# LOW-VOLTAGE SINGLE-PHASE CLOCKED QUASI-ADIABATIC PASS-GATE LOGIC FAMILY FOR LOW-POWER SUBMICRON VLSI CMOS APPLICATIONS

by

Edward K. W. Loo
B.A.Sc (First Class Honors), Simon Fraser University, 2005

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In the
School
of
Engineering Science



SIMON FRASER UNIVERSITY

2007

# APPROVAL

**Name:** **Edward Loo**

**Degree:** **Master of Applied Science**

**Title of Thesis:** **Low-Voltage Single-Phase Clocked Quasi-Adiabatic Pass-Gate Logic Family for Low-Power Submicron VLSI CMOS Applications**

**Examining Committee:**

**Chair:** **Dr. Faisal Beg**
Assistant Professor of Engineering Science

---

**Dr. Marek Syrzycki**
Co-Senior Supervisor
Professor of Engineering Science

---

**Dr. James B. Kuo**
Co-Senior Supervisor
Adjunct Professor of Engineering Science

---

**Dr. Rick Hobson**
Examiner
Professor of Engineering Science

**Date Defended/Approved:** Wednesday, July 18, 2007

# ABSTRACT

As the growing complexity of mobile electronic applications leads to prohibitively high chip power demands, the energy efficiency of the integrated circuit devices will become more significant. Energy recovering circuitry based on adiabatic principles is a relatively new technique used to implement low energy dissipating circuits. By recycling the charge stored at capacitive nodes in the circuit, adiabatic logic families can achieve very low energy dissipation. This thesis presents a novel low-voltage Quasi-Adiabatic Pass-Gate (QAPG) logic family using a single power clock in 90nm CMOS technology. A comparative analysis is performed where circuits are constructed using previously proposed low-power single-phase clocked adiabatic logic and QAPG logic. Simulations demonstrate that the new logic family is suitable for low voltage operation down to 0.25V and down to the 32nm CMOS technology node. QAPG dissipates between eleven and forty percent of the total energy consumed by the previously proposed adiabatic logic families.

**Keywords:** adiabatic logic; low-voltage CMOS; low-energy; energy recovery

**Subject Terms:** Low voltage integrated circuits; Metal oxide semiconductors, Complementary -- Design and construction; Integrated circuits -- Very large scale integration -- Design and construction

*To my family...*

# ACKNOWLEDGEMENTS

I would like to express my appreciation and gratitude to Dr. Marek Syrzycki for all the generosity, advice, and guidance given to me during my research. Your knowledge and ideas were truly helpful. Without your help, this thesis would never have existed.

I would also like to thank Dr. James B. Kuo for pointing me in this direction of research, for your continued support, and for being a great source of information. I have learned a great deal from my time working with you which I would not have learned elsewhere.

I would also like to thank Dr. Rick Hobson for his valuable advice on this work. It was a pleasure to learn from you during my undergraduate studies.

Special thanks to Benjamin Wang for his help with my layout.

Finally, I would like to thank my family for supporting me all these years and especially my sister who read through this for me. Also, thanks to my friends and labmates, Harry and Henry, for making university life bearable.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ACRONYMS

| | |
|---|---|
| ADCVS | Adiabatic Differential Cascode Voltage Switch |
| ADL | Adiabatic Dynamic Logic |
| ADSL | Adiabatic Differential Switch Logic |
| ASIC | Application-Specific Integrated Circuit |
| CAL | Clocked Adiabatic Logic |
| CLA | Carry-Lookahead Adder |
| CMC | Canadian Microelectronics Corporation |
| CMOS | Complementary Metal Oxide Semiconductor |
| IC | Integrated Circuit |
| ITRS | International Technology Roadmap for Semiconductors |
| KVL | Kirchhoff's Voltage Law |
| MOSFET | Metal-Oxide-Semiconductor Field Effect Transistor |
| MPU | Main Processing Unit |
| PFAL | Positive Feedback Adiabatic Logic |
| PTM | Predictive Technology Model |
| QAPG | Quasi-Adiabatic Pass-Gate Logic |
| SCAL-D | Source-Coupled Adiabatic Logic with Diode-Connected Transistors |
| SCRL | Split-Level Charge Recovery Logic |
| SOC | System-on-Chip |

| | |
|---|---|
| TSPC | True Single-Phase Clocking |
| VLSI | Very-Large Scale Integration |

# CHAPTER 1: INTRODUCTION

## 1.1 Trends in VLSI CMOS

Recent trends in digital electronics are moving towards high performance, highly functional portable consumer applications. This results in increasing clock frequencies, downscaling of transistor sizes, and increasing transistor counts, all of which boost energy demands in new System-on-Chip (SOC) designs. A summary of trends in the semiconductor industry for high performance Main Processing Units (MPU) and Application-Specific Integrated Circuits (ASIC) that are optimized for maximum speed performance are shown below in Table 1-1 [1,2,3].

**Table 1-1: Semiconductor technology roadmap**

| Production Year | **2007** | **2010** | **2013** | **2016** | **2020** |
|---|---|---|---|---|---|
| Technology Node (nm) | 65 | 45 | 32 | 22 | 14 |
| Physical Gate Length (nm) | 25 | 18 | 13 | 9 | 6 |
| Supply Voltage (V) | 0.8 - 1.1 | 0.7 - 1.0 | 0.6 - 0.9 | 0.5 - 0.8 | 0.5 - 0.7 |
| Functions per chip (million transistors) | 553 | 2212 | 4424 | 8848 | 17696 |
| On-chip local Clock (MHz) | 9783 | 15079 | 22980 | 39683 | 73122 |
| Maximum Allowable Power Dissipation with heat-sink (W) | 189 | 198 | 198 | 198 | 198 |

Future semiconductor processes and design methodologies must be developed to satisfy the trend of increasing clock frequencies, shorter channel lengths, and increasing transistor counts. The predicted transistor count presented in Table 1-1 roughly coincides with Moore's Law [4], which states that transistor counts will double every 18 months.

Demands for additional battery life coupled with the escalation in energy demands generate an energy dilemma for circuit designers. According to the International Technology Roadmap for Semiconductors (ITRS) [1], the maximum allowable power for a high-performance device when using a heat-sink will be 198 W before chip failure. As well, growing environmental awareness in energy efficient products increases the demand to minimize energy dissipation in electronic goods. Taking all of the above factors into account, research into low energy circuit techniques has become a very active discipline in academia and industry. Furthermore, the minimization of energy is now a major design requirement in any SOC design.

## 1.2 Energy Dissipation in VLSI

Energy dissipation in VLSI CMOS consists of two forms: static and dynamic energy [5]. Static energy dissipation consists of sub-threshold currents of MOS transistors. An example shown in Figure 1-1 indicates sub-threshold current within a CMOS inverter.

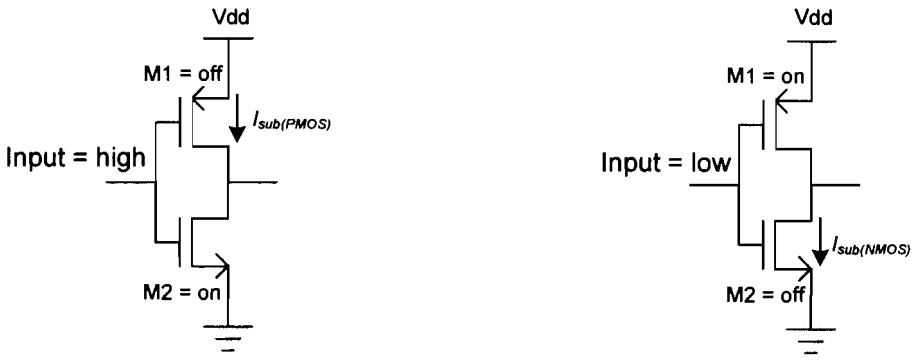



**Figure 1-1: Sub-threshold current within a CMOS inverter**

In Figure 1-1, when the input node is at logic high, **M1** is operating in the cut-off region and **M2** is operating in the saturation region. Ideally, there is no current flowing through **M1**, however due to sub-threshold (leakage) current, $I_{sub(PMOS)}$, there is power dissipated in the form of:

$$P_{static} = I_{sub} V_{dd}$$

and energy dissipated in the form of:

$$E_{static} = \int_0^t P_{static} dt \, .$$

The situation is similar when the input is at logic low (**M1** is in saturation, **M2** is in cut-off) with **M2** contributing to the sub-threshold current, $I_{sub(NMOS)}$.

Dynamic energy dissipation results from the switching of the input signals to a logic gate. In conventional CMOS logic, there are two components to

dynamic energy dissipation: short-circuit energy & switching energy. The momentary short circuit between the voltage source and ground during switching events gives rise to a short circuit current. This can be modelled by replacing $I_{sub}$ with $I_{sc}$, and a logic transition at the input node in Figure 1-1. This accounts for a small percentage of dynamic energy.

The majority of dynamic energy dissipation is due to switching activity found within a circuit. A charge of $Q = CV_{dd}$ (where $C$ is the capacitance of the circuit node and $V_{dd}$ is the operating voltage) is transferred to a node within a CMOS circuit during a change in logic states. This involves the transferring of an amount of energy, $E = \frac{1}{2}CV_{dd}^2$ to the capacitance of the node while another $E = \frac{1}{2}CV_{dd}^2$ in energy is dissipated as heat through internal resistances during this operation. During a discharge operation at this node, the energy stored at this node is dissipated. In total, an amount of energy, $E = CV_{dd}^2$, disappears from the power supply.

Figure 1-2 shows an example of dynamic energy dissipation in CMOS logic.

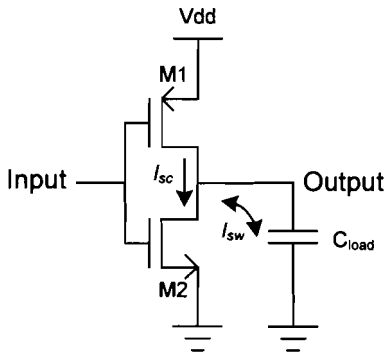
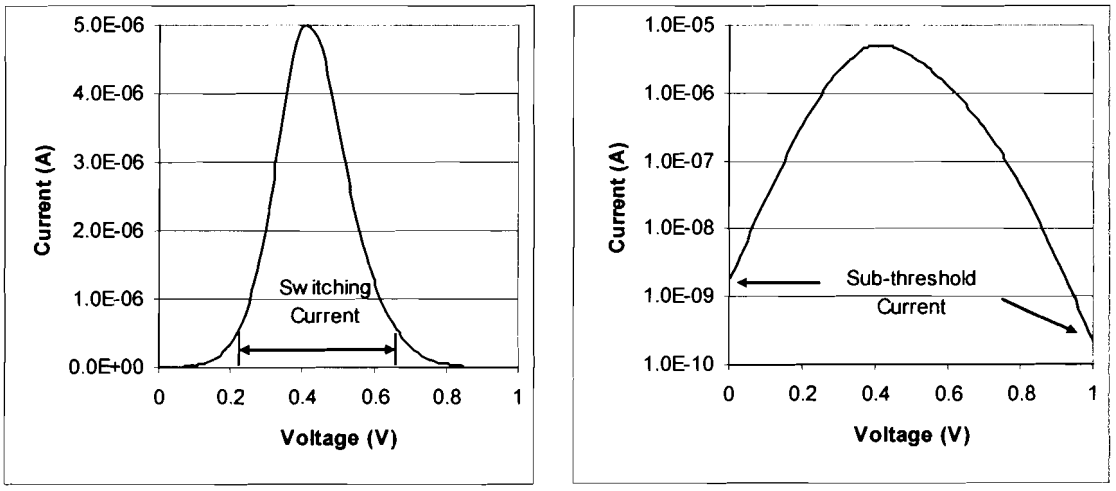



**Figure 1-2: Dynamic energy dissipation in a CMOS inverter**

Sub-threshold and switching currents of the CMOS inverter are shown in Figure 1-3.




Regular Scale

Logarithmic Scale

**Figure 1-3: CMOS inverter currents**

Conventional methods in lowering energy dissipation include minimizing capacitances by decreasing transistor sizes, lowering supply voltages, and reducing switching activity [6, 7, 11]. Each of these techniques would lower the dissipated dynamic energy as shown in the dynamic power dissipation equation:

$$P = \alpha C_{load} V_{dd}^{\ 2} f_{clk},$$

where $\alpha$ is the switching factor of the node, $C_{load}$ is the capacitance at the node, and $f_{clk}$ is the system clock frequency. Energy reduction using these methods are beneficial, however each has its limit. Switching activity can only be reduced to a certain point dependent upon the circuit. Scaling of transistors sizes can only be minimized to a certain process limit. The downscaling of CMOS technology nodes leads to decreasing supply voltages and decreasing threshold voltages which will increase leakage current [8]. Voltage reduction is a preferred method in lowering energy dissipation since a decrease in voltage would have a quadratic reduction in energy dissipation. However, there are many challenges associated with lowering supply voltages such as decreased noise margins, lowered driving capabilities, and reduced switching speeds [9].

Popular techniques in reducing energy consumption include the use of multiple threshold voltages to reduce static (leakage) energy, which is becoming more significant as CMOS technology continues to scale down [8, 27]. Multiple threshold voltages permit the use of higher threshold voltages to reduce leakage in non-critical timing paths while the lower threshold voltage is used to maintain timing performance. Multiple threshold voltages increase the number of process

steps which increase the cost of fabrication, but this is necessary to keep power down to an acceptable limit. Other ways in decreasing leakage power is to implement a body biasing technique (variable threshold voltages) to control the leakage during circuit operation. However, this again increases costs as it requires triple-well technology or other advanced fabrication techniques.

A different approach in reducing energy dissipation is to implement charge-recovery circuits. Sometimes referred to as adiabatic logic from its definition in thermodynamics [10], an adiabatic process occurs without the gain or loss of heat. In electronics, energy/power dissipation is analogous with heat. The goal of adiabatic computing is to obtain virtually no loss of energy within the system. The basic idea is that the energy is sourced and then recovered from the nodes within the circuit [12].

## 1.3 Goals of this Work

Low-energy and low-voltage circuits are important for many reasons. Energy consumption of electronic devices strains the power generation infrastructure. Heat dissipation on high-performance ICs is becoming more challenging. As well, reduced energy consumption in portable devices adds to the mobility and utility of the device.

The difference in energy and power is important. Power is energy per unit time. The overall goal of any circuit designer is to lower the power consumption for the device. This could be done simply by reducing its clock rate. However, this is not acceptable, as the time required to complete a computation will

increase since the number of clock cycles required remains the same. Clock rate reduction results in the same amount of energy consumed as if the computation was done at a faster speed as the energy per computation remains the same. The goal of a circuit is to be high-speed in addition to being low-power. In other words, the goal of a circuit is to be low energy [11].

The goal of this work was to develop a low-energy charge-recovering logic family based on adiabatic principles to minimize energy dissipation within each gate. The low-voltage single-phased clocked Quasi-Adiabatic Pass-Gate Logic (QAPG) family has been designed for low-voltage, low-energy, high-speed operation. The single-phased clocking design requires a single power clock for proper operation of the circuit. Designed using Canadian Microelectronics Corporation's (CMC) 90 nm CMOS technology, HSPICE simulations were performed on various logic gates as well as on a carry-lookahead adder (CLA). A comparative analysis was performed in which logic gates were constructed using previously proposed low-power single-phase clocked adiabatic logic families and the newly proposed QAPG logic family for the implementation of a set of adders. A layout-based simulation was then performed to verify the operation of the QAPG family when including parasitic components.

## 1.4 Organization of the Thesis

The remainder of this thesis is divided into four chapters. Chapter 2 provides a brief overview of the evolution of adiabatic logic. First, a background into adiabatic circuitry and the importance of the power clock generator will be

discussed. Then the progression of multi-phase clocking families to single-phase clocking families will be shown.

Chapter 3 presents the QAPG family. Descriptions of operation as well as various gates and simulation results will be presented and compared against previous proposed adiabatic logic families. A layout of an 8-bit CLA is presented in Chapter 4 and is compared to simulation based results in Chapter 3. A summation of the research results is offered in Chapter 5.

# CHAPTER 2: EVOLUTION OF ADIABATIC CIRCUITRY

Adiabatic circuitry is a very attractive way of lowering energy consumption in digital logic. Involving the recovery of charge in a circuit, adiabatic logic is a very different method in reducing energy in comparison to more traditional methods such as transistor scaling and supply voltage lowering. In this chapter, we discuss the progression of adiabatic circuitry, beginning first with the fundamentals of adiabatic circuitry and then moving on to previously proposed adiabatic logic families.

## 2.1 Fundamentals of Adiabatic Circuitry

Adiabatic logic accomplishes two things when reducing energy consumption; it recycles charge and it reduces dissipated energy through resistive components. To recognize the savings in energy, conventional static CMOS logic has been analyzed in order to observe its shortcomings.

### 2.1.1 Non-Adiabatic Logic

Conventional static CMOS logic is considered to be non-adiabatic logic since charge is only sourced from the power supply and all the energy supplied is dissipated within the circuit. An example of the conventional static CMOS inverter is shown in Figure 2-1.

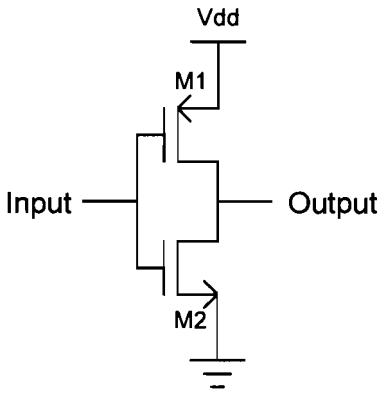



**Figure 2-1: Conventional static CMOS inverter**

Currently, the charging of capacitive nodes is the prevailing factor in energy dissipation of static CMOS logic. As discussed previously, during a charging of the output node, the total energy supplied by the power source is $E = CV_{dd}^2$. Due to the immediate switching on of **M1**, a high potential drop appears across the transistor resulting in an energy of $E = \frac{1}{2}CV_{dd}^2$ being dissipated across its on-resistance. Eventually, the capacitive output node will be evaluated to a logic low and the energy stored in the output nodal capacitance will be dissipated by the on-resistance of **M2**. The total amount of energy sourced by the power supply is dissipated entirely.

### 2.1.2 Adiabatic Logic

Contrary to conventional static CMOS logic, adiabatic logic sources and recovers energy from nodes within the logic circuits while attempting to minimize the energy dissipated by the on-resistance of the MOSFETs. By striving to

minimize resistive dissipation, more energy will be available for recovery by the power supply to be supplied again in the following clock phases. The recycling of energy and minimization of dissipation are the two fundamental aspects of adiabatic circuitry.

#### 2.1.2.1 Adiabatic Charging and Recycling

The recycling of charge in an adiabatic circuit is based on resonance. An adiabatic circuit is designed in a way such that it can be modeled as a series RLC resonant circuit as shown below in Figure 2-2.

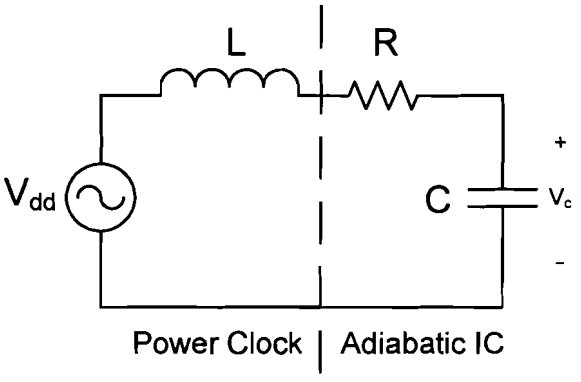



**Figure 2-2: RLC circuit**

The resonant oscillation allows the charge to travel to and from the adiabatic circuit, performing the charging and recycling. The entire adiabatic device can be divided into two components: the power clock, which contains the inductor and the oscillating power supply, and the adiabatic circuitry, which is made up of the logic circuits. The resistor, **R,** in Figure 2-2 represents the transistor on-resistances and the interconnect resistances, and the capacitor, **C**,

represents the nodal and interconnect capacitances of the adiabatic circuitry. The inductor, **L**, in the power clock is then chosen to tune the resonant frequency of the circuit, which will be the operational frequency.

Using the Kirchhoff's Voltage Law (KVL) to solve this circuit in the complex frequency domain, we obtain,

$$V(s) = I(s)\left(Ls + R + 1/Cs\right) \Rightarrow \frac{I(s)}{V(s)} = \frac{1}{Ls + R + 1/Cs}.$$

Solving for the transfer function for the voltage at the capacitor **C**, we obtain,

$$\frac{V_c(s)}{V(s)} = \frac{1/Cs}{Ls + R + 1/Cs},$$

where $I(s) = I_c(s) = V_c(s)Cs$.

Upon further manipulation, we have the transfer function in the form of a general second order equation,

$$\frac{V_c(s)}{V(s)} = \frac{1/LC}{s^2 + \frac{R}{L}s + 1/LC} \Rightarrow \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}.$$

Key parameters in the equation are the natural undamped frequency of oscillation, $\omega_n$ and the damping factor ratio $\zeta$. In the series RLC circuit, these parameters are:

$$\omega_n = \frac{1}{\sqrt{LC}},$$

$$\zeta = \frac{R}{2L\omega_n} = \frac{R}{2}\sqrt{\frac{C}{L}}.$$

We can see that if **R**=0, there would be no damping and therefore no loss within the circuit. However, this is not realistic. Due to the damping, the resonant frequency will shift due to,

$$\omega_d = \omega_n \sqrt{1-\zeta^2},$$

where $\omega_d$ is the damped frequency of oscillation. In adiabatic circuitry, $\omega_n >> \zeta$ & $\zeta$ <1, therefore,

$$\omega_d \cong \omega_n.$$

Adiabatic power clocks have been proposed in the past [14, 15, 16] for low-energy circuitry. These power clocks can be integrated into the same chip as the circuitry to simplify implementation. An example of a sinusoidal power clock generator is shown in Figure 2-3.

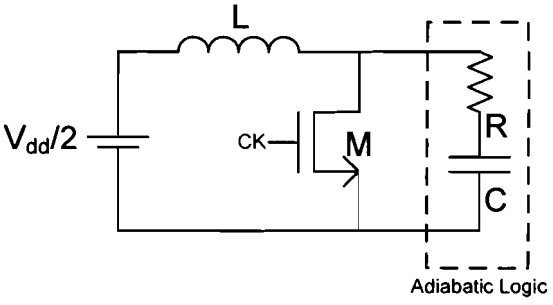



**Figure 2-3: 1N single-phase power clock generator**

The 1N single-phase power clock generator is a simple power clock generator [15] which consists of a single NMOS active device in parallel with the adiabatic logic. The inductor, **L**, can be implemented on-chip or off-chip. The NMOS device, **M**, is turned on in brief intervals to add energy lost in the RLC resonant tank.

In addition, a 1N1P single-phase power clock generator [16], shown in Figure 2-4, was proposed to generate trapezoidal ramp power clock waveforms. The PMOS, **M1**, and NMOS, **M2**, are both turned on for brief intervals which again add energy lost in the RLC resonant tank.

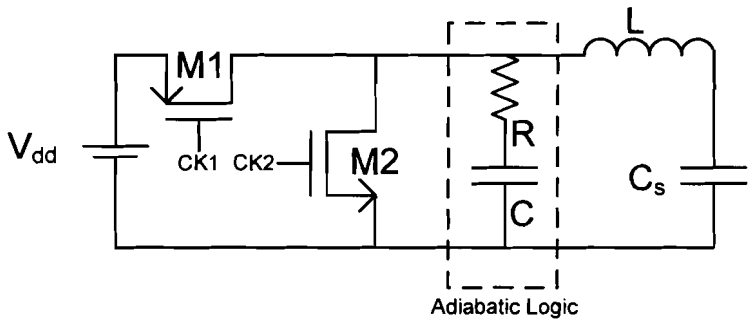



**Figure 2-4: 1N1P single-phase power clock generator**

By matching the resonant frequency of these types of power clock generators to the frequency of operation of the adiabatic circuit, we can achieve a nearly ideal charge recovery system that lowers energy consumption in digital circuitry.

### 2.1.2.2 Resistive Energy Dissipation Minimization

The second aspect of adiabatic logic is to reduce energy dissipation due to the on-resistance of transistors. As previously stated, in conventional CMOS logic, the majority of the dissipated dynamic energy is due to transitions at capacitive nodes in logic circuits. Figure 2-5 models the transition at a circuit node.

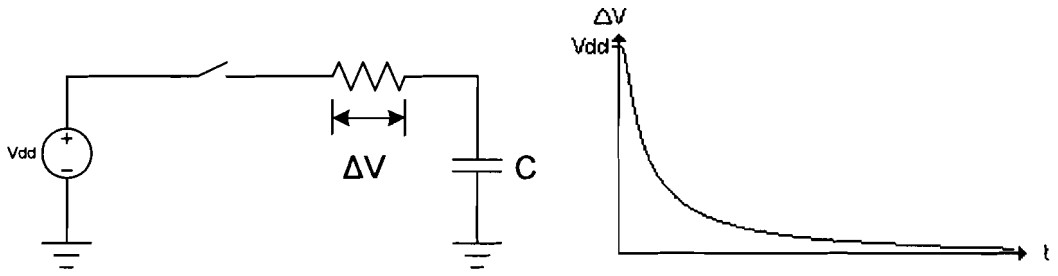



**Figure 2-5: RC circuit charging example**

The resistor in Figure 2-5 represents the resistance of the devices in the circuit while the capacitor represents the capacitance at the nodes of the circuit. Once the switch is closed, there is a sudden spike in current through the resistor and the capacitor is charged to the supply voltage. There is a large potential difference which is seen across the resistor. This results in a large amount of dissipated energy. The energy expended by the power supply is $E = CV_{dd}^2$. However, only $E = \frac{1}{2}CV_{dd}^2$ is stored in the capacitor. The resistor dissipates the other half of the power. Taking the operating frequency and switching activity of

the node into consideration, the dynamic power dissipated at the node is given by [5]:

$$P = \alpha C_{load} V_{dd}{}^2 f_{clk},$$

where again, $\alpha$ is the switching factor of the node, $C_{load}$ is the capacitance at the node, and $f_{clk}$ is the system clock frequency.

If the charge is moved gradually instead of instantaneously, it would lead to lower power dissipation when sourcing or recovering charge at circuit nodes as seen in Figure 2-6

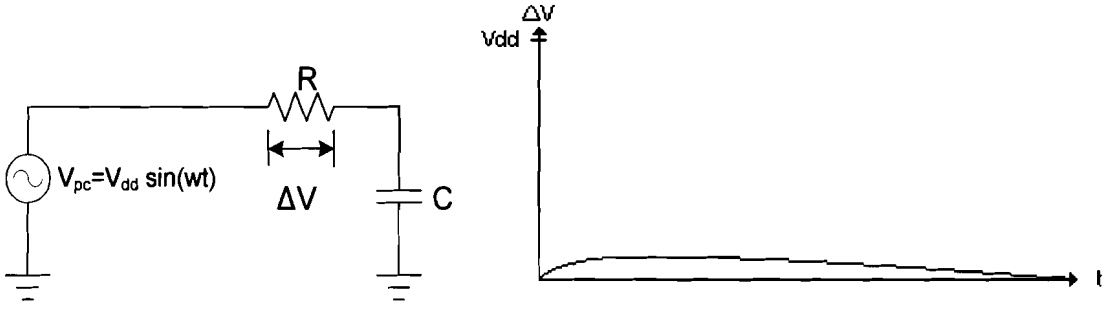



**Figure 2-6: Gradual charging of a capacitive node**

Figure 2-6 shows the previous RC circuit with a slow-changing sinusoidal voltage supply. Instead of an instantaneous switching on of the circuit, it is powered by a sinusoidal input. The potential difference across the resistor is much lower compared to Figure 2-5. This leads to lower energy dissipation [21]. Assuming a constant current source delivering charge, $Q = CV_{dd}$, for period *T*, the improvement in energy can be shown as [12]:

$$E_{diss} = PT = I^2 RT = \left(\frac{C_L V_{dd}}{T}\right)^2 RT = \left(\frac{RC_L}{T}\right) C_L V_{dd}^2 .$$

As seen from the equations above, by increasing the period, *T*, of the sinusoidal voltage supply beyond the *RC* constant of the circuit, one can achieve a decrease in resistive energy dissipation. This is in contrast with conventional CMOS, where currents regularly flow through potential differences of $V_{dd}$. In adiabatic circuitry, the power clock transitions are intentionally slowed down to minimize energy dissipation. This is usually accomplished by implementing a trapezoidal or a sinusoidal power clock waveform signal. These signals can be approximated using the resonant RLC oscillator that was discussed in the previous section.

### 2.1.3 Classification of Adiabatic Logic

Adiabatic circuits can be classified into two categories: fully-adiabatic and quasi-adiabatic. Fully-adiabatic circuits dissipate virtually zero energy by operating at very low frequencies, which would practically eliminate resistive dissipation. A well known fully-adiabatic logic family, split-level charge recovery logic (SCRL) [13], is shown below in Figure 2-7.

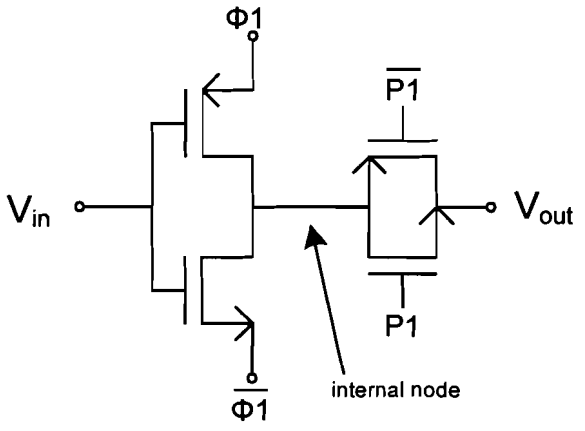



**Figure 2-7: Split-level charge recovery logic inverter [11]**

Fully-adiabatic logic stems from reversible computing, meaning the previous logic states of nodes are known in order to reverse the computation to its initial state [11]. Previous logic states must be known in order to eliminate potential differences across transistors which would cause resistive energy dissipation. In the split-level charge recovery logic shown in Figure 2-7, the initial state of the nodes and $\Phi 1$ & $\overline{\Phi 1}$ are at $V_{dd}/2$. During evaluation, $\Phi 1$ and $\overline{\Phi 1}$ become $V_{dd}$ and GND respectively, and determine the output which is passed through the transmission gate. The transmission gate is then turned off and the output is held while $\Phi 1$ and $\overline{\Phi 1}$ and the internal node is reset to $V_{dd}/2$. The output node is passed on to the following stage and then must be reset to $V_{dd}/2$ by an inverse function of the following stage as shown in Figure 2-8.

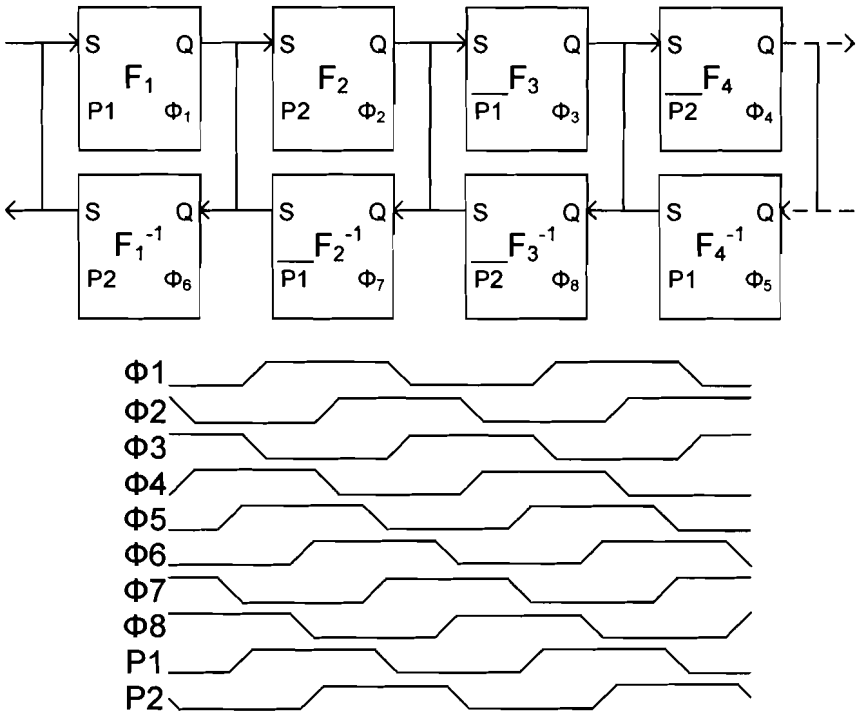



**Figure 2-8: SCRL pipeline and timing diagram [11]**

A pipeline of cascaded SCRL gates requires inverse functions that increase the complexity of fully adiabatic logic families. The number of logic gates is no less than double in order to maintain reversibility. In addition, the sheer number of power clocks required to implement this logic family produces many issues such as clock skewing, individualized power clock tuning and routing difficulties. With these drawbacks in mind, fully-adiabatic logic circuits are not a preferable solution to the energy crisis.

Alternatively, partially-adiabatic or quasi-adiabatic logic provides a favourable solution by balancing implementation issues with energy reduction. Quasi-adiabatic logic transfers charge across reduced potential drops as it tends

to operate at higher clock rates while recovering a considerable amount of nodal charge. Furthermore, quasi-adiabatic logic is easier to implement, as there are less power clocks to employ and it does not require inverse functions to retain reversibility as in the case of fully-adiabatic logic. For these reasons, in recent years researchers have mainly focused on quasi-adiabatic circuitry, although most of them shortened the classification to simply "adiabatic circuitry". These adiabatic logic families will be discussed in the following sections.

## 2.2 Multi-Phase Clocked Adiabatic Logic Families

Earlier work in adiabatic logic revolved around the use of multi-phase clocking schemes to implement the cascading of subsequent logic blocks. This was done to ensure that the previously evaluated logic output was still valid at the input of the following stage of the pipeline. The clocking structure and the required four power clocks are shown below in Figure 2-9. The trapezoidal waveform shown here may vary in shape (i.e. triangular, sinusoidal, etc...) dependent on the design of the adiabatic family.

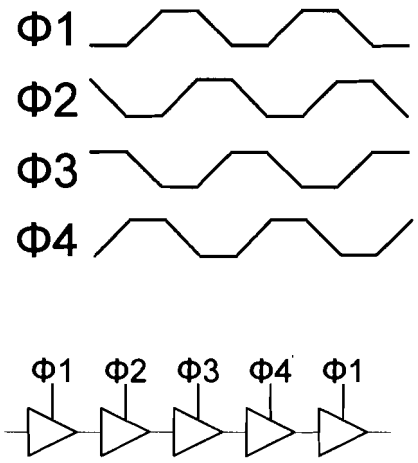



**Figure 2-9: Multi-phase clocking structure [20]**

Many multi-phase clocked adiabatic logic families were proposed and they are discussed in the following sections.

### 2.2.1 Adiabatic Dynamic Logic

Adiabatic Dynamic Logic (ADL) was one of the first proposed adiabatic logic families [17]. An ADL inverter is shown below in Figure 2-10.

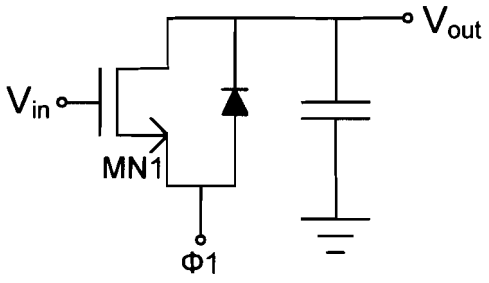



**Figure 2-10: ADL inverter [17]**

All the power is supplied from the triangular power clock $\Phi 1$. During the rising edge of $\Phi 1$, the output is driven to logic high. During the falling edge of the power clock, $V_{in}$ will either cause the output to discharge to logic low, or the logic high value to be maintained by the nodal capacitance at the output. The circuit is cascaded with four clock stages as seen in Figure 2-9, however with a triangular waveform instead of a trapezoidal waveform. A limitation of this logic family is the use of the diode for precharging the output. The built-in voltage of the diode will keep the output from fully charging. More importantly, a MOS diode adds non-adiabatic energy dissipation due to its built-in voltage during the charging of the output, no matter how gradual the charging of the output occurs.

### 2.2.2 2N-2P & 2N-2N2P

A 2N-2P inverter [18] is shown below in Figure 2-11.

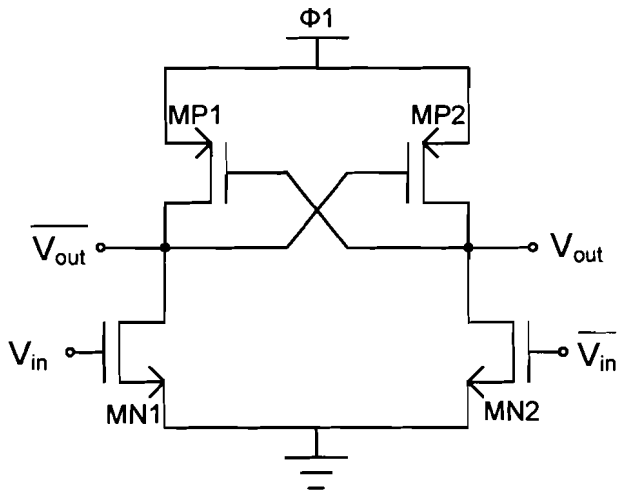



**Figure 2-11: 2N-2P inverter [18]**

2N-2P removes the diodes from previously proposed families (Figure 2-10) which caused large scale energy dissipation. This family utilizes a four phase power clock to control cascades. 2N-2P also presents a balanced capacitive load and has good speed characteristics compared to many other families. As well, 2N-2P gates have differential outputs.

2N-2N2P [18] gates are similar to 2N-2P but contain a pair of cross-coupled NMOS transistors shown in Figure 2-12.

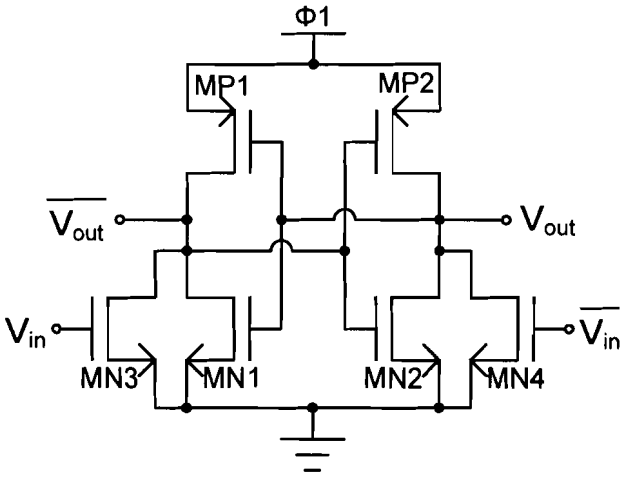



**Figure 2-12: 2N-2N2P inverter [18]**

The advantage of this structure is that the pair of cross-coupled NMOS transistors results in non-floating outputs as compared to the 2N-2P logic family.

### 2.2.3 Positive Feedback Adiabatic Logic

Shown in Figure 2-13, positive feedback adiabatic logic (PFAL) [19] is a modification of 2N-2N2P adiabatic logic.

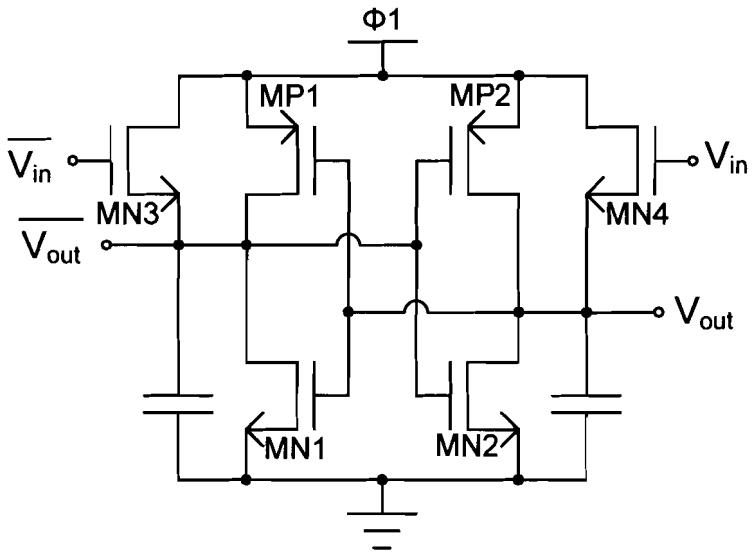



**Figure 2-13: PFAL inverter [19]**

As in 2N-2N2P, PFAL contains a latch made up of cross-coupled PMOS and NMOS transistors. It also utilizes the four phase clocking structure shown in Figure 2-9. However, PFAL differs in that the evaluation trees (**MN3** & **MN4**) are placed in parallel with the cross-coupled PMOS transistors (**MP1** & **MP2**). The main advantage of the PFAL family is that the evaluation trees in parallel with the PMOS transistors lower the overall resistance when charging internal nodes thereby reducing energy dissipation.

### 2.2.4 Adiabatic Differential Switch Logic

Adiabatic differential switch logic (ADSL) [20] was proposed to operate at low supply voltages. It incorporates bootstrapping transistors (**MN3** & **MN4**) to increase gate over-drive voltages which increase the speed of operation at low voltages. In addition, ADSL adds cut-off (**MN5** & **MN6**) and recovery (**MN1** &

**MN2**) transistors to improve the recovery of charge at internal nodes. It is a differential output family that requires a four phase clocking scheme. An ADSL inverter is shown below in Figure 2-14.

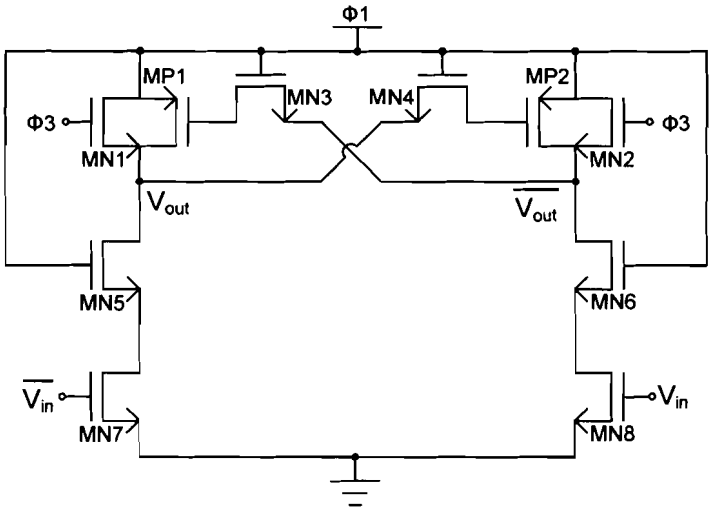



**Figure 2-14: ADSL inverter [20]**

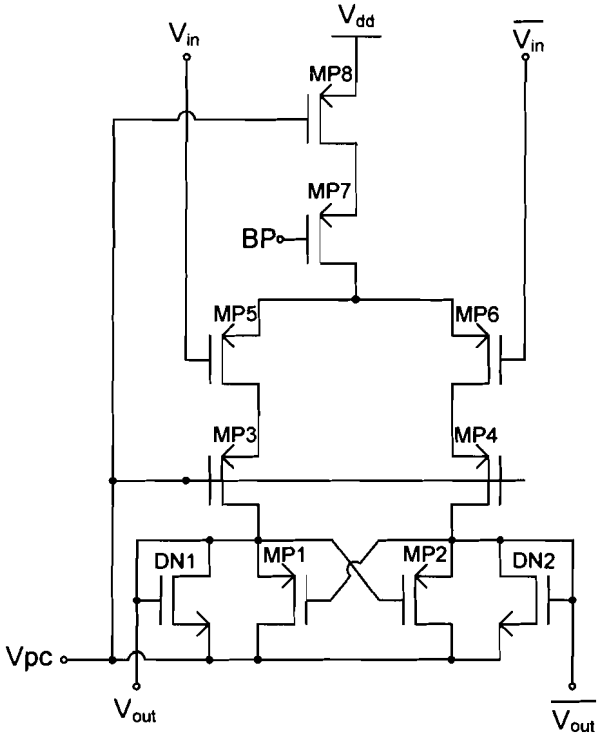
## 2.3 Single-Phase Clocked Adiabatic Logic Families

The use of multi-phase clocking schemes has many drawbacks which will severely affect the efficiency in recovering nodal charges using the resonance circuit structure. Having multiple clocks creates the possibility of clock skew (phase-shift), which will cause timing issues within the pipeline. Evaluation phases may overlap and outputs may not be valid for subsequent stages in the pipeline. Resonant tuning of each clock is more difficult due to the lack of symmetry in capacitances when looking into the circuit from the point of view of each clock. Furthermore, the sheer number of clocks requires complex clock

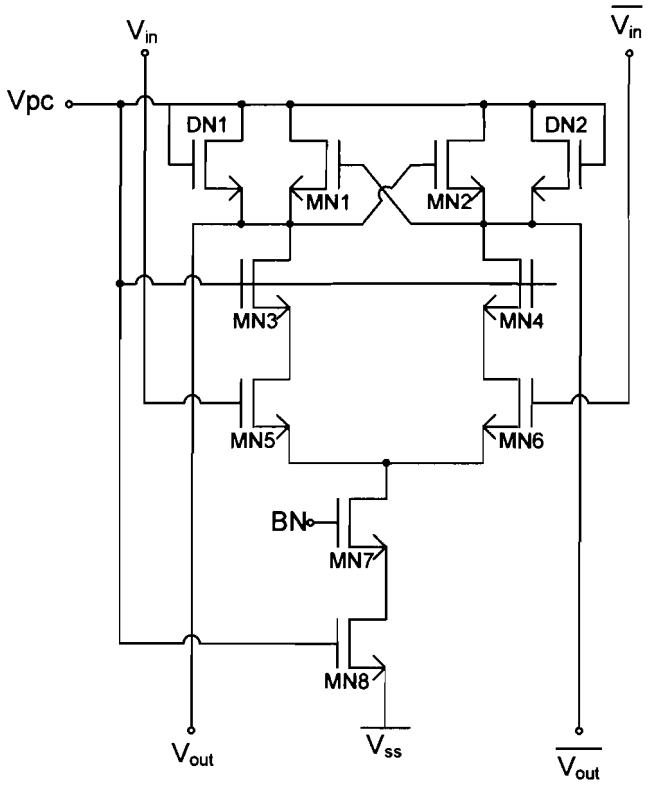
trees to be routed. The ability of an adiabatic logic family to operate using a single-phase power clock is an enormous benefit in easing implementation issues and increasing charge recovery efficiency. Previously proposed single-phased clocking adiabatic families are explained in the following sections.

### 2.3.1 Source-Coupled Adiabatic Logic with Diode-Connected Transistors

Source-coupled adiabatic logic with diode-connected transistors (SCAL-D) [21] is a true single-phase adiabatic logic family. It consists of PMOS type and NMOS types of gates. An example of these gates is shown in Figure 2-15.




(a)




(b)

**Figure 2-15: (a) PMOS buffer & (b) NMOS buffer in SCAL-D [21]**

Pipelined cascades of this family are composed of alternating NMOS and PMOS type gates. The PMOS type gates are in its evaluation phase during the rising edge of the power clock while NMOS type gates are evaluating during the falling edge. This allows for two operations to occur during a single clock cycle. Current sources (**MP7** in PMOS and **MN7** in NMOS type gates) are used to control the energy efficiency and speed of the circuit. Also, diode-connected MOS transistors, (**DN1** & **DN2**) are added to aid in the recovery of charge. A drawback in this adiabatic logic family is the use of diodes which dissipate

energy, as was the case in adiabatic dynamic logic discussed previously. Also, the use of analog circuits to tune the system does not allow this logic family the possibility to be used with automatic layout generation software.

### 2.3.2 Clocked Adiabatic Logic

Figure 2-16 displays a clocked adiabatic logic inverter (CAL) [22]. It is an adiabatic logic family related to the 2N-2N2P family that was modified for operation with a single-phase power clock.

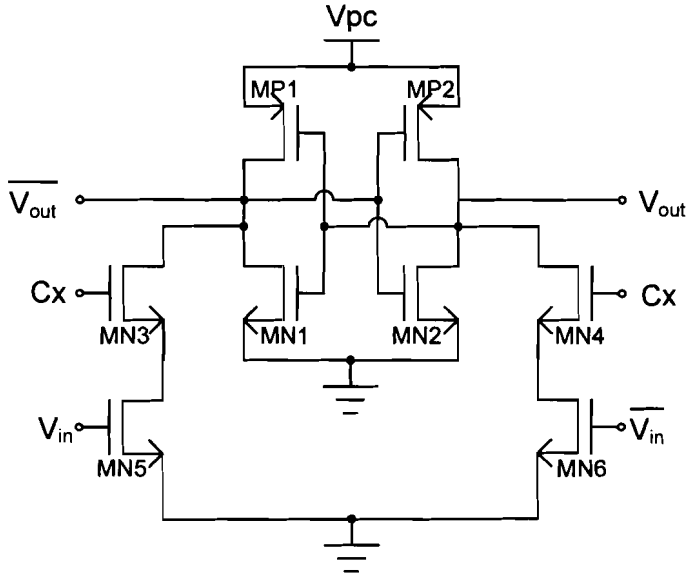


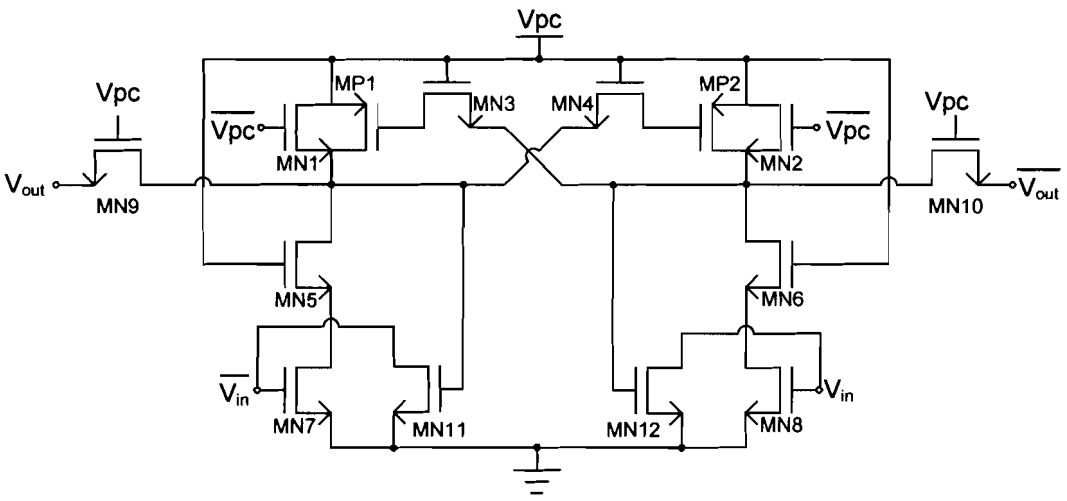

**Figure 2-16: CAL inverter [22]**

The CAL inverter differs from the 2N-2N2P family in the addition of transistors **MN3** & **MN4**, which are controlled by an auxiliary timing control clock, *Cx*. In a cascaded pipeline configuration, a single power clock and two auxiliary timing control clocks are needed. As well, due to logic evaluation occurring in

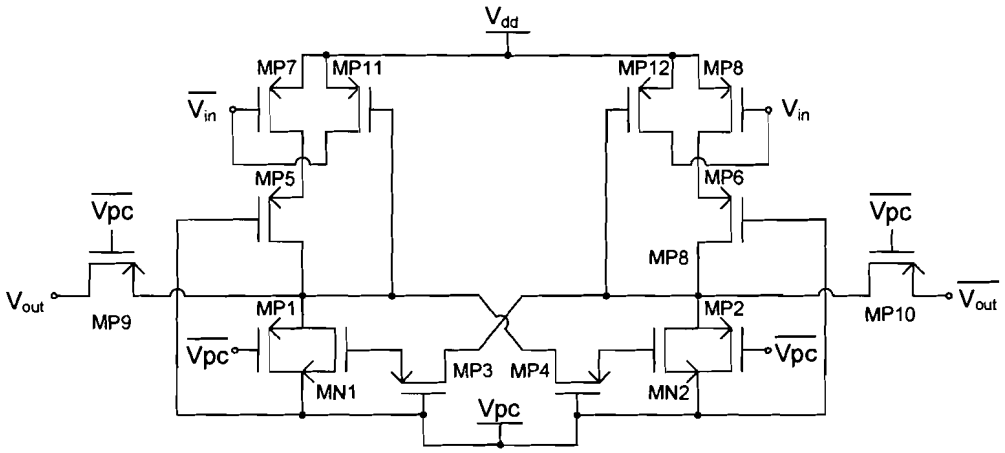
alternate clock cycles, CAL circuits can attain only half the throughput compared to other single-phase clocked adiabatic families.

### 2.3.3 True-Single-Phase-Clocking Adiabatic Differential Cascode Voltage Switch

True-Single-Phase-Clocking Adiabatic Differential Cascode Voltage Switch (TSPC ADCVS) [23] is closely related to the multi-phase clocked, ADSL. TSPC ADCVS employs the use of pass-transistors to keep logic outputs valid for subsequent stages. It was proposed to operate at low-voltages to further reduce energy dissipation. TSPC ADCVS inverters are shown in Figure 2-17.




(a)




(b)

**Figure 2-17: (a) P-type inverter & (b) N-type inverter in TSPC ADCVS [23]**

TSPC ADCVS operates in an N-P domino style similar to SCAL-D. Also, it is capable of computing two operations per clock cycle. However, it can only operate using a trapezoidal power clock, unlike its single-phase clocking counterparts which can also operate using a sinusoidal power clock that simplifies the implementation of the adiabatic logic family. The P-type TSPC ADCVS gates also tend to be less able to drive capacitive loads. Additional input boosting transistors (**MN11**, **MN12** & **MP11**, **MP12**) are used to aid in charging the inputs. These additional transistors increase the transistor count in more complex logic gates and increase energy dissipation.

# CHAPTER 3: DESIGN OF THE LOW-VOLTAGE SINGLE-PHASE CLOCKED QUASI-ADIABATIC PASS-GATE LOGIC

The low-voltage single-phased clocked quasi-adiabatic pass-gate logic (QAPG) family was designed to encompass many key features of previously proposed families. Derived from TSPC ADCVS, QAPG builds upon its existing attributes. Its focus is on operating at low-voltages while being capable of operating at high-clock rates, which is a significant attribute in adiabatic logic [24]. The quasi-adiabatic charge recovery technique was chosen to balance the performance and energy trade-off. QAPG was designed to reap the benefits of a single phase power clock in both sinusoidal and trapezoidal waveforms while achieving an efficient throughput of two operations per clock cycle. It was efficiently designed to handle a range of loads while striving to minimize energy dissipation. In this chapter, the architecture of the QAPG gates will be discussed along with its operation. A comparative analysis will be performed on simulation results of a full-adder and a carry-lookahead adder (CLA) in QAPG and other single-phase clocked adiabatic families.

## 3.1 QAPG Gates

QAPG logic gates are balanced load, differential output gates which are divided into two classes: N & P-type gates. These two classes of gates facilitate the operation of the QAPG logic family using a single-phase power clock.

### 3.1.1 QAPG N-type Logic Gates

The QAPG N-type buffer/inverter is shown in Figure 3-1 to demonstrate the basic structure of the N-type class of logic gates. The architecture of the QAPG N-type gate contains three main components: the bootstrapped cross-coupled transistors, the pass-gate evaluation tree, and the non-adiabatic holding mechanism. QAPG logic gates are based on the cross-coupled transistors (**MP1** & **MP2**), similar to ones found in static DCVS logic circuits. The cross-coupled structure was modified with the inclusion of bootstrap transistors (**MN1** & **MN2**) to increase the switching speeds when operating in the low-voltage regime [20]. The gate-source capacitance ($C_{gs}$) of the bootstrap transistors can cause nodes B1 & B2 to become lower than 0 V due to the capacitive coupling effect at those nodes. This negative potential can help to overdrive the cross-coupled transistors and assist the recovery of charge from nodes Y and $\overline{Y}$. Also, the overdriven transistors can increase the current while charging those nodes.

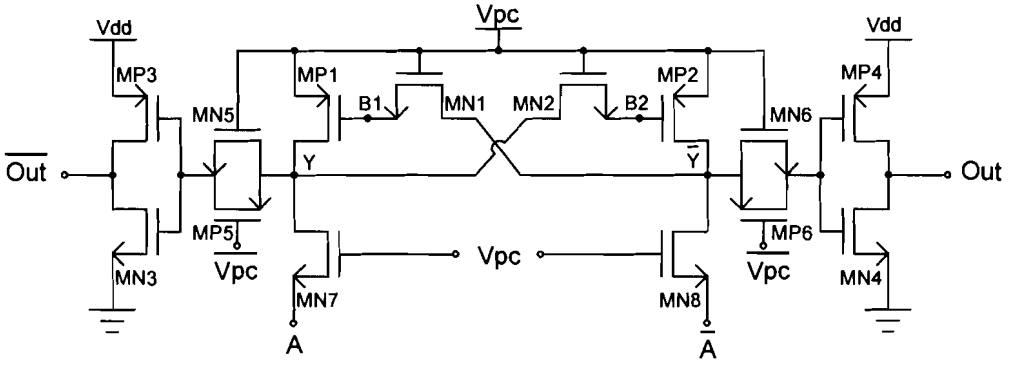



**Figure 3-1: Buffer/Inverter gate implemented in N-type QAPG**

To disconnect the pass-gate evaluation tree during the energy recovery phase of operation from the rest of the circuit, isolation transistors (**MN7** & **MN8**) are used. In order to perform different logic functions, various complementary pass-gate logic trees can be added to the source electrode of **MN7** & **MN8**. These pass-gate trees are beneficial to low-voltage operation as they increase nodal charging speed and noise resilience. Pass-gate evaluation trees also tend to have transistor counts equal to or lower than conventional pull-down networks. An example of an XOR gate implemented in N-type QAPG is shown in Figure 3-2.

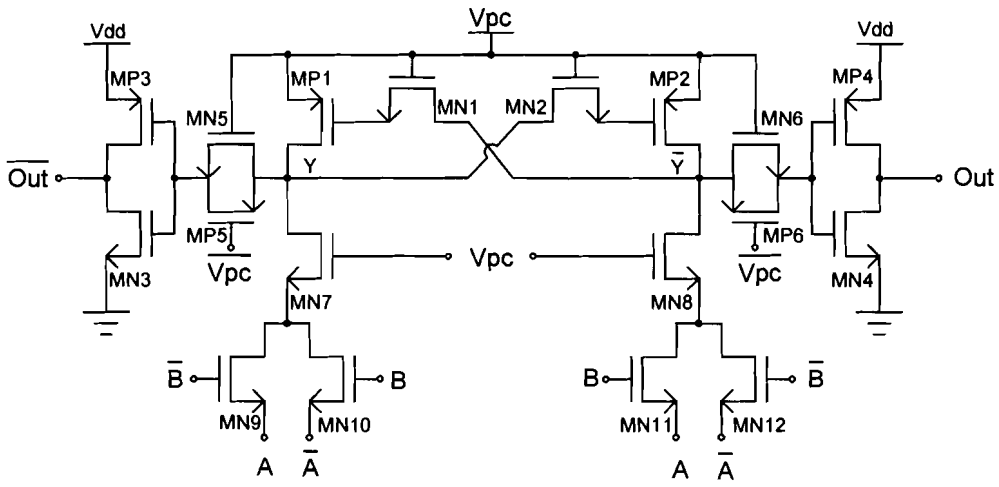



**Figure 3-2: XOR gate implemented in N-type QAPG**

The non-adiabatic holding mechanism consists of transmission gates (**MP5**, **MN5** & **MP6**, **MN6**) and static inverters (**MP3**, **MN3** & **MP4**, **MN4**). This holding mechanism allows for the cascading of QAPG gates by maintaining rail-to-rail output logic levels for subsequent stages, which was not found in TSPC

ADCVS, causing resistive energy losses. The supply voltage to the holding mechanism is the same as the peak voltage of the power clock. In addition, the non-adiabatic holding mechanism can tolerate substantially larger capacitive loading at its outputs than in TSPC ADCVS.

The operation of the N-type gate consists of two phases: ***evaluate*** and ***discharge***. As the power clock ($V_{pc}$) ramps up to logic high, each N-type logic gate enters its evaluation phase. During this phase, the pass-gate evaluation tree determines logic value of the internal nodes (Y & $\overline{Y}$). Whichever of these nodes is evaluated to a logic high is charged from the power clock. This logic value is then passed on to the non-adiabatic holding mechanism which makes available the logic gate output for subsequent stages in the pipeline. During the discharge phase, the power clock ramps down to logic low. The pass-gate evaluation tree is cut off from internal nodes (Y & $\overline{Y}$) by isolation transistors **MN7** & **MN8** while the transmission gates of the non-adiabatic holding mechanism disconnects the inverters from the internal nodes. The charges stored at the remaining nodes are then recovered through the bootstrapped cross-coupled pair and back to the power clock, fulfilling the adiabatic function.

### 3.1.2 QAPG P-type Logic Gates

The structure and operation of the QAPG P-type logic gate is analogous to the N-type logic gate. The QAPG P-type buffer/inverter is shown in Figure 3-3. The P-type logic gate maintains the same non-adiabatic holding mechanism as in the N-type logic gates; however the NMOS and PMOS transistors in the

bootstrapped cross-coupled transistors and the pass-gate evaluation tree have been swapped. An example of an XOR gate implemented in P-type QAPG is shown in Figure 3-4.

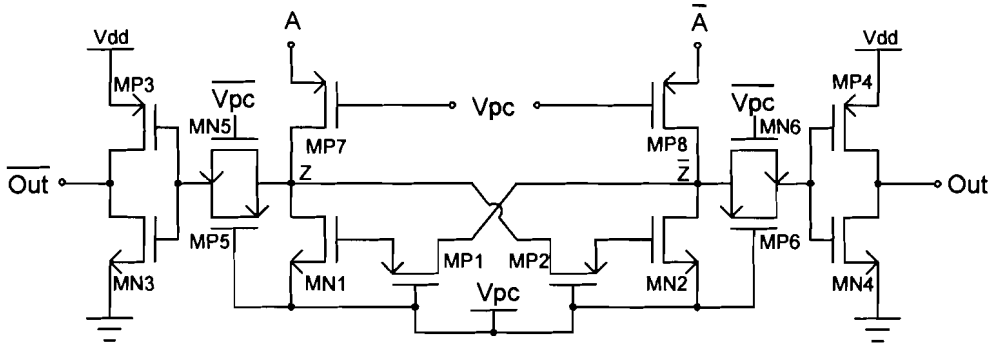



**Figure 3-3: Buffer/Inverter gate implemented in P-type QAPG**

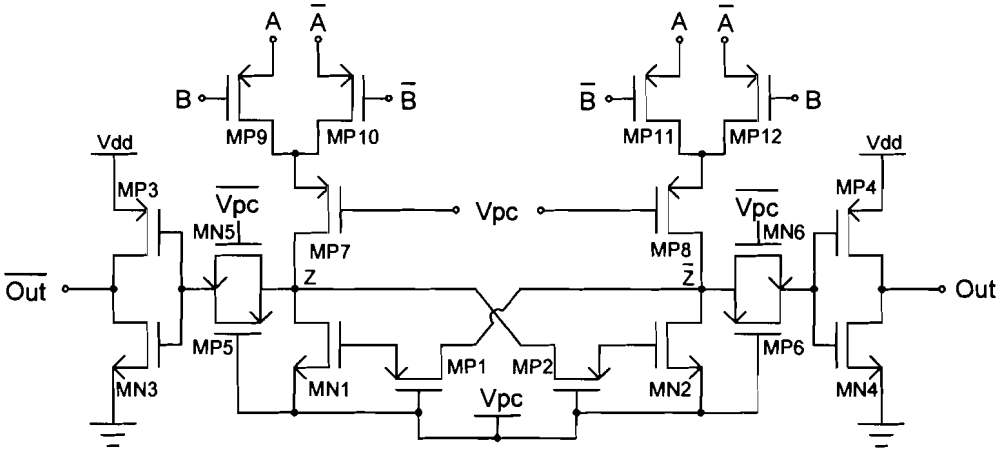



**Figure 3-4: XOR gate implemented in P-type QAPG**

The QAPG P-type logic gate operates in two phases: ***charge*** and ***evaluate***. During the charging phase, the power clock ($V_{pc}$) ramps up to logic

high. This charges one of the two complementary nodes that was previously at logic low. For example, let node Z be at logic low, and $\overline{Z}$ be at logic high, while $V_{pc}$ is at logic low. When $V_{pc}$ returns to logic high, **MP1** will turn off with the gate of **MN1** still at logic high due to parasitic capacitances at that node. In turn, $V_{pc}$ will charge node Z to logic high while node $\overline{Z}$ remains at logic high. Meanwhile, the pass-gate evaluation tree is disconnected from the circuit and the non-adiabatic holding mechanism maintains the previously evaluated logic result.

During the evaluation phase, the power clock ramps down to logic low. At this time, the pass-gate evaluation tree computes the logic values of the internal nodes Z and $\overline{Z}$. The charges from one of the nodes, which is evaluated to logic low, will be recovered to the power clock, fulfilling the adiabatic function. This operation principle differs from the N-type gate in that during the P-type evaluate phase, nodal charges are recovered by $V_{pc}$ whereas charge is recovered during the discharge phase of N-type gate operation.

## 3.2 QAPG Cascaded Cell Operation

QAPG logic circuits are constructed by cascading N-type and P-type logic gates in an N-P domino (alternating) style. This creates a pipelining effect which utilizes both edges of a single power clock. Enabling signals to propagate through an N-type and P-type cell in one clock cycle effectively performs two operations per clock cycle. Also, the use of a single power clock eliminates such disadvantages associated with multi-phase clocking adiabatic circuits as complex capacitive tuning and clock skewing. Lacking these drawbacks, the single-phase

clocking scheme can facilitate higher clock frequencies and more efficient energy recovery. As previous discussed, N-type logic gates operate in two phases, ***evaluate*** and ***discharge***, while P-type logic gates operate in ***charge*** and ***evaluate*** phases. Figure 3-5 shows these phases with respect to the power clock waveform.

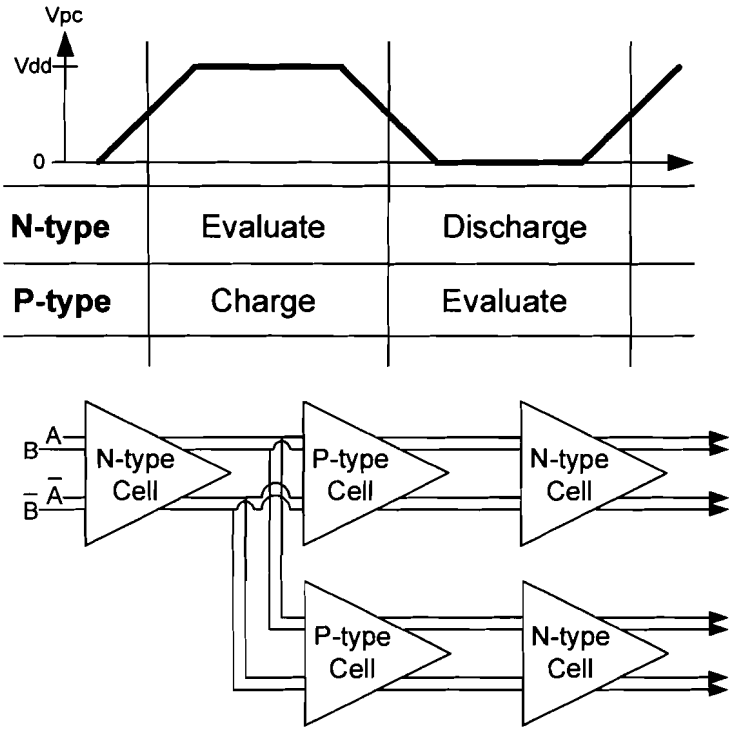



**Figure 3-5: QAPG timing phases**

At any one time during the operation of a QAPG circuit, either all N-type logic gates are in the evaluate phase and all P-type logic gates are in the charge phase, or all N-type gates are in the discharge phase and all P-type gates are in the evaluate phase. This second phase (N-type: ***discharge***, P-type: ***evaluate***) is also the phase in which charge is recovered back to the power clock.

A cascaded QAPG P-type buffer/inverter – N-type buffer/inverter is shown in Figure 3-6 to demonstrate the QAPG pipeline architecture. A transient analysis of the nodal voltages of the cascade operating at 100 MHz with a supply voltage of 0.5 V is shown in Figure 3-7. Nodes 1 & 2 from the P-type buffer/inverter displays the charge and evaluate phases which can be observed when both nodes are at logic high and when they are differential. Nodes 3 & 4 display the differential output from the non-adiabatic holding mechanism. Correspondingly, nodes 5 & 6 demonstrate the discharge and evaluate phases while, Out and $\overline{Out}$ illustrate the outputs corresponding to inputs A and $\overline{A}$ passing through a pipeline consisting of n=2 buffer/inverters (delayed by an $\frac{n-1}{2f_{clk}}$ interval).

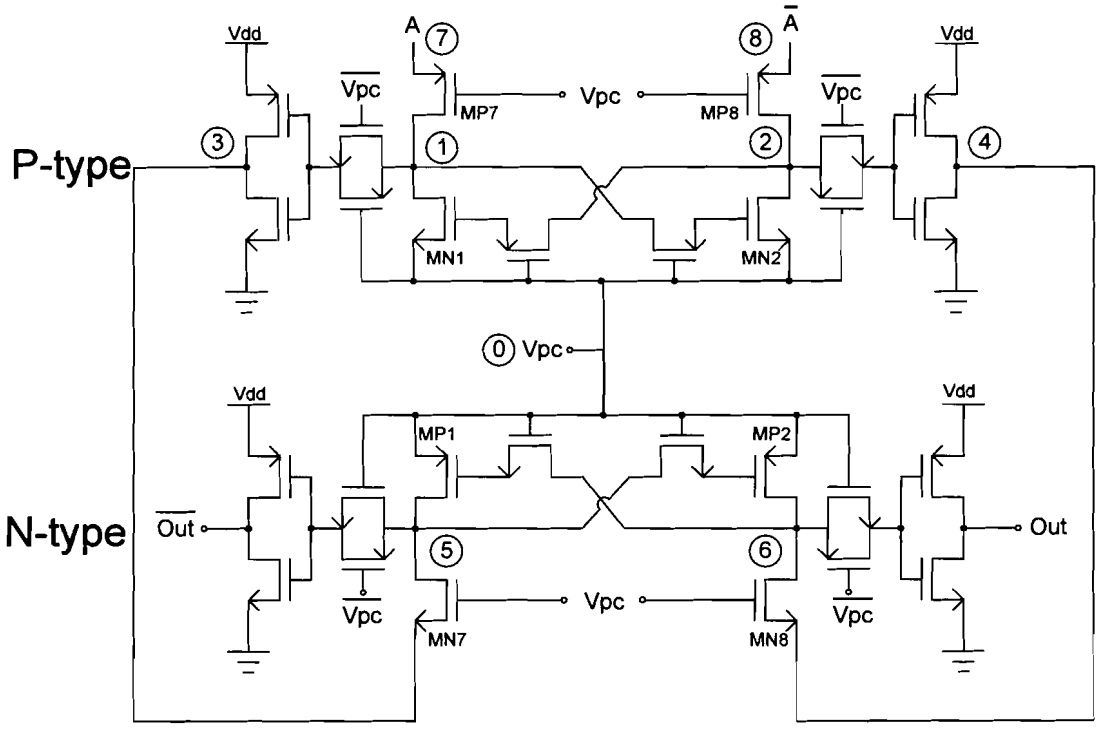



**Figure 3-6: QAPG P-N buffer/inverter cascade**

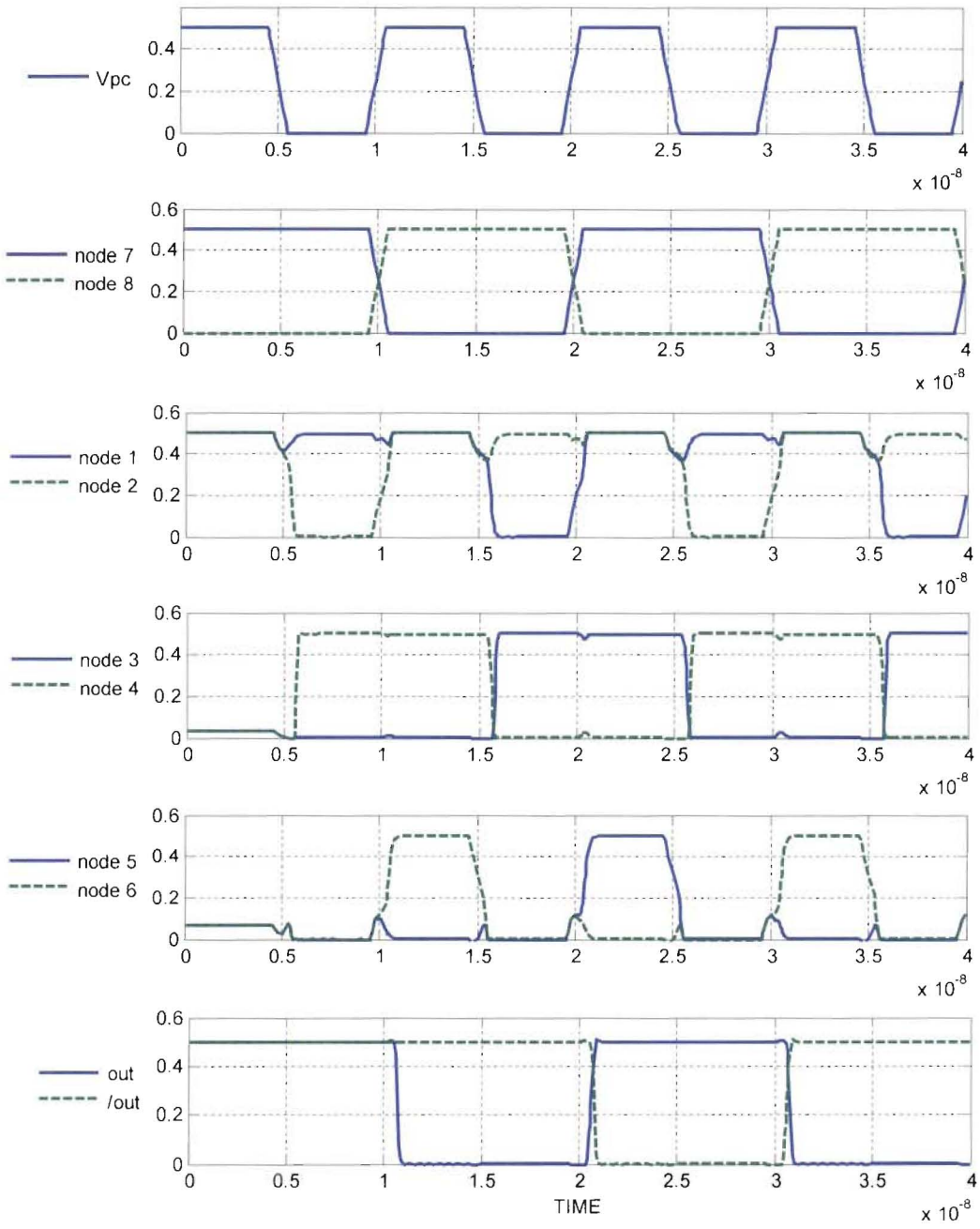



**Figure 3-7: Transient analysis of nodal voltages in a QAPG P-N buffer/inverter cascade**

## 3.3 QAPG Full-Adder Implementation

A QAPG full-adder logic circuit, shown in Figure 3-8, was designed and simulated in HSPICE using 90 nm standard CMOS process parameters in order to evaluate the energy efficiency of the QAPG logic family. Using the same schematic, full-adders were constructed in CAL [22] and TSPC ADCVS [23] for comparison between QAPG and previously proposed single-phase clocked adiabatic logic families.

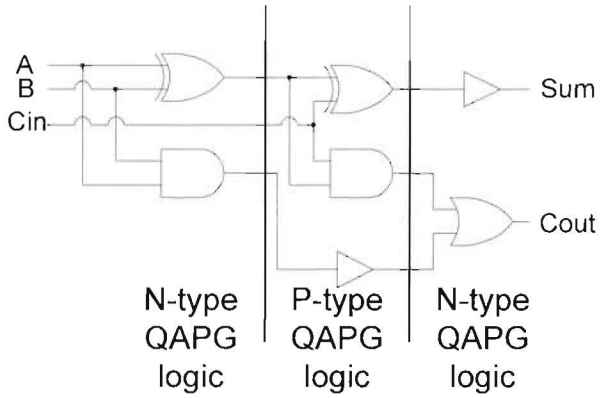



**Figure 3-8: Schematic diagram of a full-adder**

Since the objective was to minimize area and energy dissipation, minimum sizes were used for NMOS transistors in each logic family. In this case, transistor channels were made 120 nm wide, while the channel lengths were set at 100 nm, according to the design rules specifications. To compensate for the lesser driving capability of the PMOS transistors, the PMOS channel widths were doubled to 240 nm, while maintaining the same channel lengths as in the NMOS transistors. The QAPG and TSPC ADCVS full-adders have a latency of 1.5

cycles since data propagation through each pipeline stage takes half a cycle, while the CAL adder has a latency of 3 cycles. Buffers were used to propagate the correct logic values through the pipeline architecture.

### 3.3.1 Simulation Results

Using the 90 nm CMOS process, energy dissipation measurements were determined from the simulation outputs, by integrating the power over the period of simulation. These calculations were made using transient analysis data from HSPICE and with MATLAB computations. Standard threshold voltage models were used for each simulation. Simulations assumed a lossless external adiabatic power clock. Sinusoidal and trapezoidal power clocks were used for all families except for TSPC ADCVS, which was not designed for sinusoidal power clock use. The inputs to the full-adder circuit were a three bit counter, which cycled through each possible logic pattern. Energy dissipation, being the major figure of merit in adiabatic circuits, was examined throughout each simulation.

#### 3.3.1.1 Energetics

The energy flow the QAPG full-adder operating at a frequency of 100 MHz with a 0.5 V trapezoidal power clock is shown in Figure 3-9. The power clock energy is observed to be sourcing and recovering energy. As well, static energy contributes 14.5% of the total energy dissipated.

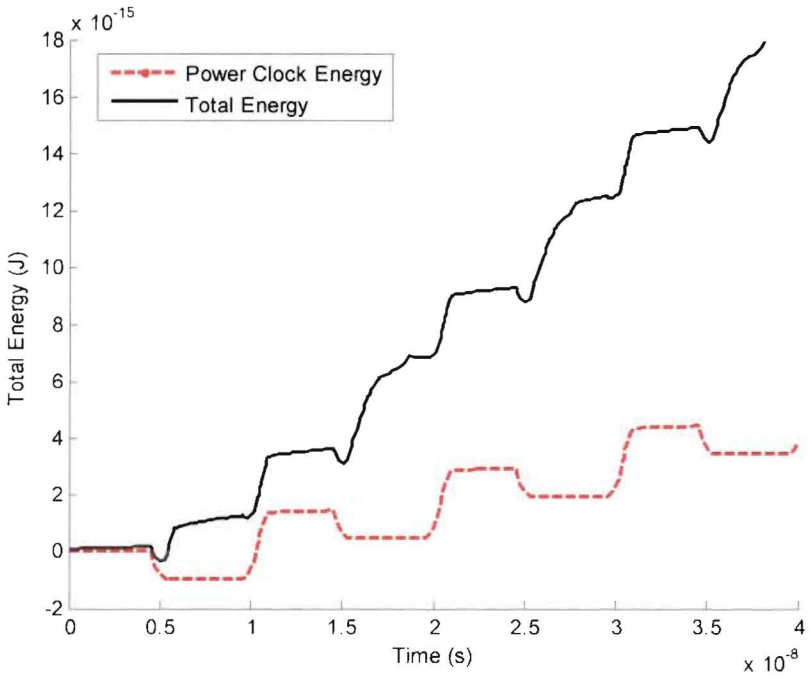



**Figure 3-9: Energetics of a QAPG full-adder operating at 100 MHz**

Figure 3-10 below displays a comparison of total energy between a conventional pipelined static CMOS full-adder and the QAPG full-adder. The pipelined CMOS full-adder was implemented with identical transistor sizes and pipelining stages as in the QAPG adder. We observe that the QAPG full-adder dissipates approximately 30% of the total energy dissipated in an equivalent static CMOS adder.

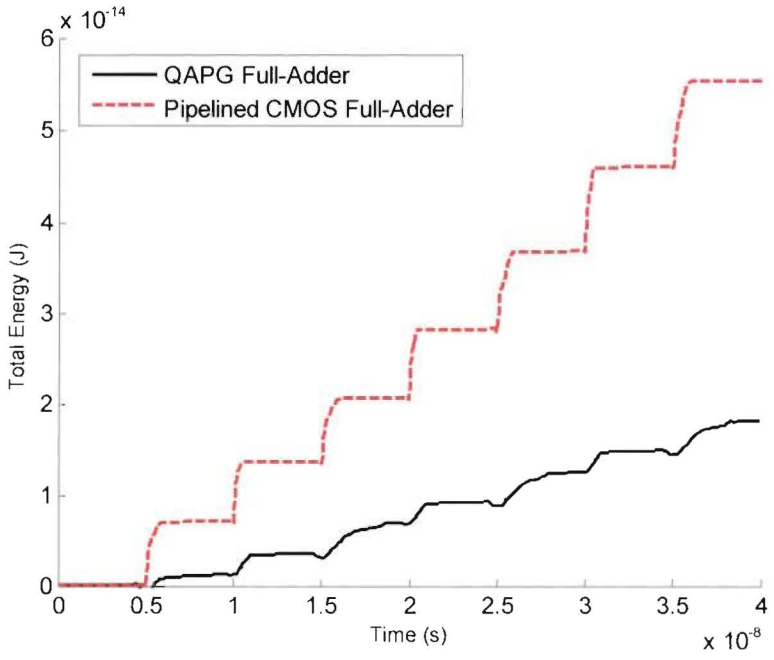



**Figure 3-10: Energy comparison between a pipelined CMOS and a QAPG full-adder**

### 3.3.1.2 Influence of Power Clock Voltage Scaling

Voltage scaling is an important feature in logic families as it allows more design flexibility for circuit designers. Comparisons between CAL, TSPC ADCVS, and QAPG logic families were performed while varying the peak-to-peak voltage of the power clock. The minimum operational supply voltage was observed in addition to the energy dissipated. The operating frequency was 10 MHz.

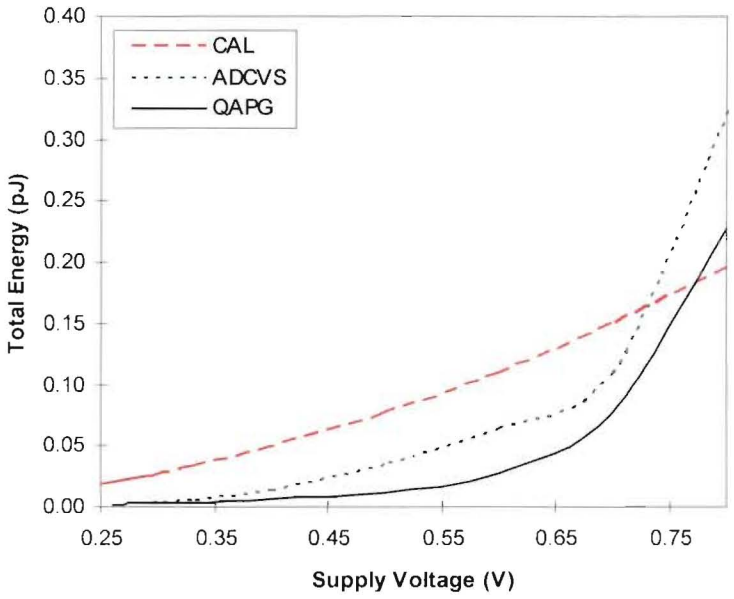



**Figure 3-11: Energy dissipation versus supply voltage for a full-adder (trapezoidal waveform)**

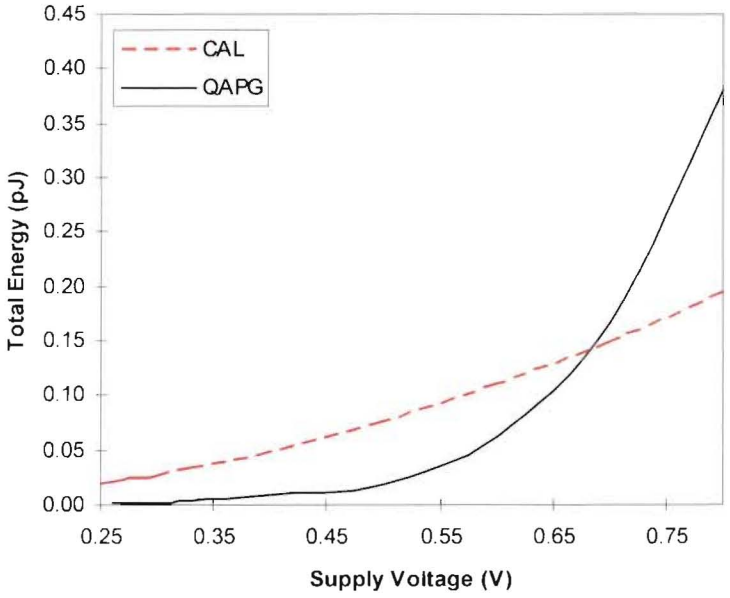



**Figure 3-12: Energy dissipation versus supply voltage for a full-adder (sinusoidal waveform)**

Figures 3-11 & 3-12 present the per addition operation (1.5 cycles) energy dissipation of the full-adders in the three single-phase clocked adiabatic logic families employing a trapezoidal and sinusoidal power clock waveform respectively. When utilizing the trapezoidal waveform, QAPG dissipates down to 11% of the energy expended in CAL and down to 35% of the energy expended in TSPC ADCVS. The most dramatic energy savings are observed at the lowest reliable supply voltage for all families which is approximately 0.3 V. Operating with the sinusoidal waveform, QAPG dissipates down to 10% of the energy expended in CAL. QAPG was observed to have excellent voltage scaling properties in this low-voltage region, with a minimum operating voltage of 0.26 V for this full-adder.

#### 3.3.1.3 Influence of Power Clock Frequency Scaling

The ability to operate at various frequencies while maintaining low-energy dissipation is a desirable attribute in adiabatic circuits. Thus, comparisons between CAL, TSPC ADCVS, and QAPG logic families were performed while varying the operating frequency of the adiabatic power clock. The maximum operational speed was observed in addition to the energy dissipated. The supply voltage used in this simulation was 0.5 V.

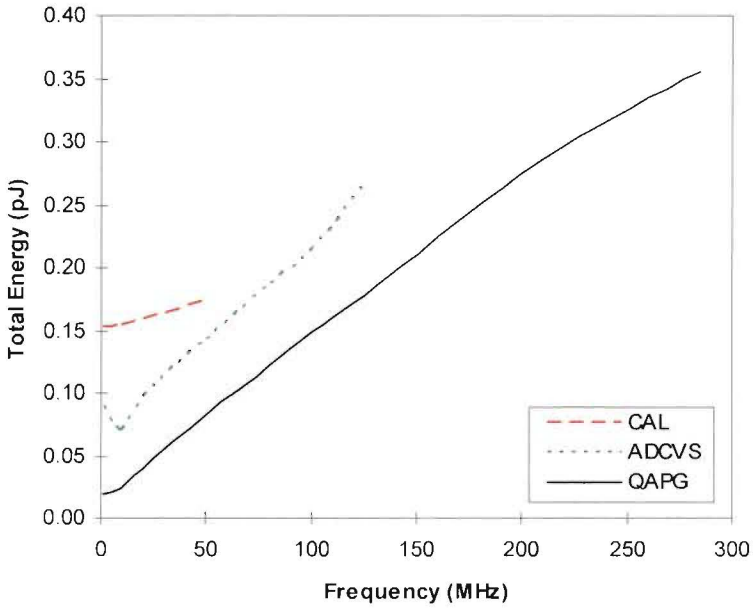



**Figure 3-13: Energy dissipation versus frequency for a full-adder (trapezoidal waveform)**

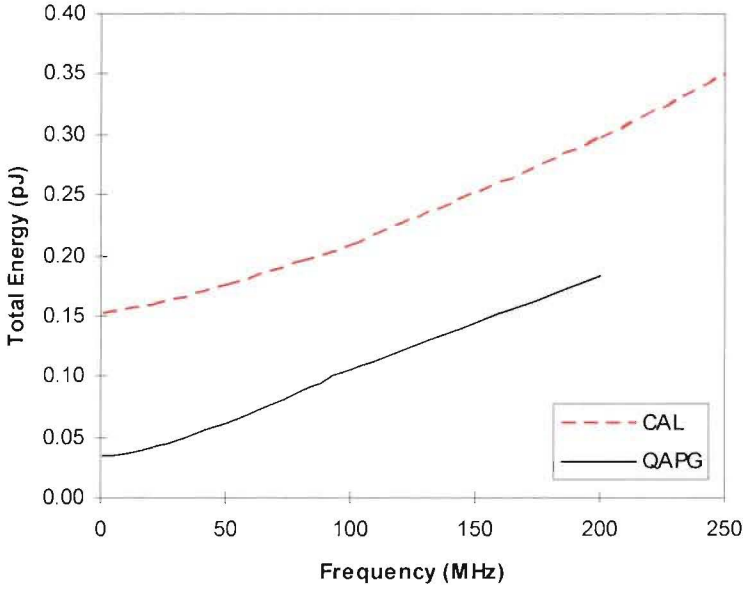



**Figure 3-14: Energy dissipation versus frequency for a full-adder (sinusoidal waveform)**

Figures 3-13 & 3-14 present the total energy dissipation (for 300 ns) as a function of operating frequency utilizing a trapezoidal and sinusoidal power clock waveform. In the case when using a trapezoidal waveform, QAPG dissipates approximately 15% of the energy expended in CAL and approximately 34% of TSPC ADCVS when operating at 10 MHz. Operating with the sinusoidal waveform at 10 MHz, QAPG dissipates down to 25% of the energy expended in CAL. QAPG is able to operate up to 285 MHz with a trapezoidal waveform and up to 200 MHz with a sinusoidal waveform at 0.5 V in this configuration. The benefits of the pass-gate evaluation trees enable the maximum operating frequency of QAPG to be more than double that of TSPC ADCVS. CAL is able to operate at much higher frequencies when utilizing a sinusoidal waveform as it was designed to be operated with a sinusoidal power-clock waveform

#### 3.3.1.4 Influence of Capacitive Loading

In addition, the ability to operate with a range of capacitive loads while maintaining low-energy dissipation is also desirable in adiabatic circuits. Thus, comparisons between CAL, TSPC ADCVS, and QAPG logic families were performed while steadily increasing the capacitive load at the outputs. The adder is operating at a 0.5 V supply voltage and at a frequency of 10 MHz.

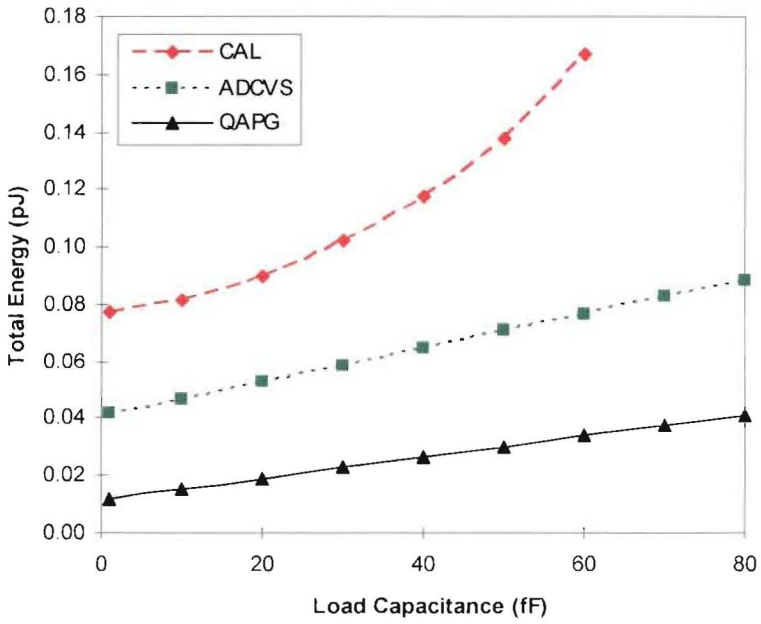



**Figure 3-15: Energy dissipation versus load capacitance for a full-adder (trapezoidal waveform)**

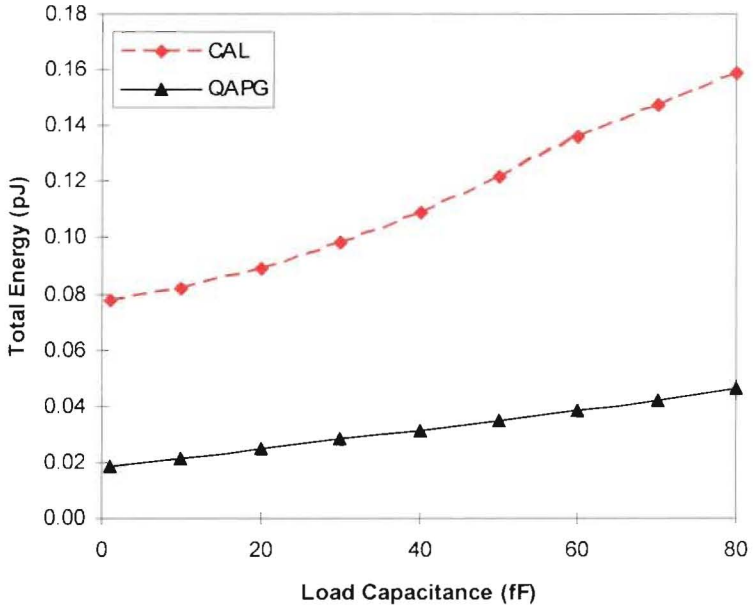



**Figure 3-16: Energy dissipation versus load capacitance for a full-adder (sinusoidal waveform)**

Figures 3-15 & 3-16 present the total energy dissipation for every addition operation (1.5 cycles) as a function of load capacitance at their outputs utilizing a trapezoidal and sinusoidal power clock waveform. When using the trapezoidal power clock waveform, QAPG dissipates 20% of the total energy expended by CAL and 40% of the TSPC ADCVS. Using the sinusoidal power clock, QAPG dissipates 30% of the total energy expended by CAL. Owing to the non-adiabatic holding mechanism, the QAPG logic gates are able to dissipate lower energy while being able to cope with larger loads.

#### 3.3.1.5 Operation in Future CMOS Technological Nodes

Since the QAPG logic family is a newly proposed adiabatic logic family, the ability of the family to function in the future within smaller technological nodes is a necessity if it were to be manufactured in industry. Simulations of the QAPG full-adder were performed using the 90 nm, 65 nm, 45 nm, and 32 nm BSIM4 model cards for the bulk CMOS processes in the Predictive Technology Model (PTM) [25]. Simulations were repeated, observing the influence of voltage and frequency scaling, and capacitive loading.

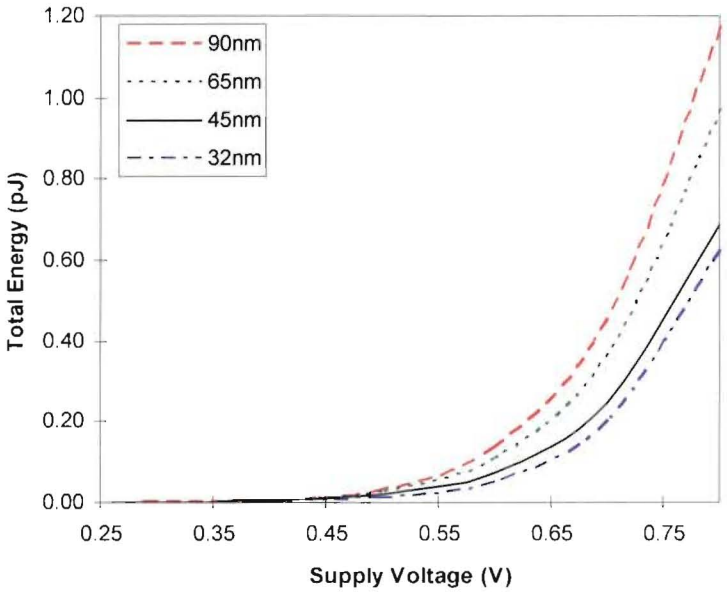



**Figure 3-17: Expected energy dissipation versus supply voltage for a full-adder for 90 nm, 65 nm, 45 nm, & 32 nm technology nodes**

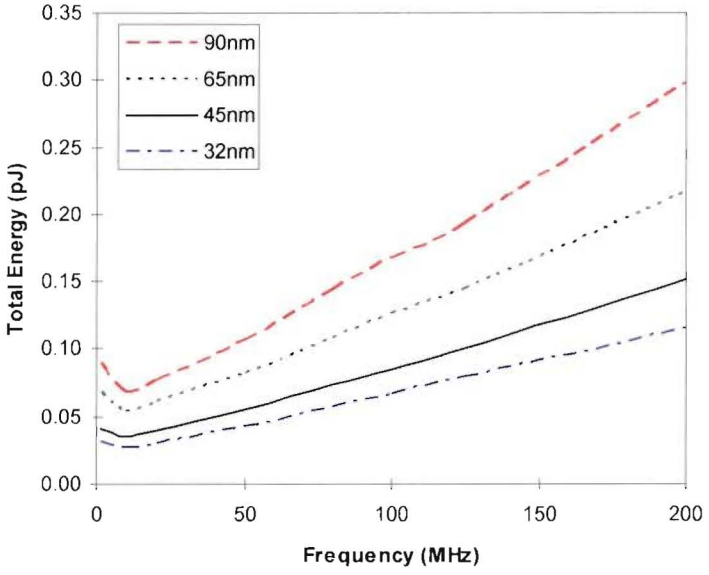



**Figure 3-18: Expected energy dissipation versus frequency for a full-adder for 90 nm, 65 nm, 45 nm, & 32 nm technology nodes**

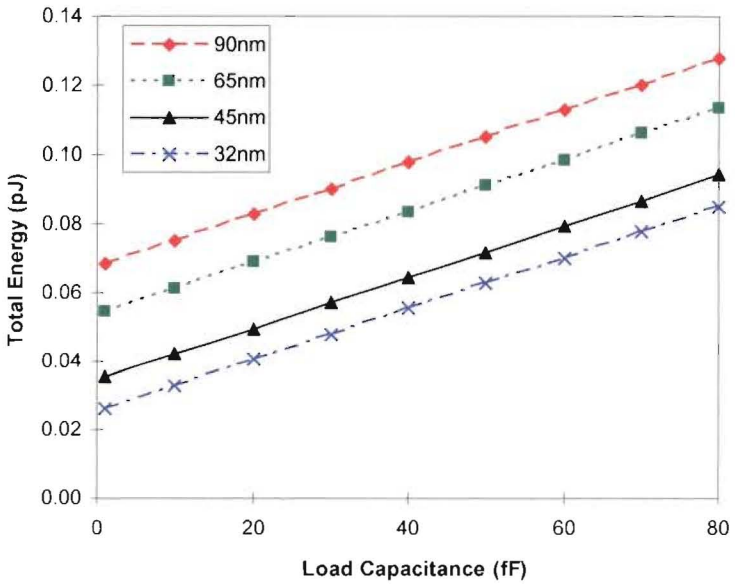



**Figure 3-19: Expected energy dissipation versus load capacitance for a full-adder for 90 nm, 65 nm, 45 nm, & 32 nm technology nodes**

Figures 3-17, 3-18, and 3-19 demonstrate the technological scaling abilities of the QAPG logic family. The QAPG full-adder was fully operational at these technology nodes and show improved energy dissipation with decreasing channel lengths, which was anticipated. With these results, we can expect the QAPG logic family to continue working as the bulk CMOS process evolves into future technology nodes as planned by ITRS.

#### 3.3.1.6 Process Corner Analysis

In order to investigate the sensitivity to process variability of the QAPG family, simulations were performed on the QAPG full-adder over the process corners of the 90 nm CMOS process. Correct circuit operation was verified and the energy dissipation of the circuit was observed and presented in Table 3-1.

The full-adder was simulated with a 0.5 V supply voltage using a trapezoidal power clock waveform and was operating at 100 MHz.

**Table 3-1: Energy dissipation of the QAPG full-adder over process variations**

| Process Corner | Energy Dissipation (J) |
|---|---|
| TT | 1.5406E-14 |
| FF | 1.5915E-14 |
| SS | 1.4858E-14 |
| SF | 1.3019E-14 |
| FS | 2.3713E-14 |

The energy dissipation was fairly consistent across TT, FF, and SS corners with FF having increased dissipation due to lower threshold voltages and SS having lower dissipation due to higher threshold voltages. The FS corner had the largest energy dissipation as it increased the difference in driving strength between the more capable NMOS and less capable PMOS transistors. This uneven strength is likely to cause longer active times during dynamic switching events. Also, as the NMOS transistors ($V_{th}$ = 0.24 V) already have a lower threshold voltage compared to PMOS transistors ($V_{th}$ = 0.29 V), the decrease in the NMOS threshold voltage when operating at a low voltage of 0.5V would significantly increase leakage.

## 3.4 QAPG 8-bit Carry-Lookahead Adder Implementation

In order to verify the block-level performance figure of merit, an 8-bit carry-lookahead adder was implemented in QAPG to observe the energy dissipation and operation in a larger circuit structure. A radix-2 Kogge-Stone adder [26], which is a type of parallel prefix carry-lookahead adder, was chosen as it is a fast adder with a fairly regular structure and a reasonably consistent fan-out [5]. The gate-level schematic of the adder design is shown in Figure 3-20. Some logic gates which have no output connection are included as they are part of cells created in the layout found in the next chapter.

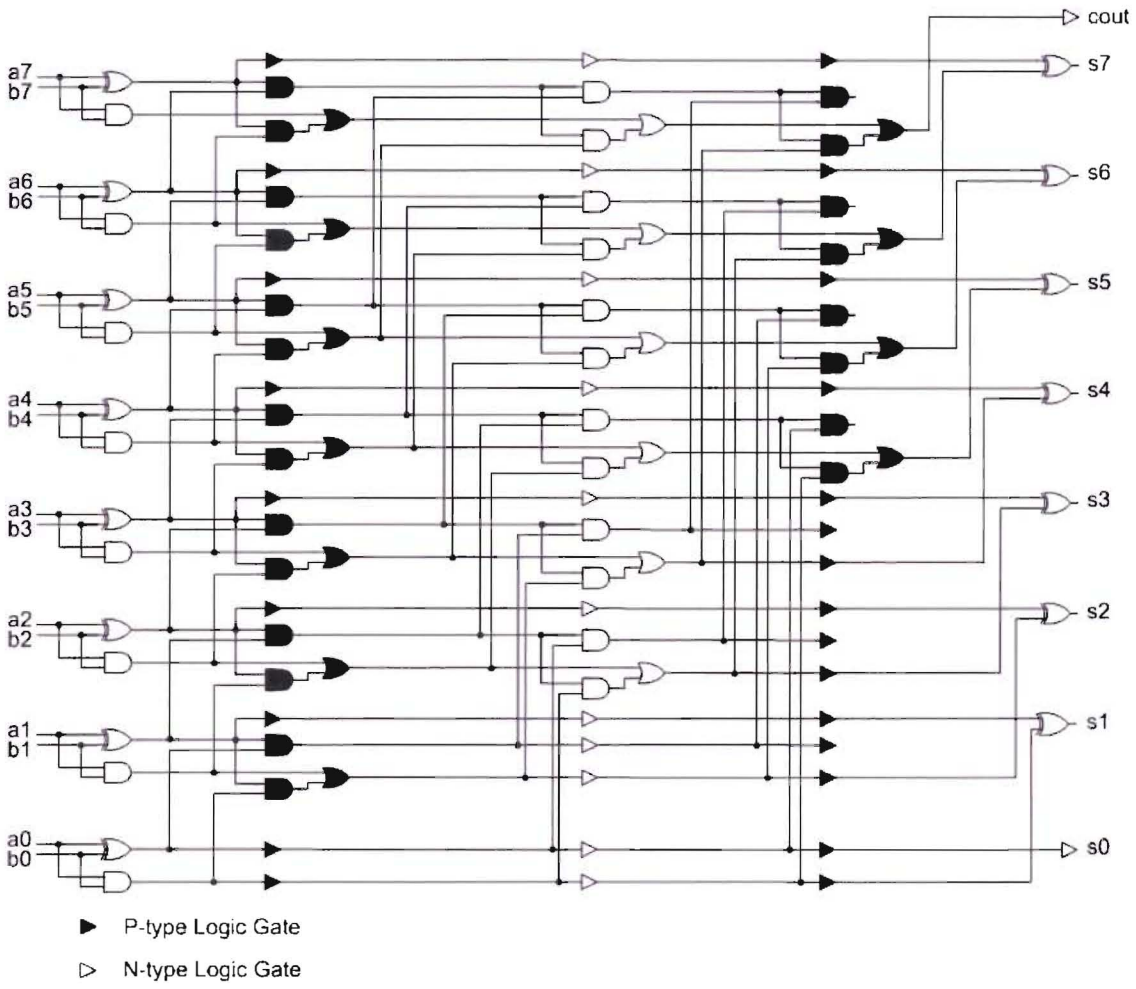



**Figure 3-20: 8-bit Kogge-Stone lookahead adder**

### 3.4.1 Simulation Results

Using simulations and analysis through HSPICE and MATLAB, energy dissipation observations were made in CAL, TSPC ADCVS and QAPG logic families. Trapezoidal and sinusoidal power clock waveforms were again used and are shown in Figures 3-21 & 3-22 respectively. The energy dissipation for an addition operation (2.5 clock cycles) was observed when operating at 1 MHz,

10 MHz, 50 MHz, and 100 MHz. The energy consumption of the adders was obtained using the smallest supply voltage capable for that family at that operating frequency. The supply voltages are shown next to each data point on the figures.

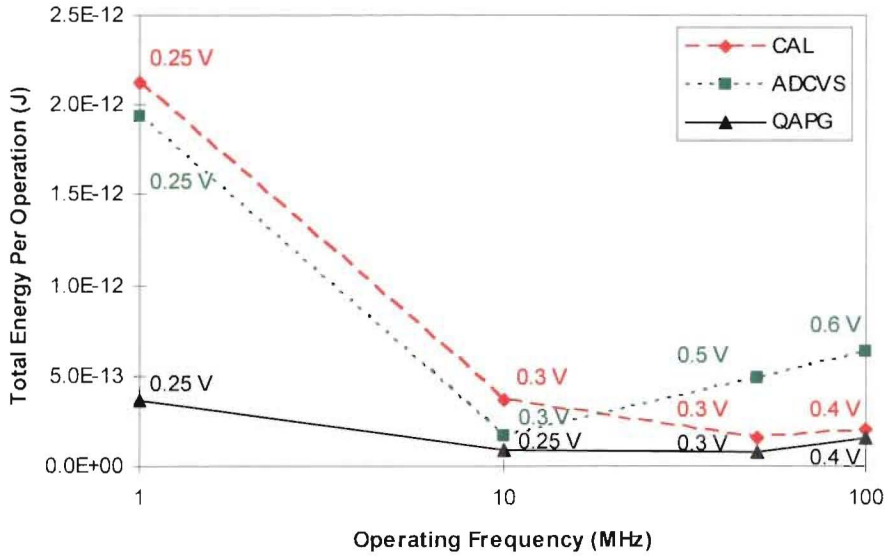



**Figure 3-21: Energy dissipation versus operating frequency for 8-bit CLAs (trapezoidal waveform)**

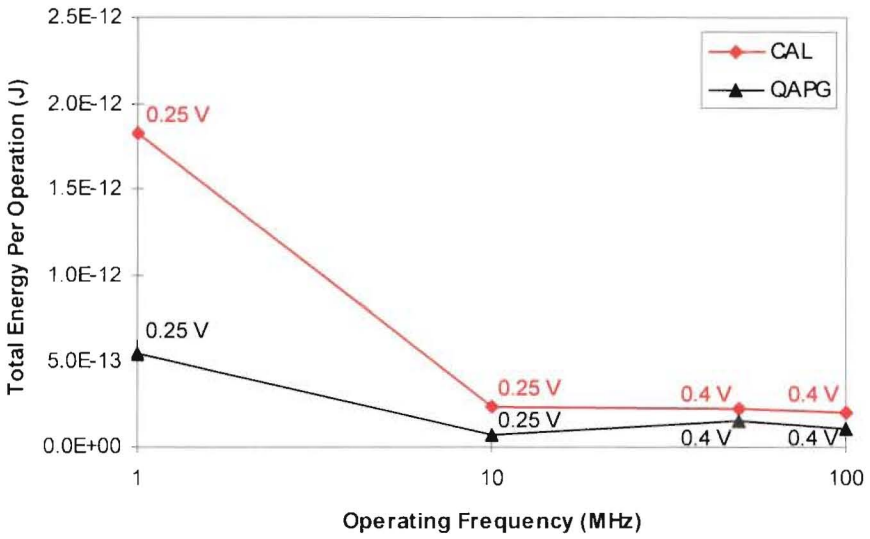



**Figure 3-22: Energy dissipation versus operating frequency for 8-bit CLAs (sinusoidal waveform)**

From these simulations, we notice that at a lower frequency, the per-operation energy dissipation is much higher at lower frequencies which are due to the increased leakage observed in submicron technologies [27]. Sinusoidal power clock waveforms tend to have slightly higher energy dissipation compared to the trapezoidal waveform at lower frequencies and lower energy dissipation at higher frequencies. This can attributed to the active time of the transmission gates in the non-adiabatic holding mechanism, which transition slower during lower frequencies. Higher supply voltages were needed at higher operating frequencies due to the more complex gates used in the design of this adder, mainly the Dot operator ($[G,P] \bullet [G',P'] = [G+PG', PP']$). QAPG dissipated less

energy than the other two families in this series of simulations. Also, QAPG was able to operate with the lowest supply voltage at each frequency tested.

# CHAPTER 4: DESIGN VERIFICATION OF THE LOW-VOLTAGE SINGLE-PHASE CLOCKED QUASI-ADIABATIC PASS-GATE LOGIC

To verify the design of the QAPG 8-bit Kogge-Stone adder, a layout of the adder circuit was created. Design verification through simulations of circuits with extracted parasitics is a cost-effective way to ensure a functioning chip after fabrication. This chapter describes the methodology in creating the circuit layout and presents the simulation results of the extracted circuit.

## 4.1 8-bit Kogge-Stone Carry-Lookahead Adder Design

The 8-bit Kogge-Stone carry-lookahead adder was designed in a cell-based, hierarchical style consisting of lower level logic gates, and mid level logic operator cells. The basic cells used in designing the adder were N-type buffer/inverter, AND, AND-OR, and XOR gates as well as P-type buffer/inverter, AND, AND-OR gates. Two types of logic operators, the propagate/generate generator and the DOT operator, were implemented using these basic cells as shown in Figure 4-1 and Figure 4-2.

The propagate/generate generator utilizes the N-type AND and XOR gates since the generator was only needed for one stage of the pipeline, while the DOT operator consists of both N-type and P-type AND and AND-OR gates. The cell-based CLA circuit is shown in Figure 4-3.

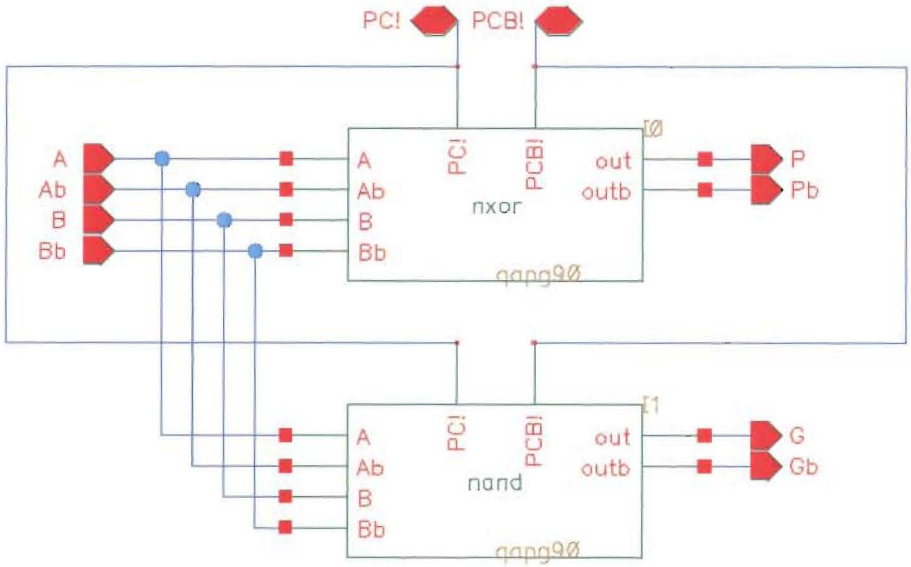



**Figure 4-1: Propagate/generate generator**

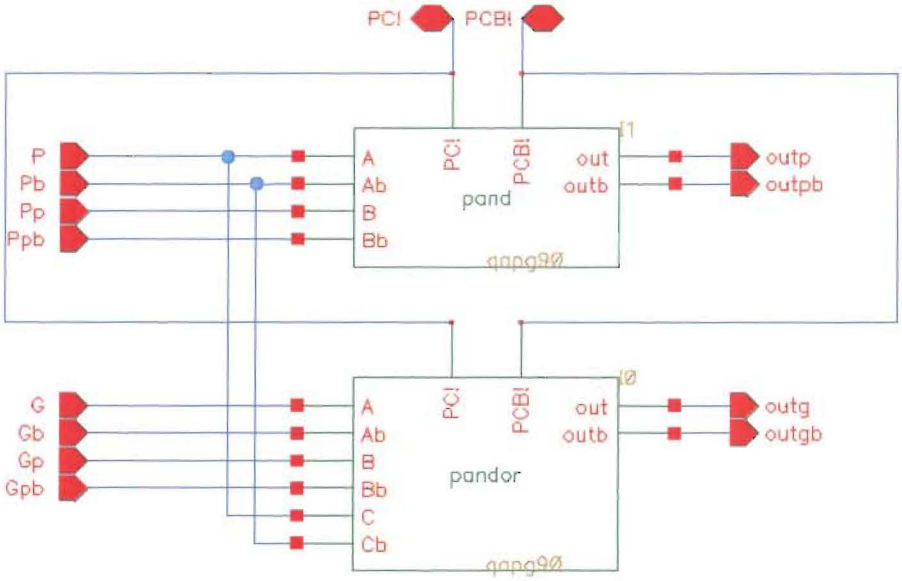



**Figure 4-2: DOT operator**

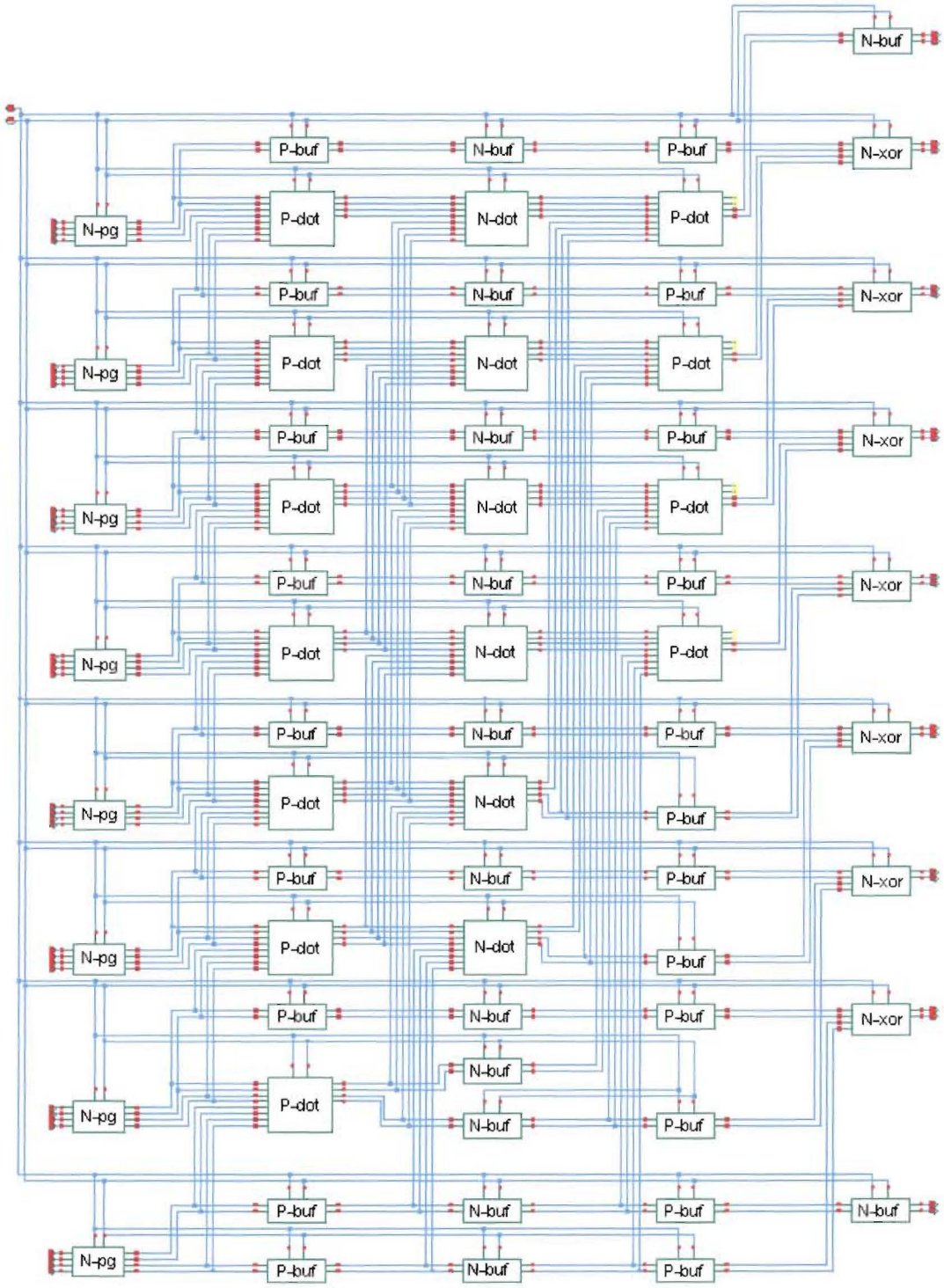



**Figure 4-3: 8-bit Kogge-Stone CLA cell-based design**

## 4.2 Cell Layout Design

Using the SKILL scripting language and Cadence Virtuoso, the layout of the cells used for the final adder layout was created. Since the complementary pass-gate logic evaluation tree would vary in size, the height of each cell was left unbounded while the width of the cells was kept at 6.84 μm. PMOS transistors were placed above the NMOS transistors in both the N-type and P-type gates. An example of an N-type propagate/generate generator layout is shown in Figure 4-4, while a P-type DOT operator layout is shown in Figure 4-5. The power rails are routed above the N-type cells and below the P-type cells.

Transistor spacing was kept to a minimum to consume the least area and reduce parasitics due to excessive routing. However, additional parasitics were obtained while conforming to area requirements in metal layers due to design rules. Internal routing for the cells was performed in Metal 1 and Metal 2 layers.

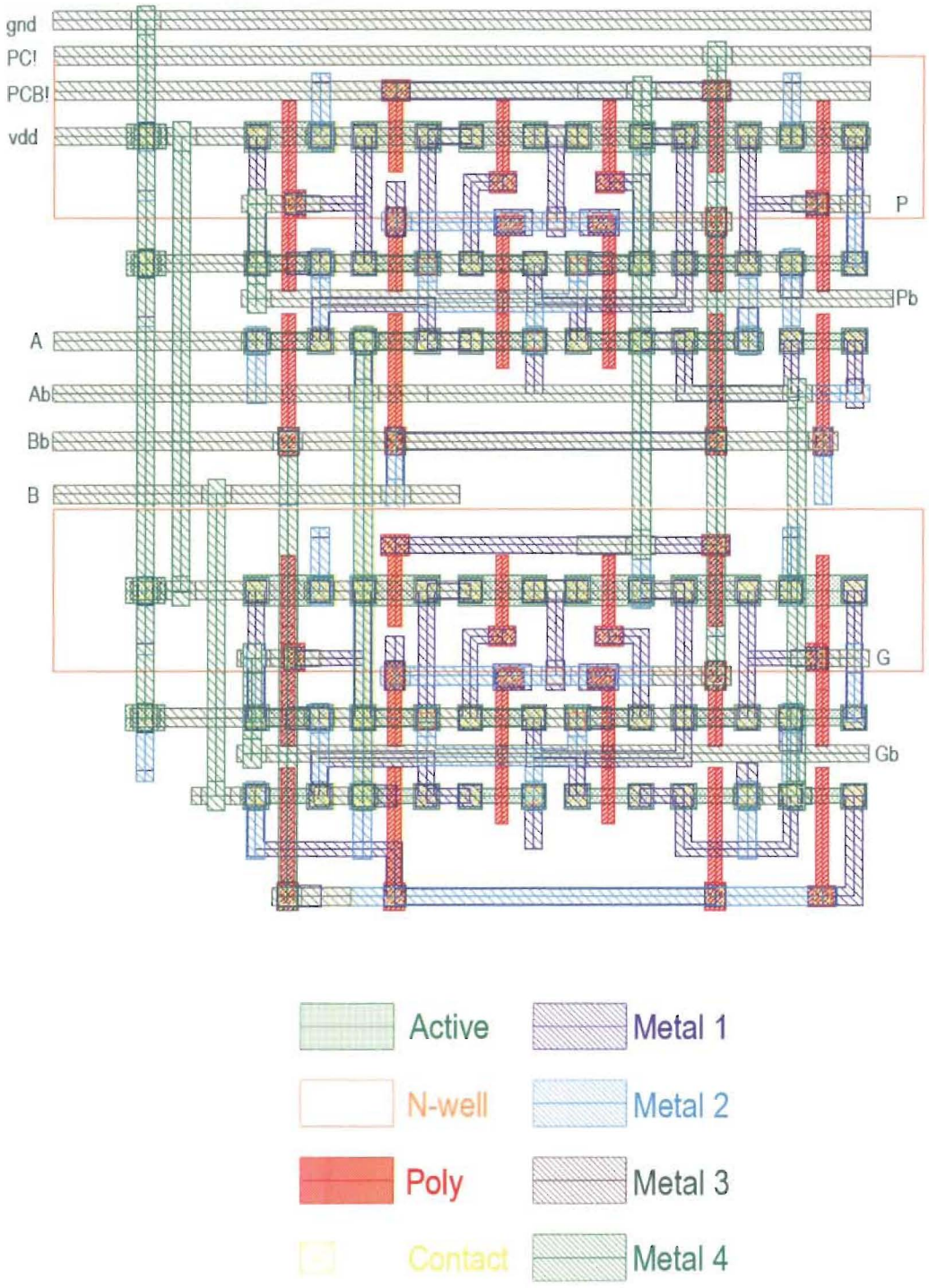



**Figure 4-4: N-type propagate/generate generator layout**

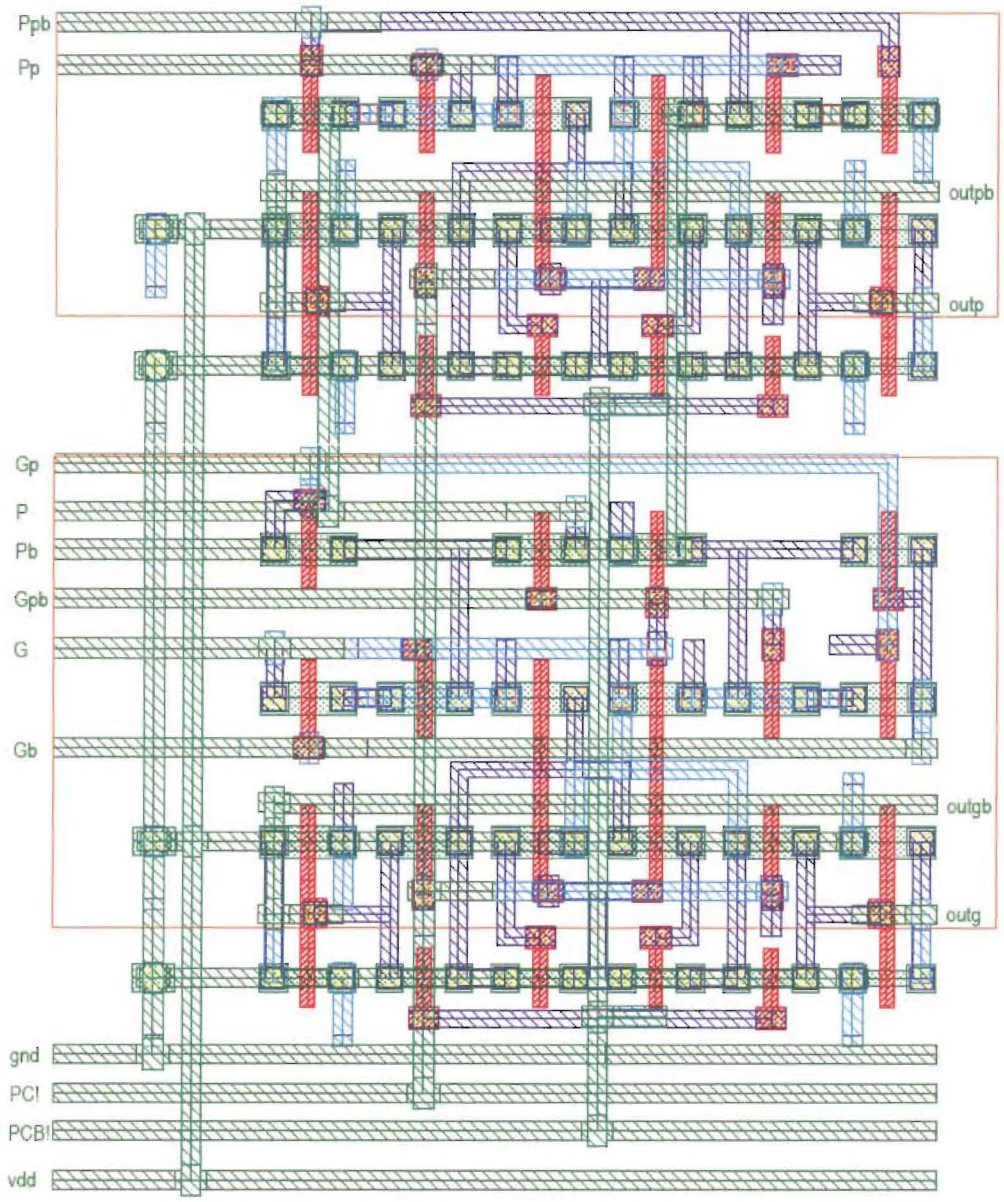



**Figure 4-5: P-type DOT operator layout**

The final adder layout is shown in Figure 4-6. The power rails run west to east, supplying the cells with power. The cascaded pipeline stages can be visually observed in the final layout. The low density of the adder layout is due to the inverted structure of the N and P-type cells. In order to share the same power rails, N-type cells are located below power rails, while the P-type cells are above the rails. Empty spaces can be found due to the varying heights of the cells. Routing between cells was accomplished using a horizontal Metal 3 layer and a vertical Metal 4 layer.

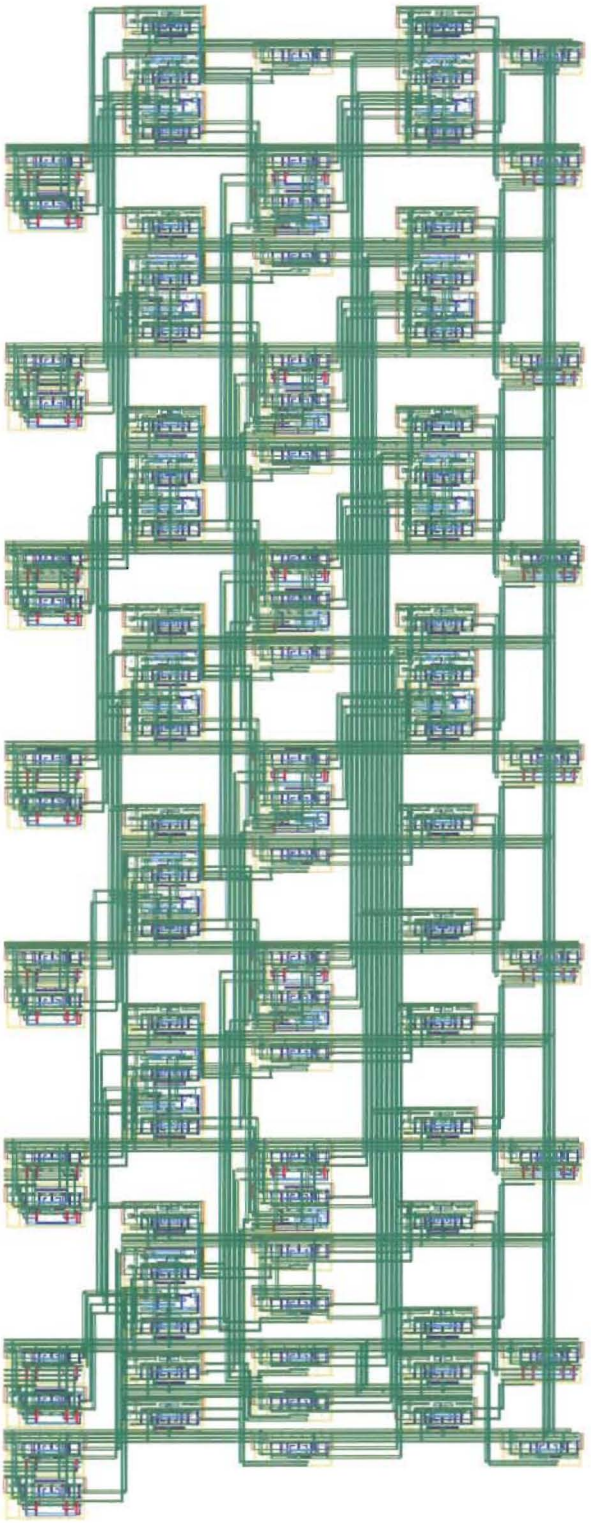


**Figure 4-6: Layout of the QAPG 8-bit Kogge-Stone CLA**

## 4.3 Verification and Simulation Results

### 4.3.1 Extracted Parasitic Netlist Simulations

Design rule checking and layout versus schematic checking was performed to physically verify the layout of the adder. Parasitic components found in the layout were then extracted and HSPICE simulations were performed to verify the correct operation of the adder. The power clock energy of the QAPG CLA is shown below in Figure 4-7 utilizing a sinusoidal and square power clock waveform operating at 0.5 V with a frequency of 100 MHz.

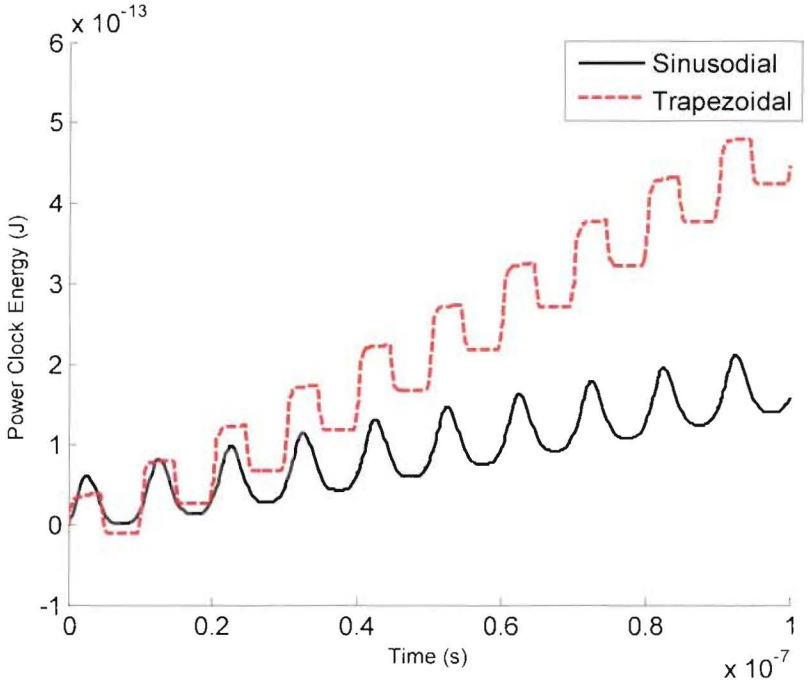



**Figure 4-7: Energy dissipation of the power clock of an extracted layout for an 8-bit Kogge-Stone CLA**

The energy plot demonstrates the sourcing and recovery of energy in the power clock. This figure resembles the shape of the power clock energy plot found in Figure 3-9, which indicate that parasitic components have a small effect on the recycling of energy in the QAPG family, however the added parasitics increase the energy dissipation of the power clock by approximately three times. The total energy dissipation per addition operation of the extracted QAPG CLA layout was analyzed and is shown in Figure 4-8. The energy dissipation was observed at 1 MHz, 10 MHz, 50 MHz, and 100 MHz using the minimum supply voltage capable of operation at the specific frequency. The supply voltages are shown next to each data point on the figure.

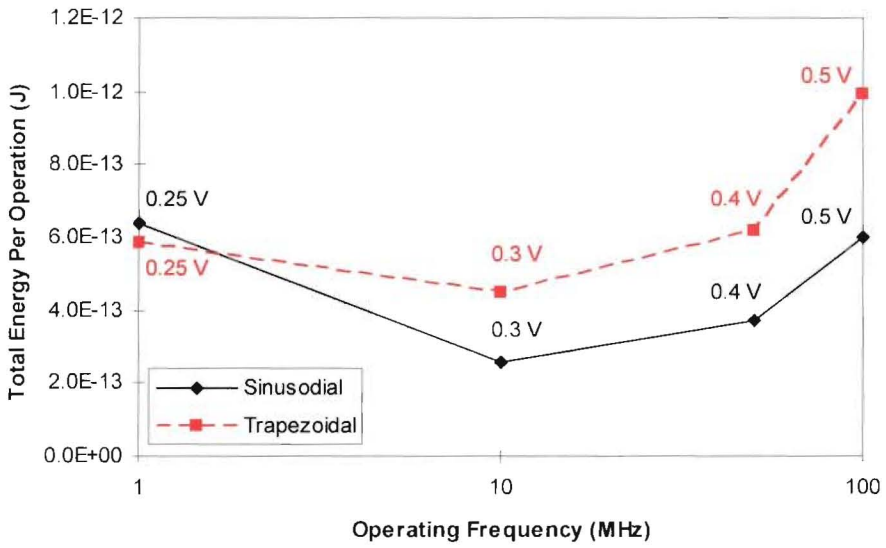



**Figure 4-8: Energy dissipation versus operating frequency of an extracted layout for an 8-bit Kogge-Stone CLA**

The total energy per addition operation was found to be 1.2 to 7.8 times higher than in Figure 3-21 and Figure 3-22, where parasitics were not included in the simulations. This is due to in part by the parasitic components, as well as the need to increase the supply voltage at certain frequencies. Observed again is the QAPG adder operating with a sinusoidal power clock dissipating less energy than with a trapezoidal power clock at higher frequencies. Also observed is that static energy contributes approximately 25% of the total energy dissipated.

### 4.3.2 Observations and Discussion

A difference found between the QAPG full-adder and CLA simulations are the minimum supply voltages at which the circuit can operate at. This is due to the pass-gate evaluation tree height, as a larger supply voltage is needed for larger trees. The largest tree height found in the full-adder was two transistors, while the CLA had three transistors. Also, as expected, the supply voltage dictates the maximum operating frequency.

Static energy was much more prominent in the extracted layout simulations. An 11% increase in static energy contribution to the total energy was observed in the post-layout simulations. This increase in leakage can be partly attributed to the shallow trench isolation induced stress phenomena documented by STMicroelectronics [28]. Threshold voltages are affected by the distance between the edge of the polysilicon gate and the edge of the active region due to the difference in thermal coefficients and mechanical properties between the Si and $SiO_2$ interface. An increasing distance between the polysilicon and active edges will lower the threshold voltage in both NMOS and

PMOS transistors, which will increase the static energy of the QAPG circuits. In the QAPG CLA layout, the distance between the edge of the polysilicon gate and active edge is slightly larger than the minimum size, leading to a lower threshold voltage and increased leakage.

# CHAPTER 5: CONCLUSION

The adiabatic approach to VLSI circuit design is an attractive method in designing low energy dissipating digital applications. By recycling the energy stored in capacitances throughout the circuit, extremely low energy dissipation can be achieved. Previously proposed adiabatic logic families use multi-phase clocking structures which were difficult to implement into the adiabatic system and would result in inefficient charge recovery. Adiabatic families using single-phase clocking structures were also previously proposed which attempted to eliminate these drawbacks. However, these families were not designed for low-voltage applications, which would help to further decrease energy dissipation.

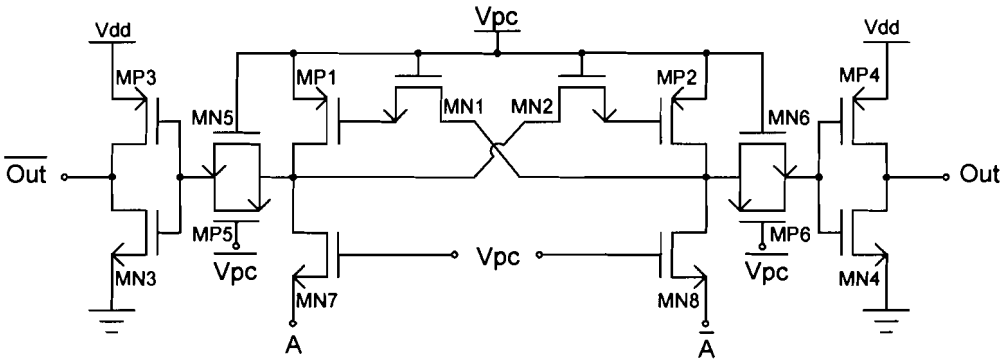
This thesis presents a low-voltage, high-speed quasi-adiabatic logic family utilizing a single-phase sinusoidal or trapezoidal clocking scheme. A full-adder and CLA were designed in the 90 nm standard CMOS process to verify the circuit operation and evaluate the energy efficiency of the adiabatic logic family. HSPICE simulations were performed, analyzing energy dissipation across varying supply voltages, operational frequencies, load capacitances, process corners, and technology nodes. Also, post-layout simulations were performed to verify correct operation of the QAPG CLA when extracted parasitics are included. QAPG is able to obtain substantial energy reduction at low supply voltages in comparison with other single-phase clocked adiabatic logic families. QAPG can dissipate between 11% and 40% of the energy consumed by other previously

proposed adiabatic logic families across a range of supply voltages, power clock frequencies, and capacitive loads.

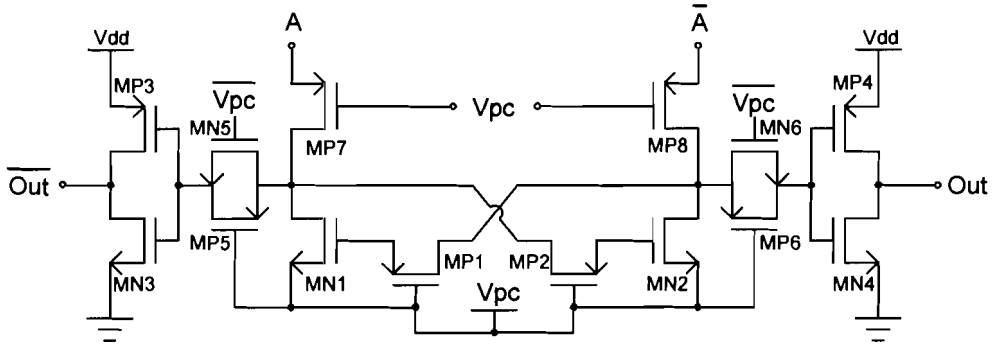
Future research in the QAPG logic would include an investigation into the effect of routing and transistor placement in layouts to minimize the effects of parasitics and leakage from the shallow trench isolation induced stress phenomena. Also, to utilize the die area more efficiently, QAPG cells can be designed to eliminate the empty portions of the die. Creation of a QAPG standard cell library for automated layout generation would be useful in designing larger and more complex circuitry. Also, an efficient adiabatic power clock generator can be investigated that can be placed on the same die as the adiabatic circuit to increase the efficiency of the charge recovery. High and low threshold voltages available in the 90 nm CMOS process can be analyzed to further optimize QAPG logic gates.

# APPENDICES

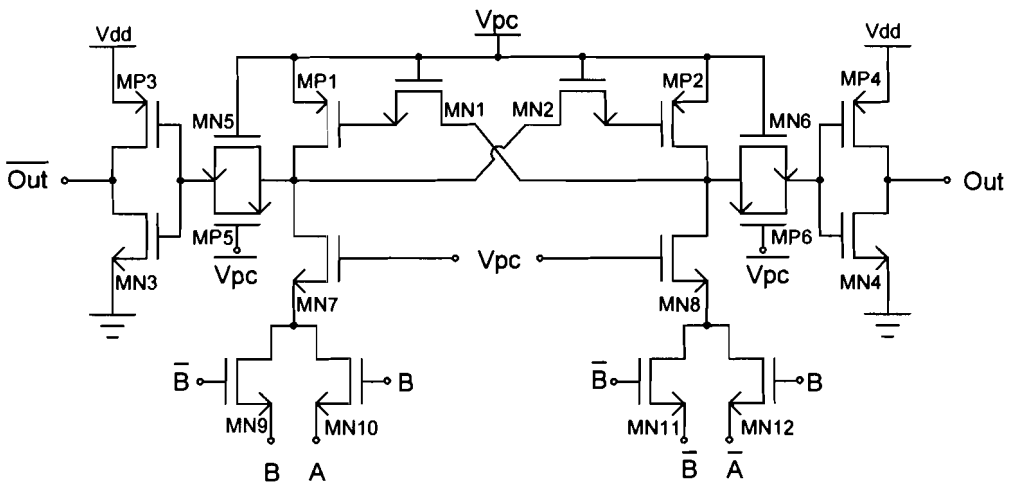
## Appendix A: QAPG Logic Gate Schematics



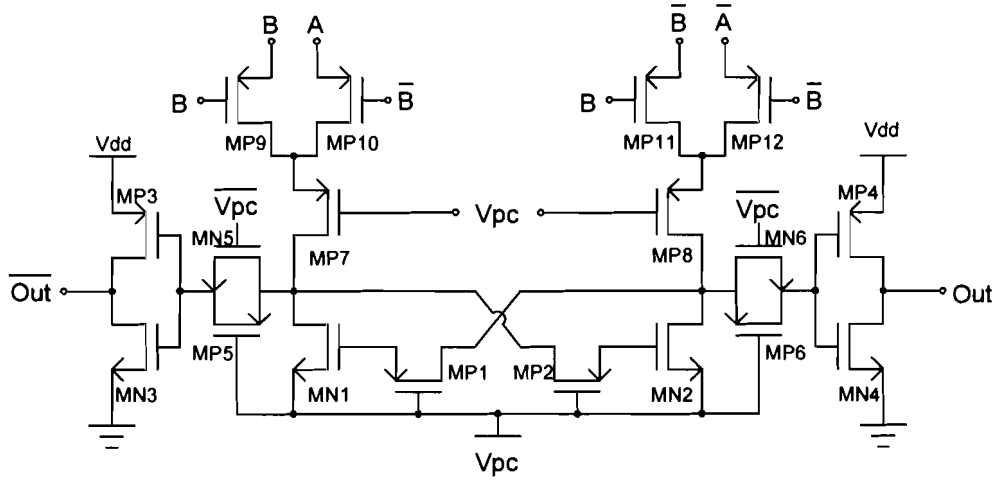

**N-type Buffer/Inverter gate**



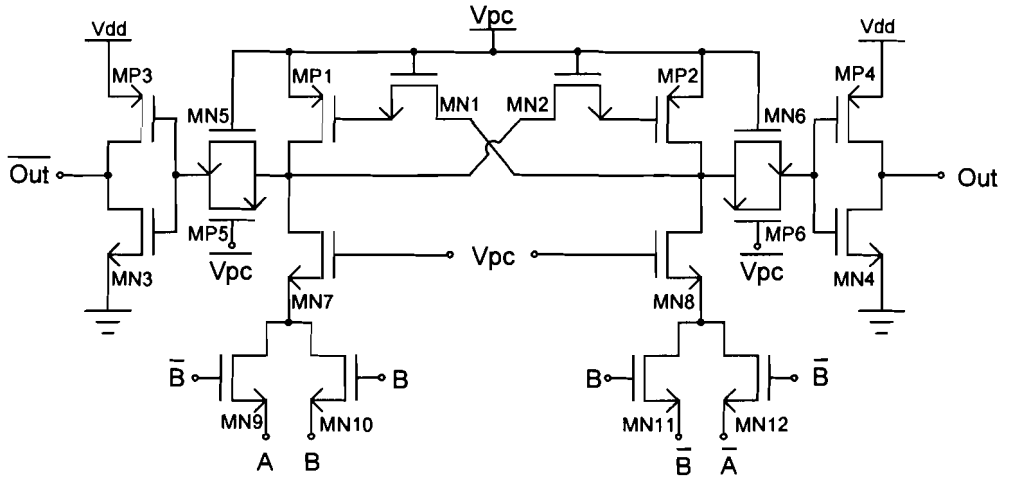

**P-type Buffer/Inverter gate**



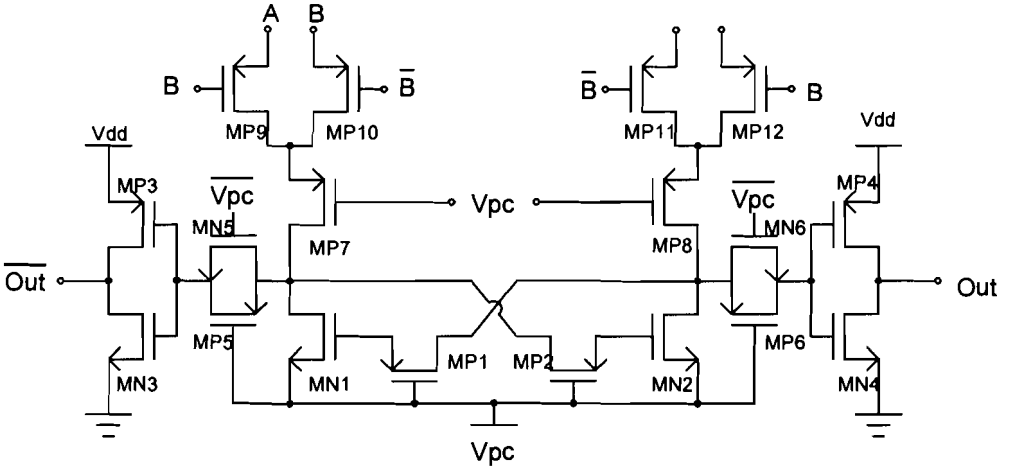

**N-type AND gate**



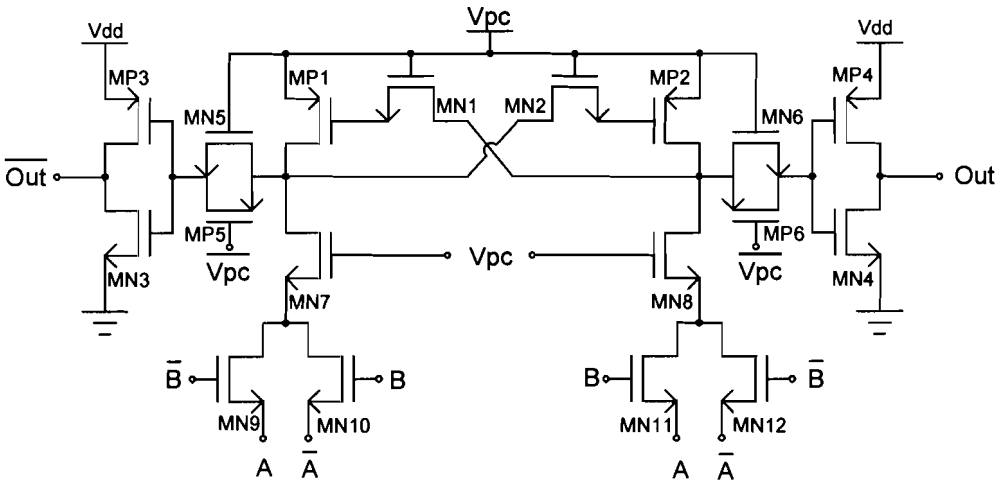

**P-type AND gate**



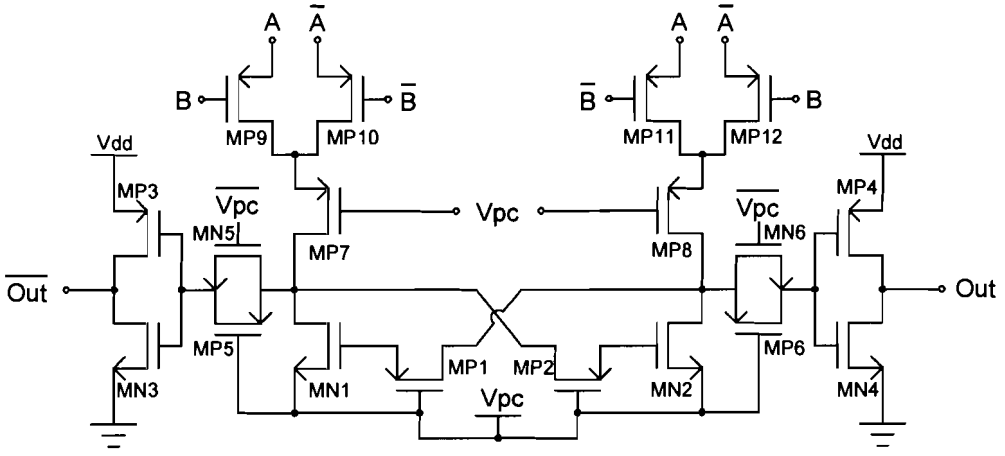

**N-type OR gate**



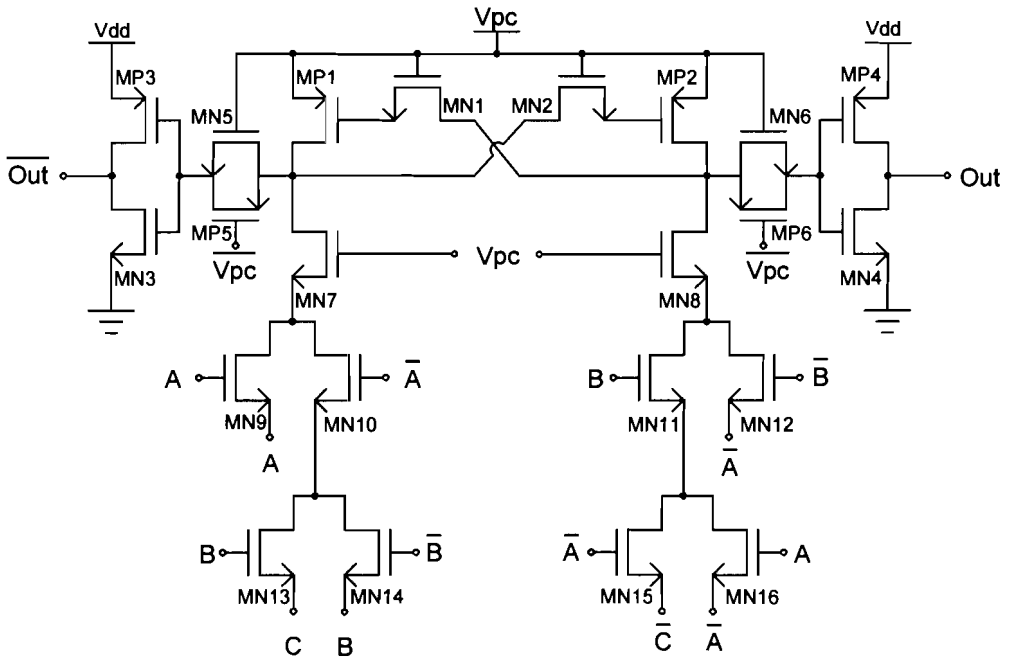

**P-type OR gate**



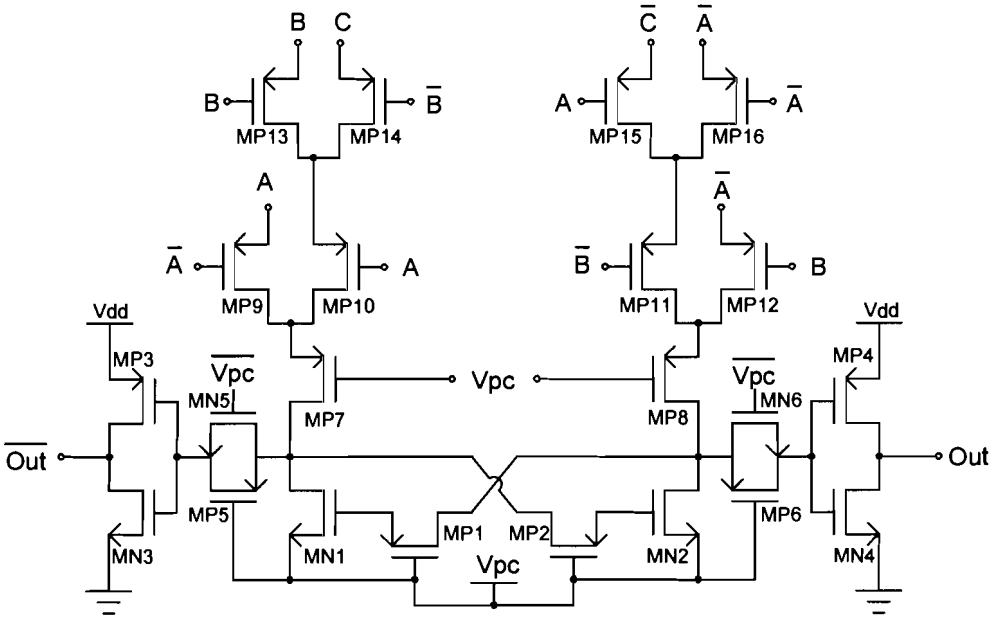

**N-type XOR gate**




**P-type XOR gate**




**N-type AND-OR gate**




**P-type AND-OR gate**

# Appendix B: QAPG 8-bit Kogge-Stone Adder Netlist

```
QAPG 8bit Kogge-Stone Adder


*declare widths of transistors (abx = 0.2)
.param widn='1*lm#'
.param widp='2*lm#'
.param widbn='1*lm#'
.param widbp='2*lm#'
.param widt='1*lm#'

.subckt QAPGXOR2p pc pcbar ina inab inb inbb static outa outbara
xMBP1 B1 pc out vdd pt_svt w=widbp
xMBP2 B2 pc outbar vdd pt_svt w=widbp
xMN1 outbar B1 pc gnd nt_svt w=widn
xMN2 out B2 pc gnd nt_svt w=widn
xMP1 outbar pc a vdd pt_svt w=widp
xMP2 out pc ab vdd pt_svt w=widp
xMP15 ab ina inb vdd pt_svt w=widn
xMP16 ab inab inbb vdd pt_svt w=widn
xMP17 a inab inb vdd pt_svt w=widn
xMP18 a ina inbb vdd pt_svt w=widn
xMPt out pc outt vdd pt_svt w=widp
xMNtb out pcbar outt gnd nt_svt w=widn
xMPbt outbar pc outbart vdd pt_svt w=widp
xMNbtb outbar pcbar outbart gnd nt_svt w=widn
xMPO1 outa outbart static static pt_svt w=widp
xMNO1 outa outbart gnd gnd nt_svt w=widt
xMPO2 outbara outt static static pt_svt w=widp
xMNO2 outbara outt gnd gnd nt_svt w=widt
.ends
.subckt QAPGXOR2n pc pcbar ina inab inb inbb static outa outbara
xMBN1 B1 pc out gnd nt_svt w=widbn
xMBN2 B2 pc outbar gnd nt_svt w=widbn
xMP1 outbar B1 pc pc pt_svt w=widp
xMP2 out B2 pc pc pt_svt w=widp
xMN1 outbar pc a gnd nt_svt w=widn
xMN2 out pc ab gnd nt_svt w=widn
xMN15 ab inbb ina gnd nt_svt w=widn
xMN16 ab inb inab gnd nt_svt w=widn
xMN17 a inb ina gnd nt_svt w=widn
xMN18 a inbb inab gnd nt_svt w=widn
xMNt out pc outt gnd nt_svt w=widn
xMNtb out pcbar outt vdd pt_svt w=widp
xMNbt outbar pc outbart gnd nt_svt w=widn
xMNbtb outbar pcbar outbart vdd pt_svt w=widp
xMPO1 outa outbart static static pt_svt w=widp
xMNO1 outa outbart gnd gnd nt_svt w=widt
xMPO2 outbara outt static static pt_svt w=widp
xMNO2 outbara outt gnd gnd nt_svt w=widt
.ends
.subckt QAPGAND2n pc pcbar ina inab inb inbb static outa outbara
xMBN3 B3 pc out1 gnd nt_svt w=widbn
```

```
xMBN4 B4 pc outbar1 gnd nt_svt w=widbn
xMP3 outbar1 B3 pc pc pt_svt w=widp
xMP4 out1 B4 pc pc pt_svt w=widp
xMN11 outbar1 pc c gnd nt_svt w=widn
xMN12 out1 pc cb gnd nt_svt w=widn
xMN15 c inb inab gnd nt_svt w=widn
xMN16 c inbb inbb gnd nt_svt w=widn
xMN17 cb inbb inb gnd nt_svt w=widn
xMN18 cb inb ina gnd nt_svt w=widn
xMN1t out1 pc outt gnd nt_svt w=widn
xMN1tb out1 pcbar outt vdd pt_svt w=widp
xMN1bt outbar1 pc outbart gnd nt_svt w=widn
xMN1btb outbar1 pcbar outbart vdd pt_svt w=widp
xMPO1 outa outbart static static pt_svt w=widp
xMNO1 outa outbart gnd gnd nt_svt w=widt
xMPO2 outbara outt static static pt_svt w=widp
xMNO2 outbara outt gnd gnd nt_svt w=widt
.ends
.subckt QAPGAND2p pc pcbar ina inab inb inbb static outa outbara
xMBP3 B3 pc out1 vdd pt_svt w=widbp
xMBP4 B4 pc outbar1 vdd pt_svt w=widbp
xMN3 outbar1 B3 pc gnd nt_svt w=widn
xMN4 out1 B4 pc gnd nt_svt w=widn
xMP11 outbar1 pc c gnd pt_svt w=widp
xMP12 out1 pc cb gnd pt_svt w=widp
xMP15 c inb inbb vdd pt_svt w=widp
xMP16 c inbb inab vdd pt_svt w=widp
xMP17 cb inbb ina vdd pt_svt w=widp
xMP18 cb inb inb vdd pt_svt w=widp
xMP1t out1 pc outt vdd pt_svt w=widp
xMN1tb out1 pcbar outt gnd nt_svt w=widn
xMP1bt outbar1 pc outbart vdd pt_svt w=widp
xMN1btb outbar1 pcbar outbart gnd nt_svt w=widn
xMPO1 outa outbart static static pt_svt w=widp
xMNO1 outa outbart gnd gnd nt_svt w=widt
xMPO2 outbara outt static static pt_svt w=widp
xMNO2 outbara outt gnd gnd nt_svt w=widt
.ends
.subckt QAPGOR2n pc pcbar ina inab inb inbb static outa outbara
xMBN3 B3 pc out1 gnd nt_svt w=widbn
xMBN4 B4 pc outbar1 gnd nt_svt w=widbn
xMP3 outbar1 B3 pc pc pt_svt w=widp
xMP4 out1 B4 pc pc pt_svt w=widp
xMN11 outbar1 pc c gnd nt_svt w=widn
xMN12 out1 pc cb gnd nt_svt w=widn
xMN15 c inb inbb gnd nt_svt w=widn
xMN16 c inbb inab gnd nt_svt w=widn
xMN17 cb inbb ina gnd nt_svt w=widn
xMN18 cb inb inb gnd nt_svt w=widn
xMN1t out1 pc outt gnd nt_svt w=widn
xMN1tb out1 pcbar outt vdd pt_svt w=widp
xMN1bt outbar1 pc outbart gnd nt_svt w=widn
xMN1btb outbar1 pcbar outbart vdd pt_svt w=widp
xMPO1 outa outbart static static pt_svt w=widp
xMNO1 outa outbart gnd gnd nt_svt w=widt
xMPO2 outbara outt static static pt_svt w=widp
xMNO2 outbara outt gnd gnd nt_svt w=widt
```

```
.ends
.subckt QAPGBUFp pc pcbar ina inab static outa outbara
xMBP1 B1 pc out vdd pt_svt w=widbp
xMBP2 B2 pc outbar vdd pt_svt w=widbp
xMN1 outbar B1 pc gnd nt_svt w=widn
xMN2 out B2 pc gnd nt_svt w=widn
xMP1 outbar pc inab vdd pt_svt w=widp
xMP2 out pc ina vdd pt_svt w=widp
xMPt out pc outt vdd pt_svt w=widp
xMNtb out pcbar outt gnd nt_svt w=widn
xMPbt outbar pc outbart vdd pt_svt w=widp
xMNbtb outbar pcbar outbart gnd nt_svt w=widn
xMPO1 outa outbart static static pt_svt w=widp
xMNO1 outa outbart gnd gnd nt_svt w=widt
xMPO2 outbara outt static static pt_svt w=widp
xMNO2 outbara outt gnd gnd nt_svt w=widt
.ends
.subckt QAPGBUFn pc pcbar ina inab static outa outbara
xMBN1 B1 pc out gnd nt_svt w=widbn
xMBN2 B2 pc outbar gnd nt_svt w=widbn
xMP1 outbar B1 pc pc pt_svt w=widp
xMP2 out B2 pc pc pt_svt w=widp
xMN1 outbar pc inab gnd nt_svt w=widn
xMN2 out pc ina gnd nt_svt w=widn
xMNt out pc outt gnd nt_svt w=widn
xMNtb out pcbar outt vdd pt_svt w=widp
xMNbt outbar pc outbart gnd nt_svt w=widn
xMNbtb outbar pcbar outbart vdd pt_svt w=widp
xMPO1 outa outbart static static pt_svt w=widp
xMNO1 outa outbart gnd gnd nt_svt w=widt
xMPO2 outbara outt static static pt_svt w=widp
xMNO2 outbara outt gnd gnd nt_svt w=widt
.ends
.subckt QAPGpgn pc pcbar ina inab inb inbb static outg outbarg outp
outbarp
xMBN1 B1 pc out gnd nt_svt w=widbn
xMBN2 B2 pc outbar gnd nt_svt w=widbn
xMP1 outbar B1 pc pc pt_svt w=widp
xMP2 out B2 pc pc pt_svt w=widp
xMN1 outbar pc a gnd nt_svt w=widn
xMN2 out pc ab gnd nt_svt w=widn
xMN5 ab inbb ina gnd nt_svt w=widn
xMN6 ab inb inab gnd nt_svt w=widn
xMN7 a inb ina gnd nt_svt w=widn
xMN8 a inbb inab gnd nt_svt w=widn
xMNt out pc outt gnd nt_svt w=widn
xMNtb out pcbar outt vdd pt_svt w=widp
xMNbt outbar pc outbart gnd nt_svt w=widn
xMNbtb outbar pcbar outbart vdd pt_svt w=widp
xMPO1 outp outbart static static pt_svt w=widp
xMNO1 outp outbart gnd gnd nt_svt w=widt
xMPO2 outbarp outt static static pt_svt w=widp
xMNO2 outbarp outt gnd gnd nt_svt w=widt

xMBN3 B3 pc out1 gnd nt_svt w=widbn
xMBN4 B4 pc outbar1 gnd nt_svt w=widbn
xMP3 outbar1 B3 pc pc pt_svt w=widp
```

```
xMP4 out1 B4 pc pc pt_svt w=widp
xMN11 outbar1 pc c gnd nt_svt w=widn
xMN12 out1 pc cb gnd nt_svt w=widn
xMN15 c inb inab gnd nt_svt w=widn
xMN16 c inbb inbb gnd nt_svt w=widn
xMN17 cb inbb inb gnd nt_svt w=widn
xMN18 cb inb ina gnd nt_svt w=widn
xMN1t out1 pc out1t gnd nt_svt w=widn
xMN1tb out1 pcbar out1t vdd pt_svt w=widp
xMN1bt outbar1 pc outbar1t gnd nt_svt w=widn
xMN1btb outbar1 pcbar outbar1t vdd pt_svt w=widp
xMP1O1 outg outbar1t static static pt_svt w=widp
xMN1O1 outg outbar1t gnd gnd nt_svt w=widt
xMP1O2 outbarg out1t static static pt_svt w=widp
xMN1O2 outbarg out1t gnd gnd nt_svt w=widt
.ends

.subckt QAPGcopn pc pcbar ina inab inb inbb inc incb ind indb static
outg outbarg outp outbarp
xMBN1 B1 pc out1 gnd nt_svt w=widbn
xMBN2 B2 pc outbar1 gnd nt_svt w=widbn
xMP1 outbar1 B1 pc pc pt_svt w=widp
xMP2 out1 B2 pc pc pt_svt w=widp
xMN1 outbar1 pc ab gnd nt_svt w=widn
xMN2 out1 pc a gnd nt_svt w=widn
xMN3 a ina ina gnd nt_svt w=widn
xMN4 a inab b gnd nt_svt w=widn
xMN5 b inb inc gnd nt_svt w=widn
xMN6 b inbb inb gnd nt_svt w=widn
xMN7 ab inbb inab gnd nt_svt w=widn
xMN8 ab inb bb gnd nt_svt w=widn
xMN9 bb inab incb gnd nt_svt w=widn
xMN10 bb ina inab gnd nt_svt w=widn
xMN1t out1 pc out1t gnd nt_svt w=widn
xMN1tb out1 pcbar out1t vdd pt_svt w=widp
xMN1bt outbar1 pc outbar1t gnd nt_svt w=widn
xMN1btb outbar1 pcbar outbar1t vdd pt_svt w=widp
xMP1O1 outg outbar1t static static pt_svt w=widp
xMN1O1 outg outbar1t gnd gnd nt_svt w=widt
xMP1O2 outbarg out1t static static pt_svt w=widp
xMN1O2 outbarg out1t gnd gnd nt_svt w=widt

xMBN3 B3 pc out gnd nt_svt w=widbn
xMBN4 B4 pc outbar gnd nt_svt w=widbn
xMP3 outbar B3 pc pc pt_svt w=widp
xMP4 out B4 pc pc pt_svt w=widp
xMN11 outbar pc c gnd nt_svt w=widn
xMN12 out pc cb gnd nt_svt w=widn
xMN15 c inb indb gnd nt_svt w=widn
xMN16 c inbb inbb gnd nt_svt w=widn
xMN17 cb inbb inb gnd nt_svt w=widn
xMN18 cb inb ind gnd nt_svt w=widn
xMNt out pc outt gnd nt_svt w=widn
xMNtb out pcbar outt vdd pt_svt w=widp
xMNbt outbar pc outbart gnd nt_svt w=widn
xMNbtb outbar pcbar outbart vdd pt_svt w=widp
xMPO1 outp outbart static static pt_svt w=widp
```

```
xMNO1 outp outbart gnd gnd nt_svt w=widt
xMPO2 outbarp outt static static pt_svt w=widp
xMNO2 outbarp outt gnd gnd nt_svt w=widt
.ends

.subckt QAPGcopp pc pcbar ina inab inb inbb inc incb ind indb static
outg outbarg outp outbarp
xMBP1 B1 pc out vdd pt_svt w=widbp
xMBP2 B2 pc outbar vdd pt_svt w=widbp
xMN1 outbar B1 pc gnd nt_svt w=widn
xMN2 out B2 pc gnd nt_svt w=widn
xMP1 outbar pc ab vdd pt_svt w=widp
xMP2 out pc a vdd pt_svt w=widp
xMP5 a inab ina vdd pt_svt w=widp
xMP6 a ina b vdd pt_svt w=widp
xMP7 b inb inb vdd pt_svt w=widp
xMP8 b inbb inc vdd pt_svt w=widp
xMP9 ab inb inab vdd pt_svt w=widp
xMP10 ab inbb bb vdd pt_svt w=widp
xMP11 bb ina incb vdd pt_svt w=widp
xMP12 bb inab inab vdd pt_svt w=widp
xMPt out pc outt vdd pt_svt w=widp
xMNtb out pcbar outt gnd nt_svt w=widn
xMPbt outbar pc outbart vdd pt_svt w=widp
xMNbtb outbar pcbar outbart gnd nt_svt w=widn
xMPO1 outg outbart static static pt_svt w=widp
xMNO1 outg outbart gnd gnd nt_svt w=widt
xMPO2 outbarg outt static static pt_svt w=widp
xMNO2 outbarg outt gnd gnd nt_svt w=widt

xMBP3 B3 pc out1 vdd pt_svt w=widbp
xMBP4 B4 pc outbar1 vdd pt_svt w=widbp
xMN3 outbar1 B3 pc gnd nt_svt w=widn
xMN4 out1 B4 pc gnd nt_svt w=widn
xMP19 outbar1 pc c gnd pt_svt w=widp
xMP20 out1 pc cb gnd pt_svt w=widp
xMP15 c inb inbb vdd pt_svt w=widp
xMP16 c inbb indb vdd pt_svt w=widp
xMP17 cb inbb ind vdd pt_svt w=widp
xMP18 cb inb inb vdd pt_svt w=widp
xMP1t out1 pc out1t vdd pt_svt w=widp
xMN1tb out1 pcbar out1t gnd nt_svt w=widn
xMP1bt outbar1 pc outbar1t vdd pt_svt w=widp
xMN1btb outbar1 pcbar outbar1t gnd nt_svt w=widn
xMP1O1 outp outbar1t static static pt_svt w=widp
xMN1O1 outp outbar1t gnd gnd nt_svt w=widt
xMP1O2 outbarp out1t static static pt_svt w=widp
xMN1O2 outbarp out1t gnd gnd nt_svt w=widt
.ends

*logic START
*define Power Clock

vpc pc gnd SIN ('vdd/2' 'vdd/2' '1/tperiod')
vpcbar pcbar gnd SIN ('vdd/2' 'vdd/2' '1/tperiod' 0 0 180)

vnn n gnd vdd
```

```
vpp p gnd vdd


x0 PC PCBar a0 a0b b0 b0b n G00 G00b P00 P00b QAPGPGn
xbpp00 PC PCBar P00 P00b p P10 P10b QAPGBUFp
xbgp00 PC PCBar G00 G00b p G10 G10b QAPGBUFp
xbpn10 PC PCBar P10 P10b n P20 P20b QAPGBUFn
xbgn10 PC PCBar G10 G10b n G20 G20b QAPGBUFn
xbpp20 PC PCBar P20 P20b p P30 P30b QAPGBUFp
xbgp20 PC PCBar G20 G20b p G30 G30b QAPGBUFp
xbnp30 PC PCBar P30 P30b n Z0 Z0b QAPGBUFn

x1 PC PCBar a1 a1b b1 b1b n G01 G01b P01 P01b QAPGPGn
xbpp01 PC PCBar P01 P01b p bP11 bP11b QAPGBUFp
xyp01 PC PCBar G01 G01b P01 P01b G00 G00b P00 P00b p G11 G11b P11 P11b
QAPGCOPp
xbpn11 PC PCBar P11 P11b n P21 P21b QAPGBUFn
xbgn11 PC PCBar G11 G11b n G21 G21b QAPGBUFn
xbbp11 PC PCBar bP11 bP11b n bp21 bP21b QAPGBUFn
xbgp21 PC PCBar G21 G21b p G31 G31b QAPGBUFp
xbbp21 PC PCBar bP21 bP21b p bP31 bP31b QAPGBUFp
xxorn1 PC PCBar bP31 bP31b G30 G30b n Z1 Z1b QAPGXOR2n

x2 PC PCBar a2 a2b b2 b2b n G02 G02b P02 P02b QAPGPGn
xbpp02 PC PCBar P02 P02b p bP12 bP12b QAPGBUFp
xyp02 PC PCBar G02 G02b P02 P02b G01 G01b P01 P01b p G12 G12b P12 P12b
QAPGCOPp
xbpp12 PC PCBar bP12 bP12b n bP22 bP22b QAPGBUFn
xyn12 PC PCBar G12 G12b P12 P12b G10 G10b P10 P10b n G22 G22b P22 P22b
QAPGCOPn
xbgp22 PC PCBar G22 G22b p G32 G32b QAPGBUFp
xbbp22 PC PCBar bP22 bP22b p bP32 bP32b QAPGBUFp
xxorn2 PC PCBar bP32 bP32b G31 G31b n Z2 Z2b QAPGXOR2n

x3 PC PCBar a3 a3b b3 b3b n G03 G03b P03 P03b QAPGPGn
xbpp03 PC PCBar P03 P03b p bP13 bP13b QAPGBUFp
xyp03 PC PCBar G03 G03b P03 P03b G02 G02b P02 P02b p G13 G13b P13 P13b
QAPGCOPp
xbpp13 PC PCBar bP13 bP13b n bP23 bP23b QAPGBUFn
xyn13 PC PCBar G13 G13b P13 P13b G11 G11b P11 P11b n G23 G23b P23 P23b
QAPGCOPn
xbgp23 PC PCBar G23 G23b p G33 G33b QAPGBUFp
xbbp23 PC PCBar bP23 bP23b p bP33 bP33b QAPGBUFp
xxorn3 PC PCBar bP33 bP33b G32 G32b n Z3 Z3b QAPGXOR2n

x4 PC PCBar a4 a4b b4 b4b n G04 G04b P04 P04b QAPGPGn
xbpp04 PC PCBar P04 P04b p bP14 bP14b QAPGBUFp
xyp04 PC PCBar G04 G04b P04 P04b G03 G03b P03 P03b p G14 G14b P14 P14b
QAPGCOPp
xbpp14 PC PCBar bP14 bP14b n bP24 bP24b QAPGBUFn
xyn14 PC PCBar G14 G14b P14 P14b G12 G12b P12 P12b n G24 G24b P24 P24b
QAPGCOPn
xbpp24 PC PCBar bP24 bP24b p bP34 bP34b QAPGBUFp
xyp24 PC PCBar G24 G24b P24 P24b G20 G20b P20 P20b p G34 G34b P34 P34b
QAPGCOPp
xxorn4 PC PCBar bP34 bP34b G33 G33b n Z4 Z4b QAPGXOR2n
```

```
x5 PC PCBar a5 a5b b5 b5b n G05 G05b P05 P05b QAPGPGn
xbpp05 PC PCBar P05 P05b p bP15 bP15b QAPGBUFp
xyp05 PC PCBar G05 G05b P05 P05b G04 G04b P04 P04b p G15 G15b P15 P15b
QAPGCOPp
xbpp15 PC PCBar bP15 bP15b n bP25 bP25b QAPGBUFn
xyn15 PC PCBar G15 G15b P15 P15b G13 G13b P13 P13b n G25 G25b P25 P25b
QAPGCOPn
xbpp25 PC PCBar bP25 bP25b p bP35 bP35b QAPGBUFp
xyp25 PC PCBar G25 G25b P25 P25b G21 G21b P21 P21b p G35 G35b P35 P35b
QAPGCOPp
xxorn5 PC PCBar bP35 bP35b G34 G34b n Z5 Z5b QAPGXOR2n

x6 PC PCBar a6 a6b b6 b6b n G06 G06b P06 P06b QAPGPGn
xbpp06 PC PCBar P06 P06b p bP16 bP16b QAPGBUFp
xyp06 PC PCBar G06 G06b P06 P06b G05 G05b P05 P05b p G16 G16b P16 P16b
QAPGCOPp
xbpp16 PC PCBar bP16 bP16b n bP26 bP26b QAPGBUFn
xyn16 PC PCBar G16 G16b P16 P16b G14 G14b P14 P14b n G26 G26b P26 P26b
QAPGCOPn
xbpp26 PC PCBar bP26 bP26b p bP36 bP36b QAPGBUFp
xyp26 PC PCBar G26 G26b P26 P26b G22 G22b P22 P22b p G36 G36b P36 P36b
QAPGCOPp
xxorn6 PC PCBar bP36 bP36b G35 G35b n Z6 Z6b QAPGXOR2n

x7 PC PCBar a7 a7b b7 b7b n G07 G07b P07 P07b QAPGPGn
xbpp07 PC PCBar P07 P07b p bP17 bP17b QAPGBUFp
xyp07 PC PCBar G07 G07b P07 P07b G06 G06b P06 P06b p G17 G17b P17 P17b
QAPGCOPp
xbpp17 PC PCBar bP17 bP17b n bP27 bP27b QAPGBUFn
xyn17 PC PCBar G17 G17b P17 P17b G15 G15b P15 P15b n G27 G27b P27 P27b
QAPGCOPn
xbpp27 PC PCBar bP27 bP27b p bP37 bP37b QAPGBUFp
xyp27 PC PCBar G27 G27b P27 P27b G23 G23b P23 P23b p G37 G37b P37 P37b
QAPGCOPp
xxorn7 PC PCBar bP37 bP37b G36 G36b n Z7 Z7b QAPGXOR2n

xbnp38 PC PCBar P37 P37b n Z8 Z8b QAPGBUFn

.end
```

## Appendix C: Sample Skill Code for a QAPG N-type XOR gate

```
; File Name: nxor2.il
; Written by: Edward Loo
; Date: june 8 2007
; Last Update: N/A
; Cell: nxor
; version: 1
;
; Height:  um
; Width:   um
;
; Input: A Ab B Bb
; Output: out outb
; InputOutput: pc! pcb!


library="qapg90"

;-------------------------------------------------------
;
;-------------------------------------------------------
procedure(nxor()
let( ( cnm nxor_cvw ma_cvw mal_cvw mp_cvw mpl_cvw mplx_cvw m2m1_cvw
m2m1x_cvw m3m2_cvw
       west  south pex
       wn wp ods ods2 mint cod3 prow nrow1 nrow2

)

cnm = "nxor"

nxor_cvw = dbOpenCellViewByType(library cnm "layout" "maskLayout" "w")
printf("Cellname: %s" cnm)

;---------------------------------------
; Open db objects
;---------------------------------------
ma_cvw = dbOpenCellViewByType(library "ma" "layout" "maskLayout" "r")
mal_cvw = dbOpenCellViewByType(library "mal" "layout" "maskLayout" "r")
mp_cvw = dbOpenCellViewByType(library "mp" "layout" "maskLayout" "r")
mpl_cvw = dbOpenCellViewByType(library "mpl" "layout" "maskLayout" "r")
mplx_cvw = dbOpenCellViewByType(library "mplx" "layout" "maskLayout"
"r")
m2m1_cvw = dbOpenCellViewByType(library "M2M1" "layout" "maskLayout"
"r")
m2m1x_cvw = dbOpenCellViewByType(library "M2M1x" "layout" "maskLayout"
"r")
m3m2_cvw = dbOpenCellViewByType(library "M3M2" "layout" "maskLayout"
"r")

;---------------------------------------
; local parameters to define coordinates
;---------------------------------------
```

```
south = 0
west = 0

wn = 0.12
wp = 0.24
ods = 0.14
ods2 = 0.07
pex = 0.08
;bottom of PMOS active
prow = south + 0.44 + pex
;bottom of NMOS active
nrow1= south-0.22-R->Abx2-wn/2-0.04
nrow2= south-0.44-R->Abx-R->Abx2-wn-ods-0.04
mint = R->Abx + 2*R->GMacc
;distance of CO on PO to OD
cod3 = 0.1

;----------------------------------------
; PMOS
;----------------------------------------
;nwell contact
dbCreateInst(nxor_cvw mal_cvw nil list(west-3*(ods+mint)-mint
prow+wp/2) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-3*(ods+mint)-mint-
0.15:prow+wp/2 west-3*(ods+mint)-mint+0.15:prow+wp/2) R->Abx)
dbCreatePath(nxor_cvw list("NP" "drawing") list(west-3*(ods+mint)-mint-
0.26:prow+wp/2 west-3*(ods+mint)-mint+0.26:prow+wp/2) R->Abx+0.04)

;M10
dbCreateInst(nxor_cvw ma_cvw nil list(west-2*(ods+mint)-ods2-mint/2-R-
>GMacc prow+wp/2 ) "R0")
dbCreateInst(nxor_cvw ma_cvw nil list(west-2*(ods+mint)-ods2-mint/2+R-
>GMacc prow+wp/2 ) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-2*(ods+mint)-
ods2:prow+wp/2 west-2*(ods+mint)-ods2-mint:prow+wp/2) wp)
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-2*(ods+mint)-ods2-
mint/2:prow+wp+R->GAov west-2*(ods+mint)-ods2-mint/2:prow-R->GAov) R-
>Pwd)

;M8
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods-mint-ods2-mint/2-R-
>GMacc prow+wp/2 ) "R90")
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods-mint-ods2-mint/2+R-
>GMacc prow+wp/2 ) "R90")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-ods-mint-
ods2:prow+wp/2 west-ods-mint-ods2-mint:prow+wp/2) wp)
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-ods-mint-ods2-
mint/2:prow+wp+R->GAov west-ods-mint-ods2-mint/2:prow-R->GAov) R->Pwd)

;M3
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods2-mint/2-R->GMacc
prow+wp/2 ) "R90")
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods2-mint/2+R->GMacc
prow+wp/2 ) "R90")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-ods2:prow+wp/2
west-ods2-mint:prow+wp/2) wp)
```

```
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-ods2-
mint/2:prow+wp+R->GAov west-ods2-mint/2:prow-R->GAov) R->Pwd)

;M2
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods2+mint/2-R->GMacc
prow+wp/2 ) "R90")
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods2+mint/2+R->GMacc
prow+wp/2 ) "R90")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west+ods2:prow+wp/2
west+ods2+mint:prow+wp/2) wp)
dbCreatePath(nxor_cvw list("PO" "drawing")
list(west+ods2+mint/2:prow+wp+R->GAov west+ods2+mint/2:prow-R->GAov) R-
>Pwd)

;M9
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods+mint+ods2+mint/2-R-
>GMacc prow+wp/2 ) "R0")
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods+mint+ods2+mint/2+R-
>GMacc prow+wp/2 ) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing")
list(west+ods+mint+ods2:prow+wp/2 west+ods+mint+ods2+mint:prow+wp/2)
wp)
dbCreatePath(nxor_cvw list("PO" "drawing")
list(west+ods+mint+ods2+mint/2:prow+wp+R->GAov
west+ods+mint+ods2+mint/2:prow-R->GAov) R->Pwd)

;M11
dbCreateInst(nxor_cvw ma_cvw nil list(west+2*(ods+mint)+ods2+mint/2-R-
>GMacc prow+wp/2 ) "R0")
dbCreateInst(nxor_cvw ma_cvw nil list(west+2*(ods+mint)+ods2+mint/2+R-
>GMacc prow+wp/2 ) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing")
list(west+2*(ods+mint)+ods2:prow+wp/2
west+2*(ods+mint)+ods2+mint:prow+wp/2) wp)
dbCreatePath(nxor_cvw list("PO" "drawing")
list(west+2*(ods+mint)+ods2+mint/2:prow+wp+R->GAov
west+2*(ods+mint)+ods2+mint/2:prow-R->GAov) R->Pwd)

;---------------------------------------
; NMOS
;---------------------------------------
;substrate contact
dbCreateInst(nxor_cvw mal_cvw nil list(west-3*(ods+mint)-mint
nrow1+wn/2) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-3*(ods+mint)-mint-
0.15:nrow1+wn/2 west-3*(ods+mint)-mint+0.15:nrow1+wn/2) R->Abx)
dbCreatePath(nxor_cvw list("PP" "drawing") list(west-3*(ods+mint)-mint-
0.26:nrow1+wn/2 west-3*(ods+mint)-mint+0.26:nrow1+wn/2) R->Abx+0.04)

;M13
dbCreateInst(nxor_cvw ma_cvw nil list(west-2*(ods+mint)-ods2-mint/2-R-
>GMacc nrow1+wn/2 ) "R0")
dbCreateInst(nxor_cvw ma_cvw nil list(west-2*(ods+mint)-ods2-mint/2+R-
>GMacc nrow1+wn/2 ) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-2*(ods+mint)-
ods2:nrow1+wn/2 west-2*(ods+mint)-ods2-mint:nrow1+wn/2) wn)
```

```
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-2*(ods+mint)-ods2-
mint/2:nrow1+wn+R->GAov west-2*(ods+mint)-ods2-mint/2:nrow1-R->GAov) R-
>Pwd)

;M7
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods-mint-ods2-mint/2-R-
>GMacc nrow1+wn/2 ) "R90")
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods-mint-ods2-mint/2+R-
>GMacc nrow1+wn/2 ) "R90")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-ods-mint-
ods2:nrow1+wn/2 west-ods-mint-ods2-mint:nrow1+wn/2) wn)
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-ods-mint-ods2-
mint/2:nrow1+wn+R->GAov west-ods-mint-ods2-mint/2:nrow1-R->GAov) R-
>Pwd)

;M0
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods2-mint/2-R->GMacc
nrow1+wn/2 ) "R90")
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods2-mint/2+R->GMacc
nrow1+wn/2 ) "R90")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-ods2:nrow1+wn/2
west-ods2-mint:nrow1+wn/2) wn)
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-ods2-
mint/2:nrow1+wn+R->GAov west-ods2-mint/2:nrow1-R->GAov) R->Pwd)

;M1
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods2+mint/2-R->GMacc
nrow1+wn/2 ) "R90")
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods2+mint/2+R->GMacc
nrow1+wn/2 ) "R90")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west+ods2:nrow1+wn/2
west+ods2+mint:nrow1+wn/2) wn)
dbCreatePath(nxor_cvw list("PO" "drawing")
list(west+ods2+mint/2:nrow1+wn+R->GAov west+ods2+mint/2:nrow1-R->GAov)
R->Pwd)

;M6
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods+mint+ods2+mint/2-R-
>GMacc nrow1+wn/2 ) "R0")
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods+mint+ods2+mint/2+R-
>GMacc nrow1+wn/2 ) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing")
list(west+ods+mint+ods2:nrow1+wn/2 west+ods+mint+ods2+mint:nrow1+wn/2)
wn)
dbCreatePath(nxor_cvw list("PO" "drawing")
list(west+ods+mint+ods2+mint/2:nrow1+wn+R->GAov
west+ods+mint+ods2+mint/2:nrow1-R->GAov) R->Pwd)

;M12
dbCreateInst(nxor_cvw ma_cvw nil list(west+2*(ods+mint)+ods2+mint/2-R-
>GMacc nrow1+wn/2 ) "R0")
dbCreateInst(nxor_cvw ma_cvw nil list(west+2*(ods+mint)+ods2+mint/2+R-
>GMacc nrow1+wn/2 ) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing")
list(west+2*(ods+mint)+ods2:nrow1+wn/2
west+2*(ods+mint)+ods2+mint:nrow1+wn/2) wn)
```

```
dbCreatePath(nxor_cvw list("PO" "drawing")
list(west+2*(ods+mint)+ods2+mint/2:nrow1+wn+R->GAov
west+2*(ods+mint)+ods2+mint/2:nrow1-R->GAov) R->Pwd)

;M5
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods2-mint/2-R->GMacc
nrow2+wn/2 ) "R90")
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods2-mint/2+R->GMacc
nrow2+wn/2 ) "R90")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-ods2:nrow2+wn/2
west-ods2-mint:nrow2+wn/2) wn)
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-ods2-
mint/2:nrow2+wn+R->GAov west-ods2-mint/2:nrow2-R->GAov) R->Pwd)

;M4
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods2+mint/2-R->GMacc
nrow2+wn/2 ) "R90")
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods2+mint/2+R->GMacc
nrow2+wn/2 ) "R90")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west+ods2:nrow2+wn/2
west+ods2+mint:nrow2+wn/2) wn)
dbCreatePath(nxor_cvw list("PO" "drawing")
list(west+ods2+mint/2:nrow2+wn+R->GAov west+ods2+mint/2:nrow2-R->GAov)
R->Pwd)

;M14
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods+mint+ods2+mint/2-R-
>GMacc nrow2+wn/2 ) "R0")
dbCreateInst(nxor_cvw ma_cvw nil list(west+ods+mint+ods2+mint/2+R-
>GMacc nrow2+wn/2 ) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing")
list(west+ods+mint+ods2:nrow2+wn/2 west+ods+mint+ods2+mint:nrow2+wn/2)
wn)
dbCreatePath(nxor_cvw list("PO" "drawing")
list(west+ods+mint+ods2+mint/2:nrow2+wn+R->GAov
west+ods+mint+ods2+mint/2:nrow2+nrow1-2*R->GAov) R->Pwd)

;M15
dbCreateInst(nxor_cvw ma_cvw nil list(west+2*(ods+mint)+ods2+mint/2-R-
>GMacc nrow2+wn/2 ) "R0")
dbCreateInst(nxor_cvw ma_cvw nil list(west+2*(ods+mint)+ods2+mint/2+R-
>GMacc nrow2+wn/2 ) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing")
list(west+2*(ods+mint)+ods2:nrow2+wn/2
west+2*(ods+mint)+ods2+mint:nrow2+wn/2) wn)
dbCreatePath(nxor_cvw list("PO" "drawing")
list(west+2*(ods+mint)+ods2+mint/2:nrow2+wn+R->GAov
west+2*(ods+mint)+ods2+mint/2:nrow2+nrow1-2*R->GAov) R->Pwd)

;M16
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods-mint-ods2-mint/2-R-
>GMacc nrow2+wn/2 ) "R90")
dbCreateInst(nxor_cvw ma_cvw nil list(west-ods-mint-ods2-mint/2+R-
>GMacc nrow2+wn/2 ) "R90")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-ods-mint-
ods2:nrow2+wn/2 west-ods-mint-ods2-mint:nrow2+wn/2) wn)
```

```
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-ods-mint-ods2-
mint/2:nrow2+wn+R->GAov west-ods-mint-ods2-mint/2:nrow2+nrow1-2*R-
>GAov) R->Pwd)

;M17
dbCreateInst(nxor_cvw ma_cvw nil list(west-2*(ods+mint)-ods2-mint/2-R-
>GMacc nrow2+wn/2 ) "R0")
dbCreateInst(nxor_cvw ma_cvw nil list(west-2*(ods+mint)-ods2-mint/2+R-
>GMacc nrow2+wn/2 ) "R0")
dbCreatePath(nxor_cvw list("OD" "drawing") list(west-2*(ods+mint)-
ods2:nrow2+wn/2 west-2*(ods+mint)-ods2-mint:nrow2+wn/2) wn)
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-2*(ods+mint)-ods2-
mint/2:nrow2+wn+R->GAov west-2*(ods+mint)-ods2-mint/2:nrow2+nrow1-2*R-
>GAov) R->Pwd)

;---------------------------------------
; Mp contact
;---------------------------------------
;M8
dbCreateInst(nxor_cvw mpl_cvw nil list(west-ods-mint-ods2-mint/2
prow+pex+wp+cod3+R->Cbx2) "R90")
;M3
dbCreateInst(nxor_cvw mpl_cvw nil list(west-ods2-mint/2 prow-pex-cod3-
R->Cbx2) "R90")
;M2
dbCreateInst(nxor_cvw mpl_cvw nil list(west+ods2+mint/2 prow-pex-cod3-
R->Cbx2) "R90")
;M9
dbCreateInst(nxor_cvw mpl_cvw nil list(west+ods+mint+ods2+mint/2
prow+pex+wp+cod3+R->Cbx2) "R90")
;M7
dbCreateInst(nxor_cvw mpl_cvw nil list(west-ods-mint-ods2-mint/2
nrow1+R->Abx+R->GAov+0.03) "R0")
;M0
dbCreateInst(nxor_cvw mpl_cvw nil list(west-ods2-mint/2+0.06 nrow1+R-
>Abx+R->GAov) "R90")
;M1
dbCreateInst(nxor_cvw mpl_cvw nil list(west+ods2+mint/2-0.06 nrow1+R-
>Abx+R->GAov) "R90")
;M6
dbCreateInst(nxor_cvw mpl_cvw nil list(west+ods+mint+ods2+mint/2
nrow1+R->Abx+R->GAov+0.03) "R0")
;M10-13
dbCreateInst(nxor_cvw mp_cvw nil list(west-2*(ods+mint)-ods2-mint/2+R-
>Pwd2 (nrow1+wn+prow)/2) "R0")
;M11-12
dbCreateInst(nxor_cvw mp_cvw nil list(west+2*(ods+mint)+ods2+mint/2-R-
>Pwd2 (nrow1+wn+prow)/2) "R0")
;M14
dbCreateInst(nxor_cvw mp_cvw nil list(west+ods+mint+ods2+mint/2
nrow2+nrow1-2*R->GAov) "R0")
;M15
dbCreateInst(nxor_cvw mpl_cvw nil list(west+2*(ods+mint)+ods2+mint/2
nrow2+nrow1-2*R->GAov) "R0")
;M16
dbCreateInst(nxor_cvw mpl_cvw nil list(west-ods-mint-ods2-mint/2
nrow2+nrow1-2*R->GAov) "R0")
```

```
;M17
dbCreateInst(nxor_cvw mp_cvw nil list(west-2*(ods+mint)-ods2-mint/2
nrow2+nrow1-2*R->GAov) "R0")
;pc
dbCreatePath(nxor_cvw list("M1" "drawing") list(west:nrow1+R->Abx+R-
>GAov+0.03 west:prow+wp/2) R->Mwd)

;---------------------------------------
; Poly (PO)
;---------------------------------------
;10-13
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-2*(ods+mint)-ods2-
mint/2:nrow1+wn+R->GAov west-2*(ods+mint)-ods2-mint/2:prow-R->GAov) R-
>Pwd)
;11-12
dbCreatePath(nxor_cvw list("PO" "drawing")
list(west+2*(ods+mint)+ods2+mint/2:nrow1+wn+R->GAov
west+2*(ods+mint)+ods2+mint/2:prow-R->GAov) R->Pwd)
;0-5
dbCreatePath(nxor_cvw list("PO" "drawing") list(west-ods2-mint/2:nrow1-
R->GAov west-ods2-mint/2:nrow2+wn+R->GAov) R->Pwd)
;1-4
dbCreatePath(nxor_cvw list("PO" "drawing") list(west+ods2+mint/2:nrow1-
R->GAov west+ods2+mint/2:nrow2+wn+R->GAov) R->Pwd)

;---------------------------------------
; M1
;---------------------------------------
;3-8
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods2-mint/2-R-
>GMacc:prow+wp/2 west-ods-mint-ods2-mint/2+R->GMacc:prow+wp/2) R->Mwd)
;3-2
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods2-mint/2+R-
>GMacc:prow+wp/2 west+ods2+mint/2-R->GMacc:prow+wp/2) R->Mbx)
;2-9
dbCreatePath(nxor_cvw list("M1" "drawing") list(west+ods2+mint/2+R-
>GMacc:prow+wp/2 west+ods+mint+ods2+mint/2-R->GMacc:prow+wp/2) R->Mwd)
;10-13
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-2*(ods+mint)-ods2-
mint/2-R->GMacc:prow+wp/2 west-2*(ods+mint)-ods2-mint/2-R-
>GMacc:nrow1+wn/2) R->Mwd)
;11-12
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+2*(ods+mint)+ods2+mint/2+R->GMacc:prow+wp/2
west+2*(ods+mint)+ods2+mint/2+R->GMacc:nrow1+wn/2) R->Mwd)
;8-7
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods-mint-ods2-
mint/2-R->GMacc:nrow1+wn/2 west-ods-mint-ods2-mint/2-R-
>GMacc:prow+wp/2) R->Mwd)
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods-mint-ods2-
mint/2+R->GMacc:nrow1+wn/2 west-ods-mint-ods2-mint/2+R-
>GMacc:prow+wp/2) R->Mwd)
;9-6
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+ods+mint+ods2+mint/2-R->GMacc:nrow1+wn/2
west+ods+mint+ods2+mint/2-R->GMacc:prow+wp/2) R->Mwd)
```

```
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+ods+mint+ods2+mint/2+R->GMacc:nrow1+wn/2
west+ods+mint+ods2+mint/2+R->GMacc:prow+wp/2) R->Mwd)
;3-0
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods2-mint/2:prow-
pex-cod3-R->Cbx2 west-ods2-mint/2-R->GMacc:prow-pex-cod3-R->Cbx2 west-
ods2-mint/2-R->GMacc:nrow1+wn/2) R->Mwd)
;2-1
dbCreatePath(nxor_cvw list("M1" "drawing") list(west+ods2+mint/2:prow-
pex-cod3-R->Cbx2 west+ods2+mint/2+R->GMacc:prow-pex-cod3-R->Cbx2
west+ods2+mint/2+R->GMacc:nrow1+wn/2) R->Mwd)
;8-9
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods-mint-ods2-
mint/2:prow+pex+wp+cod3+R->Cbx2
west+ods+mint+ods2+mint/2:prow+pex+wp+cod3+R->Cbx2) R->Mwd)
;8-10
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods-mint-ods2-
mint/2-R->GMacc:(nrow1+wn+prow)/2 west-2*(ods+mint)-ods2-mint/2+R-
>Pwd2:(nrow1+wn+prow)/2) R->Mwd)
;9-11
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+ods+mint+ods2+mint/2+R->GMacc:(nrow1+wn+prow)/2
west+2*(ods+mint)+ods2+mint/2-R->Pwd2:(nrow1+wn+prow)/2) R->Mwd)
;0-4-6
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods2-mint/2+R-
>GMacc:nrow1+wn/2 west-ods2-mint/2+R->GMacc:(nrow1+wn+nrow2)/2-0.015
west+ods2+mint/2-R->GMacc:(nrow1+wn+nrow2)/2-0.015 west+ods2+mint/2-R-
>GMacc:nrow2+wn/2) R->Mwd)
dbCreatePath(nxor_cvw list("M1" "drawing") list(west+ods2+mint/2-R-
>GMacc:(nrow1+wn+nrow2)/2-0.015 west+ods+mint+ods2+mint/2-R-
>GMacc:(nrow1+wn+nrow2)/2-0.015 west+ods+mint+ods2+mint/2-R-
>GMacc:nrow1+wn/2) R->Mwd)
;16-17
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods-mint-ods2-
mint/2+R->GMacc:nrow2+wn/2 west-ods-mint-ods2-mint/2+R-
>GMacc:(nrow1+wn+nrow2)/2-0.015 west-2*(ods+mint)-ods2-mint/2+R-
>GMacc:(nrow1+wn+nrow2)/2-0.015 west-2*(ods+mint)-ods2-mint/2+R-
>GMacc:nrow2+wn/2) R->Mwd)
;5-16
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods2-mint/2-R-
>GMacc:nrow2+wn/2 west-ods-mint-ods2-mint/2+R->GMacc:nrow2+wn/2 ) R-
>Mwd)
;4-14
dbCreatePath(nxor_cvw list("M1" "drawing") list(west+ods2+mint/2+R-
>GMacc:nrow2+wn/2 west+ods+mint+ods2+mint/2-R->GMacc:nrow2+wn/2 ) R-
>Mwd)
;14-15
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+ods+mint+ods2+mint/2-R->GMacc:nrow2+wn/2
west+ods+mint+ods2+mint/2-R->GMacc:nrow2-3*wn
west+2*(ods+mint)+ods2+mint/2-R->GMacc:nrow2-3*wn
west+2*(ods+mint)+ods2+mint/2-R->GMacc:nrow2+wn/2) R->Mwd)
;16-14
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+ods+mint+ods2+mint/2:nrow2+nrow1-2*R->GAov west-ods-mint-
ods2-mint/2:nrow2+nrow1-2*R->GAov) R->Mwd)
;15
```

```
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+2*(ods+mint)+ods2+mint/2+R->GMacc:nrow2+wn/2
west+2*(ods+mint)+ods2+mint/2+R->GMacc:nrow2-3*wn ) R->Mwd)
;16
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods-mint-ods2-
mint/2-R->GMacc:nrow2+wn/2 west-ods-mint-ods2-mint/2-R->GMacc:nrow2-
3*wn ) R->Mwd)

;16poly
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods-mint-ods2-
mint/2:nrow2+nrow1-2*R->GAov west-ods-mint-ods2-mint/2:nrow2+2*nrow1-
2*R->GAov ) R->Mwd)

;DRC M1 area requirements
;nwell
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-3*(ods+mint)-
mint:prow+wp/2 west-3*(ods+mint)-mint:prow+wp/2-0.42) R->M2wd)
;sub
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-3*(ods+mint)-
mint:nrow1+wn/2 west-3*(ods+mint)-mint:nrow1+wn/2-0.42) R->M2wd)
;10i
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-2*(ods+mint)-ods2-
mint/2+R->GMacc:prow+wp/2 west-2*(ods+mint)-ods2-mint/2+R-
>GMacc:prow+wp/2+0.42) R->M2wd)
;13i
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-2*(ods+mint)-ods2-
mint/2+R->GMacc:nrow1+wn/2 west-2*(ods+mint)-ods2-mint/2+R->GMacc-
0.265:nrow1+wn/2) R->Mbx)
;7poly
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods-mint-ods2-
mint/2:nrow1+wn+R->GAov west-ods-mint-ods2-mint/2:nrow1+wn+R-
>GAov+0.42) R->M2wd)
;0poly
dbCreateRect(nxor_cvw list("M1" "drawing") list(west-ods2-mint/2+0.06-
0.13:nrow1+R->Abx+R->GAov-0.08 west-ods2-mint/2+0.06+0.17:nrow1+R-
>Abx+R->GAov+0.12) )
;1poly
dbCreateRect(nxor_cvw list("M1" "drawing") list(west+ods2+mint/2-
0.06+0.13:nrow1+R->Abx+R->GAov-0.08 west+ods2+mint/2-0.06-0.17:nrow1+R-
>Abx+R->GAov+0.12))
;1i
dbCreatePath(nxor_cvw list("M1" "drawing") list(west+ods2+mint/2-R-
>GMacc:nrow1+wn/2 west+ods2+mint/2-R->GMacc+0.265:nrow1+wn/2) R->Mbx)
;6p
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+ods+mint+ods2+mint/2:nrow1+wn+R->GAov
west+ods+mint+ods2+mint/2:nrow1+wn+R->GAov+0.42) R->M2wd)
;11i
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+2*(ods+mint)+ods2+mint/2-R->GMacc:prow+wp/2
west+2*(ods+mint)+ods2+mint/2-R->GMacc:prow+wp/2+0.42) R->M2wd)
;12i
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+2*(ods+mint)+ods2+mint/2-R->GMacc:nrow1+wn/2
west+2*(ods+mint)+ods2+mint/2-R->GMacc:nrow1+wn/2-0.265) R->Mbx)
;5i
```

```
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-ods2-mint/2+R-
>GMacc:nrow2+wn/2 west-ods2-mint/2+R->GMacc:nrow2+wn/2-0.42) R->M2wd)
;14o
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+ods+mint+ods2+mint/2+R->GMacc:nrow2+wn/2
west+ods+mint+ods2+mint/2+R->GMacc:nrow2+wn/2+0.265) R->Mbx)
;17o
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-2*(ods+mint)-ods2-
mint/2-R->GMacc:nrow2+wn/2 west-2*(ods+mint)-ods2-mint/2-R-
>GMacc:nrow2+wn/2-0.5) R->M2wd)
;17p
dbCreatePath(nxor_cvw list("M1" "drawing") list(west-2*(ods+mint)-ods2-
mint/2:nrow2+nrow1-2*R->GAov west-2*(ods+mint)-ods2-mint/2:nrow2+nrow1-
2*R->GAov-0.5) R->M2wd)
;15p
dbCreatePath(nxor_cvw list("M1" "drawing")
list(west+2*(ods+mint)+ods2+mint/2:nrow2+nrow1-2*R->GAov
west+2*(ods+mint)+ods2+mint/2:nrow2+nrow1-2*R->GAov-0.5) R->M2wd)

;---------------------------------------
; M1M2
;---------------------------------------
;7poly
dbCreateInst(nxor_cvw m2m1x_cvw nil list(west-ods-mint-ods2-mint/2
nrow1+R->Abx+R->GAov+0.03) "R90")
;7
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-ods-mint-ods2-mint/2+R-
>GMacc nrow1+wn/2-0.03) "R0")
;0
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-ods2-mint/2+0.06 nrow1+R-
>Abx+R->GAov+0.03) "R90")
;1
dbCreateInst(nxor_cvw m2m1_cvw nil list(west+ods2+mint/2-0.06 nrow1+R-
>Abx+R->GAov+0.03) "R90")
;6
dbCreateInst(nxor_cvw m2m1x_cvw nil list(west+ods+mint+ods2+mint/2
nrow1+R->Abx+R->GAov+0.03) "R90")
;5
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-ods2-mint/2+R->GMacc
nrow2+wn/2) "R0")
;1
dbCreateInst(nxor_cvw m2m1_cvw nil list(west+ods2+mint/2-R->GMacc
nrow1+wn/2-0.03) "R0")
;pc
dbCreateInst(nxor_cvw m2m1x_cvw nil list(west nrow1+R->Abx+R-
>GAov+0.03) "R90")
;pcb
dbCreateInst(nxor_cvw m2m1x_cvw nil list(west+ods2+mint/2+R->GMacc
prow+wp+pex+cod3+R->Cbx2 ) "R0")
;out
dbCreateInst(nxor_cvw m2m1_cvw nil
list(west+2*(ods+mint)+ods2+mint/2+R->GMacc (nrow1+wn+prow)/2) "R0")
;outb
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-2*(ods+mint)-ods2-mint/2-
R->GMacc (nrow1+wn+prow)/2) "R0")
;16poly
```

```
dbCreateInst(nxor_cvw m2m1x_cvw nil list(west-ods-mint-ods2-mint/2
nrow2+2*nrow1-2*R->GAov ) "R90")
;15
dbCreateInst(nxor_cvw m2m1x_cvw nil
list(west+2*(ods+mint)+ods2+mint/2+R->GMacc nrow2-3*wn  ) "R90")
;16pin
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-ods-mint-ods2-mint/2-R-
>GMacc nrow2-3*wn) "R0")
;15poly
dbCreateInst(nxor_cvw m2m1x_cvw nil list(west+2*(ods+mint)+ods2+mint/2
nrow2+nrow1-2*R->GAov ) "R90")
;M17
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-2*(ods+mint)-ods2-mint/2
nrow2+nrow1-2*R->GAov) "R0")
;M17pin
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-2*(ods+mint)-ods2-mint/2-
R->GMacc nrow2+wn/2) "R0")
;M14pin
dbCreateInst(nxor_cvw m2m1_cvw nil list(west+ods+mint+ods2+mint/2+R-
>GMacc nrow2+wn/2) "R0")
;10
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-2*(ods+mint)-ods2-
mint/2+R->GMacc prow+wp/2)  "R0")
;11
dbCreateInst(nxor_cvw m2m1_cvw nil list(west+2*(ods+mint)+ods2+mint/2-
R->GMacc prow+wp/2) "R0")
;nwell
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-3*(ods+mint)-mint
prow+wp/2) "R0")
;13
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-2*(ods+mint)-ods2-
mint/2+R->GMacc nrow1+wn/2) "R0")
;12
dbCreateInst(nxor_cvw m2m1_cvw nil list(west+2*(ods+mint)+ods2+mint/2-
R->GMacc nrow1+wn/2 ) "R0")
;sub
dbCreateInst(nxor_cvw m2m1_cvw nil list(west-3*(ods+mint)-mint
nrow1+wn/2) "R0")

;---------------------------------------
; M2
;---------------------------------------
;7-0-1-6
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-ods-mint-ods2-
mint/2:nrow1+R->Abx+R->GAov+0.03 west+ods+mint+ods2+mint/2:nrow1+R-
>Abx+R->GAov+0.03) R->M2wd)
;7-5-1
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-ods-mint-ods2-
mint/2+R->GMacc:nrow1+wn/2-0.03 west-ods-mint-ods2-mint/2+R-
>GMacc:(nrow1+wn+nrow2)/2  west-ods2-mint/2+R->GMacc:(nrow1+wn+nrow2)/2
west-ods2-mint/2+R->GMacc:nrow2+wn/2) R->M2wd)
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-ods2-mint/2+R-
>GMacc:(nrow1+wn+nrow2)/2 west+ods2+mint/2-R->GMacc:(nrow1+wn+nrow2)/2
west+ods2+mint/2-R->GMacc:nrow1+wn/2) R->M2wd)
;16-15
```

```
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-ods-mint-ods2-
mint/2-R->GMacc:nrow2-3*wn west+2*(ods+mint)+ods2+mint/2+R-
>GMacc:nrow2-3*wn) R->M2wd)
;17-15
;dbCreatePath(nxor_cvw list("M2" "drawing") list(west-2*(ods+mint)-
ods2-mint/2:nrow2+nrow1-2*R->GAov west+2*(ods+mint)+ods2+mint/2:nrow2-
3*wn) R->M2wd)

;DRC M2 area requirements
;nwell
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-3*(ods+mint)-
mint:prow+wp/2 west-3*(ods+mint)-mint:prow+wp/2-0.5) R->M2wd)
;sub
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-3*(ods+mint)-
mint:nrow1+wn/2 west-3*(ods+mint)-mint:nrow1+wn/2-0.5) R->M2wd)
;10i
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-2*(ods+mint)-ods2-
mint/2+R->GMacc:prow+wp/2 west-2*(ods+mint)-ods2-mint/2+R-
>GMacc:prow+wp/2+0.5) R->M2wd)
;13i
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-2*(ods+mint)-ods2-
mint/2+R->GMacc:nrow1+wn/2 west-2*(ods+mint)-ods2-mint/2+R-
>GMacc:nrow1+wn/2-0.5) R->M2wd)
;11i
dbCreatePath(nxor_cvw list("M2" "drawing")
list(west+2*(ods+mint)+ods2+mint/2-R->GMacc:prow+wp/2
west+2*(ods+mint)+ods2+mint/2-R->GMacc:prow+wp/2+0.5) R->M2wd)
;12i
dbCreatePath(nxor_cvw list("M2" "drawing")
list(west+2*(ods+mint)+ods2+mint/2-R->GMacc:nrow1+wn/2
west+2*(ods+mint)+ods2+mint/2-R->GMacc:nrow1+wn/2-0.5) R->M2wd)
;11-12
dbCreatePath(nxor_cvw list("M2" "drawing")
list(west+2*(ods+mint)+ods2+mint/2+R->GMacc:(nrow1+wn+prow)/2
west+2*(ods+mint)+ods2+mint/2+R->GMacc:(nrow1+wn+prow)/2-0.5) R->M2wd)
;10-13
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-2*(ods+mint)-ods2-
mint/2-R->GMacc:(nrow1+wn+prow)/2 west-2*(ods+mint)-ods2-mint/2-R-
>GMacc:(nrow1+wn+prow)/2-0.5) R->M2wd)
;pcb
dbCreatePath(nxor_cvw list("M2" "drawing") list(west+ods2+mint/2+R-
>GMacc:prow+wp+pex+cod3+R->Cbx2 west+ods2+mint/2+R-
>GMacc:prow+wp+pex+cod3+R->Cbx2-0.5) R->M2wd)
;14o
dbCreatePath(nxor_cvw list("M2" "drawing")
list(west+ods+mint+ods2+mint/2+R->GMacc:nrow2+wn/2
west+ods+mint+ods2+mint/2+R->GMacc:nrow2+wn/2+0.5) R->M2wd)
;17o
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-2*(ods+mint)-ods2-
mint/2-R->GMacc:nrow2+wn/2 west-2*(ods+mint)-ods2-mint/2-R-
>GMacc:nrow2+wn/2-0.5) R->M2wd)
;17p
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-2*(ods+mint)-ods2-
mint/2:nrow2+nrow1-2*R->GAov west-2*(ods+mint)-ods2-mint/2:nrow2+nrow1-
2*R->GAov-0.5) R->M2wd)
;15p
```

```
dbCreatePath(nxor_cvw list("M2" "drawing")
list(west+2*(ods+mint)+ods2+mint/2:nrow2+nrow1-2*R->GAov
west+2*(ods+mint)+ods2+mint/2:nrow2+nrow1-2*R->GAov-0.5) R->M2wd)
;B
dbCreatePath(nxor_cvw list("M2" "drawing") list(west-ods-mint-ods2-
mint/2:nrow2+2*nrow1-2*R->GAov west-ods-mint-ods2-mint/2:nrow2+2*nrow1-
2*R->GAov+0.5) R->M2wd)

;---------------------------------------
; M2M3
;---------------------------------------
;9
dbCreateInst(nxor_cvw m3m2_cvw nil list(west-2*(ods+mint)-ods2-
mint/2+R->GMacc prow+wp/2)  "R0")
;8
dbCreateInst(nxor_cvw m3m2_cvw nil list(west+2*(ods+mint)+ods2+mint/2-
R->GMacc prow+wp/2) "R0")
;nwell
dbCreateInst(nxor_cvw m3m2_cvw nil list(west-3*(ods+mint)-mint
prow+wp/2) "R0")
;11
dbCreateInst(nxor_cvw m3m2_cvw nil list(west-2*(ods+mint)-ods2-
mint/2+R->GMacc nrow1+wn/2) "R0")
;10
dbCreateInst(nxor_cvw m3m2_cvw nil list(west+2*(ods+mint)+ods2+mint/2-
R->GMacc nrow1+wn/2 ) "R0")
;sub
dbCreateInst(nxor_cvw m3m2_cvw nil list(west-3*(ods+mint)-mint
nrow1+wn/2) "R0")
;15poly
dbCreateInst(nxor_cvw m3m2_cvw nil list(west+2*(ods+mint)+ods2+mint/2
nrow2+nrow1-2*R->GAov ) "R0")
;pc
dbCreateInst(nxor_cvw m3m2_cvw nil list(west+ods+mint+ods2+mint/2
nrow1+R->Abx+R->GAov+0.03 ) "R0")
;pcb
dbCreateInst(nxor_cvw m3m2_cvw nil list(west+ods2+mint/2+R->GMacc
prow+wp+pex+cod3+R->Cbx2 ) "R0")
;out
dbCreateInst(nxor_cvw m3m2_cvw nil
list(west+2*(ods+mint)+ods2+mint/2+R->GMacc (nrow1+wn+prow)/2) "R0")
;outb
dbCreateInst(nxor_cvw m3m2_cvw nil list(west-2*(ods+mint)-ods2-mint/2-
R->GMacc (nrow1+wn+prow)/2) "R0")
;M17pin
dbCreateInst(nxor_cvw m3m2_cvw nil list(west-2*(ods+mint)-ods2-mint/2-
R->GMacc nrow2+wn/2) "R0")
;M14pin
dbCreateInst(nxor_cvw m3m2_cvw nil list(west+ods+mint+ods2+mint/2+R-
>GMacc nrow2+wn/2) "R0")
;B
dbCreateInst(nxor_cvw m3m2_cvw nil list(west-ods-mint-ods2-mint/2
nrow2+2*nrow1-2*R->GAov ) "R0")
;Bb
dbCreateInst(nxor_cvw m3m2_cvw nil list(west-2*(ods+mint)-ods2-mint/2
nrow2+nrow1-2*R->GAov ) "R0")
;Ab
```

```
dbCreateInst(nxor_cvw m3m2_cvw nil list(west-ods-mint-ods2-mint/2-R-
>GMacc nrow2-3*wn ) "R0")

;---------------------------------------
; M3
;---------------------------------------
;A
dbCreatePath(nxor_cvw list("M3" "drawing") list(west-2*(ods+mint)-ods2-
mint/2-R->GMacc:nrow2+wn/2 west+ods+mint+ods2+mint/2+R-
>GMacc:nrow2+wn/2) R->M2wd)
;vdd
dbCreatePath(nxor_cvw list("M3" "drawing") list(west-3*(ods+mint)-
mint:prow+wp/2 west+2*(ods+mint)+ods2+mint/2-R->GMacc:prow+wp/2) R-
>M2wd)
;gnd
dbCreatePath(nxor_cvw list("M3" "drawing") list(west-3*(ods+mint)-
mint:nrow1+wn/2 west+2*(ods+mint)+ods2+mint/2-R->GMacc:nrow1+wn/2) R-
>M2wd)
;Bb
dbCreatePath(nxor_cvw list("M3" "drawing") list(west-2*(ods+mint)-ods2-
mint/2:nrow2+nrow1-2*R->GAov west+2*(ods+mint)+ods2+mint/2:nrow2+nrow1-
2*R->GAov) R->M2wd)

;DRC M3 area requirements

;out
dbCreatePath(nxor_cvw list("M3" "drawing")
list(west+2*(ods+mint)+ods2+mint/2+R->GMacc:(nrow1+wn+prow)/2
west+2*(ods+mint)+ods2+mint/2+R->GMacc-0.5:(nrow1+wn+prow)/2) R->M2wd)
;outb
dbCreatePath(nxor_cvw list("M3" "drawing") list(west-2*(ods+mint)-ods2-
mint/2-R->GMacc:(nrow1+wn+prow)/2 west-2*(ods+mint)-ods2-mint/2-R-
>GMacc+0.5:(nrow1+wn+prow)/2) R->M2wd)
;pc
dbCreatePath(nxor_cvw list("M3" "drawing")
list(west+ods+mint+ods2+mint/2:nrow1+R->Abx+R->GAov+0.03
west+ods+mint+ods2+mint/2-0.5:nrow1+R->Abx+R->GAov+0.03) R->M2wd)
;pcb
dbCreatePath(nxor_cvw list("M3" "drawing") list(west+ods2+mint/2+R-
>GMacc:prow+wp+pex+cod3+R->Cbx2 west+ods2+mint/2+R->GMacc-
0.5:prow+wp+pex+cod3+R->Cbx2) R->M2wd)
;16 Ab
dbCreatePath(nxor_cvw list("M3" "drawing") list(west-ods-mint-ods2-
mint/2-R->GMacc:nrow2-3*wn west-ods-mint-ods2-mint/2-R-
>GMacc+0.5:nrow2-3*wn) R->M2wd)
;B
dbCreatePath(nxor_cvw list("M3" "drawing") list(west-ods-mint-ods2-
mint/2:nrow2+2*nrow1-2*R->GAov west-ods-mint-ods2-
mint/2+0.5:nrow2+2*nrow1-2*R->GAov) R->M2wd)

;---------------------------------------
; NPlus (NP)
;---------------------------------------
;substrate
 dbCreatePolygon(nxor_cvw list("NP" "drawing")
                 list(west-3*(ods+mint)-ods2-mint-2*R->WAov:nrow2-R-
>WAov-0.14-0.69
```

```
                        west-2*(ods+mint)-ods2-mint-R->WAov:nrow2-R-
>WAov-0.14-0.69
                        west-2*(ods+mint)-ods2-mint-R->WAov:south
                        west-3*(ods+mint)-mint+0.26:south
                        west-3*(ods+mint)-mint+0.26:nrow1+wn/2-0.12
                        west-3*(ods+mint)-mint-0.26:nrow1+wn/2-0.12
                        west-3*(ods+mint)-mint-0.26:nrow1+wn/2+0.12
                        west-3*(ods+mint)-mint+0.26:nrow1+wn/2+0.12
                        west-3*(ods+mint)-mint+0.26:south
                        west-3*(ods+mint)-ods2-mint-2*R->WAov:south
                        west-3*(ods+mint)-ods2-mint-2*R->WAov:nrow2-R-
>WAov-0.14-0.69))

;nmos
dbCreateRect(nxor_cvw list("NP" "drawing") list(west-2*(ods+mint)-ods2-
mint-R->WAov:nrow2-R->WAov-0.14-0.69 west+2*(ods+mint)+ods2+mint+R-
>WAov:south))

;---------------------------------------
; PPlus (PP)
;---------------------------------------
;nwell
 dbCreatePolygon(nxor_cvw list("PP" "drawing")
                 list(west-3*(ods+mint)-ods2-mint-2*R->WAov:south
                        west-2*(ods+mint)-ods2-mint-R->WAov:south
                        west-2*(ods+mint)-ods2-mint-R-
>WAov:prow+2*pex+wp+R->WAov+0.14
                        west-3*(ods+mint)-mint+0.26:prow+2*pex+wp+R-
>WAov+0.14
                        west-3*(ods+mint)-mint+0.26:prow+wp/2-0.12
                        west-3*(ods+mint)-mint-0.26:prow+wp/2-0.12
                        west-3*(ods+mint)-mint-0.26:prow+wp/2+0.12
                        west-3*(ods+mint)-mint+0.26:prow+wp/2+0.12
                        west-3*(ods+mint)-mint+0.26:prow+wp+2*pex+R-
>WAov+0.14
                        west-3*(ods+mint)-ods2-mint-2*R-
>WAov:prow+wp+2*pex+R->WAov+0.14
                        west-3*(ods+mint)-ods2-mint-2*R->WAov:south))

;pmos
dbCreateRect(nxor_cvw list("PP" "drawing") list(west-2*(ods+mint)-ods2-
mint-R->WAov:south west+2*(ods+mint)+ods2+mint+R->WAov:prow+wp+2*pex+R-
>WAov+0.14))

;---------------------------------------
; N-Well (NW)
;---------------------------------------
dbCreateRect(nxor_cvw list("NW" "drawing") list(west-3*(ods+mint)-ods2-
mint-3*R->WAov:south west+2*(ods+mint)+ods2+mint+2*R-
>WAov:prow+wp+2*pex+R->WAov+0.14))

;---------------------------------------
; Create Pin
;---------------------------------------
dbCreateLabel( nxor_cvw list( "M3" "pin" ) list(west-ods-mint-ods2-
mint/2 nrow2+2*nrow1-2*R->GAov  ) "B" "centerCenter" "R0" "stick" R-
>Cbx )
```

```
dbCreateLabel( nxor_cvw list( "M3" "pin" ) list( west-2*(ods+mint)-
ods2-mint/2 nrow2+nrow1-2*R->GAov ) "Bb" "centerCenter" "R0" "stick" R-
>Cbx )

dbCreateLabel( nxor_cvw list( "M3" "pin" ) list(west-2*(ods+mint)-ods2-
mint/2-R->GMacc nrow2+wn/2) "A" "centerCenter" "R0" "stick" R->Cbx )
dbCreateLabel( nxor_cvw list( "M3" "pin" ) list(west-ods-mint-ods2-
mint/2-R->GMacc nrow2-3*wn ) "Ab" "centerCenter" "R0" "stick" R->Cbx )

dbCreateLabel( nxor_cvw list( "M3" "pin" )
list(west+2*(ods+mint)+ods2+mint/2+R->GMacc (nrow1+wn+prow)/2) "out"
"centerCenter" "R0" "stick" R->Cbx )
dbCreateLabel( nxor_cvw list( "M3" "pin" ) list(west-2*(ods+mint)-ods2-
mint/2-R->GMacc (nrow1+wn+prow)/2) "outb" "centerCenter" "R0" "stick"
R->Cbx )

dbCreateLabel( nxor_cvw list( "M3" "pin" ) list(west-3*(ods+mint)-mint
prow+wp/2 ) "vdd!" "centerCenter" "R0" "stick" R->Cbx )
dbCreateLabel( nxor_cvw list( "M3" "pin" ) list(west-3*(ods+mint)-mint
nrow1+wn/2) "gnd!" "centerCenter" "R0" "stick" R->Cbx )

dbCreateLabel( nxor_cvw list( "M3" "pin" )
list(west+ods+mint+ods2+mint/2 nrow1+R->Abx+R->GAov+0.03  ) "PC!"
"centerCenter" "R0" "stick" R->Cbx )
dbCreateLabel( nxor_cvw list( "M3" "pin" ) list(west+ods2+mint/2+R-
>GMacc prow+wp+pex+cod3+R->Cbx2 ) "PCB!" "centerCenter" "R0" "stick" R-
>Cbx )

 ;-------------------------------------------------------------------
 ; prBoundary
 ;-------------------------------------------------------------------
; x1 = west-3*(ods+mint)-ods2-mint-2*R->WAov
; x2 = west+2*(ods+mint)+ods2+mint+R->WAov
; y1 = nrow2-R->WAov-0.1
; y2 = prow+wp+R->WAov+pex
; d_rect = dbCreateRect( nxor_cvw list("prBoundary" "drawing")
;                        list(x1:y1 x2:y2) )
; rod_obj = rodNameShape( ?name "rod_nxor"
;                         ?shapeId d_rect )

 ;---------------------------------------
 ; Close all db objects
 ;---------------------------------------
 dbClose(ma_cvw)
 dbClose(mal_cvw)
 dbClose(mp_cvw)
 dbClose(mpl_cvw)
 dbClose(mplx_cvw)
 dbClose(m2m1_cvw)
 dbClose(m2m1x_cvw)
 dbClose(m3m2_cvw)

 dbSave(nxor_cvw)
 ;dbClose(nxor_cvw)
) ;let
) ;procedure
```

# REFERENCE LIST


[1] ITRS, "The International Technology Roadmap for Semiconductors," http://www.itrs.net/Links/2006Update/2006UpdateFinal.htm, Accessed on March 16, 2007.

[2] ITRS, "International Technology Roadmap for Semiconductors (ITRS) 2004 Update - Overall Roadmap Technology Characteristics," http://www.itrs.net/Common/2004Update/2004_000_ORTC.pdf, Accessed on March 16, 2007.

[3] ITRS, "The International Technology Roadmap for Semiconductors 2005 Edition – Executive Summary," http://www.itrs.net/Links/2005ITRS/ExecSum2005.pdf, Accessed on March 16, 2007.

[4] G.E. Moore, "Progress in Digital Integrated Electronics," International Electron Devices Meeting, Vol. 21, pp. 11-13, 1975.

[5] J. Rabaey, A. Chandrakasan, B. Nikolic, "Digital Integrated Circuits," Pearson Education International, New Jersey, 2003.

[6] K. Shimohigashi and K. Seki, "Low-Voltage ULSI Design," IEEE Journal of Solid State Circuits., Vol. 28 No.4, pp. 408-413, 1993.

[7] J.B. Kuo, J. Lou, "Low-Voltage CMOS VLSI Circuits," Wiley, New York, 1999.

[8] H.I. Chen, E.K. Loo, J.B. Kuo and M. Syrzycki, "Triple-Threshold Static Power Minimization Technique in High-Level Synthesis for Designing High-Speed Low-Power SOC Applications Using 90nm MTCMOS Technology," Canadian Conference on Electrical and Computer Engineering, Vancouver, BC

[9] J.B. Kuo, "CMOS Digital IC," McGraw-Hill, 1996.

[10] G. Carrington, "Basic Thermodynamics," Oxford Science, 1994.

[11] J.S. Denker, "A review of adiabatic computing," Proc. of the 1994 Symposium of Low Power Electronics/Digest of Technical Papers, pp. 94-97, October 1994.

[12] W.C. Athas, L.J. Svensson, J.G. Koller, N. Tzartzanis, and Y. Chou, "Low-power digital systems base on adiabatic-switching principles," IEEE Trans. on VLSI Systems, Vol. 2, No. 4, pp. 398-406, December 1994.

[13] S. Younis and T. Knight, "Asymptotically zero energy split-level charge recovery logic," Technical Report AITR-1500, MIT AI Laboratory, June 1994.

[14] S. Younis and T. Knight, "Non-dissipative rail drivers for adiabatic circuits," Proceedings of 16th Conference on Advanced Research in VLSI, Mar. 1995, pp.404-414.

[15] D. Maksimovic and V. Oklobdzija, "Integrated power clock generators for low energy logic," Proceedings of IEEE Power Electronics Specialists Conference, June 1995, pp. 61-67.

[16] D. Maksimovic, "A MOS gate drive with resonant transitions," IEEE PESC, 1991, pp. 527-532.

[17] A. Dickinson and J. Denker, "Adiabatic Dynamic Logic," IEEE Journal of Solid-Sate Circuits, Vol. 30, No. 3, March 1995.

[18] A. Kramer, J. Denker, B. Flower, and J. Moroney, "2nd order adiabatic computation with 2N-2P and 2N-2N2P logic circuits," Proceedings of 1995 International Symposium on Low power design, pp. 191-196, 1995.

[19] A. Blotti and R. Saletti, "Ultra low-power adiabatic circuit semi-custom design," IEEE transactions on VLSI systems, Vol. 12, No. 11, November 2004.

[20] Y. Zhang, H. Chen, J. Kuo, "0.8V CMOS adiabatic differential switch logic circuit using bootstrap technique of low-power VLSI," Electronic Letters, Vol. 38, No.24, November 2002.

[21] S. Kim, "True single-phase adiabatic circuitry for high-performance, low-energy VLSI," Doctor of Philosophy Dissertation, University of Michigan, 2001.

[22] D. Maksimovic, V Oklobdzija, B. Nikolic, and K. Current, "Clocked CMOS adiabatic logic with integrated single-phase power-clock supply," IEEE Transactions on VLSI Systems, Vol. 8, No. 4, pp.460-463, August 2000.

[23] H. Chen and J. Kuo, "A 0.8V CMOS TSPC adiabatic DCVS logic circuit with the bootstrap technique for low-voltage low-power VLSI," Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems, pp. 175-178, December 2004.

[24] E.K. Loo, H.I. Chen, J.B. Kuo and M. Syrzycki, "Low-Voltage Single-Phase Clocked Quasi-Adiabatic Pass-Gate Logic," Canadian Conference on Electrical and Computer Engineering, Vancouver, BC, April 2007.

[25] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm design exploration," Proceedings of the 2006 IEEE International Symposium on Quality Electronic Design, pp.585-590, February 2006.

[26] P.M. Kogge and H.S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," IEEE Transactions on Computers, vol. C-22, pp. 786-793, August 1973.

[27] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," IEEE Computer, Vol. 36, No. 12, pp. 68-75, December 2003.

[28] P. Fantini, G. Giuga, S. Schippers, A. Marmiroli, and G. Ferrair, "Modeling of STI-induced stress phenomena in CMOS 90nm flash technology," Proceedings of 34th European Solid-State Device Research Conference, pp. 401-404, September 2004.