Energy-Efficient Digital Circuit Design using Threshold Logic Gates

by

Niranjan Kulkarni

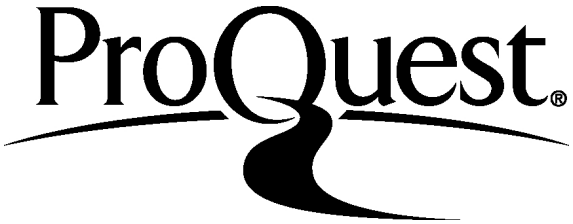
A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved November 2015 by the
Graduate Supervisory Committee:

Sarma Vrudhula, Chair
Charles Colbourn
Jae-Sun Seo
Shimeng Yu

ARIZONA STATE UNIVERSITY

December 2015



ABSTRACT

Improving energy efficiency has always been the prime objective of the custom and automated digital circuit design techniques. As a result, a multitude of methods to reduce power without sacrificing performance have been proposed. However, as the field of design automation has matured over the last few decades, there have been no new automated design techniques, that can provide considerable improvements in circuit power, leakage and area. Although emerging nano-devices are expected to replace the existing MOSFET devices, they are far from being as mature as semiconductor devices and their full potential and promises are many years away from being practical.

The research described in this dissertation consists of four main parts. First is a new circuit architecture of a differential threshold logic flipflop called PNAND. The PNAND gate is an edge-triggered multi-input sequential cell whose next state function is a threshold function of its inputs. Second a new approach, called hybridization, that replaces flipflops and parts of their logic cones with PNAND cells is described. The resulting *hybrid* circuit, which consists of conventional logic cells and PNANDs, is shown to have significantly less power consumption, smaller area, less standby power and less power variation.

Third, a new architecture of a field programmable array, called field programmable threshold logic array (FPTLA), in which the standard lookup table (LUT) is replaced by a PNAND is described. The FPTLA is shown to have as much as 50% lower energy-delay product compared to conventional FPGA using well known FPGA modeling tool called VPR.

Fourth, a novel clock skewing technique that makes use of the *completion detection* feature of the differential mode flipflops is described. This clock skewing method improves the area and power of the ASIC circuits by increasing slack on timing paths.

An additional advantage of this method is the *elimination* of hold time violation on given short paths.

Several circuit design methodologies such as retiming and asynchronous circuit design can use the proposed threshold logic gate effectively. Therefore, the use of threshold logic flipflops in conventional design methodologies opens new avenues of research towards more energy-efficient circuits.

DEDICATION

To my parents

who always encouraged and supported me throughout this endeavor

To my wife, Neha

without whose continued support this dissertation would not have been possible

ACKNOWLEDGMENTS

I would like to thank my adviser, Prof. Sarma Vrudhula, for having faith in my abilities, and for all his guidance through the years. Despite several frustrating times, he helped me keep moving with moral and intellectual support and for that I am deeply indebted to him. My sincere thanks to my committee members as well, for taking the time to review my work, attending my presentations, and offering many helpful suggestions.

I am fortunate to have collaborated with Tejaswi Gowda, who first got me interested in the field of threshold logic. I am thankful to many other colleagues who became my friends, who were there for me when I needed them and for making the otherwise grim workplace a whole lot more fun and cheerful. My special thanks to Jinghua Yang and Joseph Davis, without their collaboration, this work would not have been complete.

My family has always been a constant source of encouragement and support through these long years. I am grateful for their trust in me, and for everything they have done for me.

I gratefully acknowledge the support I received from the following agencies: The Stardust Foundation through a Science Foundation Arizona grant SRG 0211-01; The National Science Foundation for grants 1230401, 0702831 and PFI-BIC 1237856. I would also like to thank the School of Computing, Informatics and Decision Systems Engineering for granting me with necessary resources in a timely manner.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Minimizing the area and dynamic power of digital CMOS circuits have always been two central objectives of the automated circuit design. This focus on dynamic power stems from several problems arising from an increased power consumption. First and foremost are thermal constraints. The increased temperature of circuits has several debilitating effects such as reduced speed, increased leakage (wasted power), accelerated aging etc. Reducing power consumption not only alleviates these problems but also reduces packaging and cooling costs (Lin and Banerjee, 2008).

The second requirement for low power comes from limited energy capacity of batteries used to power mobile devices, (Chalmers and Sloman, 1999). As more and more transistors are packed on today's chips, the power requirement starts growing beyond the capabilities of the batteries necessitating energy-efficient designs. Improvements in battery technology to increase their capacity have not kept pace with the increase in transistor count and transistor density made possible by technology scaling. Energy efficient design is therefore critical for mobile platforms.

Efforts to reduce power consumption of digital CMOS circuits have been in progress for nearly three decades (Chandrakasan and Brodersen, 1998; Panda *et al.*, 2010). As a result, a number of well understood and proven techniques for reducing dynamic and leakage power have been incorporated into modern design practices and tools. Fig. 1.1 gives an idea about all the levels of hierarchy at which power reduction techniques have been explored.

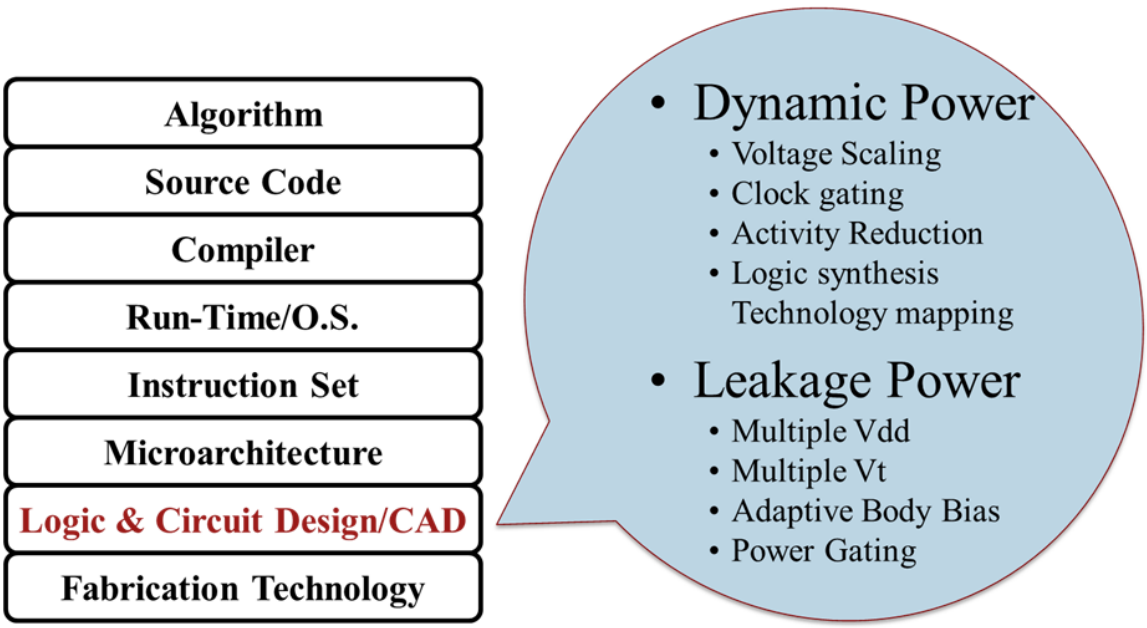


Figure 1.1: Power Reduction Techniques

Power reduction techniques

A number of power reduction techniques have been applied at different levels of the circuit design hierarchy namely at the system level (algorithm, source code and OS compiler etc) (Piguet, 2005) (Lorch, 2001), at the architecture level (instruction set, micro-architecture) (Qadri *et al.*, 2009) (Zandrahimi and Al-Ars, 2014) (Moyer, 2001) and at the circuit level (circuit design, gate design and transistor design) (Devadas and Malik, 1995) (Chandrakasan and Brodersen, 2012). At each level, the techniques range from being completely general (applicable to all designs) to being applicable to specific types of circuits(such as only controllers or arithmetic designs). The dynamic power of the static (charge driven) digital switching circuit is given by

$$P_{dyn} = C.V^2.\alpha F \tag{1.1}$$

where C denotes the capacitance that is being charged or discharged during switching, V is the supply voltage, α denotes the switching activity, and F denotes the frequency

at which the circuit is operated. Each power reduction technique at any level targets a combination of these parameters to reduce the power. At the circuit and CAD level, efficient synthesis and technology mapping methods have been shown to reduce circuit area and switching activity (Brzozowski and Kos, 1999) (Keutzer, 1988) (Tiwari *et al.*, 1993). Additional methods such as logic restructuring (Brzozowski and Kos, 1999), gate sizing under timing slacks (Coudert, 1997), technology mapping (Tiwari *et al.*, 1993), re-timing (Monteiro *et al.*, 1996), clock gating (Chandrakasan and Brodersen, 2012) etc. have been applied and a large body of work have been published on these methods. We briefly summarize the techniques that are most relevant to this work. The techniques explored in this dissertation for improving digital circuits fall in to two categories.

1. Power and area reduction through gate sizing, and

2. Power reduction through the use of alternate types of timing elements.

Power and area reduction through gate sizing under timing slack

The use of smaller and slower cells can achieve reduction in area as well as dynamic power. For example, a 4-input AND gate is smaller and slower than a tree of 2-input AND gates. In the presence of positive timing slacks, it is possible to employ slower cells. A well-known method to reduce total power without sacrificing performance is to exploit slacks available on non-critical paths. The problem of choosing a particular gate size, formally known as the "gate sizing problem" (Coudert, 1997) consists of choosing, for each node of a mapped network, a gate implementation in the library, such that some cost function (e.g. dynamic power, leakage power or area or some combination of these) is optimized under given timing constraints. Techniques described in (Chen and Sarrafzadeh, 1996) (Coudert, 1997) reduce the available timing

slack from a positive value to zero by using smaller and slower cells. The smaller cells typically require less area and have lower dynamic and leakage power. The average power improvement due to gate resizing in presence of timing slacks has been shown to be 38% (Chen and Sarrafzadeh, 1996). Similar techniques to reduce gate sizes are routinely employed in most commercial synthesis and mapping tools. Therefore, any additional timing slack introduced on the critical path without any change in clock period can reduce the total power considerably. This idea forms the basis of the power reduction ideas presented in this work. The techniques presented here maximize the timing slack on critical paths (for ASIC circuits) in order to reduce area and power.

Role of single and multi-input flip-flops in providing timing slack

In any timing path within a sequential circuit, the clock-to-Q delay and setup time of flipflops play an important role in determining the clock period. The is especially true in high speed pipelined datapath circuits. The clock period is determined by the total delay of the flipflop which is the sum of setup time and the clock-to-Q delay. Reducing the total delay results in extra slack, which can be used to reduce the size of the logic gates feeding the flipflop.

In order to reduce effective delay (also called pipeline overhead) of the flip-flops, attempts have been made to incorporate logic inside the flipflop. Dynamic and semi-dynamic flip-flops described in (Klass, 1998) (Partovi *et al.*, 1996) embed an NMOS logic block in the master latch. These flipflops have a dynamic master latch that presets and evaluates a logic function and a static back end that simply stores the value computed by the master latch for the remainder of the clock cycle. However these flip-flops pose few problems. First, their setup time grows substantially with the increased complexity of the embedded logic. Ideally the complexity of functions should not substantially affect the setup time. Second, for each logic function, a

new flipflop has to be designed, optimized and characterized from scratch which can quickly become prohibitively expensive.

Threshold Logic Flipflops

Advantages of differential mode threshold gates

The static differential threshold gates (DTGs) overcome the obstacles posed by dynamic flip-flops that embed NMOS logic. The main feature of a DTG flip-flop is computing and amplifying the conductivity difference between two networks of parallel transistors referred to as the left input network (LIN) and the right input network (RIN). If the LIN has lower impedance then the output is logic 1, otherwise it is 0. Since the impedance of either network is an integral multiple of the number of ON transistors, the Boolean function of such DTG can be described by the predicate $\sum x_i > \sum y_i$ where x_i are variables controlling the number of ON transistors in the LIN and y_i are Boolean variables controlling the number of ON transistors in the RIN. This predicate can be algebraically rearranged to yield the predicate $\sum(x_i - y_i) > 0$. This type of pseudo-boolean predicate corresponds to the class of linear threshold functions. Given a linear threshold function (which is a Boolean function), it is possible to connect the variables (and constants) to the input networks such that predicate of the DTG reduces to that of the given threshold function.

The DTGs (Strandberg and Yuan, 2000a)(Padure *et al.*, 2001b)(Leshner *et al.*, 2010) have been shown to be superior to the conventional multi-input and single input flip-flops because of following advantages: (1) They have a much lower and constant total delay whose value is almost independent of the complexity of the function being implemented (2) They impose no additional requirements from clock circuitry while exhibiting the same or lower clock load, (3) an they permit synthesis

time programmability i.e. the DTGs can be wired to implement a range of threshold functions which effectively reduces the number of cells in the library. In essence, the DTGs overcome most of the problems of flipflops embedding logic, however the type of logic they can implement is restricted to threshold logic functions as opposed to general Boolean functions.

One aspect of digital CMOS circuits that hasn't changed is how logic functions are computed. A CMOS ASIC circuit using static logic is a multi-level network of AND/OR logic gates or more complex cells such as AOI/OAI cells, in which each node computes a Boolean function of its inputs by establishing a conducting path from the supply rails to its output. However, *threshold* functions, which are a proper subset of unate Boolean functions, can be computed by fundamentally different mechanisms, and this presents the possibility of further improvements in power consumption, performance and area, which have not been thoroughly explored.

Let $X = (x_1, x_2, \cdots, x_n)$, $x_i \in \{0, 1\}$, $\boldsymbol{w} = (w_1, w_2, \cdots w_n)$, $w_i \in R$, and $T \in R$. A unate Boolean function $f(X)$ is called a threshold function if there exist weights $\boldsymbol{w}$ and a fixed threshold T such that

$$f(X) = \begin{cases} 1 & \text{if } \boldsymbol{w}'X \geq T \\ 0 & \text{otherwise.} \end{cases} \tag{1.2}$$

$$= sgn^{+}(\boldsymbol{w}'X - T)^{1} \tag{1.3}$$

Without loss of generality, the weights $\boldsymbol{w}$ and threshold T can be assumed to be positive integers. The following are two examples of threshold functions.

$$\begin{aligned} f(a, b, c) &= ab \vee bc \vee ac \\ &= [w_a = 1, w_b = 1, w_c = 1; T = 2] && (1.4) \\ &= [1, 1, 1; 2] && (1.5) \end{aligned}$$

[1] $\text{sgn}^{+}(x) = 1$ if $x \geq 0$, and $sgn^{+}(x) = 0$ otherwise (see (Siu *et al.*, 1995)).

$$g(a,b,c,d,e) = a(b \vee c \vee d \vee e) \vee b(cd \vee de \vee ce) \vee cde$$

$$= [w_a = 2, w_b = 1, w_c = 1, w_d = 1, w_e = 1; T = 3] \tag{1.6}$$

$$= [2, 1, 1, 1, 1; 3]. \tag{1.7}$$

A threshold function can be implemented in the same way as any logic function, e.g. network of logic primitives or a pull-up and pull-down network of pFETs and nFETs, etc. Such implementations are not considered here, as they offer nothing new, and in fact, can be quite inefficient for implementing threshold functions in terms of speed, power and area. The implementations of a threshold logic gate considered in this work compute the predicate in (1.2) by performing a comparison of some electrical quantity such as charge, voltage or current. This is what distinguishes such implementations of a threshold gate with any of the conventional implementations of CMOS logic functions.

The reason for examining threshold gates as logic primitives stems from the fact that they are computationally more *powerful* than the standard AND/OR logic primitives. Many common logic functions such as the n-bit parity, n-bit multiplication, division, powering, sorting, etc,. can be computed by polynomial size threshold networks of a fixed number of levels, while the same would require exponential size AND/OR networks (Siu *et al.*, 1995). A detailed treatment of the complexity of threshold networks and constructive methods for various types of arithmetic functions, including size-depth and weight-depth trade-offs is presented in (Siu *et al.*, 1995). An updated survey of the same appears in (Beiu, 2003), and an extensive survey of circuit architectures of threshold gates is given in (Beiu *et al.*, 2003). These surveys point to a large body of results which suggest that threshold gates and networks can potentially lead to significant reductions in circuit size and delay. However, the use of threshold logic gates in conventional ASIC design has not been thoroughly

explored due to the lack of efficient and reliable gate implementations and the necessary infrastructure required for automated synthesis and physical design.

In this dissertation, new methodologies for digital circuit design with threshold logic gates are described. The applications of threshold logic gates to improve energy-efficiency of two well known digital circuit design methodologies viz. - Application Specific Integrated Circuits (ASIC), Field Programmable Gate Arrays (FPGA) is described. We present the design of new circuit primitives and new logic synthesis algorithms.

The outline of the dissertation is as follows.

1. Chapter 2 describes the earlier work on the design and usage of differential mode threshold logic gates in circuit design.

2. Chapter 3 describes a new, robust architecture for a threshold gate that employs differential logic, referred to as a PNAND cell. As with other implementations in the same category, a PNAND cell is clocked, and its behavior can be abstracted to be that of a multi-input edge-triggered flip-flop (ETFF). Whereas a conventional D-type ETFF (D-FF) computes the identify function $f(x) = x$ on a clock edge, a PNAND cell computes a threshold function $f(x_1, x_2, \cdots, x_n)$, also on a clock edge. Furthermore, like the D-FF, a PNAND cell can be made scannable and have other features like asynchronous preset and clear.

   Unlike conventional logic gates or cells, the function of a PNAND cell is not determined by the cell itself. Instead, its function is determined by the signals that drive its inputs. The signal assignment affects the delay, power and *robustness* of the cell to process variations. Therefore, a specific signal assignment that results in a least fanin PNAND cell to implement a given threshold function is defined. Conversely, for a PNAND cell with k inputs (PNAND-k), and with the

specific signal assignment, the set of threshold functions that can be realized only by such a cell, and no other cell with a smaller k is determined. This leads to a small number of PNAND cells which collectively realize a substantial number of complex threshold functions, many of which would otherwise require a multi-level AND/OR network if implemented in a conventional design.

The robustness of the PNAND cell against parametric variations in 65nm technology is demonstrated (Section 3.5) through Monte Carlo simulations performed with vendor supplied variation data obtained from fabrication, and which include both global variations and local mismatch. The cells have been designed to achieve zero out 100,000 Monte-Carlo simulation trials.

3. Chapter 4 describes two novel methods for technology mapping with PNAND cells starting from a generic AND/OR netlists. Since a PNAND cell may be viewed as a complex function embedded in a flip-flop, the technology mapping algorithm explores cones of logic that end at D-FFs for threshold functions, and replaces such subcircuits with the appropriate PNAND cell. The resulting circuit will have a mixture of conventional logic gates and threshold logic cells, and will be referred to as a *hybrid* design. A useful characteristic of threshold functions that is exploited during technology mapping is the ease of determining the NPN (input Negation, input Permutation, output Negation) equivalence of two threshold functions. This is not the case with AND/OR logic. This characteristic makes it simple to check if a a threshold function found in a logic cone is NPN equivalent to any of the functions implemented by the small set of PNAND cells in the library, thereby significantly increasing the set of possible matches during technology mapping.

   The results of technology mapping with threshold gates for a number of complex

designs are presented and compared with conventional logic implementations. The PNAND cells were implemented as standard cells using a commercial 65nm technology. The netlists resulting from technology mapping with threshold cells were synthesized, and placed and routed using commercial tools. The same was done for the conventional design. The resulting final layouts were extracted and simulated for comparison with the conventional design.

4. Using the proposed PNAND as a basic cell, chapter 5 describes a novel design of a field programmable threshold logic array (FPTLA), and a method to synthesize a network of TLGs that can be mapped on to the array.

   The FPTLA can be programmed to realize any one of a number of threshold functions by assigning the appropriate signals to the inputs. This reduces the number of SRAMs required by the logic blocks by about 60%.

   It is well known that differential mode threshold logic gates can be more sensitive to mismatch due to process variations. However, mismatch in a PNAND need not make the logic block unusable. Rather, mismatch causes the *set of* threshold functions realizable by the PNAND to be altered. This leads to two different scenarios in how the array can be utilized.

   (a) A conservative approach is to simply ignore the cells whose functionality is altered by process variations. To ensure feasibility of the mapping, if a TLG network to be mapped has N TLGs, then an array of PNANDs larger than N would be required. We show that for a given probability of a cell's functionality being altered, the number of extra cells required to achieve a very high circuit yield is small.

   (b) A key characteristic of PNAND cells is that mismatch can only result in another set of *threshold* functions. That is, mismatch will not change a threshold

function into a non-threshold function – a characteristic that is demonstrated here. Hence, instead of simply ignoring the altered cells, the synthesis procedure can take in account the new set of all functions available on the array, and generate a TLG network accordingly. Note that the synthesis procedure that accounts for this additional flexibility is beyond the scope of this work.

We present experimental results by comparing FPTLA and conventional FPGA implementations of several nanopipelined combinational benchmarks. Nanopipelined circuits are path balanced in which each gate computes a function on a clock edge. The throughput and speed of such circuits is very high due to such deep level of pipelining. The power and delays of these circuits are obtained using the well established VPR tool that accurately models the *routing resources and logic blocks*.

5. Chapter 6 proposes a novel clocking method called as *local clocking* to further improve power reduction of ASIC circuits. The most important thing about local clocking is that it provides *incremental* improvements to existing circuits. Therefore whether a circuit is already hybrid or not, local clocking can still be applied. Local clocking works by deriving local clock signals from differential mode threshold gates. Most differential mode gate circuits operate using a sense-amplifier which has a differential amplified outputs (at logic 1 and logic 0 or vice versa). This state indicates the completion of computation which can be used to generate another output signal which has same pattern (albeit delay shifted) as the input clock. This skewed (delayed) output clock can be used to locally trigger other flipflops to realize additional advantages such as reduction in area and power under timing slack and elimination of hold time violations on certain short paths. A 0-1 integer programming formulation is provided

for local clocking problem that decides which flipflops provide trigger to which other flipflops while satisfying timing constraints. Finally experimental results are provided on 64-bit multipliers with K pipeline stages (K=1 to 6) with locally clocked CMOS only and hybrid circuits. The local clocking provides additional 3% to 8% improvement in post-layout dynamic power.

Chapter 2

BACKGROUND

2.1 Necessity for Low Power

Traditional ASIC (Application Specific Integrated Circuit) design involves constructing networks of CMOS (Complimentary Metal Oxide Semiconductor) logic gates to implement a given functionality. CMOS logic gates such as NAND, NOR, OR etc. consist of pullup and pull down networks and they function by establishing a conducting path between one of the supply rails and the output depending on the input combination. This paradigm has worked exceedingly well, in terms of dynamic power, size and robustness of the circuit. However there is even greater need for low power circuits due problems such as overheating and limited battery sizes of portable devices.

One of the most common way to address power problem is to scale the process geometry. As the size of transistors shrink, so does the area and power of circuits. However it's been observed that as process geometries shrink, the power density (power consumption per unit area) grows. The absolute power may drop but the power consumption per unit area (due to shrinkage in area) increases. The cooling solutions can withdraw a fixed amount of heat per unit area. Therefore shrinking process geometry does not diminish the need for low power circuits but rather makes it more severe.

The total power of CMOS circuit consists of three components.

$$P_{total} = P_{dyn} + P_{static} + P_{ss}. \tag{2.1}$$

The total power is sum of the dynamic power, the static power and the short circuit

power (when both pullup and pull-down networks are ON). Dynamic power which is the largest contributor to the total power is computed as

$$P_{dyn} = C.V^2.\alpha.F. \tag{2.2}$$

Equation 2.2 shows that a quick way to reduce power is to scale down the supply voltage (V), or the frequency (F) of operation. Indeed, techniques such as DVS (Dynamic Voltage Scaling) and DVFS (Dynamic Voltage-Frequency Scaling) are utilized routinely to minimize the power of circuits depending on the workload and task deadline requirements. However reducing voltage or frequency also reduces the speed of the circuit. Most ASIC chips must operate at certain minimum speed while some have more stringent high speed requirements. Therefore reducing voltage or frequency is beneficial only when speed is not a concern. For a given voltage and frequency, the power consumed by the circuit is a characteristic property of the circuit. Therefore the main concern of circuit designers is to improve *energy delay product* rather than power. Energy delay product (EDP) is computed as

$$EDP = E_{tot}/F, \tag{2.3}$$

where E_{tot} is total energy expenditure and F is the frequency of operation of the circuit. The unit of this metric is Joules $\times$ seconds per cycle which is equivalent to the amount of energy spent multiplied by the amount of time taken for one operation. A circuit with the lowest expenditure of energy and time combined for one unit of operation is the most desired. EDP is directly related to the *energy efficiency*. A circuit with lower EDP is said to be more energy-efficient (Gonzalez and Horowitz, 1996). We can improve the EDP by speeding up the circuit without increasing the power, or lowering the power without decreasing the speed. The ASIC and FPGA circuit designs proposed in this thesis explore both of these ideas.

2.2 Dynamic Power Reduction Methods

The power reduction methods tried so far span several decades of research. The survey in (Devadas and Malik, 1995) gives an overview of most of the power reduction techniques. A brief summary of these ideas is presented here to give the reader an idea of how various approaches achieve power reduction.

Reducing Switching Activity

The *clock gating* technique enables the clock only when the circuit has a data input to process. The enable circuit typically produces a signal that is used in clock distribution networks to enable/disable propagation of the clock. The clock gating technique reduces activity factor (α) in the Equation 2.2. Another way to reduce the switching activity is restructure the logic. (Brzozowski and Kos, 1999) shows that a different synthesis of the same function can lead to lower switching activity.

Another important contributor to the dynamic power is glitching. It can contribute between 10% - 40% to the dynamic power. A glitch is a spurious transition of a signal caused by unbalanced path delays in combinational circuits. Techniques to minimize glitching by path balancing are presented in (Vijayakumar and Kundu, 2014) and (Lemonds and Shetti, 1994). Glitches are propagated by combinational gates from inputs to their outputs. However sequential elements such as flipflops do not propagate glitches. A retiming solution that introduces flipflops on most active wires to reduce power is described in (Monteiro *et al.*, 1996).

Reducing Capacitance

Almost all the remaining techniques focus on reducing the capacitance of the circuit. The capacitance is directly proportional to the transistor (or logic gate) sizes,

as well as their number. At the transistor level, transistor resizing technique in (Tan and Allen, 1994) minimizes the transistor sizes in a logic gate while meeting the delay constraints. The combinational networks can be reduced in size by techniques such as don't care based optimization (Shen *et al.*, 1992). Another approach for technology independent optimization is by improved factorization of logical expressions based on modified kernel extraction methods (Roy and Prasad, 1992).

Technology mapping step involves generating technology dependent realization of logic networks using a given set of library primitives. Techniques to minimize power by technology mapping are presented in (Keutzer, 1988) (Tiwari *et al.*, 1993).

The power reduction methods in sequential circuits involve modifying state encoding, retiming etc. The state encoding optimization technique which tries to reduce the state transition graph and the number of states is presented in (Tsui *et al.*, 1994).

2.3 Static Power Reduction Methods

The static or leakage power of the circuit P_{static} is a function of supply voltage, temperature and the static currents through all the transistors in the circuit. The contribution of static power compared to the dynamic power increases with miniaturization. Therefore a number of techniques have been tried to reduce the static power as well.

Most of the fabrication processes allow for multiple threshold voltage (V_T) transistors , typically at three levels: low, standard and high. A transistor with high V_T has much less static current. However at the same time, high V_T transistors are also slower. Therefore a common technique is to use standard cells with high V_T on non-critical paths. This method is presented in (Roy *et al.*, 2003).

Connecting transistors in series results in a substantial reduction in leakage. Therefore placing a sleep transistor between the supply rail and one of the pullup/pulldown

networks reduces the leakage substantially(Kumar *et al.*, 2013). The same stacking technique is extended to *power gating* wherein an entire block of logic is supplied power through a large PMOS transistor which is turned off when the circuit is not operated.

Controlling the supply voltage is another common technique to reduce static current. The supply voltage of memory modules is scaled down to a minimum possible value that ensures data retention. If a logic block is not required to operate faster a given certain speed its voltage can be reduced to a value that meets the speed requirements.

2.4 Threshold Logic: A New Approach

Static CMOS logic has been the dominant design methodology for digital circuits for the past several decades. Circuits consists of networks of AND, OR and NOT gates. As described above, techniques for reducing power consumption of CMOS circuits have been thoroughly investigated, and there hasn't been, nor likely to be, any new developments in reducing power consumption of static CMOS logic.

AND, OR and NOT functions are special cases of threshold functions, a natural question to investigate is whether networks of more general threshold functions can lead to faster, more compact, and lower power circuits. This is the main direction of this research.

A large of body theoretical work on threshold networks has demonstrated the numerous advantages in terms of size (i.e. number of gates), and speed (i.e. number of levels). We now briefly describe some of the many advantages of threshold networks over the conventional logic networks.

The symmetric functions are a subclass of Boolean functions whose output is only a function of number of 1's in the input vector. For example, parity or majority

functions can simply count the number of 1's in the inputs to be able to produce the output value. Symmetric functions of n inputs can be realized using $O(\sqrt{n})$ threshold gates with depth of 3 (constant depth). Whereas symmetric functions such as parity would require $O(n)$ AND-OR-NOT gates with depth of $O(\log n)$.

Arithmetic circuits are the most commonly used blocks in today's ASIC circuits. Many n-bit functions such as multiplication and exponentiation require exponentially many AND-OR-NOT gates. In contrast, a threshold networks of these functions require at most a polynomial in n number of threshold gates and very small depth (logic levels). For example, Addition and Comparison can be computed with depth-2 and polynomial number of threshold gates. Multiplication can be computed with polynomial sized circuit with depth of 4. Along with multiplication, powering, division and sorting can also be computed with depth of 4 or less. It should be noted that the integer weights of the threshold gates used in above networks are also polynomially bounded.

The above results and others suggest that, at least in theory, threshold networks can potentially be far more compact and faster than conventional logic networks. Whether these advantages can be realized in practice depends on the physical characteristics of threshold gates. In this dissertation, we demonstrate that a combination of threshold gates and conventional logic can indeed result in smaller and lower power circuits, than conventional logic networks, without sacrificing performance.

2.5 Threshold Logic Gates

If the predicate (1.2) in the definition of a threshold function is to be realized by a single circuit, it must be able to selectively add some physical quantity that represents the weights and compare that with a fixed quantity (threshold) having the same type as the weights. Typically the physical quantity that is added is one

of the charge, current, magnetic or electric fields. Depending on which physical quantity is utilized, there exists a range of threshold circuits that have been designed till date. The comprehensive survey in (Beiu *et al.*, 2003) shows nearly 50 different implementations based on different principles. Here we provide a short synopsis of some those implementations.

Threshold logic implementations can be broadly classified into two types: (1) static or combinational or (2) sequential (clocked). The static implementations essentially treat a threshold function as any other logic function, and therefore, a single pair of complementary pull-up and pull-down networks can be used to implement the given threshold function. However such implementations are prohibitively large and slow for all but small functions. Examples of such implementation are (J., 1964; Z. and W., 1967; Hidalgo-Lopez *et al.*, 1995). Another early design (J, 1973) uses multiple inverters driving a common node. The widths of these inverters are adjusted based on weights of the threshold functions which forms a non-linear voltage divider at the output node.

Among the sequential implementations, one of the earliest implementations is based on switched capacitors (CTL) (Özdemir *et al.*, 1996). Their main drawback was large delay, area, DC power and the need for precise capacitance values. Although the variant balanced CTL (B-CTL) (López-García *et al.*, 2004) eliminated the need for precise capacitance values by comparing relative voltages, it still retained the other drawbacks. Neuron MOS TLG (Shibata and Ohmi, 1991), which performs similar summation of voltage using capacitors eliminates the usage of a reference threshold voltage. Several modified versions of Neuron MOS TLGs (Kotani *et al.*, 1995, 1998) were proposed as improvements of the basic version. Later versions of the same used a sense-amplifier to sense the differential voltage after summation (Huang and Wang, 2000; Celinski *et al.*, 2001), which improved the sensitivity and power dissipation.

The self-timed threshold logic (STTL) uses capacitor sharing to minimize capacitor areas and can include negative weights without inverting inputs. However all these implementations still utilized precise capacitance devices which are hard to fabricate and also require large area.

Threshold logic gate architectures based on comparing conductances include (1) Single input Current Sensing Differential Logic (SCSDL) (Strandberg and Yuan, 2000b), (2) Differential Current Switch Threshold Logic (DCSTL) (Padure *et al.*, 2001b), (3) Current Mode Threshold Logic (CMTL) (Bobba and Hajj, 2000) and some of its variants known as discharged CMTL (DCMTL) and equalized CMTL (ECMTL), and Threshold Logic Latch (TLL) (Leshner *et al.*, 2010; Leshner, 2010). A high fanin current mode threshold logic gate similar to CMTL is presented in (Dara *et al.*, 2012), along with a analysis of its delay based on device models without accounting for process variations. Technology independent analysis of the robustness of threshold gates is presented in (Dechu *et al.*, 2006), in which any type of variation is modeled as perturbations in the weights and threshold.

Researchers have recognized that several of the post-CMOS nano technologies [1] which make it possible to efficiently and naturally implement threshold functions. (Gupta and Jha, 2005) describes nano-pipelined implementations of threshold gates using RTDs and (Lageweg *et al.*, 2001) describes a novel design of an n-input linear threshold gate, requiring one tunnel junction and $n+2$ capacitors. Using an improved design of a SET based threshold logic gate, a full adder design is described in (Lageweg *et al.*, 2002). (Sulieman and Beiu, 2004) describes the optimal threshold gate based design of a 16-bit parallel-prefix adder using single electron transistors (SET). SETs

[1] Resonant tunneling diodes (RTD)(Prost *et al.*, 2000), carbon nanotube FETs (CNFET) (Appenzeller *et al.*, 2005) and carbon nanowires; single electron transistors (SET) (Lageweg *et al.*, 2001), quantum cellular automata (QCA) (Blair and Lent, 2003; Jian *et al.*, 1998), non-charge based devices such as spin transfer torque magnetic tunnel junctions (STT-MTJ), etc.

have been shown to realize latches, flip-flops and majority gates (Lageweg *et al.*, 2001). A threshold gate has been built with an RTD and an HFET (Chen *et al.*, 1995). A primitive gate in a QCA is a majority gate which is a threshold function (Zhang *et al.*, 2005).

2.6 Synthesis and Mapping using Threshold Logic

Most, if not all of the prior work on synthesis of threshold networks is technology independent, and the earliest work dates back to the 1960s. A comprehensive treatment of the subject through 1970s appears in (Muroga, 1971). Some of the methods summarized in (Muroga, 1971) are: (1) Minterm expansion, which uses exponentially many majority gates; (2) Akers' method based on self dualization and positivization; (3) Minnick's synthesis based on iterative solution of linear programming problems along with artificial variables; (4) Hughes method which is similar to Minnick's method but doesn't use artificial variables; (5) Winder's method based on geometric interpretation of monotonic properties of threshold gates and finally an optimum network synthesis for threshold networks based on integer linear programming. These techniques are generally not suitable for present day industrial designs.

Reference (Siu *et al.*, 1995) presents a thorough treatment of the fundamental complexity results of threshold networks up to 1995, and (Beiu, 2003) covers many additional results through 1999. Most of these results point to the fact that threshold logic networks require fewer number and levels of gates compared to conventional AND-OR-NOT networks. A more recent threshold synthesis procedure is due to Zhang et al. (Zhang *et al.*, 2005) which shows an average of 52% reduction in gate count compared to Boolean networks for MCNC benchmarks. An improved method based on iterative decomposition of BDD of a general Boolean function is presented in (Gowda *et al.*, 2011).

It should be noted that most of the prior work on threshold synthesis assumes either an abstract threshold gate or nano-device based gates which are hard to fabricate with required precision. But the MOSFET based differential mode threshold gates (DTGs) that can compete with static CMOS style gates are essentially sequential by nature. It is possible to develop a balanced or nano-pipelined network consisting only of DTGs. Such networks, although faster than any conventional static CMOS gate networks, pose several additional challenges such as increased complexity of clock distribution, increased power and area etc. in ASIC domain [2] . Inspired from the work in (Leshner *et al.*, 2010), the present work in ASIC domain (Chapters 3 and 4) further extend the idea of hybrid networks that consist of an optimal mix of PNANDs and conventional CMOS standard cells.

2.7 Threshold Logic in Field Programmable Environment

Compared to standard cell based circuits, the attempts to utilize threshold logic in a field programmable environment have been very scarce, and only a few implementations exist. (Rajendran *et al.*, 2010) describe a novel memristor based threshold logic gate array. The computing block in this architecture consists of a threshold gate where memristance of the device act as weights. The computing block consists of current mirrors which are used to add currents through the memristive devices which is then compared with a reference current. Although they show substantial improvements in power (about 75%) compared to MOSFET based 4-input LUT FPGA, the absolute delay of the threshold gates are quite high (12 times that of LUT). Another implementation that uses a nano-device is presented in (Nukala *et al.*, 2012). The computing block in this gate array is a threshold gate consisting of STT-MTJ (spin

[2]However in field programmable circuit domain, the clock is already distributed and therefore a nano-pipelined implementation can take advantage of it. In fact, chapter 5 explores this idea.

transfer torque magnetic tunnel junction). This device switches its resistive state if a current larger than certain magnitude (switching current) is passed through it. A parallel network of pFETs is employed to tune the current passing through STT-MTJ. If enough inputs are 1, then the current through the STT-MTJ is greater than switching magnitude and it switches to the state of low resistance. Therefore the STT-MTJ acts as a natural threshold logic gate.

Chapter 3

THRESHOLD LOGIC FLIPFLOP

This chapter explains the architecture, operation and cell level analysis of the proposed threshold gate PNAND.

3.1 Architecture

Figure 3.1 shows the schematic of the threshold gate with k inputs, henceforth referred to as PNAND-k. It consists of three main components: (1) two groups of parallel pFET transistors referred to as the left input network (LIN), and right input network (RIN), (2) a sense amplifier (SA), which consists of a pair of cross coupled NAND gates, and a (3) set-reset (SR) latch. The cell is operated in two phases: reset (CLK = 0) and evaluation (CLK $0 \rightarrow 1$). For the moment, ignore the transistors $M9$ and $M10$.

Reset phase: With CLK = 0, the two discharge devices $M18$ and $M19$ pull nodes $N5$ and $N6$ low, which turn off $M5$ and $M6$, disconnecting all paths from $N1$ and $N2$ to ground. In addition, $M7$ and $M8$ are active, which results in $N1$ and $N2$ being pulled high. The nFETs $M3$ and $M4$ are ON. With $N1$ and $N2$ being high, the state of the SR latch does not change.

Evaluation Phase: This corresponds to when CLK $0 \rightarrow 1$. A input that results in ℓ active devices in the LIN and r active devices in the RIN is denoted by ℓ/r. The *signal assignment* procedure (to be explained shortly) will ensure that $\ell \neq r$. Assume that $\ell > r$. As a result, the conductance of the LIN is higher than that of the RIN.

As the discharge devices $M18$ and $M19$ are turned off, both $N5$ and $N6$ will rise to 1. Due to the higher conductivity of the LIN, node $N5$ will start to rise first,

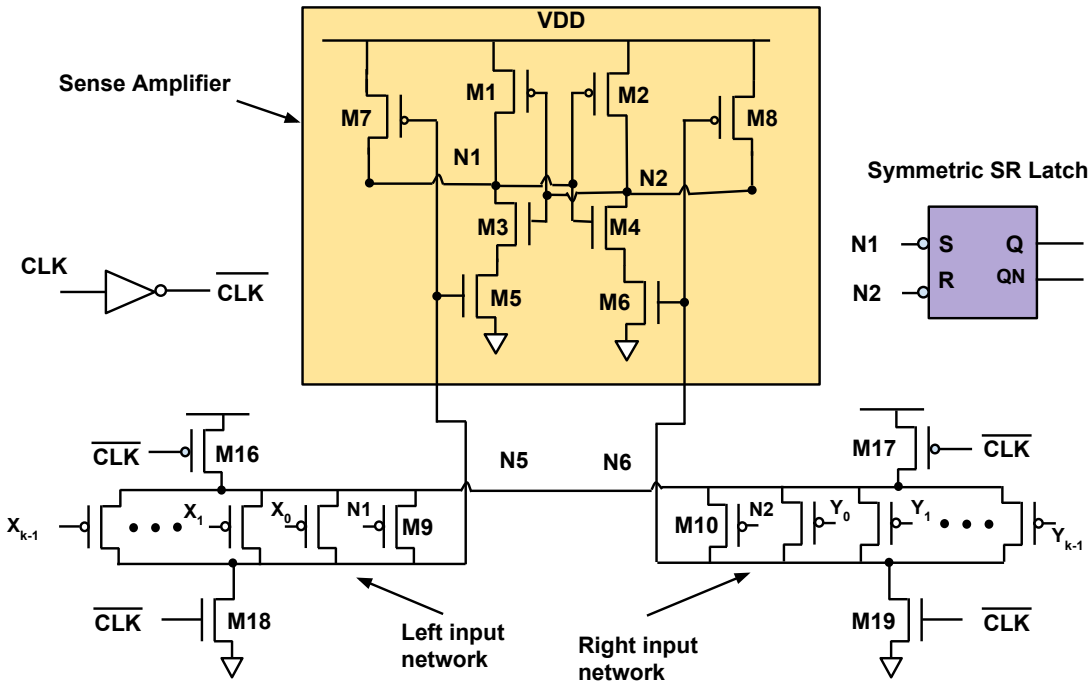


Figure 3.1: PNAND Cell Design

which turns on $M5$. With $M3 = 1$, $N1$ will start to discharge through $M3$ and $M5$. The delay in the start time for charging $N6$ due to the lower conductance of the RIN allows for $N1$ to turn on $M2$ and turn off $M4$. Thus, even if $N2$ starts to discharge, its further discharge is impeded as $M2$ turns on, resulting in $N2$ getting pulled back to 1. As a result, the output node $N1$ is 1 and $N2$ is 0. As the circuit and its operation are symmetric, if $\ell < r$, then the evaluation will result in $N1 = 0$ and $N2 = 1$.

The active low SR-latch stores the signals $N1$ and $N2$. During reset, when $(N1, N2) = 1$, the SR-latch retains its state. After evaluation, if $(N1, N2) = (0, 1)$, the output $Q = 0$, and if $(N1, N2) = (1, 0)$, $Q = 1$, providing a dual-rail output for the threshold function being computed. Therefore, once evaluated after rising edge of the CLK, the output Q of the cell is stable for the remaining duration of the clock cycle. Hence, it operates like an edge-triggered flip-flop, that computes a threshold

function.

Since it is the "difference in conductivity" between the LIN and RIN that is sensed and amplified, the greater the difference, the faster and more "reliably" the cell operates. In the layout of the PNAND cell, several steps were taken to ensure robustness to process variations and signal integrity. A symmetric SR-latch (Nikolic *et al.*, 2000) was used to ensure near identical load on node $N1$ and $N2$ and near equal rise and fall delays. The source nodes of $M16$ and $M17$ are shorted so that the transistors in the LIN and RIN have nearly identical V_D, V_G and V_S before clock rises. The sizes of the pull-down devices in the differential amplifier were optimized, as were the sizes of the input transistors in the LIN and RIN to maximize the conductivity difference for the input combination that results in the worst-case *contention* between the LIN and RIN, while keeping the RC delay of the input networks as low as possible. In addition, to further improve the robustness of the cell, an internal feedback is created with transistors $M9$ and $M10$ in the LIN and RIN, driven by $N1$ and $N2$, respectively.

These additional transistors $M9$ and $M10$ in the input networks serve as keepers to avoid the situation where $N5$ and $N6$ might be in a high impedance states (HiZ1, HiZ0). Assume for the moment that $M9$ and $M10$ are not present, and consider the situation in which there are k active devices in the LIN and none in the RIN. After reset, $N5$ and $N6$ are both 0. When the clock rises to 1, $N5$ will rise to 1, and $N6$ will be HiZ0, and the circuit will correctly evaluate with $N1 = 0, N2 = 1$. Note that $M4$ is inactive and $M3$ is active. Now suppose that while $CLK = 1$ the inputs change, and all transistors in the LIN become inactive and some k transistors in the RIN become active. $N5$ is now HiZ1, and $N1$ will remain at 0, keeping $M4$ inactive. However, $N6$ rises to 1, turning on $M6$, but as long as $M4$ remains inactive, and $M2$ active, no change will take place. $N5$, being HiZ1, is susceptible to being discharged.

If that happens, $N1$ rises, activating $M4$, and discharging $N2$, which results in the output being complemented.

Transistors $M9$ and $M10$ ensure that $N5$ and $N6$ do not become HiZ0 or HiZ1. In the above situation, once $N1 = 0$ during evaluation, the presence of $M9$ driven by $N1$ ensures that $N5 = 1$. Hence, after evaluation and while $CLK = 1$, any change in the input state will not affect the side that determined the output, i.e. was discharged first, and hence the output will not be disturbed.

Figure 3.2 shows waveforms from the SPICE simulation of a PNAND-9 netlist extracted from layout, with and without $M9$ and $M10$ transistors. For the specific signal assignment used in the technology mapping, the maximum number of active devices in the LIN or RIN among all the functions realized by a PNAND-9 is 5. Therefore the simulation shown in Figure 3.2 starts with applying a 5/4 input, which results in $N1 = 0$, $N2 = 1$ and $Q = 1$. While CLK is held at 1, the input is switched to 0/5, so that $N5 = HiZ1$. Next, $N5$ is discharged to ground through a capacitor, which turns off $M5$ and turns on $M7$, pulling $N1$ to 1. This turns on $M2$ and turns off $M4$, and $N2$ discharges to ground. The result is that while $CLK = 1$, if the input switched leaving $N5$ or $N6$ in a HiZ1 state, it is possible to switch the output due to coupling noise. The same experiment, when repeated with the presence of the keeper transistors $M9$ and $M10$, does not result in the output being disturbed.

3.2 Asynchronous Preset, Clear and Scan implementations

The PNAND cell is a multi-input flip-flop, therefore it is necessary for it to have typical features of a D-flip-flop such as asynchronous preset and clear as well as scan. Evidently the PNAND cell operates quite differently compared to a master-slave D-flip-flop. In master-slave DFF, the asynchronous preset and clear inputs must change the state of both master and slave to make it independent of the state of the clock.

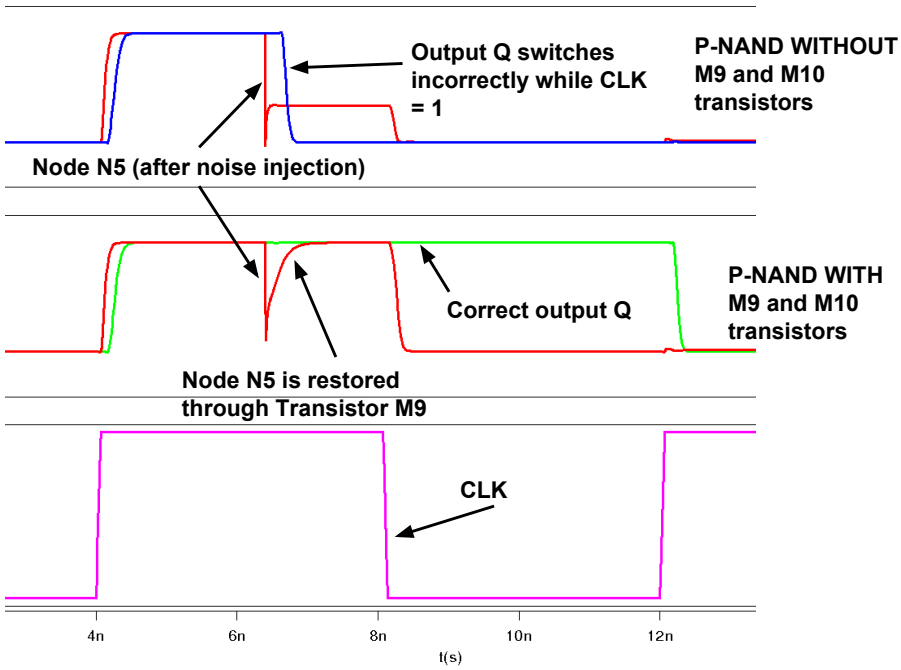


Figure 3.2: Importance of Feedback Transistors in the Input Networks of PNAND

However in PNAND, the SR latch always follows the nodes $N1$ and $N2$ irrespective of the clock. Hence the preset,clear and scan mechanisms need to change the state of the sense amplifier.

Fig. 3.3 shows the implementation of asynchronous preset PREZ (active low) and asynchronous clear CLR (active high). The additional transistors required in the basic PNAND to implement preset and clear are shown with thicker lines. When CLK is 0, $N1$ and $N2$ are both 1. **When PREZ is asserted (set to 0)** and CLR = 0, the signal PRE rises to 1. Therefore transistor $M9$ is off cutting supply to node $N1$. $N1$ discharges through the NMOS transistor $M10$ causing cell output Q to rise. **When CLR is asserted** (CLR = 1), node $N2$ discharges through transistor $M12$ causing latch output Q to fall. When both CLR and PREZ are asserted, then CLR alone operates and PREZ is filtered out indicating this is CLR dominated circuit. The NOR gate can actually be moved outside the cell and shared across all cells

which allows the cell to have a single active high preset (PRE) signal. When CLK = 1, the state of $N1$ and $N2$ can be opposite. In this case, the NMOS transistors $M10$ and $M12$ must be sized to overcome PMOSes $M1$ and $M2$.

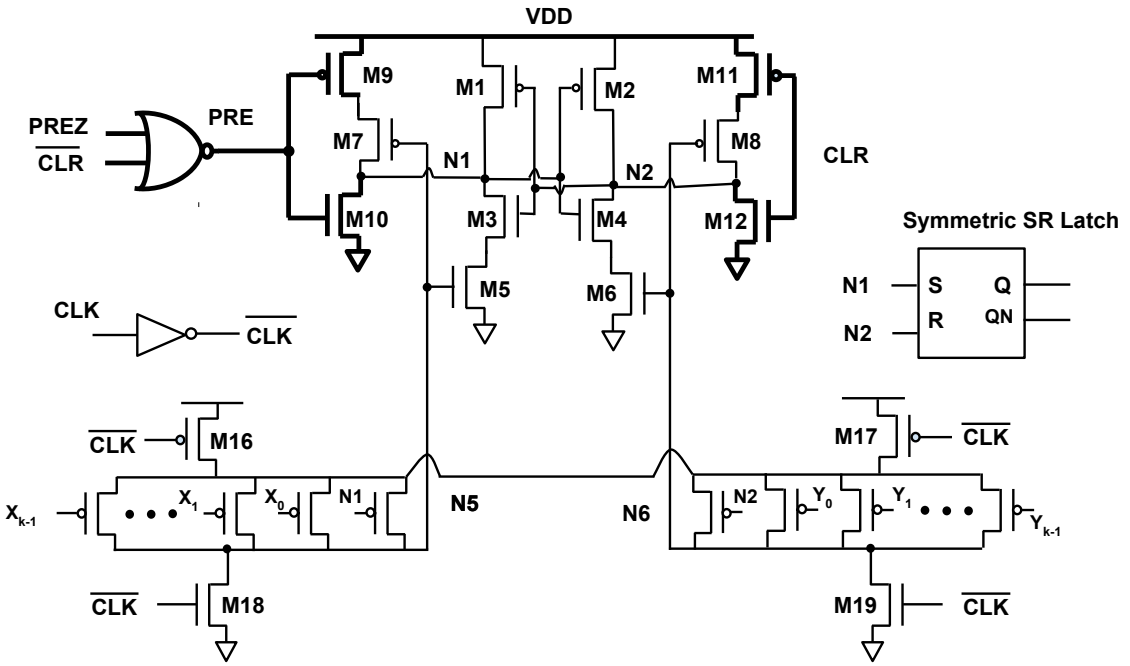


Figure 3.3: PNAND Cell Design with Asynchronous Set and Reset

If PNAND cells are to replace flipflops, scan capability is essential. The simplest way to make a D-FF scannable is to use a 2:1 mux that selects between the input D and the test input (TI), depending on whether or not the test mode is enabled (TE). This is not practical for a multi-input flip-flop like the PNAND cell. Although there exist several ways to implement scan for a PNAND cell, the one shown in Figure 3.4 has negligible impact on the cell's performance and robustness during normal operation. All other variations were significantly worse in this regard.

The additional transistors for scan are labeled as $S1$ through $S6$. In the normal mode, the signals TE (test enable) and TI (test input) are both 0, which disables the scan related transistors ($S1$ through $S4$), and reduces the circuit function to the one

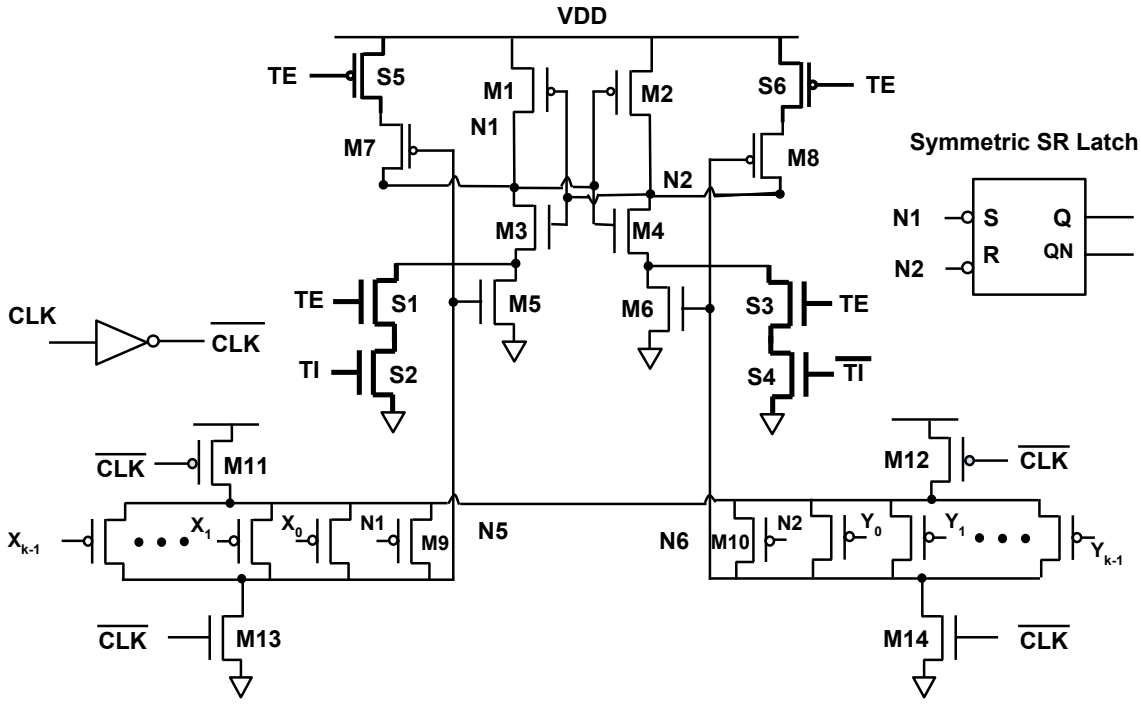


Figure 3.4: PNAND Cell Design with Scan

shown in Figure 3.1.

We can see that in the scan mode, the TE signal itself acts as a clock for a PNAND. In fact regular clock CLK must be held 0 for the scanning mechanism to work. Therefore, if a circuit has a mix of D-FFs and PNAND cells, the PNAND cells must be part of a separate scan chain. A common TE signal is used for both the scan chains. However the way this TE signal is operated is different from the conventional scanning mechanism. First the signal TE is held high and data is scanned into regular flipflops (conventional way). Once this is finished, the CLK is held 0 and following procedure is performed to stored data bits in PNANDs. Signal GTI (global test input) is the entry point for the scan data input to the PNAND chain.

1. Set CLK = 0 and TE = 0.

2. Set GTI = i'th bit of the input ($i = 0$ initially).

3. Set TE = 1. Each PNAND registers its TI input.

4. Set TE = 0.

5. Increment i and repeat until the end of stream.

Note that as long as CLK = 0, the toggling of TE signal alone does not alter the data already stored in the conventional D-Flipflop scan chain. Therefore at the end of this procedure both PNANDs and flipflops store the required set of bits and regular clocking can proceed. The pullup transistors S5 and S6 are included to eliminate a DC path during testing. In absence of these transistors, when TE is asserted ($0 \rightarrow 1$), while CLK = 0, $M7$ is active, and there is a DC path $VDD \rightarrow M7 \rightarrow M3 \rightarrow S1 \rightarrow S2 \rightarrow GND$.

Fig. 3.5 shows the waveforms obtained via SPICE simulation of a scan chain of four PNAND-9 cells. The CLK is set to 0. The nodes Q1, Q2, Q3 and Q4 are output nodes of 4 cells that are initialized to 0. The scan pattern being registered is (Q1, Q2, Q3, Q4) = 1010.

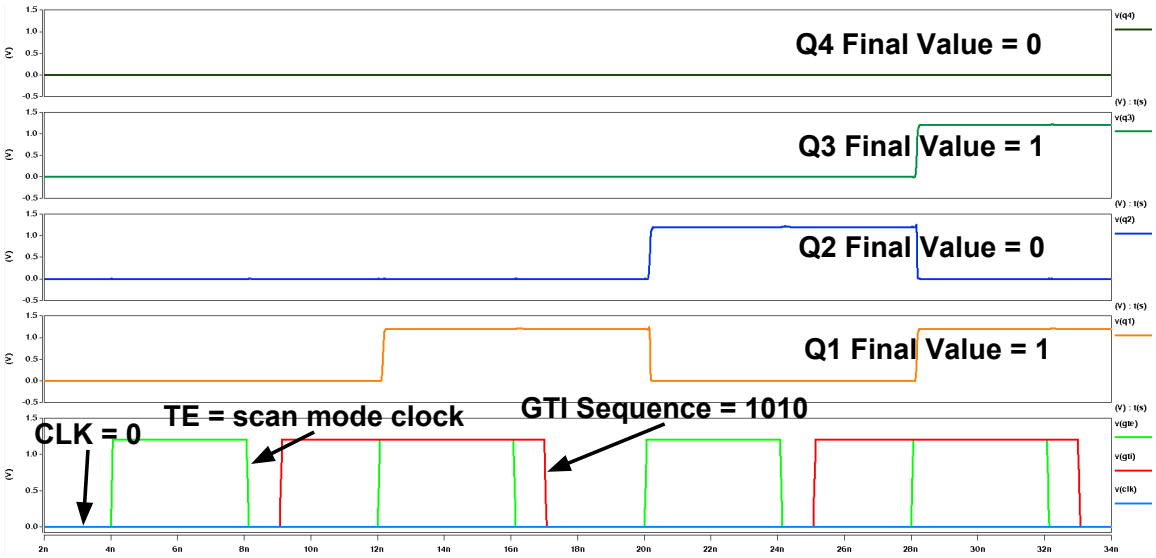


Figure 3.5: Operation of a Scan Chain Consisting of Four PNAND-9 Cells

3.3 Threshold Functions Realized by a PNAND

This section explains how PNAND cells can be used to compute a restricted set [1] of threshold functions. This is done by connecting the appropriate signals to the inputs in the LIN and RIN. If x_i and y_i are the signals that drive the gates of the transistors in the LIN and RIN, respectively, then the function of the PNAND cell can be expressed as

$$\begin{aligned} &PNAND(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_k) \\ &= \begin{cases} 1 & \text{if} \sum_{i=1}^{k} x_i > \sum_{i=1}^{k} y_i, \\ 0 & \text{if} \sum_{i=1}^{k} x_i < \sum_{i=1}^{k} y_i, \\ X & \text{if} \sum_{i=1}^{k} x_i = \sum_{i=1}^{k} y_i. \end{cases} \end{aligned} \tag{3.1}$$

X in Equation 3.1 denotes an *indeterminate* state. This condition must be eliminated. Let $f(X) = [\boldsymbol{w}, T]$, be a threshold function, and $\delta = \max_{X \ni f(X)=0} \mid \boldsymbol{w}'X - T \mid$. That is δ is the magnitude of the largest weighted minterm in the off-set of f. In (Siu *et al.*, 1995) it shown that that $(\boldsymbol{w}'X - T + \delta/2) \neq 0$ and $\text{sgn}^+(\boldsymbol{w}'X - T) = sgn^+(\boldsymbol{w}'X - T + \delta/2)$. This means that if the threshold T is replaced by $T + \delta/2$, then the inequality $\boldsymbol{w}'X \geq T$ in (1.2) becomes a strict inequality $\boldsymbol{w}'X > T + \delta/2$, without changing the threshold function. If the weights $\boldsymbol{w}$ and threshold T are integers with minimum possible values, then $\delta = 1$. Then

$$\boldsymbol{w}'X \geq T \equiv \boldsymbol{w}'X \geq T - 0.5 \equiv 2\boldsymbol{w}'X \geq 2T - 1. \tag{3.2}$$

Therefore, the indeterminate state can be eliminated by doubling the weights and replacing the threshold T with $2T - 1$.

[1] It should be noted even though a given PNAND-k implements a limited set of functions, for any arbitrary threshold function H, there exists a k such that PNAND-k can implement H.

Let $W = \sum_{i=1}^{k} w_i$. The inequality in (3.2) can be realized *as is* by a PNAND in which the LIN has $2W$ transistors, whose gates are driven by the inputs, and the RIN has $2T-1$ active transistors, and $2(W-T)+1$ inactive transistors [2] . Hence a PNAND-2W will be required if the inequality (3.2) is used. As an example, consider the threshold function $f(a,b,c) = a \vee bc \equiv 2a+b+c \geq 2 \equiv 4a+2b+2c > 3$. If f is to be implemented with this inequality then a PNAND-8 will be necessary, with the following signal assignment.

$$\underbrace{\bar{a},\bar{a},\bar{a},\bar{a},\bar{b},\bar{b},\bar{c},\bar{c}}_{\text{Left Input Network}} \quad | \quad \underbrace{0,0,0,1,1,1,1,1}_{\text{Right Input Network}} \quad . \tag{3.3}$$

Any inequality equivalent to (3.2) can be realized by assigning the appropriate signals to the inputs networks of a PNAND cell. The choice of the inequality determines the number of inputs of the PNAND cell and will affect the area, power, delay and noise margin. In general, any constant appearing on the RHS of inequality (3.2) will correspond to that many pFETs with their gates driven by 0. Thus inequality (3.2) requires $2T-1$ transistors in the RIN with gates at 0. Eliminating constant 0 literals (pFETs driven by 0) from the LIN and RIN will result in the least number of transistors in the input networks that are required to realize the given function. This can be done by transferring $2T-1$ literals from the LIN to the RIN. After the transfer the total weight of the LIN and RIN will be $2W-2T+1$, and $2T-1$, respectively. Hence a PNAND-k, where $k = \max(2W-2T+1, 2T+1)$, will be the smallest PNAND (i.e. one with the least number of transistors in the LIN and RIN) that can realize the given threshold function. Note that $(1-x)$ in the integer domain is the same as $\bar{x}$ in the Boolean domain, and because the LIN and RIN consist of pFETs, a positive literal x in the inequality corresponds to a pFET driven by $\bar{x}$, and a negative literal

[2] The LIN and RIN have the same number of transistors to ensure a symmetric cell.

to a pFET driven by x. Consider again $f(a,b,c) = a \vee bc$.

$$\begin{aligned} 4a+2b+2c > 3 &\equiv 3a+b+c > 3-a-b-c \\ &\equiv 3a+b+c > (1-a)+(1-b)+(1-c). \end{aligned} \tag{3.4}$$

The signal assignment corresponding to (3.4) is

$$\underbrace{\bar{a},\bar{a},\bar{a},\bar{b},\bar{c},}_{\text{Left Input Network}} \quad | \quad \underbrace{a,b,c,1,1}_{\text{Right Input Network}} \tag{3.5}$$

which implies that a PNAND-5 can be used to realize f, instead of a PNAND-8 corresponding to (3.3). Since $2T-1$ is odd, and $2W$ is even, transferring $2T-1$ literals from the LIN to the RIN will result in the sum of weights in the LIN being odd. This will be the minimum number of inputs required in the LIN and RIN. Therefore, the library need only have PNAND cells with odd number of inputs in the LIN and RIN.

There can be several choices of $2T-1$ literals than can be transferred from the LIN to the RIN. Among these, those that assign each literal to both the LIN and RIN are preferred (see Subsection 3.4), and ties can be broken arbitrarily. We refer to this as the *optimal signal assignment* (OSA).

The previous section described a signal assignment (OSA) that results in the smallest PNAND to realize a given threshold function. In this section it is shown how the set of functions realized by PNAND-k under OSA, for $k = 3,5,7,9$ can be enumerated. Given k, the pairs of (W,T) that satisfy the condition $k = \max(2W - 2T+1, 2T+1)$ are given by

$$\begin{aligned} (W,T) = \Bigg\{ &\left(\frac{k+1}{2},1\right), \left(\frac{k+3}{2},2\right), \cdots, \left(k,\frac{k+1}{2}\right) \\ &\left(\frac{k+1}{2},\frac{k+1}{2}\right), \left(\frac{k+3}{2},\frac{k+1}{2}\right), \cdots \left(k-1,\frac{k+1}{2}\right)\Bigg\} \end{aligned} \tag{3.6}$$

Given the valid set of (W, T) pairs associated with a given k, the weights of the threshold functions that correspond to W is simply the set of partitions of the integer W, without regard to their order. For example, with $W = 3$, the partitions are $(2, 1)$ and $(1, 1, 1)$. Hence to enumerate the set of threshold functions that can be realized by a PNAND-k with the OSA, the set of valid (W, T) are determined, and for each W in that set, the set of integer partitions of W are enumerated. Table 3.1 shows the set of functions for $k = 1, 3, 5, \cdots$. A PNAND-3 can realize 3 non-degenerate threshold functions, and a PNAND-5 can realize 8 non-degenerate threshold functions. The number of non-degenerate functions for PNAND-7 and PNAND-9 are 18 and 42, respectively. Thus a library of PNAND-k cells with $k = 3, 5, 7, 9$ can realize 71 basic functions. Note that with threshold functions, it is easy to identify NPN equivalent functions. The minimum, positive form of the weights and threshold are stored in the library. Then if a NPN equivalent threshold function is discovered during technology mapping, its weights and threshold are easily converted to positive form, and then the library is searched for that combination. This results in a substantial increase in the number of possible functions implementable by a given PNAND cell. On the other hand, checking for NPN equivalence of arbitrary Boolean functions is an NP-hard problem whereas it is polynomial (in terms of n = number of inputs of the function) time problem when restricted to threshold functions.

3.4 PNAND Delay Characteristics

Unlike a traditional edge-triggered flipflop, the function implemented by a PNAND cell is determined by the signals connected to its inputs. Thus, without a priori knowledge of its use, delay characterization of a PNAND-k cell would have to consider each of the 2k inputs (see Figure 3.1) in the LIN and RIN as independent signals. This would be practically impossible to carry out. Fortunately, the delay of a PNAND cell

Table 3.1: Functions Realized by PNAND-K, for K = 3, 5, 7, 9

K	(W,T)	Weight; Threshold	Threshold fn	Boolean fn	Modified Threshold fn	Signal assignment
1	$(1,1)$	$(1;1)$	$a \geq 1$	a	$2a > 1$	$(a'\|a)$
3	$(2,1)$	$(1,1;1)$	$a+b \geq 1$	$a+b$	$2a+2b > 1$	$(a',a',b'\|b,1,1)$
	$(3,2)$	$(1,1,1;2)$	$a+b+c \geq 2$	$ab+ac+bc$	$2a+2b+2c > 3$	$(a',a',b'\|b,c,c)$
	$(2,2)$	$(1,1;2)$	$a+b \geq 2$	ab	$2a+2b > 3$	$(a',1,1\|a,b,b)$
5	$(3,1)$	$(1,1,1;1)$	$a+b+c \geq 1$	$a+b+c$	$2a+2b+2c > 1$	$(a',a',b',b',c'\|c,1,1,1,1)$
	$(3,3)$	$(1,1,1;3)$	$a+b+c \geq 3$	abc	$2a+2b+2c > 5$	$(a',1,1\|a,b,b,c,c)$
	$(4,2)$	$(2,1,1;2)$	$2a+b+c \geq 2$	$a+bc$	$4a+2b+2c > 3$	$(a',a',a',b',c'\|a,b,c,1,1)$
	$(4,2)$	$(1,1,1,1;2)$	$a+b+c+d \geq 2$	$ab+ac+\cdots+cd$	$2a+2b+2c+2d > 3$	$(a',a',b',c',d'\|b,c,d,1,1)$
	$(4,3)$	$(2,1,1;3)$	$2a+b+c \geq 3$	$a(b+c)$	$4a+2b+2c > 5$	$(a',a',a',b',c'\|a,b,c,1,1)$
	$(4,3)$	$(1,1,1,1;3)$	$a+b+c+d \geq 3$	$abc+abd+acd+bcd$	$2a+2b+2c+2d > 5$	$(a',b',c',1,1\|a,b,c,d,d)$
	$(5,3)$	$(2,1,1,1;3)$	$2a+b+c+d \geq 3$	$ab+ac+ad+bcd$	$4a+2b+2c+2d > 5$	$(a',a',b',c',d'\|a,a,b,c,d)$
	$(5,3)$	$(1,1,1,1,1;3)$	$a+b+c+d+e \geq 3$	$abc+abd+\cdots+cde$	$2a+2b+2c+2d+2e > 5$	$(a',b',c',d',e'\|a,b,c,d,e)$
7	**18 functions**					
9	**42 functions**					

depends primarily on the *number* of inputs that have a certain characteristic, not on which ones have that characteristic [3] . This allows for efficient characterization of its delay and power, and development of a cell library for use in technology mapping.

Without loss of generality, assume that the number of active transistors in the LIN is greater than the number in the RIN. Similar to other differential threshold gates (Strandberg and Yuan, 2000b; Padure *et al.*, 2001a; Leshner, 2010), the delay of a PNAND cell is the sum of two delays: (1) the input network delay, which is the time from the clock edge to when $N5$ transitions to 1; (2) the sense amplifier delay, which is the time $N5$ turns on $M5$ and turns off $M7$, and $N1$ discharges to ground. For a given number of active devices in either the LIN or the RIN, the input configurations that exhibit the worst-case total delay or power are those in which the difference in the number of active devices between the LIN and RIN is minimum, i.e. 1.

For PNAND-k, the input network delay is maximum for the 1/0 configuration as this requires charging $N5$ to 1 only through one pFET. Let M_k denote the maximum number of active devices that can occur in either the LIN or RIN. The value of M_k depends on the function and the signal assignment, with smaller values being preferred. In general $M_k \in [(k + 1/2), k]$. The sense amplifier delay is maximum for the $M_k/(M_k - 1)$ configuration because this represents the maximum possible *contention* between the two input networks. The configuration that results in the maximum contention is also the one that results in the maximum power consumption of the cell. For the OSA, $M_k = (k + 1)/2$, which is the minimum possible value, and the configuration with the maximum contention will be $\frac{k+1}{2}/\frac{k-1}{2}$. Thus for a PNAND-k, the configurations that need to be examined for delay and power characterization are $1/0, 2/1, \cdots, \frac{K+1}{2}/\frac{K-1}{2}$.

[3]This is indeed true of all differential threshold logic gates that rely on the relative difference in impedances between two paths determined by the inputs, e.g. SCSDL (Strandberg and Yuan, 2000b), DCSTL (Padure *et al.*, 2001a), TLL (Leshner *et al.*, 2010; Leshner, 2010), among others

As stated earlier, signal assignment can have a substantial impact on the delay and power of a PNAND cell. Moreover, not all functions would exhibit the same worst-case input configurations. Tables 3.2 and 3.3 show two different signal assignments for the function $f(a, b, c) = a \vee bc$, and the input configurations associated with each minterm. The signal assignment in Table 3.2 has both worst-case input configurations present, namely 1/0 for maximum input network delay, and 4/3 for maximum contention. The signal assignment in Table 3.3 is preferred because its worst-case input network delay (corresponding to 2/1) and sense amplifier delay (corresponding to 3/2) are lower than in assignment in Table 3.2. Thus the cell library has multiple instances of each PNAND-k, corresponding to the various worst-case input configurations that are possible, with each instance sized appropriately for each configuration. During technology mapping, when a PNAND-k cell is selected to replace a threshold subcircuit, the optimal instance of that cell is used. Table 3.4 provides a comparison of the total delay (layout extracted setup time plus clock-to-output delay) relative to a D-FF of each of the PNAND cells for various possible input configurations. Therefore a value in the table < 1 indicates that PNAND is faster whereas a value $>= 1$ indicates that DFF is faster. We can see that PNAND cells (that compute a complex threshold function) are almost always faster than a D-FF which computes mere identity (buffer) function of its D-input.

3.5 Robustness

The evaluation of a threshold function by a PNAND cell is based on the relative difference in conductance between the LIN and RIN. Ideally, these conductances should be determined only by the number of active devices in the input networks. Thus, asymmetry in a PNAND cell might lead to an imbalance that would incorrectly favor one side over the other, independent of the input. Process corners as well

Table 3.2: A *Bad* Signal Assignment for $f(a,b,c) = a \vee bc$ Denoted by $a + 2b + 2c > 3 - 3a \rightarrow (a', b', b', c', c'|a, a, a, 1, 1)$

abc	L	R
000	0	3
001	2	3
010	2	3
011	**4**	**3**
100	**1**	**0**
101	3	0
110	3	0
110	5	0

Table 3.3: A *Good* Signal Assignment for $f(a,b,c) = a \vee bc$ Denoted by $3a + b + c > 3 - a - b - c \rightarrow (a', a', a', b', c'|a, b, c, 1, 1)$

abc	L	R
000	0	3
001	1	2
010	1	2
011	**2**	**1**
100	**3**	**2**
101	4	1
110	4	1
110	5	0

Table 3.4: Delays of PNAND-K w/wo Scan, Relative to D-FF in 65nm LP process. Layout Extracted Netlist Simulated at PVT = SS/1.1V/105°C/, Input and Clock slews = 70ps, Output Load = 20fF.

$K \rightarrow$	3		5			7				9				
#LIN/#RIN	1/0	2/1	1/0	2/1	3/2	1/0	2/1	3/2	4/3	1/0	2/1	3/2	4/3	5/4
No Scan	0.70	0.63	0.89	0.76	0.74	0.95	0.79	0.75	0.78	1.01	0.83	0.79	0.81	0.84
Scan	0.74	0.67	0.97	0.81	0.78	1.02	0.85	0.82	0.85	1.12	0.91	0.87	0.89	0.93

process variations however, can not only delay the computation, but also disrupt the race causing an invalid output and making PNAND circuits more sensitive to process variations. The process or PVT corners at which the PNAND cells were evaluated, using a commercial 65nm process were: P $\in$ {slow, typical, fast}, V $\in$ {1.1V, 1.2V, 1.3V}, and T $\in \{-40°C, 25°C, 105°C\}$. All the PNAND cells, for each of the input configurations shown in Table 3.4 were successfully simulated at three PVT corners: $SS/1.1V/105°C$, $TT/1.2V/25°C$, $FF/1.3V/-40°C$.

Next, the impact of process variations was examined through Monte Carlo simulations which included global and local variations. A global variation in a circuit parameter p refers to the same amount of statistical perturbation in p for all circuit components that are affected by p. On the other hand, a local variation or mismatch in p means that the amount of statistical perturbation in p varies, independently from component to component. The symmetry of the PNAND cells makes them much more sensitive to local mismatch than conventional static CMOS logic. To ascertain the impact, 100,000 Monte Carlo simulations were performed assuming both global and local variations in the device V_t, and β, with the magnitude of variations obtained from foundry data (referred to as a *statistical corner*). The voltage and temperature were chosen to be $1.2V$ and $-40°C$. The low temperature corner was chosen because the reliability is directly proportional to the impedance difference of the networks which in turn is directly proportional to the average resistance of each transistor in the input networks. This resistance is the lowest (worst case) at very low temperatures. The smaller the impedance difference between the input networks, the lower is the voltage difference amplified by the sense-amplifier and lower the reliability of the cell. Table 3.5 shows the results of the simulations. The data indicates that D-flip flop has worse spread of delays compared to some of the lower cases such as 3/2, 2/1 and 1/0 irrespective of size of input networks. However the larger cases 5/4 and

4/3, as expected, have larger spread compared to the D-flip flop and smaller cases of PNAND. Note that none of the cells have any functional failures in 100K Monte-Carlo simulations i.e. every trial functioned correctly albeit with different delays.

3.5.1 Comparison with TLL

Among the conductance based, differential threshold logic gates, TLL has been shown significantly better in terms of power dissipation, delay and robustness against process variations (Leshner *et al.*, 2010; Leshner, 2010). Furthermore, the PNAND architecture described above most closely resembles the TLL architecture, and for this reason, we elaborate on how PNAND significantly improves on the TLL design.

In TLL architecture, the clock input directly feeds drain terminals of transistors. Due to this the capacitive load on the clock is variable. This is ill suited for characterization of clock pin capacitance and thereby the design of the clock tree. In PNAND, the clock input drives only one inverter, and therefore its clock pin capacitance is fixed and slightly smaller than that of the D-flip-flop. The TLL does not consider the problem of coupling noise that can alter the computed value due to floating nodes. As discussed in Section 3.1 and demonstrated in Figure 3.2, the feedback in the PNAND cell eliminates this problem. TLL also behaves as a multi-input edge triggered flipflop that computes a threshold function on the clock edge. Its scan mechanism described in (Leshner, 2010) introduces additional transistors on the normal evaluation path, significantly reducing its robustness and speed. In contrast, the scan mechanism of the PNAND is completely non-intrusive and has minimal impact on its robustness and delay. Finally, the method of signal assignment (Leshner, 2010) (referred to as "balanced signal assignment") does not yield the *best worst-case* for several threshold functions. For example, the threshold function $F(a, b, c, d) = [3, 2, 1, 1; 3]$ exhibits the worst case of 5/4 for the balanced assignment which is

Table 3.5: Delay Distribution of Layout Extracted D-FF and PNAND Cells Subject to Global Variations and Local Mismatch in 100,000 Monte-Carlo Trials.

	Setup time (ps)	μ C2Q (ps)	σ (ps)	$< 0\sigma$	$[0\sigma,1\sigma)$	$[1\sigma,2\sigma)$	$[2\sigma,3\sigma)$	$[3\sigma,4\sigma)$	$[4\sigma,5\sigma)$	$[5\sigma,6\sigma)$	$> 6\sigma$
D-FF	54	177	5	15598	68943	12680	2361	367	45	6	0
PNAND-3 1/0	-14	191	5	16023	67753	13754	2244	224	2	0	0
PNAND-3 2/1	-14	175	4	16635	67844	13117	2246	145	13	0	0
PNAND-5 1/0	-5	219	6	15777	68079	13742	2187	186	29	0	0
PNAND-5 2/1	-5	190	5	15593	68195	13722	2207	268	15	0	0
PNAND-5 3/2	-5	187	5	15310	69193	12634	2478	374	11	0	0
PNAND-7 1/0	-2	228	7	15785	68814	12551	2622	228	0	0	0
PNAND-7 2/1	-2	196	5	15515	68807	13134	2348	196	0	0	0
PNAND-7 3/2	-2	189	5	15343	68795	13261	2321	265	15	0	0
PNAND-7 4/3	-2	191	6	14306	71695	10572	2469	747	151	30	30
PNAND-9 1/0	1	267	8	15473	68524	13223	2595	170	15	0	0
PNAND-9 2/1	1	225	6	16180	68026	13470	2168	141	15	0	0
PNAND-9 3/2	1	213	5	16123	68516	12759	2383	219	0	0	0
PNAND-9 4/3	1	215	6	15352	69096	12535	2583	403	15	16	0
PNAND-9 5/4	1	222	8	13575	72823	10253	2487	461	263	47	91

$(a', a', a', b', b', b', b', c', d'|a, a, a, c, d, 1, 1, 1, 1)$. On the other hand, the worst input case of F for the proposed optimal signal assignment $(a', a', a', a', b', b', c', d', d'|a, a, b, b, c, 1, 1, 1, 1)$ is 4/3 which is substantially more robust than 5/4 case.

3.6 PNAND-1 (KVFF) Design

A single input PNAND cell (PNAND-1) implements a single threshold function viz. buffer function ($F(x) = x$). Therefore a single input PNAND cell is functionally equivalent to a conventional master-slave D-flipflop. However the design complexity of a single input PNAND can be substantially *reduced* with the knowledge that the worst input case being exercised is (1/0) which can make PNAND as robust as MSFF. The result of this is a new design of an edge-triggered flipflop called as KVFF. The architecture of KVFF is shown in Fig. 3.6.

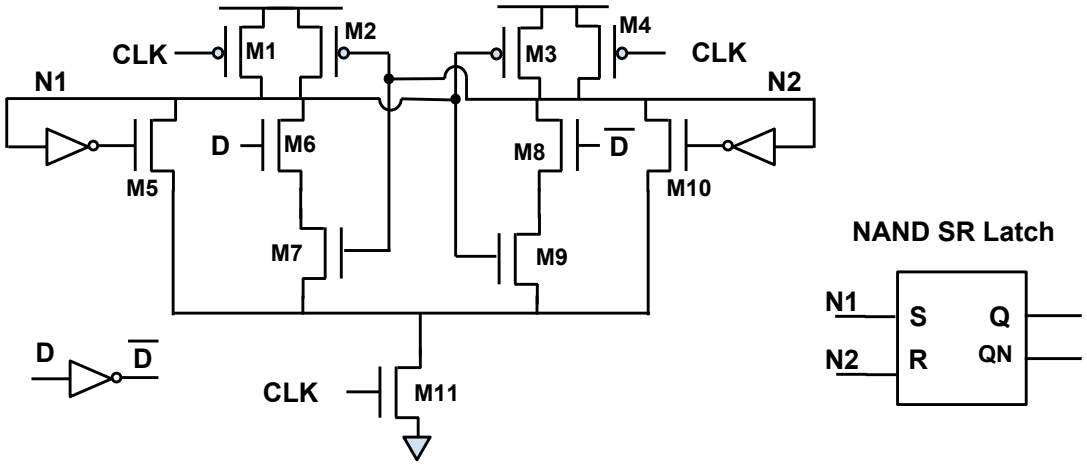


Figure 3.6: KVFF Architecture

Operation of a KVFF

When the clock signal CLK = 0, following is the steady state of the circuit.

1. transistor M11 is OFF.

2. Since M11 is OFF, nodes N1 as well as N2 have no path to ground.

3. Transistor M1 and M4 are ON and pull up both nodes N1 and N2 to logic 1.

4. The NAND SR latch driven by N1 (=1) and N2(=1) maintains its last state at the primary output Q of the flip-flop.

5. The keeper transistors M5 and M10 are OFF.

When CLK transitions from 0 to 1, the flipflop latches input D as follows. Assume D = 1.

1. Transistor M11 turns ON whereas M1 and M4 (pullup transistors) turn OFF.

2. Transistor M6 and M7 are ON.

3. This causes node N1 to discharge and pulled down to logic 0.

4. Since $\bar{D} = 0$, transistor M8 is OFF which causes node N2 to float at 1.

5. As soon as node N1 falls to logic 0, PMOS transistor M3 (driven by node N1) turns ON. This causes node N2 to remain firmly at logic 1.

6. As a result, the circuit stabilizes in a state where N1 = 0 and N2 = 1.

7. The NAND SR latch responds to inputs N1 (=0) and N2 (=1), and sets the output Q to logic 1. As a result, input D is latched and appears at output Q.

8. The keeper transistor M5 turns ON, due to which node N1 has another path to ground through M5. Note that even if input D switches at this point, the node N1 remains firmly at logic 0 and node N2 remains firmly at logic 1.

When D = 0, the nodes N1 and N2 switch roles in above series of steps and SR latch sets its output Q to 0.

3.6.1 Scan & Asynchronous Preset and Clear Architecture

The scan feature can be introduced for KVFF using same idea as conventional D-flipflop where scan data input (TI) and D input are connected to a 2:1 MUX with scan enable (TE) input connected as a select signal of the MUX. Asynchronous preset (PRE) and clear (CLR) however require additional circuitry. Fig. 3.7 shows a CLR dominated asynchronous preset and clear implementation.

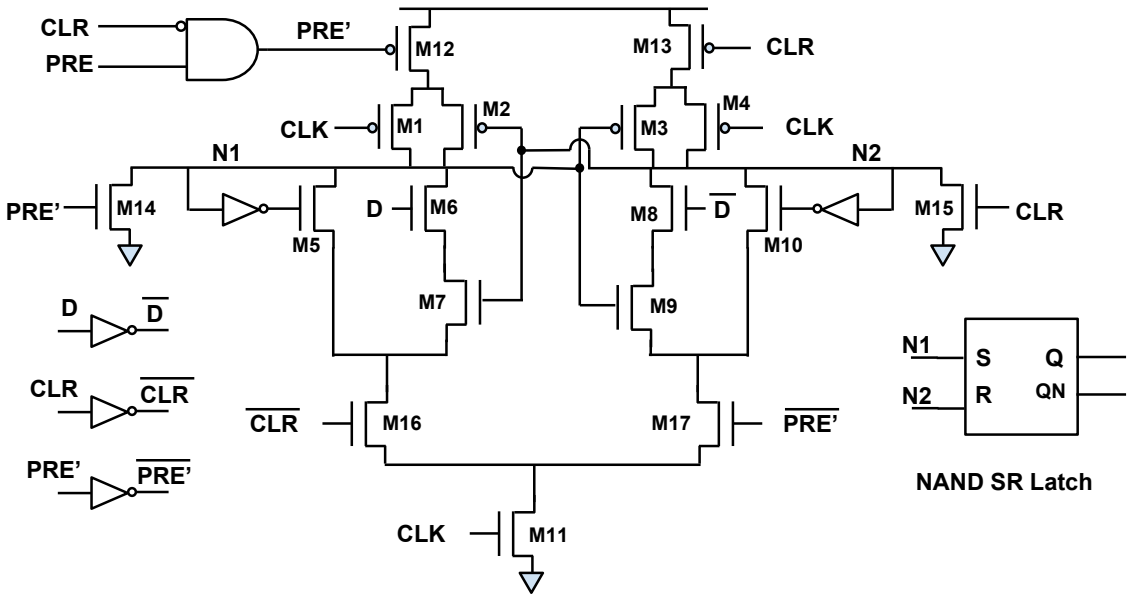


Figure 3.7: KVFF Architecture with Asynchronous Preset and Clear

There is an internal signal called PRE' derived from CLR and PRE. Note that PRE' = PRE if CLR = 0. When CLR = 1, PRE' = 0. Therefore both clear and preset cannot be asserted high. So the only values assumed by PRE' and CLR are (0,0), (0,1) and (1,0). Note that both of these are active high signals.

1. When CLR = PRE = PRE' = 0, transistors M12, M13, M16 and M17 are ON and M14 and M15 are OFF. The switch level view of the circuit reduces to regular KVFF from Fig. 3.6 and works as regular flipflop.

2. When CLR = 0 and PRE = 1, we have PRE' = 1. There are multiple possible states of the cell before PRE is asserted.

   (a) Suppose CLK = 0 i.e. cell is in reset state. In this case, M11 is OFF therefore no path to ground exists for N1 or N2 and both of these nodes are asserted high through M1 and M4 transistors. In this state when PRE' is asserted 1, then transistors M12 turns OFF. Therefore N1 has no path to VDD. On the other hand, N1 is provided a path to ground through M14 which is ON. As a result, node N1 discharges to assume logic state 0. At the same time, $\overline{PRE'}$ is 0 and transistor M17 is OFF. Therefore node N2 has no path to ground. Since CLR = 0, and N1 = 0, node N2 charges to VDD through M3 and M13 both of which are ON. As a result (N1,N2) = (0,1) which sets the NAND latch asserting output Q to 1. When PRE is de-asserted, node N1 and N2 receive their previous states viz (1,1) and SR latch maintains the Q = 1 state until next rising edge of the clock.

   (b) Suppose CLK = 1. Therefore the cell is in evaluation state. There are two sub-cases depending on the values of (N1,N2).

      i. Suppose (N1,N2) = (0,1). Output Q = 1 in this state. Asserting PRE (and thereby PRE') in this state, turns M14 and ON and M12 OFF. Therefore N1 has no path to VDD and has a path to ground. The opposite is true for N2 i.e. it has no path to ground and has a path to VDD. Therefore N1 and N2 simply maintain their state and output Q remains 1.

      ii. Assume (N1,N2) = (1,0). Output Q = 0 in this state. When PRE and therefore PRE' is asserted, node N2 loses path to ground and starts to float. At the same time, N1 is discharged through M14 and therefore

transistor M3 is turned ON. Therefore N1 and N2 flip their states i.e. new state is (N1,N2) = (0,1) which causes output Q to rise to 1.

3. When CLR = 1, then PRE' = 0 irrespective of the value of PRE. In this case, the operation is similar to case 2 above except that the final state of nodes (N1, N2) = (1, 0) irrespective of their current state. This causes the latch to reset and output Q is de-asserted i.e. pulled low.

Advantages of KVFF

KVFF is better than both the ubiquitous master slave D-flipflop (MSFF) and the differential mode strong-arm flipflop (SAFF). In general, the KVFF is faster and more energy efficient than the MSFF and more robust than the SAFF. The SAFF (Fig. 3.8) has an equalizer NMOS transistor. This results in the differential feedback (sense-amplifier) to evaluate a 2-1 case (both N1 and N2 partially discharge before settling in opposite values) upon clock trigger. The sizing of transistors determines the robustness, despite the fact that SAFF is a single input threshold gate. On the other hand, the operation of the KVFF is similar to PNAND with a 1/0 input case (N1 discharges whereas N2 is held at 1), which is much more robust. Table 3.6 shows the result of 100,000 Monte Carlo trials at 0.8V, -40C and statistical corner. It shows that the SAFF has *functional failures* because an incorrect bit was latched due to process variations. Even in the instances that correctly latched the input bit, the delay of SAFF had a very large standard deviation. Neither the KVFF nor the MSFF had any failures at this corner.

For comparison of energy efficiency, we selected an MSFF cell from a commercial standard cell library and we designed, sized and laid out a KVFF and an SAFF cells with same drive strength. We chose the worst case PVT corner for all of the simulations which were SS/1.1V/105C.

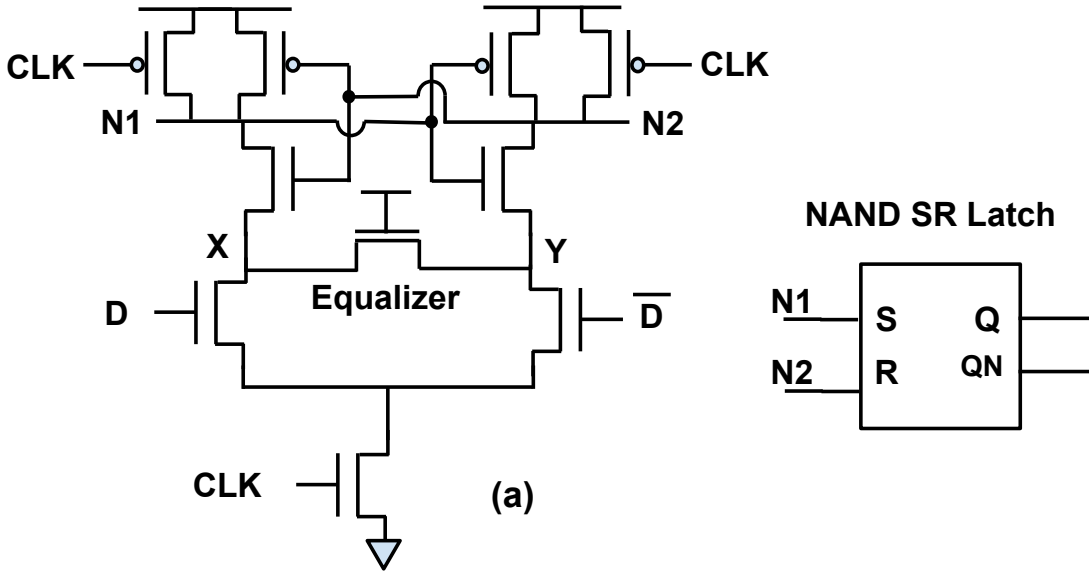


Figure 3.8: Differential Mode Strong Arm Flipflop Architecture

Table 3.6: 65nm Technology Comparison of Clock-to-Q delay Across Process Variations. The Simulation Corner is Statistical at 0.8V and -40C for 100,000 Monte-Carlo Trials. All Delays in Picoseconds

	# Failures	Max	Min (ps)	Mean (ps)	Stdev (ps)
MSFF	0	773	376	525	45
SAFF	2235	3859	360	581	192
KVFF	0	928	435	580	57

To satisfy the setup time constraints, the total delay, which is the sum of the clock-to-Q delay (t_{c2q}) and the setup time (t_{su}) must be considered. Thus we use $t_{tot} = t_{su} + t_{c2q}$ to compare the delay of all the flipflops.

Table 3.7 shows the delay, energy and energy delay product (EDP) comparison for the flipflops. We can see that KVFF is 25% faster (t_{tot}) compared to MSFF of same drive strength. Although slightly bigger in layout size, KVFF is 20% more energy efficient (smaller EDP) compared to MSFF.

Table 3.8 shows additional data to compare the DFF, the KVFF and PNAND cells

Table 3.7: 65nm Technology Design Comparison. The Simulation Corner is Slow/slow, 1.1V VDD and $105°C$. The Load Cap is $20fF$. Signal Transition Times are $70ps$. Identical Drive Strengths were used for the Flipflops.

	t_{su} (ps)	t_{c2q} (ps)	t_{tot} (ps)	Energy (fJ)	EDP $(fJ \times ps)$	Area (μm^2)
MSFF	90	264	354	15	5245	7.8
SAFF	-4	258	254	13	3271	8.32
KVFF	-11	277	266	16	4233	9.36

Table 3.8: 65nm Technology Comparison of Flipflop Characteristics. The Load Cap is $20fF$. All Input Signal Slews are $5ps$. Identical Drive Strengths used for all the Flipflops.

	Setup Time (ps)			Hold Time (ps)			C2Q Delay (ps)			Area (um^2)
	fast	slow	typ	fast	slow	typ	fast	slow	typ	
DFF	40	115	70	15	-5	5	83	219	133	7.8
KVFF	-6	-16	-10	39	114	67	91	242	145	10.4
PNAND-3	-17	-46	-28	28	73	45	86	221	135	19.76
PNAND-5	-11	-35	-17	27	73	44	93	244	148	21.84
PNAND-7	-6	-12	-8	27	73	44	98	256	155	24.96
PNAND-9	-1	-0.9	0.9	29	76	47	118	305	187	27.04

of different number of inputs. Their timing are compared at three P/V/T corners, referred to as typical (typ), slow and fast, where typ P/V/T is TT/1.2V/25C, slow P/V/T is SS/1.1V/105C and fast P/V/T is FF/1.3V/-40C.

Above results indicate that the PNAND cells are much faster compared to the conventional DFF. Note that the delays of the conventional DFF are *without any logic* driving it. The PNANDs on the other hand, compute complex threshold functions. Therefore replacing the logic and a flipflop with a PNAND generates a large timing slack which reduces the size of the the feeder logic.

Chapter 4

TECHNOLOGY MAPPING WITH THRESHOLD GATES

So far we have discussed the architecture, operation and characteristics of individual PNAND cells. This chapter describes how to incorporate PNAND cells into a general ASIC circuit with the goal of reducing area and power without sacrificing performance. Since the resulting circuits will consist of conventional standard cells (e.g. NAND, NOR, AOI, MUX, etc.), and PNAND cells, they will be referred to as *hybrid* circuits, and the process of generating them will be referred to as *hybridization*.

A PNAND cell can replace a single output sub-circuit whose function is one of the threshold functions in the library, and whose output has a clock cycle latency of one. Figure 4.1 shows three different circuits that can be replaced by a single PNAND cell. Note that replacing circuits shown in (b) and (c) with a single PNAND will reduce the total number of flip-flops in the resulting hybrid circuit, which may result in additional reduction in power. However one drawback of such replacement is it would not be possible to perform a simple functional equivalence check between the original circuit and its hybrid version, because the one-to-one correspondence between the DFFs and PNANDs would be lost. For this reason, hybridization is restricted to maintain the correspondence. Hence only the replacement of the type shown in Figure 4.1 (a), i.e. replacement of a single flip-flop along with some or all of its fan-in cone by a PNAND, is allowed.

Existing tools are not capable of performing synthesis or technology mapping with PNAND cells for the following reasons. First, the function of a PNAND cell is determined only after signals are connected to its inputs. Even if the set of functions realized by all the PNAND cells is enumerated, the SOP representations of many

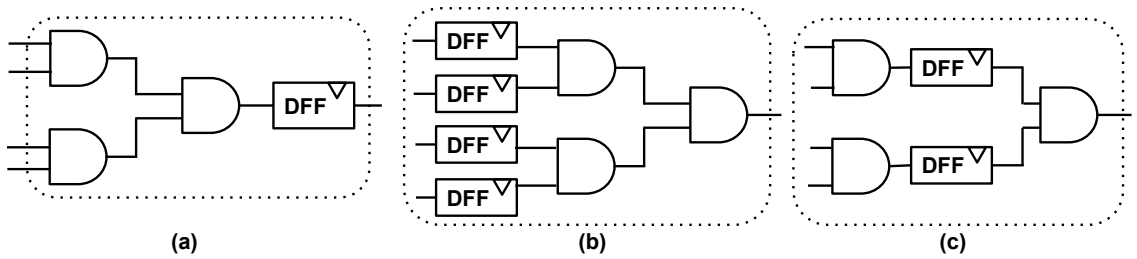


Figure 4.1: Subcircuits Replaceable by a Single PNAND Cell

functions are prohibitively large. Second, existing tools do not distinguish between a threshold function from any other Boolean function. This makes the task of determining the NPN equivalence of a threshold subcircuit and one of the functions realized by PNAND cells in the library, computationally prohibitive, because the number of input variables of some of the threshold functions implementable by PNAND is large (≥ 6). The approach followed here is to *preprocess* a conventional netlist by replacing suitably chosen subcircuits with PNAND cells, and then optimize the resulting netlist using the conventional synthesis and physical design tools, treating the PNAND cells as non-removable function blocks, but using their timing, power and capacitance data present in the characterized libraries.

4.1 PNAND vs Conventional Circuits

Comparing a PNAND with a D-FF alone does not demonstrate its full advantages because a PNAND embeds a multi-input logic function, which might require several levels of logic when synthesized with conventional logic cells. To illustrate this, consider the circuit shown in Figure 4.2. The cone of logic feeding the output D-FF is the threshold function $f(a, b, c, d, e) = [2, 1, 1, 1, 1; 4]$, whose netlist was synthesized using a 65nm technology. The output D-FF and the logic cone can be replaced with a PNAND-7. After synthesis, and place & route using a commercial 65nm library, both the netlists were extracted, simulated and respective circuit parameters were

obtained at the PVT corner $SS/1.1V/105°C$. As is evident, the PNAND cell has substantially lower total delay, as well as lower dynamic and leakage power. There are several reasons for this. First, replacement of the logic cone with a single cell has the obvious advantage of eliminating the logic cells thereby eliminating the internal switching activity of the cells, and reducing the leakage. Second, a PNAND cell exhibits a lower input and clock capacitance. The lower input capacitance and the reduced total delay (equal to delay of the logic absorbed plus setup time of D-flip-flop minus setup time of PNAND) would result in additional slack for any feeder logic, allowing synthesis tools to shrink the feeder to absorb the additional slack. In this example there is no feeder logic because the entire cone was a a threshold function (by design). However the feeder logic would be present in general circuits because threshold functions are a small subset of Boolean functions, and the threshold cell library is limited.

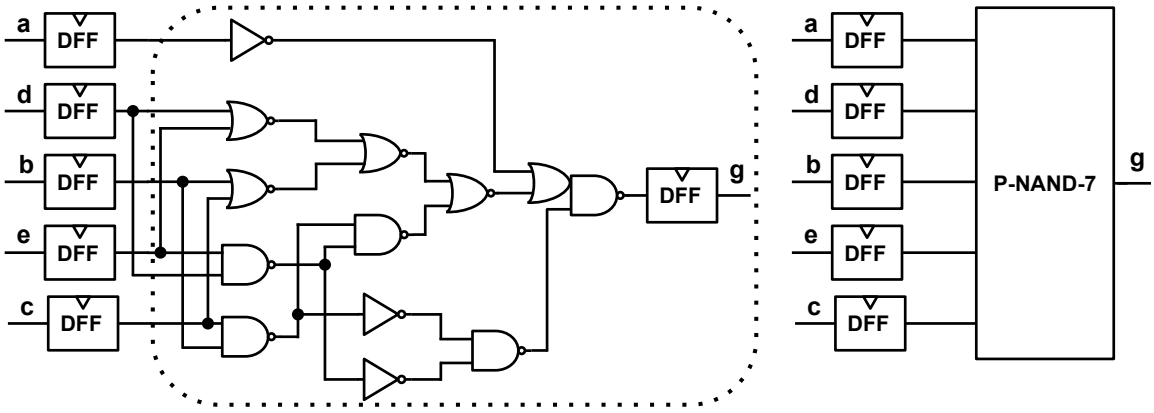


Figure 4.2: Comparison of PNAND with Functionally Equivalent Network of Standard Cells

Besides reduction in dynamic power, delay and area, the PNAND circuit also exhibits significantly lower leakage. To explore the dependence of leakage on the input state, both circuits were simulated for all 32 input test patterns. Figure 4.3 shows the variation in leakage over all input patterns. The minimum, maximum and

Table 4.1: Comparison of PNAND with Conventional Standard Cell Implementation of Circuit Shown in Figure 4.2 (at SS, 105°C, 1.1V)

Parameter	Delay	Cell Area	Power	Leakage	Total circuit cap
Conv.	515 ps	54 μm^2	105 μW	34 nW	78 fF
PNAND	276 ps	33 μm^2	76μW	9.3 nW	56 fF
Δ (%)	49%	28%	27%	72%	28%

average leakage of the conventional logic implementation are 3 to 4 times greater than that of the PNAND circuit. Moreover, the leakage of the PNAND circuit doesn't vary at all over the input patterns.

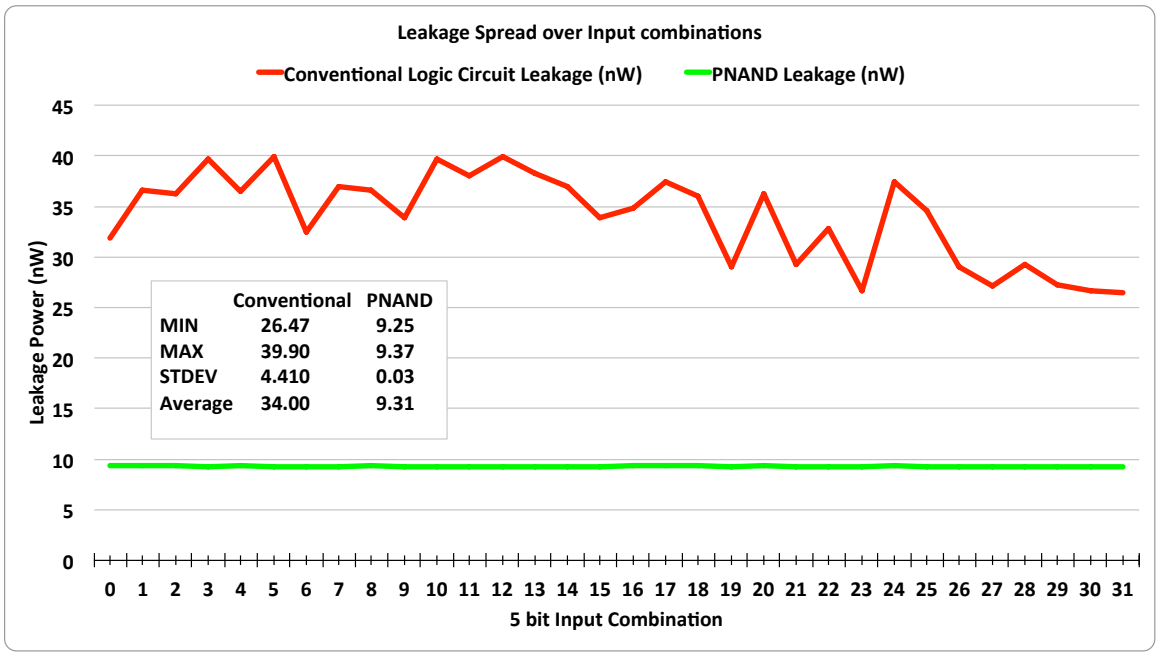


Figure 4.3: Comparison of Leakage of PNAND with Functionally Equivalent Network of Standard Cells

Although Fig. 4.2 shows a threshold function driving a flipflop, it is possible that no threshold function drives a flipflop in the given circuit. In that case, an arbitrary function driving a flipflop can be *decomposed* such that resultant representation consists of PNAND driven by additional logic. Such a decomposition is depicted in

Figure 4.4.

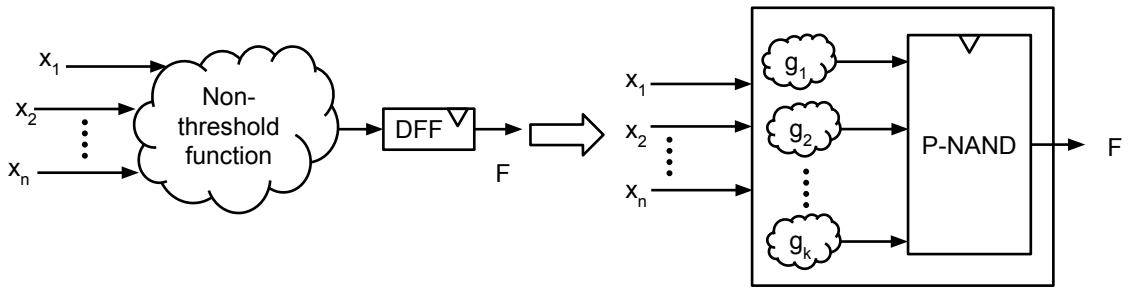


Figure 4.4: Decomposition of a Nonthreshold Function w.r.t a Threshold Function

Following sections describe two methods that can decompose a Boolean function into a set of functions (denoted as g functions) driving a threshold function (denoted as H) implementable by a PNAND.

4.2 BDD Based Decomposition using Cut Enumeration

In this section, we explore a BDD based method to produce such a decomposition. This method consists of enumeration of cuts present at the inputs of each flipflop and converting each cut (logic gates + flipflop) into an equivalent decomposed version and finally choosing the best decomposition that maximizes the absorbed logic. For the BDD based method, the combinational logic part of the circuit, whose cuts are enumerated, is specifically synthesized with 2-input NAND gates.

4.2.1 Enumeration of Cuts

The combinational circuit in a cone is represented as a directed acyclic graph (DAG) where the nodes are gates and directed edges are the interconnects. A *source* (*sink*) vertex is one that has no incoming (outgoing) directed arcs. A *cut* in a DAG $G = (N, A)$ is a minimal set of edges C such that every directed path from any of the source vertices to any of the sink vertices contains at least one member of C.

Given a DAG $G = (N, A)$, a *line dependency* graph (LDG) (Kagaris and Tragoudas, 1999a) is an undirected graph $L = (V, E)$ where $V = A$ i.e. each edge in the DAG G is a vertex in the LDG L, and two vertices in L are adjacent if and only if the corresponding edges in G lie on the same directed path. In (Kagaris and Tragoudas, 1999a) it is proved that a maximum independent set in an LDG is the same as a maximum cut in the DAG. This idea can be extended by observing that any maximal independent set in LDG is a cut in the DAG. Figure 4.5 shows the relation between cuts and maximal independent sets. Thus, enumerating cuts of size k in a DAG is the same as enumerating maximal independent sets (MIS) of size k in the corresponding LDG. Furthermore these type of cuts, called as *strong line-cuts* (or strong cuts in short), are not same as conventionally enumerated cuts. A detailed discussion of these cuts and related theory are provided in the appendix.

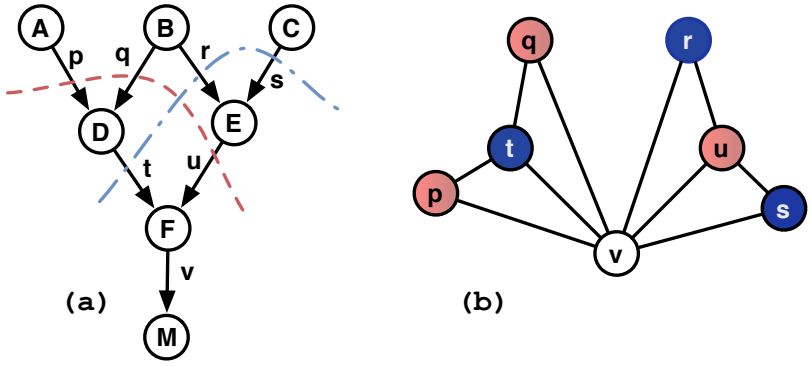


Figure 4.5: a) DAG with Strong Cuts Annotated. b) Corresponding Maximal Independent Sets in LDG

There exists a large body of literature on enumerating the maximal independent sets no larger than k. The method of cut enumeration used in the hybridization procedure is based on the heuristic presented in (Eppstein, 2001a). The novelty of the proposed method involves speeding up this heuristic by transforming the LDG by adding and removing certain vertices and edges without eliminating any maximal

independent set of size k.

A vertex u in an LDG can be removed if and only if the smallest MIS containing u is larger than k. Since an MIS is the same as a cut in the DAG, the smallest MIS corresponds to the minimum cut. Using network flow, the size of the minimum cut containing edge u can be determined. Similarly an edge (u, v) can be added to the LDG if and only if the smallest MIS containing both u and v is larger than k. The complexity of cut enumeration can be shown to be lower for the modified LDG. Since neither vertex removal nor edge addition eliminates a k sized MIS, it follows that all the k sized MISs of the original LDG are same as the k sized MISs in the modified LDG. Hence, no cuts are lost.

This idea is illustrated by the DAG shown in Figure 4.5 (a). Suppose that the cuts of size at most 2 are sought. In that case, the vertices p, q, r, s can be removed (from corresponding LDG in Figure 4.5 (b)), since the smallest MIS containing any one of them is larger than 2. The remaining LDG, consisting of the vertices $\{t, u, v\}$, contains two MISs i.e. $\{u, t\}$, $\{v\}$ which corresponds to cuts in DAG. If the cuts of size 3 or less are required then we can add edges (p, r),(q, r),(p, s),(q, s) so that resultant LDG does not have an MIS of size 4 i.e. $\{p, q, r, s\}$. The speedup obtained by the proposed method over (Eppstein, 2001a) is about an order of magnitude.

4.2.2 Threshold Decomposition

This section explains a threshold decomposition method that enables us to represent an arbitrary Boolean function using a threshold function in PNAND library. The decomposition method is an extension of the BDD based disjoint decomposition method first proposed by Lai et. al.(Lai *et al.*, 1993).

We briefly explain the idea from (Lai *et al.*, 1993) for better understanding of the proposed threshold decomposition technique.

Definition 1. *A Boolean function* $F(x_0, x_1, \cdots, x_{n-1})$ *is said to be* ***decomposable*** *under a* bound set *of variables* $\{x_0, x_1, \cdots, x_{i-1}\}$ *and a* free set *of variables* $\{x_i, \cdots, x_{n-1}\}$ *for* $0 < i < n$*, if there exists a function* H *such that* $F = H(g_0(x_0, \cdots, x_{i-1}), \cdots, g_{k-1}(x_0, \cdots, x_{i-1}), x_i, \cdots, x_{n-1})$ *for* $0 < k \leq i$.

The arguments g_i of the H-function are referred to as the g-functions or g-variables. The bound sets and the free sets are arbitrary choices and each different choice yields a different decomposition. Since the free set and the bound set are disjoint, this is called disjoint decomposition.

Suppose we are given a Boolean function F as in Figure 4.6(a), and a bound set $B = \{x_0, x_1, x_2\}$. A bound set in the BDD corresponds to a horizontal cut in the BDD. For example, as shown in Figure 4.6(a), the cut for the bound set B is set of nodes $S = \{p, q, r\}$. Note that a cut in the BDD is not related to the cuts in a logic cone mentioned earlier.

Step 1: For each of the nodes in the cut, a unique code is assigned. For N nodes, each code can be represented as a bit sequence of length $k = log_2 N$. The assignment of a code is strictly arbitrary and different assignments yield different decompositions. As will be explained shortly, the problem of threshold decomposition is the same as finding an assignment for which the resulting H function is a function in the PNAND library. In the example, node p is assigned code **10**, node q = **01** and node r = **00**.

Step 2: After encoding the nodes of the cut, the g-functions are constructed. There are always exactly as many g-functions as the length of the code (k) viz. $g_0, ..., g_{k-1}$. The BDD of g_i is computed using the BDD of F as follows. For each node f in the cut with encoding $(b_0, ...b_{k-1})$, replace the node f with a constant, b_i. In the above example, the length of code is 2. Hence there are two g-functions, g_0 and g_1. Figure 4.6(b) shows computation of g_0 where each node in the cut (p,q,r) is replaced with first bit of its code. Similarly, Figure 4.6 (c) shows computation of g_1 where

each node in the cut is replaced with second bit of its code.

Step 3: The final step is to define the H-function. Equations (4.1) and (4.2) below show construction of H.

$$T_i = [(b_0^i \bar{\oplus} g_0) \vee \cdots \vee (b_k^i \bar{\oplus} g_k)] \wedge f_i,\ 1 \leq i \leq N \tag{4.1}$$

$$H = T_1 \vee T_2 \vee \cdots \vee T_N \tag{4.2}$$

where $f_i \in S$ and $(b_0^i, b_1^i, \cdots, b_k^i)$ is its encoding. Fig. 4.6 (d) shows the BDD for H. By definition of H, the code of each node in the cut is exactly same as the input combination of g-variables traced from the root in the BDD of H. For example, the node p is encoded as 10 in original function F. This node is computed from H by setting $g_0, g_1 = 1, 0$. Therefore every minterm of g-variables corresponds to the encoding of a node in the original cut. Note that some minterms of g-variables correspond to an unassigned code such as **11**. These can be arbitrarily assigned to any node in the cut. For example, the unassigned code **11** is assigned to node p which resulted in reduction of BDD of H. The construction of H and g functions completes the decomposition of F.

The above disjoint decomposition procedure can be intuitively explained as follows. For a given assignment A of bound variables, the original function F evaluates to certain node in the cut, say f. If f is assigned a certain minterm $(g_0, \cdots, g_{k-1})$ of g-variables (code), then we make sure that for the same assignment A of bound variables, the g-functions evaluate to exactly that minterm (code). For example, given an assignment $(x_0, x_1, x_2) = (1, 1, 0)$ of bound variables, function F evaluates to node q. Since node q is reachable by g-variable assignment $(g_0, g_1) = (0, 1)$, we ensure that g_0 evaluates to 0 on $(x_0, x_1, x_2) = (1, 1, 0)$ and g_1 evaluates to 1 on the same.

4.2.3 Necessary Conditions

The procedure for threshold decomposition is an extension of the BDD based decomposition procedure where an H-function is sought that is a threshold function. An arbitrary encoding of nodes in the cut determines the H-function. It is then possible to determine whether it is a threshold function. Hence the general problem to be solved can be stated as follows: *Given a set of nodes in the cut of an arbitrary BDD, determine an encoding of these nodes for which the resultant H-function is threshold.* This problem can be shown to be NP-Hard. Hence we present a heuristic for threshold function decomposition based on the following necessary conditions, which can be easily established from the basic properties of threshold functions. The necessary conditions for H to be a threshold function are:

1. Every node in the cut must be a threshold function. This follows directly from the fact that every Shannon co-factor of a threshold function is also a threshold function.

2. There must exist a common weight vector W for every function (node) in the cut. Therefore every node N_i in the cut is a threshold function of the form $N_i = [W; T_i]$.

There are couple of approaches to design exact algorithms to ensure that these conditions are satisfied. Based on the number of nodes in the BDD of F and in the cut. it is possible to put an upper bound on the maximum weight of resultant threshold function. Using the maximum weight information, it is possible to derive a circuit-SAT based formulation. The single output circuit produced by this formulation is satisfiable if and only if a valid decomposition exists. In fact every solution of this problem, yields a different decomposition. However, solving the circuit-SAT was found to be impractical even for small instances of input BDD.

Alternatively, it is also possible to determine the required encoding by searching of the encoding space. The solution space can be narrowed down significantly by prohibiting certain encodings using the properties of Threshold logic.

Note that a decomposition algorithm is run on every horizontal cut (n of them) in the BDD of every cut/sub-circuit in the cone for each flipflop. Under these conditions, both of the exact algorithms were found to be too slow to be practical.

4.2.4 Threshold Decomposition Heuristic

The exact algorithms are slow and the solution generated by them may not be in PNAND library. Therefore in this work, we resort to a fast heuristic targeted specifically to the PNAND library. The proposed algorithm (algorithm 1) is given below.

Algorithm 1 starts by checking two necessary conditions (lines 3-9) described in section 4.2.3. At the end of step 9, a threshold function in the PNAND library which contains the common weight vector W_c for all functions in the cut is obtained. If one not found, failure is returned. The variables corresponding to weights in W/W_c are precisely the g-variables. The size of support set of H is known beforehand hence we look at only those functions in PNAND library having the desired size of support set. Since nodes in S must be co-factors of H w.r.t the g-variables, lines 10-15 check to ensure that the co-factors w.r.t all minterms of the g-variables belong to S and only S. The minterm associated with $f_i \in S$ is its encoding. Given an encoding, the g-functions can be computed using the BDD of F as described in section 4.2.2.

The condition in step 2 is implemented using the *isThreshold* algorithm from (Gowda *et al.*, 2011). The procedure *isThreshold* algorithm is not exact, and is pessimistic, i.e., it will not declare a non-threshold function as being a threshold function, but it may declare a threshold function as non-threshold function. However,

```
ThresholdDecompostion(F,S = {f_1, ··· , f_N});
input  : A Boolean function F and a cut denoted by set of nodes
         S = {f_1, ··· , f_N}
output: A set of decompositions D of F s.t. H-function is a threshold
         function in PNAND library
D = φ;
if any f_i ∈ S is not threshold then
    return φ;
end
Find a common weight vector W_c for all functions in S such that
f_1 = [W_c; T_1], ··· , f_N = [W_c; T_N];
if no valid W_c exists then
    return φ;
end
foreach function [W; T] in PNAND library s.t. W_c ⊂ W do
    foreach minterm m of variables in W/W_c do
        f = eval([W; T], m);
        if f ∈ S then
            Assign code m to function f;
        end
    end
    if all functions in S are assigned a code then
        H = [W; T];
        G = g-functions (using method in 4.2.2);
        D = D ⋃ (H, G);
    end
end
return D;
```

Algorithm 1: Threshold Decomposition Algorithm

for all the 338 functions in the PNAND library, *isThreshold* correctly identifies them as threshold. The computation of the common weight vector (W_c) is also an equally hard a problem as identifying threshold function itself. For exactness, it can be solved using ILP. Since ILP run-times are not practical, the implementation employs a heuristic to speed it up. The heuristic finds a common weight vector between every pair of nodes in S and checks if resultant common weight vector works for all nodes in S. The procedure to find a common weight vector between a pair of nodes called *tryEqualizeWeights* is described in (Gowda *et al.*, 2011).

Consider the special cut in the BDD of F for which $S = \{F\}$. This corresponds to the case where F itself is a threshold function. In this case, the g-functions do not exist, and the bound set is empty. Such cuts are most preferable since any logic due to g-functions is eliminated. Extending the idea further, we observe that it is preferable to have bound set as small as possible to minimize the g-functions. Therefore the cuts in the BDD are examined from top to bottom until a decomposition is found.

Note that different variable orderings of the BDD lead to different decompositions. In general, orderings that have smaller BDDs lead to better threshold decompositions and are therefore preferred.

An example of the threshold decomposition procedure is shown in the Figure 4.7. The input is a non-threshold function $ab + cd$ (Figure 4.7 (a)). Cut 1 is not valid since the top node (which represents $ab + cd$) is not a threshold function. Cut 2 is not valid because no common weight vector for the children can be found (lines 6-9). Cut 3 is valid since all necessary conditions are satisfied. The common weight vectors of nodes c and 1 are respectively $[w_c = 1, w_d = 1; T = 2]$ and $[w_c = 1, w_d = 1; T = 0]$. The common weight vector $[1, 1]$ is then searched in the PNAND library for a match. A candidate match containing common weight vector is $[2, 1, 1; 2]$. Since the co-factors of $[2, 1, 1; 2]$ w.r.t to g_0 having weight 2, are $[w_c = 1, w_d = 1; T = 0]$ and

$[w_c = 1, w_d = 1; T = 2]$, this function yields valid decomposition. Figure 4.7 (b) and (c) show the construction of g_0 and H respectively. The final decomposition is $F = g_0 + cd$ and $g_0 = ab$.

4.3 ILP Based Decomposition

This section describes an ILP based method of producing functional decomposition which is more effective and allows for more flexibility in decomposition such as don't care sets. The formal problem statement of this method is as follows. Let $F(x_1, \cdots, x_n)$ be a given Boolean function and let $L(g)$ denote some appropriate *cost* of a function g that is to be minimized. For instance, $L(g)$ could denote the number of literals in a minimum SOP form of g or an optimally factored form of g. Let $H(g_1, g_2, \cdots, g_k) = [w_1, \cdots, w_k; T]$ be a given threshold function. Then the problem is to determine functions $g_1(x_1, \cdots, x_n), g_2(x_1, \cdots, x_n), \cdots, g_k(x_1, \cdots, x_n)$ such that $F = H(g_1, g_2, \cdots, g_k)$ and $\sum_{i=1}^{k} L(g_i)$ is minimum.

The condition that the two circuits in Figure 4.4 be functionally equivalent leads to a necessary condition on the decomposition. Let M_1 and M_0 be minterms in the on-set and off-set of F, respectively. Then, for the circuit in Figure 4.4(b), $H(g_1(M_1), \cdots, g_k(M_1)) = 1$, and $H(g_1(M_0), \cdots, g_k(M_0)) = 0$. Therefore, $\sum_{i=1}^{k} w_i g_i(M_1) \geq T$ and $\sum_{i=1}^{k} w_i g_i(M_0) < T$ must be true. It is important to note that as long as these two conditions are satisfied, the g functions can contain any subset of minterms of variables of F, including those in the on-set and off-set of F.

Figure 4.8 shows an example of one possible decomposition of $F(a, b, c, d) = (\bar{a} + \bar{b})(c+d)+cd$. The threshold function $H(g_1, g_2, g_3) = g_1g_2+g_1g_3+g_2g_3$. This is a 2-out-of-3 majority function that can be implemented by a PNAND-3. One decomposition of F results in $g_1(a, b, c, d) = \bar{a} + \bar{b}$, $g_2(a, b, c, d) = c$ and $g_3(a, b, c, d) = d$. Notice that the g functions cover both on-set and off-set minterms of F. The decomposition

also demonstrates the substantial area and delay improvements that are possible with decomposition. Also, if the g functions turn out to be single literals, then F itself is a threshold function.

Different choices for g functions are compared based on their cost, which is $\sum_{i=1}^{k} L(g_i)$. The time required to compute this cost function can be very high. For instance, it can be based on the minimum SOP or the optimally factored form of each of the g functions. If it is a one time effort, it may be acceptable. However, in the present application, it is part of an inner loop of the hybridization procedure, and hence is not practical for all but very simple circuits. One way to simplify the evaluation of the cost of the g functions is to restrict them to be unate. This is because the minimum SOP of a unate function is unique and is easily obtained by ensuring that the SOP expression is minimal with respect to single cube containment (Hachtel and Somenzi, 1996). Then the total number of literals in the minimum SOP is an accurate reflection of the number of two input AND/OR gates required to realize the function. In the statement of the decomposition problem given above, the functions F and H are given. The function F is extracted from the netlist and H is one of the threshold functions in the cell library. One way to ensure that the g functions are unate is to ensure that F is unate. In the following, an efficient solution to the problem of computing the g functions is presented, given a unate function F and a threshold function H. Without loss of generality, F and g_i can be assumed to positive unate functions.

4.3.1 0-1 ILP formulation

The decomposition algorithm must compute the g functions which are inputs to the H function. Each g_i function is a subset of all the minterms M_j of the variables from the support set of F i.e. $x_1, x_2, \cdots, x_n$. One way to explore all possible g_i

functions is by associating each minterm M_j with each function g_i. This is done by introducing a Boolean variable $m_{i,j}$, which is 1 if M_j belongs to the function g_i, and 0 otherwise. However as discussed earlier, computing cost of g_i functions using a linear expression of $m_{i,j}$ variables is not possible. Therefore instead of associating a minterm M_j with a g_i function, we associate a unate cube C_j with a g_i function. The total cost of the g_i function is now the total number of literals in all the cubes of g_i, which is now a linear expression.

Let C denote the set of positive cubes over the set of variables $\{x_1, x_2, \cdots, x_n\}$. That is

$$C = \{\phi, x_1, x_2, \cdots, x_n, x_1x_2, \cdots, x_{n-1}x_n, \cdots, x_1x_2\cdots x_n\}.$$

Let the Boolean variable $c_{i,p} = 1$ if and only if the cube $C_p \in C$ is added to the g-function g_i. Since each cube contains one or minterms, there is a relationship between the Boolean flags $c_{i,p}$ and $m_{i,j}$. This relationship is expressed using following constraints.

$$c_{i,p} \leq m_{i,j}, \quad 1 \leq i < k, \quad C_p \in C, M_j \subseteq C_p, \tag{4.3}$$

The constraint (4.3) states that if any cube C_p is added to the function g_i, then all the minterms M_j contained in C_p must also added to g_i. We now express the functional equivalence constraints briefly described in Section 4.3.

$$\sum_{i=1}^{k} w_i.m_{i,j} \geq T, \quad \forall M_j \subseteq F \tag{4.4}$$

$$\sum_{i=1}^{k} w_i.m_{i,j} < T, \quad \forall M_j \subseteq \overline{F} \tag{4.5}$$

Constraints (4.4) and (4.5) ensure that any association of cubes (and therefore minterms) to the g_i functions must ensure that resultant decomposition computes the original

function F.

Finally, we describe the objective function for this formulation. As stated earlier, whenever a cube C_p is associated with a g_i function, the cost of g_i increases by number of literals in the cube C_p. Assume $Cost(C_p)$ denotes the pre-defined cost of cube C_p. The cost of each g_i function, denoted by integer variable C_{g_i} is denoted as follows.

$$C_{g_i} = \sum_{C_p \in C} Cost(C_p).c_{i,p} \tag{4.6}$$

The objective function is the sum of costs of all the g_i functions, i.e.,

$$\min \quad \sum_{i=1}^{k} C_{g_i} \tag{4.7}$$

Equations (4.3), (4.4), (4.5), (4.6) and (4.7) denote a complete 0-1 ILP formulation of the decomposition problem.

As an example, consider a unate function $F = ac + ad + bc + bd + cd$ and $H(g_1, g_2, g_3) = [1, 1, 1; 2] = g_1.g_2 + g_2.g_3 + g_1.g_3$. The above formulation seeks a distribution of the unate cubes to the three functions g_1, g_2 and g_3. The ILP solution is $g_1 = a + b$ (i.e. g_1 has two cubes, a and b), $g_2 = c, g_3 = d$. Indeed we see that $F = g_1.g_2 + g_2.g_3 + g_1.g_3 = (a + b).(c) + (a + b).(d) + (c).(d)$.

The proposed 0-1 ILP formulation automatically allows for don't cares of F. According to the constraints (4.4) and (4.5) only the minterms that are contained F and $\overline{F}$ are considered for equivalence check. If a minterm is not present in F or $\overline{F}$, then the ILP solver minimizes the g functions by appropriately distributing the don't care minterms into F and $\overline{F}$. This can result in simultaneous reduction of g functions and/or the reduction of the H function. For example, assume that the don't care set for the function on the left hand side is found to be $DC = \overline{a} \vee \overline{b}$ in Fig. 4.8. The decomposition would result in the elimination of the NAND gate and reduces the H function from majority to a two input OR gate.

4.3.2 Speeding up the 0-1 ILP

The 0-1 ILP formulation above has a large number of variables along with loose constraints. We can reduce the number of variables and better constrain the solution space with following modifications.

Cube Containment

When a cube C_p is added to the function g_i, none of the cubes contained in C_p need be added to g_i. Therefore additional constraints on $c_{i,p}$ variables are as follows.

$$c_{i,p} + c_{i,q} \leq 1, \quad 1 \leq i \leq k,\ C_q \in C,\ C_p \in C,\ C_q \subset C_p \tag{4.8}$$

Symmetries

If the H function is symmetric with respect to a group of g_is, then there will exist a large number of equivalent optimal solutions. To eliminate some of the symmetric solutions, we enforce the constraint on costs of each of the g_i functions. For each symmetry group of g_i variables denoted as $(g_u, g_{u+1}, \cdots, g_v)$, the following set of constraints are added.

$$C_{g_j} \leq C_{g_{j+1}}, \quad u \leq j < v \tag{4.9}$$

Reduced cube-set

A cube c contains (Boolean containment and NOT set containment) a cube d if all the literals in cube c are also present in cube d. A given positive cube C_p will be part of the optimal set of g functions if and only if it contains at least one cube of original function F. As an example, consider function $F = ab + bcd$. The cube-set $C = \{\phi, a, b, c, d, ab, bc, bd, cd, bcd\}$ suffices to yield optimal decomposition of F wrt any H function. The remaining cubes such as cube ac can never be part of an optimal

decomposition of F with respect to any threshold function. This significantly reduces the total number of cubes $C_p \in C$ and thereby, the number of $c_{i,p}$ variables. Here is the proof why this cube-set reduction works.

Theorem 1. *A given positive cube c will be part of the optimal set of g functions if and only if it contains at least one cube of original function F.*

Proof. We prove this by contradiction. Without loss of generality assume that F is a positive unate function of all its variables. Also assume that for some threshold function H and an optimal decomposition denoted by g_i functions, there exists an irredundant cube $c \in g_1$ which does not contain any cube of F. Therefore $g_1 = c + Q$ where Q denotes remaining cubes in g_1. An expression for F can be written as $F = H = g_1.R + S$ where R and S are expressions containing all g_i variables except g_1. If we replace g_1 by its definition then we can represent original function as $F = (c + Q).R + S = c.R + Q.R + S$. Note that cube c contains each of the cubes in $c.R$. However c does not contain any cube in F therefore all the cubes in $c.R$ must be redundant and wholly contained in $Q.R + S$. As a result all the cubes in the expression $c.R$ can be eliminated without changing the definition of F. We can also eliminate these cubes by simply eliminating cube c from g_1 altogether. However if we remove cube c from g_1 then we have a g_1 function with smaller cost. This contradicts the fact that original decomposition denoted by g_i variables is optimal. □

4.3.3 Unate Function Enumeration

We note that the ILP based procedure requires a unate function driving a flipflop. We describe a procedure that enumerates only unate functions similar to how strong cut enumeration enumerates function required for BDD based decomposition. The procedure to enumerate the unate functions driving a flipflop is outlined in Algorithm

2. This procedure is indeed a form of cut enumeration. However an important distinction between conventional cut enumeration and this procedure is that it enumerates only cuts that are unate functions.

The input to the procedure in Algorithm 2 is an abstract gate level netlist constructed from the synthesized netlist generated by the synthesis tool. The synthesized netlist from the synthesis tool consists of cells such MUX, OR, NAND etc. However not each of these functions is unate. Therefore an extra step is applied that converts each gate into small sub-network consisting solely of unate functions. As an example, if a gate is a MUX denoted by $f = a.\bar{s} + s.b$, then this cell is converted into two gates represented as $f = g + s.b$ and $g = a.\bar{s}$. Note that f is a unate function now. This way we can ensure that every flipflop is driven by a unate gate.

The procedure in Algorithm 2 works by repeated absorption of fanin gates in a breadth first manner. If a function is not unate after absorption of a fanin gate, the search beyond that gate is terminated.

4.3.4 Hybridization Procedure

This section outlines the procedure that uses the decomposition methods described above to convert a sequential network into a hybrid network. The outer hybridization procedure takes as an input, a cell level netlist synthesized using RTL compiler and the list of flip-flops in this network. It returns the minimum area hybrid netlist. The outer procedure is simple. For each flipflop, it determines a decomposition (using either of above two methods) consisting of a PNAND cell and a set of g_i functions that drive its inputs. Among all the decompositions, the one that provides maximum reduction in logic is chosen and that flipflop along with unate function F feeding it, is replaced with this decomposition. Each time a flip-flop is replaced, the resultant netlist is stored and re-synthesized.

```
EnumUnateFunctions(F, N);
input  : A fanin logic cone of a flipflop F, a fixed integer N indicating the
         number of unate functions to enumerate
output: A list of L of N unate functions each of which driving the flipflop F
L = φ;
Q = queue();
// g denotes the logic gate driving flipflop F;
Q.append(g);
while (Q.size() > 0 && N > 0) do
    g = Q.pop();
    S =fanin gates of g;
    foreach h ∈ S do
        u =merge(g, h);
        if u is not unate then
            continue;
        end
        Q.append(u);
        L = L ⋃ {u};
        N = N − 1;
    end
end
```

Algorithm 2: Procedure to enumerate unate functions in a logic cone driving a flipflop

The flip-flops are replaced by PNAND one-by-one in the order of their criticality because PNANDs can most benefit the critical paths by providing slacks. The logic cones on critical paths are larger and reduce substantially under timing slacks and low input capacitance supplied by PNANDs. Another reason why PNANDs must reside on the critical paths is because they have higher hold time than the conventional D-flip-flops. Since the critical paths are longer, the effort of the synthesis tools to meet the hold time is minimal. In fact, the PNANDs on non critical paths can force the synthesis tools to insert buffers to meet hold time worsening the circuit area and power. Among all the intermediate re-synthesized netlists obtained from above procedure, the minimum area netlist is chosen as the final hybrid netlist.

In practice, the optimal decomposition can be pre-computed for all the flip-flops in parallel, say on a cluster of computers, and utilized in the order of criticality. This significantly speeds up the computation. Therefore the whole procedure can be run in near constant time provided enough resources are available to process large circuits.

4.4 Experimental Results

In this section, we present the results of experiments carried out on several large function blocks. For comparison, two complete implementations of each circuit were carried out, one using conventional logic design and the other being a hybrid design. All the final results are based on simulations performed on extracted netlists of the circuits after placement and routing were successfully completed, and the designs met the timing requirements and passed the equivalence check.

4.4.1 Methodology

The methodology followed is summarized below.

1. A library of four PNAND cells (PNAND-3, PNAND-5, PNAND-7 and PNAND-

9) was created and the layout of each was optimized to achieve 0 errors in 100K Monte Carlo simulations for the input case that exhibits the maximum contention, accounting for both global variations and local mismatch.

2. Each of the four cells was then characterized for different worst-case delay input combinations. This is to allow a more accurate calculation of function-specific delays during the synthesis phase. A new approach to characterize the setup and hold times of PNANDs was developed and verified. The conventional approach to characterize the setup and hold times of DFFs is not valid, as PNANDs are multi-input flipflops.

3. For a given circuit C, minimum period synthesis was performed using Cadence RT compiler, resulting in a netlist C_{minP}. The library cells used during synthesis were characterized at the PVT corner (SS, 105C, 1.1V).

4. The hybridization of netlist C_{minP} was performed to produce a hybrid netlist $H(C_{minP})$. We specifically show results for ILP based decomposition method because not only it can be shown to be theoretically superior, but indeed practical results of ILP based method are better.

5. Logic synthesis of the netlist $H(C_{minP})$ was carried out using Cadence RT compiler, with the target clock period of $minP$, to produce a netlist $H^*(C_{minP})$. During this stage, the PNAND cells were left untouched.

6. Both C_{minP} and $H^*(C_{minP})$ were successfully placed and routed using Cadence Encounter, with the target clock period of $minP$, ensuring that C_{minP} and $H^*(C_{minP})$ could be operated at the same clock frequency. The multi-corner analysis feature of the tool was used in order to best meet setup and hold timing constraints. The setup corner was (SS, 105C, 1.1V), whereas the hold corner

(FF, -40C, 1.3V). The tool was also provided with another nominal operation corner (at which the circuit will actually operate post-fabrication) which is TT, 25C, 1.2V.

7. The layout extracted netlists were then used to simulate the circuits for dynamic and leakage power. The placed and routed netlists (both conventional as well as hybrid) were simulated in verilog for a large number of inputs patterns with a fixed switching activity of each input. The resulting switching activities in the circuit were saved in a VCD (value change dump) file. This file is then given to the Synopsys power measurement tool, *PrimeTime*, along with circuit parasitics and libraries. The tool produces near accurate values of the average dynamic and leakage power for both CMOS and hybrid circuits. In order to ensure that the characterized power data for hybrid circuits (specifically P-NAND cells) was accurate, the power of PNAND cells was estimated using both HSPICE and PrimeTime and found them to be within 1%.

4.4.2 Circuits

Experiments were conducted on five large commonly used circuit blocks. These are (1) a 32-bit, 2-stage Wallace Tree multiplier, (2) a 28-bit, 4-tap digital FIR Filter, (3) a 3-stage, 64-bit floating point multiplier, (4) a 32-bit MIPS core without a floating point unit, and (5) a 10-stage standard encryption circuit implementing the AES algorithm, with 128-bit key and 128-bit plain text input.

4.4.3 Results

Table 4.2 shows the comparison of the area and total wire length of the conventional logic circuits and hybrid circuits. The results demonstrate a substantial

reduction in both the total area and wire-length for the multipliers and the filter. In general, datapath circuits greatly benefit from hybridization in terms of area and wire length. The other circuits, namely the MIPS core and the AES circuit, while still smaller (without a performance loss), show less improvement.

	Area (mm^2)			Total wire length (mm)		
Circuit	Conv.	Hybrid	% Δ	Conv.	Hybrid	% Δ
Multiplier	0.043	0.032	25	135	95	29
Filter	0.124	0.091	27	350	245	30
FPU	0.094	0.082	13	398	323	19
MIPS	0.076	0.067	11	321	293	9
AES	0.7	0.6	15	3177	3080	3

Table 4.2: Area and Wire-length Reduction

Table 4.3 shows the comparison of dynamic and leakage power of the circuits at nominal operation corner (TT, 25C, 1.2V) and at 30% primary input switching activity. The results indicate a very substantial reduction in both dynamic as well as leakage power of the hybrid circuits compared to the conventional . Note again, this is without a performance loss. As stated earlier, the power reduction is due to the absorbed logic as well as the logic that reduces under timing slacks. The leakage of the circuits is a small percentage of dynamic power because the particular 65nm LP (low power) process, as opposed to the GP (general purpose) process. The ratio of leakage to dynamic power is substantial in a GP process, and therefore we can expect that for GP processes the improvement in leakage will further add to the improvement in total power.

We also explored how the reductions in dynamic power would vary as a function of the operational frequency and the input switching activities. Figures 4.9 and 4.10

	Dynamic Power (mW)			Leakage Power (μW)		
Circuit	Conv.	Hybrid	% Δ	Conv.	Hybrid	% Δ
Multiplier	35.9	22.7	37	4.22	2.13	49
Filter	67.1	43	36	13.68	6.71	51
FPU	46.8	35.9	23	7.68	5.59	27
MIPS	31.6	22.6	29	5.61	3.88	31
AES	205	170	17	55.2	45.5	18

Table 4.3: Dynamic and Leakage Power Reduction @ TT, 25°C, 1.2V and 30% Switching Activity

Table 4.4: Variation in the Dynamic Power

	Standard Deviation of Dynamic Power (mW)		
Circuit	Conventional	Hybrid	% Δ
Multiplier	2.9289	1.7248	41
Filter	8.6197	4.8469	44
FPU	4.5985	3.2108	30
MIPS	6.9563	5.5251	35
AES	3.079	2.0606	33

shows the results for the Wallace Tree multiplier. Similar results were obtained for all the other circuits. The improvements in dynamic power of the hybrid multiplier was demonstrated across all frequencies. Similarly, the advantages of the hybrid multiplier was consistent across a wide range of input switching activities.

Another important characteristic of hybrid circuits is the reduced standard deviation of dynamic power across clock cycles. Table 4.4 shows the relative reduction in the power variation for various circuits.

Reduced power variation in power is useful in two ways. As the variation in the

current demanded by the chip reduces, the load fluctuations on the power grid is reduced. This may help free power grid and/or decap cell area.

The second advantage of reduced variations is increased resistance to the differential power profile attack on the cryptographic processors such as AES. The AES circuit can reveal bits of the encryption key based on temporal power profile as the chip encrypts/decrypts a particular plain-text. A reduction in variation makes it harder to reveal bits of the encryption key. Figure 4.11 shows the instantaneous power (per cycle) over a number of cycles and corresponding standard deviation. The two plots have identical scales on x and y axes (not shown due to size restriction). The horizontal dark line in the middle of each curve represents mean dynamic power. The mean powers are aligned for comparison of the power variation. The left plot shows power variations for conventional circuits whereas the right one shows the same for hybrid circuits. It can be seen that the standard deviation of the power is considerably less for hybrid AES.

4.4.4 Advantages due to KVFF

Recall that KVFF is a single input PNAND architecture introduced in Section 3.6 which is basically a new type of D-flipflop. Note that even through KVFF does not itself absorb any logic, it still has smaller setup time than conventional master-slave D-flipflop. Therefore hybridization (aka resynthesis) with KVFFs and conventional CMOS library cells also provides certain advantages due to reduction of logic under timing slack. One might be interested in these results is because KVFFs are as robust as conventional D-flipflops. Therefore all the advantages gained by using KVFFs come without degrading the circuit robustness in any way. KVFF can simply be added to the conventional library of cells to obtain these advantages. Table 4.5 shows the improvements in dynamic power due to KVFF alone.

	Dynamic Power (mW)			Leakage Power (μW)		
Circuit	Conv.	KVFF	% Δ	Conv.	KVFF	% Δ
Multiplier	35.9	31.5	12	4.22	3.91	7.3
Filter	67.1	56.3	16	13.68	10.54	23
FPU	46.8	39	16	7.68	6.61	14
MIPS	31.6	31.3	1	5.61	5.48	2.3
AES	205	191.4	6.6	55.2	53.25	3.5

Table 4.5: Dynamic and Leakage power reduction @ TT, 25°C, 1.2V and 30% Switching Activity due to KVFF

Comparing Tables 4.3 and 4.5, we can see that improvements due to KVFF alone are typically less than what hybridization can provide. This is due to the lack of absorption of logic. An additional advantage of KVFFs is that they are smaller in size compared to PNANDs and therefore incur less burden on routing and cell area during automated place & route.

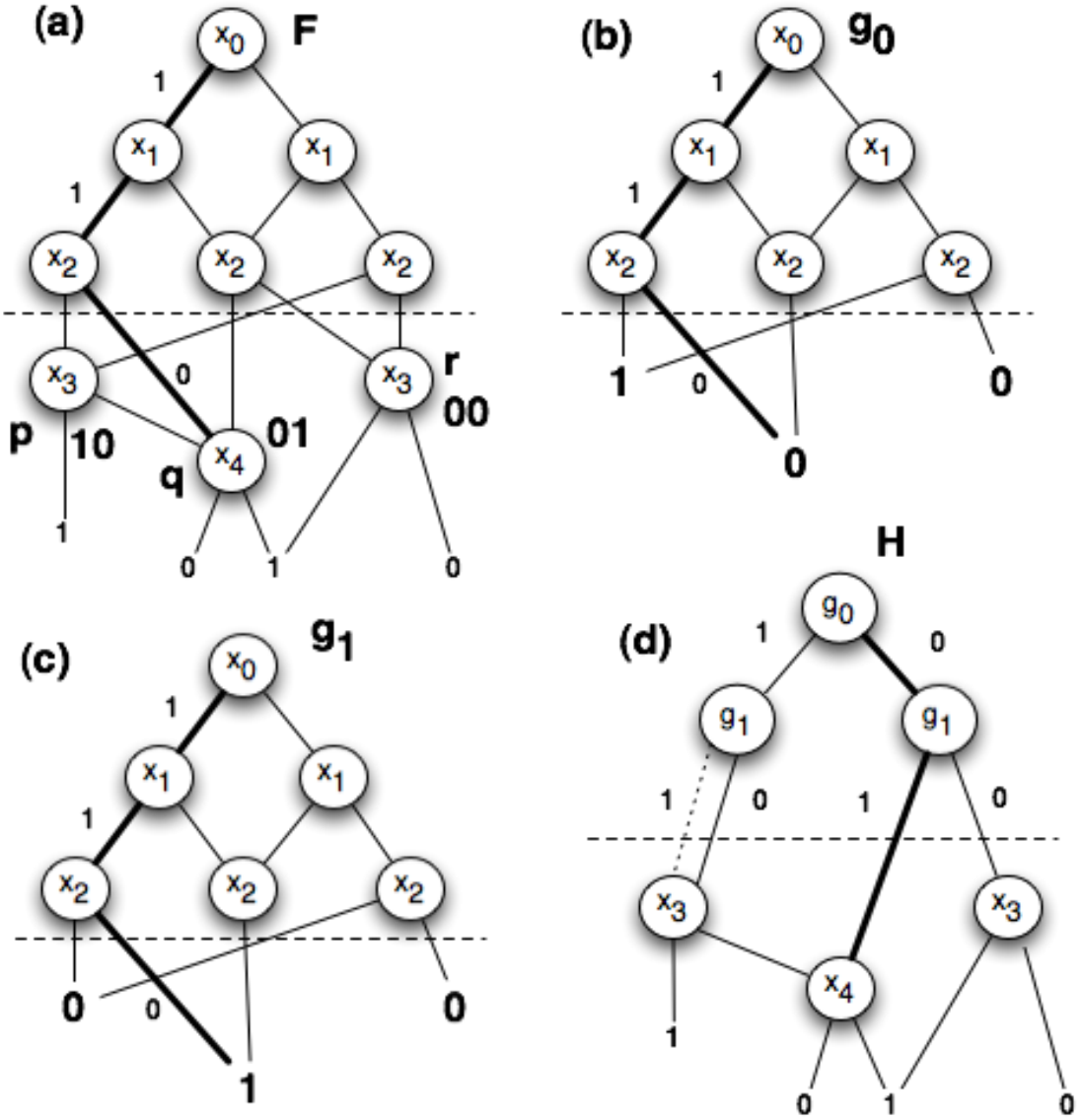


Figure 4.6: (a) Function F to be Decomposed. (b),(c) g-functions (d) H-function

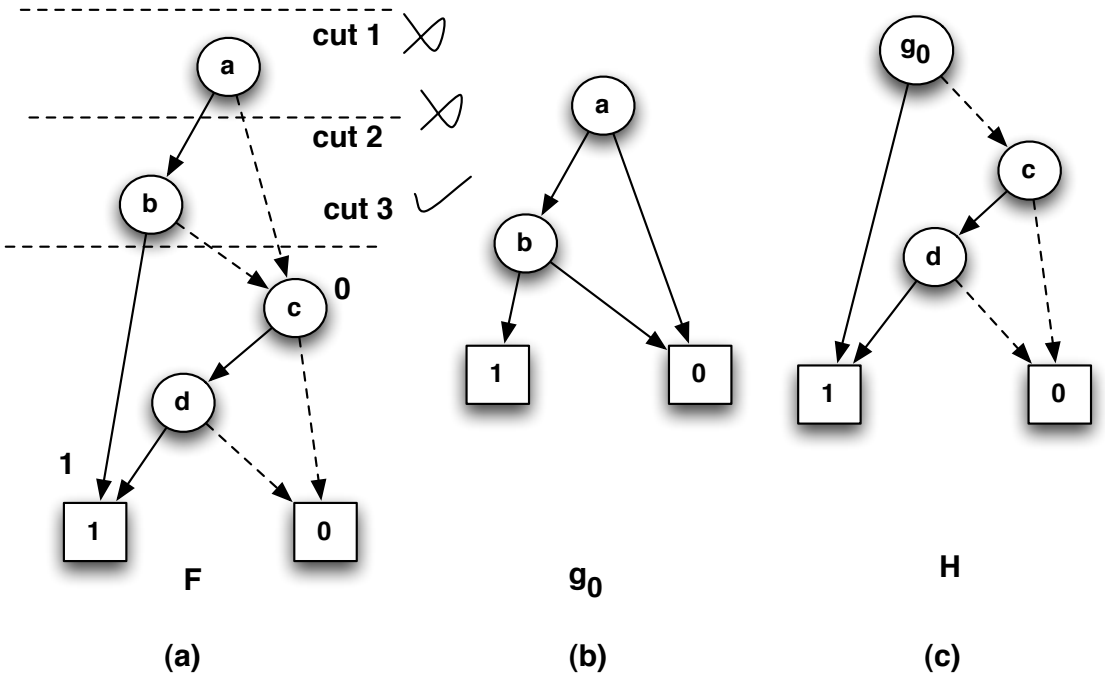


Figure 4.7: An Example of Threshold Decomposition (a) Non-threshold Input Function F and Cut (b) Computation of g_0 (c) H-function

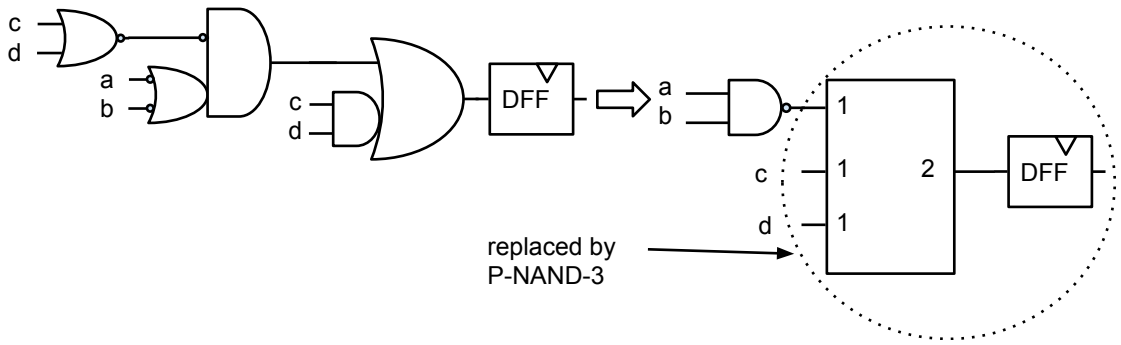


Figure 4.8: A Motivational Example where a Non-threshold Function is Decomposed into a PNAND-3 Driven by a Single NAND Gate

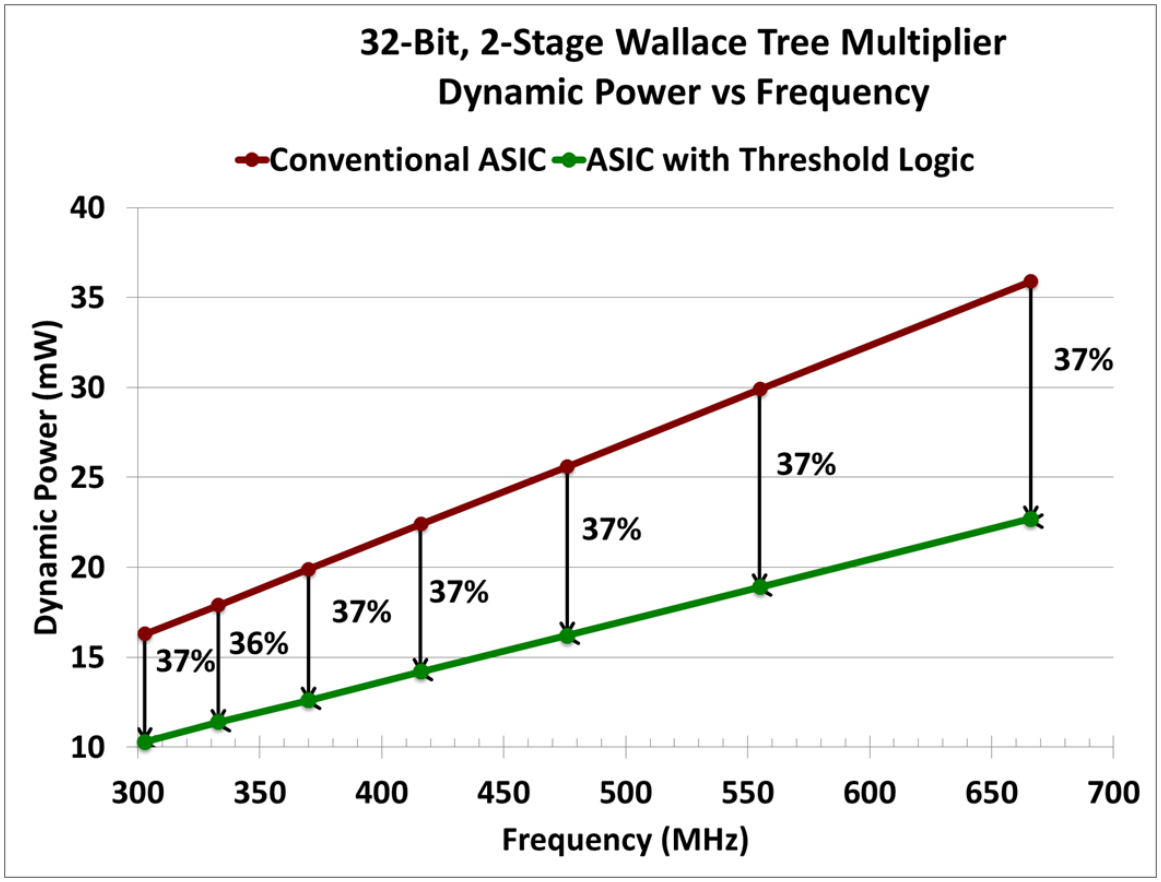


Figure 4.9: Advantages of Hybrid Circuits are Maintained Irrespective of the Operating Frequency

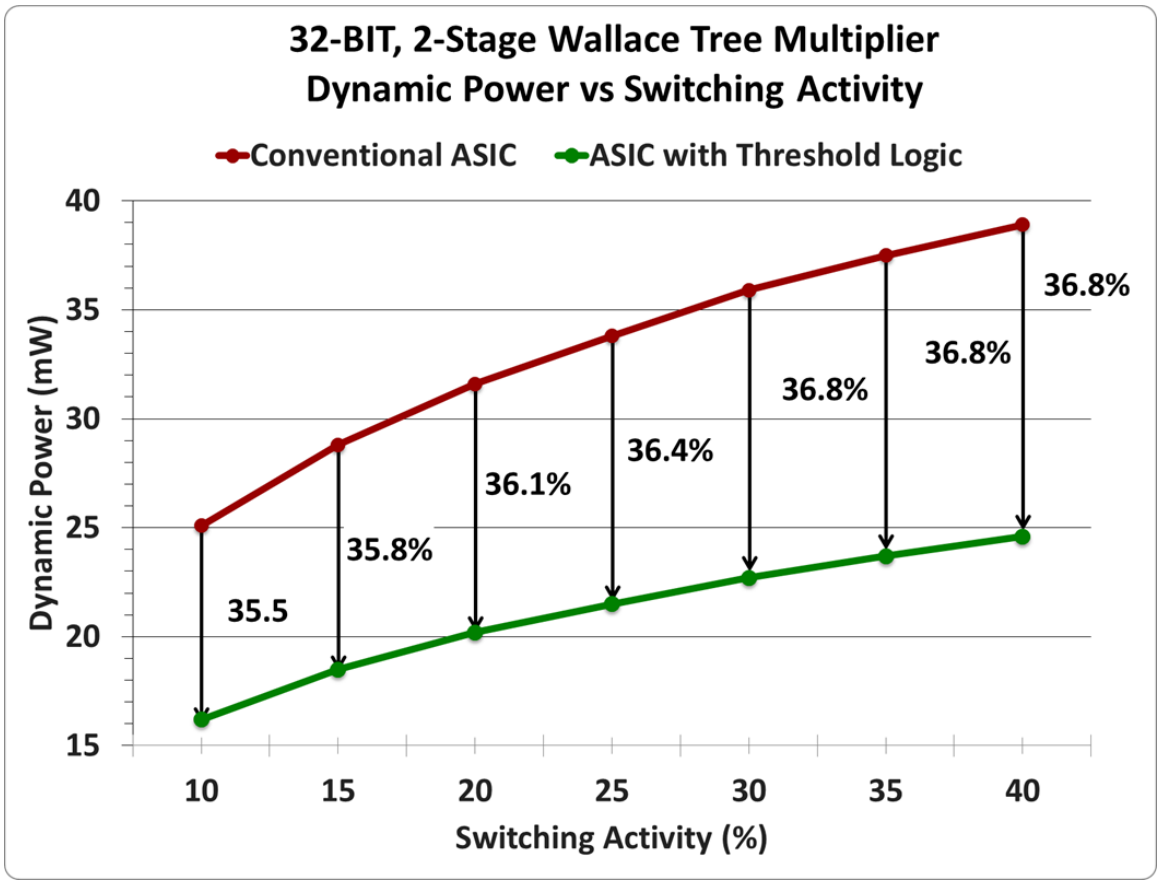


Figure 4.10: Hybrid Circuits are More Energy-efficient at Higher Switching Activity

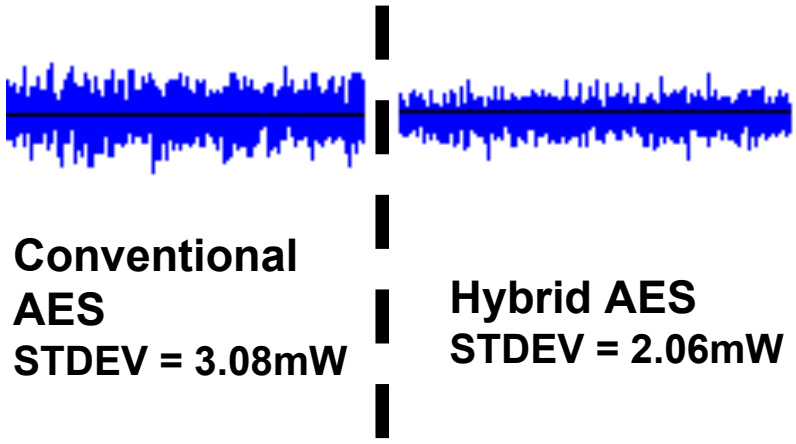


Figure 4.11: Variations in Dynamic Power of AES Circuit

Chapter 5

FIELD PROGRAMMABLE THRESHOLD GATE ARRAY

5.1 Overview

In this chapter, we present the architecture of a novel programmable logic array, referred to as Field Programmable Threshold-Logic Array (FPTLA), in which the basic cells are PNANDs. A PNAND can be programmed to implement different threshold logic functions by routing appropriate signals to their inputs. This reduces the number of SRAMs inside the logic blocks by about 60% compared to conventional CLBs, without adding any significant overhead in the routing infrastructure. Since a PNAND is essentially a multi-input, edge-triggered flipflop that computes a threshold function, a network of PNANDs forms a *nano-pipelined* circuit. The advantages of such a network are demonstrated on a set of deeply pipelined datapath circuits implemented on FPTLAs and conventional FPGAs using the well established FPGA design framework VTR (Verilog To Routing) and VPR (Versatile Place and Route) (Rose *et al.*, 2012). The results indicate that an FPTLA can achieve up to 2X improvement in delay for nearly the same energy and logic area compared to the conventional LUT based FPGA. Although differential mode circuits can potentially be more sensitive to process variations, FPTLAs can be made robust to such variations without sacrificing their improved energy efficiency and performance over FPGAs. We also present a novel method to synthesize a network of threshold logic gates (TLGs) that can be mapped onto FPTLA.

Below we summarize some of the key features of the proposed design, methods and the methodology.

1. The FPTLA can be programmed to realize any one of a number of threshold functions by assigning the appropriate signals to the inputs of PNAND logic blocks (referred to as DTGB which stands for differential threshold gate block). This reduces the number of SRAMs required by the logic blocks by about 60%.

2. Since a PNAND cell can implement a variety of threshold functions, characterizing its behavior at different temperature values would require performing MC simulations of each function at each temperature value. We show that this is not necessary. Specifically, we demonstrate a property that we refer to as *functional monotonicity with temperature*. If F_{T_i} denotes the set of threshold functions that can be reliably computed by a PNAND at temperature T_i, then the $F_{T_i} \subseteq F_{T_j}$, for $T_j > T_i$. This means that if a PNAND cell is programmed to realize a function f, and it is verified that it indeed computes f at temperature T_1, then it is guaranteed to compute f at any temperature T_2, for $T_2 \geq T_1$.

3. As we have seen, PNANDs are sensitive to mismatch due to process variations. However, mismatch in a PNAND need not make the logic block unusable. Rather, mismatch causes the *set of* threshold functions realizable by the PNAND to be altered. This leads to two different scenarios in how the array can be utilized.

   (a) A conservative approach is to simply ignore the cells whose functionality is altered by process variations. To ensure feasibility of the mapping, if a TLG network to be mapped has N TLGs, then an array of PNANDs larger than N would be required. We show that for a given probability of a cell's functionality being altered, the number of extra cells required to achieve a very high circuit yield is very small.

   (b) A key characteristic of PNAND cells is that mismatch can only result in

another set of *threshold* functions. That is, mismatch will not change a threshold function into a non-threshold function – a characteristic that is demonstrated here. Hence, instead of simply ignoring the altered cells, the synthesis procedure can take in account the new set of all functions available on the array, and generate a TLG network accordingly. Note that the synthesis procedure that accounts for this additional flexibility is beyond the scope of this work.

4. We present experimental results by comparing FPTLA and conventional FPGA implementations of several nanopipelined datapath circuits. Nanopipelined circuits are path balanced in which each gate computes a function on a clock edge. The throughput and speed of such circuits is very high due to such deep level of pipelining. The power and delays of these circuits are obtained using the well established VPR tool that accurately models the **routing resources and logic blocks**.

5.2 PNAND as a Majority Gate

There are many ways to map signals to the inputs of a PNAND-K, while satisfying the constraint that the LIN and RIN never have the same number of active devices. The different signal assignments will result in different delay, power and robustness to process variations. As our target implementation is an FPTLA, having 2K inputs for every cell would result in excessive wiring congestion. For this reason we have chosen to wire the LIN and RIN internally, so that the primary inputs drive the gates of the RIN and their complements drive the gates of the LIN. Figure 5.1 shows a PNAND-5 wired internally in such a manner.

A m out of n majority function is 1 if at least m of the inputs are 1. A PNAND-K wired in the manner described realizes a $(K+1)/2$ out of K majority function. Whenever the LIN has n devices ON, the RIN has $K-n$ devices ON. To ensure

they are not equal, K must be odd. In that case, whenever at least $(K+1)/2$ inputs are 1, the LIN will have more ON transistors then the RIN, resulting in the output Q of PNAND evaluating to 1. For example, the PNAND-5 cell shown in Figure 5.1 can realize 3 out of 5 majority when all the inputs are independent signals. In the notation of a threshold function it is $f(a, b, c, d, e) = [1, 1, 1, 1, 1; 3]$.

Fig. 5.1 shows how to connect 5 inputs a, b, c, d, e to reduce a PNAND-5 to a 3 out of 5 majority i.e.

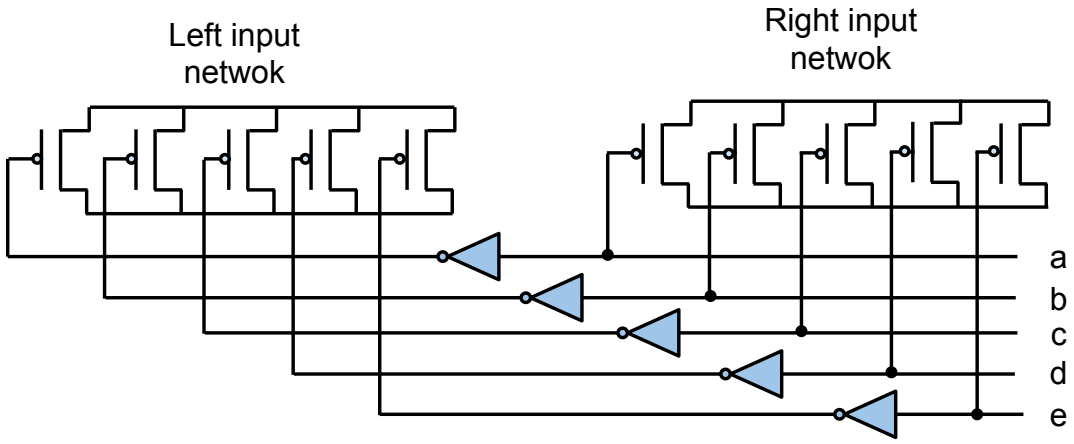


Figure 5.1: Realizing a 3 out of 5 Majority using PNAND-5

Given a PNAND-K implementing its majority function, the only input configurations m/n that will occur are where $m + n = K$. For example, a PNAND-7 implementing a 4 out of 7 majority has only 8 cases that occur at its inputs – (4/3, 3/4), (5/2, 2/5), (6/1, 1/6) and (7/0, 0/7). The delay of a PNAND has two components: the *propagation delay* through the LIN or RIN, which is the time required to charge node N5 or N6, and the sense-amp delay to correctly distinguish the difference between two rising voltage waveforms. For an input configuration m/n, the propagation delay is, to a first order, inversely proportional to the $\max(m, n)$. The maximum sense-amp delay occurs for all input configurations for which $|m - n| = 1$,

and among those, for which m or n is maximum. Thus for a PNAND configured as a majority gate, the same configuration exhibits the maximum propagation delay and sense amp delay, namely $m = (K+1)/2$ and $n = (K-1)/2$, or vice-versa.

Any threshold function $f = [w_1, w_2, \cdots w_n|T]$ can be implemented using a T out of W majority function, where $W = \sum w_i$ (Muroga, 1971). For example, consider a threshold function $f(a,b,c) = a \vee bc = [2,1,1;2]$, where $w_a = 2$, $w_b = w_c = 1$ and $T = 2$. Here $T = 2$ and $W = 4$. Given a 2 out of 4 majority function $g(p,q,r,s) = [1,1,1,1;2]$ f can be realized by a PNAND by simply connecting signal a to the inputs p and q, signal b to r and signal c to the input s of the function g. However note that the 2 out of 4 majority is realizable using a 3 out of 5 majority by simply setting one of its input signal to 1. Therefore any majority function has a fixed subset of threshold functions it can implement under all possible assignment of signals (including 0 and 1) to its inputs.

Consider implementing a 4-input AND gate f(a,b,c,d) using the 4 out of 7 majority $g(p,q,r,s,t,u,v)$. The assignment $p = a$, $q = b$, $r = c$, $s = d$, $t = u = v = 0$ realizes this function. Only when all of a, b, c, and d are 1, the threshold of 4 out of 7 majority is reached and output is 1 realizing a 4 input AND gate. Table 5.1 lists all the functions that are implementable by a PNAND-7 cell configured as a 4 out of 7 majority function, along with the required signal assignment. Incidentally, this is indeed the set of **all** functions that PNAND-7 can implement in general. Therefore we can see that in terms of ability to implement functions, a PNAND-K is as powerful as a $(K+1)/2$ out of K majority function.

The support set of each of the functions is assumed to be $a, b, c, \cdots$. Note that for some functions, the Boolean expression has too many terms to enumerate.

An important advantage of the proposed FPTLA is that a PNAND majority gate is programmed to compute a threshold function just *by routing appropriate input sig-*

nals (including constants 1 and 0) to its input ports. Since PNAND functions require complemented signals, K SRAMs for PNAND-K and other circuitry are required to compute the complements of the signals.

5.3 FPTLA Architecture

In this section, we describe a gate array architecture consisting of PNAND cells that we shall refer to as Field Programmable Threshold Gate Array (FPTLA). As stated earlier, a PNAND cell can be viewed as an edge triggered multi-input flipflop that computes threshold functions. Hence a network of PNANDs constitutes a gate-level or nanopipelined circuit, where each element is clocked, and each path has exactly same number of threshold gates. It appears as a network of flipflops with no intervening combinational logic gates. This pipelined circuit is then mapped onto a gate array architecture, which is described below.

The FPTLA architecture is similar to the conventional island style FPGA programmable fabric (Fig. 5.2). However the logic blocks in FPGA are replaced with PNAND blocks (DTGB). A DTGB consists of a PNAND-7 majority gate shown in Fig.5.1. Each input of the DTGB can be fed to the majority in true or complemented form using an XOR gate and a configurable SRAM. A DTGB has total of 8 inputs and 1 output and only 7 SRAMs. Compared to a DTGB, a standard 4-input LUT based CLB has more than 2X transistors considering those in SRAM bits. Also note that PNAND-K requires K SRAMs whereas LUT-K requires 2^K SRAMs.

The C blocks denote Connection boxes. The C boxes connect the channel wires to the input and output pins of the DTGBs. The S-boxes or the switch boxes allow wires to switch between vertical and horizontal wires. The purpose of this routing grid is to connect appropriate signals to the inputs of each of DTGB including the required constants.

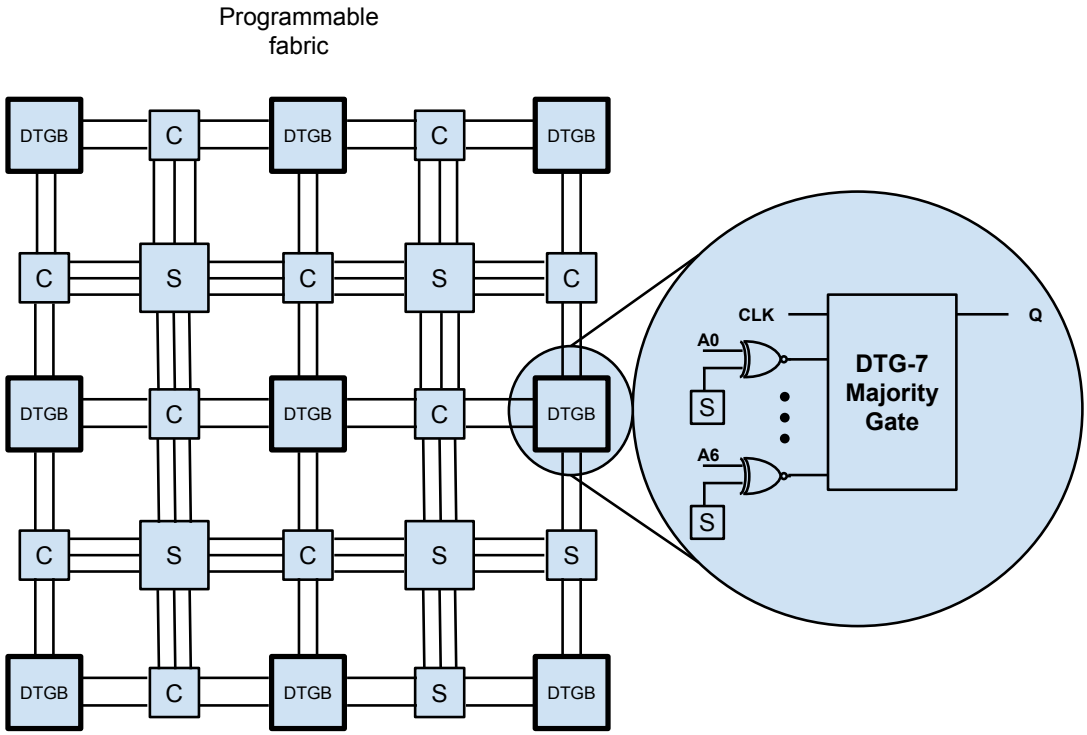


Figure 5.2: A PNAND Gate Array

It should be noted that the DTGB is **not** functionally equivalent to an LUT but rather an alternate programmable unit. There have been similar attempts where an alternate functional unit such as ULMs (Zilic and Vranesic, 1996) were used. In this work, we specifically focus on a programmable threshold logic element that can implement a certain set of threshold functions, and compare the resulting design with a conventional FPGA realizing the same overall function.

Alternative designs of logic blocks are possible where an LUT is augmented with a PNAND. It is also possible to model FPTLA as a learning network where weights of variables can be altered by routing appropriate signals to the inputs of DTGBs. However these considerations are beyond the scope of the present work.

5.4 Synthesis of Nanopipelined Threshold Networks

Any threshold synthesis algorithm requires identifying whether given Boolean function is threshold or not. Since the set of threshold functions is restricted to those that can be realized by a PNAND-7 (configured as a 4-outof-7 majority), a simple ILP based identification approach suffices. Thus we can limit our identification procedure to only those threshold functions that can be implemented by a PNAND-7.

The proposed threshold synthesis method is an area minimization technique based on the method in (Zhang and Cotofana, 2005). The central idea of the area minimization technique is a well known operation in Boolean synthesis called *node elimination*. In this operation, the logic expression (SOP) of the node is substituted into the logic expressions of all its fanout nodes. In traditional Boolean synthesis, this operation has no particular constraints and simple objectives such as minimization of fanin/fanout or speed (levels) and area (number of gates) can be followed. However in the proposed threshold network synthesis, a substitution is permitted if and only if the resultant logic function is also threshold. Fig. 5.3 (a) shows an example of how a threshold gate is eliminated by substituting its expression into that of its fanout gates. Fig. 5.3 (b) shows a case where substitution is not allowed as the resultant function is not threshold.

The node elimination technique can be iteratively applied to an initial synthesis of a threshold logic network. The elimination of nodes in each iteration results in a network of fewer and larger threshold gates. This is the basis of the area minimization technique.

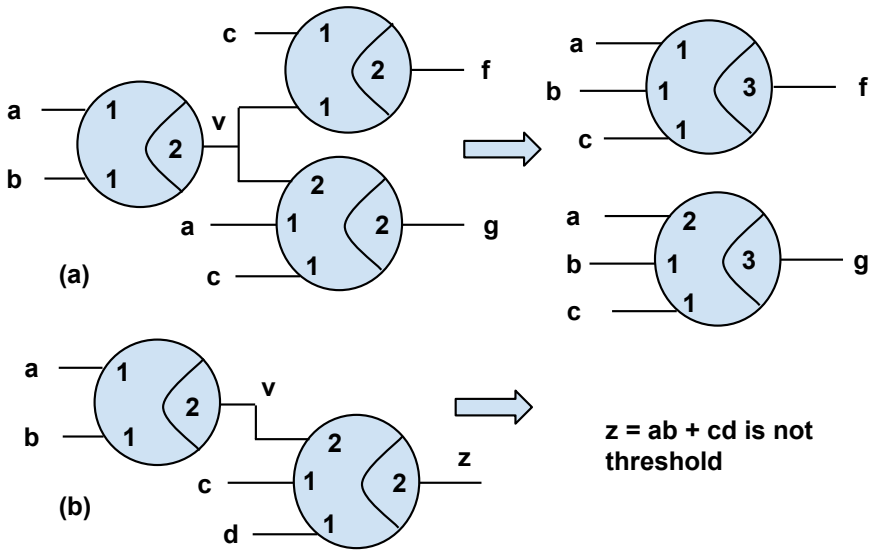


Figure 5.3: (a) Legal Function Substitution (b) Illegal Function Substitution Leading to a Non-threshold Function

5.4.1 Area Minimization Algorithm

The area minimization algorithm works on a directed acyclic graph (DAG) representing a netlist of threshold logic gates. Let $D(N, A)$ denote the circuit DAG. N denotes the set of nodes and A denotes the set of directed edges in D. The outline of the procedure is shown in Algorithm 3.

The input to the algorithm can be any netlist, where the nodes represent threshold functions that can be implemented by PNAND-7. This includes a conventional AND/OR network. Hence in our implementation, we start with a 2-input AND-OR network. At each step in the algorithm, the largest subset of nodes that can be removed simultaneously is obtained and these nodes are then eliminated. The process is repeated until no nodes can be eliminated. In an elimination step, the set of all nodes that can be eliminated is obtained. Since all such nodes may not be eliminated simultaneously, a graph H is constructed denoting the relationship between nodes that can be simultaneously eliminated. A pair of removable nodes (u,v) are adjacent

```
MinimizeArea(G);
input  : A threshold network G
output: A reduced area threshold network
while True do
    S = set of nodes that can be eliminated in G;
    //Construct an undirected graph H(V, E);
    V = S;
    foreach pair of nodes (u, v) ∈ S do
        if (u, v) ∈ A OR (v, u) ∈ A OR u and v have a common fanout then
            Add edge (u,v) to E;
        end
    end
    //Assign weights to the nodes in H;
    foreach node u ∈ V do
        // N_u is the set of successors for gate u;
        // C is a large constant;
        W_u = −C ∗ |N_u|;
    end
    // Compute maximum weighted independent set in H.
    T = MWIS(H, W);
    if |T| == 0 then
        break;
    end
    G = Eliminate all nodes in T from D by substitution;
end
return G;
```

Algorithm 3: Area Minimization Algorithm

in H if and only if they share an edge or a common fanout gate in the circuit DAG and therefore cannot be simultaneously removed. The largest subset that can be eliminated is obtained as a maximum weighted independent set in H. The weights assigned to each node are based on the number of their fanout nodes. It is desirable to eliminate the nodes with the fewest number of fanout nodes first. Therefore they are assigned the highest possible weight. Other criteria can also be used to assign weights. For example, nodes that increase the complexity (or sum of weights) of the resultant threshold function upon substitution can be assigned lower weights whereas the nodes that decrease the complexity can be assigned higher weights.

5.4.2 Buffer Insertion

An important requirement in nano-pipelined circuits is that each path from any primary input to any primary output have the same number of clocked elements. Often, the logic circuits consist of unbalanced paths that need be balanced by using timing buffers. In the proposed architecture, a single DTGB can be used as a buffer gate. However it is desired that the number of buffers should be as small as possible since buffers do not contribute to useful computation. The buffer insertion problem for path balancing of directed acyclic graphs is a well studied problem and an LP formulation is provided in (Hu *et al.*, 1994). However in this work we use a much simplified algorithm to insert buffers. Following is the simpler buffer insertion algorithm used in this work. Note that using a sophisticated buffering algorithm or synthesis methods that minimize the buffers can further improve results shown here.

The main idea behind the buffer insertion algorithm is that each threshold gate must be (and need be) buffered only as many times as its farthest feed-forward successor (fanout node). Any other successor (which must require fewer buffers) can use the output of one of the buffers in the cascade to ensure balanced paths.

LEVELIZE(G);

input : A threshold network G

output: The threshold network G with all paths of equalized using buffers

foreach *node* $u \in G$ **do**

  compute maximum distance L_u of node u from primary inputs;

end

foreach *node* $u \in G$ **do**

  $B_u = -1$;

  foreach *directed edge* (u, v) *in* G **do**

    $B_u = \max(B_u, L_v - L_u - 1)$;

  end

  Insert a cascade of buffer nodes named u_1, u_2, $\cdots$, u_{B_u} after the node u;

  foreach *directed edge* (u, v) *in* G **do**

    index $= L_v - L_u - 1$;

    if *index* > 0 **then**

      replace edge (u, v) with (u_{index}, v);

    end

  end

end

Algorithm 4: Buffer insertion algorithm to balance all paths

5.5 Robustness and Temperature Monotonicity

Differential mode logic is generally more sensitive to process variations, and PNANDs are no exception. The robustness of the PNAND cell was explored using Monte-Carlo simulations. The reliability of the cell depends on the difference in the conductance of the two networks as perceived by the sense amplifier. The robustness of the cell is higher if this difference is larger. The PNAND cell was optimized to increase the difference in the conductance.

There are two fundamentally different viewpoints to assess the reliability of a cell.

1. A PNAND cell is said to work if and only if it correctly computes the output for all possible input patterns.

2. Alternatively, if a cell does not implement the majority function, it can be shown that the cell implements some threshold function. Consider the inequality $\sum_i W_i x_i \geq T$. The process variations simply perturb the value of the weights represented by the conductance of each of the transistors in the input networks. Regardless of the perturbation in the weights from the nominal, the resultant expression is still an arithmetic inequality and therefore represents a threshold function. In a field programmable environment, we can determine the set of functions of a cell post fabrication and an intelligent mapping procedure can still use the cells which do not necessarily implement a large majority function.

In this work however, we restrict our analysis to option (1) mentioned above. Therefore a PNAND cell instance (represented by a trial in Monte-Carlo simulation) is said to succeed if it correctly implements its majority function. The maximum number of function alterations occur at the Slow-NMOS-Fast PMOS (asymmetric corner) at -40C. The worst input configuration for PNAND-7 is 4/3 (or 3/4). The

optimized PNAND-7 was simulated for 100,000 Monte-Carlo trials at 0.9V supply voltage for the worst-case input configuration. We observed that the 94320 out of the 100,000 circuits computed the 4/3 case correctly. This means that the probability of a PNAND-7 cell computing the any of the functions in Table 5.1 is 0.9432.

5.5.1 Temperature Monotonicity

As temperature increases, we observed that the probability of correct computation of a PNAND cell also increases. However, whether or not a given cell that works at lower temperature also works at a higher temperature needs to be asserted. For this exploration, we chose a PNAND-9 implementing 5 out of 9 majority, because it was far less robust than a PNAND-7. The objective is to determine that the effect of temperature on the success rate of the cell.

First, using Monte Carlo sampling, 2000 instances of a PNAND-9 were generated, simulating process variations. Each of the instances were simulated for temperatures from -40C to 120C, in increments of 10C. At each temperature, the set of cells that correctly computed the output were noted. The result is that set of properly working cells at temperature T_i was a subset of the cells that worked at temperature T_j, for $T_j > T_i$. Fig. 5.4 displays the results in a Venn diagram. The innermost set named F1825 (named based on the number of successes in 2000 viz. 1825) is the set of cells that passed the success criteria at -40C. The successive circles show the sets that worked at progressively higher temperatures. Thus a PNAND cell that passes the success criteria at lower temperatures is guaranteed to work at higher temperatures. The monotonicity of temperature has an important role to play in the practical usability of the field programmable array of PNAND cells. A cell in the array need only be tested post fabrication at the lowest allowed temperature and can be certified to work at all the higher temperatures.

There is a simple explanation for this behavior. Suppose there are n devices ON in the LIN and $n-1$ ON in the RIN. Let Z_L and Z_R denote the impedances of the LIN and RIN, then $Z_L \approx \frac{1}{n}$ and $Z_R \approx \frac{1}{(n-1)}$. Then the greater the difference $|Z_L - Z_R| = \frac{1}{n(n-1)}$, the more likely that it will compute the correct value. As n increases, each individual impedance decreases, as does their difference (quadratically). The increase in temperature counteracts this effect by increasing the impedance of the two input networks. Moreover, the temperature effect is nonlinear. Thus the increase of the impedance of the LIN and RIN need not be the same, and in fact, the impedance of the RIN increases slightly more than that of the LIN, resulting in a increased likelihood of the output being 1. This is the principal reason why there are fewer failures of PNANDs at higher temperatures.

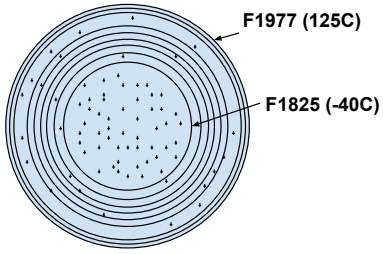


Figure 5.4: Monotonicity of Temperature

5.6 Experimental Results

The experiments were performed using the well known VTR (Verilog-to-routing) framework along with customized technology and architecture definition files. The conventional LUT based FPGA circuits were constructed using a 4-input LUT since it has been shown that LUT size of 4 results in the most area-efficient FPGAs (Rose *et al.*, 1990). Although we also present the results for 6-input LUT.

5.6.1 Parameters for VPR

The VPR tool requires an architecture specification file and a technology file to place and route a netlist and compute its delay and power. The CLB used for FPGA consists of a 4-input LUT driving a D-flipflop and an output mux that selects between the LUT output and the D-flipflop output. The architecture file was populated with delay and power data measured from a complete spice model of CLB. The muxes and the D-flipflop used in the spice model were the fastest implementations chosen from a commercial 65nm standard cell library. The fastest 2:1 mux is a transmission gate based mux with an output driver and has 12 transistors. The D-flipflop has 40 transistors. Therefore, seventeen 2:1 muxes, 17 SRAMs and a D-flipflop together result in (17 x 12 + 17 x 6 + 40 =) 346 transistors in a CLB.

The DTGB architecture used is shown in Fig. 5.2. It consists of an optimized PNAND-7 laid out as a standard cell and has 36 transistors. The fastest XOR gate from 65nm commercial library has 10 transistors. Therefore one PNAND-7, 7 XORs, 7 inverters and 7 SRAMs result in (36 + 7 x 10 + 7 x 2 + 7 x 6 =) 162 transistors in a DTGB. In a similar manner, entire DTGB was modeled in SPICE and detailed measurements such as setup times, delays at the experimental corner (nominal, 1.2V, 25C), energy per toggle of each of the input pins and static power were provided. The required fields in both FPGA and FPTLA architecture specification files were populated using the measured data. The routing related fields such as wire delays, and capacitances were calculated based on the 65nm technology files. Additional routing related fields such as number of tracks per channel that C-box and S-box connect to (F_c, F_s), and the configurations for clock tree were kept identical for both FPGA and FPTLA, since the clock pin capacitances of both logic blocks are same.

Table 5.2 shows the comparison of the characteristics of a CLB and DTGB.

The input capacitance of a PNAND cell is extremely small since the input network transistors are nearly minimum size. Additionally a DTGB computes only on clock edge. Therefore the input pin toggle energy of DTGB is much smaller compared to that of an LUT. On the other hand, the clock toggle energy of CLB consists of energy of a single D-flipflop driving the output mux, whereas DTGB computes and stores the output of a threshold function.

The threshold synthesis and the buffer insertion procedure were implemented in Python. The threshold identification uses the ILP solver package lpsolve (v 5.5). The results are shown (Table 5.3) for 6 combinational datapath circuits suitably nanopipelined, placed and routed using VPR for both FPGA and FPTLA.

The first circuit is a 128-bit comparator, which compares a pair of 128-bit unsigned integers a and b and has three outputs which denote whether a ¿ b, a ¡ b or a == b. The second one is a 16-bit bit-level sorter with 16 inputs and 16 outputs. A bit-level sorter treats its inputs as an array of bits and produces sorted array. The sorter is a useful computing block that enables efficient implementations of large symmetric functions such as large majorities, parity and counting functions that are typically used in arithmetic circuits. The remaining 4 circuits are ISCAS combinational benchmarks. We specifically chose one small, two medium and one large to denote the scalability of our approach. Note that the threshold netlist of Comparator and Sorter were manually designed whereas the ISCAS benchmarks were synthesized using the proposed synthesis method.

5.6.2 Circuit Comparison Results

Table 5.3 shows the comparison between FPGA and FPTLA implementations of various circuits for several parameters.

Since FPTLA circuits are nanopipelined, they are inherently fast. Therefore the

conventional FPGA implementations were also pipelined using retiming for minimum delay with k stages where k was varied from 2 to the same number of pipeline stages as FPTLA circuits. Then the circuit that exhibited the minimum EDP was chosen for comparison. For 4 out of 6 circuits, the lowest EDP FPGA-circuits were also the fastest. For C3540, the fastest FPGA implementation was 1.87ns which was still 1.6X slower than the FPTLA. Moreover at this speed, the EDP of the FPTLA was 1.6X better than FPGA implementation. For C6288, the minimum EDP of FPGA is almost same as the FPTLA however that particular FPGA implementation is 4.3X slower than FPTLA.

The channel width of the FPTLA implementations was same or better across all the circuits. The column labeled *# Cells* represents N, for $N \times N$ array.

Although the FPTLA has more cells, the individual DTGB has half the number of transistors of a CLB. A smaller logic block (with much fewer SRAMs) can result in shorter lengths of wire which leads to smaller buffers, muxes and other routing resources for FPTLA. Without optimized layouts of DTGB and CLB, the impact of logic block sizes on routing infrastructure and the entire gate array remains somewhat speculative. However we provide comparison of number of transistors (column # LB xtors) in all the logic blocks in the grid for both FPGA and FPTLA implementations. In passing, we note that the impact of routing infrastructure, although substantial in conventional FPGAs, can be greatly minimized in near future with the help of emerging nano-technologies such as RRAM (Resistive RAM) based interconnects. For example, (Cong and Xiao, 2014) demonstrates an RRAM based routing infrastructure that has about 96% lower silicon area and 70% lower power than the routing infrastructure in conventional FPGAs. In our experiments we observed that, on average more than 85% of the FPTLA delay and 90% of total energy is consumed by routing infrastructure whereas for conventional FPGA implementations, these ratios

(on average) were 57% and 74%.

It is also possible to scale down the voltage of FPTLA until the performance of FPTLA and FPGA are matched. We expect the voltage scaling to provide as much as 2X improvement in power at the same performance. However the VTR tool does not support voltage scaling at present.

Similar results are shown in Table 5.4 with 6 input LUT are used in FPGAs. The FPTLA numbers are same however the FPGA is implemented with 6-input LUT. The 6 input LUT almost always reduce the number of gates (size of FPGA grid) but they do not significantly reduce clock period. On the other hand, they exhibit higher energies.

Martin et al. (Martin *et al.*, 2002) propose Energy-Delay squared product (ED^2) as a better voltage independent metric over Energy-Delay product (ED). They show that an optimal ED^2 implies optimal energy (E) and delay (D). For example, consider two circuits with identical ED products at a given supply voltage. We can reduce the supply voltage and equalize the delay of two circuits. In this case, the circuit with lower delay will consume lower energy because of quadratic dependence of energy on voltage. Therefore in addition to ED, we also present comparison of normalized ED^2 (Fig. 5.5).

5.6.3 Leakage Power and Glitching

FPGAs are often plagued by substantial static power due to their large areas. In our experiments we use LP (low power) commercial models of 65nm transistors which offer extremely low leakage current. Therefore static power portion of total power is substantially smaller (0.01%) for all the circuits in FPGA as well as FPTLA. However we note that a DTGB has 50% lower leakage than CLB (excluding SRAMs) and has 60% fewer SRAMs. Additionally we can scale down the voltage of entire

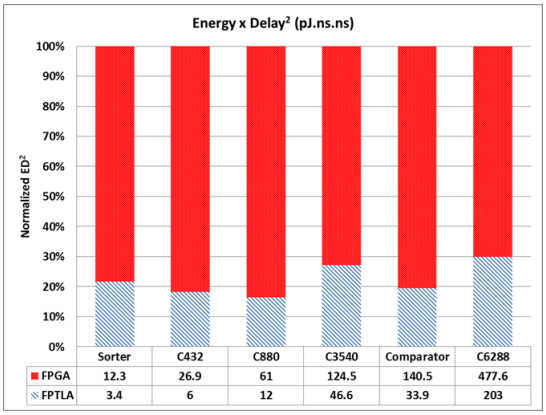


Figure 5.5: Comparison of ED^2 between FPGA and FPTLA Circuits

FPTLA (routing infrastructure, SRAMs, DTGBs) to match speed of FPGA which will further reduce the static power of FPTLA.

The second important contributor to energy wastage is the *glitching*. In non-pipelined circuits, the glitching can account for more than 70% of dynamic power (Lamoureux *et al.*, 2008). FPTLA circuits have nearly zero glitching since these circuits are nanopipelined. A pipelined FPGA may have such low glitching only when pipelining is deep enough resulting in many LUTs used as mere buffer gates. Coupling this with higher dynamic and static power of CLBs, ED of heavily pipelined conventional FPGA circuits can be substantially higher. For example, the ED of an FPGA implementation of the 128-bit comparator with same number of pipeline stages as FPTLA counterpart, is 2.5X higher than ED of the FPTLA circuit and ED^2 is 5.3X higher.

The switching activity of primary inputs used in all our experiments was set to 20%. It should be noted that as the primary input switching activity increases, the non-pipelined circuits often suffer from excessive glitching. The DTGB power is fairly constant over all switching activities. We compared the ED advantage of the 128-bit comparator for various switching activities between FPGA and FPTLA

implementations. We observed that the advantages in ED for FPTLA grow from 1.7X at 10% switching activity to 2.1X at 30% switching activity.

Another important characteristic to be noted is that the energy per clock toggle of PNAND over all of its input patterns is almost same, with standard deviation less than 10% of the mean. This would result in much more accurate estimation of power, as well as greater resilience to the power profile attacks for cryptographic processors.

5.6.4 Circuit Yield

Given a circuit with several PNAND cells, a few cells are bound to compute a different function than the majority under process variations. Therefore some redundancy needs to be introduced in order to ensure a high circuit yield. Given a circuit having N cells the redundancy is defined as additional m cells that need to be present in the chip to ensure that at least N cells work with a very high probability. If $P_{4/7}$ denotes the probability that a 4 out of 7 majority gate functions correctly, the probability of success (yield) of the circuit Y is given by the following expression.

$$Y = \sum_{K=0}^{m} \binom{N+m}{K} {P_{4/7}}^{N+m-K} (1 - P_{4/7})^{K} \tag{5.1}$$

The probability that a PNAND cells functions correctly is 0.9432. Table 5.5 shows the redundancy (additional cells in gate array i.e. m) required for each circuit in order to ensure that $Y \geq 0.999$ and also for $Y \geq 0.999999$. It can be seen that a very high yield can be ensured with very few additional cells.

Table 5.1: Threshold Functions Implementable by 4/7 Majority

#	Function	Programming pattern	Boolean expression
1	[1; 1]	a, 1, 1, 1, 0, 0, 0	a
2	[1,1; 1]	a, b, 1, 1, 1, 0, 0	a+b
3	[1,1; 2]	a, b, 1, 1, 0, 0, 0	ab
4	[1,1,1; 2]	a, b, c, 1, 1, 0, 0	ab+bc+ac
5	[1,1,1; 1]	a, b, c, 1, 1, 1, 0	a+b+c
6	[1,1,1; 3]	a, b, c, 1, 0, 0, 0	abc
7	[1,1,1,1; 2]	a, b, c, d, 1, 1, 0	6 terms
8	[1,1,1,1; 3]	a, b, c, d, 1, 0, 0	4 terms
9	[1,1,1,1,1; 3]	a, b, c, d, e, 1, 0	10 terms
10	[2,1,1; 2]	a, a, b, c, 1, 1, 0	a+bc
11	[2,1,1; 3]	a, a, b, c, 1, 0, 0	ab+ac
12	[2,1,1,1; 3]	a, a, b, c, d, 1, 0	a(b+c+d)+bcd
13	[1,1,1,1; 1]	a, b, c, d, 1, 1, 1	a+b+c+d
14	[1,1,1,1; 4]	a, b, c, d, 0, 0, 0	abcd
15	[1,1,1,1,1; 2]	a, b, c, d, e, 1, 1	10 terms
16	[1,1,1,1,1; 4]	a, b, c, d, e, 0, 0	5 terms
17	[1,1,1,1,1,1; 3]	a, b, c, d, e, f, 1	20 terms
18	[1,1,1,1,1,1; 4]	a, b, c, d, e, f, 0	15 terms
19	[1,1,1,1,1,1,1; 4]	a, b, c, d, e, f, g	35 terms
20	[2,1,1,1; 2]	a, a, b, c, d, 1, 1	a+bc+cd+bd
21	[2,1,1,1; 4]	a, a, b, c, d, 0, 0	a(bc+cd+bd)
22	[2,1,1,1,1; 3]	a, a, b, c, d, e, 1	8 terms
23	[2,1,1,1,1; 4]	a, a, b, c, d, e, 0	7 terms
24	[2,1,1,1,1,1; 4]	a, a, b, c, d, e, f	15 terms
25	[2,2,1,1; 3]	a, a, b, b, c, d, 1	a(b+c+d)+b(c+d)
26	[2,2,1,1; 4]	a, a, b, b, c, d, 0	ab+acd+bcd
27	[2,2,1,1,1; 4]	a, a, b, b, c, d, e	7 terms
28	[3,1,1,1; 3]	a, a, a, b, c, d, 1	a+bcd
29	[3,1,1,1; 4]	a, a, a, b, c, d, 0	a(b+c+d)
30	[3,1,1,1,1; 4]	a, a, a, b, c, d, e	5 terms

Table 5.2: CLB and DTGB Properties Measured using Detailed SPICE Models (supply = 1.2V)

Property	CLB	DTGB	Ratio
No. of Transistors	346	162	2.1X
I/P toggle Energy (fJ)	43.5	3.2	13.6X
Clock toggle Energy (fJ)	10.6	15.1	0.7X
Delay LUT (ps)	292	129	2.26X
Static Power (nW)	5.5	2.93	1.87X

Table 5.3: FPGA and FPTLA Circuit Results at Fixed 1.2V Supply with LUT-4

	Delay(ns)			EDP (pJns)			Channel Width		# Cells		# LB xtors	
Circuit	FPGA	FPTLA	Ratio	FPGA	FPTLA	Ratio	FPGA	FPTLA	FPGA	FPTLA	FPGA	FPTLA
Sorter	1.74	0.91	1.9X	7.0	3.8	1.9X	8	8	11	14	41866	31752
C432	1.84	0.83	2.2X	14.6	7.3	2.0X	10	8	17	20	99994	64800
C880	2.31	0.88	2.6X	26.4	13.6	1.9X	10	10	19	24	124906	93312
C3540	2.39	1.13	2.1X	52.1	41.3	1.3X	14	14	24	32	199296	165888
128-bit Comparator	2.29	1.08	2.1X	61.4	31.4	2.0X	14	10	28	31	271264	155682
C6288	2.61	1.4	1.9X	183	145	1.3X	18	18	38	47	499624	357858

Table 5.4: FPGA and FPTLA Circuit Results at Fixed 1.2V Supply with LUT-6

	Delay(ns)			EDP (pJns)			Channel Width		# Cells		# LB xtors	
Circuit	FPGA	FPTLA	Ratio	FPGA	FPTLA	Ratio	FPGA	FPTLA	FPGA	FPTLA	FPGA	FPTLA
Sorter	1.91	0.91	2.1X	15.07	3.8	4X	8	8	10	14	82600	31752
C432	1.81	0.83	2.2X	25.03	7.3	3.4X	8	8	17	20	238714	64800
C880	2.62	0.88	3.0X	42.22	13.6	3.1X	14	10	15	24	185850	93312
C3540	2.82	1.13	2.5X	57.34	41.3	1.4X	16	14	21	32	364266	165888
128-bit Comparator	2.26	1.08	2.1X	122.89	31.4	3.9X	16	10	27	31	602154	155682
C6288	2.26	1.4	1.6X	223.71	145	1.5X	18	18	28	47	647584	357858

Table 5.5: Redundancy m to be Added for Each Circuit

Circuit	# Gates	m (@Y=99.9%)	m (@Y=99.9999%)
Sorter	184	30	30
C432	521	20	30
C880	763	30	40
C3540	1407	50	60
128-bit Comparator	939	60	70
C6288	952	90	100

Chapter 6

NEW CLOCK SKEWING STRATEGY

6.1 Introduction

In the previous chapters, we introduced a new type of threshold gate referred to as PNAND, that integrates a threshold function with an edge triggered flipflop. An important characteristic of PNAND cells is that they produce two signals, labeled by N1 and N2 (see Fig. 3.1) that are complements of each other at the end of the evaluation phase. Thus N1 and N2 together signal the end of the computation. After the clock transitions from 0 to 1, (N1, N2) = (0,1) or (1,0) signals the end of computation. This means that when the *NAND(N1,N2)* transitions from 0 to 1, the computation has been completed. Thus every PNAND or KVFF can be made to generate a completion signal. This property can be exploited in several ways, one of which is the use of *local clocking* to further improve the design in terms of power, area and performance. In this chapter, we present a novel clock skewing strategy that exploits the availability of completion signals with KVFFs.

6.1.1 Overview of Clock Skewing

A general sequential circuit (see Figure 6.1), consists of a network of combinational *clouds* (acyclic network of logic gates), interconnected by registers (Maheshwari and Sapatnekar, 1999). Conventional design of such networks is based on the assumption that every register receives the clock signal (assuming single phase clocking) at exactly the same time. In practice, guaranteeing simultaneity of clock arrival times is not possible due to gate and interconnect delays. The difference in the clock arrival times

at two registers is referred to as the clock *skew* between those registers. There is an extensive body of literature, spanning two decades, on the optimal design of a clock networks aimed at minimizing the skew, and accounting for it in the maximization of the clock frequency (Maheshwari and Sapatnekar, 1999; Ivan S. Kourtev, 2009).

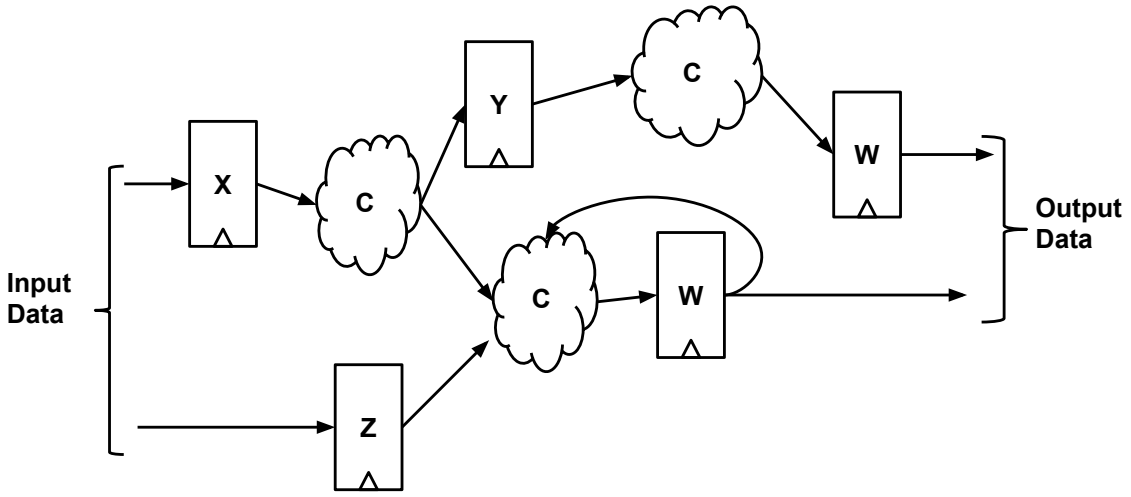


Figure 6.1: A General Synchronous Digital Circuit Architecture

Clock skew can also be deliberately introduced to improve the performance (Fishburn, 1990), power and even area of a digital circuit. This topic has also been extensively developed and modern design tools routinely perform clock skew based design optimization. This paper describes a new approach to the generation and utilization of skewed clocks for the purpose of reducing dynamic power without sacrificing performance. Figure 6.2 shows a simple example of how clock skewing can be used to improve the performance and area of a sequential logic circuit. Clock skewing is implemented by introducing delay buffers in the clock distribution network. Let A_i denote the arrival time of the clock at register R_i. Then the clock skew between two registers, R_i and R_j, which are separated by a combinational cloud, is $A_i - A_j$. Fig. 6.2 (b) shows how clock skewing can used to reduce the clock period from 3 to 2, by delaying the clock input to flipflop Y by 1 unit. The general problem of clock skew

scheduling is to determine the arrival times of the clock signal to each register subject various timing constraints, with the goal of optimizing some objective function such clock period (Fishburn, 1990), dynamic power (Vijayakumar and Kundu, 2014), etc. This is most often formulated as a linear programming problem.

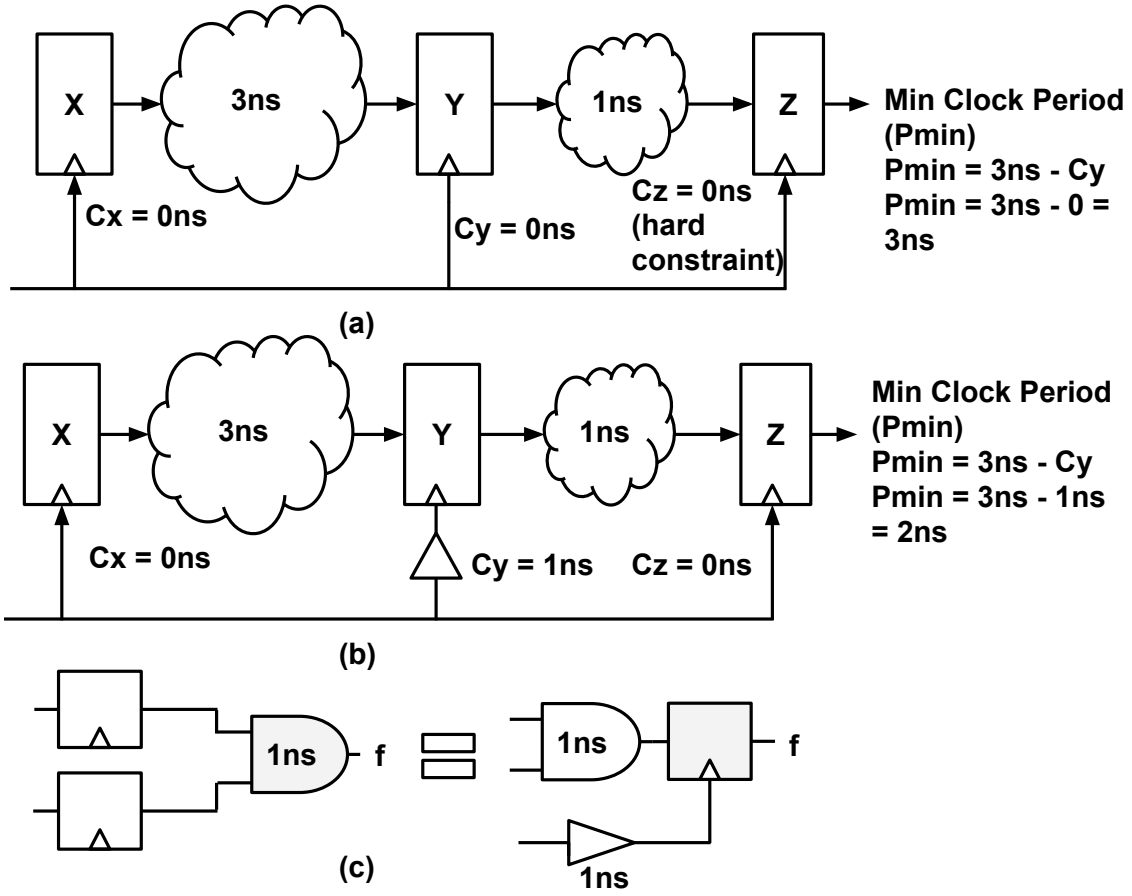


Figure 6.2: (a) Clock Skewing Used to Increase Speed of the Circuit (b) Clock Skewing Used to Reduce Number of Flipflops

Clock skewing is closely related to *retiming*. Their equivalence was first shown in (Rahul B. Deokar, 1995). Figure 6.2(b) shows a simple example, in which deliberate clock skewing is equivalent to retiming, and results in fewer registers. In fact, clock skewing can theoretically achieve the same advantages as retiming without altering the number and positions of the registers. However, in practice, it cannot

simply replace retiming, because it requires much greater precision in timing of clock arrivals. Another use of clock skewing is to improve the robustness of the circuit in the presence of process variations. A quadratic programming formulation of the clock skew scheduling problem to maximize the robustness of the circuit in presence of timing inaccuracies is presented in (Ivan S. Kourtev, 2009). Clock skewing has also been used to reduce the power of the circuit, specifically power wasted due to glitching (Vijayakumar and Kundu, 2014).

In this chapter, a new clock skewing technique is shown which extends the use of clock skewing to reduce the dynamic power, without introducing any additional buffers in the clock network. The key idea behind reducing dynamic power is based on the observation that it is possible to introduce timing slacks on selected paths with large combinational logic cones without violating timing constraints. Any positive slack on such paths is exploited by technology mapping and layout tools to reduce the size and power of the combinational logic. While slack-based optimization is not new, our approach to both creating the slack and not requiring additional buffers is new. Not requiring buffers to skew the clock has some obvious advantages. It can reduce the buffer area, and dynamic as well as the leakage power. Moreover, it places less constraints on the design and results in faster timing closure. This is because, buffer delays need to be very precise to achieve timing closure when introducing deliberate clock skew. When that precision is not achievable, it limits the use of clock skewing.

An important consequence of the proposed method is that it eliminates hold-time violations on certain short paths, again without introducing any buffers in the datapath. A short path is a register-to-register path with little or no logic in between the registers. These paths are prone to hold-time violation, which occurs when the signal being latched by a flipflop is not held constant for a sufficient duration required

by the flipflop. Hold-time violations are important because they render the circuit unusable as they cannot be eliminated by external controls such as the supply voltage or the clock frequency.

The main contributions are summarized below.

1. Using KVFFs with completion detection we introduce a new clocking scheme, referred to as local clocking, that optimally determines the sources and destinations of the skewed clocks.

2. We present a precise formulation of the optimization problem to maximize the slack, which is later reclaimed by area minimization. The proposed method is applicable to general ASIC circuits.

3. We demonstrate how local clocking can eliminate hold-time violations.

6.1.2 KVFF with Completion Detection

Fig. 6.3 shows the architecture of the KVFF flipflop with an output local clock.

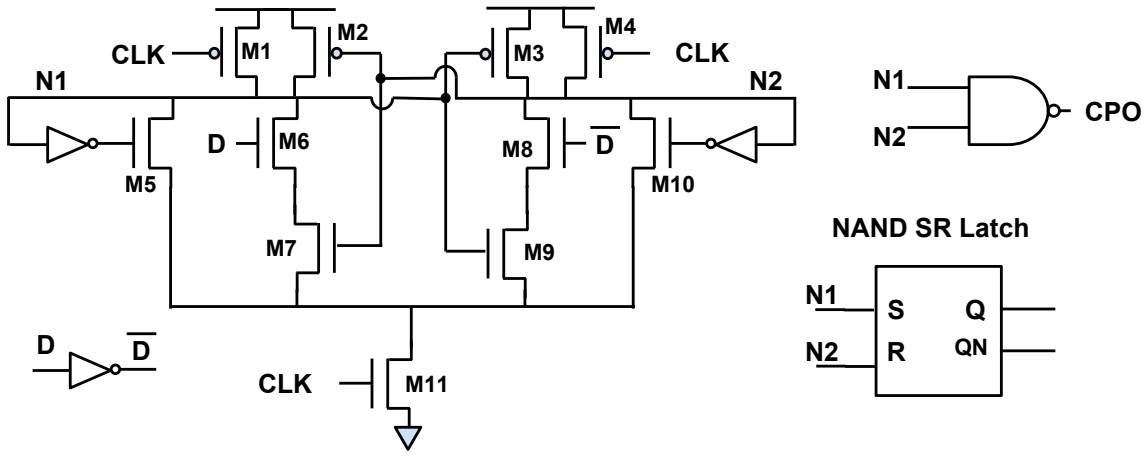


Figure 6.3: The KVFF Architecture With Output Local Clock

When CLK = 0, N1 and N2 are both 1, therefore CPO is also 0. Similarly when CLK = 1, N1 and N2 assume opposite values depending on the value of input D. But in either case one of N1 or N2 is 0 and therefore CPO = 1. Therefore we note that output CPO follows input CLK but is a combinationally delayed. Also CPO changes only after the input state (D) is successfully read. Therefore output CPO is a "safe" clock signal to be used to clock other flipflops as input D may be altered only after it is latched by the flipflop. It is possible to use a simple buffer to produce output clock. However, the output clock in such cases cannot guarantee that input D will not change before it is stored by the flipflop.

6.1.3 Clock Skewing using KVFF with Completion Detection

Fig. 6.4 shows an example circuit where skewing with local clock can provide substantial reduction in the circuit power and area without changing the clock period. The large cone used in this circuit is the MSB (most significant bit) of the 64-bit multiplier's output. The flipflop Y feeds the local clock to the flipflop X thereby providing an additional slack to the flipflop X. Due to this, the area in the feeder of the flipflop X can be reduced considerably without any performance loss. Table 6.1 shows the advantages in terms power, area and delay after automated place and route(P & R) using Cadence Encounter, of the circuit in Fig. 6.4 compared to the conventional globally clocked circuit.

The main advantage in this particular example are the area and power reduction of the combinational logic under additional slack provided by the delayed local clocking. The problem of reducing gate sizes under timing slack is a well studied problem and is routinely implemented in commercial synthesis and layout tools. Note that the clock period of the circuit was not changed i.e. both the circuits (original and locally clocked) were processed at the same clock period which was 2.6ns.

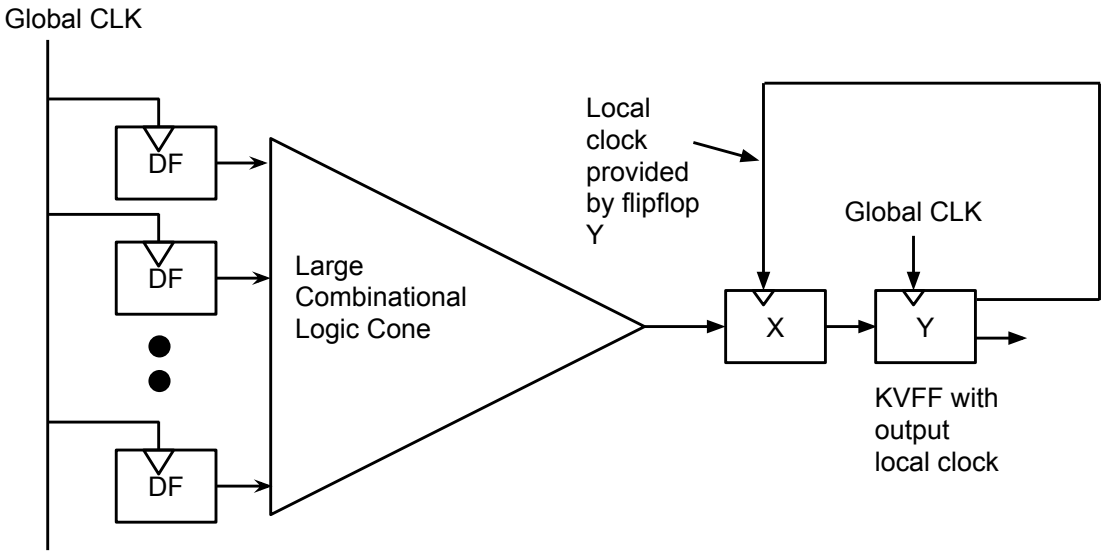


Figure 6.4: A Motivational Example that Exhibits Maximum Advantages due to Local Clocking

Table 6.1: Advantages of Local Clocking for the Circuit from Fig. 6.4 @ TT,1.2V,25C

Parameter	Area	Power	Leakage	Total circuit cap
Globally clocked	127410 μm^2	163.7 mW	13.3 μW	264 pF
Locally clocked	84698 μm^2	90.8 mW	6.6 μW	156.3 pF
Δ (%)	33%	44%	50%	40%

Since the flipflop X receives a local clock, the total capacitance driven by the global clock tree reduces. Note that this doesn't reduce the total power of the clock tree, but local clocking definitely does not increase burden on existing clock tree but rather alleviates it slightly.

In this particular example, the flipflop Y produces an output clock trigger *after* it has finished latching its D-input. The local clock signal that triggers flipflop X, is therefore *hazard free* i.e. it does not overwrite the data bit latched by flipflop Y. This is the main advantage of the local clock generated using completion detection. The

path from flipflop X to flipflop Y cannot have hold-time violation irrespective of the operational and process corner, and clock/data slews in this circuit. Therefore local clocking can potentially eliminate hold-time violations in such cases.

An alternative application of this local clocking can be simply to eliminate hold-time violations on maximum possible number of short paths. However in this work, we focus on the reduction of power and area by providing skewed clocks to the flipflops with large input combinational logic cones.

6.2 Optimal Clock Assignment

This section describes the generalized problem formulation of the local clock assignment problem. Informally the problem can be stated as follows. Given a sequential circuit consisting only of edge triggered entities, decide the clock input of each of the register to be either the global clock or to be the output clock of any local register such that maximum registers with the largest feeders receive local clocks and all the timing constraints are satisfied. A 0-1 integer programming formulation of the problem is provided as follows.

6.2.1 Notations

Given any sequential synchronous circuit consisting flipflops and combinational logic, it can be represented by a directed graph $G = (N, A)$ where N denotes the set of nodes and A is the set of arcs (directed edges). Each flipflop R_i in the circuit is a node in this graph. There is a directed edge (R_i, R_j) if there is a directed combinational path starting from the flipflop R_i and ending at the flipflop R_j.

For any flipflop R_i assume that,

1. A_i denotes arrival of **clock** (whether local or global) at flipflop R_i

2. E_i denotes the delay from input clock (CLK) to output clock (CPO)

3. S_i denotes setup time

4. H_i denote the hold time

5. T_i denotes the clock-to-output delay

6. $D_{i,j}$ denotes the *maximum* combinational path delay from flipflop R_i to flipflop R_j.

7. $d_{i,j}$ denotes the *minimum* combinational path delay from flipflop R_i to flipflop R_j.

8. P is the clock period of the circuit.

6.2.2 ILP Formulation

Following indicator variables are the decision variables of this formulation.

$$x_{i,j} = \begin{cases} 1 & \text{If } R_i \text{ is clocked by the output clock of } R_j \\ 0 & \text{otherwise} \end{cases}$$

Clock uniqueness constraints: A flipflop R_i receives clock from at most one other flipflop.

$$\forall i, \ \sum_{j=1}^{n} x_{i,j} \leq 1 \tag{6.1}$$

Note that if R_i does not receive clock from any flipflop R_j i.e. $\sum_{j=1}^{n} x_{i,j} = 0$ then it is clocked by the global clock by default.

local clock fan-out constraints: A flipflop can provide local clock to at most a fixed number of flipflops. If this number is increased arbitrarily, the advantages obtained by local clocking are lost due to complex local clock sub-trees. This fan-out

bound depends on several parameters such as process technology, size of the circuit etc. In this formulation, we fix the local clock fan-out to be at most 1.

$$\forall j, \ \sum_{i=1}^{n} x_{i,j} \leq 1 \tag{6.2}$$

Arrival Constraints: When a flipflop R_i is locally clocked by a flipflop R_j, the arrival of clock at R_i depends on the arrival of clock at flipflop R_j and the delay of flipflop R_j in producing output clock.

$$\forall i, A_i = \sum_{j=1}^{n} x_{i,j}.(A_j + E_j) \tag{6.3}$$

Let $y_{i,j} = x_{i,j}.(A_j + E_j)$ and indicate the timing relationship between arrivals of the clocks. But these would be nonlinear constraints. They can be made linear using following additional set of constraints. Let u denote the upper bound on the arrival times of the flipflops. u can be suitably chosen depending on the latency (arrival) constraints of the circuit.

$$\forall i, j, \quad y_{i,j} \leq u.x_{i,j} \tag{6.4}$$

$$\forall i, j, \quad y_{i,j} \leq A_j + E_j \tag{6.5}$$

$$\forall i, j, \quad y_{i,j} \geq A_j + E_j - u.(1 - x_{i,j}) \tag{6.6}$$

$$\forall i, j, \quad y_{i,j} \geq 0 \tag{6.7}$$

We can verify that when $x_{i,j} = 0$, $y_{i,j} = 0$ for a given i and j values. Similarly when $x_{i,j} = 1$, $y_{i,j} = (A_j + E_j)$. "No cycle constraints" (described below) ensure that a given A_i is never a function of itself.

No cycle constraints: The $x_{i,j}$ variables denote an adjacency matrix of a directed graph. This directed graph shows the local clocking relationship between the flipflops and is referred to as the "clock graph". Note that the clock graph is different from the flipflop graph (which denotes data dependencies between flipflops) described above.

We can see that the clock graph cannot have cycles, i.e. there cannot exist a set of flipflops each of which receives clock from a flipflop in the same set. Therefore the "no cycle constraints" are encoded as follows.

Let L_i denote an index in the topological ordering of the flipflops (wrt flipflop graph). Note that a valid topological order exists for a directed graph with no cycles (DAG).

$$\forall(i,j), \quad L_i \geq x_{i,j}.L_j + \epsilon \tag{6.8}$$

In the presence of a cycle of flipflops (remember that an edge in this cycle indicates clocking relationship and not data relationship), such as $R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_4 \rightarrow R_1$. We have an inconsistency $L_1 > L_2 > L_3 > L_4 > L_1$ i.e. L_1 is strictly greater than itself. L_i variables have a valid assignment if and only if there is no cycle in the clock graph.

Note that $M_{i,j} = x_{i,j}.L_j$ is a non-linear quantity. We can linearize these constraints as follows. Let v denote the upper bound on the values of L_i variables. Given a graph of n flipflops this upper bound is any constant $> (n+1).\epsilon$

$$\forall i,j, \quad M_{i,j} \leq v.x_{i,j} \tag{6.9}$$

$$\forall i,j, \quad M_{i,j} \leq L_j + \epsilon \tag{6.10}$$

$$\forall i,j, \quad M_{i,j} \geq L_j + \epsilon - v.(1 - x_{i,j}) \tag{6.11}$$

$$\forall i,j, \quad M_{i,j} \geq 0 \tag{6.12}$$

Setup time constraints: Given a pair of flipflops $(R_i, R_j) \in A$ of the flipflop graph, the setup constraints can be modeled by calculating the required and arrival times at the flipflop R_j and ensuring the required time is higher than the arrival time. For a register-to-register path between registers R_i and R_j, the arrival time of data at R_j is calculated as $A_i + T_i + D_{i,j}$. The required time at register R_j is calculated as

$A_j - S_j + P$. Setup time constraints are satisfied when

$$A_i + T_i + D_{i,j} \leq A_j - S_j + P. \tag{6.13}$$

Rearranging above inequality, the setup constraints are stated as follows.

$$A_i - A_j \leq P - T_i - D_{i,j} - S_j \tag{6.14}$$

The term $P - T_i - D_{i,j} - S_j$ denotes the setup slack available on the path from R_i to R_j. These setup time slacks can be obtained from any commercial synthesis tool for a given pair of flipflops.

Hold time constraints: The hold time constraints for a pair of flipflops $(R_i, R_j) \in A$ of the flipflop graph are as follows.

$$A_i - A_j \geq H_j - T_i - d_{i,j} \tag{6.15}$$

Objective Function: The objective function is to maximize the slack imparted to the largest cones of logic. Therefore the goal is to maximize the total area of the cones rooted at flipflops that can receive local clocks. Let F_i denote the size of the cone of logic driving a flipflop R_i. Note that F_i are known constants. The objective used in the proposed technique is as follows.

$$Max. \sum_{i=1}^{n} F_i (\sum_{j=1}^{n} x_{i,j}) \tag{6.16}$$

One needs to be careful about situations where a solver might produce a lot of local clocks for flipflops with small logic cones as opposed to a few with large cones. The latter is more beneficial as it reduces more logic area under slack while simultaneously reducing layout burden of extra local clock wires. In order to make sure the solutions of these types are produced, we can restrict the objective function to the flipflops whose cones are sufficiently large.

Although not demonstrated in this work, an alternative objective function can be used if the goal is to minimize the probability of hold-time violations in the circuit. The short paths (denoted as pairs of flipflops (R_i, R_j)) can be assigned weights depending on the probability of observing a hold time violation on them. If the probability of observing a hold-time violation on a path is high then the weight of the path is larger. This weight is also an indicator of the number of buffers required on the path to remove the violation. If these weights are denoted as $W_{i,j}$ (known constants) then an alternate objective function to maximize the total weight of all the paths is as follows. This objective ensures that very few short paths remain that need to be buffered to fix hold-time violations on them.

$$Max. \sum_{i=1}^{n} \sum_{j=1}^{n} W_{i,j}.x_{i,j} \tag{6.17}$$

Placement Dependency

If the local clock wire from flipflop R_j to flipflop R_i is too long then it often degrades the clock signal while simultaneously increasing the delay at the receiving flipflop. In some extreme cases, it is possible that the wire delays can be sufficiently large to force the place and route tool to not be able to meet the timing. Therefore care needs to be taken to ensure that the local clock wires are short. This can be done by considering the placement of the flipflops. An additional constraint can be imposed in above formulation where flipflop R_i can receive skewed local clock from flipflop R_j if the Manhattan distance between R_i and R_j is sufficiently small. The maximum value of this distance depends on the process technology and wire density of the design under consideration. Note that it is also possible for a placement tool to place the instances to be amenable for local clocking. However this idea is not explored in this dissertation.

6.3 Experimental Results

Experiments were conducted on several complex circuit blocks. These include a 64-bit signed integer multiplier, a 28-bit 4-tap FIR filter, and a 32-bit simplified MIPS processor. Unlike the filter and the MIPS processor, the 64-bit multiplier was pipelined, and provided an opportunity to explore throughput versus power by varying the number of pipeline stages. The number of stages was varied from $K = 0$ to $K = 5$. $K = 0$ denotes the circuit with only input and output flipflops. Also, beyond 5 stages, the conventional design showed no improvement in throughput while consuming greater energy.

All the circuits were synthesized by Cadence RTL compiler using a commercial 65nm LP cell library. They were then placed and routed using Cadence Encounter, at the respective peak frequency of the conventional design, i.e. the one with only conventional DFFs. Estimates of power were obtained using PrimeTime, with fully extracted post-layout netlists. Note that the delay on the local arcs are affected by the placement of the registers. Therefore, in our formulation the local arcs were restricted to the flipflops that were physically not too far apart.

Fig. 6.5 shows the percentage improvement in the ratio of the dynamic power to frequency, leakage and area using KVFFs and local clocking, over the conventional design. The results demonstrate that circuits with KVFFs providing skewed clocks are consistently better compared to the conventional globally clocked circuits. The best improvement is obtained for $K = 3$. The length of critical path in this circuit contains about 10-12 gates, which is typical of pipelined datapaths.

Additional circuit results for a 28-bit, 4-tap FIR filter and a 32-bit pipelined MIPS are shown in Table 6.2. These results further demonstrate the significant improvements due to KVFF and local clocking.

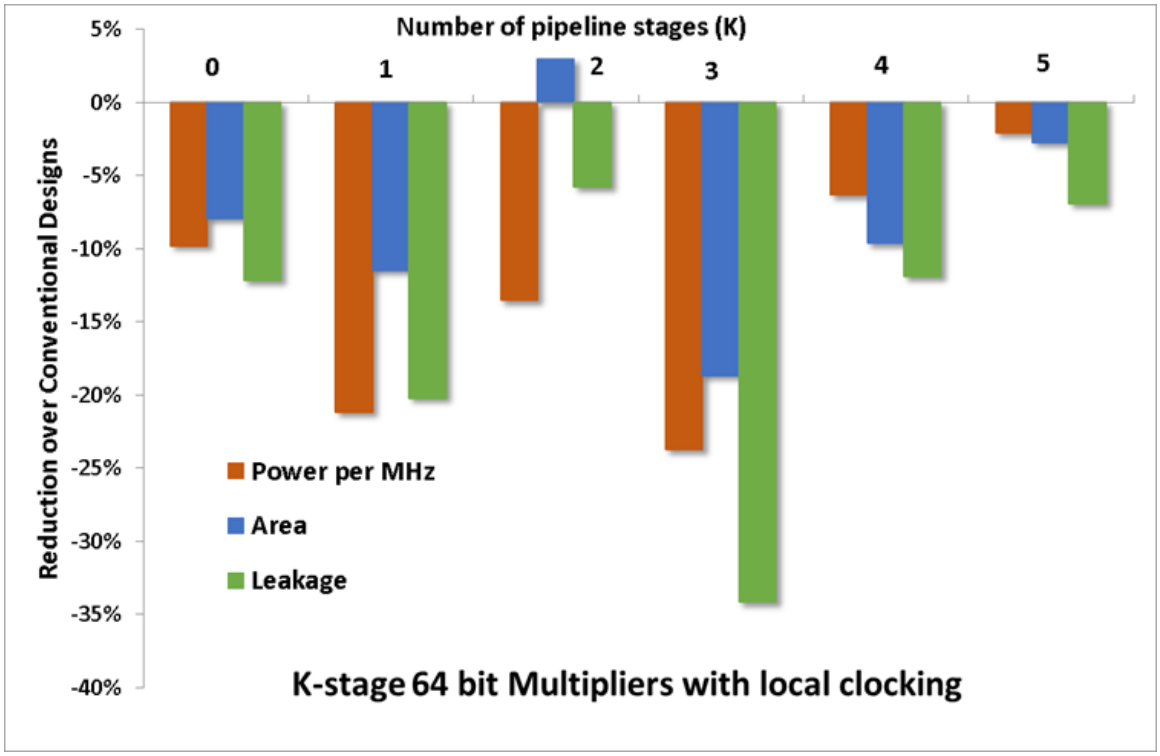


Figure 6.5: Improvements due to Local Clocking and KVFF Flipflops in Conventional CMOS Circuits

Table 6.2: Improvements due to Local Clocking Compared to the Conventional Globally Clocked CMOS Versions of 28-bit FIR Filter and 32-bit MIPS

	Power per MHz	Area	Leakage
Filter	22.5 %	12.4 %	32.1 %
MIPS	23.4 %	15.4 %	21.8 %

An important point to be noted is that local clocking is not always feasible. For example, three other circuits were tried viz. 2-stage 128-bit integer comparator, 32-bit sorter and 128-bit, 10-stage Advanced Encryption Standard (AES) circuit. local ILP solution did not introduce any skewed clock arcs in these circuits resulting in no improvements. local clocking generally results in significant improvements in circuits with unequal or unbalanced paths such as in Fig. 6.4. This is a natural consequence of the fact that no skews can be introduced on perfectly balanced paths without

violating setup time constraints.

The local clocking can further improve the power of hybrid circuits as well. Table 6.3 shows the dynamic power of 64-bit multipliers. The column labeled *Conventional* is the conventional globally clocked circuit and *hybrid* denotes the hybrid version of the corresponding circuit. The column names with suffix *LC* denote the locally clocked versions of the respective circuits and %Δ denotes the improvement of local clocking (*Hybrid LC*) over globally clocked *hybrid* circuits. It is seen that the hybrid versions of 64-bit multipliers exhibit additional reduction of about 3%-8% in power due to local clocking.

Table 6.3: Dynamic Power (mW) Improvements due to Local Clocking, Post-layout @ TT, 1.2V, 25C

K	Conventional	Conv. LC	Hybrid	Hybrid LC	%Δ	Frequency (MHz)
0	63.7	-	45.6	-	-	345
1	98.5	94.6	67.4	64.4	4.5	526
2	126.8	123.4	104.8	96.9	7.5	709
3	149.9	130.3	116.9	107.2	8.3	800
4	178	172.9	155.2	151.7	2.3	909
5	216.2	212.5	211.8	203.5	3.9	1053

6.3.1 Duty Cycle Considerations

It is desired that the duty cycle of the local clock should be about same as the input clock i.e. 50%. In order to ensure this is the case, a SPICE simulation was carried out where a KVFF with local clock drives the clock input of a regular flipflop. Fig. 6.6 provides the results of the spice simulation. We can see that the duty cycle of the local clock is about 52% which is close to 50%.

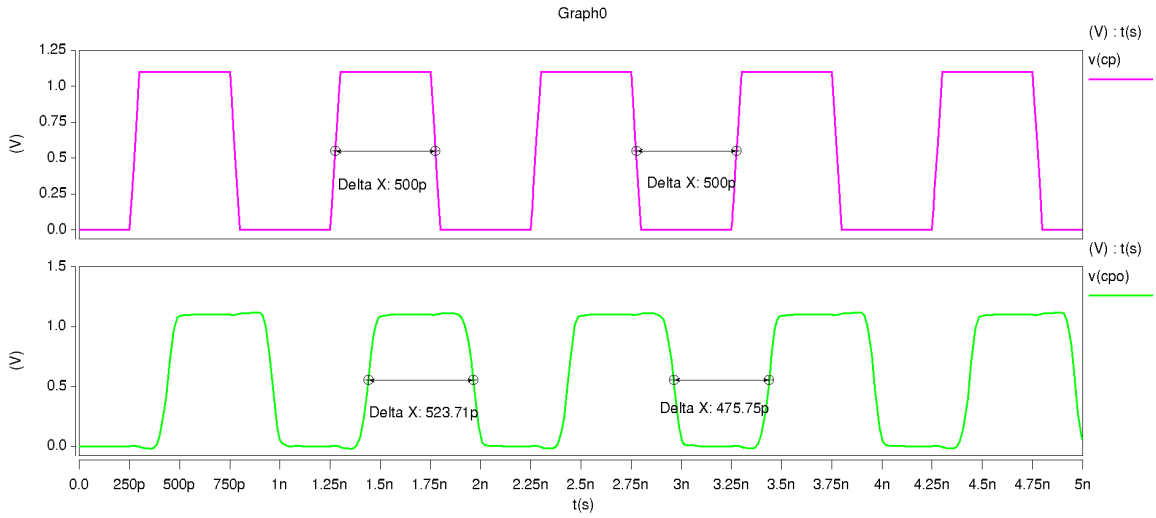


Figure 6.6: Spice Simulation Showing that the Duty Cycle of a Local Clock is Near 50%

6.3.2 Elimination of Hold Violations

The local clock can be used to eliminate hold-time violations when the local clock arc is as shown in Fig. 6.7(b). This is demonstrated by SPICE simulations of the circuits in Fig. 6.7(a) and (b). The global clock arrival uncertainty between the pairs of flipflops is simulated using RC delay. For small values of R and C, both circuits showed no hold violations. However, as the values of R and C (and hence resultant delay) were increased, the circuit in Fig. 6.7(a) exhibited hold violation confirmed by incorrect output waveforms. However no amount of RC delay on the local clock resulted in hold-time violations indicating elimination of hold-time violation.

6.3.3 Power-on Initialization

The flipflop Y in Fig. 6.4 provides local clock to the flipflop X and flipflop X feeds data input to flipflop Y. Therefore the state of the clock received by flipflop X and its output are interdependent. It is possible for the output of flipflop X to be in state where $D = \overline{D} = VDD/2$, especially when the circuit is powered up. When the

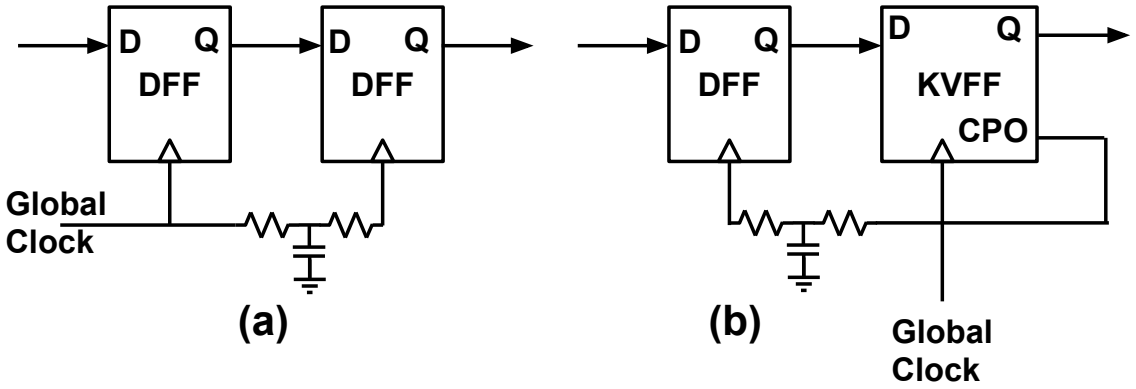


Figure 6.7: (a) Regular Flipflops Exhibit Hold Time Violation as RC Delay Increases (b) No Amount of RC Delay Induces A Hold Time Violation When Local Clocking is Used

clock goes low, the flipflop Y resets and nodes N1 and N2 (Fig. 6.3) are 1 forcing local clock signal CPO to be 0. However when external clock of flipflop Y rises while $D = \overline{D} = VDD/2$, then the evaluation might not finish (because of existence of a DC path since both the transistors M6 and M8 are ON) and can cause output clock CPO to never rise. Due to this, the flipflop X does not change its state since its clock is stuck at 0. This leads to a dead loop where no state change happens despite toggling of the external clock. Notice that this is only a problem after a chip power-up because once $D \neq \overline{D}$ then the feedback works perfectly as expected.

This issue can be fixed using a synchronous or asynchronous reset signal for flipflops X and Y. However in absence of such signals, a more economical solution is to size the transistors in the KVFF such that if $D = \overline{D} = VDD/2$, then it evaluates one way or another leading to a rise of the CPO signal.

A completely non-intrusive and exact solution for the initialization issue, in absence of synchronous/asynchronous reset signals, exists if the given flipflops are scan flipflops. All that is required is careful re-ordering of scan chain. For example, consider a chain of scan flipflops as in Fig. 6.8. In non-scan (regular) mode, the initialization issue exists for the pairs of flipflops (X1,Y1) and (X2,Y2). However in scan mode, we

can ensure that CPO output of both Y1 and Y2 flipflops rises (i.e. it is not stuck at 0) by connecting the TI (test-input) input of these flipflops as shown. This ensures that the flipflops Y1 and Y2 receive a proper test-input in each cycle and ensures that all flipflops are properly initialized. This simply requires the scan mode to be enabled for as many cycles as there are KVFF flipflops after the chip is powered on.

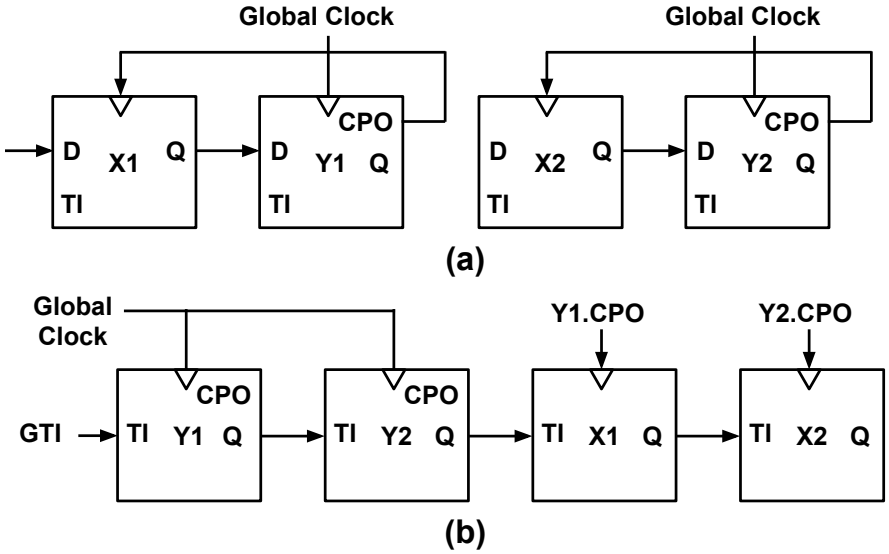


Figure 6.8: (a) Two Independent Pairs of Flipflops (X1,Y1) and (X2,Y2) in a Circuit with Possible Initialization Issue (b) Their Positions in a Single Scan Chain to Eliminate Metastability

Local clocking strategy, due to its dependency on knowledge of circuit timing, clock arrival timings, cell placements etc. is to be ideally run by an automated place & route tool. This type of clocking can be used as an additional step in circuit optimization after placement and clock tree synthesis and routing steps are finished. This step can help meet timing constraint easier i.e. without requiring significant cell resizing or buffering. Local clocking can also reclaim area under timing slack. It can even be used to *eliminate* hold violations on some short paths which might otherwise require large buffer area to fix. In practice, it is also possible to create multiple versions of local clock generating cells that provide a range of skews. The clock tree

optimization then can be further improved to take advantage of these multiple skews to best achieve some or all of the above objectives.

This chapter explored a new idea of producing skewed local clock that can be used to reduce the dynamic power of the circuits and to eliminate hold time violation on given short paths. Traditional approach to skew based optimization is to compute a skew schedule and synthesize a clock distribution tree that meets the schedule. The local clocking idea changes the traditional paradigm of skew based optimization from the synthesis of clock trees to an in-place optimization. The local clocking technique is additive to the existing circuit optimization techniques and therefore can be applied to any existing circuit to further improve its power or to eliminate hold time issues.

Chapter 7

FUTURE WORK

This chapter outlines some open problems and new ideas that can potentially lead to substantial improvements in the dynamic power and area of the sequential circuits. Certain novel sequential circuit architectures and methods that can be used to mitigate problems such as clock distribution tree power, reducing number of flipflops etc are also mentioned. Finally, implementations of asynchronous circuits using PNAND is also shown.

Some of these methods and ideas specifically apply to the large pipelined data-path circuits such adders, multipliers, and other arithmetic circuits etc. Indeed the data-path circuits consume majority of power and are the most critical any general purpose sequential circuits that perform data processing.

7.1 Retiming

7.1.1 Retiming for Minimum Clock Period

The problem of retiming using flipflops was first studied by (Leiserson and Saxe, 1991). Retiming involves either moving the flipflops or introducing a skew to each flipflop (Rahul B. Deokar, 1995) such that both the setup and the hold time constraints are satisfied while clock period is minimized. In these retiming problems, the logic itself is not modified. However when PNANDs are introduced at a node then they can absorb some of the logic and speed up the circuit further. This is different from a conventional flipflop that does not absorb logic at all. *The most important point is that the setup time of PNAND and its output delay doesn't depend on the*

amount of absorbed logic. This is the main difference between conventional logic in flipflop designs and PNAND.

The presence of logic in a flipflop is further complicated by the fact that PNANDs introduced at different nodes may absorb different amounts of logic. This is because PNANDs specifically implement threshold logic functions. Therefore different nodes may have different sizes of logic functions that are threshold by nature.

An intuitive solution to the problem of retiming for minimum delay is to retime using conventional flipflops followed by hybridization. However this solution will not be optimal. Fig. 7.1 shows a small example where a simple circuit is retimed for minimum delay. The positions of retimed flipflops are shown by small dark rectangles. Assuming each gate delay is 1 ns, the clock period is 2ns. Even if we replace the flipflops and feeding logic by the PNANDs, the clock period is still 2ns (due to AND and XNOR gates).

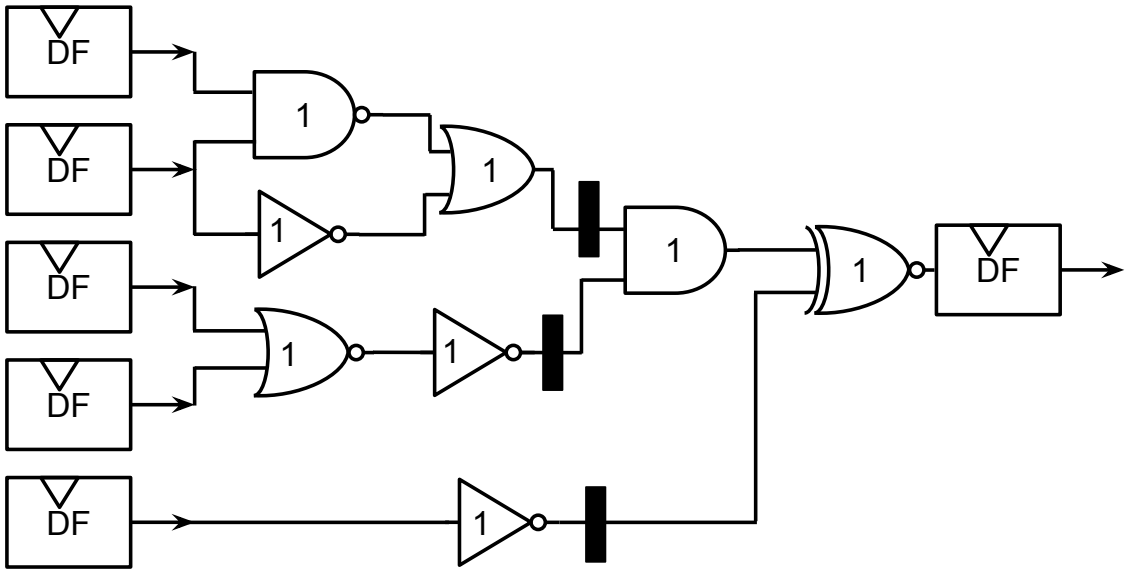


Figure 7.1: Retiming for Minimum Delay using Conventional Flipflops Leads to 2ns Clock Period

On the other hand, Fig. 7.2 shows that there exists a different retiming solution

with 1ns of clock period. Therefore the retiming with PNANDs is a new problem and requires an in-depth investigation.

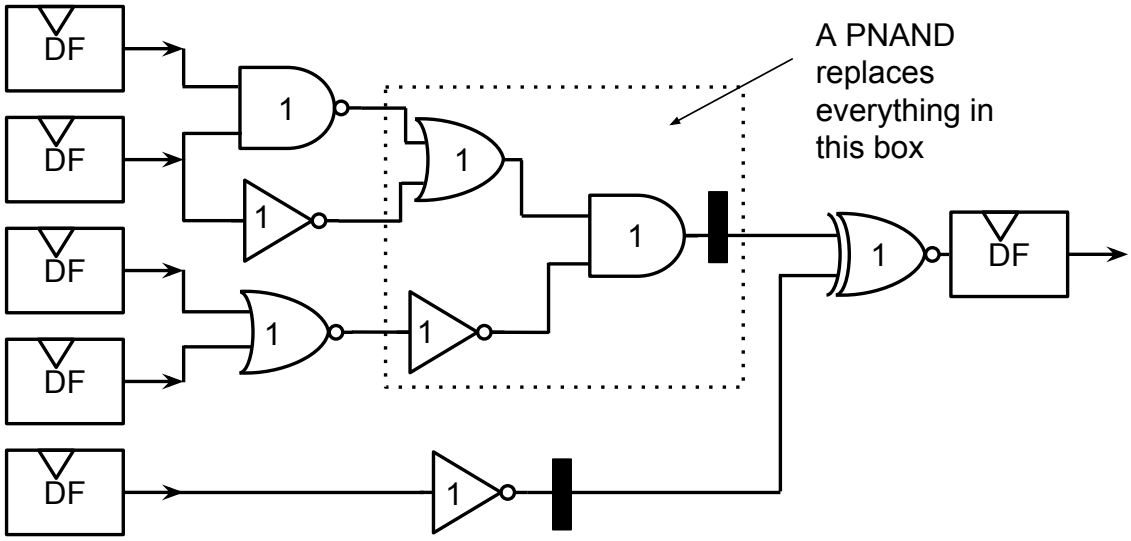


Figure 7.2: Retiming for Minimum Delay using PNANDs Leads to Smaller (1ns) Clock Period

7.1.2 Retiming for Minimum Area

Retiming for minimizing area involves minimizing the number of sequential elements (flipflops) without changing the clock period. This problem is addressed in (Leiserson and Saxe, 1991). However when the conventional flipflops are used, there is less room for reducing the number of sequential elements. But with PNANDs, there is a larger space to move PNANDs as they can absorb logic to make-up for the degradation in critical path delay. Fig. 4.1(b) and 4.1(c) show how a single PNAND can potentially replace multiple flipflops thereby reducing the number of sequential elements. Therefore it is possible to reduce number of flipflops using PNANDs when the logic cones are not rooted at flipflops.

7.2 Novel Clock Distribution

This section proposes another new clocking strategy inspired from the local clocking idea presented in Chapter 6. The objective of this architecture is to minimize the load on global clock tree and eliminate hold time violations on most short paths. The architecture of this clocking mechanism is depicted using a small circuit example in Fig. 7.3

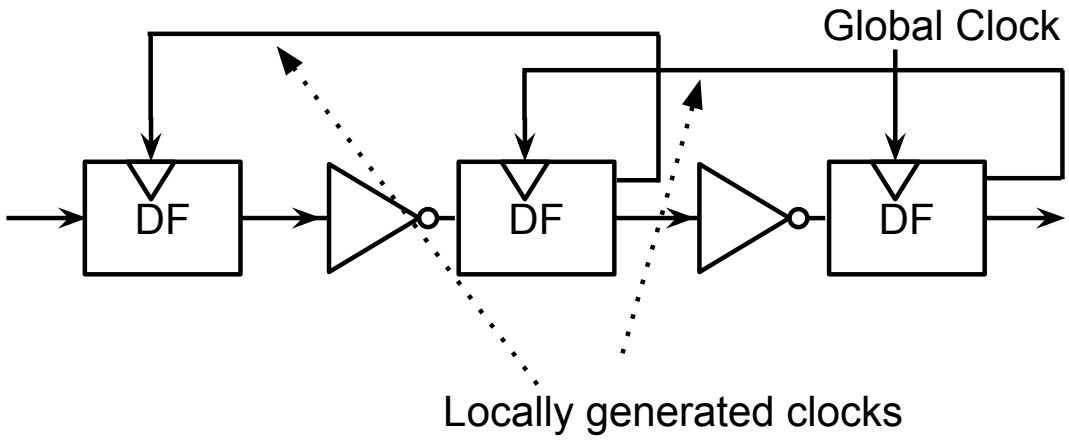


Figure 7.3: A Novel Clocking Distribution using Local Clocks

The chain of inverters with flipflops depict a datapath pipeline circuit in an abstracted way. Notice how global clock only triggers only the final stage flipflops. The output flipflops upon consuming their input data, trigger previous stage for new data which upon latching their input, in turn, trigger their previous stage. On each rising edge of the clock, every k'th stage latches its input data data and *requests* a new data from the (k-1) th stage. Therefore the data is pulled towards outputs and the clocking strategy is known as "pull-data clocking". The idea behind the architecture is to distribute a single global clock into multiple local clocks. An important point to

note that this circuit is still a synchronous circuit and therefore can be analyzed for timing using existing tools.

7.3 Asynchronous Circuit Design

7.3.1 Dual Rail Circuits

Unlike conventional synchronous circuits, asynchronous circuits do not have a central synchronizing clock signal. The computing elements/blocks communicate with each other using the handshake protocols (Sparso and Furber, 2002). An alternative to the handshake protocols is to use dual rail circuits which can detect validity and value of Boolean signals represented as two signals. This section discusses only at an architectural level how to design dual rail circuits using PNANDs.

A dual-rail circuit operates on logical signals represented using two physical signals/wires. The main requirement of the dual-rail representation is the necessity to represent an invalid state along with Boolean values 0 and 1. As such, a signal x represented as two rails $x.f$ and $x.t$ has the following meaning as shown in Table 7.1. *Note* : Some dual-rail circuits use (0,0) as an invalid state.

$x.f$	$x.t$	x
1	1	invalid
1	0	0
0	1	1
0	0	not used

Table 7.1: Logic Values of Signal x Represented using Dual Signals $x.f$ and $x.t$

The functional behavior of a dual-rail logic gate is -

1. When all the inputs are valid, the output must be the given function of inputs.

2. When all the inputs are invalid, the output must be invalid.

3. When some of the inputs are invalid and others are valid, the output must retain the last state (last valid or invalid state)

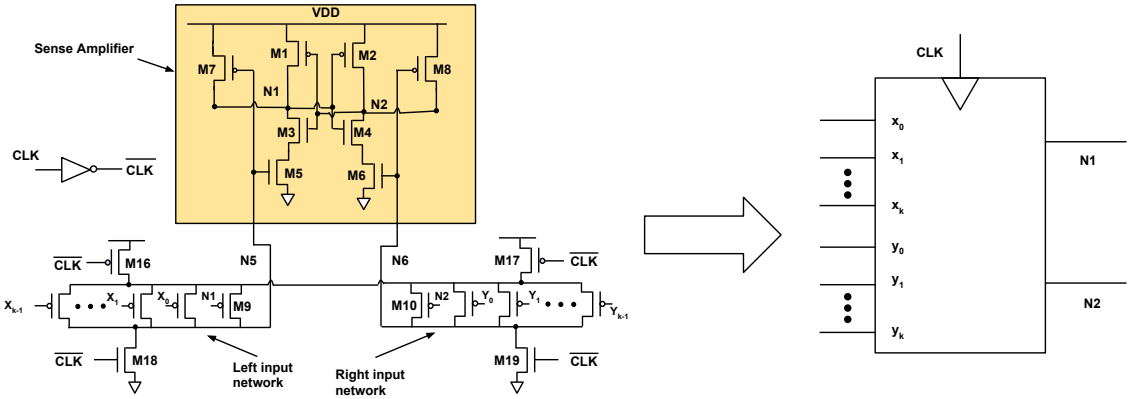


Figure 7.4: PNAND Circuit Abstraction

The original PNAND circuit can abstracted as shown in Fig. 7.4. The simple function of such an abstracted circuit is as follows.

1. When CLK = 0, PNAND is in reset phase and N1 = N2 = 1.

2. When CLK goes from 0 to 1, N1 = 0 and N2 = 1 if and only if the $\sum x_i > \sum y_i$. If $\sum x_i < \sum y_i$, then N1 = 1 and N2 = 0.

3. For all other states and transitions at inputs, N1 and N2 maintain their last state.

Fig. 7.5 shows a PNAND-5 implementing a dual rail 3 out of 5 majority function. The inputs (a,b,c,d,e) are all dual-rail and so is the output z. An n-input muller-C element raises its output to 1 if and only if all of its inputs are 1. Its output is 0 if all of its inputs are 0. For any other input combination, the muller-C element maintains its last state. The circuit in Fig. 7.5 functions as follows.

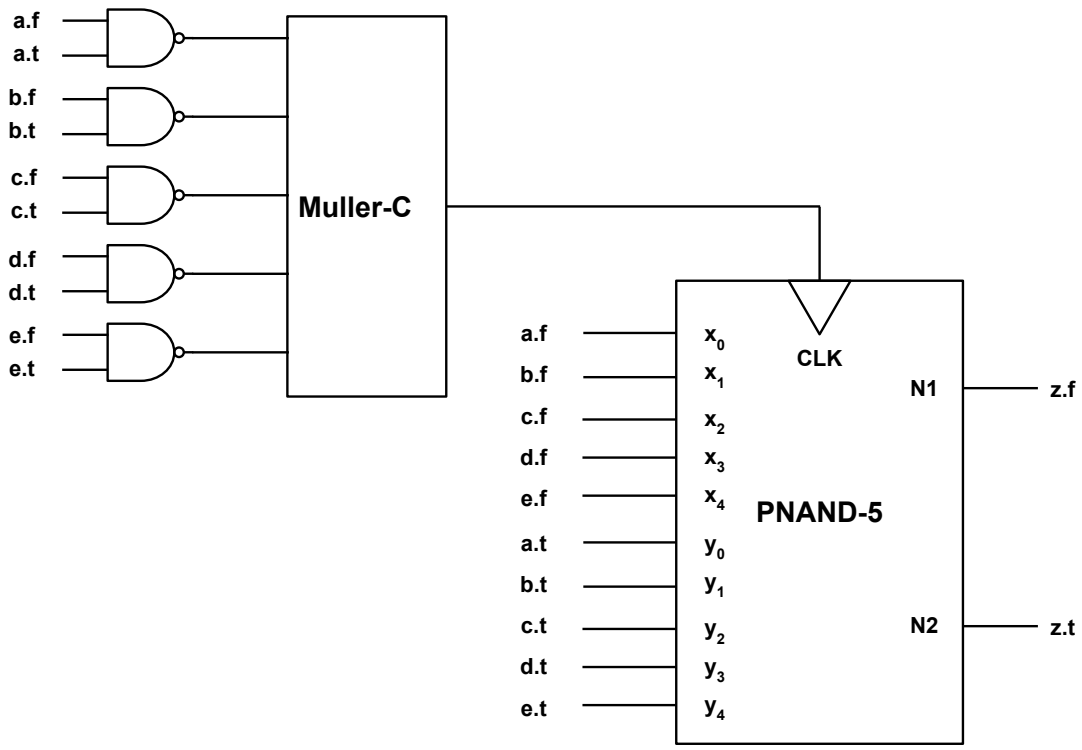


Figure 7.5: Implementing Dual-Rail 3 out of 5 Majority Function using PNAND-5

1. When all dual-rail inputs are valid i.e. (1,0) or (0,1), the outputs of the NAND gates are 1, due to which output of the muller-C gate is 1. The output of muller-C acts as a CLK signal for PNAND and hence it evaluates majority function and stores the computed value.

2. When all dual-rail inputs are invalid (1,1), the outputs of all the NAND gates 0, due to which output of muller-C is 0. Since CLK of the PNAND is now 0, the PNAND resets and nodes N1 and N2 are pulled to 1 making dual-rail output as invalid.

3. When some dual rail inputs are invalid, some outputs of the NAND gates are 0

but not all, then the muller-C element maintains the last state. Hence PNAND maintains the last state (either computed or reset).

Note that the dual rail circuits need an acknowledgment signal to notify the completion of computation. This signal is the completion detection signal $D = \overline{(N1 \,\&\, N2)}$.

7.3.2 Comparison with CMOS based Asynchronous Implementations

Dual-rail circuits can be designed using conventional static logic gates by defining each output rail as a single-rail Boolean function of all the input rails. For example, a CMOS dual-rail AND gate is defined in Fig. 7.6.

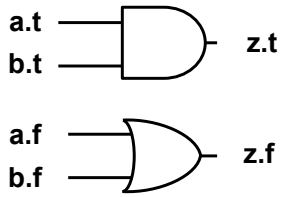


Figure 7.6: A Dual Rail AND Gate using CMOS Gates

Note that this dual-rail AND gate doesn't retain its output when the inputs are partially invalid. In order to satisfy this condition, dual-rail latches are needed as shown in Fig. 7.7.

The functionality of the dual rail latches in Fig. 7.7 is as follows.

- If EN = 0, N1 = N2 = 1.
- If EN = 1, N1 = I0 and N2 = T0.

When both inputs are valid, the muller-C produces a 1 at its output and the latches are enabled causing output rails to evaluate the input. When both the inputs

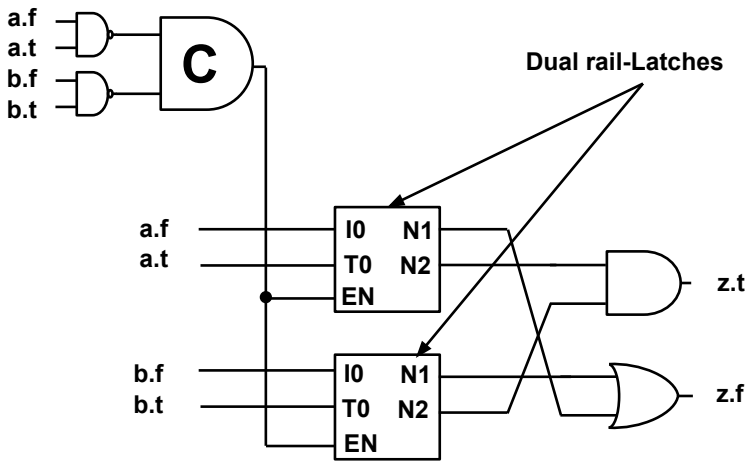


Figure 7.7: A Dual Rail AND gate using CMOS Gates and Latches

are invalid, muller-C produces a 0 at its output and the latches produces invalid inputs to the static gates leading to invalid output z. When one input is valid and not another, the muller-C, the latches and the output z maintain their last state.

We can see that a significant amount of logic is needed to produce a delay-insensitive dual rail static 2-input AND gate. Another way to create a purely static latch-less implementation is using Delay Insensitive Minterm Synthesis using muller-C (DIMS)(Sparso and Furber, 2002), where a k-input DI gate requires 2^k as many k input muller-C gates. The outputs of these gates are combined using two static OR gates to generate both rails of the output function. Fig. 7.8 shows a 2 input AND gate using DIMS idea.

Similar to the PNAND implementation, the dynamic logic dual-rail AND gate is defined as shown in Fig. 7.9.

The circuit in Fig. 7.9 operates as follows.

1. When both the inputs are invalid, EN = 0 and outputs are precharged to 1 i.e. output is also invalid.

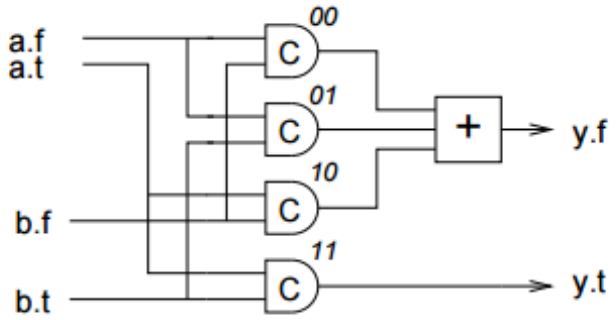


Figure 7.8: A DIM Synthesis of Two Input AND Gate

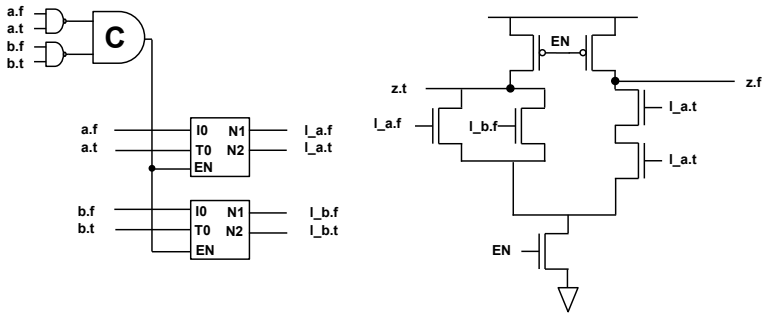


Figure 7.9: A Dynamic Dual Rail AND Gate

2. When EN = 1, depending on the state, one of the nodes discharges to produce a complemented dual-rail output. In general, the dual rail dynamic gates are defined similar to CMOS style gates. Each rail is defined as individual function and its NMOS network appears on the two sides of the dynamic gate.

In terms of *operation* as a function of their inputs, all three implementations discussed above are identical. We can connect these gates in any fashion to produce a netlist or a block realizing a larger function such as a multiplier. An important thing to note here is that none of these gates have the output latch boundaries i.e. output is stored only as long as inputs do not change to a different valid state or an

invalid state. It is not stored until it has been consumed by the fanouts. Therefore conceptually, these gates are *combinational* gates in asynchronous domain.

We can compare the implementation of dual-rail CMOS/dynamic logic gates with the PNAND based implementation in terms of number of transistors or *area* required to implement a given function. Table 7.2 shows the number of transistors required to implement few complex functions.

Function	*Static (with latches)*	*Dynamic (with latches)*	*PNAND*
Full Adder	102	84	72
2/3 majority	78	71	46
3/5 majority	214	157	64

Table 7.2: Number of Transistors Required to Implement Dual-rail DI Implementations

It can be seen from Table 7.2 that the number of transistors required for PNAND implementations are much less in many cases especially for large majorities.

The performance of these circuits depends on the specific implementation of a function (or how it is synthesized) and transistor sizing. However we note that the PNAND implementation requires a constant number of stages/levels to compute a threshold function. Static implementations may use large multi-level networks and dynamic implementations will end up with complex series-parallel stacks of NMOSs; both of which are expected to be slower than the PNAND implementations.

Note : PNANDs are "inherently dual-rail" unlike static or dynamic implementations where each rail is defined explicitly as an independent function. PNAND requires dual-rail inputs (for signal assignment) and always produces dual-rail output signals (N1,N2) as a result of evaluation.

7.3.3 Relaxing Delay Insensitivity

The delay insensitivity criteria can be relaxed to obtain a trade-off between power and area. For the PNAND gate shown in Fig. 7.5, note that out of all the input signals to the gate, one of the signals is always the latest depending on the circuit topology. Therefore only such a latest signal changing to a valid state could trigger the computation. Note that under any assumption about the arrival times of the signals, the delay insensitivity no longer holds.

Fig. 7.10 (a) shows the delay insensitive PNAND based circuit. This circuit computes only when all its inputs are valid. Fig. 7.10 (b) shows the same circuit triggered only by the latest signal c (say). Note that circuit (b) is delay sensitive and therefore a configurable delay element is added to ensure it works post manufacture. Due to removal of the NAND gates and the muller-C element, the circuit (b) is faster, smaller and lower power compared to the circuit (a).

In order to obtain a smoother trade-off between delay-insensitivity and power, "k" latest signals can be used instead of all the signals. When k = 1, we get the circuit in Fig. 7.10 (b) and when k = n i.e. all inputs are considered the circuit becomes completely delay-insensitive.

7.3.4 Comparison with Null Convention Logic

Null Convention Logic (Fant and Brandt, 1996) is another well-known design methodology for delay insensitive threshold logic gates. This section explains the architecture of these gates and a primitive comparison of these gates with PNANDs in terms of number of transistors.

The logic function of an NCL "k out of n" majority gate is similar to a muller-C element and is described below.

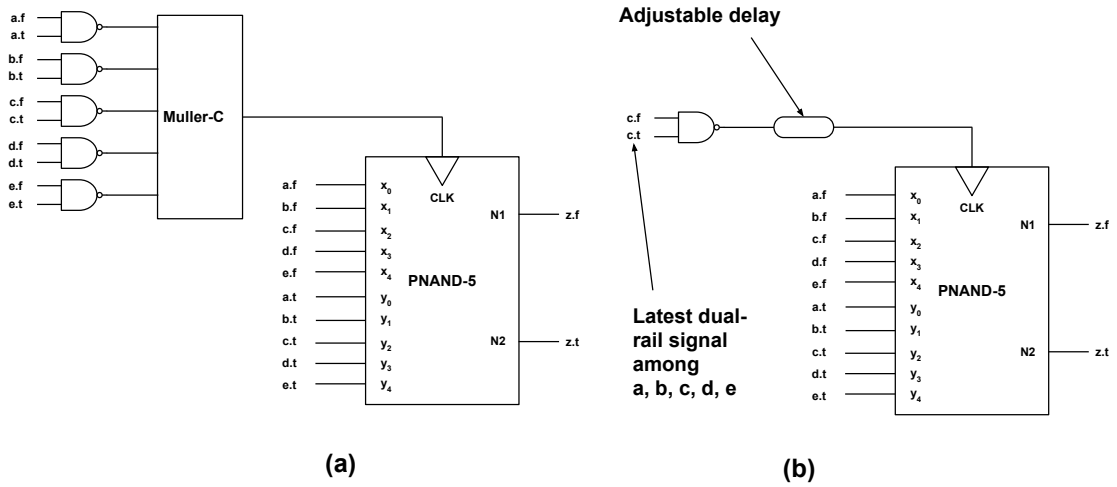


Figure 7.10: (a) Delay-insensitive PNAND Gate (b) Delay-sensitive Gate Triggered Only by the Latest Arriving Signal

- If k or more inputs are 1, output is 1
- If (and only if) all inputs are 0, output is 0
- For any other combination, the output of the gate is the last state.

In essence, an NCL gate generalizes the functionality of a muller-C element. The conventional n-input muller-C element is same as an "n out of n" majority NCL gate. Fig. 7.11 shows the architecture of a static 2 out of 3 majority NCL gate (Fant and Brandt, 1996).

The circuit in Fig. 7.11 works as per description above. For example, when all the inputs are 0, the node ZN is charged to 1 and output Z falls. Since none of the NMOS transistors in a box labeled "Hold0" are ON, the node ZN rises without contention. When any 2 out of three inputs (A,B,C) are ON, the node ZN is pulled low through one of the branches (again without contention) and the output Z rises to 1. For any other state of the inputs, transistors in both boxes labeled Hold1 and Hold0 are ON and the node ZN and therefore output Z maintains its state.

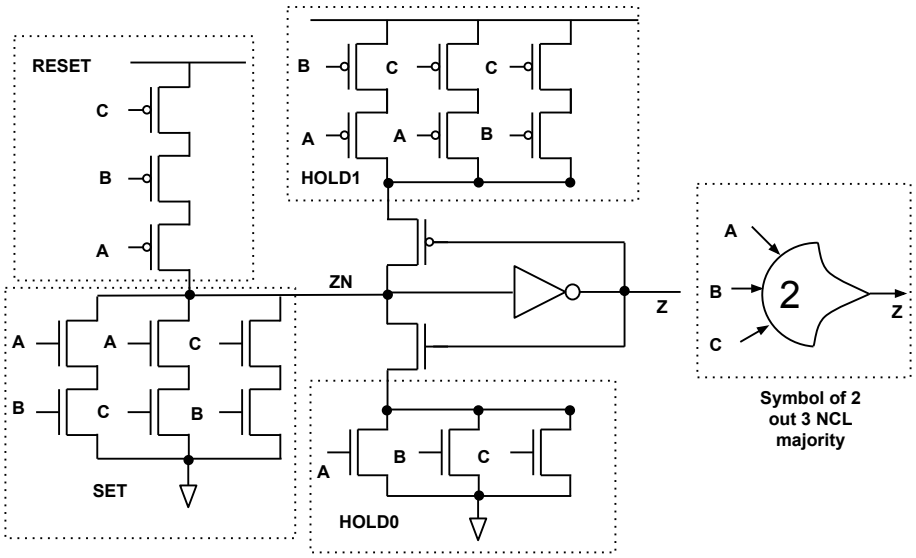


Figure 7.11: 2 out of 3 Static NCL Majority Gate

NCL gates have an inbuilt latch (notice the feedback in Fig. 7.11 where output Z feeds back into the circuit). NCL majority gates can be connected in certain fashion to realize dual-rail delay insensitive primitives. For example, Fig. 7.12 shows a half adder constructed using NCL majority gates. *An important distinction to be made for NCL gate is that the gate architecture assumes (0,0) as an invalid state of the dual-rail input.*

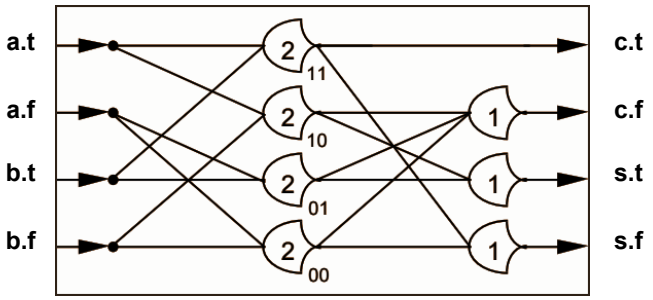


Figure 7.12: Dual Rail Delay Insensitive NCL Half Adder

The circuit in Fig. 7.12 works as follows.

1. When all the inputs are invalid i.e. (0,0), all the majorities produce 0 and output s and c are invalid i.e. (0,0) each.

2. When all the inputs are valid, the circuit produces sum and carry indicating Boolean addition of its inputs. For example, assume A.t = B.f = 0 and A.f = B.t = 1. Therefore logically input A = 0 and B = 1. As such, output of 2 out of 3 majority (marked as "01") is 1. Therefore c.t = 0 and c.f = 1 i.e. carry output is 0. Similarly s.t = 1 and s.f = 0 indicating sum = 1.

The synthesis of an arbitrary n-input function can be done by synthesizing the function using majorities. The positive rail (t) and negative rail (f) are computed independently similar to the static or the dynamic circuits. NCL based synthesis improves over DIMS synthesis by replacing static n out of n muller-C elements with generalized *muller-Majorities* aka NCL gates.

In order to create a logic network (such as a datapath pipeline) using delay-insensitive primitives (aka primitives with completion detection), each primitive must hold its output until it has been consumed by all of its fanouts. This is achieved by primitives that behave like a state machine whose next state depends on the states of its fanins and fanouts. Therefore each of the dual-rail gates described in Question 2 (above) must be augmented with additional logic that converts it into such state machine. Before delving into the implementation specifics, we first describe how such a state-machine primitive behaves.

Generalized DI primitive

Each primitive has fanin gates that produce new data for it and has fanouts that consume the output data from it and it operates as a state machine with two states (compute and reset). A primitive is known to be in compute state when it has valid

dual-rail output (1,0) or (0,1) and in reset state when the output is invalid (1,1). Therefore the output of the primitive represents its state.

The primitive knows about the states of its fanins by looking at its input rails i.e. if input rails are all (0,1) or (1,0) fanins are in compute state. If they are (1,1) the primitive deduces that the fanins are in reset state. The primitive knows about the states of its fanouts by looking at acknowledge (ACK) signals from fanouts. If the acknowledge signal is high, the corresponding fanout gate is in compute state and if it is low, it is in reset state. A generalized form of such state-machine is shown in Fig. 7.13 (Sparso and Furber, 2002).

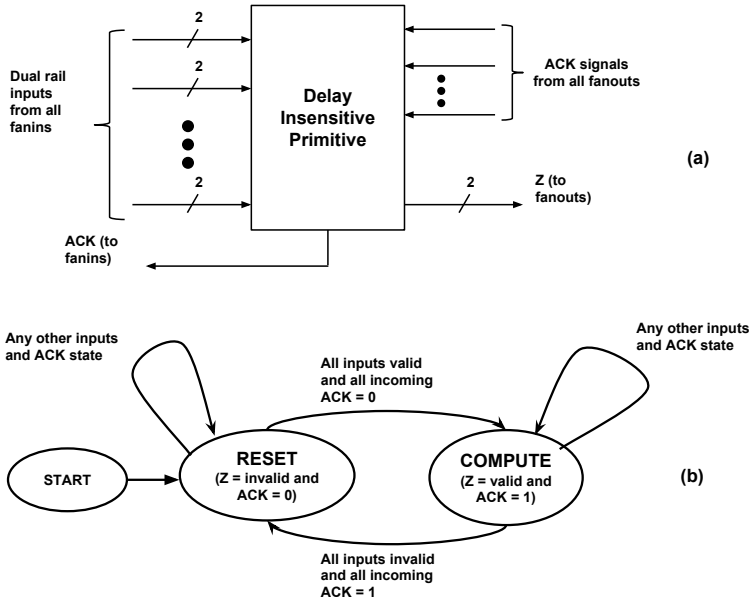


Figure 7.13: (a) Generalized Primitive Used to Construct DI Netlist (b) State Transitions for the Primitive

1. Each primitive/gate starts in a reset state when powered up, independent of the state of other gates. This is usually achieved using a global reset signal which is de-asserted once all the gates are in reset state.

2. A primitive enters a compute state (and computes a valid output token) if and only if all its fanins are in compute state and all its fanouts are in reset state.

3. The primitive enters a reset state if and only if all its fanins are in reset state and all its fanouts are in compute state.

4. For any other state of fanins and fanouts, the primitive maintains its current state.

The DI primitives can be connected together to create a network that behaves as an asynchronous delay-insensitive pipeline. One can imagine waves of successive computations (progressing through the network in a topological fashion) in which each gate bobs up (compute state) and down (reset state). The delay of individual gates is immaterial for the functioning of this circuit. For example, if certain gate F doesn't change it's state for a long time then it impacts state of all predecessors and successors who will wait on the state change of the gate F. *Note*: It is not possible to implement DI primitive using conventional synchronous state machines which require a global clock to trigger the state change.

7.3.5 Implementing DI Primitive using PNANDs

Implementation of such state machine using PNANDs is shown in Fig. 7.14. The circuit operates as follows.

1. The circuit has a global reset signal (not shown) that resets the PNAND when powered up. The global reset is de-asserted when all the gates are in the reset state.

2. When inputs of this gate are first asserted valid, the outputs of NAND gates and muller-C1 are 1. Note that all the fanouts are in reset state and incoming

acknowledgement signals are low. The inverted acknowledgments are therefore high and output of muller-C2 is also 1. This causes the two input muller-C element to produce 1 which triggers PNAND and the gate enters compute state. PNAND asserts its dual-rail output valid which is read by its fanouts. Also in the compute state, the ACK signal to fanins is 1.

3. As a result of acknowledgement to fanins, the fanins enter reset state (at some point in time). However gate maintains its asserted output until all fanouts have acknowledged it. When all fanouts acknowledge this gate by raising incoming ACK signals, inputs to all the muller-C elements are 0 which produces CLK = 0 for PNAND and as a result the gate enters reset state by de-asserting its output. For any other intermediate state/transitions/glitches of inputs, the PNAND primitive maintains its current state.

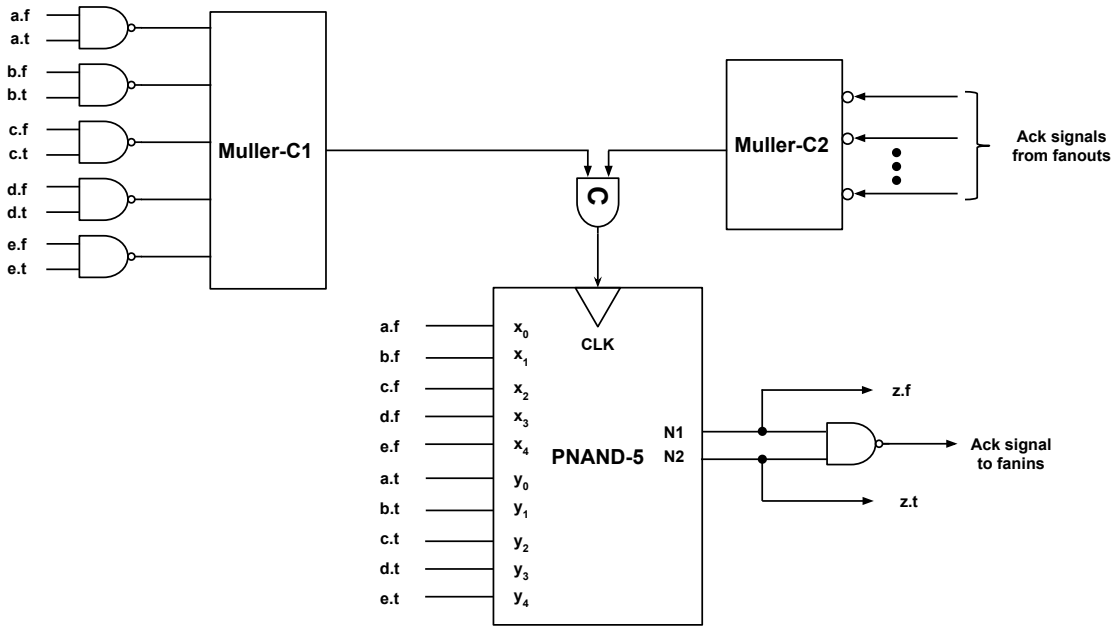


Figure 7.14: Implementation of Generalized DI Primitive using PNAND

7.3.6 Implementing DI Primitive using NCL Majorities

Fig. 7.15 shows an NCL dual-rail gate (such as in Fig. 7.12) augmented with additional NCL 2 out of 2 majorities to implement generalized primitive from Fig. 7.13. The circuit works as follows. *Recall that invalid state for any dual-rail input to the NCL gates is (0,0)* -

1. Initially data inputs to this gate are invalid i.e. (0,0). Therefore all four 2 out of 2 majorities produce a 0 at outputs. Hence the inputs to dual-rail NCL gate itself are all (0,0) and it sets output z = (0,0) which is invalid output. At the same time, all fanouts are also in reset state leading to inverted ACK signals to k out of k majority being 1. Therefore "control" input is 1 in the reset state of the gate. Similarly, inputs to the 2 out of 4 majority at the bottom are 0 and therefore it produces ACK = 0. Therefore in reset state output is invalid and ACK = 0.

2. When all data inputs from fanins are valid (note that "control" signal is 1), the inputs are relayed to the dual-rail NCL gate which computes valid output z. At the same time, 2 of the 4 inputs to the 2 out of 4 majority at the bottom are 1 and therefore it produces ACK = 1. Therefore in compute state ACK = 1 and output z is valid

A potential hazard situation in case of PNAND primitive occurs when the internally generated CLK signal of PNAND arrives before the (t) or (f) rails of the inputs feeding its data inputs. However the chance of this occurring is extremely rare due to the delay in generating the CLK signal. Moreover, this is adjustable *inside* the primitive. Outside the primitive, unbounded wire delays can exist and the network will function correctly.

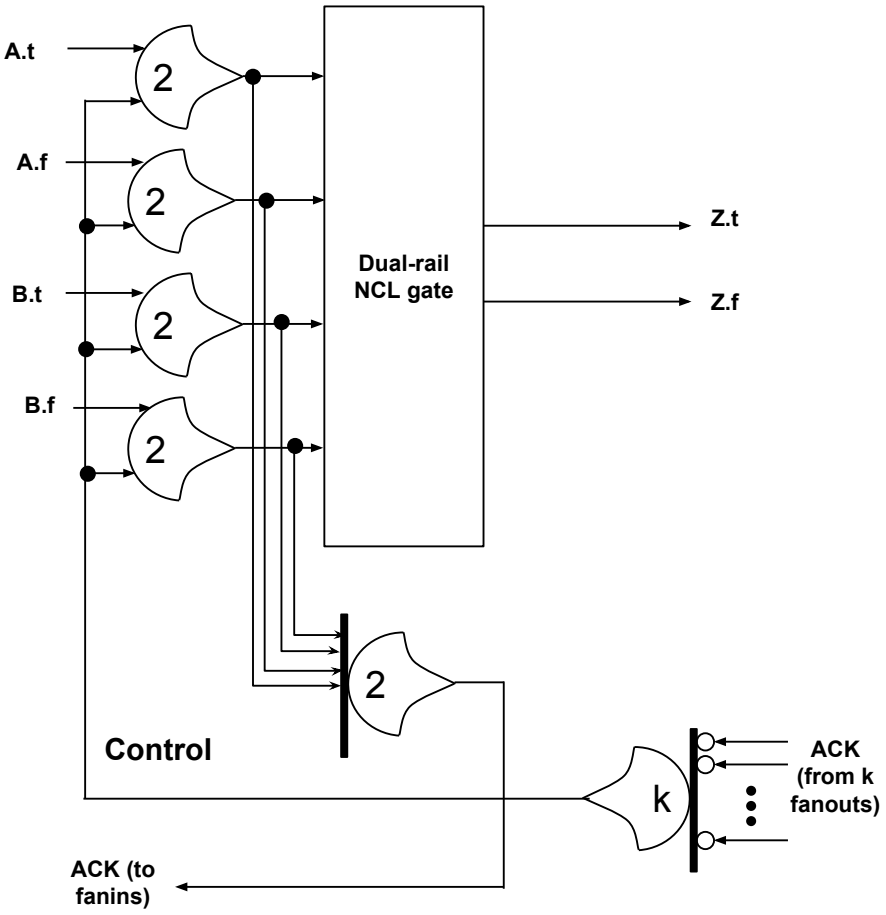


Figure 7.15: Implementation of Generalized DI Primitive using NCL Majority Gates

A potential hazard situation in the network of of NCL DI primitive is described as follows. The DI primitive computes the ACK signal *independent* of the state of the gate and *prior* to the state change of the gate. The state of the gate is determined by the output of the gate which is either valid (compute) or invalid (reset). Since ACK is computed and sent to the fanins "before" the state of the gate changes, it is possible that the fanin primitive might invalidate the data bit before dual-rail NCL gate computes the output z. This hazard situation occurs *between* the two interconnected DI primitives which can make NCL networks fail under certain conditions.

We can compare the resources required to implement DI circuits using PNAND vs NCL in terms of number of transistors utilized by the primitives. *Note : The majority function is implemented using a pull-up, pull-down network in NCL majority gates.* Therefore larger majorities such as 2 out 4 majority or 3 out of 5 majority are not only slower, but also consume significantly higher area and power due to static implementations. This is the main difference between implementing dual rail delay-insensitive logic gates using NCL majorities and PNANDs. Table 7.3 shows the number of transistors required to implement some complex dual-rail functions as a single primitive. *Note* : A dual-rail 2 out of 3 majority function is not same as the NCL majority gate from Fig. 7.11 which computes a 2 out of 3 majority of its single-rail inputs. The building blocks of dual-rail primitives using NCL technology are NCL majority gates.

Function	*NCL*	*PNAND*
Half Adder	124	64
Full Adder	176	72
2/3 majority	44	48
3/5 majority	168	68
4/7 majority	596	88
5/9 majority	2564	108

Table 7.3: Number of Transistors Required to Implement Dual-rail Function as a Single Gate

7.4 Novel Scanning Mechanism

This section outlines the design of a novel scanning mechanism along with the modifications in the single and multi-input differential mode flipflops that are required

to implement the proposed scheme. Scanning the data serially in and out of the chip is indispensable for functional testing of integrated circuits post manufacture. The main advantage of this proposed scanning mechanism is elimination of hold time violation in the scan-chain during test/scan mode.

7.4.1 Flipflop Architecture

This section provides the functional description of a flipflop that is necessary in the proposed scan-mechanism. A possible architectural implementation of such a flipflop is provided in Section 7.4.5. Fig. 7.16 shows the inputs and outputs required for usage of the new scan mechanism.

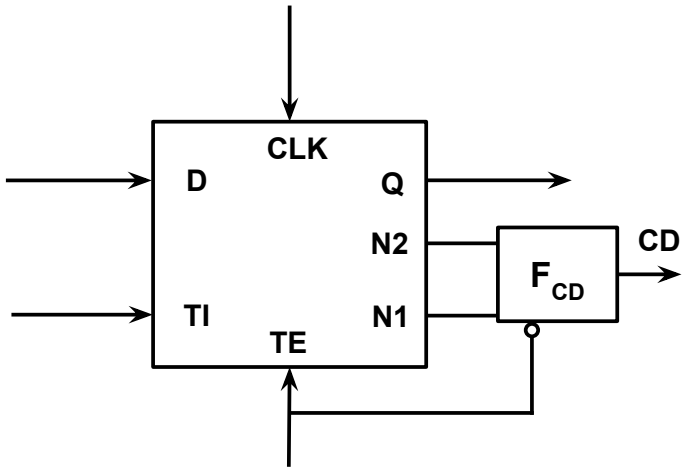


Figure 7.16: Flipflop Architecture Required for the Proposed Scan Mechanism

The description and function of each of the input and outputs signals is as follows:

1. *Input signal D* is same as data input signal D typically found in master-slave D-type flipflops.

2. *Input signal TI* is a test input. Typically the input signal TI is driven by a flipflop that precedes the given flipflop in a scan chain.

3. *Input signal CLK* is a synchronous clock for the normal (as opposed to test) mode operation. Note that this clock is NOT a scan clock.

4. *Input signal TE* acts as trigger for the test mode. It is necessary that CLK must be 0 whenever TE is changed. Similarly TE must be 0 when CLK is operated.

5. *Output signals N1 and N2* have the following functionality.

   (a) With TE = 0 and CLK = 0, N1 = 1 and N2 = 1.

   (b) With TE = 0, when CLK changes from 0 to 1, N1 = $\overline{D}$ and N2 = D.

   (c) With CLK = 0, when TE changes from 0 to 1, N1 = $\overline{TI}$ and N2 = TI.

   (d) TE = 1 and CLK = 1, leads to race condition and values of outputs N1 and N2 are unpredictable.

   (e) For all other inputs and transitions not described above, nodes N1 and N2 must retain their last state.

   (f) *Output signal Q* is defined as a function of the signals N1 and N2. This function is defined by the following truth table.

N1	*N2*	*Q*
1	1	last state
1	0	0
0	1	1

Table 7.4: Logic Values of Output Q

   (g) The combination N1 = N2 = 0 is illegal and will result in a race condition leading to unpredictable state of output Q.

The functional block F_{CD} produces another output signal called CD which stands for completion detection. This block takes inputs TE, N1 and N2. Its Boolean function is described as follows.

$$CD = TE \wedge (\overline{N1} \vee \overline{N2})$$

where '$\wedge$' is a Boolean AND operation and '$\vee$' is a Boolean OR operation. An implementation of this function is shown in Fig. 7.17.

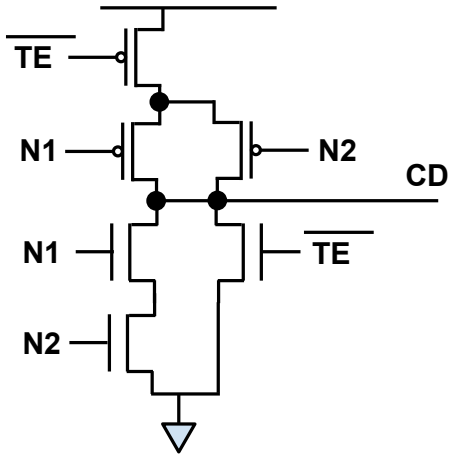


Figure 7.17: Implementation of Function F_{CD} from Fig. 7.16

7.4.2 Proposed Scan Architecture

Given a set of flipflops that satisfy above functionality, they can be connected in a chain as shown in Fig. 7.18. The D-inputs of each of the flipflop are driven by the logic which is not shown.

7.4.3 Scan Chain Operation

First we describe the purpose of each signal followed by how to operate each signal from Fig. 7.18 in order to scan-in and scan-out data along the chain.

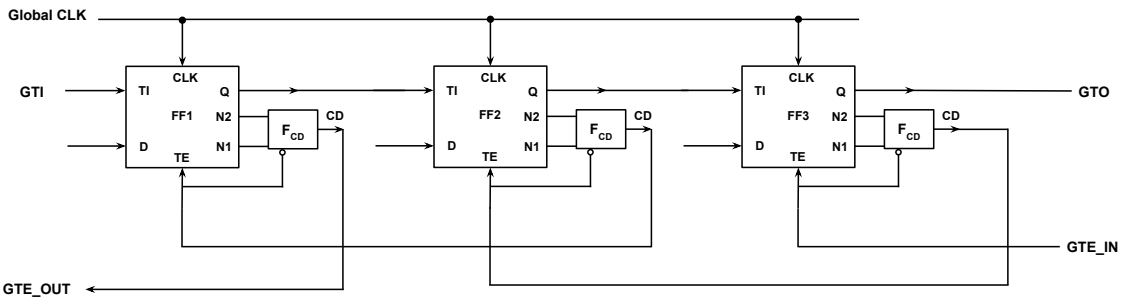


Figure 7.18: Proposed Connections to Create Scan-Chain using Flipflop from Fig. 7.16

Signal Description

1. *Input signal GTI* is Global Test Input signal. This signal is directly controlled by a chip-wide primary input and is an entry point of the scan chain.

2. *Input signal GTE_IN* is Global Test Enable signal. This signal is held at logic 0 in normal mode. If we want to operate scan, this signal is used as a control. Notice that GTE_IN doesn't fan-out to each and every flipflop as in conventional scan.

3. *Input signal Global CLK* refers to the clock used in normal mode operation.

4. *Output signal GTO* is the output line of the scan-chain and serve as the tail-end of the scan-chain from where data bits are pulled out serially.

5. *Output signal GTE_OUT* is the indicator signal that the data bit at GTI has been registered and can change. GTE_OUT can be connected to GTE_IN signal of another scan-chain in case we want to merge two scan-chains (from the same or different chips) into a single one.

Normal Mode Operation

In normal mode operation, input signal GTE_IN is held stable at logic 0. As a result, TE input of the flipflop FF3 (Fig. 7.18) is 0. Note that when TE = 0, CD = 0. Therefore TE input of second flipflop (FF2) is also 0 which in turn makes TE input of FF1 zero. As a result, TE input of all the D-flipflops is logic 0. Each flipflop simply acts as a normal D-flipflop ignoring TI input.

Scan-in and Scan-out Operation

Typically the scan-in and the scan-out operations are performed at the same time i.e. while existing data bits stored in the registers are being pulled out from output GTO, new data is introduced into the chain through GTI input. Following is the series of steps to scan-in a new data bit.

1. Global CLK = 0 and is held at 0 until all the steps below are completed. For all flipflops, N1 = 1 and N2 = 1.

2. GTE_IN that was 0 in normal mode is now asserted i.e. GTE_IN= $0 \rightarrow 1$. This is TE(FF3).

3. When TE(FF3) = 1, the outputs N1 and N2 of flipflop FF3 must be $\overline{TI}$ and TI respectively and Q = TI. Flipflop FF3 copies the state stored in flipflop FF2. Finally since TE(FF3) is 1, signal CD(FF3) rises to 1.

4. Note that signal CD(FF3) is input signal TE(FF3). FF2 in turn copies the state stored in FF1.

5. As a result, the rising trigger applied at GTE_IN ripples up the scan-chain forcing each flipflop to copy the state from its predecessor in the chain. The first flipflop FF1 copies the state applied at the input GTI. Effectively a new

bit is scanned into the chain. The rising of output GTE_OUT signals that the new bit has been successfully scanned in.

6. Now a falling trigger is applied at GTE_IN i.e. GTE_IN= 1 $\rightarrow$ 0. TE input of FF3 falls which causes its output CD to fall as well. Note that output Q of FF3 is un-affected since falling transition on TE doesn't change Q. The falling trigger also ripples up the chain making no change in the state.

7. When signal GTE_OUT falls to 0, a new data bit can now be introduced at the input GTI and step 2 on-wards are repeated until the chain is filled with new data.

7.4.4 Advantages of the Proposed Scan Mechanism

1. No hold time violations in the scan-chain itself. This is because in the test mode operation, the current flipflop copies the state of the previous flipflop. The previous flipflop doesn't start latching its inputs until the current flipflop has finished consuming output of the previous flipflop. This is the main advantage of the completion detection (CD) circuitry.

2. Elimination of a single global test-enable signal (TE) thereby reducing routing resources. The global test-enable TE signal is routed to all the flipflops which requires buffers and routing area.

3. Elimination of the separate scan clock which reduces routing resources required to route the scan clock.

4. Synthesis and automatic layout tools do not have to optimize timing related to scan mechanism due to asynchronous operation of scan testing.

5. If two independently developed hardware blocks that use this scanning mechanism reside on a same chip, then their scan inputs/outputs can simply be cascaded to make a single larger chain. This reduces the scan IO pin overhead without worrying about errors in scanning mechanism itself.

However despite these advantages *what remains to be analyzed* is the power and area overhead of this scan mechanism. The flipflops that implement aforementioned functionality are typically differential mode flipflops. There exist several variants of these flipflops that are not only faster but more energy efficient compared to the traditional master-slave flipflops such as KVFF and PNANDs. The additional functional block F_{CD} can be embedded inside these differential mode flipflops. There is no sizing restriction on this block and hence the transistors shown in Fig. 7.17 can be minimum size allowed by a process technology.

Additionally, existing scan mechanism requires a global test-enable TE signal which is routed to each and every flipflop and therefore is a high fanout net. All the buffers required to route test-enable signal can now be eliminated. Additionally, the differential mode flipflops that are amenable to the proposed scan mechanism, are typically faster. Due to their higher speed, the logic surrounding these flipflops can be reduced. The area overhead is expected to be negative indicating an improved area of the circuit.

7.4.5 Flipflop Implementation

A possible implementation of the flipflop described in Fig. 7.16 is shown in Fig. 7.19 which we can see is derived from original PNAND architecture.

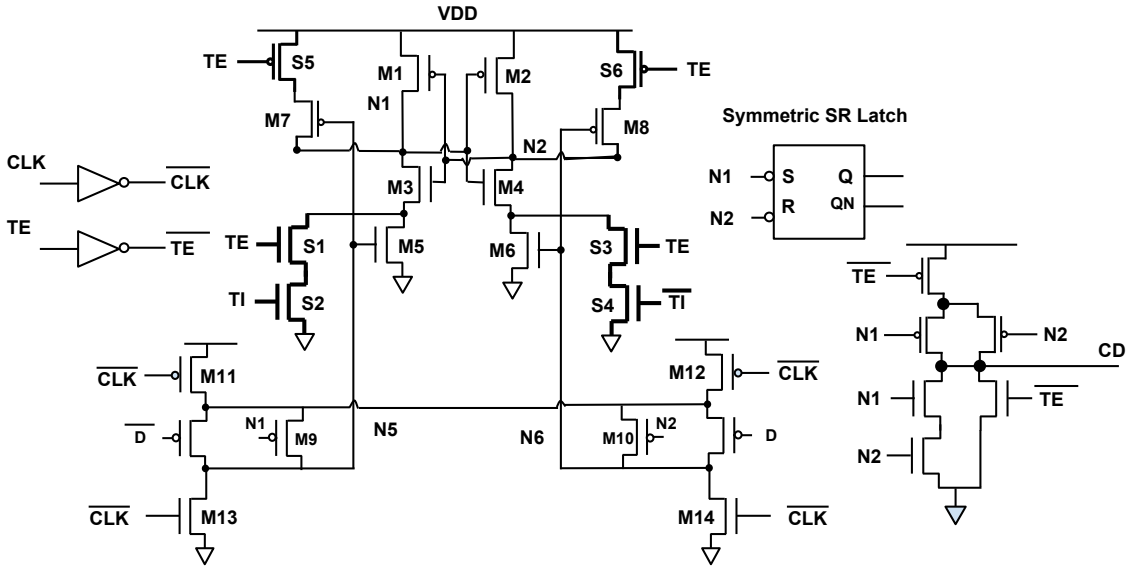


Figure 7.19: A Possible Implementation of the Flipflop Shown in Fig. 7.16

REFERENCES

Appenzeller, J., Y.-M. Lin, J. Knoch, Z. Chen and P. Avouris, "Comparing carbon nanotube transistors - the ideal choice: a novel tunneling device design", Electron Devices, IEEE Transactions on **52**, 12, 2568–2576 (2005).

Beiu, V., "A survey of perceptron circuit complexity results", in "Intl. Joint Conf. Neural Networks IJCNN03", vol. 2, pp. 989–994 (2003).

Beiu, V., J. M. Quintana and M. J. Avedillo, "VLSI Implementations of Threshold Logic - A Comprehensive Survey", IEEE Trans. Neural Networks **14**, 1217–1243 (2003).

Blair, E. and C. Lent, "Quantum-Dot Cellular Automata: an Architecture for Molecular Computing", in "Simulation of Semiconductor Processes and Devices, 2003. SISPAD 2003. International Conference on", pp. 14 – 18 (2003).

Bobba, S. and I. Hajj, "Current-mode threshold logic gates", in "Computer Design, 2000. Proceedings. 2000 International Conference on", pp. 235–240 (2000).

Brzozowski, I. and A. Kos, "Minimisation of power consumption in digital integrated circuits by reduction of switching activity", in "EUROMICRO Conference, 1999. Proceedings. 25th", vol. 1, pp. 376–380 vol.1 (1999).

Byskov, J., "Algorithms for k-colouring and finding maximal independent sets", in "Proc. Symp. on Discrete Algorithms", pp. 456–457 (PA, USA, 2003).

Byskov, J. M., "Enumerating maximal independent sets with applications to graph colouring", Oper. Res. Lett. **32**, 6, 547–556, URL `http://dx.doi.org/10.1016/j.orl.2004.03.002` (2004).

Case, M. L., A. Mishchenko and R. K. Brayton, "Cut-based inductive invariant computation", in "Proc. IWLS'08", pp. 172–179 (2008).

Celinski, P., J. Lopez, S. Al-Sarawi and D. Abbott, "Low power, high speed, charge recycling cmos threshold logic gate", Electronics Letters **37**, 17, 1067–1069 (2001).

Chalmers, D. and M. Sloman, "A survey of quality of service in mobile computing environments", Communications Surveys, IEEE **2**, 2, 2–10 (1999).

Chandrakasan, A. and R. Brodersen, eds., *Low Power CMOS Design* (Wiley-IEEE Press, 1998).

Chandrakasan, A. P. and R. W. Brodersen, *Low power digital CMOS design* (Springer Science & Business Media, 2012).

Chatterjee, S., *On algorithms for technology mapping, PhD Thesis* (ProQuest, 2007).

Chatterjee, S., A. Mishchenko and R. K. Brayton, "Factor cuts", in "Proc. ICCAD '06", pp. 143–150 (2006).

Chen, D.-S. and M. Sarrafzadeh, "An exact algorithm for low power library-specific gate re-sizing", in "Proceedings of the 33rd Annual Design Automation Conference", DAC '96, pp. 783–788 (ACM, New York, NY, USA, 1996), URL `http://doi.acm.org/10.1145/240518.240666`.

Chen, K. J., K. Maezawa and M. Yamamoto, "Novel current-voltage characteristics of an inp-based resonant- tunneling high electron mobility transistor and their circuit applications", in "International Electron Devices Meeting", pp. 379–382 (1995).

Cong, J. and Y. Ding, "FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs", IEEE Trans. CAD **13**, 1, 1–12 (1994).

Cong, J., G. Han and Z. Zhang, "Architecture and compiler optimizations for data bandwidth improvement in configurable processors", IEEE Trans. on VLSI Syst. **14**, 9, 986–997 (2006).

Cong, J., C. Wu and Y. Ding, "Cut ranking and pruning: enabling a general and efficient FPGA mapping solution", in "Proc. FPGA'99", pp. 29–35 (ACM, New York, 1999).

Cong, J. and B. Xiao, "Fpga-rpi: A novel fpga architecture with rram-based programmable interconnects", Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **22**, 4, 864–877 (2014).

Cormen, T., C. Leiserson, R. Rivest and C. Stein, *Introduction to algorithms* (MIT Press, Cambridge, MA, 2001).

Corneil, D. G. and Y. Perl, "Clustering and domination in perfect graphs", Discrete Applied Mathematics **9**, 1, 27–39 (1984).

Coudert, O., "Gate sizing for constrained delay/power/area optimization", Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **5**, 4, 465–472 (1997).

Dara, C., T. Haniotakis and S. Tragoudas, "Delay analysis for an n-input current mode threshold logic gate", in "VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on", pp. 344–349 (2012).

Dechu, S., M. Goparaju and S. Tragoudas, "A metric of tolerance for the manufacturing defects of threshold logic gates", in "Defect and Fault Tolerance in VLSI Systems, 2006. DFT '06. 21st IEEE International Symposium on", pp. 318–326 (2006).

Devadas, S. and S. Malik, "A survey of optimization techniques targeting low power circuits", in "in Proc. Design Automation Conf", pp. 242–247 (1995).

Een, N., A. Mishchenko and N. Sorensson, "Applying logic synthesis for speeding up SAT", Lect. N. Comp. Sci. **4501**, 272 (2007).

Eppstein, D., "Small Maximal Independent Sets and Faster Exact Graph Coloring", in "Proceedings of the 7th International Workshop on Algorithms and Data Structures", WADS '01, pp. 462–470 (Springer-Verlag, London, UK, 2001a), URL http://portal.acm.org/citation.cfm?id=645933.673363.

Eppstein, D., "Small maximal independent sets and faster exact graph coloring", Lect. N. Comp. Sci. pp. 462–470 (2001b).

Fant, K. and S. Brandt, "Null convention logictm: a complete and consistent logic for asynchronous digital circuit synthesis", in "Application Specific Systems, Architectures and Processors, 1996. ASAP 96. Proceedings of International Conference on", pp. 261–273 (1996).

Fishburn, J., "Clock skew optimization", Computers, IEEE Transactions on **39**, 7, 945–951 (1990).

Ford, L. and D. Fulkerson, "Flow in networks", Princeton University Press, Princeton, NJ (1962).

Golumbic, M., *Algorithmic graph theory and perfect graphs* (North-Holland, 2004).

Gonzalez, R. and M. Horowitz, "Energy dissipation in general purpose microprocessors", Solid-State Circuits, IEEE Journal of **31**, 9, 1277–1284 (1996).

Gowda, T., S. Vrudhula and N. Kulkarni, "Identification of threshold functions and synthesis of threshold networks", IEEE Transactions on Computer-Aided Design (TCAD) **30**, 5, 665–677 (2011).

Gupta, P. and N. Jha, "An algorithm for nanopipelining of rtd-based circuits and architectures", Nanotechnology, IEEE Transactions on **4**, 2, 159–167 (2005).

Hachtel, G. D. and F. Somenzi, *Logic Synthesis and Verification Algorithms* (Kluwer Academic, 1996).

Hidalgo-Lopez, J., J. Tejero, J. Fernandez and A. Gago, "New types of digital comparators", in "Circuits and Systems, 1995. ISCAS '95., 1995 IEEE International Symposium on", vol. 1, pp. 29–32 vol.1 (1995).

Hu, X. S., S. C. Bass and R. G. Harber, "Minimizing the number of delay buffers in the synchronization of pipelined systems", Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **13**, 12, 1441–1449 (1994).

Huang, H.-Y. and T.-N. Wang, "Cmos capacitor coupling logic (c3l) circuits", in "ASICs, 2000. AP-ASIC 2000. Proceedings of the Second IEEE Asia Pacific Conference on", pp. 33–36 (2000).

Ivan S. Kourtev, E. G. F., Baris Taskin, *Timing Optimization Through Clock Skew Scheduling* (Springer US, 2009).

J, L., "Threshold gate circuits employing field-effect transistors", URL http://www.google.com/patents/US3715603, uS Patent 3,715,603 (1973).

J., M. V., "Majority logic circuit using a constant current bias", URL http://www.google.com/patents/US3155839, uS Patent 3,155,839 (1964).

Jian, P. S., G. Haddad, P. Mazumder and J. Schulman, "Resonant Tunneling Diodes: Models and Properties", Proceedings of the IEEE **86**, 4, 641 –660 (1998).

Kagaris, D. and S. Tragoudas, "Maximum Weighted Independent Sets on Transitive Graphs and Applications", Integr. VLSI J. **27**, 77–86, URL http://portal.acm.org/citation.cfm?id=309614.309619 (1999a).

Kagaris, D. and S. Tragoudas, "Maximum weighted independent sets on transitive graphs and applications", Integration, the VLSI Journal **27**, 1, 77–86 (1999b).

Keutzer, K., "Dagon: technology binding and local optimization by dag matching", in "Papers on Twenty-five years of electronic design automation", pp. 617–624 (ACM, 1988).

Klass, F., "Semi-dynamic and dynamic flip-flops with embedded logic", in "VLSI Circuits, 1998. Digest of Technical Papers. 1998 Symposium on", pp. 108–109 (1998).

Kotani, K., T. Shibata, M. Imai and T. Ohmi, "Clocked-neuron-mos logic circuits employing auto-threshold-adjustment", in "Solid-State Circuits Conference, 1995. Digest of Technical Papers. 41st ISSCC, 1995 IEEE International", pp. 320–321 (1995).

Kotani, K., T. Shibata, M. Imai and T. Ohmi, "Clock-controlled neuron-mos logic gates", Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on **45**, 4, 518–522 (1998).

Kulkarni, N., N. Nukala and S. Vrudhula, "Minimizing area and power of sequential cmos circuits using threshold decomposition", in "Proceedings of the International Conference on Computer-Aided Design", ICCAD '12, pp. 605–612 (ACM, New York, NY, USA, 2012), URL http://doi.acm.org/10.1145/2429384.2429514.

Kumar, P., S. Pradeep and S. Pratibha, "Lssr: Lector stacked state retention technique a novel leakage reduction and state retention technique in low power vlsi design", (2013).

Lageweg, C., S. Cotofana and S. Vassiliadis, "A linear threshold gate implementation in single electron technology", in "Proc. IEEE Computer Society Workshop on VLSI", pp. 93–98 (2001).

Lageweg, C. R., S. D. Cotofana and S. Vassiliadis, "A full adder implementation using set based linear threshold gates", in "Proceedings 9th IEEE International conference on electronics, circuits and systems - ICECS 2002", pp. 665–669 (2002).

Lai, Y.-T., M. Pedram and S. B. K. Vrudhula, "Bdd based decomposition of logic functions with application to fpga synthesis", in "Proceedings of the 30th international Design Automation Conference", DAC '93, pp. 642–647 (ACM, New York, NY, USA, 1993), URL http://doi.acm.org/10.1145/157485.165078.

Lamoureux, J., G. Lemieux and S. J. E. Wilton, "Glitchless: Dynamic power minimization in fpgas through edge alignment and glitch filtering", Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **16**, 11, 1521–1534 (2008).

Leiserson, C. E. and J. B. Saxe, "Retiming synchronous circuitry", Algorithmica **6**, 1-6, 5–35, URL `http://dx.doi.org/10.1007/BF01759032` (1991).

Lemonds, C. and S. Shetti, "A low power 16 by 16 multiplier using transition reduction circuitry", in "Proceedings of the International Workshop on Low Power Design", pp. 139–142 (1994).

Leshner, S., *Modeling and Implementation of Threshold Logic Circuits and Architectures*, Ph.D. thesis, Arizona State University (2010).

Leshner, S., B. Krzysztof and S. Vrudhula, "Design of a robust, high performance standard cell threshold logic family for deep sub-micron technology", in "Proceedings of the IEEE International Conference on Microelectronics", pp. 52–55 (Cairo, Egypt, 2010).

Lin, S.-C. and K. Banerjee, "Cool chips: Opportunities and implications for power and thermal management", Electron Devices, IEEE Transactions on **55**, 1, 245–255 (2008).

Ling, A. C., J. Zhu and S. D. Brown, "BddCut: Towards scalable symbolic cut enumeration", in "Proc. ASP-DAC'07", pp. 408–413 (2007).

López-García, J., J. Fernández-Ramos and A. Gago-Bohórquez, "A balanced capacitive threshold-logic gate", Analog Integr. Circuits Signal Process. **40**, 1, 61–69, URL `http://dx.doi.org/10.1023/B:ALOG.0000031434.48142.a3` (2004).

Lorch, J. R., *Operating systems techniques for reducing processor energy consumption*, Ph.D. thesis, University of California, Berkeley (2001).

Maheshwari, N. and S. Sapatnekar, *Timing Analysis and Optimization of Sequential Circuits* (Kluwer Academic, 1999).

Martin, A. J., M. Nyström and P. I. Pénzes, "Power aware computing", pp. 293–315 (Kluwer Academic Publishers, Norwell, MA, USA, 2002), URL `http://dl.acm.org/citation.cfm?id=783060.783076`.

Mishchenko, A., R. Brayton, J.-H. R. Jiang and S. Jang, "Scalable don't-care-based logic optimization and resynthesis", in "Proc. FPGA '09", pp. 151–160 (ACM, NY, 2009).

Mishchenko, A., S. Chatterjee, R. Brayton, X. Wang and T. Kam, "Technology mapping with boolean matching, supergates and choices", (2005).

Mishchenko, A., S. Cho, S. Chatterjee and R. Brayton, "Combinational and sequential mapping with priority cuts", in "Proc. ICCAD'07", pp. 354–361 (2007).

Monteiro, J., S. Devadas and A. Ghosh, "Retiming sequential circuits for low power", International journal of high speed electronics and systems **7**, 02, 323–340 (1996).

Moon, J. and L. Moser, "On cliques in graphs", Israel Journal of Mathematics **3**, 1, 23–28 (1965).

Moyer, B., "Low-power design for embedded processors", Proceedings of the IEEE **89**, 11, 1576–1587 (2001).

Muroga, S., *Threshold Logic and its Applications* (Wiley-Interscience New York, 1971).

Nikolic, B., V. G. Oklobdzija, V. Stojanovic, W. Jia, J. K.-S. Chiu and M. Ming-Tak Leung, "Improved sense-amplifier-based flip-flop: design and measurements", Solid-State Circuits, IEEE Journal of **35**, 6, 876–884 (2000).

Nukala, N., N. Kulkarni and S. Vrudhula, "Spintronic threshold logic array (stla) - a compact, low leakage, non-volatile gate array architecture", in "Nanoscale Architectures (NANOARCH), 2012 IEEE/ACM International Symposium on", pp. 188–195 (2012).

Özdemir, H., A. Kepkep, B. Pamir, Y. Leblebici and U. Çilingiroğlu, "A capacitive threshold-logic gate", Solid-State Circuits, IEEE Journal of **31**, 8, 1141–1150 (1996).

Padure, M., S. Cotofana, C. Dan, M. Bodea and S. Vassiliadis, "A new Latch-based Threshold Logic Family", in "CAS 2001 Proceedings. International Semiconductor Conference, 2001.", vol. 2, pp. 531 –534 vol.2 (2001a).

Padure, M., S. Cotofana, C. Dan, S. Vassiliadis and M. Bodea, ""A new latch-based threshold logic family"", in "International Semiconductor Conference", vol. 2, pp. 531–534 (2001b).

Pan, P. and C.-C. Lin, "A new retiming-based technology mapping algorithm for LUT-based FPGAs", in "Proc. FPGA '98", pp. 35–42 (ACM, New York, 1998).

Pan, P. and C. L. Liu, "Optimal clock period FPGA technology mapping for sequential circuits", ACM TODAES **3**, 3, 437–462 (1998).

Panda, P. R., A. Shrvastava, B. Silpa and K. Gummidipudi, *Power-efficient System Design* (Springer, 2010).

Partovi, H., R. Burd, U. Salim, F. Weber, L. DiGregorio and D. Draper, "Flow-through latch and edge-triggered flip-flop hybrid elements", in "Solid-State Circuits Conference, 1996. Digest of Technical Papers. 42nd ISSCC., 1996 IEEE International", pp. 138–139 (1996).

Peddersen, J., S. Shee, A. Janapsatya and S. Parameswaran, "Rapid embedded hardware/software system generation", in "Int. Conf. on VLSI Design", pp. 111–116 (2005).

Piguet, C., *Low-power processors and systems on chips* (CRC Press, 2005).

Prost, W., U. Auer, F. J. Tegude, C. Pacha, K. F. Goser, G. Janssen and R. T. Van Der, "Manufacturability and robust design of nanoelectronic logic circuits based on resonant tunneling diodes", Int. J. Circ. Theory Appl. **28**, 537–552 (2000).

Qadri, M. Y., H. S. Gujarathi and K. D. McDonald-Maier, "Low power processor architectures and contemporary techniques for power optimization–a review", Journal of computers **4**, 10, 927–942 (2009).

Rahul B. Deokar, S. S. S., "A fresh look at retiming via clock skew optimization", in "Design Automation, 1995. DAC '95. 32nd Conference on", pp. 310–315 (1995).

Rajendran, J., H. Manem, R. Karri and G. Rose, "Memristor based programmable threshold logic array", in "Nanoscale Architectures (NANOARCH), 2010 IEEE/ACM International Symposium on", pp. 5–10 (2010).

Rose, J., A. E. Gamal, S. Member and A. Sangiovanni-vincentelli, "Architecture of field-programmable gate arrays: The effect of logic block functionality on area efficiency", Proceedings of the IEEE **25**, 1217–1225 (1990).

Rose, J., J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson and J. Anderson, "The vtr project: Architecture and cad for fpgas from verilog to routing", in "Proceedings of the 20th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", pp. 77–86 (ACM, 2012).

Roy, K., S. Mukhopadhyay and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits", Proceedings of the IEEE **91**, 2, 305–327 (2003).

Roy, K. and S. Prasad, "Syclop: Synthesis of cmos logic for low power applications", in "Computer Design: VLSI in Computers and Processors, 1992. ICCD'92. Proceedings, IEEE 1992 International Conference on", pp. 464–467 (IEEE, 1992).

Shen, A., A. Ghosh, S. Devadas and K. Keutzer, "On average power dissipation and random pattern testability of cmos combinational logic networks", in "Proceedings of the 1992 IEEE/ACM international conference on Computer-aided design", pp. 402–407 (IEEE Computer Society Press, 1992).

Shibata, T. and T. Ohmi, "An intelligent mos transistor featuring gate-level weighted sum and threshold operations", in "Electron Devices Meeting, 1991. IEDM '91. Technical Digest., International", pp. 919–922 (1991).

Siu, K.-Y., V. Roychowdhury and T. Kailath, *Discrete Neural Computation: A Theoretical Foundataion*, Information and Systems Sciences Series (Prentice Hall, Englewood Cliffs, New Jersey, 1995).

Sparso, J. and S. Furber, *Principles Asynchronous Circuit Design* (Springer, 2002).

Strandberg, R. and J. Yuan, ""Single input current-sensing differential logic (SCSDL)"", in "International Symposium on Circuits and Systems", vol. 1, pp. 764–767 (2000a).

Strandberg, R. and J. Yuan, "Single Input Current-Sensing Differential Logic (SCSDL)", in "Proceedings. ISCAS 2000 Geneva. The IEEE International Symposium on Circuits and Systems, 2000.", vol. 1, pp. 764–767 vol.1 (2000b).

Sulieman, M. and V. Beiu, "Characterization of a 16-bit threshold logic single-electron technology adder", in "Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on", vol. 3, pp. III–681 (IEEE, 2004).

Takata, T. and Y. Matsunaga, "An Efficient Cut Enumeration for Depth-optimum Technology Mapping for LUT-based FPGAs", in "Proceedings of the 19th ACM Great Lakes Symposium on VLSI", GLSVLSI '09, pp. 351–356 (ACM, New York, NY, USA, 2009), URL http://doi.acm.org/10.1145/1531542.1531622.

Tan, C. H. and J. Allen, "Minimization of power in vlsi circuits using transistor sizing, input ordering, and statistical power estimation", in "Proceedings of the 1994 International Workshop on Low Power Design", pp. 75–80 (1994).

Tiwari, V., P. Ashar and S. Malik, "Technology mapping for low power", in "Design Automation, 1993. 30th Conference on", pp. 74–79 (IEEE, 1993).

Tsui, C.-Y., M. Pedram, C.-A. Chen and A. M. Despain, "Low power state assignment targeting two-and multi-level logic implementations", in "Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design", pp. 82–87 (IEEE Computer Society Press, 1994).

Vijayakumar, A. and S. Kundu, "Glitch power reduction via clock skew scheduling", in "VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on", pp. 504–509 (2014).

Z., F. R. and S. E. W., "Direct coupled, current mode logic", URL http://www.google.com/patents/US3321639, uS Patent 3,321,639 (1967).

Zandrahimi, M. and Z. Al-Ars, "A survey on low-power techniques for single and multicore systems", in "Proceedings of the 3rd International Conference on Context-Aware Systems and Applications", ICCASA '14, pp. 69–74 (ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2014), URL http://dl.acm.org/citation.cfm?id=2762722.2762735.

Zhang, L. and S. Cotofana, "An input weights aware synthesis tool for threshold logic networks", in "In Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc 2005", pp. 578–583 (2005).

Zhang, R., P. Gupta, L. Zhong and N. K. Jha, "Threshold Network Synthesis and Optimization and Its Application to Nanotechnologies", IEEE Transactions on Computer-Aided Design **24**, 1 (January 2005).

Zilic, Z. and Z. Vranesic, "Using bdds to design ulms for fpgas", in "Field-Programmable Gate Arrays, 1996. FPGA '96. Proceedings of the 1996 ACM Fourth International Symposium on", pp. 24–30 (1996).

APPENDIX A

CUT ENUMERATION

This chapter provides a detailed discussion of the strong line cut enumeration idea introduced in Section 4.2.1. The proposed line cut enumeration is applicable for general technology mapping as well. Therefore we treat line cut enumeration as an independent problem and discuss in depth in this chapter.

A.1 Introduction

Technology mapping (TM) is a process of transforming a *generic Boolean network*, which is a network consisting of primitive gates (e.g. AND/OR), into an equivalent *mapped network*, that consists of a network of *cells* from a given technology library. Depending on the target implementation, the library cells can correspond to either lookup tables (LUT) in the case of an FPGA, or to a pre-designed set of standard cells in the case of an ASIC. The measures of delay, power, area, or some combination of them serve as an objective function to optimize during the transformation process.

TM is formalized as a graph covering problem. A given Boolean network is represented as a *directed acyclic graph* (DAG) $G = (V, E)$, which is referred to as the *subject graph*. The cells in the library, being single output Boolean functions, are also represented by DAGs, and each is referred to as a *pattern graph*. A feasible solution of TM is a complete covering of the subject graph with one or more of the pattern graphs (see Figure A.1). A core step in this process is to identify a subgraph in the subject graph to match with one or more pattern graphs. In the *structural approach* to TM (Chatterjee, 2007), the selection of sub-graphs is achieved by computing a *cut*.

The recent works on structural approaches to TM (Mishchenko *et al.*, 2007; Chatterjee *et al.*, 2006; Cong *et al.*, 1999; Ling *et al.*, 2007) are based on a particular type of a cut, called *minimal node-cut*. A *node-cut* C_v associated with a node $v \in V$ is a subset of nodes in the transitive fanin cone of v such that every path from the primary inputs to v includes a node in C_v. Note that in the existing literature, the definition of a node-cut restricts it to be *minimal*, i.e., no proper super-set of a node cut is a legal node cut. In this chapter, we will explicitly refer to a minimal node cut, if that is the case. By definition, A *k-feasible minimal node cut* of a node v is a minimal node cut of cardinality k (Chatterjee *et al.*, 2006).

A cut together with a node v defines a single-sink subgraph that is either directly mappable onto a k-input LUT in the case of an FPGA, or constitutes a match candidate for a NPN-equivalent standard cell, for an ASIC design, provided such an entity exists in the given library. In the literature k-feasible node cuts are usually simply referred as *k-feasible cuts* as no other type of cut is usually considered for technology mapping (Mishchenko *et al.*, 2007; Chatterjee *et al.*, 2006; Cong *et al.*, 1999; Ling *et al.*, 2007).

In Figure A.1, associated with node $G0$, the sets $\{G6, G7\}$, and $\{G6, G8\}$ are 2-feasible minimal node cuts, whereas $\{G2, G5, G7\}$ is a 3-feasible minimal node cut. When the cut $\{G6, G7\}$ is selected, the actual sub-graph replaced by a cell in TM consists of gates $\{G8, G0\}$. Similarly, when the cut $\{G2, G5, G7\}$ is chosen the subcircuit consisting of gates $\{G8, G6, G0\}$ is replaced.

A node may have many associated k-feasible node cuts, which result in different coverings. In TM, the *quality* of a covering is usually taken to be the delay of the critical path, and/or the total area of the mapped circuit. Hence, to evaluate the quality of the complete covering, cuts also need to be *evaluated*. The quality of a cut

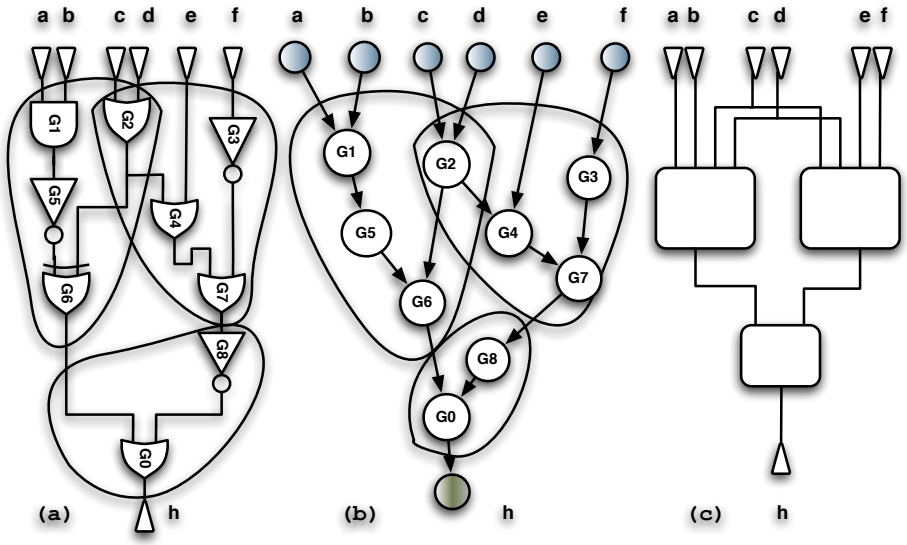


Figure A.1: a) A Covered Boolean Network. b) Its Graph Representation. c) The Network Mapped on the Library Gates.

is typically a combined measure of the subcircuit that will be replaced by a library cell and the subcircuit that feeds that cell. Since neither of these can be evaluated without knowing the cut, enumeration of cuts is a core task in TM. Consequently, there has been a significant amount of effort devoted to the development of efficient algorithms for enumerating k-feasible node cuts (Chatterjee *et al.*, 2006; Cong and Ding, 1994; Ling *et al.*, 2007; Pan and Liu, 1998).

While structural technology mapping turned out to be very successful, minimal node cuts bear some bias that eventually limits the number of possible matches. As demonstrated in (Mishchenko *et al.*, 2005), this may result in excluding many feasible, high quality matches from the evaluation. The authors of (Mishchenko *et al.*, 2005) address this problem by constructing so-called *supergates* i.e. single output networks of library gates that are added to the library in order to increase the number of matches. This demonstrates that enhancing a match space could yield significant benefits for structural technology mapping.

Existing work on TM focuses on minimal node cuts. However, including non-minimal node cuts can increase substantially increase the possible matches. Consider Figure A.2. The node cut $C = \{a, b, x, d\}$ is not minimal since $\{a, b, d\}$ is also a (minimal) node cut. However C corresponds to the function $ab + x + d$. This happens to be a linearly separable (threshold) function, which would never be found by any cut enumerator that enumerates only minimal node cuts. Another representation of a cut, based on edges, is called a *line cut*, which includes both minimal and non-minimal node cuts.

Definition 2. *(a) A line cut is a minimal set of directed edges in a single sink DAG which when removed, eliminates all paths from any of source (primary input) to the sink i.e. produces an "S-T bipartition".*

(b) A line cut is k-feasible if its cardinality (number of edges) is k or smaller.

(c) A line cut is called a strong line cut (Kagaris and Tragoudas, 1999b) if no two edges in the line cut are on the same directed path.

Note that line cuts and node cuts are simply two representations of same entity, and either form can be converted into other. They both partition the nodes of the DAG into two mutually exclusive sets S and T. In such an S-T partition, the set S of nodes must contain primary inputs and T must contain the sink node v.

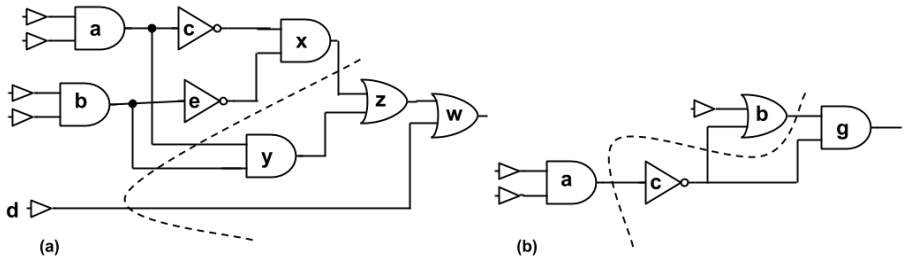


Figure A.2: (a) Unidirectional Node Cut Denoted as $\{a, b, x, d\}$ (b) Bidirectional Node Cut Denoted by $\{a, b\}$

Figure A.2 a) shows a line cut consisting of edges $\{(x, z), (a, y), (b, y), (d, w)\}$. The node cut corresponding to this line cut is $C = \{a, b, x, d\}$, which is not minimal. Since minimal and non-minimal node cuts are useful, it appears that line cuts are the cuts that should be enumerated. A cut can also be classified as being unidirectional or bidirectional. It is unidirectional if all the edges between S and T originate in S and terminate in T. Otherwise it is bidirectional.

In Figure A.2(a) the line cut $\{(x, z), (a, y), (b, y), (d, w)\}$ (corresponding node cut being $C = \{a, b, x, d\}$) generates $S = \{a, b, c, d, e, x\}$, and $T = \{y, z, w\}$, and is unidirectional. In Figure A.2(b), the line cut $\{(a, c), (b, g)\}$ (corresponding node cut is $\{a, b\}$) has $S = \{a, b\}$ and $T = \{c, g\}$, and is bidirectional since (b, g) is a "forward edge" and (c, b) is a "backward edge". Note also that the minimal node cut $\{a, b, d\}$ would identify $ab + a'b' + d$ as a function to be replaced by some cell from the library. However this is neither a member of any well defined class of functions, e.g. threshold functions, nor it is a function that would be in a typical cell library. Thus, minimal cuts are not always the most useful or desirable. In addition, we show that bidirectional cuts are not necessary and discarding them does not degrade quality of TM with regard to critical path delay. Thus we need only enumerate unidirectional node cuts (minimal or non-minimal).

We establish a one-to-one correspondence between unidirectional node cuts and *strong* line cuts (Kagaris and Tragoudas, 1999b). This correspondence is important because there exists a well established relation between strong line cuts in a DAG and independent sets in its corresponding *line dependency graph* (LDG) (Kagaris and Tragoudas, 1999b). This allows for construction of fast strong line cut enumeration techniques based on enumeration of independent sets. Since the latter is a problem well researched in computational graph theory, techniques exist that can be directly applied to enumerating line cuts by enumerating independent sets. Figure A.3 shows classification of cuts and their relationships. The proposed technique for enumerating strong line cuts was used in (Kulkarni *et al.*, 2012) to find threshold functions in a logic circuit.

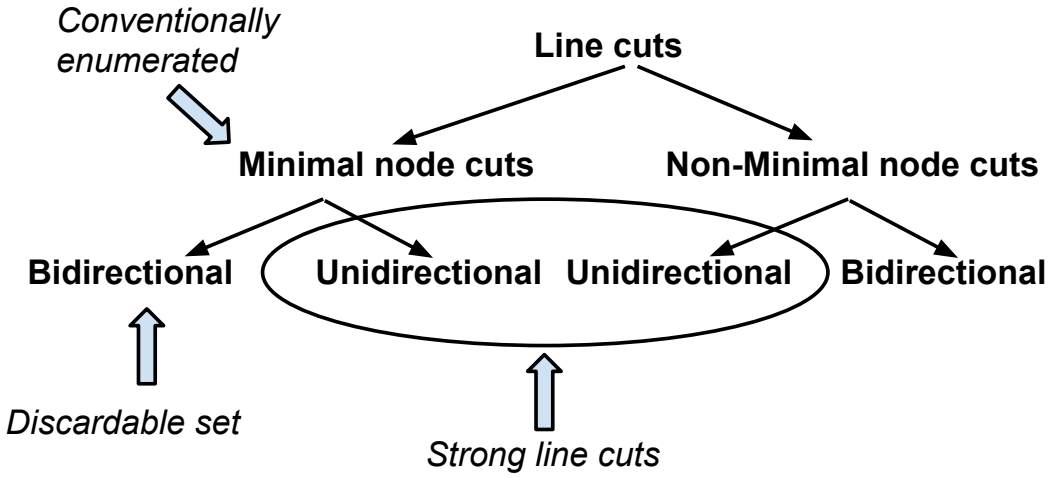


Figure A.3: Classification of Cuts and Their Relationships

The new ideas in this chapter include:

- Introduction of unidirectional node cuts and a proof showing that restricting the cut space to only unidirectional node cuts does not degrade the quality of mapping for delay.
- Establishing the equivalence unidirectional node cuts and strong line cuts.
- An efficient k-feasible strong line cut enumeration algorithm based on the relationship between a DAG and its LDG.
- A general framework and the specific implementation of a pruning technique for k-feasible strong line cut enumeration.
- An efficient implicit representation of bounded size MISs of a graph that allows for both unconstrained and constrained enumeration of such MISs.

A.2 Related Work

The importance of cut computation in TM for FPGAs was first identified by the Cong *et al.* (Cong and Ding, 1994). They developed a novel and elegant network flow based algorithm that directly identified a single, depth-optimal, k-feasible node cut, without enumerating cuts. Later, Pan *et al.* (Pan and Lin, 1998; Pan and Liu, 1998) developed an efficient algorithm for enumerating cuts that avoided the large computational requirements of network flow. More recently, Ling *et. al* (Ling *et al.*, 2007) developed a novel scheme for implicitly encoding cuts using Binary Decision Diagrams (BDD). This representation allowed for extraction of cuts when the value of a cut could be computed recursively. However, the authors admit that BDD approach is not very well suited for cut enumeration since non-cuts, which dominate cuts, are also implicitly included and need to be pruned during enumeration.

Finding a computationally efficient way of encoding and enumerating cuts is of fundamental importance to technology mapping. Recently Takata *et al.* (Takata and Matsunaga, 2009) proposed a top-down scheme for the procedure of (Pan and Liu, 1998) and demonstrated speed-ups of 3x-8x for larger $k = 8, 9$. Unfortunately, since the number of cuts of size at most k is of $O(n^k)$, cut enumeration algorithms inherently suffer from poor scalability. To alleviate this problem, techniques for ranking and pruning of cuts were first proposed by Cong *et al.* in (Cong *et al.*, 1999). The basic observation of this work is that for certain optimization objectives it is possible to narrow the search down efficiently and extract depth-maximal or area-minimal cuts directly. Similar ideas, referred to as *priority cuts*, were proposed by Mischenko *et al.* in (Mishchenko *et al.*, 2007), where appropriate seeding of the procedure from (Pan and Liu, 1998) assured enumeration of only $O(n^2)$ priority cuts instead of $O(n^k)$ cuts. These can be further sorted by quality, and pruned. An alternative approach to pruning was proposed by Chatterjee *et al.* in (Chatterjee *et al.*, 2006) where they introduced hierarchical partitioning of the cut space based on a novel concept that is similar to algebraic factorization. The authors showed that while complete factorization may still suffer from poor scalability, partial factorization of the cut space could yield good, practical solutions with very short runtimes. Takata (Takata and Matsunaga, 2009) proposed a partial enumeration scheme that enumerates only a subset called *label cuts*. The scheme improves scalability of cut enumeration and guarantees to maintain the circuit's depth, at the expense of small increase in the estimated network area.

All the works identified above, and many others have demonstrated that *structural technology mapping*, the core of which involves cut enumeration, leads to far superior solutions than the traditional graph/tree matching based algorithms. Cut enumeration has also found uses in related applications such as re-synthesis through rewriting (Mishchenko *et al.*, 2009), application specific instruction set extension generation/optimization (Cong *et al.*, 2006), hardware/software co-design (Peddersen *et al.*, 2005), model checking in verification (Case *et al.*, 2008), and SAT problem preprocessing for simplification (Een *et al.*, 2007).

The use of a line dependency graph (LDG) derived from a DAG was proposed by Kagaris *et al.* (Kagaris and Tragoudas, 1999b) to compute the maximum strong cut in a circuit for the purpose of delay testing. Based on the observation that an LDG is a transitively-oriented graph, hence a *comparability graph* (Golumbic, 2004), they provide an efficient and elegant algorithm that computes a maximum independent set of the LDG using network flow. This set represents a maximum strong cut in the corresponding DAG. While their approach generated interest in the area of delay-testing, we will demonstrate that there is still greater opportunity for further exploration and exploitation of the DAG/LDG duality for strong cut enumeration.

A.3 Strong line cuts

We now describe the relation between DAGs and their corresponding line dependency graphs (LDG). An LDG is an undirected graph derived from a DAG that encodes the *transitive dependencies* between DAG edges (Kagaris and Tragoudas, 1999b). Each edge e of the DAG has a corresponding node v in the LDG. Two

nodes of an LDG are connected if and only if their corresponding lines in the DAG are *transitively dependent*, i.e., there exists a path in the DAG from any source to any sink that contains both edges. Consequently, if an edge in DAG is transitively dependent on another edge, by definition the corresponding nodes in LDG will be neighbors. Since LDGs are by definition *transitively oriented*, they are also *comparability graphs* (Golumbic, 2004).

An *independent set* (IS) in a graph $G(V, E)$ is a set S of vertices no two of which share an edge. A *maximal independent set* (MIS) is an independent set which is not a proper subset of any other independent set in the graph.

Lemma 2. *(From (Kagaris and Tragoudas, 1999b)) A strong line cut of a DAG forms a maximal independent set (MIS) in its corresponding LDG.*

Fig. A.4 illustrates the relation between DAGs and LDGs established by Lemma 2, on an example that we will use throughout this chapter for illustration. The direct consequence of this lemma is that enumerating all k-feasible strong line cuts in a DAG is equivalent to enumerating all maximal independent sets of size $\leq k$ in the LDG.

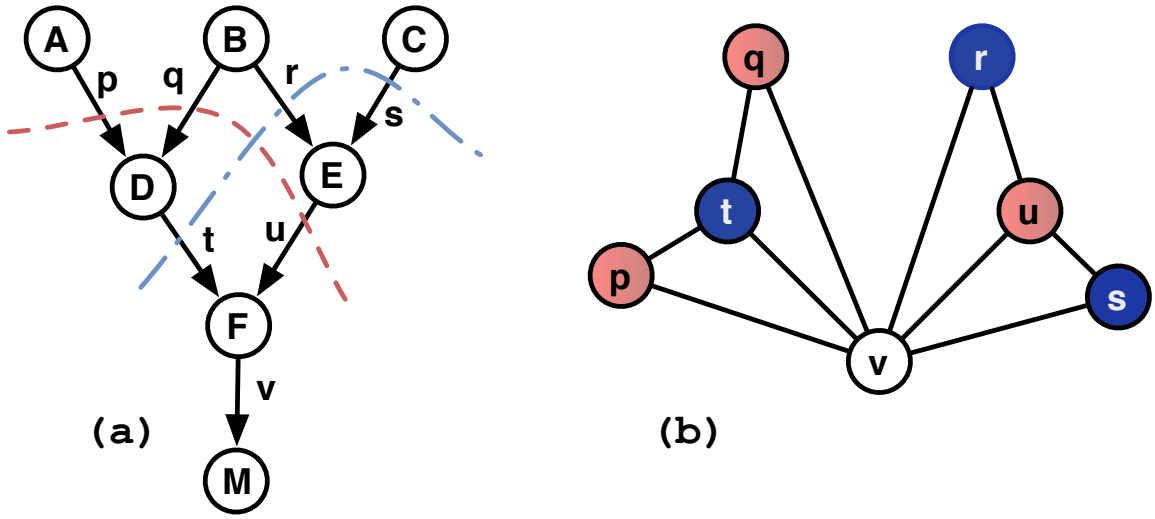


Figure A.4: a) a DAG with Strong Cuts Annotated. b) The Corresponding Maximal Independent Sets in LDG.

A.3.1 Relationship between Unidirectional Node Cuts and Strong Line Cuts

In this section, we show the equivalence between unidirectional node cuts and strong line cuts. We also establish the fact that if the cut space is restricted to unidirectional node cuts then the quality of technology mapping for minimizing delay remains same.

In the following we restrict the DAG to be a transitive fan-in cone of some gate in a circuit, since in TM only transitive fan-in cones of gates/nodes are considered for enumerating cuts. v refers to the root node whose transitive fan-in cone is being considered.

Lemma 3. *A strong line cut corresponds to a unidirectional set of edges crossing an $S-T$ partition.*

Proof. For an arbitrary node $u \in S$, there exists a path $u \rightsquigarrow v$. This is straightforward from the definition of DAG considered here which is the transitive fan-in cone of the node v. Assume that the $S-T$ partition corresponding to a strong line cut C_v is bidirectional, i.e., there exists a directed edge (p,q) such that $p \in T$ and $q \in S$. Then for some $x \in S$ and $y \in T$, $(x,y) \in C_v$, there must exist a path $x \rightarrow y \rightsquigarrow p$. Since edge (p,q) exists, there must exist a path $x \rightarrow y \rightsquigarrow p \rightarrow q$. Since $q \in S$, the root node v must be reachable from q through another edge in the cut C_v, say (r,s). Therefore we have a complete path that looks like $x \rightarrow y \rightsquigarrow p \rightarrow q \rightsquigarrow r \rightarrow s \rightsquigarrow v$. Note that edges (x,y) and (r,s) both belong to the cut. This is clearly a contradiction, since no two lines in the cut C_v should lie on same path. Also $(x,y) \neq (r,s)$ because that would lead to a directed cycle $x \rightarrow y \rightsquigarrow p \rightarrow q \rightsquigarrow x$ in the directed acyclic graph under consideration.

Conversely, assume a unidirectional node cut in which all the edges are from S to T, and suppose the corresponding line cut is not a strong line cut. Then there must exist at least edges $e_1 = (x,y)$ and $e_2 = (u,w)$ in the line cut that are on the same path to node v (the output). Assume e_1 precedes e_2 in the path. By definition of a $S-T$ partition, $x \in S$, $y \in T$, $u \in S$ and $w \in T$. However, since $y \rightsquigarrow u$, we have an edge starting from T and ending in S, which contradicts the assumption that it is a unidirectional node cut. □

Lemma 3 confirms that a strong line cut must be unidirectional and a unidirectional cut must be a strong line cut. Note that the cardinality of a strong line cut and the unidirectional node cut can be different. The reason we convert back from a strong line cut to a node cut (which is unidirectional) is that eventually a node cut is what is mapped onto a cell. A node cut form of a line cut would always require smaller library cell whether mapping is done using standard Boolean functions or threshold functions.

Next we show that restricting node cuts to unidirectional node cuts will not increase the critical path delay when that is the objective function being minimized by TM. Note that in TM, the delay of path is the sum of the delays of gates in the path. We show that the set of paths to the output node v in a bidirectional cut is the same as those in the corresponding unidirectional cut.

Figure A.5(a) shows a classification of the edges in a bidirectional cut. $T = X \cup Y$, where X is the set of logic cones (node and all nodes in its fanin cone) whose output has a directed edge to some node in S, and Y is the set of nodes with no paths to S. Note $v \in Y$. TM would replicate X in S and then replace T with some appropriate cell in the library. This is depicted in Figure A.5(b). Four types of edges can be identified in the $S-T$ partition: (1) E_1 are edges from S to X, (2) E_2 are edges from X to S, (3) E_3 are edges from S to Y, and (4) E_4 are edges from X to Y.

Now a path from input node in S to the output node $v \in T$ can be one three types:

1. $\textcircled{S} \xRightarrow{E_1} X \xRightarrow{E_4} Y \Longrightarrow v$.
2. $\textcircled{S} \xRightarrow{E_2} S \xRightarrow{E_3} Y \Longrightarrow v$.

3. $\textcircled{S} \stackrel{E_3}{\Longrightarrow} Y \Longrightarrow v$.

Note that every one of the above paths (sequence of nodes) in the graph of Figure A.5(a) also exists in the graph shown in Figure A.5(b). Now consider the corresponding unidirectional cut shown in Figure A.5(c). Every path that exists in Figure A.5(a) also exists in Figure A.5(c), and visa versa. This shows that there is no disadvantage to retaining only unidirectional cuts.

A.4 Cut enumeration

Enumerating MISs is a key step in many computational graph theory problems (Eppstein, 2001b; Byskov, 2003, 2004). In TM, because there is a fixed library of functions, MIS enumeration needs to be restricted to sets of size $\leq k$. Without any restrictions, the number of MISs in arbitrary graphs grows exponentially in the size of the graph (Byskov, 2004). However, in TM, the size k of the MIS is bounded above by some constant, and independent of n, which is the size of the graph. Therefore the number of MISs of size $\leq k$ is $\leq n^k$. A brute force approach in which all subsets of size $\leq k$ are examined and those that are not an MIS are discarded, is not practical for even realistic values of n and k. Existing algorithms exploit specific properties of small MISs to facilitate enumeration (Eppstein, 2001b). We now describe a method that can significantly speedup existing MIS enumeration algorithms by pruning away many MISs that will not be part of the final solution.

A.4.1 MIS Pruning

The LDG of a DAG encodes MISs, many of which have sizes $> k$. The basic idea in the pruning algorithm is to (efficiently) transform an LDG into a new, smaller, and denser graph G' which contains all the MISs of size $\leq k$ of the original LDG, and as few other (parasitic) MISs as possible. The objective is to construct a transformation which is computationally efficient and would significantly reduce the runtime of enumeration.

The graph G' to be constructed must satisfy the following conditions: every vertex v of G' as well as every disconnected pair of vertices in G' must independently be a part of some MIS of size $\leq k$ of the original graph G. This condition translates into two steps of the pruning algorithm. In the first step, for each vertex v we attempt to determine the size of the smallest MIS to which v belongs. If this MIS is of size $\leq k$ then v is included in G'. The second step decides if any two disconnected vertices in G may *safely* share an edge in G', implying that they will not be part of any MIS. Again for each pair of disconnected vertices (u, v) we attempt to determine the size of the smallest MIS containing both of the vertices. If such MIS is of size $> k$ then an edge (u, v) is added to G'. This is the key step in the of following pruning algorithm.

There is no known polynomial procedure to compute the size of the smallest MIS that contains a given vertex or a pair of vertices for comparability graphs (Corneil and Perl, 1984). Hence we *approximate* the size of the minimum MIS containing a vertex v or a pair of vertices (u, v) of a given LDG by exploiting the duality between MISs in LDGs and strong line cuts in DAGs. The minimum MIS in an LDG is the minimum strong cut in the DAG. We use this fact to approximate the minimum MIS size in an LDG by means of a flow computation in its corresponding DAG.

Input: A DAG $D(V_N, E_N)$, An LDG $G = (V, E)$ and an integer k
Output: Graph $G' = (V', E')$ characterized above

$dl = \Phi$, $el = \Phi$;
for *vertex* v *in* G **do**
  λ = Min-MIS(D, G, v);
  if $\lambda > k$ **then**
    $dl = dl \bigcup v$;
  end
end
for *disconnected pair* (u, v) *in* G *such that* $u \notin dl$ *and* $v \notin dl$ **do**
  λ = Min-MIS(D, G, u, v);
  if $\lambda > k$ **then**
    $el = el \bigcup (u, v)$;
  end
end
$E' = E \bigcup el$;
$V' = V - dl$;
return $G'(V', E')$;

Algorithm 5: Algorithm to prune MISs of an LDG

It is well known that a minimum s-t cut (min-cut) is equivalent to the maximum flow the sink t receives from the source s (Cormen *et al.*, 2001; Ford and Fulkerson, 1962). The size of a cut is a sum of capacities of edges involved in it. If unit edge capacities are assigned, then the size of a cut is equivalent to number of edges. We note that an edge with a capacity of ∞ can never be part of a finite size cut. The size of the minimum MIS in an LDG containing a vertex v or a pair of vertices (u, v) is approximated by computing the min-cut in the DAG with unit edge capacities. The procedure (Alg. 6) assigns a capacity of ∞ to the dependent lines of the given line (e.g. corresponding to node v) and a capacity of 1 to all other lines. The capacities of edges attached to s and t are always ∞. This is because a min-cut must consist of circuit lines only. Finally it returns the size of the minimum s-t cut of the network (λ).

Input: A DAG $D(V_N, E_N)$, LDG G and a line v
Output: Approximate size of minimum MIS (strong cut) containing v

for u *in* G **do**
  $capacity[u] = 1$;
end
for *neighbor* w *of* v *in* G **do**
  $capacity[w] = \infty$;
end
return Min-cut(D,*capacity*);

Algorithm 6: Min-MIS procedure for single edge in DAG

Lemma 4. *Let S_{min} be the minimum strong cut containing line v. Let λ be the size of a min-cut of the network with capacities modified based on line v. Then $\lambda \leq |S_{min}|$.*

Proof. The Min-MIS procedure never assigns a capacity of ∞ to any line $l \in S_{min}$. Thus $|S_{min}|$ is finite. It immediately follows that S_{min} is an arbitrary s-t cut, and cannot be smaller than the min-cut of the network. □

Lemma 4 states that the size of the minimum strong cut is guaranteed to be greater than or equal to the size of a min-cut. Hence if the result of Min-MIS is $> k$ then the size of a minimum strong cut is also $> k$. Consequently, the vertex in G for which Min-MIS was computed to be $> k$ can be safely discarded from G', or an unconnected pair of vertices for which Min-MIS was computed can be connected in G'.

As an example, consider the LDG in Fig. A.4(b), and suppose we wish to check whether vertex p belongs to an MIS of size ≤ 2. This is equivalent to determining if line p belongs to minimum strong cut of size at most 2 in the DAG. We assign capacities to the edges, as shown in Fig. A.6(a). Line p is assigned a capacity of 1 and its dependent lines, t and v, are assigned a capacity of ∞. After this, the s-t minimum cut size (λ) is determined. In this example, it is lines p, q and u, and its size is 3 — i.e. $\lambda = 3$.

Theorem 5. *Let S be an MIS in G such that $|S| \leq k$. Then S is also an MIS in G'.*

Proof. From Lemma 4 every vertex of S must be present in G' and that no edge was added between any two of its vertices. Thus S is an independent set in G'. Assuming S is not a MIS in G', there exists an independent set S' in G' such that $S \subset S'$. It follows that S' is also an independent set in G contradicting the fact that S is an MIS in G. □

Lemma 6. *Pruning algorithm runs in $O(n^3)$.*

Proof. Let n be the number of nodes, m be the number of edges in DAG. The second *for* loop in pruning algorithm runs in $O(m^2)$ and dominates the overall complexity. Determining the min-cut takes $O(km)$ time (Cormen *et al.*, 2001). Since $k \leq n$ and is independent of n, the pruning has fixed complexity of $O(m^3)$. We know that $m \leq \Delta n$ where Δ is maximum degree found in the DAG. For most of the circuits with limited fanin (and fanout) capacities Δ can be regarded as a small constant independent of n. Hence the time complexity of the pruning procedure is $O(n^3)$. □

As a result of the pruning transformation we perform MIS enumeration on G' instead of G. Note that not all MISs in G' are of size k in G. However, our experiments demonstrate that using G' instead of G to enumerate MISs significantly reduces the enumeration time. In fact, the enumeration runtimes we observe in our experiments are practical for all of the evaluated benchmark circuits, suggesting that the approximation of a *Min-MIS* used in pruning must be quite accurate.

A.4.2 Enumerating MISs

As stated earlier, there are many known MIS enumeration techniques in computational graph theory. In fact, the choice of enumeration algorithm is independent of MISs pruning. In our experiments, we used a recursive procedure that is basically a simplified form of the algorithm presented in (Byskov, 2003). The idea is to recursively enumerate MISs that contain a specific node and then those that do not contain that same node.

Note, that due to the pruning transformation being approximate, G' may still contain some MISs of size $> k$. Since we are interested only in maximal independent sets of size $\leq k$, we simply discard, on the fly, the MISs whose size is greater than k.

Unfortunately, the number of MISs of a graph increases exponentially as a function of its size (Moon and Moser, 1965). However in practice we found that while running it on a pruned graph G', enumeration time is dominated by pruning time for sufficiently small k. Hence, even the simple recursive algorithm that we used is still efficient. More sophisticated approaches to enumerate MISs of size $\leq k$ (Byskov, 2003; Eppstein, 2001b) could be used to improve runtime for with values of k. Their application on the transformed graph may further improve total runtimes as well as scalability of the solution.

A.4.3 Results

The procedures described in this chapter were implemented in C++ and run on a 2GHz PC with 2 GB of RAM. The results of these runs are summarized in Table A.1. We ran the simple cut enumeration algorithm after pruning the MISs, and enumerated all cuts in the ISCAS benchmark circuits. The starting point was an AND-INVERTER Graph (AIG) obtained from (Mishchenko *et al.*, 2007). Note that MIS enumeration is not possible with any of the existing MIS enumeration schemes. In fact, even a graph with as few as a hundred nodes would not be practical.

The proposed pruning procedure makes it possible to exhaustively enumerate large numbers of strong line cuts in reasonable time. We also evaluated the combined effect of constraining the cone size and increasing the value of k. The results demonstrate that for sufficiently small k, line cut enumeration is dominated by our pruning transformation and as such is practically polynomial. For larger values of k however the procedure could however benefit from a more efficient MIS enumeration procedures.

This chapter presented a novel cut enumeration framework that exploits duality between enumerable entities in DAGs and LDGs. Apart from resource efficient computational procedure, it also introduces into the area of technology mapping, the concept of k-feasible strong line cuts (or unidirectional cuts) that are distinct from conventional node cuts. The advantages are two-fold. On one hand they are enumerable with low per-unit computational effort. On the other hand they potentially open up a new space available to be explored by the technology mapper without degrading the quality of the mapping. Line cuts provide choices not available to node cut based technology mappers. More importantly line cuts inherently mitigate some of the structural bias of node cuts and unlike node cuts they guarantee a mappable cut of a circuit.

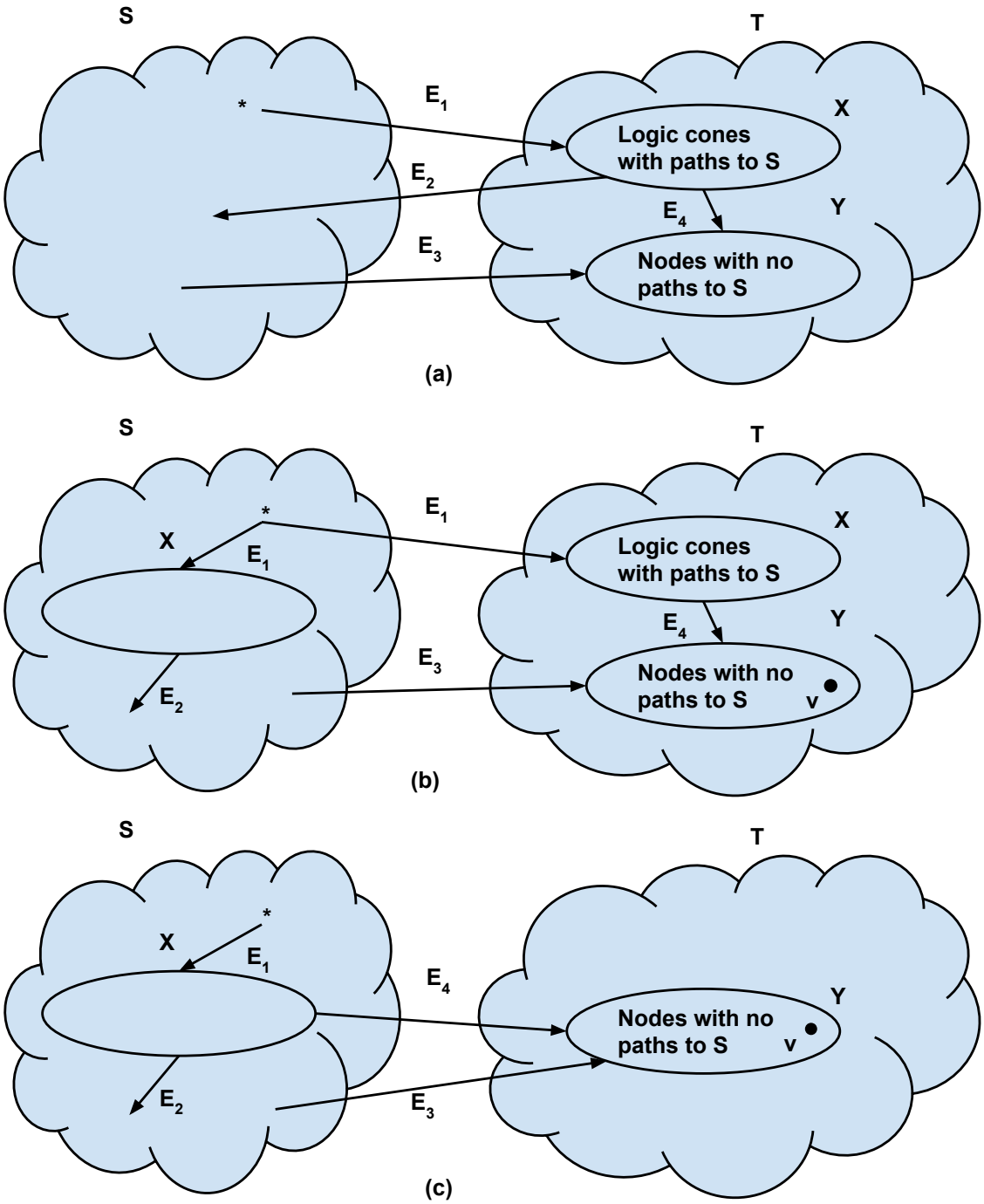


Figure A.5: a) Classification of Edges in a Bidirectional Cut b) Replication after TM, (c) Classification of Edges in Corresponding Unidirectional Cut.

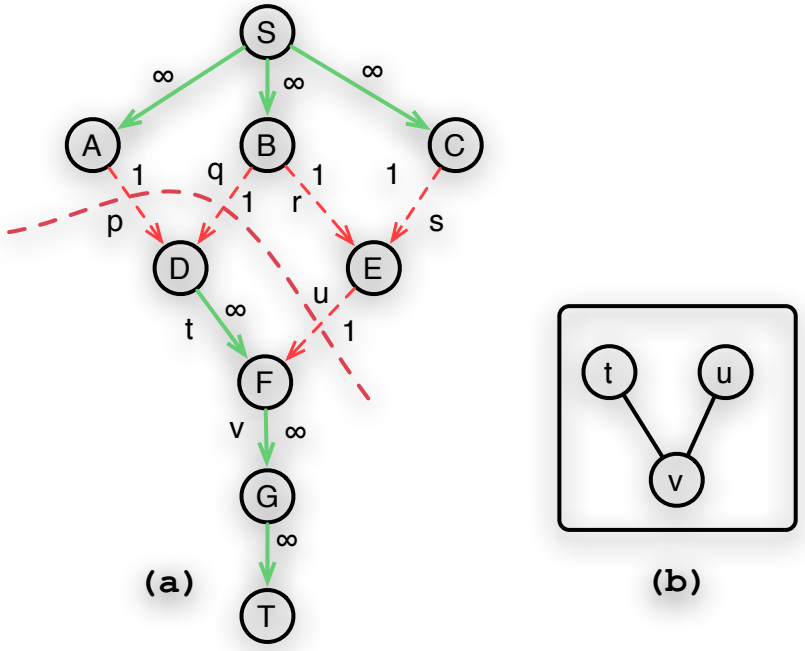


Figure A.6: a) Formation of $s - t$ Boolean Network for Determination of a Cut Containing Line p. b) LDG from Fig. A.4 Pruned for $k \leq 2$.

Table A.1: Running Times for Enumeration

			$K = 6$, cone size = no limit			$K = 6$, cone size = 300			$K = 10$, cone size = 100		
circuit	# inputs	# nodes	#cuts after pruning	pruning time (s)	enum. time (s)	#cuts after pruning	pruning time (s)	enum. time (s)	#cuts after pruning	pruning time (s)	enum. time (s)
c432	36	356	4317	0.93	0.02	4253	0.98	0.02	382,697	2.22	1.60
c1355	41	1133	228,950	49.04	3.22	129,994	15	2.5	26,194,458	29.25	286.06
c1908	33	1793	9,519,182	48.12	9.43	2,099,375	18.49	5.02	1,311,442,759	36.46	6510.5
c6288	32	4864	1,494,636	6085.06	190.86	737,587	480.83	63.96	150,921,850	379.75	4570.26
c7552	207	7233	5,949,016	182.96	17.43	2,521,039	55.48	10.75	3,113,268,991	193.59	7046.68