V_{dd}$ are in red (solid line). (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)
voltage, $k_{fit}$ is a model-fitting parameter, $\sigma$ is the drain induced barrier lowering (DIBL) factor, and $V_{th}$ is threshold voltage. Load capacitance $C_L$ is defined in Equation (2), where $\gamma$ is the logical effort of the gate and $W_{fanout}$ is the sum of the widths of the load gates.

$$D = \frac{k_{fp} \cdot C_L \cdot V_{dd}}{2 \cdot n \cdot \mu \cdot C_{ox} \cdot L \cdot (\frac{kT}{q})^2} \cdot \frac{1}{(\ln(\frac{e^{\frac{1}{2} \cdot n \cdot \mu \cdot C_{ox} \cdot W \cdot (\frac{kT}{q})^2}}{\sigma \cdot V_{dd} - V_{th} \cdot n \cdot (\frac{kT}{q})^2}))^2}$$  \hspace{1cm} (1)

$$C_L = C_{ox} \cdot L \cdot (\gamma \cdot W + W_{fanout})$$ \hspace{1cm} (2)

We have also considered two sources of power dissipation in an IC. One is from gate switching, in which the ICs dissipate power by charging the load capacitances of wires and gates. The other source is leakage power. Even if the gates do not switch, they dissipate power due to the leakage current. Equation (3) describes the leakage power model, and Equation (4) describes the gate-level switching power model, where $\alpha$ is the activity factor and $f$ is the frequency.

$$P_{\text{leakage}} = 2 \cdot n \cdot \mu \cdot C_{ox} \cdot W \cdot L \cdot \frac{kT}{q} \cdot V_{dd} \cdot e^{\frac{\sigma \cdot V_{dd} - V_{th}}{n \cdot (\frac{kT}{q})^2}}$$ \hspace{1cm} (3)

$$P_{\text{switching}} = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f$$ \hspace{1cm} (4)

5. Pipelining

We discuss our pipelining design in this section. We have focused on the 2-stage pipelining on combinational circuits. Note that our approach is applicable but not limited to the 2-stage pipelining, in fact, it can be easily extended to an n-stage pipelining, which we will explain in details at the end of the section. We explain our pipelining algorithm in two parts. The first part is the summarize of our intuition from Example 1. The key idea is to push as much non-critical path logic as possible to the first pipeline stage so that they can be set to low voltage. This part is our core algorithm, and it provides the majority of energy reduction compared to the non-optimized pipelining when combined with dual-Vdd. The second part of our algorithm is developed on the top of the first part, with the goal of further polishing the previous part algorithm so that the final solution of pipelining becomes nearly “optimal”.

5.1. Pipelining improvement I - stage split

Our first part of pipeline improvement has two major steps, respectively “critical path identification” and “splitting pipeline stages”. As the name suggested, the first step is to identify all the critical
paths in the original circuit under the single $V_{dd}$. This can be easily achieved through the standard critical path identification method, such as dynamic programming proposed by Kirkpatrick [20]. The clock frequency of pipeline circuit is decided by the larger delay of the pipeline stages. Therefore, to achieve the optimal pipelining delay, the gates on the critical paths should be evenly split into two groups regarding delay and then distributed to the two pipeline stages. Assume that the original critical path delay is $d_c$, then the optimal delay after pipelining is $d_f/2 + d_{fp}$, where $d_{fp}$ is the delay of flip-flops.

However, the key remaining question is which pipeline stage should be assigned to the non-critical path logic to be assigned. Fig. 3 depicts an example circuit with critical paths represented by solid black lines. The optimal cuts on the critical paths are represented by the dashed lines in the first figure of 3(a) and (b), with which the original critical path delay is split into half. Then two pipeline strategies are presented in the second figure of 3(a) and (b), where the key difference is the assignment of non-critical path logic. Note that we assume in both pipeline cases, the assignment on non-critical path logic does not introduce new critical paths. In other words, the delay in each pipeline stage stays to be $d_f/2 + d_{fp}$.

The dual-$V_{dd}$ optimization is applied after pipelining. We have observed a difference in the dual-$V_{dd}$ assignment between the pipelining in the last figure of 3(a) and (b). The number of gates assigned on high $V_{dd}$ of the second stage pipeline in 3(a) is significantly larger than that of 3(b). It is because the critical path gates in the second stage pipeline are also driven by the non-critical path logic. Thus to put the critical path gates on high $V_{dd}$, all the non-critical path logic that provides fan-ins to them also needs to be assigned to high $V_{dd}$. Nevertheless, to put the non-critical path logic on high $V_{dd}$ will not further reduce the delay of pipeline stages, with the only consequence to increase the circuit power consumption.

Motivated by the above example, we summarize our key idea of pipeline design as following.

- Keep the optimal cut on the critical paths under the single $V_{dd}$.
- Include as much non-critical path logic as possible to the first pipeline stage without creating a new critical path.

**Algorithm 1 Pipelining Improvement I - Stage Split**

**Input:** C - original circuit.

**Output:** 2-stage pipelining on C. The gates in the first pipeline stage are in $S_1$ (close to inputs), and the gates in the second pipeline stage are in $S_2$ (close to outputs).

1. $S_1 = \emptyset$, $S_2 = \emptyset$, $d = 0$.
2. Identify all the critical paths $c_{pi}, i \in \{0, 1, ..., n\}$ in C with delay $d_c$.
3. $d = d_c$.
4. Append all gates in $c_{pi}$ to $S_2$.
5. For all gates $g_i$ in $c_{pi}$ (inputs → outputs, BFS):
   6. If $g_i$ is directly driven by gates in $S_1$;
   7. If Max(Delay($S_1$, $g_i$), Delay($S_2$)) < $d$:
      8. $S_1$ = append($g_i$).
   9. $S_2$ = remove($g_i$).
10. $d = \max(\text{Delay($S_1$), Delay($S_2$)})$.
11. EndIf.
12. EndFor.
13. EndIf.
14. $d_f = \text{Delay($S_1$)}$, $d_{fp} = \text{Delay($S_2$)}$.
15. $S_{all} = \{\text{all gates in C} \setminus S_1\}$.
16. For all gates $g_i$ not in $c_{pi}$ (inputs → outputs, BFS):
   17. If $g_i$ is directly driven by gates in $S_{all}$;
   18. If Delay($S_1$, $g_i$) = $d_f$ and Delay($S_2$, $g_i$) ≥ $d_f$:
      19. $S_1$ = append($g_i$).
   20. $S_2$ = remove($g_i$).
21. EndIf.
22. EndFor.
23. EndFor.
24. Return $S_1$, $S_2$.

With the above procedures, we guarantee that at the second pipeline stage, the gates on the critical path depend on as little non-critical path logic as possible since most of the logic has been included in the first pipeline stage. Consequently, when putting gates of the critical paths to high $V_{dd}$ in the second pipeline stage, the unnecessary power overhead from non-critical fan-ins is minimized.

The algorithmic process to find such pipeline design is presented in Algorithm 1. In the algorithm, we first identify all the critical paths $c_{pi}, i \in \{0, 1, ..., n\}$ in the original circuit. Then we traverse all the gates on the critical paths to find cuts on the paths to minimize the maximum delay between the upper stage ($S_1$) and the lower stage ($S_2$) given a single $V_{dd}$. With the finalized cut on the critical paths, we then traverse through all the gates on the non-critical paths with breadth-first search (BFS). As long as a gate does not create new critical paths for $S_1$, we incorporate it to $S_1$. Note that at the end of the algorithm, both the delay of $S_1$ and the delay of $S_2$ equal to the target delay ($d_f/2 + d_{fp}$).

5.2. Pipelining improvement II - search optimal cuts

Fig. 4 shows an example of 2 pipelining designs. The original circuit topology is presented (a), and the two pipelining designs together with dual-$V_{dd}$ assignment are shown in (b) and (c). As shown in Fig. 4(a), the original circuit has an upper part and a lower part where the upper part is small in width and the lower part has a larger width. Meanwhile, the two parts employ approximately the same delay, so that the half-to-half delay cuts ($c_{cut}$) on the critical paths are right on the boundary between the two parts. According to algorithm 1, the gates on the critical stages should be evenly split into two groups in terms of delay and then distributed to the two pipeline stages. Therefore, the upper part of the circuit is assigned to pipeline stage 1 and the lower part is assigned to pipeline stage 2. Note that in this example, for the simplicity of demonstration, we do not discuss the assignment of non-critical path logic. For example, the assumption can be that there is no non-critical path logic near the pipeline stage boundary. Then after dual-$V_{dd}$ assignment, the configuration of the circuit is presented in Fig. 4(b). Now that we consider another pipelining scheme in Fig. 4(c). Although this pipelining design does not include the half-to-half cut of the original critical path, by adjusting the high $V_{dd}$ and the low $V_{dd}$, the circuit can still guarantee both pipelining stages to achieve the target delay. More importantly, if we consider the stage 2 of pipelining in Fig. 4(c), the high voltage part now has a much smaller width compared to the stage 2 in Fig. 4(b). It indicates that fewer gates in (c) need to be assigned to the high voltage to achieve the same target delay compared to (b), which can reduce the overall circuit power. Note that it is still not clear to tell which pipeline design between (b) and (c) will employ less power. On one hand, (c) requires fewer gates on high voltage while on the other hand, the $Vdd'$, in (c) need to be higher than the $Vdd_{l}$ in (b) to meet the same target delay ($d/2$).

The observation we have from the above example is that it is not necessary to keep the half-to-half critical path cut when splitting the two pipelining stages. The target delay can also be met in the step of dual-$V_{dd}$ assignment through the adjustment of high/low voltage pair. Depending on the topology of the circuit, the actual split of pipelining stages can be moved upwards or downwards from the critical path cut so that fewer gates need to be assigned to the high voltage.

We summarize the pipelining improvement process in Algorithm 2. The key idea is to iteratively move the original half-to-half cut on the critical path towards the inputs or the outputs, and then apply Algorithm 1 based on the new cut. There are a few conceptual details in the algorithm flow we would like to clarify. Firstly, in each iteration of the loop, we move the original $c_{cut}$ by $i$ gates. Assume that there are $c$ critical paths in a circuit, then the cuts on all the $c$ critical paths will be moved simultaneously towards the inputs or the outputs by $i$ gates. While the $i$ gates on each critical path may have their own delays, this only serves as an approximate way to iterate through the possible cuts near the original $c_{cut}$. In the second,
pipelining the original circuit. The red dashed line indicates the half-to-half cut \( \text{cuth}_i \) on the critical path. (b) Pipelining+dual-V\(_{dd}\) based on \( \text{cuth}_i \). (c) Another pipelining+dual-V\(_{dd}\) design following the \( \text{cuth}_i \). (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

Fig. 4. (a) The original circuit under the single V\(_{dd}\). The red dashed line indicates the half-to-half cut \( \text{cuth}_i \) on the critical path. (b) Pipelining+dual-V\(_{dd}\) based on \( \text{cuth}_i \). (c) Another pipelining+dual-V\(_{dd}\) design following the \( \text{cuth}_i \). (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

The constant \( k \) is defined to offer a range for the search of cuts on critical paths. We would later show that for most circuits, the cut that provides the minimum circuit power is usually either exactly the original \( \text{cuth}_i \) or some cuts close to \( \text{cuth}_i \). Consequently, to refine a range for the cuts search would save the number of search iterations.

Algorithm 2 Pipelining Improvement II - Search Optimal Cuts

**Input:** \( C \)- original circuit, \( k \)- constant.

**Output:** 2-stage pipelining \( < S_1, S_2 > \) on \( C \).

1. Identify all critical paths on the circuit.
2. Perform \( \text{cuth}_i \) on the critical paths so that the delay of all paths are split to half.
3. \( P_{\text{min}} = \infty \), \( < S_1, S_2 > = \emptyset \).
4. For \( i \) from 0 to \( k \):
   5. \( \text{cuth}_i \) = move \( \text{cuth}_i \) towards outputs by \( i \) gates.
   6. \( \text{cuth}^\text{new} \) = move \( \text{cuth}_i \) towards inputs by \( i \) gates.
   7. \( < S^1_i, S^2_i > \) = apply pipelining stage split (Algo 1) based on \( \text{cuth}_i \).
   8. \( < S^1_i, S^2_i > \) = apply pipelining stage split (Algo 1) based on \( \text{cuth}_i \).
   9. \( P_i \) = circuit power after dual-V\(_{dd}\) on \( < S^1_i, S^2_i > \).
10. \( \text{If} \ P_i < P_{\text{min}} \):
11. \( P_{\text{min}} = P_i \).
12. \( < S_1, S_2 >= < S^1_i, S^2_i > \).
13. EndIf.
14. \( P_i = \) circuit power after dual-V\(_{dd}\) on \( < S^1_i, S^2_i > \).
15. \( \text{If} \ P_i < P_{\text{min}} \):
16. \( P_{\text{min}} = P_i \).
17. \( < S_1, S_2 >= < S^1_i, S^2_i > \).
18. EndIf.
19. EndFor.
20. Return \( < S_1, S_2 > \).

The Algorithm 2 is naturally combined with Algorithm 1 and dual-V\(_{dd}\) assignment. It is designed on the top of stage split algorithm to search further for an optimal pipelining solution. Fig. 5 shows an example of running Algorithm 2 on c880 circuit. In the example, the constant \( k \) is set to 4. The circuit power on the y-axis is calculated after pipelining plus dual-V\(_{dd}\) assignment. We can see from the result that when all the single cuts from \( \text{cuth}_i \) are moved towards outputs by one gate, the circuit will achieve the minimum power consumption. In most circuits, the minimum overall power is associated with a circuit cut that is close to \( \text{cuth}_i \). The intuition is that when a cut is close to inputs/outputs, one of the circuit pipelining stage will have a long delay and a large number of gates, forcing the high/low voltages to be increased to meet the target delay. Consequently, the overall circuit power will increase. It explains why we set constant \( k \) as the search limit in Algorithm 2.

5.3. Extension for N-Stage Pipelining

Both our pipelining improvement I and improvement II focus on 2-stage pipelining. However, they can be easily extended to the N-stage pipelining (\( N > 2 \)).

For Algorithm 1, the same idea can be applied to all the boundaries between pipeline stages. For pipeline stage \( i \) and stage \( i+1 \), the goal is to push as much non-critical path logic from stage \( i+1 \) to stage \( i \) as possible without creating a new critical path. So that for each pipeline stage (other than stage 1), fewer non-critical path gates need to be assigned to high voltage.

As for the second pipelining improvement, as shown in Algorithm 2, the following modifications need to be addressed. First of all, an \( N \)-stage pipelining will have a \( \text{cuth}_N \) that evenly splits all the critical paths to \( N \) segments. To be more specific, \( \text{cuth}_N \) actually involves \( N-1 \) separate individual cuts on the critical paths. To search for an improved set of cuts, all the individual cuts in \( \text{cuth}_N \) need to be moved towards either inputs or outputs (each individual cut in \( \text{cuth}_N \) does not need to be moved in the same direction). Similar to the 2-stage...
pipelining, after exhausting all the cut possibilities on critical paths that are within \( k \) gates distance, the cut that provides minimum power consumption after pipelining and dual-\( V_{dd} \) is selected as the optimal cut.

6. Dual \( V_{dd} \) assignment

We start this section by demonstrating the algorithmic flow of finding the optimal dual \( V_{dd} \) assignment for a single pipeline stage. Then we extend the proposed algorithm to find the assignment on a circuit with multiple pipeline stages. In the second case, the selected voltage pair is shared among all the pipeline stages.

6.1. Dual \( V_{dd} \) assignment on a single pipeline stage

There exist two essential issues when applying dual-\( V_{dd} \) to circuits. The first is what voltages should be used, and the second is which part of the circuit should be assigned to high \( V_{dd} \) (or low \( V_{dd} \)). We assume that in our design, only the gates with high \( V_{dd} \) can drive the gates with low \( V_{dd} \), thus we do not need to use the high-cost level converters inside each pipeline stage.

To leverage the above two issues, we propose the algorithmic flow in Fig. 6 to heuristically approximate the best voltage pairs and the corresponding coverage. We assume two gate sets in the circuit, respectively the high voltage set and the low voltage set. Initially, all gates in the circuit are in the low voltage set. In each iteration, we choose one gate from the low voltage set which is on the current critical path with the shortest arrival time and place it to the high voltage set. Afterwards, given the current high/low voltage set split, we use binary search to traverse the pairs of voltages that meet the given target delay and find the pair that achieves the smallest power consumption. We repeat these steps until all gates in the circuit have been placed in the high voltage group. From there, we choose the lowest point of power consumption from all the explored iterations. In practice, the minimum power is frequently achieved when only a small subset of gates are assigned to high \( V_{dd} \).

The voltage for high voltage group and the voltage for low voltage group do not keep the same across iterations. Instead, they kept being changed with binary search as long as they met the following constraints. (1) Each pipeline stage must meet the target delay. (2) The voltage in high voltage group must be equal or higher than the voltage in low voltage group. That being said, the voltage used in the initial circuit will be the same as the voltage used at the end of the algorithm because all gates belong to a single voltage group.

An example of dual voltage assignment flow is shown on Fig. 7. In each iteration, the new critical path of the circuit is circled with a dashed line. As shown in the example, one gate on the critical path of the low voltage group (red cells) is set to high voltage group (red cells) in every iteration. The solution with the overall minimum power consumption is listed using a red rectangle.

Fig. 6. Algorithmic flow of dual voltage assignment.

6.2. Dual \( V_{dd} \) assignment on multiple pipeline stages

The key issue to consider in the dual-\( V_{dd} \) assignment on multiple pipeline stages is that only a single pair of high voltage and low voltage is allowed across the whole circuit (as shown in Fig. 9(b)). However, if we simply apply the optimization flow on every single pipeline stage, then each of them will generate its optimal voltage pair as shown in Fig. 9(a) and these pairs are not likely to equal to each other. The reason we only allow two voltages to be used is because it is always expensive in terms of design and manufacturing cost to add a new voltage to a circuit. We would show that it is not worth it to use more than two voltages since compared to using multiple pairs of voltages, a single pair of voltage will only compromise a small percent of circuit performance.

To find the optimal dual \( V_{dd} \) pair across the whole pipeline circuit, we extend the algorithmic flow presented in Fig. 6. In the first step, we apply dual-\( V_{dd} \) assignment on each single pipeline stage, but instead of only recoding the optimal dual-\( V_{dd} \) pair for each stage, a pool of voltage pairs distinguished by certain voltage scales together with their power performance are all recorded. In the next step, an overall power evaluation is applied on each candidate voltage pair, and the pair with the least overall power is selected as the final pair.

Fig. 10 presents the power evaluation process for different pairs of voltages applied to the c880 circuit. In Fig. 10(a), the dual-\( V_{dd} \) assignment algorithm is applied on the first stage of c880. The corresponding stage 1 power consumptions under different voltage pairs are plotted with the colormap. Similarly, the same process is applied to the
stage 2 of c880, and the result is shown in Fig. 10(b). By adding the results of two stages together, we can easily generate the power consumption of the overall circuit for different voltage pairs, where the result is plotted in Fig. 10(c). We can clearly see that with the voltage pair $<0.53V, 0.76V>$, the overall circuit employs the least power consumption.

The above approach can be easily extended to circuit designs with more than two pipeline stages. By applying the same dual-$V_{dd}$ assignment algorithm to all $n$ stages, and then adding up the power consumption under each voltage pair, the pair that achieves globally minimum power consumption is easily discovered.

7. Experiments

We demonstrate our experimental results on our proposed approach in this section. We first explain our experiment setup, including the cell library, power model, as well as the benchmarks we are using. Then we present the power/delay results on benchmark circuits to illustrate the effectiveness of our optimization algorithm.
However, it moves the non-critical gates with the goal of minimizing the following way. It still splits stages at the half of circuit critical paths, assignment on the circuit. Our baseline pipelining is designed in the ing parasitics capacitances [21]. We then simulate to apply static timing analysis and power analysis to each circuit to retrieve delay/power information before and after our optimization.

In the next step, we have designed 4 circuit states to iteratively evaluate the performance of our algorithm. The first state is when the whole circuit is under the initial $V_{dd}$. The critical path delay of state 1 is recorded and the half of critical path delay is set as the target delay for each single stage in pipelining. The second state is to apply a baseline pipelining algorithm together with the dual-$V_{dd}$ optimization on the circuit. Our baseline pipelining is designed in the following way. It still splits stages at the half of circuit critical paths, however, it moves the non-critical gates with the goal of minimizing the size of cuts. With the above process, it targets to minimize the number of gates.

### 7.1. Experiment setup

We set the initial $V_{dd}$ to 0.7 V and the threshold voltage $V_{th}$ to 0.3 V. For our dual-$V_{dd}$ optimization, we consider high and low $V_{dd}$ within the range from 0.4 V to 1.0 V. We only consider the 2-stage pipelining in our experiment. We evaluate a subset of ISCAS-85 benchmark circuits within the range from 0.4 V to 1.0 V. We only consider the 2-stage pipelining in our experiment.

**Table 1**

<table>
<thead>
<tr>
<th>Circuit</th>
<th>c432</th>
<th>c499</th>
<th>c880</th>
<th>c1355</th>
<th>c1908</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of gates</td>
<td>160</td>
<td>202</td>
<td>383</td>
<td>546</td>
<td>880</td>
</tr>
<tr>
<td>Target delay(ns)</td>
<td>2.03</td>
<td>2.12</td>
<td>1.91</td>
<td>2.16</td>
<td>2.47</td>
</tr>
<tr>
<td>S1-Power($\mu$W)</td>
<td>374.8</td>
<td>536.9</td>
<td>798.2</td>
<td>1158.5</td>
<td>1816.7</td>
</tr>
<tr>
<td>S2-Power($\mu$W)</td>
<td>324.8</td>
<td>516.2</td>
<td>678.9</td>
<td>1001.5</td>
<td>1480.4</td>
</tr>
<tr>
<td>S2--&lt; low $V_{dd}$, high $V_{dd}$ &gt; (V)</td>
<td>$&lt; 0.54, 0.75$</td>
<td>$&lt; 0.52, 0.75$</td>
<td>$&lt; 0.53, 0.76$</td>
<td>$&lt; 0.56, 0.76$</td>
<td>$&lt; 0.55, 0.74$</td>
</tr>
<tr>
<td>S3-Number of gates: &lt;stage 1, stage 2&gt;</td>
<td>&lt; 78, 82 &gt;</td>
<td>&lt; 80, 122 &gt;</td>
<td>&lt; 82, 301 &gt;</td>
<td>&lt; 337, 209 &gt;</td>
<td>&lt; 489, 391 &gt;</td>
</tr>
<tr>
<td>S3-Power($\mu$W)</td>
<td>282.1</td>
<td>516.2</td>
<td>678.9</td>
<td>955.3</td>
<td>1434.8</td>
</tr>
<tr>
<td>S3--&lt; low $V_{dd}$, high $V_{dd}$ &gt; (V)</td>
<td>$&lt; 0.51, 0.77$</td>
<td>$&lt; 0.52, 0.75$</td>
<td>$&lt; 0.53, 0.76$</td>
<td>$&lt; 0.55, 0.76$</td>
<td>$&lt; 0.54, 0.75$</td>
</tr>
<tr>
<td>S4-Power($\mu$W)</td>
<td>282.1</td>
<td>516.2</td>
<td>678.9</td>
<td>955.3</td>
<td>1434.8</td>
</tr>
<tr>
<td>S4--&lt; low $V_{dd}$, high $V_{dd}$ &gt; (V)</td>
<td>$&lt; 0.51, 0.77$</td>
<td>$&lt; 0.52, 0.75$</td>
<td>$&lt; 0.53, 0.75$</td>
<td>$&lt; 0.55, 0.75$</td>
<td>$&lt; 0.54, 0.75$</td>
</tr>
<tr>
<td>S4-Number of gates: &lt;stage 1, stage 2&gt;</td>
<td>&lt; 78, 82 &gt;</td>
<td>&lt; 80, 122 &gt;</td>
<td>&lt; 82, 301 &gt;</td>
<td>&lt; 337, 209 &gt;</td>
<td>&lt; 489, 391 &gt;</td>
</tr>
<tr>
<td>Circuit</td>
<td>c2670</td>
<td>c3450</td>
<td>c5315</td>
<td>c6288</td>
<td>c7552</td>
</tr>
<tr>
<td>Number of gates</td>
<td>1193</td>
<td>1669</td>
<td>2307</td>
<td>2416</td>
<td>3512</td>
</tr>
<tr>
<td>Target delay(ns)</td>
<td>3.09</td>
<td>3.54</td>
<td>3.38</td>
<td>13.58</td>
<td>2.71</td>
</tr>
<tr>
<td>S1-Power($\mu$W)</td>
<td>2355.8</td>
<td>3339.9</td>
<td>4582.0</td>
<td>5376.8</td>
<td>7026.1</td>
</tr>
<tr>
<td>S2-Power($\mu$W)</td>
<td>1862.0</td>
<td>2901.8</td>
<td>3673.3</td>
<td>5119.9</td>
<td>5985.2</td>
</tr>
<tr>
<td>S2--&lt; low $V_{dd}$, high $V_{dd}$ &gt; (V)</td>
<td>$&lt; 0.52, 0.75$</td>
<td>$&lt; 0.53, 0.75$</td>
<td>$&lt; 0.57, 0.76$</td>
<td>$&lt; 0.53, 0.76$</td>
<td>$&lt; 0.56, 0.75$</td>
</tr>
<tr>
<td>S3-Number of gates: &lt;stage 1, stage 2&gt;</td>
<td>&lt; 549, 644 &gt;</td>
<td>&lt; 631, 1038 &gt;</td>
<td>&lt; 803, 1504 &gt;</td>
<td>&lt; 1503, 913 &gt;</td>
<td>&lt; 1079, 2433 &gt;</td>
</tr>
<tr>
<td>S3-Power($\mu$W)</td>
<td>1667.6</td>
<td>2281.8</td>
<td>3224.5</td>
<td>4873.5</td>
<td>5322.9</td>
</tr>
<tr>
<td>S3--&lt; low $V_{dd}$, high $V_{dd}$ &gt; (V)</td>
<td>$&lt; 0.53, 0.76$</td>
<td>$&lt; 0.54, 0.76$</td>
<td>$&lt; 0.55, 0.74$</td>
<td>$&lt; 0.53, 0.75$</td>
<td>$&lt; 0.52, 0.76$</td>
</tr>
<tr>
<td>S4-Power($\mu$W)</td>
<td>1521.5</td>
<td>2114.2</td>
<td>3224.5</td>
<td>4662.9</td>
<td>5006.0</td>
</tr>
<tr>
<td>S4--&lt; low $V_{dd}$, high $V_{dd}$ &gt; (V)</td>
<td>$&lt; 0.54, 0.75$</td>
<td>$&lt; 0.55, 0.76$</td>
<td>$&lt; 0.55, 0.74$</td>
<td>$&lt; 0.54, 0.76$</td>
<td>$&lt; 0.52, 0.76$</td>
</tr>
<tr>
<td>S4-Number of gates: &lt;stage 1, stage 2&gt;</td>
<td>&lt; 778, 415 &gt;</td>
<td>&lt; 908, 761 &gt;</td>
<td>&lt; 1081, 1226 &gt;</td>
<td>&lt; 1774, 642 &gt;</td>
<td>&lt; 2255, 1257 &gt;</td>
</tr>
</tbody>
</table>

**Fig. 10.** Power consumption of (a) stage 1, (b) stage 2, (c) stage 1 and 2 of circuit c880 on different high/low voltage pairs.

**Fig. 11.** Normalized power consumption comparisons between 4 circuit states tested on ISCAS-85 benchmark circuits.
The simulation results are presented in Table 1 with the detailed information of power, delay, voltage pairs, and the number of gates in each stage for each individual circuit. To better visualize the circuit power under different circuit states, we have plotted the normalized power consumption for each circuit state in Fig. 11.

To summarize the improvement of each step (as shown in Fig. 12): compared to the power consumption under initial $V_{dd}$, the introduce of dual-$V_{dd}$ together with baseline pipelining can provide 3.9%–21.0% (Avg. 13.8%) power saving. Compared to state 2, The combination of dual-$V_{dd}$ with pipelining improvement I (state 3) can provide 0%–21.4% (Avg. 7.5%) power saving. With the appliance of pipelining improvement II, it can further reduce circuit power by 0%–8.7% (Avg. 3.3%). We draw the following conclusions from the results.

First of all, the power consumption of our pipeline design achieves better performance compared to the baseline pipeline design when combined with the dual-$V_{dd}$ optimization. In all the tested circuits, our pipelining algorithm is at least equally well performed as the standard pipelining algorithm regarding power consumption.

Secondly, our pipelining achieves more power saving when the circuit has a larger number of gates. It is because for a larger circuit, there usually exists more non-critical path logic near the optimal critical path cut. Therefore in our pipelining improvement I, such logic is merged to the first stage of pipelining. The gates in such logic are now switched from high $V_{dd}$ to low $V_{dd}$ without compromising circuit delay. From the results, we can clearly see that if many more gates are assigned to stage 1 in our pipelining than the baseline pipelining, it often suggests that the power saving between the two pipeline designs is also significant.

Thirdly, there also exists some situation when our pipelining helps little compared to the baseline pipelining, such as circuit c499, where the power saving is 0%. When taking a closer look at the circuit, it is because the area of the circuit near the optimal critical path cut is ultra “thin”, which in other words, exists almost no non-critical path logic. As a result, the stage split between the baseline pipelining and our pipelining stays the same, causing no power saving available.

At last, our pipelining improvement II when combined with the improvement I show an average 3.3% performance enhancement compared to pipelining improvement I along. The result is expected because on some circuits, the optimal pipelining split is at the half-to-half critical delay cut ($cut_{th}$). One observation is that on smaller circuits, the power reduction caused by pipelining improvement II easily becomes 0% while in larger circuits, the reduction is much more significant. In such small circuits, even a minor move away from $cut_{th}$ will create a relatively large imbalance between the size/delay of two pipeline stages, making it much harder to adjust the voltage pairs in a low power way to ensure both stages still meet the target delay. However, on larger circuits, such restriction is more relaxed, offering more space to move $cut_{th}$ around in search for a better pipelining solution.

To visualize the effect of our algorithm, Fig. 13 presents the final standard cell layout using our pipelining improvement I, II and dual-$V_{dd}$ assignment on benchmark circuits c7552 (3512 cells). The red cells include all the flip-flops and the gates placed at the high $V_{dd}$ and the blue corresponds to the low $V_{dd}$ gates. For benchmark c7552, 22% of the cells are placed at the higher $V_{dd}$.

8. Conclusion

A novel design of pipelining and dual-$V_{dd}$ assignment is proposed to enable circuit low power consumption. While our pipeline design maintains the optimal pipeline delay on the circuit, it allocates as much non-critical path logic as possible from high $V_{dd}$ to low $V_{dd}$ to reduce power consumption. Our approach is by far the first attempt to optimize circuit pipelining in conjunction with dual-$V_{dd}$ optimization. We have tested our approach on the ISCAS-85 benchmark circuits, and the results suggest that in all benchmark circuits, our pipelining algorithm performs at least equally well as the standard pipelining algorithm after dual-$V_{dd}$ assignment. An average power saving of 10.4% is observed when comparing our pipelining to the baseline pipelining. We have further polished our pipelining algorithm by introducing flexible cut positions on the critical paths. The results on the benchmark circuits indicate that this improvement further saves circuit power by an average of 3.3%.

Acknowledgment

This work was supported in part by the NSF under Award CNS-0958369, and Award CNS-1059435.
References


