UNIVERSITY OF CALIFORNIA
RIVERSIDE

A Power Reduction Technique Through Dynamic Runtime Algorithm For CMOS VLSI Circuits

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

by

Syed Md. Jaffrey Al-Kadry

September 2012

Dissertation Committee:

Dr. Philip Brisk, Chairperson
Dr. Ertem Tuncel
Dr. Qi Zhu

The Dissertation of Syed Md. Jaffrey Al-Kadry is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to express my heartiest gratitude to my supervisor Dr. Philip Brisk, Assistant Professor, Department of Computer Engineering, UC Riverside. His instruction, guidance and wisdom helped me a lot to perform this thesis.

I also thank the present Graduate Advisor and Professor Dr. Ertem Tuncel for helping me overcoming the hard situations.

I also thank all the professors of my department for the help and information they provided during the whole time of my thesis. Lastly, my wife, Dr. Nilufa Akhter, has given me her unconditional and all time support to accomplish this work.

Finally, I am grateful to the Almighty for giving me the strength and courage to complete this thesis.

To My Wife and Mother

ABSTRACT OF THE DISSERTATION

A Power Reduction Technique Through Dynamic Runtime Algorithm For CMOS VLSI Circuits

by

Syed Md. Jaffrey Al-Kadry

Doctor of Philosophy, Graduate Program in Electrical Engineering
University of California, Riverside, September 2012
Dr. Philip Brisk, Chairperson

All digital circuits have design margins for delay and power consumption. This thesis introduces an algorithm that exploits the design margin for delay to reduce power consumption instead, through the novel application of body bias to the transistors on the critical path. A runtime circuit monitors the activity of critical paths, and applies body bias to transistors on non-critical paths for specific input vectors where the value computed by the critical path is a don't care. In sub-100 nm CMOS devices, the application of adaptive body bias reduced leakage power while slightly increasing the signal propagation delay. When a portion of the circuit does not use up the whole clock cycle, the available slack can be used to reduce leakage power dissipation without compromising performance.

Contents

List of Figures **ix**

List of Tables **xi**

1 Introduction **1**
1.1 Thesis Layout 2

2 Background And Motivation **4**
2.1 Background: Design Margins 4
2.2 Controlling Value 5
2.3 Time Margin Detection and Utilization 6

3 Literature Review On Power Minimization **8**
3.1 Sub-threshold Leakage 8
3.2 Power Reduction Through Adaptive Body Bias 13
3.3 Leakage Characteristics of 45nm CMOS 15
3.4 Adaptive Leakage Control and Speed 16

4 Adaptive Body Bias Circuit: Example **18**
4.1 Reverse Body Bias Circuit 18
4.2 Forward Body Bias Circuit 21
4.3 Activity Factor 23
4.4 Application of ABB to Arithmetic Circuits 24
4.4.1 Example 1: 8-bit Comparator 24
4.4.2 Example 2: 32-bit Carry-Propagate Adder 26
4.4.3 Example 3: 16×16-bit Multiplier 27
4.5 Pre-computation Block 28

5 Algorithms **33**
5.1 Algorithm For Timing Improvement 33
5.1.1 An Alternate Approach 36

6 Simulations And Analysis **39**
6.1 Simulation Results and Setup 39
6.2 Layout Implementation 50

7 Future Directions **60**

8 Conclusions **63**

Bibliography **66**

A Sample Blif File AND Mapping **69**

B Simulation Setup Sample **80**

C Coding Implementation **82**

List of Figures

3.1 *Leakage components in a MOS structure* 9
3.2 *Leakage path in forward body bias* 10
3.3 *Effect of threshold voltage reduction on delay and leakage* 12
3.4 *Effect of substrate bias on leakage components for a 70 nm predictive technology [17]* 14
3.5 *45nm CMOS characteristics of* I_{drain} 15
3.6 *50nm CMOS characteristics of* I_{drain} 16
3.7 *Basic idea of pre-computation-based adaptive body bias control* 17

4.1 *Self-adjusting threshold voltage scheme* 19
4.2 *Standby power reduction scheme* 20
4.3 *Charge pump circuit [26]* 21
4.4 *Forward body bias (FBB) circuit* 21
4.5 *Forward body bias (FBB) circuit using band gap reference [29]* 22
4.6 *Pre-computation-based ABB control applied to a comparator* 26
4.7 *Pre-computation-based ABB Control applied to a 32-bit carry-propagate adder* 30
4.8 *Pre-computation-based ABB control applied to a* 16×16*-bit multiplier* . 31
4.9 *Timing diagram of the 32-bit carry-propagate adder's clock cycle* 31
4.10 *Implementations of pre-computation block* 32

5.1 *Propagation of control logic* 34
5.2 *Backtracking a critical path* 35
5.3 *Detection of control logic and switching the body bias of the circuit to slow it down and save power* 36

6.1 *Power Reduction in 32 bit Full Adder* 40
6.2 *Power Reduction in 16 bit Multiplier* 41
6.3 *Improvement in Delay By The Scheme* 42
6.4 *Power Saving for Basic Gates From ZBB to ABB of 200mV* 44
6.5 *Standard Cell Leakage Power Comparison with and without body Bias* . 47
6.6 *Standard Cell Switching Power Comparison with and without body Bias* 48
6.7 *Body Switching Power* 48
6.8 *Standard Cell Delay* 49
6.9 *VDD and VSS position along with body pickup in a standard CMOS circuit* 51
6.10 *Internal and External Metal Routing* 52
6.11 *Standard cells with same height not joined together* 53

6.12 *Standard cells with same height joined together* 54
6.13 *Extra metal used in supply and ground* 54
6.14 *Less Poly routing at gates* 55
6.15 *Inverter Layout Comparison* 56
6.16 *Buffer Layout Comparison* 57
6.17 *NAND gate Layout Comparison* 57
6.18 *AND-OR-Invert gate Layout Comparison* 58
6.19 *NOR gate Layout Comparison* 58

List of Tables

6.1 *Comparison of Improvement in Different Circuit* 42
6.2 *Power and Delay data for FBB on Basic Cells* 43
6.3 *Comparison of Power Reduction for Inputs Pre-computation at Different Frequencies* 45
6.4 *Comparison of Power Reduction Delay for One Cycle and Sequential Cycles Separately* 46
6.5 *Relationship Between Number of Switching and Power Reduction* 46
6.6 *Total power consumption* 47
6.7 *Comparison on Total Area* 59

Chapter 1

Introduction

Reducing leakage power has been an important challenge for VLSI circuit designers for a long time. Current integrated circuits (ICs) in the market today leak considerably more power than the theoretical minimum; thus, there are many opportunities to reduce different aspects of leakage through innovative research. The leading industrial trend for performance optimization is aggressive scaling; unfortunately, scaling increases leakage current. As a result, the total power consumption of ICs is steadily increasing and approaching the maximum acceptable limits suggested by the ITRS Roadmap [1]. Up to 40% of the total power consumed by a high-performance microprocessor can be attributed to leakage [2].

MOS devices leak current due to sub-threshold leakage, gate oxide leakage, and junction leakage. Process optimization has enabled IC designers to reduce the threshold voltage to around 200 mV today via miniaturization, however, sub-threshold leakage can exceed 50% of the total power consumption of an IC. Unless high-k solutions become widespread, digital CMOS circuits will inevitably pay the cost of high leakage power, whereas, analog circuits do not suffer nearly as much [3]

. Circuit design involves tradeoffs between delay and power. Historically, designers have chosen to expend an increasing amount of current to speed up circuits, with the ground current budget or allowable temperature acting as limits. If device-level improvements cannot be obtained, the next step is move to the block or behavioral levels for optimizations. Researchers constantly strive to find suitable margins in other design parameters that can be sacrificed in order to reduce leakage.

This thesis describes a novel circuit-level scheme that allows the delay margin to be exploited for energy reduction at runtime using body biasing. Body biasing is a technique that improves efficiency by reclaiming lost performance of margins due to variations [4]. If a circuit computes an output earlier than needed, we change the body bias of the MOS device to reduce the leakage current and thus save power. This thesis introduces an algorithm to detect the activity of a circuit at runtime and determines whether or not a sufficient timing margin exists in the delay scenario that body biasing can exploit. Doing so significantly reduces the power consumed during that cycle of execution; aggregated over time, a significant energy savings is achieved.

1.1 Thesis Layout

This thesis is composed of seven chapters:

Chapter 1, introduces the current work and problem statement.

Chapter 2, presents background and motivation for the proposed scheme; it describes how design margins controlling values can be exploited to change the bottleneck of a circuit, which provides an opportunity to save power.

Chapter 3 introduces leakage power and describes a methodology to reduce it through a body bias circuit, and develops a relationship between body bias, power, and delay;

the chapter also discusses the application of body bias to control leakage and speed.

Chapter 4 presents three examples of mainstream arithmetic circuits whose power can be reduced by applying our scheme; circuits to implement forward and reverse body biasing are described.

Chapter 5 summarizes an algorithm that identifies opportunities for power reduction via body biasing in general circuits; the key step is to exploit controlling logic that minimizes the operation time of the circuit under consideration. Pseudocode for the algorithms is provided and its time complexity is analyzed.

Chapter 6 describes our experimental setup and results. The simulation infrastructure is described, results are presented, and data is analyzed. A practical approach to circuit layout involving body biasing is described as well.

Chapter 7 suggests directions for future work, including opportunities to improve the methods and techniques presented in this thesis by incorporating them with other related issues.

Chapter 8 summarizes the work and concludes the thesis.

Chapter 2

Background And Motivation

2.1 Background: Design Margins

Process, voltage, and temperature (PVT) variations significantly complicate delay specification for CMOS circuits, and statistical analysis is required to ensure a proper design flow. Slow devices with random threshold voltage variation may cause the arrival time of signals to diverge significantly from design-time assumptions and expectations. Circuit designers must ensure that even the slowest signal arrives at its output long before it is required.

One approach to reduce the uncertainty associated with random delay is to add redundancy, ensuring that at least one copy of a circuit behaves according to design-time assumptions; however, an over-design of delay specification is always preferable, i.e., the circuit must behave properly and achieve high speeds regardless of the state of the system and across all possible signal combinations. When parallel data bits propagate through the logic levels of circuits, their timing must be equalized; if one particular bit is slower than the others, then the remaining bits must be slowed down to ensure synchronous circuit behavior, i.e., the clock speed is determined by the arrival time of

the slowest bit.

Although critical path analysis is mature, the fundamental nature of the problem has changed due in no small part to the physical properties of deep sub-micron CMOS technologies, and PVT variations in particular. For example, different rise and fall times of logic gates cause the timing of each path in the circuit to be partly dependent on the logic state; changing the state may change which path is critical[5]. In particular, the slowest timing path may be inactive for a large number of all possible logic states. We aim to exploit this low activity factor in this thesis.

In VLSI circuits, PVT variations, which have arisen due to technology miniaturization, significantly complicate specification and design. Static timing analysis has become more critical, but increasingly difficult to perform effectively. Traditional static timing analysis is too pessimistic, and has been replaced by statistical static timing analysis to compensate for the higher level of variation; however, statistical static timing analysis is computationally intensive and does not easily scale for large circuits. That being said, it is possible to address certain constraints and achieve some specific goals by exploiting knowledge of the slowest data paths and their activity factors. For example, circuit redesign to exploit the available timing margins can save power, as described in the remainder of this thesis.

2.2 Controlling Value

For any combinational logic gate or netlist, a **Controlling Value** at the input refers to a subset of inputs that can alone determine the output logic state irrespective of all other inputs, for at least some values. For example, if an input to an AND gate is a logical zero (controlling value), then the output of the AND gate will be zero, regardless

of the values of the other bits. In other words, a controlling value effectively renders all other inputs as don't cares. In general, AND, OR, NAND, and NOR gates have controlling values, while XOR and XNOR gates do not. In this thesis, we will exploit controlling values to enable us to ignore the don't care inputs, which will lead to power saving as a result.

2.3 Time Margin Detection and Utilization

The data propagation time for a given circuit varies, depending on the input vector. Our scheme requires us to find input vectors that have the shortest possible propagation time, which occur commonly in real-life applications; these vectors enable us to identify controlling values, which produce stable logic states far faster than the worst-case path in the circuit. We exploit this time margin to reduce power by switching the body bias of the circuit to slow it down.

The basic idea is to find a controlling value at the nodes of the longest timing path in a netlist. If we can recognize such a controlling value, we set the logic of the next stage immediately, and do not wait for the other signals to arrive, as they will not affect the output. To reduce power, we adjust the body bias of the circuit that produces the other signals to slow them down; the signal values that arrive are incorrect, however, the controlling values effectively "suppress" the incorrect signals, ensuring that the errors do not propagate to the output of the logic stage.

As an example, suppose that the final gate in a combinational logic path is a NAND. One of the NAND inputs is a controlling signal, and the other input is produced by a large logic function that has a much higher timing delay than the controlling signal. If the controlling signal is zero, then the output of the NAND gate is one, regardless of

the value of the other input. We can therefore slow down the logic that computes the other input's value by adjusting the body bias. Even if the value computed is incorrect, the mistake will never propagate beyond the NAND gate, due to the controlling value.

Somewhat more generally, suppose that the delay of the longest path has a delay of T1; the designers will set the period of the cock accordingly. If a controlling value can be found, then, in favorable situations, the delay can be reduced to T2<T1, which provides a margin of T1-T2 that we can exploit. In sub-100 nm processes, Adaptive Body Biasing is one such technique that can exploit this margin to reduce energy. Body biasing affects the logic path that computes the controlling values as well. Therefore, it is important that we do not slow down the path beyond the margin outlined above; otherwise, errors will occur, and performance will suffer.

The ability to effectively identify controlling values is the fundamental innovation that lends credence to the feasibility of this thesis.

Chapter 3

Literature Review On Power Minimization

3.1 Sub-threshold Leakage

In deep sub-micron technologies, shrinking CMOS transistors has led to an increase in the contribution of leakage to total power consumption; in the past, dynamic switching power was a larger contributor than leakage to total power consumption, however, this is no longer the case. Today, leakage may contribute up to 50% of total power consumption in a typical microprocessor [6]. The ITRS roadmap indicates that there are many possibilities to reduce leakage current, which occurs primarily in the source and drain of a MOS device. Leakage depends on the sub-threshold characteristic of MOS as well as the threshold voltage V_{TH}[7].

There are several sources of leakage in a MOS device. The four most significant sources are:

1. Sub-threshold current

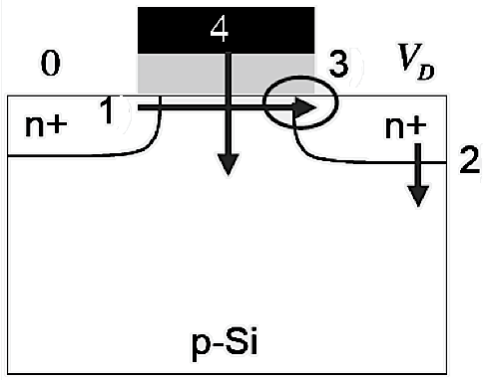


Figure 3.1: *Leakage components in a MOS structure*

2. Junction leakage

3. Gate-induced drain leakage (GIDL)

4. Gate-leakage

Figure 3.1 depicts the leakage components in a generic MOS structure. Our work focuses primarily on the sub-threshold leakage current, which is the current that flows between the source and drain when the gates-to-source voltage is below the threshold. The sub-threshold region is often called the weak inversion region [8]; although beyond the scope of this work, weak inversion is a widely used and efficient operating mode in analog applications [9]. In a digital circuit, this current is viewed as parasitic leakage in a state where there would be no current under ideal circumstances. Leakage current depends on sub-threshold voltage and temperature. Although it is possible to implement temperature tracking schemes in CMOS, it is generally easier to design and fabricate circuits to manipulate the threshold voltage.

After fabrication, the threshold voltage, V_{TH}, of a transistor can be manipulated by changing the body-to-source voltage. In bulk MOSFETs, the formula for V_{TH} is:

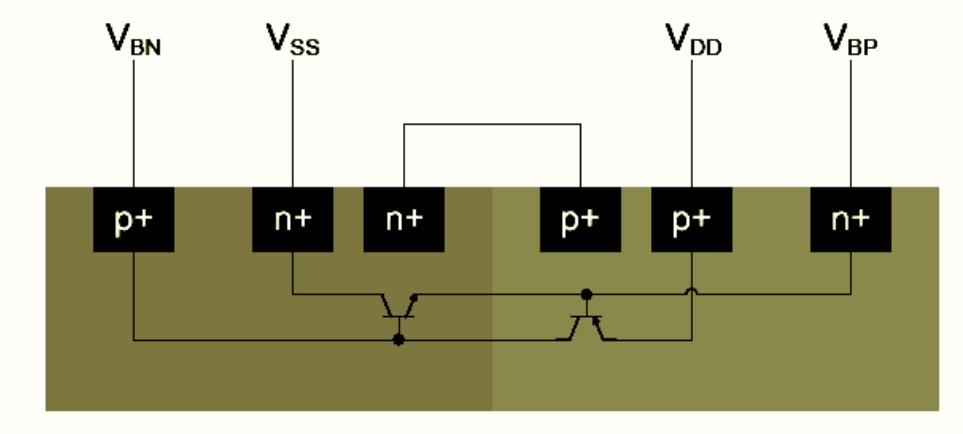


Figure 3.2: *Leakage path in forward body bias*

$$V_{TH} = V_{TH_0} + \gamma(\sqrt{|2\phi_F - V_{BS}|} - \sqrt{|2\phi_F|}), \tag{3.1}$$

where

$$\gamma = \frac{t_{ox}}{\epsilon_{ox}}\sqrt{2\epsilon_{si}qN_A}, \tag{3.2}$$

$$\phi_F = \frac{kT}{q}\ln\frac{N_A}{N_i}, \tag{3.3}$$

ϕ_F is the work function potential, V_{BS} is the body-to-source potential difference, γ is the body effect, and V_{TH_0} is the device threshold without any body bias applied. If a negative voltage is applied to the substrate in an NMOS transistor, the depletion width is increased so that a higher gate voltage would be required to form an inversion layer channel under the gate, which increases V_{TH}. This effect is called a reverse body bias (RBB); similarly, applying a positive voltage at the body will reduce V_{TH}, which is called a forward body bias (FBB).

Applying body bias without limits in either direction can negative affect per-

formance. The forward bias is limited by the flow of leakage current across the P-N junction. A potential latch-up scenario is also present under FBB, as Figure 3.2 shows. The thyristor-like P-N-P-N junction causes a runaway effect, and eventually breaks down. Oowaki et al. show that up to 0.5V FBB can be applied without inducing a latch-up [10]. The RBB limit is set by high leakage and the breakdown of the reverse biased drain-body junction during burn-in [11]. Within these limits, both RBB and FBB can be used to improve performance. FBB can boost the clock frequency, while RBB can reduce active leakage.

The following equation relates the delay of logic propagation to the drain current:

$$T_d = 1/4 \times C_L V_{DD}(1/I_{Dn} + 1/I_{Dp}), \tag{3.4}$$

where I_{Dn} and I_{Dp} are the drain currents of NMOS and PMOS, which are manipulated by V_{BS}. Thus, adjusting the source-to-body voltage can modulate the signal propagation time through the circuit. Figure 3.3 shows the relationship between leakage current and propagation delay: a linear increase in delay yields an exponential reduction in leakage current.

The following equation governs sub-threshold leakage current:

$$I_{ds}^{sub} = I_0 . \frac{W}{L} . e^{(\frac{V_{gs} - V_{TH} - V_{off}}{nV_T})} . (1 - e^{(-\frac{V_{ds}}{V_T})}), \tag{3.5}$$

The overdrive voltage, V_{od}, is the voltage applied at the gate in excess of V_{TH}, and governs the speed of switching strength. Due to device scaling, both V_{TH} and V_{od} are decreasing due to ESD requirements and reliability issues. Lowering the threshold voltage tends to increase the leakage current; thus, leakage current, and its contribution

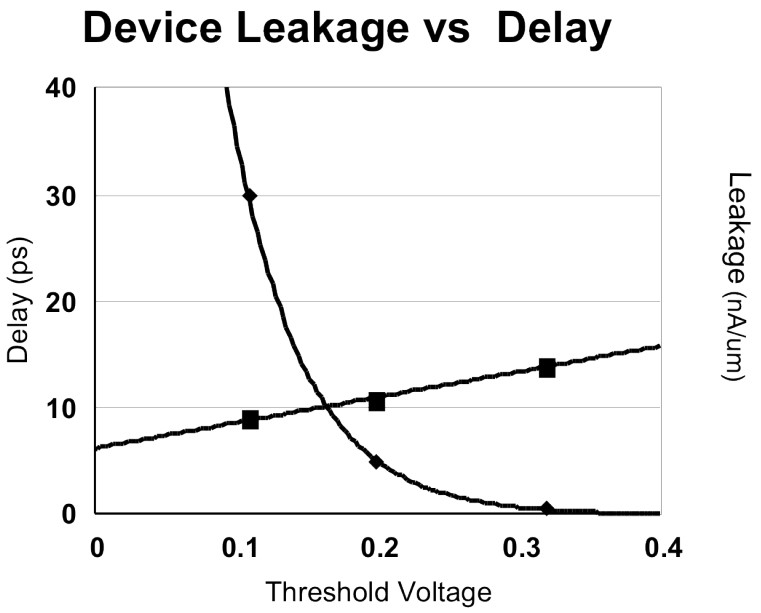


Figure 3.3: *Effect of threshold voltage reduction on delay and leakage*

to total power consumption, tend to increase with further device scaling.

On the other hand, increasing V_{TH} would reduce the leakage current, but increase delay. The relationship between V_{TH} and delay is given by the following equation:

$$t_d = 2C_L V_{dd}/(\beta)(V_{dd} - V_{TH})^{\alpha}, \tag{3.6}$$

and plotted in Figure 3.3. Thus, decreasing V_{TH} by applying body bias can reduce the propagation delay, t_d.

Our work focuses on the Berkeley Predictive Technology Model [12] and we run HSPICE simulations to determine the optimal body bias for a given technology, and to characterize the leakage and delay behavior.

3.2 Power Reduction Through Adaptive Body Bias

Adaptive Body Bias (ABB) refers to the process by which either a forward or reverse body bias (FBB or RBB) can be applied. Researchers have found that FBB is useful for process-variation immunity [13] ,[11] and for aggressively scaled low- temperature CMOS circuits at the 10 nm gate length and 1.5 nm oxide thickness [14]. RBB, meanwhile, can reduce overall leakage and maximize efficiency.

The optimal RBB value to minimize leakage in a given technology depends on many factors [12]; the performance improvement in the active mode of a device by applying RBB includes reduced sensitivity to V_{TH}, frequency response, and the short channel effect [15], [16]. To best exploit these parameters, it is necessary to determine the correct level of body bias to apply with appropriate resolution.

ABB [15] [6] employs a bias generator circuit to find an optimal solution over different technology generations. Under the 65nm predictive model, the devices are found to work slower under RBB in comparison to zero body bias (ZBB). Monotonically increasing the delay with RBB suggests that an optimal body bias voltage should be chosen on the basis of the allowable increase in signal propagation delay. As noted previously, a linear increase in propagation delay yields an exponential reduction in leakage current. This provides the opportunity to bias the transistors in a manner that significantly reduces leakage power consumption while minimally impacting performance.

Many techniques have been applied to reduce the power consumption of digital circuits. One approach uses pre-computation to turn off portions of a circuit to reduce dynamic power consumption, and guarding to reduce the leakage current. The pre-computation circuit detects inputs prior to the calculation, and determines whether

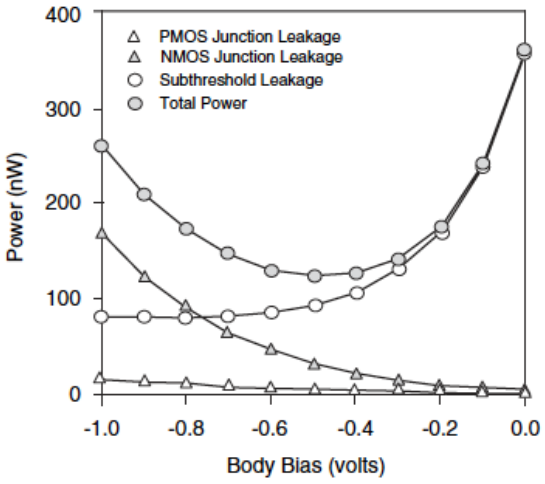


Figure 3.4: *Effect of substrate bias on leakage components for a 70 nm predictive technology [17]*

or not certain parts of a circuit will affect the output, given the specific input vector [18]. When this occurs, the pre-computation circuit can disable the clock signal to the unneeded portion of the circuit to reduce switching activity; this is called signal gating [19]. Guarding is similar, but completely turns off a portion of the circuit to reduce leakage power [20].

The optimal choice of body bias is highly dependent on the technology at hand. The next section will show that FBB reduces leakage as a result of the BTBT tunneling effect in smaller technologies. Our objective, however, is not to manipulate the threshold; instead, our goal is to apply ABB to reduce power when opportunities exist to slow down portions of the circuit that do not affect the outputs, for commonly occurring input vectors. Similar to the works cited above, a pre-detection circuit will identify conditions where certain input variables do not affect the outputs, and ABB will be applied to appropriate portions of the logic accordingly. This will slow down these regions of the circuit to reduce power consumption; however, since the values produced by this subset of logic are effectively don't cares (for a given input vector); any errors induced by the slower propagation delay will not be visible to the circuit outputs.

3.3 Leakage Characteristics of 45nm CMOS

The leakage characteristics of CMOS devices change dramatically as devices scale, especially in the sub-100nm region; in these modern technologies, the band-to-band tunneling (BTBT) leakage at the source-drain junction and the overall gate leakage are highly sensitive to scaling [21]. Body bias voltage has an optimal range at which the leakage is low. FBB increases the sub-threshold leakage, while RBB increases BTBT; both cases increase total leakage.

Figure 3.5 illustrates the effect of substrate bias on leakage components for a predictive 45nm technology, which follows the trend of real fabrication data [21]: the minimum leakage current, as a function of body bias, is achieved at 30% of the normal supplied voltage for NMOS, and 0.7 V_{DD}, as expected, for PMOS. Without changing the circuit structure, leakage power consumption can be minimized if the substrate is switched to the optimal voltage.

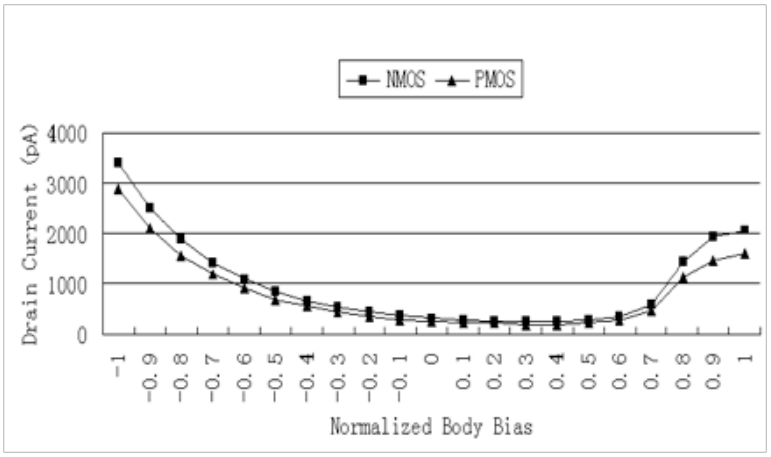


Figure 3.5: *45nm CMOS characteristics of* I_{drain}

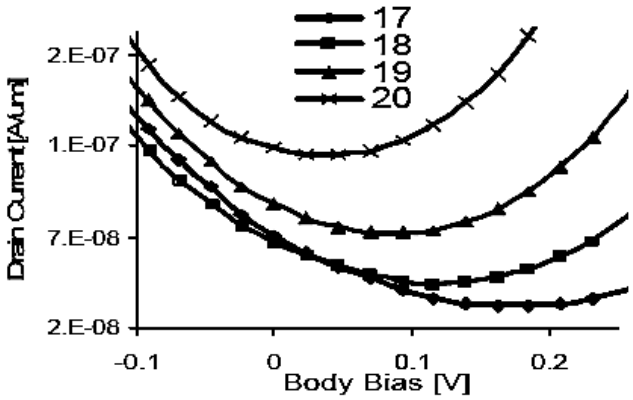


Figure 3.6: *50nm CMOS characteristics of* I_{drain}

as depicted by Neau et al. [21]

3.4 Adaptive Leakage Control and Speed

As noted previously, applying body bias to reduce leakage increases the gate delay. We investigate this relationship using an inverted. We found that delay does not increase as rapidly as leakage decreases. We have shown that if NMOS body bias voltage increases from 0 to 0.3V, the gate delay will increase by 5.5%, while the gate leakage reduces by 36.5% [22]. This mechanism can be exploited in two separate contexts. In a static context, it can be applied to non-critical paths in the circuit where a 5.5% increase will not cause them to become critical. In a dynamic context, which is the primary focus of this work, it can be applied to paths that do not affect the circuit outputs for specific input vectors; a pre-computation detection circuit detects these input vectors and applies FBB to the appropriate paths accordingly. A separate ABB circuit will create and control the substrate biasing. In Figure 3.7, a substrate voltage selector circuit switches the body of the main circuit, depending on the signal computed by the pre-computation circuit. The subsequent sections will describe the ABB circuit, pre-computation schemes for three arithmetic circuits: a comparator, adder, and multiplier, and an algorithm that

identifies pre-computation schemes for combinational logic networks.

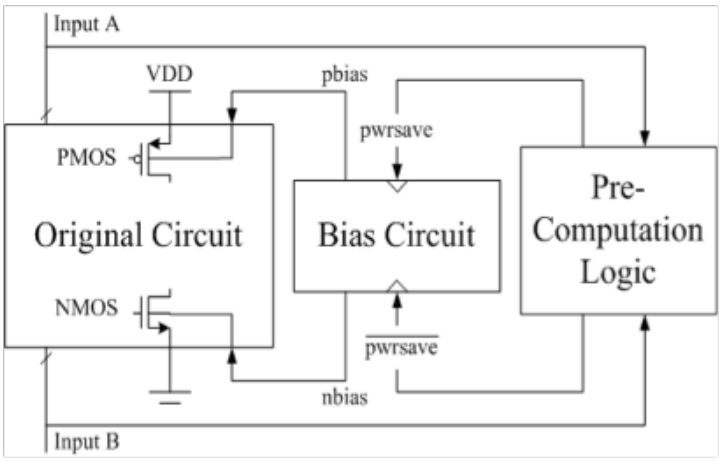


Figure 3.7: *Basic idea of pre-computation-based adaptive body bias control*

Chapter 4

Adaptive Body Bias Circuit: Example

As discussed in the previous chapter, there are two types of body bias: reverse body bias (RBB) and forward body bias (FBB). If both options are available, then we refer to their collective exploitation as Adaptive Body Bias (ABB). The term zero body bias (ZBB) refers to the default case, where no body bias is applied.

4.1 Reverse Body Bias Circuit

There are several implementations of RBB already in the market [25]. RBB requires a voltage level that is above the supply and below the ground potential since the body bias voltage must be greater than the potential of the source terminal for PMOS and below the potential of the source terminal for NMOS; in other words, the body must be connected to $VDD + \delta V$ and $VSS - \delta V$ respectively. If these two potentials are provided by an external source, then some I/O pads must be dedicated to provide them; in this case, it suffices to connect those bodies to those pads. On the other hand,

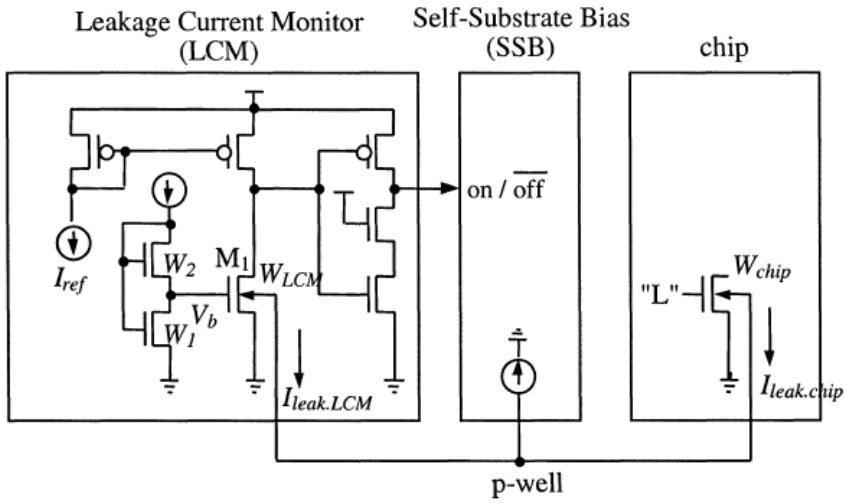


Figure 4.1: *Self-adjusting threshold voltage scheme*

some chips produce these "beyond the rails" potentials internally, and do not require an external body bias. They use two extra bus lines that are routed throughout the chip, but do not connect to any I/O pads.

Figure 4.1 depicts a self-adjusting threshold (SAT) voltage scheme, as suggested by Tadahiro et al. [26]. The SSB circuit generates the required bias voltage, and a leakage monitoring scheme (LCM) turns it on and off, depending on a reference current which is compared to the leakage [27].

As leakage increases, more charge is pumped out of the substrate and V_{TH} increases. When $I_{leakLCM}$ becomes smaller than I_{ref}, the LCM stops the SSB. By intermittently turning this circuit off and on, a target value of V_{TH} can be achieved. Since the SAT cannot reduce standby power dissipation, it can only reduce activate power. To reduce standby power, it is possible to switch the body bias between RBB and ZBB using an SPR (Figure 4.2) circuit and Switch scheme [28]. The advantage of this approach is that it can switch the substrate voltage within $0.1\mu s$.

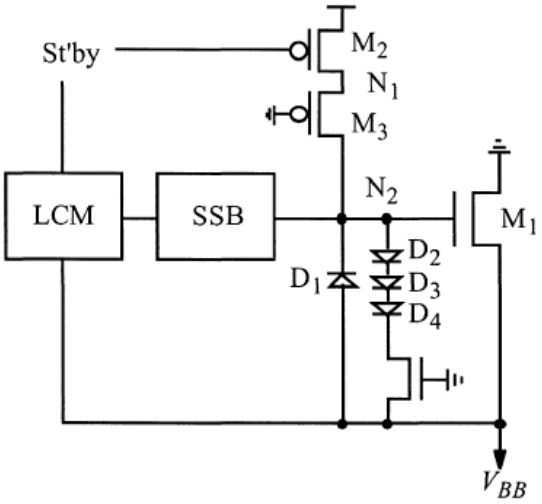


Figure 4.2: *Standby power reduction scheme*

Figure 4.3 shows an on-chip bias solution, which is costly in terms of area and driving current requirements; however, it does not offset the gain. Rather than employing complex circuitry, a simple charge pump, although larger in size, can be employed. P-channel transistors of the diode configuration are connected in series; their intermediate nodes are switch by two out-of-phase signals, $\phi 1$ and $\phi 2$, which alternately turns on each transistor and transfers charge from the P-well to VSS. After several pulses, the charge depletion from the P-well yields a voltage lower than VSS. This circuit is widely used in DRAMs and E^2PROMs. The time required to generate the bias depends on the size of the P-well along with its capacitance; the driving current penalty is approximately $100\mu A$ [26]. We have used implemented this charge pump and used it in our studies; we have found that it is a reasonable means by which to generate bias.

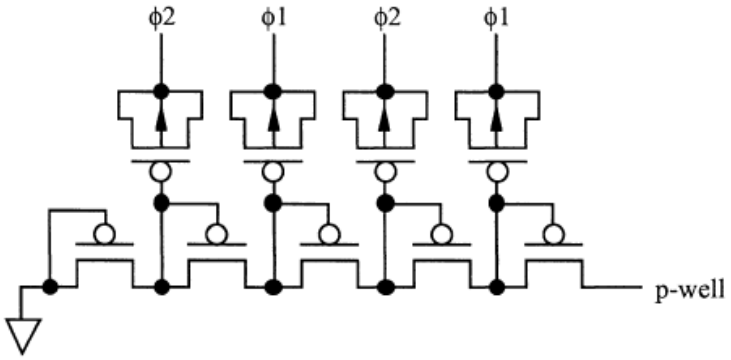


Figure 4.3: *Charge pump circuit [26]*

4.2 Forward Body Bias Circuit

External bias can be used in certain cases to generate the reference voltage of a circuit. An on-chip bias circuit scaled to the supply voltage is generally preferred, because it allows us to track the variation of fabrication of parameters within the main certain. We present a novel body bias circuit which can be used to apply adaptive leakage control on the body of one or more transistors.

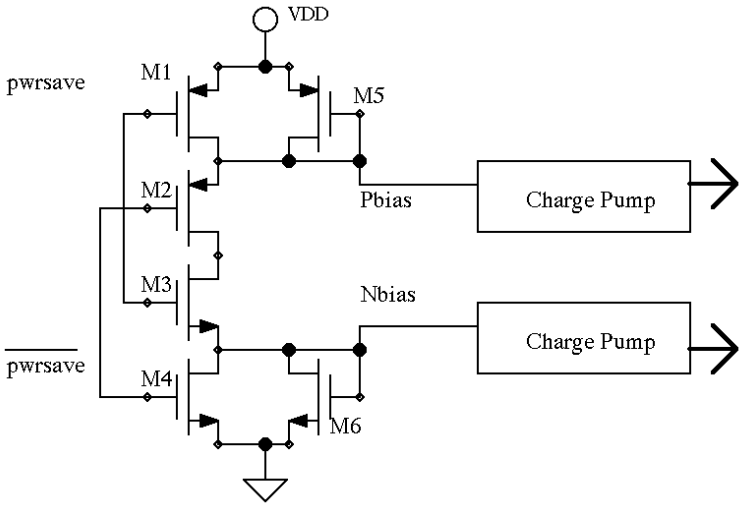


Figure 4.4: *Forward body bias (FBB) circuit*

The fundamental challenge is to generate the necessary body bias voltage that can apply to both PMOS and NMOS transistors. If the body bias control circuit controls a single transistor then the solution is straightforward; however, this is unrealistic due to the area overhead. Instead, our body control circuit provides body bias for hundreds of transistors at once; therefore, an aggregated tradeoff must be made involving its area, drive, and leakage issues.

Figure 4.4 shows an FBB body bias control circuit. Transistors M5 and M6 are used to set the desired values of bias body voltages at nodes "pbias" (for PMOS) and "nbias" (for NMOS). Transistors M1, M2, M3 and M4 switch the nodes "pbias" and "nbias" to either rails or to the body bias values generated by the circuit. Without loss of generality, if control signal "*pwrsave = 1*", then M1 and M cut off and M2 and M3 open; this creates a DC path through M2, M3, M5 and M6. Similarly, when "*pwrsave = 0*", a path is opened between M1, M4, M5, and M6. The circuit shown in Figure 4.4 is a conceptual illustration; a realistic implementation would require a drive capability of the generated voltage pbias and nbias. Since this circuit needs to drive the large capacitances of the P-well and N-well, a charge pump is connected to these nodes; it is similar to the pump circuit described above.

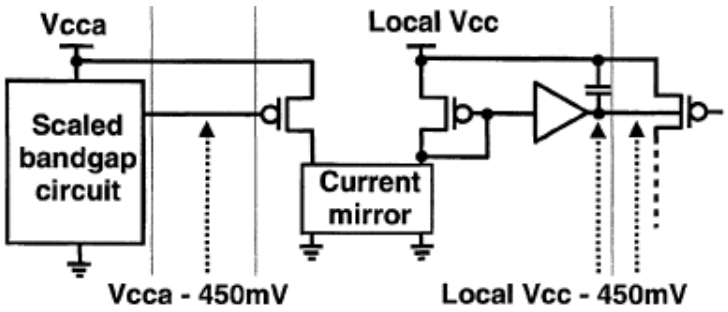


Figure 4.5: *Forward body bias (FBB) circuit using band gap reference [29]*

For BiCMOS process a Band Gap reference is readily available which is immune to process, supply voltage, and temperature (PVT) variations. A scaled version of the Band Gap can be used to generate the optimal body bias. If an NPN transistor is not available in a true CMOS process, the use of substrate PNP can generate a reasonably usable Band Gap circuit. Banba et al. [29] proposed a 450mV FBB circuit from the scaled band gap analog circuit and connected it to a digital circuit using an isolation technique. As shown in Figure 4.5, the bias voltage is generated and converted to a proportional current through a PMOS transistor. This current is steered and mirrored in another PMOS transistor to generate the same voltage across the gate to source, and is also used as a bias voltage for the substrate of the main circuit. A similar approach is applied to NMOS, except the MOS and reference voltages are complemented with an NMOS transistor and a ground reference; a buffer is also inserted to ensure isolation and to provide a high drive strength. The buffer can be implemented using a simple differential amplifier whose output connects to one of its input. The capacitor improves the noise performance and stability, but impacts the switching time of the substrate.

4.3 Activity Factor

Theoretically, the ratio of time of "*pwrsave = 1*" to total running time directly determines how much leakage power is saved by the Adaptive Body Switching. The following equation shows their relationship:

$$TotalPR_{leakage}\% = T_{pwrsave}\% \times avgPR_{leakage}\% \tag{4.1}$$

In equation 4.1, $TotalPR_{leakage}\%$ is the total leakage power reduction in the whole running time, $T_{pwrsave}\%$ is the ratio for time of "*pwrsave = 1*" to total running time, and $avgPR_{leakage}\%$ is the average of leakage power reduction for all input vectors that

makes "*pwrsave = 1*" . Therefore, if the average of leakage power reduction for each input vector is about 20% and the ratio for time of "*pwrsave = 1*" is 50%, then ideally the total leakage power reduction in the whole running time is about 10%. If the leakage power is 40% of the total power, the total power reduction will be 4%. So it can be seen that the amount of power save is not only depending on the leakage current contribution of the devices but also the activity factors of the circuits.

4.4 Application of ABB to Arithmetic Circuits

Pre-computation is a well-established technique that can reduce dynamic power consumption. We use pre-computation to selectively detect situations where part of a circuit can be turned off during a given clock cycle, without affecting the output. Rather than explicitly putting these sub-circuits to sleep, we apply body bias to reduce power, which slows down the signal propagation; timing errors that may occur as a result do not propagate to any observable outputs. Our implementation uses pre-computation-based forward body bias control, as shown in Figure 3.7. After the pre-computation logic block detects and input vector that can be exploited, it sends a pwrsave" signal to the bias circuit. If "*pwrsave = 1*" , the ABB circuit sets the substrate voltages of transistors in the original circuit to the bias voltage values; otherwise, normal substrate voltages are applied.

Here, we describe three examples

4.4.1 Example 1: 8-bit Comparator

Our first example is an n-bit comparator circuit, e.g., greater-than or less-than. The basic idea is to split the circuit two parts: the most significant part (MSP),

consisting of m bits; and the least significant part (LSP) consisting of $n-m$ bits. Let A and B be the values being compared. If $MSP_A = MSP_B$, then the comparison result of LSP_A and LSP_B will determine the output of the comparator; we can detect quite quickly that the MSP does not affect the output.

Let d be the delay of the comparator circuit, and d_{MSP} and d_{LSP} be the respective delays of the MSP and LSP respectively. Under normal synchronous operation, $d = d_{MSP} + d_{LSP}$, which is the delay of the critical path. Given our pre-computation scheme, the answer will be known after time d_{LSP}, as the MSP will not affect the output. Nonetheless, the clock period of the entire system is greater than or equal to d. Thus, we have a timing budget of $d - d_{MSP}$ available for exploitation. Through the application of body bias, we slow down the LSP to save power; as long as the biased delay does not exceed d, the correct values will appear at the comparator outputs at the end of the current cycle.

On the other hand, if $MSP_A \neq MSP_B$, then the MSP alone will determine the output of the comparator, and the LSP output (and carry-propagation into the MSP) do not affect the output of the circuit. Therefore, we can ignore the LSP output and apply body bias to the MSP to reduce energy, as long as the delay of the biased MSP does not exceed d.

Figure 4.6 shows an 8-bit example, with $m = 4$. In this example, $A =$ 00001010 and $B =$ 00001110 are stored in registers R1 and R2 respectively. The pre-computation circuit detects that $MSP_A = MSP_B = 0000$. In the following cycle, the pre-computation circuit sends signal $pwrsave = 1$ so that the ABB control circuit biases the LSP. Concurrently, the comparison of $LSP_A = 1010$ and $LSP_B = 1110$ is executed under the biased voltage, as controlled by the ABB circuit. The multiplexer selects the

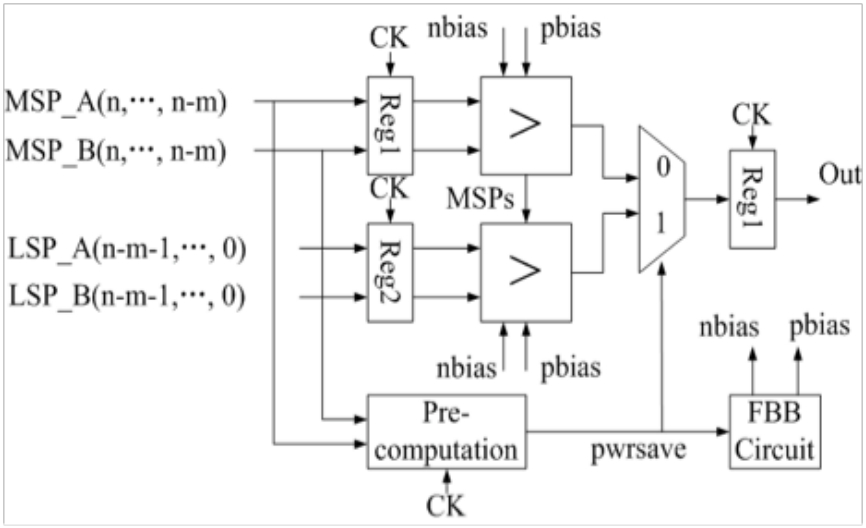


Figure 4.6: *Pre-computation-based ABB control applied to a comparator*

LSP output and forwards it to the output of the comparator circuit.

Similarly, if $A = 10111010$ and $B = 00001110$, then $MSP_A \neq MSP_B$, so body biasing can be applied to the MSP circuit, and the LSP output is ignored.

The reduction in power consumption depends highly on the value of m, which balances delay and power. Additionally, there is a tradeoff between the cost of the pre-computation circuit and the power reduction of the comparator. Lastly, there are many different conditions that a pre-computation-based ABB circuit could detect; this example used one specific case, but many others are possible.

4.4.2 Example 2: 32-bit Carry-Propagate Adder

Carry-propagate adders are fundamental digital VLSI circuits. When designing the functional units of a data-path, the word length is usually based on the maximum dynamic range of the input data, which may be much larger than actual data ranges [23]. For example, consider an n-bit ripple carry adder. If we know that the precision of the circuit will never exceed mn for a certain time period during the execution of an

application, then it suffices to guard the $n - m$ most significant bits, as their outputs are always zero; however, this approach offers no savings when full precision is required.

Figure 4.7(a) shows a 32-bit carry-propagate adder split into two 16-bit parts, once again called MSP and LSP. If the MSPs of inputs A and B are all zeroes, then the LSPs (including the LSP carry-out) will wholly determine the output of the computation. Therefore, we can apply body bias to the LSPs to save energy, while slowing down their operation, effectively converting the 32-bit adder to a slowed down 16-bit adder. As the MSP produces no usable outputs, it can be slowed down using body biasing as well. The multiplexer (MUX) placed between the LSP and MSP and the XNOR gate ensure correct output behavior when the LSP propagates a carry-out into the MSP.

Pre-computation can also identify situations where the MSP and/or LSP of both inputs remain the same in two consecutive clock cycles, as shown in Figure 4.7(b). Let $A(t)$, $B(t)$, $A(t+1)$ and $B(t+1)$ denote the values of inputs A and B (stored in registers R1 and R2 respectively) at times t and $t+1$ respectively. As an example, suppose that: $A(t) = 512A1111$, $A(t+1) = A1FF1111$, $B(t) = F9F08000$, and $B(t+1) = 50008000$. In this case, the respective LSPs of both A and B remain the same, so the outputs of the LSP circuits will be the same as well. Therefore, it suffices to apply body biasing to the LSP circuits to reduce power.

4.4.3 Example 3: 16×16-bit Multiplier

Multipliers are another common arithmetic circuit used in digital VLSI. Here, we use a 16×16 -bit carry save adder (CSA) array multiplier, where a carry-propagate adder sums the final result. We split the multiplier into three basic units, (*), (**), and (***), as shown in Figure 4.8. The pre-computation logic is based on the following observation: if the highest m bits of both operands are all zero, then the highest $2m$ bits

of the output are also zero; in essence, the circuit performs a $(16-m) \times (16-m)$-bit operation, rather than exploiting the full precision. The reduced-precision operation can be slowed down through the application of body bias without affecting the outputs, which reduces power.

Similar to the adder, when the m LSP or MSP bits of both operands remain the same during two consecutive clock cycles, the m low-order bits of the multiplier, and the internal carry-out bits, remain unchanged. Thus, body bias can be applied to the internal circuitry that performs the lower-order $m \times m$-bit operation, leading to a reduction in energy, as slowing down the computation does not adversely affect the observable outputs.

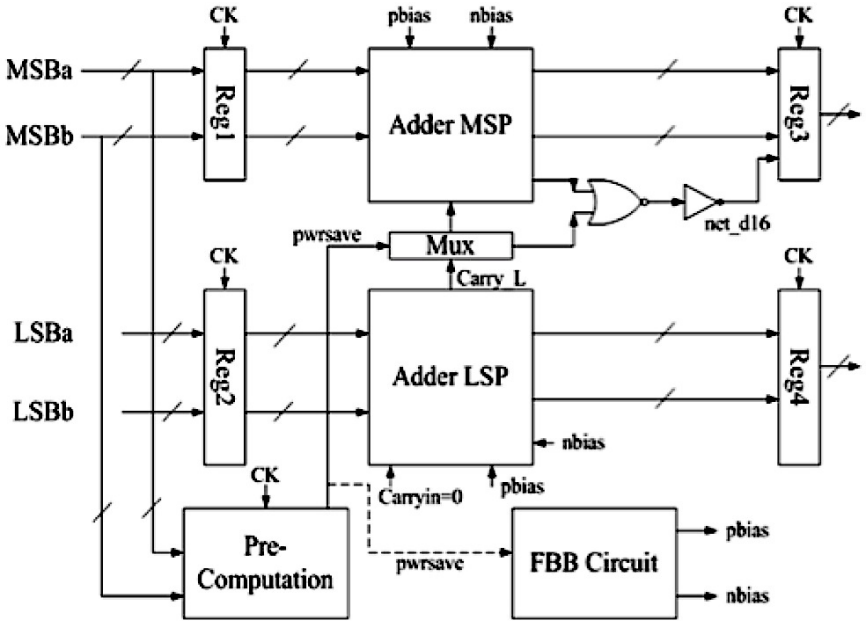
This approach easily generalizes to other multiplier structures as well, however, the primary difference is that way that internal carries are handled; addressing this issue is left open for future work.

4.5 Pre-computation Block

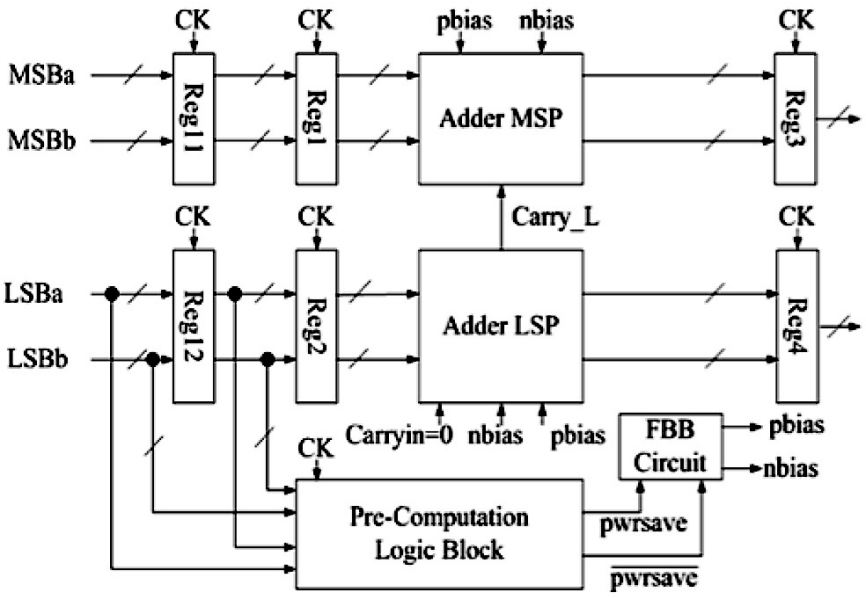
Each of the three components described above requires a slightly different pre-computation block, depending on the input conditions that we wish to check. Figure 4.10(a) presents the generic structure of the pre-computation block when only the current cycle's inputs are considered, while Figure 4.10(b) presents the generic structure of the block when the current and previous cycle's inputs are considered; this circuit requires an additional register stage on its output. Both circuits employ a hierarchical NAND/NOR tree structure. The primary difference is that consider two cycle's worth of inputs doubles the total number of inputs to the tree.

Without loss of generality, assume that the registers used in a system containing

the 32-bit carry-propagate adder are sensitive to the rising edge of a synchronous clock. The delay between data in consecutive cycles is 1 cycle, and the signal "pwrsave" sent by pre-computation must be available before the rising edge of the clock; thus, the register used in the pre-computation block should be sensitive to the falling edge of the synchronous clock. Figure 4.9 shows the timing diagram of the pre-computation circuit.



(a) Inputs Pre-computation in One Clock Cycle



(b) Inputs Pre-computation in Sequential Clock Cycles

Figure 4.7: *Pre-computation-based ABB Control applied to a 32-bit carry-propagate adder*

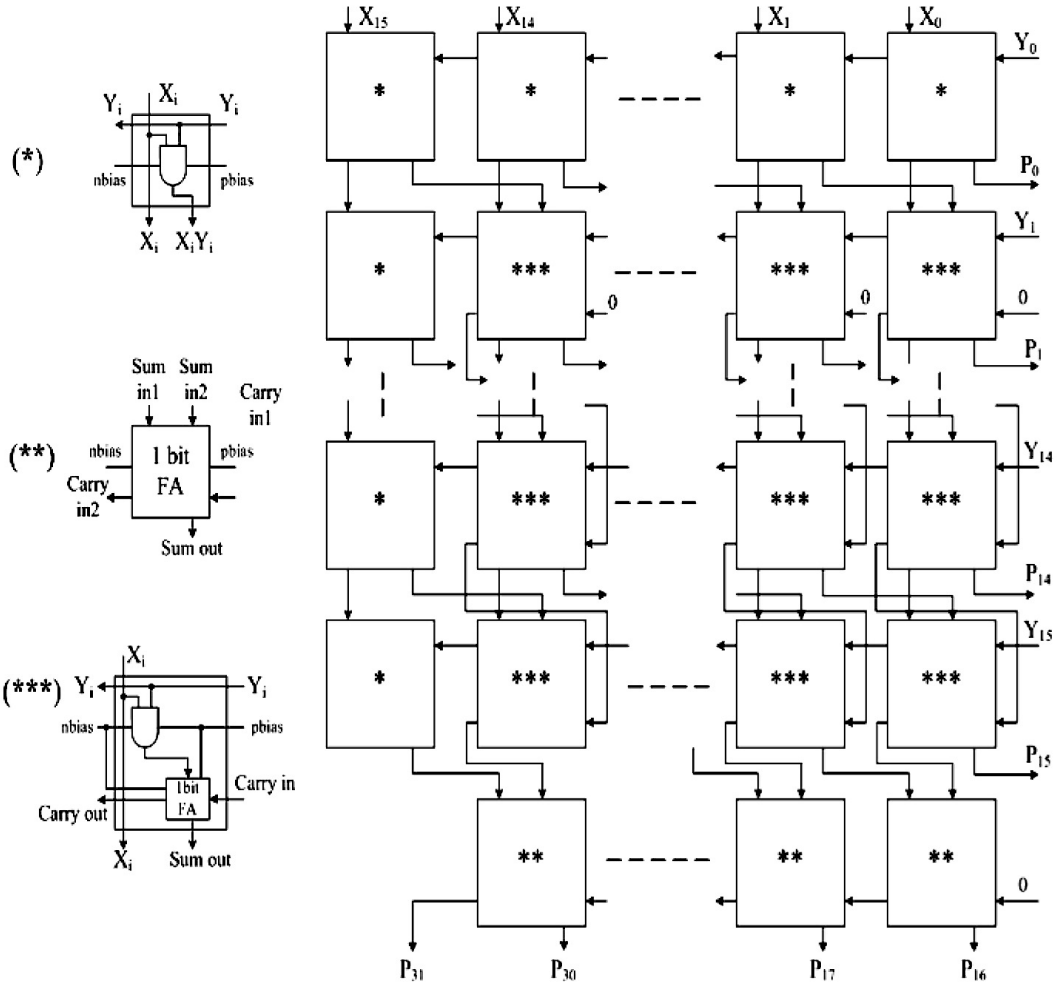


Figure 4.8: *Pre-computation-based ABB control applied to a* 16×16*-bit multiplier*

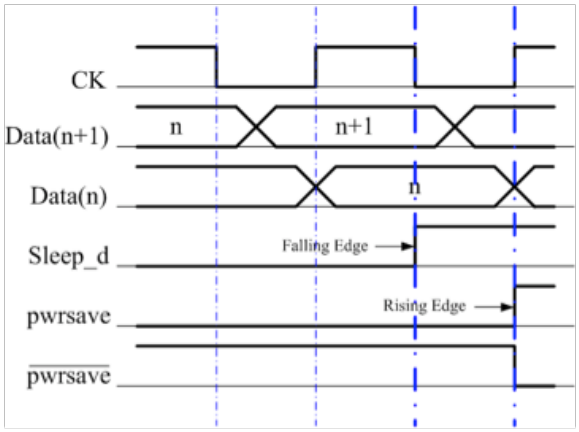
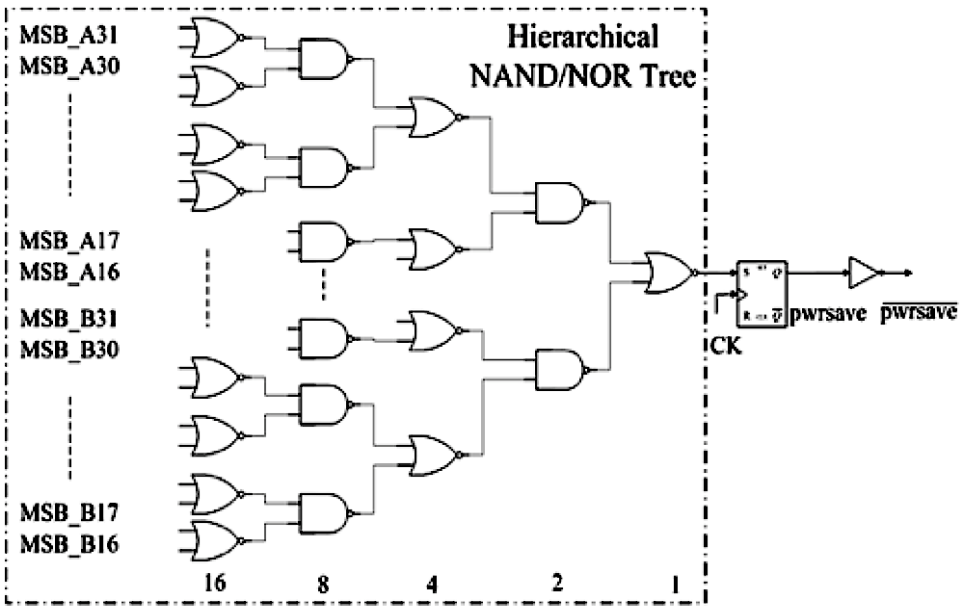
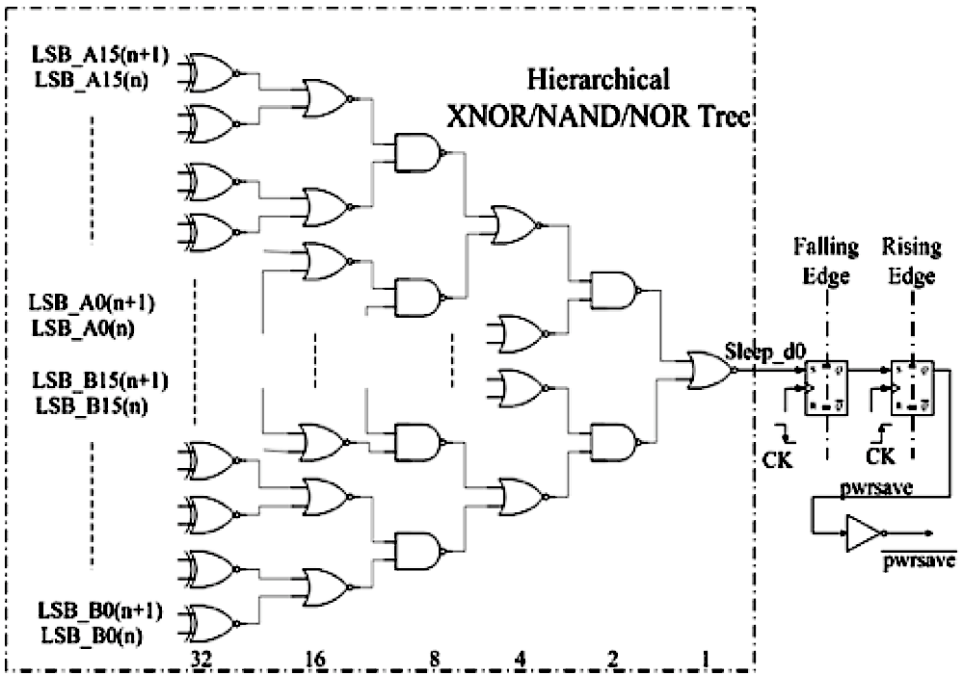


Figure 4.9: *Timing diagram of the 32-bit carry-propagate adder's clock cycle*



(a) Inputs Pre-computation in One Clock Cycle



(b) Inputs Pre-computation in Sequential Clock Cycles

Figure 4.10: *Implementations of pre-computation block*

Chapter 5

Algorithms

5.1 Algorithm For Timing Improvement

The pre-computation circuits described the in the preceding chapter were designed to meet the needs of a set of specific arithmetic components. We desire a generalized methodology to identify opportunities to apply body biasing to any logic circuit, and to automatically generate the pre-computation circuit in response. This chapter describes our general approach.

Chapter 3 discussed the concept of controlling logic values for certain types of gates in a standard library: AND, OR, NAND, NOR, INV. These gates share a unique property that any one of the inputs can deterministically set the state of the output. Once a controlling input is identified, the other input states can be ignored, and their arrival times become immaterial (Figure 5.1). When a controlling input for a specific gate is detected, the arrival time of that gate is equal to the arrival time of the controlling input, ignoring all other fanins.

The algorithms described in this chapter are implemented in C++, and are

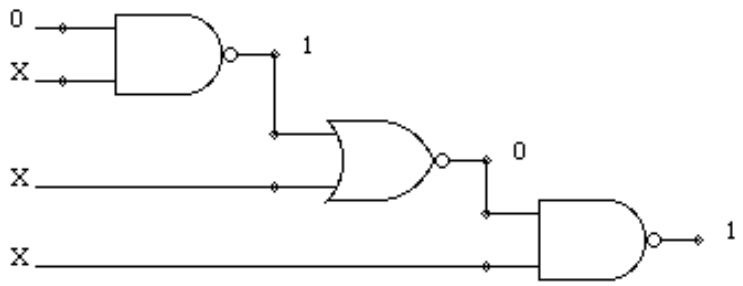


Figure 5.1: *Propagation of control logic*

integrated with the ABC [24] tool for simulation and verification purposes. The first step is to map the circuit to a standard cell library. Second, we identify the critical path; we use a straightforward topological sort for this purpose. When body bias is applied, other paths may become critical as well (but only for specific input patterns that are detected by the pre-computation circuit). Figure 5.2 shows an example; in this case, the critical path is BCDF.

Unlike traditional critical path analysis, we want to identify other slow paths that may still have a timing margin that can be exploited for power reduction; moreover, the timing a specific path will vary, depending on the controlling inputs that we can find.

The next step is to simulate the logic and timing of the circuit together. For an n-input circuit, there are 2^n possible different input vectors; thus, it is computationally costly to enumerate all possible inputs. To prune the search space, we consider each input individually. We set the logic state of each input to zero (and then one) to see if the input is a controlling input to any of the gates that follow. If it is a controlling input, the first level of the circuit's logic state is known, and is then used to determine the logic state of the second level. We propagate this method to reach the output (figure

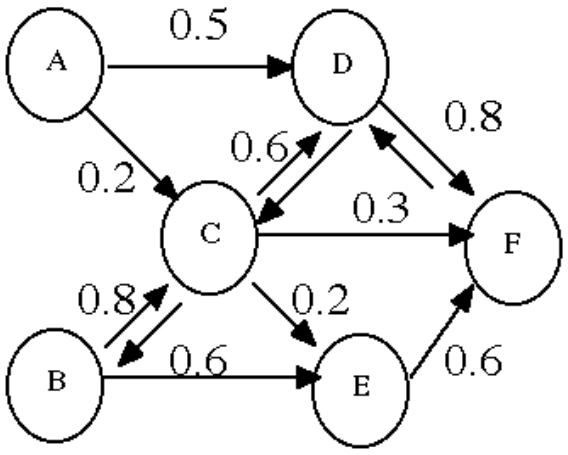


Figure 5.2: *Backtracking a critical path*

5.3). If any input controls the logic state of the original critical path, then it is safe to ignore the arrival time of all branches to that particular gate; thus, for the specific value of the controlling input under consideration, the critical path through the gate is reduced.

For example, in figure 5.3, the number in parentheses shows the signal propagation delay through each gate. Ignoring the pre-computation and body bias switch logic at the bottom of the Figure, critical path delay is 17; however, the fourth input, at the bottom of the Figure, is a controlling input, when its value is zero. When this occurs, the delay of the circuit becomes 5, which is the delay of the NAND gate; this saves 12 units of time, during which we can detect the controlling logic's value, and then slow down the longer critical path by applying body bias, thereby saving power.

The process described in the preceding paragraph is repeated for each input, and for values of 0 and 1. In principle, the process could enumerate all combinations of inputs, and all possible values of controlling vectors for each group of selected inputs;

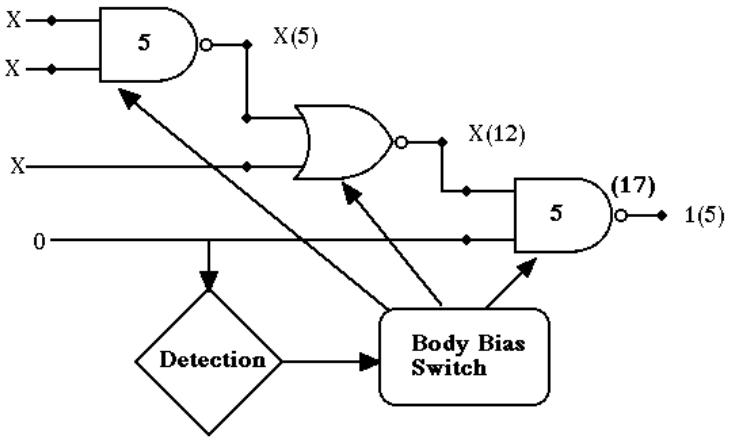


Figure 5.3: *Detection of control logic and switching the body bias of the circuit to slow it down and save power*

however, this approach would have an exponential time complexity in terms of the number of inputs. After processing each input individually, we choose the input (and value) that yields the fastest timing path as our controlling logic input.

Next, we consider every possible pair of inputs and apply the same methodology; here, there are 4 possible input combinations (00, 01, 10, 11) that could act as controlling values for each pair of inputs. To limit the runtime of our algorithm, we do not search for larger input combinations. After processing each pair of inputs, we once again choose the input pair (and value) that yields the fastest timing path as our controlling logic input. In general, pairs of inputs yields shorter timing paths than considering individual inputs, however, the overall benefit varies from circuit-to-circuit.

5.1.1 An Alternate Approach

An alternative approach to identify controlling inputs and their controlling values is to start with the nodes in the netlist, and search backward. Here, we examine each node and look for a controlling input. Once a controlling input to each gate is found, the preceding gate can be tracked for its controlling input. This process repeats

recursively from the primary outputs to the primary inputs of the circuit. Alternatively, the search can limit its focus to nodes on the critical path of the circuit, or nodes only on near-critical paths. Imposing a controlling condition on a particular node can reduce the signal propagation time in the circuit. This approach offers that advantage that multiple controlling inputs and values for a specific node can be identified at once. Thus, it is easier to find large groups of controlling inputs in this fashion, rather than exhaustively enumerating all combinations of a fixed number of inputs and searching for gates that each combination controls.

If we examine each of the nodes of a potentially slow path and look for the controlling input we can quickly determine if there is combination of inputs that can reduce the arrival time. For a given node, the controlling inputs (and every controlling value that they may take) create a condition that the pre-computation logic can detect; when this condition is true, body biasing is applied to the chosen path. The pre-computation detector computes its output much faster than the critical path of the circuit; therefore, it can dynamically switch the body of all transistors on the critical path to slow them down and reduce leakage current. The designer must make sure that the ABB switching does not add more delay than it saved on the critical path's control; we ensure that this condition is true at all times.

Pseudocode for the algorithm is presented below in Algorithm 1:

Algorithm 1 Algorithm for arrival time reduction using control value search

Logic $\Leftarrow 0$

for each input vector **do**

  for each fan-out **do**

    if *gate* = *control gate* **then**

      Logic out = $predefined$ *output*

      $T_{propagation} = Truncated$

    else

      Update $T_{propagation}$ *by adding*

    end if

    if *Output Node Reached* **then**

      Save Total $T_{propagation}$ *and exit*

    else

      Go to next $fanout$

    end if

  end for

  Compare propagation time and save lowest one

end for

Logic $\Leftarrow 1$

Go to next input vector

Chapter 6

Simulations And Analysis

6.1 Simulation Results and Setup

We built and tested the bias generator circuit independently from the logic circuits, and we report their power and performance separately. The bias generator circuit is small enough to evaluate using SPICE, whereas, the benchmark circuits are too large and are not in an appropriate format for SPICE simulation; it is simply not computationally feasible to analyze the benchmark circuits in this way.

A second challenge is that power consumption and delays depend highly on the state. For an n-input circuit, there can be 2^n states, and the power consumption between the states can vary by a factor as large as 3; thus, reporting average power for a specific circuit is useless. Instead, power data could be provided for a given activity factor and a signal probability distribution; however this is computationally costly. To address this concern, we simulated a statistically significant number of states for each benchmark circuit, and report the averages.

To estimate power reduction, we use a heuristic. First, we count the number of cells in a benchmark circuit after technology mapping to a standard library. For each

cell, we know the average power reduction that can be achieved by adjusting the body bias. Therefore, we multiply the total cell count by the power saved to approximate the total reduction in power consumption.

Our simulations ensured that the amount of delay added by body biasing is far less than the timing margin, which ensures correct circuit behavior; secondly, we ensured that the body bias circuit has sufficient drive capability to swing the body bias in real-time. We verified these two properties through extensive simulation.

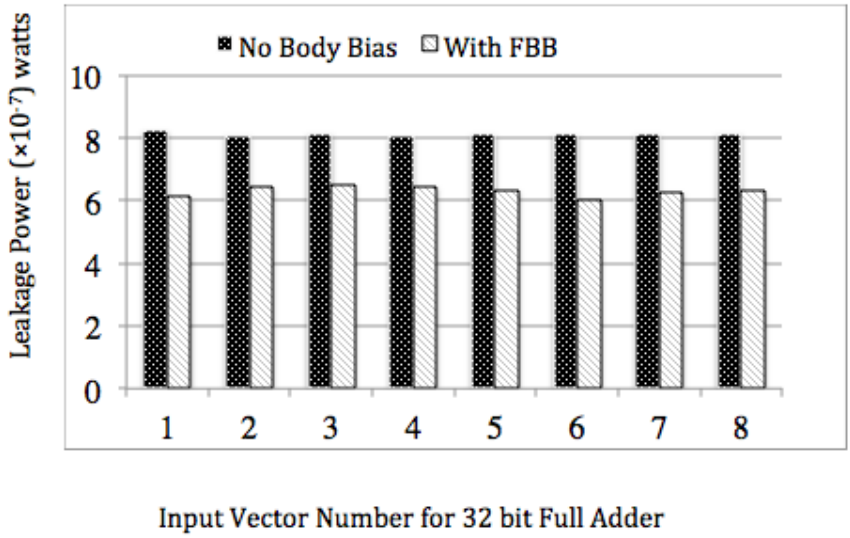


Figure 6.1: *Power Reduction in 32 bit Full Adder*

The transistor-level simulations were based on a 45nm BSIM4 predictive model[8]. The spice models are taken from the ASU Predictive Technology Model (PTM) group [8] with permission. HSPICE simulations were conducted on the ISCAS 89 benchmarks using this model. Our technology mapper converts the benchmark files to a netlist of library cells; the library cells are custom designed IP blocks (by us) which are extended with additional inputs and silicon for body bias control and adaptation.

After mapping, we convert each circuit to a SPICE file. A series of stimuli

is applied to the inputs and we measured the power consumption. We used exhaustive search to find the critical path, identify controlling values, and measure power consumption. Body bias was then applied and propagation delay and power consumption were measured as well. The data reported is the power saved by body biasing and how much delay is introduced as a result. For example, HSPICE simulations of a 32-bit adder and a 16x16-bit multiplier are performed at the transistor level. Figures 6.1 and 6.2 show the comparison on the leakage power of the full-adder and the multiplier with and without body biasing for different input vectors. Typical savings in power range from 20-25% for a 32-bit adder, and 14-20% for a 16x16 multiplier.

Our critical path analysis algorithm treats each circuit as a black box; we ran the algorithm on different ISCAS 89 benchmark circuits. We fund that there were significant opportunities for power savings under a wide variety of input logic combinations. Our algorithm was implemented in C++ on a Windows PC, and was done within the ABC logic synthesis framework, customized for testing.

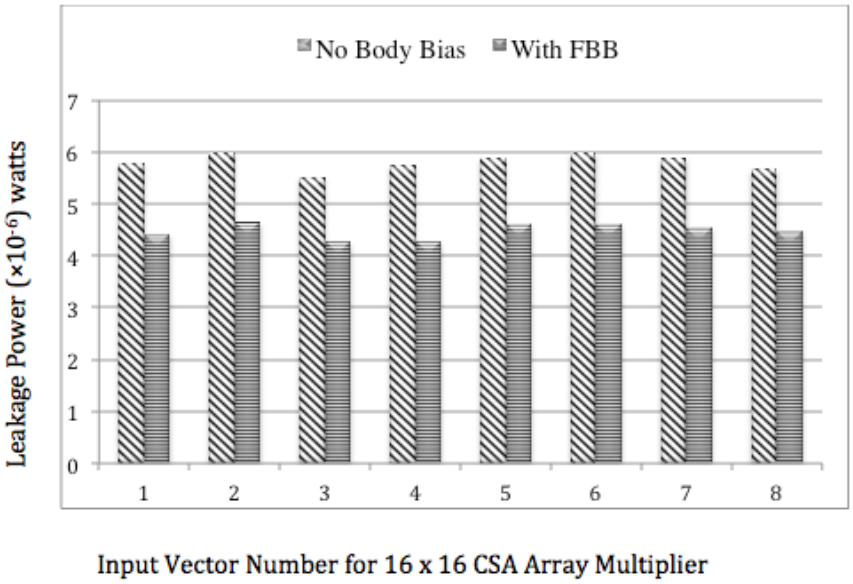


Figure 6.2: *Power Reduction in 16 bit Multiplier*

Circuit Name	Delay of Critical Path	Delay After Applying Algorithm	% improvement
s6669	18.8	14.3	23.93
s444	6.8	5.1	25
s5378	12.4	10	19.35
s38417	18.4	12.6	31.52
c2670	15.7	12.3	21.65
i10	30.8	14.79	51.98
s9234_1	22.1	13.4	39.36

Table 6.1: *Comparison of Improvement in Different Circuit*

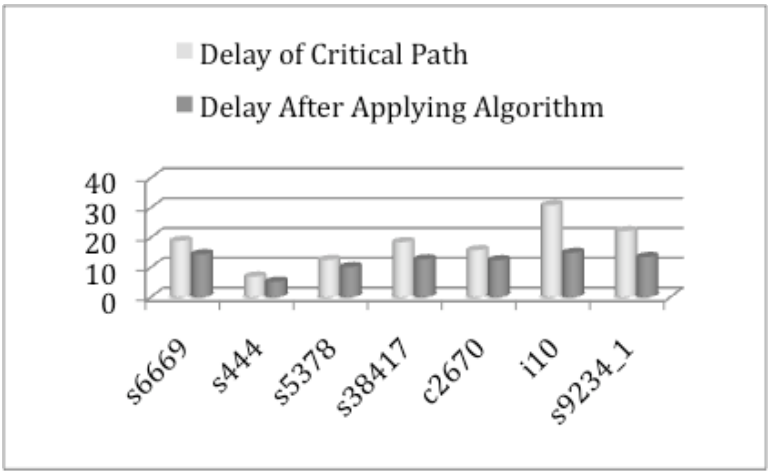


Figure 6.3: *Improvement in Delay By The Scheme*

Figure 6.3 shows the reduction in critical path delay after applying our algorithm to each circuit; the improvement in delay is greater than 20% for all of the circuits we tested. The improvement varies from one circuit to another depending primarily on their respective size, the presence of non-controllable XOR and XNOR gates, and the overall circuit topology. That being said, our tool was always able to identify portions of a circuit where controlling values can be found; moreover, it was always able to identify a timing margin that could be exploited to reduce power via body bias adjustments.

Next, we investigate delay and power tradeoffs using FBB. We simulate an

Cell Name	picoseconds				Nano watts (Power Save)	
	Downward		Upward Delay			
	FBB	ZBB	FBB	ZBB	FBB	ZBB
Nand	44.4	42.4	25.6	24.4	11.3383	14.75
Nor	10.45	10.13	51.11	48.89	12.0212	17.9888
and	48.66	46.78	66.05	64.14	3.2532	4.7836
xnor	81.81	78.44	94.34	90.54	26.1157	34.6658
xor	83.23	79.23	94.92	90.6	23.1375	34.6653

Table 6.2: *Power and Delay data for FBB on Basic Cells*

inverter chain in HSPICE using a 65nm Berkeley Predictive Model as a motivating example. All simulations are done at a temperature of 100 C, which is the approximate temperature of an actual circuit. We found that introducing 5% delay reduced leakage power by more than 30%; it is important to note that this inverter chain required mapping to appropriate standard cells in our library that have been modified to support ABB.

Our library contains AND-OR-INVERT, OR-AND-INVTER, 3-input NOR and NAND, and many other gates. We simulated each gate using the model described above, and we observed reduced leakage when given an FBB of 200mV. The delay increase varied from cell to cell; we report the worst possible case here. An FBB voltage of 200 mV adds a maximum of 5% delay with respect to zero body bias (ZBB); the leakage current reduction was 23-33%.

These experiments demonstrate that identifying controlling values can reduce the arrival time of critical bits by at least 19.35%; however, only a maximum of 10% delay is added with a small FBB voltage of 200 mV, resulting in power savings of at least 23%. Figure 6.4 and Table 6.2 show the results of the simulations.

Our scheme imposes an area penalty for the body bias circuit and pre-detection

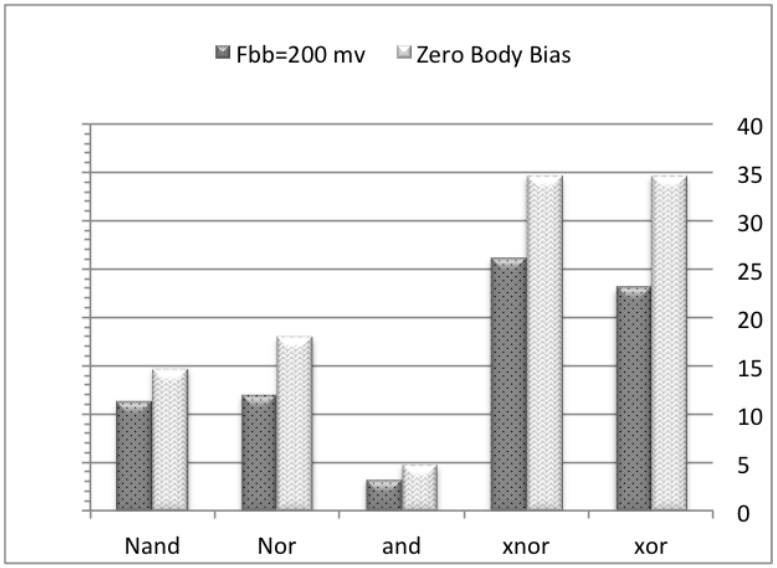


Figure 6.4: *Power Saving for Basic Gates From ZBB to ABB of 200mV*

logic; however, the power savings and speedup attained by our approach are easily justified. In static simulations, pre-computation-based ABB control logic largely reduces the leakage power in a digital VLSI circuit. The power consumption of a given circuit varies, depending on the different input vectors; for an adder or multiplier, the leakage power reduction is still approximately 20%. Thus, for a static circuit, we achieve a 20-25% reduction in total power.

We performed dynamic simulations at frequencies of 100MHz and 50MHz. Table 6.3 compares the power reduction in one clock cycle and multiple sequential cycles separately using the same input vectors for both frequencies. The simulation demonstrates that our approach has a greater effect on the multiplier than the adder; this indicates that larger circuits are more likely to benefit from our techniques than smaller ones. On the other hand, the operating frequency also has a large influence on the efficiency of power reduction using this scheme. Since the bias circuit is less sensitive when the system works at a lower frequency, the pre-computation-based ABB scheme

Clock Type	Work Frequency (MHz)	Power Reduction	
		32bit FA	16*16 MUL
One Cycle	100	0.41%	2.56%
	50	0.75%	3.49%
Sequential Cycles	100	0.51%	2.97%
	50	1.06%	3.73%

Table 6.3: *Comparison of Power Reduction for Inputs Pre-computation at Different Frequencies*

is more effective; however, the ratio of leakage power to total power is relatively small, which limits the power consumption that can be achieved using our technique.

To investigate the influence of the number of bits of MSP m on dynamic power reduction, and the corresponding increase in critical path delay, we performed some experiments on a 32-bit adder (n=32bit) and a 16x16-bit multiplier (n=16bit) at 100MHz. Table 6.4 lists the simulation results. Our results demonstrate that the FBB circuit saves more power when the number of bits of MSP m are larger than one half of the full range of the number of bits; the delay of the adder increases, while the delay of the multiplier remains almost unchanged; however, increasing m increases the power consumed by the pre-computation logic. In general, our simulation results suggest that, if a circuit has 1uW to 10uW dynamic power reduction, then the increase in critical path delay due to our body biasing scheme is 0.01ns to 0.1ns.

We simulate the FBB circuit performance on the 32 bit adder to investigate the relationship between the amount of "pwrsave" signal switching and the total power reduction in a specified period of $500ns$, as shown in Table 6.5; the experiment employs the same input pattern as before. Decreasing the amount of switching generally leads to

Cycle Type	m	32bit FA		16*16 MUL	
		PR	Delay	PR	Delay
One	n/2	0.41%	4.26%	2.56%	2.81%
	2n/3	3.96%	1.92%	3.83%	2.86%
Sequential	n/2	0.51%	3.91%	2.97%	1.39%
	2n/3	0.66%	0.84%	3.25%	1.39%

Table 6.4: *Comparison of Power Reduction Delay for One Cycle and Sequential Cycles Separately*

No. of Switching	50	26	6	2	0
PR of 32bit FA	1.27%	1.98%	2.55%	3.01%	4.13%

Table 6.5: *Relationship Between Number of Switching and Power Reduction*

greater reductions in total power consumption. As mentioned earlier, if the percentage of leakage power is as large as 40% of total power, then the upper bound of the ideal total power reduction is 4%, which is close to the simulation results for the non-switching situation.

Table 6.6 compares the power consumption of the FBB circuit, the pre-computation circuit, the adder and the multiplier; the results indicate that the pre-computation-based FBB control scheme is reasonable, but tradeoffs involving the power consumption of different components need to be considered. Moreover, if the pre-computation-based FBB control is used in conjunction with pre-computation-based guarding technology[19], then additional power could be saved; we estimate that the savings would be approximately 35.1% for the 32-bit adder.

Next we investigate the switching power consumption of the standard library

Cell Name	Power Consumption in Micro-Watts
FBB Circuit for Full Adder	5.22
Pre-computation Logic	5.82
32-bit Full Adder MSP	14
32-bit Full Adder LSP	26.3
16×16 CSA Array Multiplier	921

Table 6.6: *Total power consumption*

cells that we developed. Our library consists of INV, NAND2, NOR2, AOI21, and XORA. For each cell, we measure leakage and switching power. We performed measurements with and without body bias. Figure 6.5 shows the leakage power performance of standard cell library and Figure 6.6 shows the switching power.

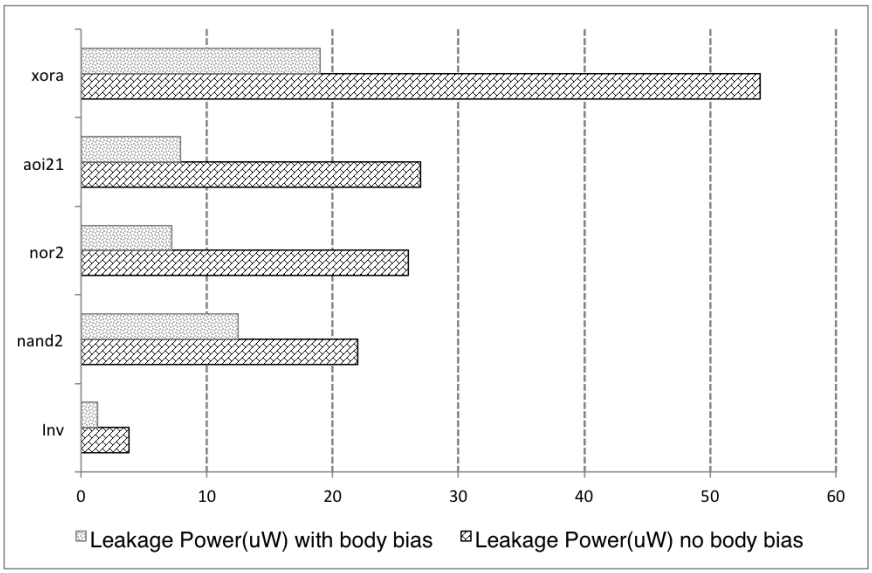


Figure 6.5: *Standard Cell Leakage Power Comparison with and without body Bias*

As expected, leakage power is reduced far more than switching power. Figure 6.7 shows the amount of power consumed by switching the body in each standard cell; body switching power consumption is nominal compared to leakage and switching power

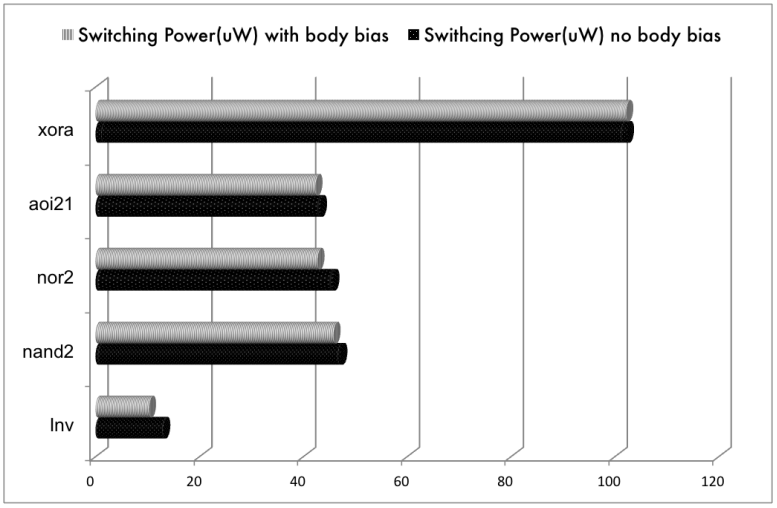


Figure 6.6: *Standard Cell Switching Power Comparison with and without body Bias*

consumption. Moreover, it is important to note that the body is not switched as often as the logic, as it is controlled by the pre-computation circuit; therefore, the impact of this particular phenomenon has a relatively small impact on total power consumption within the cell.

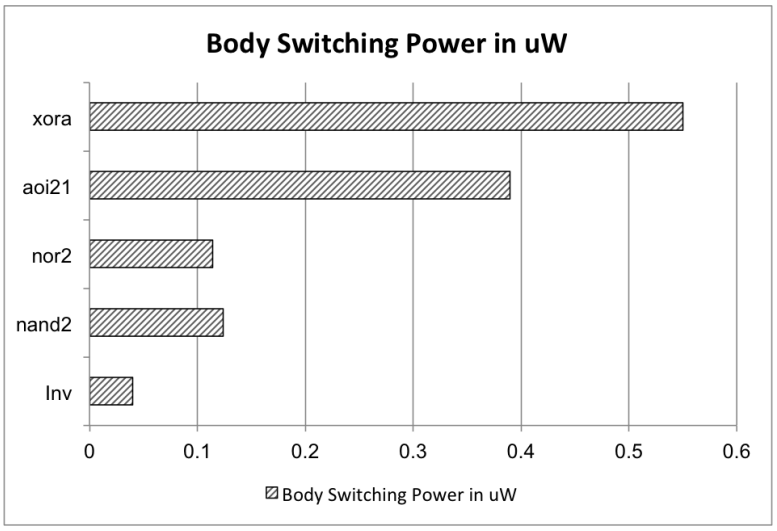


Figure 6.7: *Body Switching Power*

Although switching the body bias in each standard cell saves power, a delay

penalty is introduced; however, the amount of delay that is added depends on the complexity and structure of the standard cell design. Figure 6.8 shows that the additional delay introduced is small, at most 10% in the worst case. In contrast, Table 6.1 reported a 19% improvement in propagation delay for different benchmark circuits. Thus, we can conclude that adjusting the body bias does not adversely affect delay, when used in conjunction with our timing improvement algorithm.

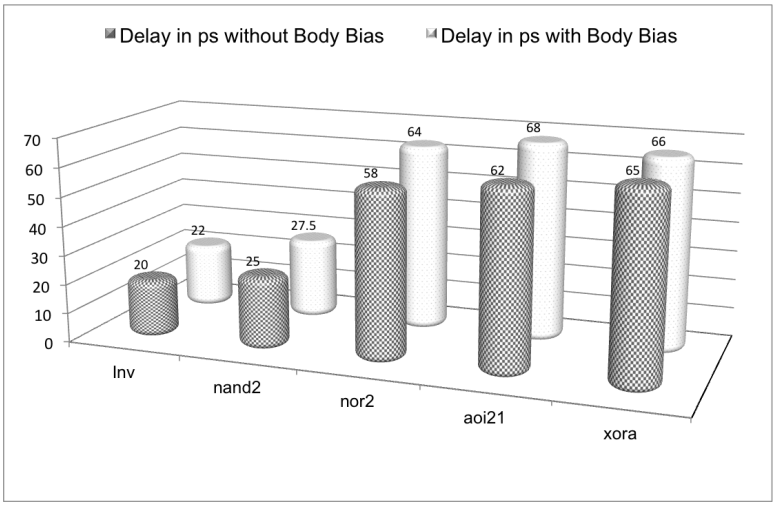


Figure 6.8: *Standard Cell Delay*

The area overhead associated with the body bias and predictor circuit is approximately 10% in terms of transistor count; however, this figure depends on the complexity and transistor count of the benchmark circuits. For a larger scheme and fast operations, intermediate charge pumpers may be required to enable a quick substrate voltage swing; this will increase the speed of the circuit at the cost of additional area. A detailed investigation of these tradeoffs is beyond the scope of this work.

6.2 Layout Implementation

CMOS circuit layout is important for successful chip fabrication; it is tedious, rigorous, and complicated, and many otherwise unforeseen issues regarding the netlist manifest themselves once the circuit is laid out. We use Tanner L-Edit 13.0 for layout in our experiments. We target a generic 65nm technology obtained from Cadence [30].

Our motivation is to demonstrate that the area penalty incurred by our scheme is not significantly. We perform standard cell layout with and without the proposed body biasing scheme, and compare the two to observe the area penalty, which is well below the 15% threshold we claimed earlier. Our results include screenshots of the laid out standard cells.

Standard cell layout is a semi-automated process; when there are no errors, the entire process is automated; however, standard library cells must be manually designed, first, in order to facilitate automatic layout. Different technology nodes and fabs use different standard library cells to implement a given circuit. Each technology node, therefore, requires its own manual layout phase. Standard cells are introduced into the place-and-route (PNR) tools to connect them properly.

During fabrication, mask generation for photolithography is a crucial step. During layout, it is important to remember that the mask generation and chemical vapor deposition (CVD) processes used during fabrication have their own constraints and limitations. Therefore, each manufacturer provides design rule checks (DRCs) for the layout to ensure that the fabricated circuit matches the layout. The PNR tools always capture the DRC efficiently, but the manually-designed standard cells must also obey DRC rules. We check each standard cell in our library manually, to verify that no DRC errors are present.

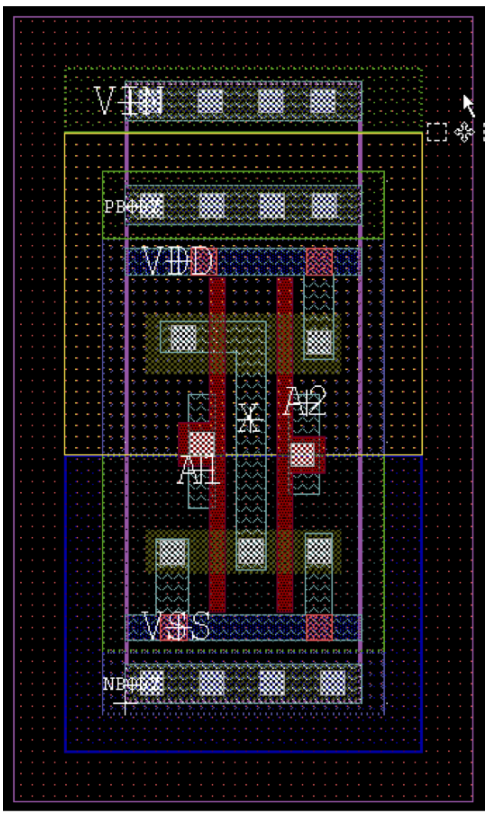


Figure 6.9: *VDD and VSS position along with body pickup in a standard CMOS circuit*

Another important phase is layout vs. schematic (LVS). After completing the layout, the circuit must be checked to ensure that if there is a one-to-one match with the netlist; this ensures that the layout is a faithful representation of the netlist. PNR tools are aware of the process and ensure that LVS checking does not generate any errors.

Standard cells maintain a standard rectangular shape, with a 2:1 aspect ratio; for complicated cells, this aspect ratio is compromised. PMOS transistors are placed at the top and NMOS transistors at the bottom. The power rails (supply and ground) are placed at the top and bottom of all cells, to facilitate side-by-side placement. Figure 6.9 shows a standard CMOS logic circuit, which demonstrates the power rail position and layout placement of PMOS, NMOS, and other layers.

We assume a double Metal single Poly technology process. Power and ground

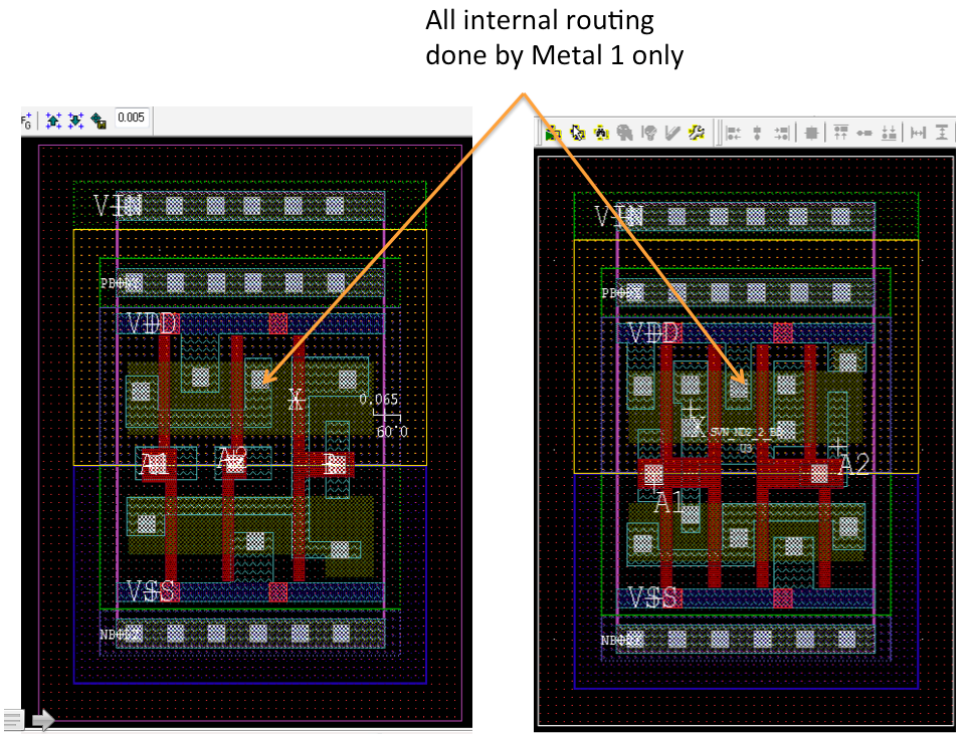


Figure 6.10: *Internal and External Metal Routing*

routing use Metal 2, while internal routing is done using Metal 1. Inter-cell connections are usually done using Metal 1 (light blue), but may use Metal 2 (dark blue) in case of conflicts. Figure 6.10 presents an example. The standard cells have a power rail at the top edge and a ground rail at the bottom edge, which reduces the routing penalty. Each standard cell has dedicated ports for input, output, pbody, nbody, and substrate pickups.

A common practice is to keep all logic cells at the same height, but to vary their width. This wastes space for smaller cells (e.g., inverters), because the DRC requirement is much less; however, it improves connectivity with other cells. Placing common points (body, power, ground) side-by-side, common wells are merged seamlessly without extra metal routing or DRC requirements. Figures 6.11 and reffig: same height snap illustrate

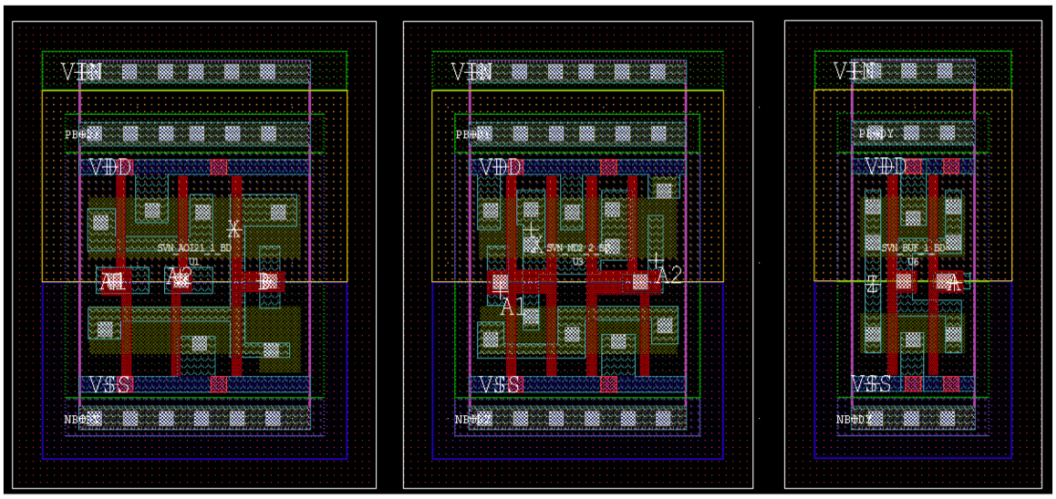


Figure 6.11: *Standard cells with same height not joined together*

this scenario.

It is common practice to provide an extra layer of metal routing (Metal 1 and 2 in parallel) for power and ground. This helps the elevated level of currents present in these nodes, builds up a higher power supply capacitance, and provides quieter ground and rails against nearby switching activity. The arrow in Figure 6.13 illustrates this approach to routing.

Poly routing is kept as small as possible due to gate capacitance. Metal is frequently used instead of poly routing, even within a cell; however, this is kept aligned as much as possible, to assist with interconnectivity between cells and to reduce area. Figure 6.14 demonstrates this approach.

Any unabated issues within standard cells, such as the usage of wider metals where there is node sharing, avoiding sharp bends for narrow wires (135 degrees instead of 90 degrees), the use of extra contact/VIA at input and output nodes, etc., will propagate throughout the chip. We have made a conscientious effort to keep these

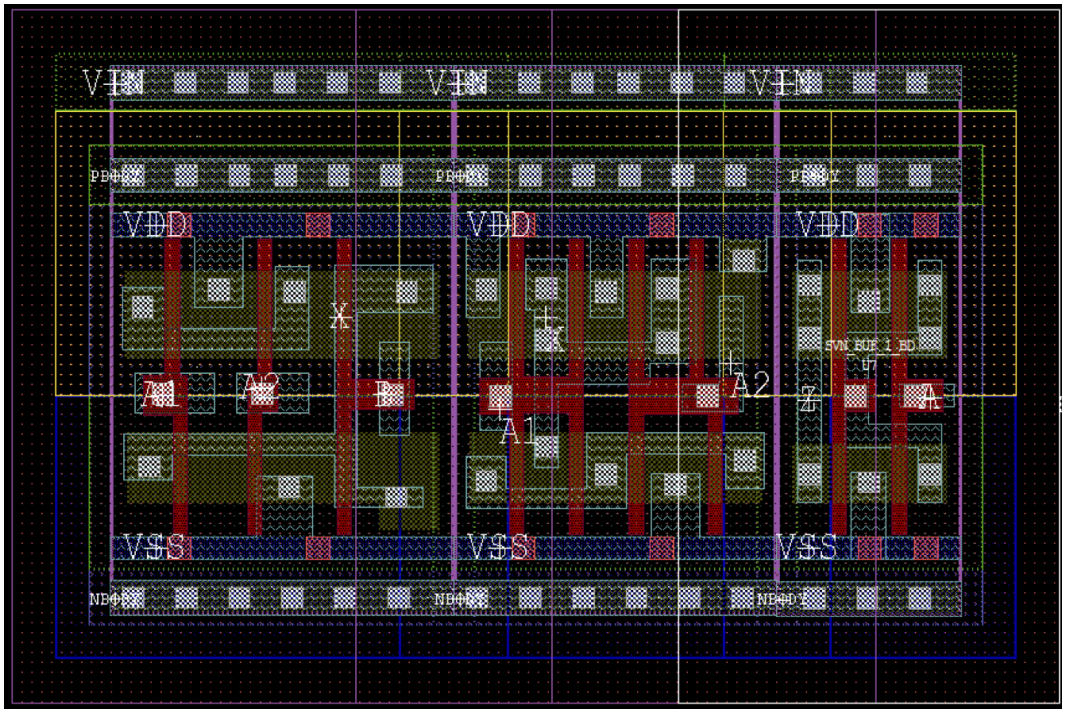


Figure 6.12: *Standard cells with same height joined together*

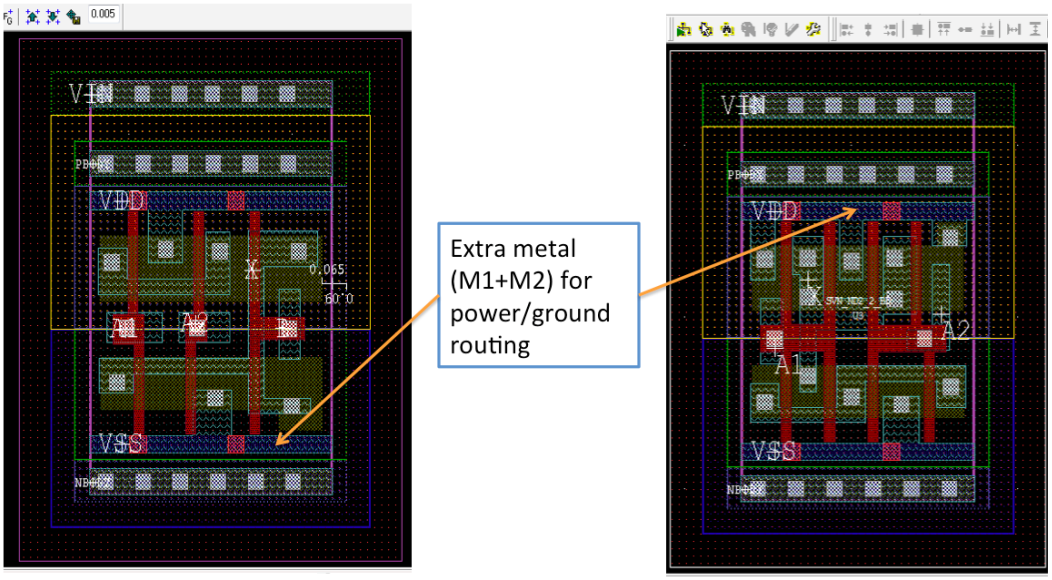


Figure 6.13: *Extra metal used in supply and ground*

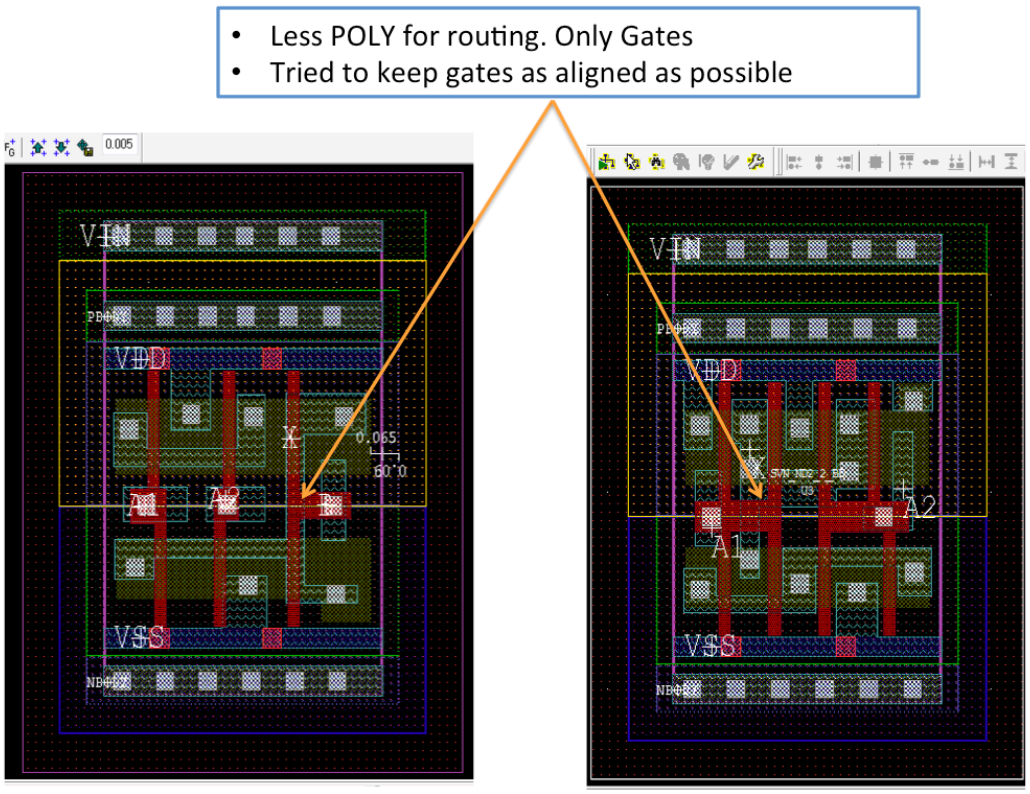


Figure 6.14: *Less Poly routing at gates*

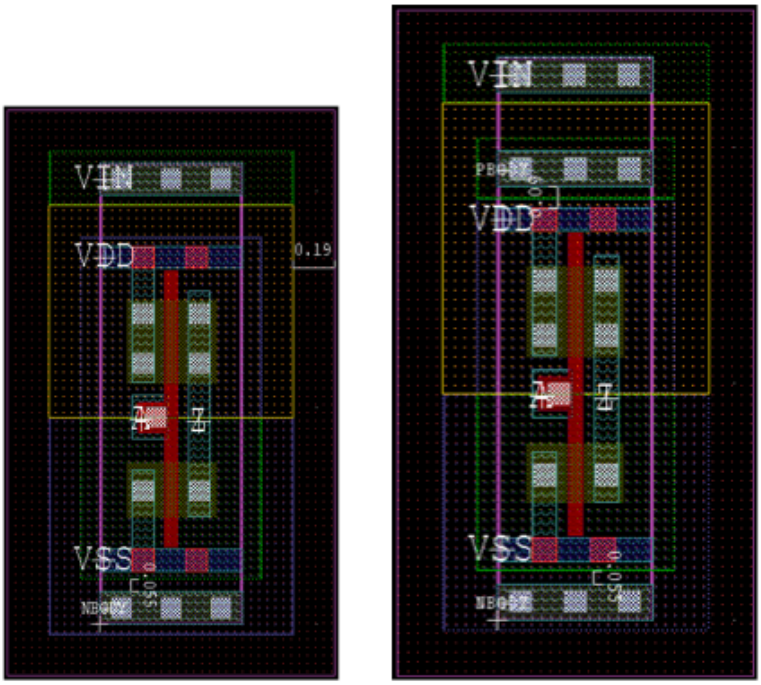


Figure 6.15: *Inverter Layout Comparison*

issues in mind when designing our standard cells. Figures 6.15 and 6.19 depict some of our standard cells with and without support for body biasing.

To summarize, Table 6.7 reports the area penalty for use of a dedicated body port out, as required by our scheme. This includes the complete layout area penalty for extra body pickup, placement, and routing. The area penalty for most of the standard cells ranges from 9-12%, plus an additional overhead of at most 1% to account for extra pbias and nbias throughout the chip after the entire circuit is placed and routed.

Lastly, we must account for the overhead of the body bias circuit. This varies, depending on which approach to the design of this circuit is taken. Chapter 3 showed that many state-of-the-art body bias circuits are used in industry, and in most cases,

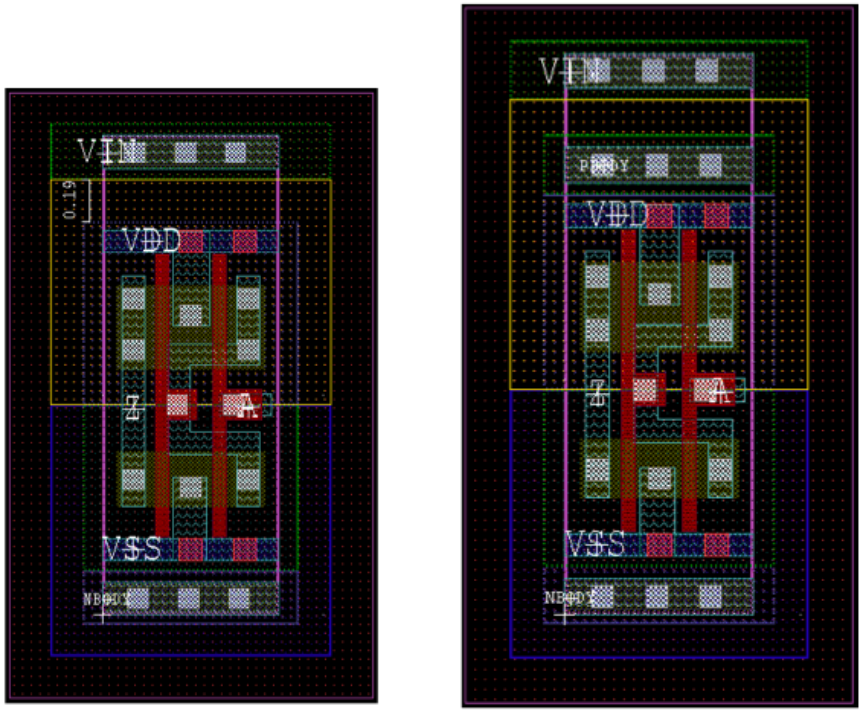


Figure 6.16: *Buffer Layout Comparison*

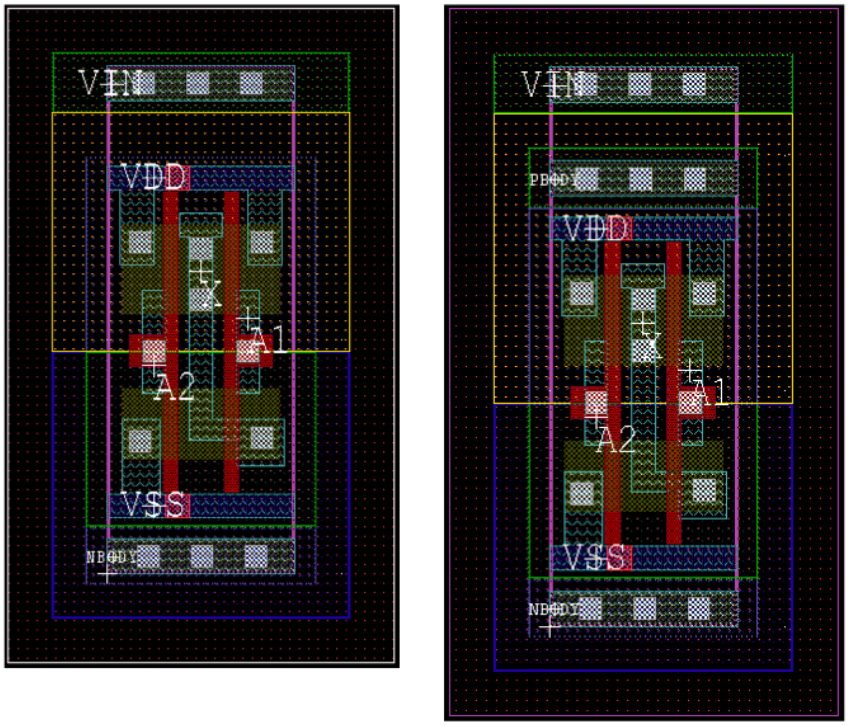


Figure 6.17: *NAND gate Layout Comparison*

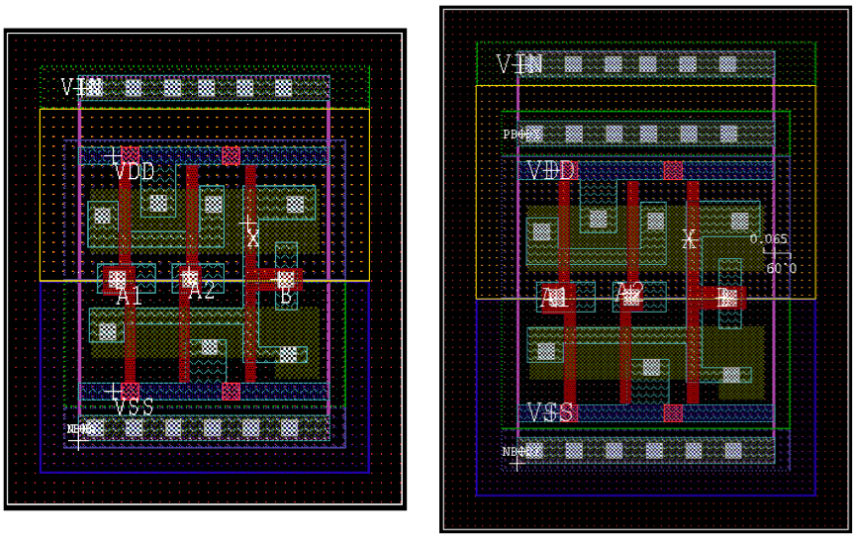


Figure 6.18: *AND-OR-Invert gate Layout Comparison*

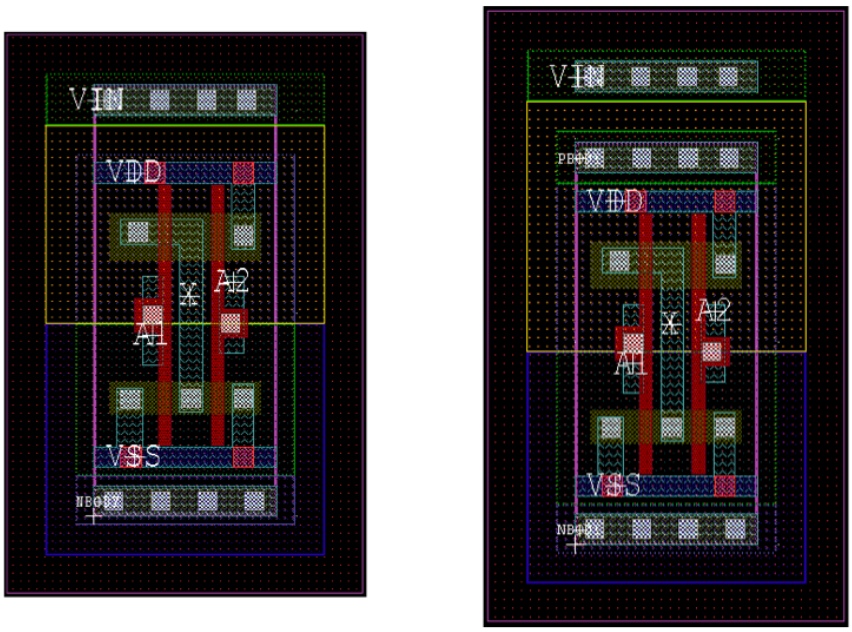


Figure 6.19: *NOR gate Layout Comparison*

Cell Name	Area with Body Port-out(µm2)	Area without Body Port-out(µm2)	% of Increament
AOI21	3.546	3.233	9.68%
BUF_1	1.908	1.74	9.66%
INV_1	1.574	1.409	11.71%
ND2_1	1.904	1.736	9.68%
ND2_2	3.332	3.038	9.68%
ND3_1	2.559	2.333	9.69%
NOR2_1	2.142	1.913	11.97%
XOR	9.044	8.246	9.68%
XNOR	10.615	9.678	9.68%

Table 6.7: *Comparison on Total Area*

the publications that introduced these circuits claim an area overhead of less than 10%. Conservatively, we assume that that area overhead will never exceed 15%. Our work uses the circuit shown in Figure 4.4, which has limited drive power, but an area penalty of less than 5%.

Chapter 7

Future Directions

The algorithm introduced in this thesis was successful for most of the test cases we tried. We believe that the general approach can be extended for further leakage minimization. If the search for control values can help to improve timing and to expand slack time, there may be some potential for adjustable body bias-aware IP core design. In particular, our approach suggests that there are benefits to create logic structures that are cascades of logic gates, with known controllable inputs. Thus, the pre-detection logic and body biasing generation circuit can be integrated into the IP core. This way, our approach can be generalized to platform-based design, in a way that eliminates the need for SoC architects to worry about the underlying details.

Another challenge is to develop algorithms that can identify control points larger than individual gates and controlling input groups larger than 1 or 2 bits in a tractable amount of time. Ideally, such an approach would sidestep the costly enumeration methods that were presented and used in this thesis. Moreover, the likelihood of saving power would be increased if we could find a larger number of controlling input vectors that can be exploited, as opposed to looking at just one or two input bits and

gates at a time.

Another research direction is to define a dynamic library for technology mapping. The concept of a dynamic cell library not new, but our goal would be based entirely on power optimization. The basic idea is to search for margins that are created during the mapping of circuits to standard cells. To reduce leakage, the use of transistors in stacks are very common. Stacking reduces the leakage loss by reducing the sub-threshold and overdrives, and stacks are present in many designs as a natural consequence. Transistors are historically stacked under the assumption of equal probability among input vectors, i.e., similar transistors are stacked [31]; however, the activity factors vary from path to path, as our research has shown. Our proposal is that the choice of the transistor should be consistent with the signal probability incidences and available time margin of the signal path.

The feasibility of the aforementioned approach is buttressed by the observation of signal probability propagation through the various paths of a circuit [32]. For example, consider a 2-input NAND gate with inputs A and B. If A has a signal probability higher than B, then reducing leakage along the path that computes A will yield a greater reduction in power than reducing leakage along the path that computes B. If B has a larger time margin than A, then B will have a greater opportunity for leakage reduction through the substrate bias. Moreover, for the same NAND gate, we can find a combination of different size and substrate bias for the transistors, and take this into account during technology mapping. Current technology mappers do not take these factors into account, and are therefore sub-optimal for power.

In fact, this is an argument for larger standard cell libraries that provide many different implementations of each cell, especially in terms of stack combinations that are consistent with power optimization as described above. Such a library would contain

the same basic gates, but with combinations of different types of MOS, some of which are optimized for speed or area, while others are optimized for different body biases. This will create a large design space that can be explored during mapping, but will yield better overall performance and power. The mapper would need to consider each gate, its input logic, and the timing margin, and take these factors into account when choosing a library cell. To bring this idea to fruition, extensive work must be done to choose the best possible combination of library cells. Of course, limitations imposed by fabrication and manufacturing capabilities must also be taken into account.

One final research direction is to examine the change in heat generation that results from body bias shifting. It would be necessary to look closely at the block level to determine the relationship between the statistical distribution of power and input logics [33]. A path-based statistical timing analysis and examination of the crucial parameters responsible for heat generation would be performed, with the overall goal being to optimize the parameters to reduce the amount of heat that would be generated in common use cases. The amount of parameter variation due to heat generation is large. It is difficult to control such a high range of variation with proposed body biasing scheme only. Other techniques may be used in conjunction with the current one to increase effectiveness.

Chapter 8

Conclusions

We have introduced a systematic design methodology for deep sub-micron CMOS technology that dynamically applies substrate bias to reduce power consumption of digital circuits. Controlling input values are detected for specific nodes on the critical path. These controlling inputs are non-critical, and certain values allow the output of the node to be computed in a manner that is independent of the critical path; the output is therefore computed much faster. When this occurs, forward body bias can be applied to transistors on the critical path to reduce power, as doing so slows down the signal propagation; the slow signal is not problematic because it does not affect the output of the node in question. The total reduction in power consumption depends on the circuit and the probability that controlling input values occur dynamically; however, we have found that the approach is successful in most of our test cases.

Based on our experience, we list the following conclusions, which comprise the outcomes of this research effort:

1. Substrate biasing can effectively reduce the power consumption of a circuit; the application of substrate biasing in conjunction with input-sensitive dynamic pre-

detection logic further increases the efficiency.

2. The reduction in power depends on the activity factor of the circuit; it is not possible to enumerate all possible delay vs. power scenarios for large circuits, but random input simulation yielded favorable results.

3. Circuits dominated by AND, OR, NAND, and NOR gates benefit significantly from our scheme; circuits dominated by XOR and XNOR gates do not, because they have no controlling inputs. Composite library gates such as OR-AND-INV or AND-OR-INV are not feasible, unless we search for two or more controlling inputs.

4. The area penalty, including the additional cost of routing, was quite small: no larger than 15% for all of our benchmarks.

5. The substrate switching delay was small based on our simulations, however, the actual delay could be much larger than our simulation values when parasitic extraction is applied to a fabricated chip.

6. Our approach is simpler and smaller (in terms of on-chip area) than complicated loop-based tracking techniques for body bias voltage; however, the more complicated approaches do have the potential to achieve further power reductions.

7. Our approach performs best on circuits with long logic depth, and a large number of non-critical inputs, which are good candidates to be controlling inputs. Our approach is less effective on circuits where all paths are balanced.

8. Temperature control using our technique is not feasible, as we do not employ loop tracking. Moreover, the range of temperature variation is much higher than the range our scheme can cover.

9. The primary challenge to implement the scheme is the careful design of the charge pump and the buffer stages that follow the body bias generator. The area penalty and power consumption increase significantly with the drive capability of this circuit.

Altogether, we believe that the advantages of this scheme significantly outweigh its drawbacks, and that our scheme can be successfully implemented in modern deep sub-micron CMOS technologies.

Bibliography

[1] Semiconductor Industry Association, "The International Technology Roadmap for Semiconductors: 2007 Update".

[2] Alan J. Drake, Kevin J. Nowka, Richard B. Brown, "Evaluation of Dynamic-Threshold Logic for Low- Power VLSI Design in 0.13m PD-SOI," IFIP International Conference on Very Large Scale Integration (VLSI-SoC), Darmstadt, Germany, Dec. 1-3, 2003, pp. 363-368. Published, 2003.

[3] Bashir M. Al-Hashimi (Editor) (2006). "System on a Chip: Next Generation Electronics. Institution of Engineering and Technology". p. 429. ISBN 0863415520.

[4] T. Kobayashi and T. Sakurai, "Self-adjusting threshold-voltage scheme (SATS) for low voltage high-speed operation", Proceedings of the IEEE Custom Integrated Circuits Conference,1994.

[5] T. Ramakrishnan and L. Kinney, "Extension of the Critical Path Tracing Algorithm", IEEE Design Automation Conf., Jun. 1990, pp. 720-723.

[6] J. Tschanz, S. Narendra, Y, Ye, B. Bloechel, S. Borkar, V. De, "Dynamic Sleep transistor and Body Bias for Active Leakage Power Control of Microprocessors", IEEE J. Solid-State Circuits, vol. 38, pp. 1838-1845,Nov. 2003.

[7] Alan J. Drake, Richard B. Brown, Jeffrey L. Burns, "Dynamic-Threshold Logic for Low-Power VLSI Design", VLSI-SOC 2003: 263.

[8] Yannis Tsividis (1999). "Operation and Modeling of the MOS Transistor (Second Edition ed.)". New York: McGraw-Hill. p. 99. ISBN 0-07-065523-5.

[9] Eric A. Vittoz (1996). "The Fundamentals of Analog Micropower Design". In Chris Toumazou, Nicholas C. Battersby, and Sonia Porta. Circuits and systems tutorials. John Wiley and Sons. p. 365372. ISBN 9780780311701

[10] Y. Oowaki, M. Noguchi, S. Takagi, D. Takashima, M. Ono, Y. Matsunaga, K. Sunouchi, H. Kawaguchiya, S. Matsuda, M. Kamoshida, T. Fuse, S. Watanabe, A. Toriumi, S. Manabe, and A. Hojo, "A Sub-0.1 m Circuit Design with Substrate-over-Biasing," in DIGEST OF TECHNICAL PAPERS OF THE SOLID STATE CIRCUITS CONFERENCE. IEEE INC, 1998, pp. 8889.

[11] S. Narendra, A. Keshavarzi, B. Bloechel, S. Borkar and V. De, "Forward body bias for microprocessors in130nm technology generation and beyond", IEEE J. Solid-State Circuits, 38(5):696701, May 2003.

[12] Berkeley Predictive Technology Model, 2009,http://ptm.asu.edu/, accessed on 01/01/2008

[13] M. Miyazaki, G. Ono, and K. Ishibashi. "A1.2-GIPS/W Microprocessor using speed-adaptive threshold-voltage CMOS with forward bias".IEEE J. Solid-State Circuits, 37(2):210217, February 2002

[14] Y. Taur. "CMOS scaling beyond 0.1m: how far can it go?" In Proc. Intl. Symp. VLSI Tech., Systems and Appl., pages 69, 1999.

[15] J. Tschanz, et. al., "Adaptive Body Bias for Reducing Impacts of Die-to-Die and Within-Die Parameter Variations on Microprocessor Frequency and Leakage," 2002 ISSCC, Digest of Technical Papers, pp 477-479

[16] M. Miyazaki, G. Ono, T. Hattori, K. Shiozawa, K. Uchiyama, and K. Ishibashi, "A 1000-MIPS/W microprocessor using speed-adaptive threshold-voltage CMOS with forward bias," in IEEE ISSCC Dig. TechPapers, 2000, pp. 420421.

[17] Volkan Kursun, Eby G. Friedman, "Multi-voltage CMOS Circuit Design", p. 61. ISBN 978-0-470-01023-5

[18] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power", IEEE Transactions on VLSI Systems, pp. 426-436, December 1994.

[19] Zhijun Huang; Ercegovac, M.D.; "On signal-gating schemes for low-power adders", Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on, 2001 Page(s): 867 -871 vol.1

[20] V. Tiwari, S. Malik and P. Ashar, "Guarded Evaluation: Pushing power management to the Logic Synthesis/Design Level", InternationalSymposium on Low Power Design, 1995.

[21] Cassondra Neau, Kaushik Roy, "Optimal Body Bias Selection for leakage Improvement and Process Compensation Over Different Technology generations", ISLPED'03, August 25-27, 2003.

[22] Xin He, Syed Al-Kadry, Afshin Abdollahi, "Adaptive Leakage Control on Body Biasing for Reducing Power Consumption in CMOS VLSI Circuit", 10th Intl Symposium on Quality Electronic Design, 2009

[23] Afshin Abddollahi, Massoud Pedarm, Farzan Fallah, Indradeep Ghosh, "Precomputation-based Guarding for Dynamic and Leakage Power Reduction", 2003 IEEE International Conference on Computer Design (ICCD'03), pp.90, 2003

[24] ABC: A System for Sequential Synthesis and Verification, http://www.eecs.berkeley.edu/ alanmi/abc/

[25] T. Kuroda, T. Fujita, S. Mita, T. Nagamatu, S. Yoshioka, K. Suzuki, F. Sano, M. Norishima, M. Murota, M. Kako, M. Kinugawa, M. Kakumu, and T. Sakurai, "A 0.9V 150MHz lOmW $4mm^2$ 2-D discrete cosine transform core processor with variable threshold-voltage scheme," IEEE J. Solid-State Circuits, vol. 31, no. 11, pp. 1770-1779, Nov. 1996.

[26] Tadahiro Kuroda and Takayasu Sakurai, "Body Biasing", Chapter 5, Leakage In Nanometer CMOS Technologies, pp-105-139, 2005, ISBN-0-387-25737-3

[27] T. Kobayashi and T. Sakurai, "Self-adjusting threshold-voltage scheme (SATS) for low-voltage high-speed operation," Proc. CICC'94, pp. 271-274, May 1994.

[28] K. Seta, H. Hara, T. Kuroda, M. Kakumu, and T. Sakurai, "50% active-power savingwithout speed degradation using standby power reduction (SPR) circuit," ISSCC Dig. Tech. Papers, pp. 318-319, Feb. 1995.

[29] H. Banba, H.Shiga, A. Umezawa, T. Miyabe, T. Tanzawa, S. Atsumi, and K. Sakui, "A CMOS band-gap reference circuit with sub-IV operation," Symposium on VLSI Circuits Dig. Tech. Papers, pp. 228-229, June 1998.

[30] Cadence Worldwide University Software Programs, 2012, http://www.cadence.com/support/university/Pages/default.aspx, accessed on 03/01/2012.

[31] Siva G. Narendra, Anantha Chandrakasan; "Leakage In Nanometer CMOS Technologies"; Chapter 2, pp-23-40; 2005, ISBN-0-387-25737-3

[32] Gary K. Yeap; "Practical Low Power Digital VLSI Design"; 1997, ISBN: 978-0-7923-8009-2

[33] Srivastava A; Sulvester D; Blaauw D; "Statistical Analysis and Optimization for VLSI: Timing and Power"; 2005, ISBN- 0-287-25738-1

Appendix A

Sample Blif File AND Mapping

This is the netlist for one of the benchmark circuits from the ISCAS 89 that was mapped during our experimentation.
First, we describe the circuit prior to mapping:

```
.model s444
.inputs G0 G1 G2
.outputs G118 G167 G107 G119 G168 G108
\text{
.latch     G11_in        G11  0
.latch     G12_in        G12  0
.latch     G13_in        G13  0
.latch     G14_in        G14  0
.latch     G15_in        G15  0
.latch     G16_in        G16  0
.latch     G17_in        G17  0
.latch     G18_in        G18  0
.latch     G19_in        G19  0
.latch     G20_in        G20  0
.latch     G21_in        G21  0
.latch     G22_in        G22  0
.latch     G23_in        G23  0
.latch     G24_in        G24  0
.latch     G25_in        G25  0
.latch     G26_in        G26  0
.latch     G27_in        G27  0
.latch     G28_in        G28  0
.latch     G29_in        G29  0
.latch     G30_in        G30  0
.latch     G31_in        G31  0

.names G12 G13 [25]
00 1
.names G11 [25] [26]
```

```
01 1
.names G14 [26] [27]
10 1
.names G0 G11 [28]
00 1
.names [27] [28] G11_in
01 1
.names G11 G12 [30]
11 1
.names G12 [30] [31]
10 1
.names G11 [30] [32]
10 1
.names [31] [32] [33]
00 1
.names G0 [33] [34]
00 1
.names [27] [34] G12_in
01 1
.names G13 [30] [36]
11 1
.names G13 [36] [37]
10 1
.names [30] [36] [38]
10 1
.names [37] [38] [39]
00 1
.names G0 [39] [40]
00 1
.names [27] [40] G13_in
01 1
.names G12 G13 [42]
11 1
.names G11 [42] [43]
11 1
.names G14 [43] [44]
11 1
.names G14 [44] [45]
10 1
.names [43] [44] [46]
10 1
.names [45] [46] [47]
00 1
.names G0 [47] [48]
00 1
.names [27] [48] G14_in
01 1
.names G31 [27] [50]
00 1
```

```
.names G16 G17 [51]
00 1
.names G15 [51] [52]
01 1
.names [50] [52] [53]
00 1
.names G18 [53] [54]
11 1
.names G15 [50] [55]
10 1
.names G15 [55] [56]
10 1
.names [50] [55] [57]
00 1
.names [56] [57] [58]
00 1
.names G0 [58] [59]
00 1
.names [54] [59] G15_in
01 1
.names G16 [55] [61]
11 1
.names G16 [61] [62]
10 1
.names [55] [61] [63]
10 1
.names [62] [63] [64]
00 1
.names G0 [64] [65]
00 1
.names [54] [65] G16_in
01 1
.names G16 [50] [67]
10 1
.names G15 [67] [68]
11 1
.names G17 [68] [69]
11 1
.names G17 [69] [70]
10 1
.names [68] [69] [71]
10 1
.names [70] [71] [72]
00 1
.names G0 [72] [73]
00 1
.names [54] [73] G17_in
01 1
.names G15 G16 [75]
```

```
11 1
.names G17 [50] [76]
10 1
.names [75] [76] [77]
11 1
.names G18 [77] [78]
11 1
.names G18 [78] [79]
10 1
.names [77] [78] [80]
10 1
.names [79] [80] [81]
00 1
.names G0 [81] [82]
00 1
.names [54] [82] G18_in
01 1
.names G20 G21 [84]
00 1
.names G19 [84] [85]
01 1
.names [54] [85] [86]
10 1
.names G22 [86] [87]
11 1
.names G19 [54] [88]
11 1
.names G19 [88] [89]
10 1
.names [54] [88] [90]
10 1
.names [89] [90] [91]
00 1
.names G0 [91] [92]
00 1
.names [87] [92] G19_in
01 1
.names G20 [88] [94]
11 1
.names G20 [94] [95]
10 1
.names [88] [94] [96]
10 1
.names [95] [96] [97]
00 1
.names G0 [97] [98]
00 1
.names [87] [98] G20_in
01 1
```

```
.names G20 [54] [100]
11 1
.names G19 [100] [101]
11 1
.names G21 [101] [102]
11 1
.names G21 [102] [103]
10 1
.names [101] [102] [104]
10 1
.names [103] [104] [105]
00 1
.names G0 [105] [106]
00 1
.names [87] [106] G21_in
01 1
.names G19 G20 [108]
11 1
.names G21 [54] [109]
11 1
.names [108] [109] [110]
11 1
.names G22 [110] [111]
11 1
.names G22 [111] [112]
10 1
.names [110] [111] [113]
10 1
.names [112] [113] [114]
00 1
.names G0 [114] [115]
00 1
.names [87] [115] G22_in
01 1
.names G2 G23 [117]
00 1
.names G2 G23 [118]
11 1
.names [117] [118] [119]
00 1
.names G0 [119] G23_in
01 1
.names G20 G21 [121]
01 1
.names G0 G23 [122]
01 1
.names [121] [122] [123]
11 1
.names G19 [123] [124]
```

```
01 1
.names G21 G22 [126]
10 1
.names G19 G20 [125]
10 1
.names G23 [125] [127]
01 1
.names [126] [127] [128]
11 1
.names G0 G24 [129]
01 1
.names [128] [129] [130]
01 1
.names [124] [130] [131]
00 1
.names G22 G23 [132]
00 1
.names [125] [132] [133]
11 1
.names G24 [133] [134]
10 1
.names G19 G20 [135]
00 1
.names G23 [135] [136]
11 1
.names G22 G23 [137]
11 1
.names [136] [137] [138]
00 1
.names G0 G21 [139]
01 1
.names [138] [139] [140]
11 1
.names [134] [140] G25_in
01 1
.names G19 G22 [142]
01 1
.names G0 [142] [143]
01 1
.names G0 [108] [144]
01 1
.names [143] [144] [145]
00 1
.names [129] [139] [146]
00 1
.names [145] [146] G26_in
11 1
.names G21 G24 [148]
00 1
```

```
.names [125] [148] [149]
11 1
.names G21 G22 [150]
00 1
.names G24 [150] [151]
01 1
.names G0 [151] [152]
00 1
.names [149] [152] [153]
01 1
.names G0 G22 [154]
01 1
.names [135] [154] [155]
11 1
.names [146] [155] [156]
10 1
.names [131] [156] [157]
00 1
.names G17 [157] [158]
01 1
.names [131] [156] [159]
10 1
.names [158] [159] G28_in
00 1
.names [122] [126] [161]
11 1
.names G21 G22 [162]
01 1
.names G0 [162] [163]
01 1
.names [161] [163] [164]
00 1
.names G20 [164] [165]
00 1
.names G19 [165] [166]
01 1
.names [130] [166] [167]
00 1
.names [131] [167] [168]
00 1
.names G17 [168] [169]
01 1
.names [131] [167] [170]
10 1
.names [169] [170] G29_in
00 1
.names G20 G21 [172]
10 1
.names G0 G24 [173]
```

```
00 1
.names [172] [173] [174]
11 1
.names G19 [174] G30_in
11 1
.names G1 G31 [176]
00 1
.names G1 G31 [177]
11 1
.names [176] [177] [178]
00 1
.names G0 [178] G31_in
01 1
.names [131] G24_in
0 1
.names [153] G27_in
0 1
.names G27 G118
1 1
.names G29 G167
0 1
.names G25 G107
1 1
.names G28 G119
0 1
.names G30 G168
1 1
.names G26 G108
1 1
.end
```

After mapping to our defined library, the blif representation is as follows:

```
.model C:\abc70930\examples\s444
.inputs G0 G1 G2 G11 G12 G13 G14 G15 G16 G17 G18 G19 G20 G21 G22 G23 G24 \
 G25 G26 G27 G28 G29 G30 G31
.outputs G118 G167 G107 G119 G168 G108 n19 n24 n29 n34 n39 n44 n49 n54 n59 \
 n64 n69 n74 n79 n84 n89 n94 n99 n104 n109 n114 n119
.default_input_arrival 0 0
.gate inv1  a=G29 O=G167
.gate inv1  a=G28 O=G119
.gate inv1  a=G11 O=n53
.gate inv1  a=G14 O=n54_1
.gate nor2  a=G13 b=G12 O=n55
.gate aoi21 a=n55 b=n53 c=n54_1 O=n56
.gate nor3  a=n56 b=G11 c=G0 O=n19
.gate xorb  a=G12 b=G11 O=n58
.gate nor2  a=n56 b=G0 O=n59_1
.gate nand2 a=n59_1 b=n58 O=n60
.gate inv1  a=n60 O=n24
```

```
.gate inv1  a=G13 O=n62
.gate nand2 a=G12 b=G11 O=n63
.gate xorb  a=n63 b=n62 O=n64_1
.gate nand2 a=n64_1 b=n59_1 O=n65
.gate inv1  a=n65 O=n29
.gate nand3 a=G13 b=G12 c=G11 O=n67
.gate xorb  a=n67 b=n54_1 O=n68
.gate nand2 a=n68 b=n59_1 O=n69_1
.gate inv1  a=n69_1 O=n34
.gate inv1  a=G0 O=n71
.gate inv1  a=G12 O=n72
.gate nand3 a=n62 b=n72 c=n53 O=n73
.gate aoi21 a=n73 b=G14 c=G31 O=n74_1
.gate inv1  a=G15 O=n75
.gate inv1  a=G16 O=n76
.gate inv1  a=G17 O=n77
.gate nand3 a=n77 b=n76 c=n75 O=n78
.gate nand2 a=n78 b=G18 O=n79_1
.gate oai21 a=n79_1 b=n74_1 c=n71 O=n80
.gate nand2 a=n74_1 b=G15 O=n81
.gate inv1  a=G31 O=n82
.gate nand2 a=n73 b=G14 O=n83
.gate nand2 a=n83 b=n82 O=n84_1
.gate nand2 a=n84_1 b=n75 O=n85
.gate aoi21 a=n85 b=n81 c=n80 O=n39
.gate oai21 a=n74_1 b=n75 c=G16 O=n87
.gate nand3 a=n84_1 b=n76 c=G15 O=n88
.gate aoi21 a=n88 b=n87 c=n80 O=n44
.gate nand2 a=G16 b=G15 O=n90
.gate oai21 a=n90 b=n74_1 c=G17 O=n91
.gate inv1  a=n90 O=n92
.gate nand3 a=n92 b=n84_1 c=n77 O=n93
.gate aoi21 a=n93 b=n91 c=n80 O=n49
.gate nand2 a=n92 b=G17 O=n95
.gate oai21 a=n95 b=n74_1 c=G18 O=n96
.gate nor2  a=G18 b=n77 O=n97
.gate nand3 a=n97 b=n92 c=n84_1 O=n98
.gate aoi21 a=n98 b=n96 c=n80 O=n54
.gate inv1  a=G19 O=n100
.gate inv1  a=G20 O=n101
.gate inv1  a=G21 O=n102
.gate nand3 a=n102 b=n101 c=n100 O=n103
.gate nand4 a=n78 b=G22 c=G18 d=n103 O=n104_1
.gate oai21 a=n104_1 b=n74_1 c=n71 O=n105
.gate oai21 a=n79_1 b=n74_1 c=G19 O=n106
.gate inv1  a=n79_1 O=n107
.gate nand3 a=n107 b=n84_1 c=n100 O=n108
.gate aoi21 a=n108 b=n106 c=n105 O=n59
.gate nand3 a=n78 b=G19 c=G18 O=n110
```

```
.gate oai21 a=n110 b=n74_1 c=G20 O=n111
.gate nor2  a=G20 b=n100 O=n112
.gate nand3 a=n112 b=n107 c=n84_1 O=n113
.gate aoi21 a=n113 b=n111 c=n105 O=n64
.gate nand2 a=G20 b=G19 O=n115
.gate inv1  a=n115 O=n116
.gate nand3 a=n116 b=n78 c=G18 O=n117
.gate oai21 a=n117 b=n74_1 c=G21 O=n118
.gate nor2  a=n115 b=G21 O=n119_1
.gate nand3 a=n119_1 b=n107 c=n84_1 O=n120
.gate aoi21 a=n120 b=n118 c=n105 O=n69
.gate nand3 a=G21 b=G20 c=G19 O=n122
.gate inv1  a=n122 O=n123
.gate nand3 a=n123 b=n78 c=G18 O=n124
.gate oai21 a=n124 b=n74_1 c=G22 O=n125
.gate nor2  a=n122 b=G22 O=n126
.gate nand3 a=n126 b=n107 c=n84_1 O=n127
.gate aoi21 a=n127 b=n125 c=n105 O=n74
.gate inv1  a=G2 O=n129
.gate inv1  a=G23 O=n130
.gate oai21 a=n130 b=n129 c=n71 O=n131
.gate aoi21 a=n130 b=n129 c=n131 O=n79
.gate nor2  a=G23 b=G22 O=n133
.gate nand4 a=G21 b=n101 c=G19 d=n133 O=n134
.gate inv1  a=G24 O=n135
.gate nor2  a=n135 b=G0 O=n136
.gate nand2 a=n136 b=n134 O=n137
.gate nor2  a=G20 b=G19 O=n138
.gate nand4 a=G23 b=G21 c=n71 d=n138 O=n139
.gate nand2 a=n139 b=n137 O=n84
.gate aoi21 a=n133 b=n112 c=n135 O=n141
.gate nand2 a=n138 b=G23 O=n142
.gate nand2 a=G23 b=G22 O=n143
.gate nand4 a=n142 b=G21 c=n71 d=n143 O=n144
.gate nor2  a=n144 b=n141 O=n89
.gate nor2  a=G24 b=G21 O=n146
.gate inv1  a=n146 O=n147
.gate nand2 a=n116 b=n71 O=n148
.gate nand2 a=G22 b=n71 O=n149
.gate oai21 a=n149 b=G19 c=n148 O=n150
.gate aoi21 a=n147 b=n71 c=n150 O=n94
.gate aoi21 a=n146 b=n112 c=G0 O=n152
.gate oai21 a=n147 b=G22 c=n152 O=n99
.gate nand2 a=n84 b=G17 O=n154
.gate inv1  a=n138 O=n155
.gate oai22 a=n146 b=G0 c=n155 d=n149 O=n156
.gate nand2 a=n156 b=n154 O=n104
.gate nand3 a=G22 b=n102 c=n71 O=n158
.gate inv1  a=G22 O=n159
```

```
.gate nand4 a=n159 b=G21 c=n71 d=G23 O=n160
.gate nand2 a=n160 b=n158 O=n161
.gate nand2 a=n161 b=n138 O=n162
.gate nand2 a=n162 b=n137 O=n163
.gate nand2 a=n163 b=n154 O=n109
.gate nor2  a=n148 b=n147 O=n114
.gate inv1  a=G1 O=n166
.gate oai21 a=n82 b=n166 c=n71 O=n167
.gate aoi21 a=n82 b=n166 c=n167 O=n119
.gate buf   a=G27 O=G118
.gate buf   a=G25 O=G107
.gate buf   a=G30 O=G168
.gate buf   a=G26 O=G108
.end
```

Appendix B

Simulation Setup Sample

Here, we present a sample spice file to illustrate how the standard library cells were simulated to obtain power and delay data.

```
**************************test_bias_aoi21.sp**********************
**********************
.GLOBAL gnd!

.SUBCKT AOI21_1 X  A1 A2 B pbias nbias vd1
MPB X B N1 pbias pmos L=0.045U W=0.225U M=1
MPA2 N1 A2 vd1 pbias pmos L=0.045U W=0.225U M=1
MPA1 N1 A1 vd1 pbias pmos L=0.045U W=0.225U M=1
MNB X B gnd! nbias nmos L=0.045U W=0.225U M=1
MNA2 N2 A2 gnd! nbias nmos L=0.045U W=0.225U M=1
MNA1 X A1 N2 nbias nmos L=0.045U W=0.225U M=1
.ENDS

x1 o1 vg1 vg2 vg3 pbias nbias AVDD / AOI21_1

x3 o2 o1 vdd gnd! pbias nbias vdd / AOI21_1
x4 o2 o1 vdd gnd! pbias nbias vdd / AOI21_1
x5 o2 o1 vdd gnd! pbias nbias vdd / AOI21_1
x6 o2 o1 vdd gnd! pbias nbias vdd / AOI21_1

*MPA2 X2 X vd1 pbias pmos L=0.045U W=0.460U
*MNA2 X2 X gnd! nbias nmos L=0.045U W=0.180U

vvd1 VDD 0 0.7
vvg1 vg1 0 pwl(0 0 360p 0 360.001p 0.7 720p 0.7
+720.001p 0 1080p 0 1080.001p 0.7 1440p 0.7 1440.001p 0 1800p 0)
vvg2 vg2 0 0.7
vvg3 vg3 0 0
*vg2 A2 0 0
rs1 AVDD vdd 1

*XI1 in1 in2 pbias nbias inv M=1
*XI2 in2 in3 pbias nbias inv M=1
*RL in3 0 1k
```

```
vp1 vdd pbias 0$pwl(0 0 360p 0 360.001p 0.2 720p 0.2 720.001p 0
+1080p 0 1080.001p 0.2 1440p 0.2 1440.001p 0 1800p 0)
vn1 nbias gnd! 0$pwl(0 0 360p 0 360.001p 0.2 720p 0.2 720.001p 0
+1080p 0 1080.001p 0.2 1440p 0.2 1440.001p 0 1800p 0)

*.option runlvl=0
*.OPTIONS POST PROBE
.probe i(rs)
.MODEL pmos pmos LEVEL=54
.MODEL nmos nmos LEVEL=54
.OP
*.DC Vnbias 0 2 0.00001
.tran .01p 2n
.PRINT TRAN POWER I(Rs)
*.PRINT DC I(r0) I(r1) I(r2) I(r3) V(op) v(op2) I(Rs)
*Vin vin gnd! DC 0
*Vvcc vcc gnd! DC 0.9
*Vnbias vnbias gnd! 0
.MEAS TRAN PAVG AVG power FROM = 360ps TO = 1079ps

.temp 100

.END

* PTM 45nm Metal Gate / High-K

.model  nmos  nmos level=54 +version = 4.0
*****model omitted for simplicity**********

******************end of file*******************************
```

Appendix C

Coding Implementation

The following piece of code implements the main algorithm presented in the thesis; it was implemented in Berkeley's ABC tool [20], which is open source and performs logic synthesis and validation.

```
\footnotesize
/************** [JK] start************************/
// print the network in this method.
int Abc_PrintMap(Abc_Frame_t * pAbc){

FILE* fp;
Abc_Obj_t * pNode, * pDriver, * pWorstOut;
Vec_Ptr_t * vNodes;
Vec_Ptr_t * vCritical;
Abc_Ntk_t * pNtk;
int i, DelayCur, fForward=1, Direction=0;
Abc_Time_t * pTime;
    float tArrivalMax;
Mio_PinPhase_t PinPhase;
    Mio_Pin_t * pPin;
Abc_ManTime_t *   pManTime;

if(pAbc->pNtkCur == NULL)
return 1;
fp = fopen("critical_output.txt", "w");

//print network name
fprintf(fp, "Network Name: %s\n\n", pAbc->pNtkCur->pName);

assert( Abc_NtkIsMappedLogic(pAbc->pNtkCur) );
    Abc_NtkTimePrepare( pAbc->pNtkCur );
    vNodes = Abc_NtkDfs( pAbc->pNtkCur, 1 );

tArrivalMax = -ABC_INFINITY;
fprintf(fp, "********The List of Primary Outputs********\n\n");
for ( i = 0; i < vNodes->nSize; i++ )
{
pNode=vNodes->pArray[i];
```

```
Abc_NodeDelayTraceArrival_AA(pNode);
if(pNode->fMarkA==1)
continue;//if it is not a primary output do nothing

pTime   = Abc_NodeArrival(pNode);
        fprintf(fp, "%s\t\t%f\n",Abc_ObjName(pNode),pTime->Worst);
if ( tArrivalMax < pTime->Worst )
{
tArrivalMax = pTime->Worst;
pWorstOut=pNode;
}
}
Vec_PtrFree( vNodes );

    // get the latest arrival times

    pNode = pWorstOut;
fprintf(fp, "\n\n***********End of POs*********\n");
fprintf(fp, "\n\n The critical path is: \n");
while(pNode!=NULL)
{
pTime   = Abc_NodeArrival(pNode);
fprintf(fp, " ====> %s",Abc_ObjName(pNode));
printf(" ====> %s,[%f]",Abc_ObjName(pNode),pTime->Worst);

if(pTime->Fall<pTime->Rise)
{
if ( pNode->fMarkC == 0 )
(
pNode= pNode->pWorstFaninRise;
}
else
(
pNode= pNode->pWorstFaninFall;
Direction=0;
}
}
else
(
if ( pNode->fMarkC == 0 )
(
pNode= pNode->pWorstFaninFall;
}
else
(
pNode= pNode->pWorstFaninRise;
Direction=1;
}
}
```

```
}

fclose(fp);
return 0;
}

//method for show_map to create the map first and then call print routine.
int Abc_CommandMapAndPrint( Abc_Frame_t * pAbc, int argc, char ** argv )
int fError;
fError = Abc_CommandMap(pAbc, argc, argv );
if(Abc_PrintMap(pAbc)==0)
printf("\nWrite to the file (critical_output.txt) is successfull\n");
return fError;
}

/************************** [JK] end*********************************/

/***************************[AA] start*********************************/
void Abc_NodeDelayTraceArrival_AA( Abc_Obj_t * pNode  )
(
Abc_Obj_t * pFanin;
    Abc_Time_t * pTimeIn, * pTimeOut;
    float tDelayBlockRise, tDelayBlockFall;
    Mio_PinPhase_t PinPhase;
    Mio_Pin_t * pPin;
    Abc_ManTime_t *   pManTime;
int i;
    // start the arrival time of the node
    pTimeOut = Abc_NodeArrival(pNode);
    pTimeOut->Rise = pTimeOut->Fall = -ABC_INFINITY;
    // go through the pins of the gate
    pPin = Mio_GateReadPins(pNode->pData);
//pNode->fMarkA=0;
    Abc_ObjForEachFanin( pNode, pFanin, i )
    (
pFanin->fMarkA=1;
        pTimeIn = Abc_NodeArrival(pFanin);
        // get the interesting parameters of this pin
        PinPhase = Mio_PinReadPhase(pPin);
tDelayBlockRise = (float)Mio_PinReadDelayBlockRise( pPin );
        tDelayBlockFall = (float)Mio_PinReadDelayBlockFall( pPin );
        // compute the arrival times of the positive phase
        if ( PinPhase != MIO_PHASE_INV )  // NONINV phase is present
        (
pNode->fMarkC = 0;
            if ( pTimeOut->Rise < pTimeIn->Rise + tDelayBlockRise )
(
pTimeOut->Rise = pTimeIn->Rise + tDelayBlockRise;
pNode->pWorstFaninRise=pFanin;
```

```
}
            if ( pTimeOut->Fall < pTimeIn->Fall + tDelayBlockFall )
(
pTimeOut->Fall = pTimeIn->Fall + tDelayBlockFall;
pNode->pWorstFaninFall=pFanin;
}
}
        if ( PinPhase != MIO_PHASE_NONINV )  // INV phase is present
        (
pNode->fMarkC = 1;
            if ( pTimeOut->Rise < pTimeIn->Fall + tDelayBlockRise )
(
pTimeOut->Rise = pTimeIn->Fall + tDelayBlockRise;
pNode->pWorstFaninRise=pFanin;
}
            if ( pTimeOut->Fall < pTimeIn->Rise + tDelayBlockFall )
(
pTimeOut->Fall = pTimeIn->Rise + tDelayBlockFall;
pNode->pWorstFaninFall=pFanin;
}
        }
        pPin = Mio_PinReadNext(pPin);
    }
    pTimeOut->Worst = ABC_MAX( pTimeOut->Rise, pTimeOut->Fall );
}

/*************************[AA] end*****************************/

//////[JK] FOR IDENTIFYING THE EFFECT OF SIDE INPUTS/////////
void Abc_CheckLogic( Abc_Frame_t * pAbc )
(
FILE* fp;
Abc_Obj_t * pNode, * pDriver, * pWorstOut;
Vec_Ptr_t * vNodes, * vCritical;

int i, DelayCur, fForward=1, Direction=0;
Abc_Time_t * pTimeIn, * pTimeOut, * pTime;
    float Min_tArrivalMax=2, tArrivalMax = -ABC_INFINITY;
Mio_PinPhase_t PinPhase;
    Mio_Pin_t * pPin;
Abc_ManTime_t *   pManTime;

fp = fopen("critical_nodepath.txt", "w");
 Abc_NtkTimePrepare( pAbc->pNtkCur );
    vNodes = Abc_NtkDfs( pAbc->pNtkCur, 1 );
assert( Abc_NtkIsMappedLogic(pAbc->pNtkCur) );

/////get new worst timing//////
Min_tArrivalMax = PropLogic(pAbc);
```

```
//fprintf(fp,"%d\n\n",vNodes->nSize);
printf("There are %d internal nodes apart from PIs
\n\nPrimary outputs are\n",vNodes->nSize);
for ( i = 0; i < vNodes->nSize; i++ )
(
pNode=vNodes->pArray[i];

if(pNode->fMarkA==1)
continue;
Abc_NodeDelayTraceArrival_AA(pNode);
pTime   = Abc_NodeArrival(pNode);
        //fprintf(fp, "%s\t\t%f\t%f\n",Abc_ObjName(pNode)
        ,pTime->Worst,pTimeOut->Worst);
//printf("%s\t\t%f\t%f\n",Abc_ObjName(pNode)
,pTime->Worst,pTimeOut->Worst);
if ( tArrivalMax < pTime->Worst )
(
tArrivalMax = pTime->Worst;
pWorstOut=pNode;

}

}
pNode = pWorstOut;
fprintf(fp, "\n\n***********End of POs*************\n");
fprintf(fp, "\n\n The critical path is: \n");
printf("\n\n The critical path is: \n");
while(pNode!=NULL)
(
pTime   = Abc_NodeArrival(pNode);
fprintf(fp, " ====> %s,[%f]",Abc_ObjName(pNode),pTime->Worst);
printf(" ===> %s,[%f]",Abc_ObjName(pNode),pTime->Worst);
//pTime   = Abc_NodeArrival(pNode);
if(pTime->Fall<pTime->Rise)
(
if ( pNode->fMarkC == 0 )
(
pNode= pNode->pWorstFaninRise;
}
else
(
pNode= pNode->pWorstFaninFall;
Direction=0;
}
}
else
(
if ( pNode->fMarkC == 0 )
(
```

```
pNode= pNode->pWorstFaninFall;
}
else
(
pNode= pNode->pWorstFaninRise;
Direction=1;
}
}
}
fprintf(fp, " \n\n %f",Min_tArrivalMax);
printf(" \n\nwith constant input this reduces to %f\n\n",Min_tArrivalMax);
fclose(fp);
}

char ** Min_PI1,** Min_PI2,** Min_PO,** Min_PO_LOCAL;
int Min_Log1, Min_Log2;

//////////////////////////[jk]////////////////////////
float PropLogic(Abc_Frame_t * pAbc)
(
Abc_Obj_t * pFanout, * pFanin, * pNode, *pPi1,*pPi2;
Abc_Time_t * pTimeIn, * pTimeOut, * pTime;
Vec_Ptr_t * vNodes, * vCritical;
float tDelayBlockRise, tDelayBlockFall,tArrivalMax = -ABC_INFINITY;
float Min_tArrivalMax = ABC_INFINITY ;
Mio_PinPhase_t PinPhase;
//Mio_Gate_t * pRoot;
Mio_Pin_t * pPin;
Abc_ManTime_t *   pManTime;
int i,PI_iter1,PI_iter2,Node_iter, Logic1, Logic2;

vNodes = Abc_NtkDfs( pAbc->pNtkCur, 1 );
Abc_ClearTimeLogic(pAbc);

Abc_NtkForEachPi( pAbc->pNtkCur, pPi1, PI_iter1)
%//iterating over each primary input of the circuit
(
//printf("\nPI1=%s\n",Abc_ObjName(pPi1));
Logic1=0;
while(Logic1<2) //going the whole process first with 0 and then with 1
(
pPi1->fMarkB = Logic1;
Abc_NtkForEachPi( pAbc->pNtkCur, pPi2, PI_iter2)
%//iterating over each primary input of the circuit
(
if(pPi1!=pPi2)
( //printf("\nPI2=%s\n",Abc_ObjName(pPi2));
Logic2=0;
while (Logic2<2)
```

```
(
pPi2->fMarkB = Logic2;
%//setting the logic of the current PI to 0 or 1
//printf("Logic = %d\t",Logic);

for ( Node_iter = 0; Node_iter < vNodes->nSize; Node_iter++ )
%//Node iteration for logic propagation
(
pNode=vNodes->pArray[Node_iter];
pNode->fMarkB=2;

pTimeOut= Abc_NodeArrival(pNode);

pPin = Mio_GateReadPins(pNode->pData);

Abc_ObjForEachFanin( pNode, pFanin, i )
(
pFanin->fMarkA=1;
%// flag for this loop for time adjust is visited
%and for POs this is not visited as Fanin
Set_logic_based_delay(pNode,pPin,pFanin);
pTimeOut = Abc_NodeArrival(pNode);
pTimeOut->Worst = ABC_MAX( pTimeOut->Rise, pTimeOut->Fall );
pPin = Mio_PinReadNext(pPin);

}//end of fanin iteration

if(pTimeOut->Worst>tArrivalMax)
(
tArrivalMax = pTimeOut->Worst;
Min_PO_LOCAL=Abc_ObjName(pNode);
}
//printf("%s\t%f\n",Abc_ObjName(pNode),pTimeOut->Worst);

}//end of node iteration


if(Min_tArrivalMax >= tArrivalMax)
%//remember the best critical condition
(
Min_tArrivalMax = tArrivalMax;
Min_PI1=Abc_ObjName(pPi1);
Min_PI2=Abc_ObjName(pPi2);
Min_PO=Min_PO_LOCAL;
Min_Log1=Logic1;
Min_Log2=Logic2;
}
//printf("Logic2 = %d\t",Logic2);
Logic2++;//changing the logic2, occurring only once
```

```
}//end of logic2 iteration
}
}//end of PI2 iteration
//printf("Logic1 = %d\t",Logic1);
Logic1++; //changing the logic1, occurring only once
}//end of logic1 iteration

}//end of PI1 iteration

printf("\n\n Max Arrival Time = %f\n
Primary Input Id = %s\t%s\n
Logic of this node = %d\t%d\n\n
and the output node is %s\n"
   ,Min_tArrivalMax,Min_PI1,Min_PI2,Min_Log1,Min_Log2,Min_PO);
return Min_tArrivalMax;
}

void Abc_ClearTimeLogic(Abc_Frame_t * pAbc)
(
int i;
Vec_Ptr_t * vNodes;
Abc_Obj_t * pNode;
Abc_Time_t * pTimeOut;
Abc_NtkTimePrepare( pAbc->pNtkCur );
    vNodes = Abc_NtkDfs( pAbc->pNtkCur, 1 );
assert( Abc_NtkIsMappedLogic(pAbc->pNtkCur) );

////clearing everything
for ( i = 0; i < vNodes->nSize; i++ )
(
pNode=vNodes->pArray[i];
pTimeOut = Abc_NodeArrival(pNode);
pTimeOut->Rise = pTimeOut->Fall = ABC_INFINITY;
pNode->fMarkB=2;

}
////////////////////////////
}

void Abc_ctrl_logic(Abc_Obj_t * pNode,Abc_Obj_t * pFanin)
(
Mio_Gate_t * pRoot;
pRoot=pNode->pData;

if(pFanin->fMarkB==0 && !strncmp( pRoot->pName, "and", 3 ))
pNode->fMarkB=0;
else if ( pFanin->fMarkB==1 && !strncmp( pRoot->pName, "or", 2 ) )
pNode->fMarkB=1;
else if ( pFanin->fMarkB==0 && !strncmp( pRoot->pName, "nand", 4 ) )
```

```
pNode->fMarkB=1;
else if ( pFanin->fMarkB==1 && !strncmp( pRoot->pName, "nor", 3 ))
pNode->fMarkB=0;

else if ( pFanin->fMarkB==0 && !strncmp( pRoot->pName, "inv", 3 ))
pNode->fMarkB=1;
else if ( pFanin->fMarkB==1 && !strncmp( pRoot->pName, "inv", 3 ))
pNode->fMarkB=0;

else if ( !strncmp( pRoot->pName, "buf", 3 ) )
pNode->fMarkB = pFanin->fMarkB ;
}

void Set_logic_based_delay
(Abc_Obj_t *pNode,Mio_Pin_t *pPin,Abc_Obj_t *pFanin)
{
float tDelayBlockRise = (float)Mio_PinReadDelayBlockRise( pPin );
float tDelayBlockFall = (float)Mio_PinReadDelayBlockFall( pPin );
Mio_PinPhase_t PinPhase = Mio_PinReadPhase(pPin);
Abc_Time_t *pTimeIn,* pTimeOut;

pTimeIn= Abc_NodeArrival(pFanin);
pTimeOut = Abc_NodeArrival(pNode);
pTimeOut->Rise = pTimeOut->Fall = -ABC_INFINITY;

Abc_ctrl_logic(pNode,pFanin);
//pTimeOut = Abc_NodeArrival(pFanin);
if (pNode->fMarkB !=2)
{
if ( PinPhase != MIO_PHASE_INV )  // NONINV phase is present
{
pNode->fMarkC = 0;

if(pFanin->fMarkB==1)
{
pTimeOut->Rise = pTimeIn->Rise + tDelayBlockRise;
pTimeOut->Fall = -ABC_INFINITY;
pNode->pWorstFaninRise=pFanin;
}
else if(pFanin->fMarkB==0)
{
pTimeOut->Fall = pTimeIn->Fall + tDelayBlockFall;
pTimeOut->Rise = -ABC_INFINITY;
pNode->pWorstFaninFall=pFanin;
}
else{}
}
if ( PinPhase != MIO_PHASE_NONINV )  // INV phase is present
{
```

```
pNode->fMarkC = 1;

if(pFanin->fMarkB==0)
{
pTimeOut->Rise = pTimeIn->Fall + tDelayBlockRise;
pTimeOut->Fall = -ABC_INFINITY;
pNode->pWorstFaninRise=pFanin;
}
else if(pFanin->fMarkB==1)
{
pTimeOut->Fall = pTimeIn->Rise + tDelayBlockFall;
pTimeOut->Rise = -ABC_INFINITY;
pNode->pWorstFaninFall=pFanin;
}
else{}
}
}
else
{
// compute the arrival times of the positive phase
if ( PinPhase != MIO_PHASE_INV )  // NONINV phase is present
{
pNode->fMarkC = 0;
if ( pTimeOut->Rise < pTimeIn->Rise + tDelayBlockRise )
{
pTimeOut->Rise = pTimeIn->Rise + tDelayBlockRise;
pNode->pWorstFaninRise=pFanin;
}
if ( pTimeOut->Fall < pTimeIn->Fall + tDelayBlockFall )
{
pTimeOut->Fall = pTimeIn->Fall + tDelayBlockFall;
pNode->pWorstFaninFall=pFanin;
}
}
if ( PinPhase != MIO_PHASE_NONINV )  // INV phase is present
{
pNode->fMarkC = 1;
if ( pTimeOut->Rise < pTimeIn->Fall + tDelayBlockRise )
{
pTimeOut->Rise = pTimeIn->Fall + tDelayBlockRise;
pNode->pWorstFaninRise=pFanin;
}
if ( pTimeOut->Fall < pTimeIn->Rise + tDelayBlockFall )
{
pTimeOut->Fall = pTimeIn->Rise + tDelayBlockFall;
pNode->pWorstFaninFall=pFanin;
}
}
}
```

```
}

//////////////Recreating the worst situation/////////////////
int Abc_WorstCaseRecreate(Abc_Frame_t * pAbc)
{
Abc_Obj_t * pFanout, * pFanin, * pNode, *pPi1,*pPi2;
Abc_Time_t * pTimeIn, * pTimeOut, * pTime;
Vec_Ptr_t * vNodes, * vCritical;
float tDelayBlockRise, tDelayBlockFall,tArrivalMax = -ABC_INFINITY;
float Min_tArrivalMax = ABC_INFINITY ;
Mio_PinPhase_t PinPhase;
//Mio_Gate_t * pRoot;
Mio_Pin_t * pPin;
Abc_ManTime_t *   pManTime;
int i,PI_iter1,PI_iter2,Node_iter, Logic1, Logic2;
//char ** Min_PI1,** Min_PI2,** Min_PO,** Min_PO_LOCAL;

Abc_NtkTimePrepare( pAbc->pNtkCur );
vNodes = Abc_NtkDfs( pAbc->pNtkCur, 1 );
Abc_NtkForEachPi( pAbc->pNtkCur, pPi1, PI_iter1)
%//iterating over each primary input of the circuit
{
if(Abc_ObjName(pPi1)!=Min_PI1)
continue;
Logic1=Min_Log1;
pPi1->fMarkB = Logic1;
Abc_NtkForEachPi( pAbc->pNtkCur, pPi2, PI_iter2)
%//iterating over each primary input of the circuit
{
if(Abc_ObjName(pPi2)!=Min_PI2)
continue;
Logic2=Min_Log2;
pPi2->fMarkB = Logic2;
for ( Node_iter = 0; Node_iter < vNodes->nSize; Node_iter++ )
%//Node iteration for logic propagation
{
pNode=vNodes->pArray[Node_iter];
pNode->fMarkB=2;
pTimeOut= Abc_NodeArrival(pNode);
pPin = Mio_GateReadPins(pNode->pData);

Abc_ObjForEachFanin( pNode, pFanin, i )
{
Set_logic_based_delay(pNode,pPin,pFanin);
pTimeOut = Abc_NodeArrival(pNode);
pTimeOut->Worst = ABC_MAX( pTimeOut->Rise, pTimeOut->Fall );
pPin = Mio_PinReadNext(pPin);
```

```
}//end of fanin iteration

if(pTimeOut->Worst>tArrivalMax)
{
tArrivalMax = pTimeOut->Worst;
Min_PO_LOCAL=Abc_ObjName(pNode);
}
}//end of node iteration

}//end of PI2 iteration


}//end of PI1 iteration
}
//int ObjPropagateLogic( Hop_Obj_t * pObj, Vec_Vec_t * vLevels, int Level )
//{
//    Vec_Ptr_t * vSuper;
//    Hop_Obj_t * pFanin;
//    int fCompl, i;
//    // store the complemented attribute
//    fCompl = Hop_IsComplement(pObj);
//    pObj = Hop_Regular(pObj);
//    // constant case
// if ( pObj->fMarkB<3 )
//    {
// return pObj->fMarkB;
//    }
//    // PI case
//    if ( Hop_ObjIsPi(pObj) )
//    {
//        fprintf( pFile, "%s%s", fCompl? "!" : "", pObj->pData );
// return pObj->fMarkB;
//    }
//    // AND case
//    Vec_VecExpand( vLevels, Level );
//    vSuper = Vec_VecEntry(vLevels, Level);
//    Hop_ObjCollectMulti( pObj, vSuper );
//    fprintf( pFile, "%s", (Level==0? "" : "(") );
//    Vec_PtrForEachEntry( vSuper, pFanin, i )
//    {
//      Hop_ObjPrintEqn( pFile,Hop_NotCond(pFanin, fCompl),vLevels,Level+1);
//      if ( i < Vec_PtrSize(vSuper) - 1 )
//      fprintf( pFile, " %s ", fCompl? "+" : "*" );
//    }
//    fprintf( pFile, "%s", (Level==0? "" : ")") );
//    return;
//}
//
////////////////////////////////[JK] End ////////////////////////
```